Dynare Manual

Version 4.1.3 (draft)

COLLABORATORS			
	TITLE : Dynare Manual		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Stéphane Adjemian, Houtan Bastani, Michel Juillard, Ferhat Mihoubi, George Perendia, Marco Ratto, and Sébastien Villemot	August 24, 2010	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

List of Tables

Chapter 1

Introduction

1.1 What is Dynare ?

Dynare is a pre-processor and a collection of MATLAB® and GNU Octave routines which solve, simulate and estimate non-linear models with forward looking variables. It is the result of research carried at CEPREMAP by several people (see Laffargue (1990), Boucekkine (1995), Juillard (1996), Collard and Juillard (2001a) and Collard and Juillard (2001b)).

When the framework is deterministic, Dynare can be used for models with the assumption of perfect foresight. Typically, the system is supposed to be in a state of equilibrium before a period `1` when the news of a contemporaneous or of a future shock is learned by the agents in the model. The purpose of the simulation is to describe the reaction in anticipation of, then in reaction to the shock, until the system returns to the old or to a new state of equilibrium. In most models, this return to equilibrium is only an asymptotic phenomenon, which one must approximate by an horizon of simulation far enough in the future. Another exercise for which Dynare is well suited is to study the transition path to a new equilibrium following a permanent shock. For deterministic simulations, Dynare uses a Newton-type algorithm, first proposed by Laffargue (1990), instead of a first order technique like the one proposed by Fair and Taylor (1983), and used in earlier generation simulation programs. We believe this approach to be in general both faster and more robust. The details of the algorithm can be found in Juillard (1996).

In a stochastic context, Dynare computes one or several simulations corresponding to a random draw of the shocks. Dynare uses a Taylor approximation, up to third order, of the expectation functions (see Judd (1996), Collard and Juillard (2001a), Collard and Juillard (2001b), and Schmitt-Grohe and Uribe (2002)).

It is also possible to use Dynare to estimate model parameters either by maximum likelihood as in Ireland (2004) or using a Bayesian approach as in Rabanal and Rubio-Ramirez (2003), Schorfheide (2000) or Smets and Wouters (2003).

Currently the development team of Dynare is composed of S. Adjemian, H. Bastani, M. Juillard, F. Mihoubi, G. Perendia, M. Ratto and S. Villemot. Several parts of Dynare use or have strongly benefited from publicly available programs by G. Anderson, F. Collard, L. Ingber, O. Kamenik, P. Klein, S. Sakata, F. Schorfheide, C. Sims, P. Soederlind and R. Wouters.

Chapter 2

Installation and configuration

2.1 Software requirements

Packaged versions of Dynare are available for Windows® XP/Vista, Debian GNU/Linux and Ubuntu. Dynare should work on other systems, but some compilation steps are necessary in that case.

In order to run Dynare, you need at least one of the following:

- MATLAB® version 6.5 or above; note that no toolbox is needed by Dynare,
- GNU Octave version 3.0.0 or above.

Some installation instructions for GNU Octave can be found on Dynare Wiki.

If you plan to use options use_dll (in particular when computing third order approximations with k_order_solver), you will need to install the necessary requirements for compiling MEX files on your machine. If you are using MATLAB under Windows, install a C++ compiler on your machine, and configure it with MATLAB: see instructions on the Dynare wiki. Users of Octave under Linux should install the package for MEX file compilation (under Debian or Ubuntu, it is called `octave3.2-headers` or `octave3.0-headers`). Users of MATLAB under Linux and MacOS, and users of Octave under Windows, normally need to do nothing, since a working compilation environment is available by default.

2.2 Installation of Dynare

After installation, Dynare can be used in any directory on your computer. It is best practive to keep your model files in directories different from the one containing the Dynare toolbox. That way you can upgrade Dynare and discard the previous version without having to worry about your own files.

2.2.1 On Windows®

Execute the automated installer called `dynare-4.x.y-win.exe` (where 4.x.y is the version number), and follow the instructions. The default installation directory is `c:\dynare\4.x.y`.

After installation, this directory will contain several sub-directories, among which `matlab`, `mex` and `doc`.

The installer will also add an entry in your Start Menu with a shortcut to documentation files and to the uninstaller.

Users of MATLAB 64-bit also need to install Microsoft Visual C++ runtime libraries in order to have functional MEX files.

Note that you can have several versions of Dynare coexisting (for example in `c:\dynare`), as long as you correctly adjust your path settings (see Section 2.3.3).

2.2.2 On Debian GNU/Linux and Ubuntu

Please refer to Dynare Wiki for detailed instructions.

Dynare will be installed under `/usr/share/dynare` and `/usr/lib/dynare`. Documentation will be under `/usr/share/doc/dynare`.

2.2.3 For other systems

You need to download Dynare source code from the Dynare website and unpack it somewhere.

Then you will need to recompile the pre-processor and the dynamic loadable libraries. Please refer to Dynare Wiki.

2.3 Configuration

2.3.1 For MATLAB®

You need to add the `matlab` subdirectory of your Dynare installation to MATLAB® path. You have two options for doing that:

- Using the **addpath** command in the MATLAB® command window:

  Under Windows®, assuming that you have installed Dynare at the standard location, and replacing "4.x.y" by correct version number, type:

  ```
  addpath c:\dynare\4.x.y\matlab
  ```

  Under Debian GNU/Linux or Ubuntu, type:

  ```
  addpath /usr/share/dynare/matlab
  ```

  MATLAB® will not remember this setting next time you run it, and you will have to do it again.

- Via the menu entries:

  Select the "Set Path" entry in the "File" menu, then click on "Add Folder...", and select the `matlab` subdirectory of your Dynare installation. Note that you *should not* use "Add with Subfolders...". Apply the settings by clicking on "Save". Note that MATLAB® will remember this setting next time you run it.

2.3.2 For GNU Octave

You need to add the `matlab` subdirectory of your Dynare installation to Octave path, using the **addpath** at the Octave command prompt.

Under Windows®, assuming that you have installed Dynare at the standard location, and replacing "4.x.y" by correct version number, type:

```
addpath c:\dynare\4.x.y\matlab
```

Under Debian GNU/Linux or Ubuntu, there is no need to use the **addpath** command; the packaging does it for you.

If you are using an Octave version strictly older than 3.2.0, you will also want to tell to Octave to accept the short syntax (without parentheses and quotes) for the **dynare** command, by typing:

```
mark_as_command dynare
```

If you don't want to type this command every time you run Octave, you can put it in a file called `.octaverc` in your home directory (under Windows® this will generally by `c:\Documents and Settings\USERNAME\`). This file is run by Octave at every startup.

2.3.3 Some words of warning

You should be very careful about the content of you MATLAB® or Octave path. You can display its content by simply typing **path** in the command window.

The path should normally contain system directories of MATLAB® or Octave, and some subdirectories of your Dynare installation. You have to manually add the `matlab` subdirectory, and Dynare will automatically add a few other subdirectories at runtime (depending on your configuration). You must verify that there is no directory coming from another version of Dynare than the one you are planning to use.

You have to be aware that adding other directories to your path can potentially create problems, if some of your M-files have the same names than Dynare files. Your files would then override Dynare files, and make Dynare unusable.

Chapter 3

Dynare invocation

In order to give instructions to Dynare, the user has to write a *model file* whose filename extension must be `.mod`. This file contains the description of the model and the computing tasks required by the user. Its contents is described in Chapter 4.

Once the model file is written, Dynare is invoked using the **dynare** command at the MATLAB® or Octave prompt (with the filename of the `.mod` given as argument).

In practice, the handling of the model file is done in two steps: in the first one, the model and the processing instructions written by the user in a *model file* are interpreted and the proper MATLAB® or GNU Octave instructions are generated; in the second step, the program actually runs the computations. Boths steps are triggered automatically by the **dynare** command.

3.1 dynare

dynare — executes Dynare

Synopsis

`dynare` *`FILENAME`*[.mod] [`noclearall`] [`debug`] [`notmpterms`] [`savemacro` [=*`FILENAME`*]] [`onlymacro`] [`nolinemacro`] [`warn_uninit`] [`cygwin`] [`msvc`]

Description

dynare executes instruction included in *`FILENAME`*`.mod`. This user-supplied file contains the model and the processing instructions, as described in Chapter 4.

Details

dynare begins by launching the preprocessor on the `.mod` file. By default (unless `use_dll` option has been given to model), the preprocessor creates three intermediary files:

***`FILENAME`*`.m`** Contains variable declarations, and computing tasks

***`FILENAME`*`_dynamic.m`** Contains the dynamic model equations

***`FILENAME`*`_static.m`** Contains the long run static model equations

These files may be looked at to understand errors reported at the simulation stage.

dynare will then run the computing tasks by executing *`FILENAME`*`.m`.

Options

noclearall By default, **dynare** will issue a **clear all** command to MATLAB® or Octave, thereby deleting all workspace variables; this options instructs **dynare** not to clear the workspace

debug Instructs the preprocessor to write some debugging information about the scanning and parsing of the `.mod` file

notmpterms Instructs the preprocessor to omit temporary terms in the static and dynamic files; this generally decreases performance, but is used for debugging purposes since it makes the static and dynamic files more readable

savemacro[=*FILENAME*] Instructs **dynare** to save the intermediary file which is obtained after macro-processing (see Section 4.14); the saved output will go in the file specified, or if no file is specified in *FILENAME*`-macroexp.mod`

onlymacro Instructs the preprocessor to only perform the macro-processing step, and stop just after. Mainly useful for debugging purposes or for using the macro-processor independently of the rest of Dynare toolbox.

nolinemacro Instructs the macro-preprocessor to omit line numbering information in the intermediary `.mod` file created after the maco-processing step. Useful in conjunction with `savemacro` when one wants that to reuse the intermediary `.mod` file, without having it cluttered by line numbering directives.

warn_uninit Display a warning for each variable or parameter which is not initialized. Initialization should be done through Section 4.3 or load_params_and_steady_state for parameters, or through initval, endval or load_params_and_steady_state for endogenous and exogenous.

cygwin Tells Dynare that your MATLAB® is configured for compiling MEX files with Cygwin (see Section 2.1). This option is only available under Windows, and is used in conjunction with use_dll.

msvc Tells Dynare that your MATLAB® is configured for compiling MEX files with Microsoft Visual C++ (see Section 2.1). This option is only available under Windows, and is used in conjunction with use_dll.

Output

Depending on the computing tasks requested in the `.mod` file, executing command **dynare** will leave in the workspace variables containing results available for further processing. More details are given under the relevant computing tasks.

The `M_`, `oo_` and `options_` structures are also saved in a file called *FILENAME*`_results.mat`.

Examples

```
dynare ramst
dynare ramst.mod savemacro
```

Chapter 4

The Model file

Dynare commands are either single instructions or a block of instructions. Each single instruction and each element of a block is terminated by a semicolon (`;`). Blocks of instructions are terminated by **end;**.

Most Dynare commands have arguments and several accept options, indicated in parentheses after the command keyword.

In the description of Dynare commands, the following conventions are observed:

- optional arguments or options are indicated between square brackets `[]`
- repreated arguments are indicated by ellipses `...`
- mutually exclusive arguments are separated by vertical bars `|`
- *INTEGER* indicates an integer number
- *DOUBLE* indicates a double precision number. The following syntaxes are valid: `1.1e3`, `1.1E3`, `1.1d3`, `1.1D3`
- *EXPRESSION* indicates a mathematical expression valid outside the model description (see Section 4.2)
- *MODEL_EXPRESSION* indicates a mathematical expression valid in the model description (see Section 4.2 and model)
- *VARIABLE_NAME* indicates a variable name starting with an alphabetical character and can't contain `()+-*/^=!;:@#.` or accentuated characters
- *PARAMETER_NAME* indicates a parameter name starting with an alphabetical character and can't contain `()+-*/^=!;:@#.` or accentuated characters
- *LATEX_NAME* indicates a valid LaTeX expression in math mode (not including the dollar signs)
- *FUNCTION_NAME* indicates a valid MATLAB® function name
- *FILENAME* indicates a filename valid in the underlying operating system; it is necessary to put it between double quotes when specifying the extension or if the filename contains a non-alphanumeric character

4.1 Variable declarations

Declarations of variables and parameters are made with the following commands:

- var
- varexo
- varexo_det
- parameters
- change_type
- predetermined_variables

4.1.1 var

var — declares endogenous variables

Synopsis

`var` *VARIABLE_NAME* [$*LATEX_NAME*$] [[,]*VARIABLE_NAME* [$*LATEX_NAME*$]...] ;

Description

This required command declares the endogenous variables in the model. See Conventions [7] for the syntax of *VARIABLE_NAME*. Optionally it is possible to give a LaTeX name to the variable.

var commands can appear several times in the file and Dynare will concatenate them.

Example

```
var c gnp q1 q2;
```

4.1.2 varexo

varexo — declares exogenous variables

Synopsis

`varexo` *VARIABLE_NAME* [$*LATEX_NAME*$] [[,]*VARIABLE_NAME* [$*LATEX_NAME*$]...] ;

Description

This optional command declares the exogenous variables in the model. See Conventions [7] for the syntax of *VARIABLE_NAME*. Optionally it is possible to give a LaTeX name to the variable.

Exogenous variables are required if the user wants to be able to apply shocks to her model.

varexo commands can appear several times in the file and Dynare will concatenate them.

Example

```
varexo m gov;
```

4.1.3 varexo_det

varexo_det — declares exogenous deterministic variables in a stochastic model

Synopsis

`varexo_det` *VARIABLE_NAME* [$*LATEX_NAME*$] [[,]*VARIABLE_NAME* [$*LATEX_NAME*$]...] ;

Description

This optional command declares exogenous deterministic variables in a stochastic model. See Conventions [7] for the syntax of *VARIABLE_NAME*. Optionally it is possible to give a LaTeX name to the variable.

It is possible to mix deterministic and stochastic shocks to build models where agents know from the start of the simulation about future exogenous changes. In that case stoch_simul will compute the rational expectation solution adding future information to the state space (nothing is shown in the output of stoch_simul) and forecast will compute a simulation conditional on initial conditions and future information.

varexo_det commands can appear several times in the file and Dynare will concatenate them.

Example

```
varexo m gov;
varexo_det tau;
```

4.1.4 parameters

parameters — declares parameters

Synopsis

`parameters` *PARAMETER_NAME* [$*LATEX_NAME*$] [[,]*PARAMETER_NAME* [$*LATEX_NAME*$]...] ;

Description

This command declares parameters used in the model, in variable initialization or in shocks declarations. See Conventions [7] for the syntax of *PARAMETER_NAME*. Optionally it is possible to give a LaTeX name to the parameter.

The parameters must subsequently be assigned values, see Section 4.3.

parameters commands can appear several times in the file and Dynare will concatenate them.

Example

```
parameters alpha, bet;
```

4.1.5 change_type

change_type — modify the type of declared variables/parameters

Synopsis

`change_type` (var | varexo | varexo_det | parameters) *VARIABLE_NAME* | *PARAMETER_NAME* [[,] *VARIABLE_NAME* | *PARAMETER_NAME* ...] ;

Description

Changes the types of the specified variables/parameters to another type: endogenous, exogenous, exogenous deterministic or parameter.

It is important to understand that this command has a global effect on the `.mod` file: the type change is effective after, but also before, the **change_type** command. This command is typically used when flipping some variables for steady state calibration: typically a separate model file is used for calibration, which includes the list of variable declarations with the macro-processor, and flips some variable.

Example

```
var y, w;
parameters alpha, bet;
...
change_type(var) alpha, bet;
change_type(parameters) y, w;
```

Here, in the whole model file, `alpha` and `beta` will be endogenous and `y` and `w` will be parameters.

4.1.6 predetermined_variables

predetermined_variables — declare some endogenous variables as predetermined

Synopsis

`predetermined_variables` *VARIABLE_NAME* [*VARIABLE_NAME*...] ;

Description

In Dynare, the default convention is that the timing of a variable reflects when this variable is decided. The typical example is for capital stock: since the capital stock used at current period is actually decided at the previous period, then the capital stock entering the production function is `k(-1)`, and the law of motion of capital must be written:

```
k = i + (1-delta)*k(-1)
```

Put another way, for stock variables, the default in Dynare is to use a "stock at the end of the period" concept, instead of a "stock at the beginning of the period" convention.

The **predetermined_variables** is used to change that convention. The endogenous variables declared as predetermined variables are supposed to be decided one period ahead of all other endogenous variables. For stock variables, they are supposed to follow a "stock at the beginning of the period" convention.

Example

The following two program snippets are strictly equivalent.

Using default Dynare timing convention:

```
var y, k, i;
...
model;
y = k(-1)^alpha;
k = i + (1-delta)*k(-1);
...
end;
```

Using the alternative timing convention:

```
var y, k, i;
predetermined_variables k;
...
model;
y = k^alpha;
k(+1) = i + (1-delta)*k;
...
end;
```

4.2 Expressions

Dynare distinguishes between two types of mathematical expressions: those that are used to describe the model, and those that are used outside the model block (*e.g.* for initializing parameters or variables, or as command options). In this manual, those two types of expressions are respectively denoted by *MODEL_EXPRESSION* and *EXPRESSION*.

Unlike MATLAB® or Octave expressions, Dynare expressions are necessarily scalar ones: they cannot contain matrices or evaluate to matrices[1].

Expressions can be constructed using integers (*INTEGER*), floating point numbers (*DOUBLE*), parameter names (*PARAMETER_NAME*), variable names (*VARIABLE_NAME*), operators and functions.

4.2.1 Parameters and variables

Parameters and variables can be introduced in expressions by simply typing their names. The semantics of parameters and variables is quite different whether they are used inside or outside the model block.

4.2.1.1 Inside the model

Parameters used inside the model refer to the value given through parameter initialization or homotopy_setup when doing a simulation, or are the estimated variables when doing an estimation.

Variables used in a *MODEL_EXPRESSION* denote *current period* values when neither a lead or a lag is given. A lead or a lag can be given by enclosing an integer between parenthesis just after the variable name: a positive integer means a lead, a negative one means a lag. Leads or lags of more than one period are allowed. For example, if `c` is an endogenous variable, then `c(+1)` is the variable one period ahead, and `c(-2)` is the variable two periods before.

When specifying the leads and lags of endogenous variables, it is important to respect the following convention: in Dynare, the timing of a variable reflects when that variable is decided. A control variable - which by definition is decided in the current period - must have no lead. A predetermined variable - which by definition has been decided in a previous period - must have a lag. A consequence of this is that all stock variables must use the "stock at the end of the period" convention. Please refer to Mancini-Griffoli (2007) for more details and concrete examples.

Leads and lags are primarily used for endogenous variables, but can be used for exogenous variables. They have no effect on parameters and are forbidden for local model variables (see model).

4.2.1.2 Outside the model

When used in an expression outside the model block, a parameter or a variable simply refers to the last value given to that variable. More precisely, for a parameter it refers to the value given in the corresponding parameter initialization; for an endogenous or exogenous variable, it refers to the value given in the most recent initval or endval block.

4.2.2 Operators

The following operators are allowed in both *MODEL_EXPRESSION* and *EXPRESSION*:

- binary arithmetic operators: `+`, `-`, `*`, `/`, `^`
- unary arithmetic operators: `+`, `-`
- binary comparison operators (which evaluate to either `0` or `1`): `<`, `>`, `<=`, `>=`, `==`, `!=`

The following operators are allowed in *MODEL_EXPRESSION*:

[1] Note that arbitrary MATLAB® or Octave expressions can be put in a `.mod` file, but those expressions have to be on separate lines, generally at the end of the file for post-processing purposes. They are not interpreted by Dynare, and are simply passed on unmodified to MATLAB® or Octave. Those constructions are not addresses in this section.

- steady state operator: `STEADY_STATE(`*`MODEL_EXPRESSION`*`)`. This operator is used to take the value of the enclosed expression at the steady state. A typical usage is in the Taylor rule, where you may want to use the value of GDP at steady state to compute the output gap.
- expectation operator: `EXPECTATION(`*`INTEGER`*`)(`*`MODEL_EXPRESSION`*`)`. This operator is used to take the expectation of some expression using a different information set than the information available at current period. For example, `EXPECTATION(-1)(x(+1))` is equal to the expected value of variable `x` at next period, using the information set available at the previous period. In practice, Dynare solves this by creating an auxiliary variable equal to `AUX_EXPECT_LAG_1 = x(+2)`, and by replacing the expectation operator by `AUX_EXPECT_LAG_1(-1)`. Note that a value of `0` for the time shift component is reserved for partial information models (not yet fully implemented).

4.2.3 Functions

The following standard functions are allowed in both *`MODEL_EXPRESSION`* and *`EXPRESSION`*:

- exponential: `exp(`*`x`*`)`
- natural logarithm: `log(`*`x`*`)` (or equivalently `ln(`*`x`*`)`)
- base 10 logarithm: `log10(`*`x`*`)`
- square root: `sqrt(`*`x`*`)`
- trigonometric functions: `sin(`*`x`*`)`, `cos(`*`x`*`)`, `tan(`*`x`*`)`, `asin(`*`x`*`)`, `acos(`*`x`*`)`, `atan(`*`x`*`)`
- maximum and minimum: `max(`*`a, b`*`)`, `min(`*`a, b`*`)`
- gaussian cumulative distribution function: `normcdf(`*`x, μ, σ`*`)` (note that `normcdf(`*`x`*`)` is equivalent to `normcdf(`*`x`*`, 0, 1)`)

In a *`MODEL_EXPRESSION`*, no other function is allowed.[2]

In an *`EXPRESSION`*, it is possible to use any arbitrary MATLAB® or Octave function, provided that this function has scalar arguments and return value.

4.3 Parameter initialization

When using Dynare for computing simulations, it is necessary to calibrate the parameters of the model. This is done through parameter initialization.

Syntax

```
PARAMETER_NAME = EXPRESSION ;
```

Example

```
parameters alpha, bet;

beta = 0.99;
alpha = 0.36;
A = 1-alpha*beta;
```

[2]This is due to the fact that the Dynare preprocessor performs a symbolical derivation of all model equations, and therefore needs to know the analytical derivatives of all the equations in the model equations. In the future, we should add support for other usual functions, and implement an interface to let the user define custom functions, for which he would provide the analytical derivatives.

4.4 Model declaration

The model is declared inside a model block.

Note that it is possible to output the list of model equations to a LaTeX file, using the write_latex_dynamic_model command, or the write_latex_static_model (for the steady state model).

4.4.1 model

model — declares the model equations

Synopsis

`model` [(*OPTION* [, *OPTION*...])] ;
[*MODEL_EXPRESSION* = *MODEL_EXPRESSION* ; | *MODEL_EXPRESSION* ; | # *VARIABLE_NAME* = *MODEL_EXPRESSION* ;]...

`end` ;

Description

The equations of the model are written in a block delimited by **model** and **end** keywords.

There must be as many equations as there are endogenous variables in the model, except when computing the unconstrained optimal policy with ramsey_policy.

The syntax of equations must follow the conventions for *MODEL_EXPRESSION* as described in Section 4.2. Each equation must be terminated by a semicolon (`;`).

When the equations are written in homogenous form, it is possible to omit the `=0` part and write only the left hand side of the equation.

Inside the model block, Dynare allows the creation of *model-local variables*, which constitute a simple way to share a common expression between several equations. The syntax consists of a pound sign (`#`) followed by the name of the new model local variable (which must *not* be declared as in Section 4.1), an equal sign, and the expression for which this new variable will stand. Later on, every time this variable appears in the model, Dynare will substitute it by the expression assigned to the variable. Note that the scope of this variable is restricted to the model block; it cannot be used outside.

Options

`linear` Declares the model as being linear. It spares oneself from having to declare initial values for computing the steady state, and it sets automatically `order=1` in stoch_simul.

`use_dll` Instructs the preprocessor to create dynamic loadable libraries (DLL) containing the model equations and derivatives, instead of writing those in `M`-files. You need a working compilation environment, *i.e.* a working `mex` command (see Section 2.1 for more details). Using this option can result in faster simulations or estimations, at the expense of some initial compilation time.[3]

`block` Perform the block decomposition of the model, and exploit it in computations. See Dynare wiki for details on the algorithm.

`bytecode` Instead of M-files, use a bytecode representation of the model, *i.e.* a binary file containing a compact representation of all the equations.

`cutoff` = ***DOUBLE*** Threshold under which a jacobian element is considered as null during the model normalization. Only available with option `block`. Default: `1e-15`

[3] In particular, for big models, the compilation step can be very time-consuming, and use of this option may be counter-productive in those cases.

mfs = *INTEGER* Controls the handling of minimum feedback set of endogenous variables. Only available with option `block`. Possible values:

- `0`: all the endogenous variables are considered as feedback variables (Default).
- `1`: the endogenous variables assigned to equation naturally normalized (*i.e.* of the form `x=f(Y)` where `x` does not appear in `Y`) are potentially recursive variables. All the other variables are forced to belong to the set of feedback variables.
- `2`: in addition of variables with `mfs = 1` the endogenous variables related to linear equations which could be normalized are potential recursive variables. All the other variables are forced to belong to the set of feedback variables.
- `3`: in addition of variables with `mfs = 2` the endogenous variables related to non-linear equations which could be normalized are potential recursive variables. All the other variables are forced to belong to the set of feedback variables.

Examples

Example 1: elementary RBC model

```
var c k;
varexo x;
parameters aa alph bet delt gam;

model;
c =  - k + aa*x*k(-1)^alph + (1-delt)*k(-1);
c^(-gam) = (aa*alph*x(+1)*k^(alph-1) + 1 - delt)*c(+1)^(-gam)/(1+bet);
end;
```

Example 2: use of model local variables

The following program:

```
model;
# gamma = 1 - 1/sigma;
u1 = c1^gamma/gamma;
u2 = c2^gamma/gamma;
end;
```

...is formally equivalent to:

```
model;
u1 = c1^(1-1/sigma)/(1-1/sigma);
u2 = c2^(1-1/sigma)/(1-1/sigma);
end;
```

Example 3: a linear model

```
model(linear);
x = a*x(-1)+b*y(+1)+e_x;
y = d*y(-1)+e_y;
end;
```

4.4.2 write_latex_dynamic_model

write_latex_dynamic_model — create a LaTeX file containing the (dynamic) model

Synopsis

```
write_latex_dynamic_model ;
```

Description

If your `.mod` file is `FILENAME.mod`, then Dynare will create a file called `FILENAME_dynamic.tex`, containing the list of all the dynamic model equations.

If LaTeX names were given for variables and parameters (see var, varexo, varexo_det, parameters), then those will be used; otherwise, the plain text names will be used.

Time subscripts (`t`, `t+1`, `t-1`, ...) will be appended to the variable names, as LaTeX subscripts.

Note that the model written in the TeX file will differ from the model declared by the user in the following dimensions:

- the timing convention of predetermined_variables will have been changed to the default Dynare timing convention; in other words, variables declared as predetermined will be lagged on period back,
- the expectation operators will have been removed, replaced by auxiliary variables and new equations as explained in the documentation of the operator,
- for stochastic models, variables with leads or lags greater or equal than two will have been removed, replaced by new auxiliary variables and equations.

4.4.3 write_latex_static_model

write_latex_static_model — create a LaTeX file containing the (static) model

Synopsis

```
write_latex_static_model;
```

Description

If your `.mod` file is `FILENAME.mod`, then Dynare will create a file called `FILENAME_static.tex`, containing the list of all the equations of the steady state model.

If LaTeX names were given for variables and parameters (see var, varexo, varexo_det, parameters), then those will be used; otherwise, the plain text names will be used.

Note that the model written in the TeX file will differ from the model declared by the user in the some dimensions, see write_latex_dynamic_model.

4.5 Initial and terminal conditions

For most simulation exercises, it is necessary to provide initial (and possibly terminal) conditions. It is also necessary to provide initial guess values for non-linear solvers. The following statements are used for those purposes:

- initval
- endval
- histval
- resid
- initval_file

In many contexts (determistic or stochastic), it is necessary to compute the steady state of a non-linear model: initval then specifies numerical initial values for the non-linear solver. The command resid can be used to compute the equation residuals for the given initial values.

Used in perfect foresight mode, the types of forward-loking models for which Dynare was designed require both initial and terminal conditions. Most often these initial and terminal conditions are static equilibria, but not necessarily.

One typical application is to consider an economy at the equilibrium, trigger a shock in first period, and study the trajectory of return at the initial equilbrium. To do that, one needs initval and shocks (see Section 4.6.

Another one is to study, how an economy, starting from arbitrary initial conditions converges toward equilibrium. To do that, one needs initval and endval;

For models with lags on more than one period, the command histval permits to specify different historical initial values for periods before the beginning of the simulation.

4.5.1 initval

initval — specifies numerical starting values for finding the steady state and/or initial values for simulations

Synopsis

```
initval;
VARIABLE_NAME = EXPRESSION; ...

end;
```

Description

The **initval** block serves two purposes: declaring the initial (and possibly terminal) conditions in a simulation exercise, and providing guess values for non-linear solvers.

In a deterministic (*i.e.* perfect foresight) model

First, it provides the initial conditions for all the endogenous and exogenous variables at all the periods preceeding the first simulation period (unless some of these initial values are modified by histval).

Second, in the absence of an endval block, it sets the terminal conditions for all the periods succeeding the last simulation period.

Third, in the absence of an endval block, it provides initial guess values at all simulation dates for the non-linear solver implemented in simul.

For this last reason, it necessary to provide values for all the endogenous variables in an **initval** block (even though, theoretically, initial conditions are only necessary for lagged variables). If some exogenous variables are not mentionned in the **initval** block, a zero value is assumed.

Note that if the **initval** block is immediately followed by a steady command, its semantics is changed. The steady command will compute the steady state of the model for all the endogenous variables, assuming that exogenous variables are kept constant to the value declared in the **initval** block, and using the values declared for the endogenous as initial guess values for the non-linear solver. An **initval** block followed by steady is formally equivalent to an **initval** block with the same values for the exogenous, and with the associated steady state values for the endogenous.

In a stochastic model

The main purpose of **initval** is to provide initial guess values for the non-linear solver in the steady state computation. Note that if the **initval** block is not followed by steady, the steady state computation will still be triggered by subsequent commands (stoch_simul, estimation...).

It is not necessary to declare `0` as initial value for exogenous stochastic variables, since it is the only possible value.

This steady state will be used as the initial condition at all the periods preceeding the first simulation period for the two possible types of simulations in stochastic mode:

- in stoch_simul, if the `periods` options is specified
- in forecast (in this case, note that it is still possible to modify some of these initial values with histval)

Example

```
initval;
c = 1.2;
k = 12;
x = 1;
end;

steady;
```

4.5.2 endval

endval — specifies terminal values for deterministic simulations

Synopsis

```
endval ;
VARIABLE_NAME = EXPRESSION ; ...

end ;
```

Description

The **endval** block makes only sense in a determistic model, and serves two purposes.

First, it sets the terminal conditions for all the periods succeeding the last simulation period.

Second, it provides initial guess values at all the simulation dates for the non-linear solver implemented in simul.

For this last reason, it necessary to provide values for all the endogenous variables in an **endval** block (even though, theoretically, initial conditions are only necessary for forward variables). If some exogenous variables are not mentionned in the **endval** block, a zero value is assumed.

Note that if the **endval** block is immediately followed by a steady command, its semantics is changed. The steady command will compute the steady state of the model for all the endogenous variables, assuming that exogenous variables are kept constant to the value declared in the **endval** block, and using the values declared for the endogenous as initial guess values for the non-linear solver. An **endval** block followed by steady is formally equivalent to an **endval** block with the same values for the exogenous, and with the associated steady state values for the endogenous.

Example

```
var c k;
varexo x;
...
initval;
c = 1.2;
k = 12;
x = 1;
end;

steady;

endval;
c = 2;
k = 20;
x = 2;
end;

steady;
```

The initial equilibrium is computed by steady for `x=1`, and the terminal one, for `x=2`.

4.5.3 histval

histval — specifies historical values before the start of a simulation

Synopsis

```
histval ;
VARIABLE_NAME(INTEGER) = EXPRESSION ; ...


end ;
```

Description

EXPRESSION is any valid expression returning a numerical value and can contain already initialized variable names.

In models with lags on more than one period, the optional **histval;** ... **end;** block permits to specify different historical initial values for different periods.

By convention in Dynare, period 1 is the first period of the simulation. Going backward in time, the first period before the start of the simulation is period 0, then period -1, and so on.

If your lagged variables are linked by identities, be careful to satisfy these identities when you set historical initial values.

Example

```
var x y;
varexo e;

model;
x = y(-1)^alpha*y(-2)^(1-alpha)+e;
...
end;

initval;
```

```
x = 1;
y = 1;
e = 0.5;
end;

steady;

histval;
y(0) = 1.1;
y(-1) = 0.9;
end;
```

4.5.4 resid

resid — display the residual of the static equations

Synopsis

```
resid;
```

Description

This command will display the residuals of the static equations of the model, using the values given for the endogenous in the last initval or endval block (or the steady state file if you provided one).

4.5.5 initval_file

initval_file — use an external to specify a path for exogenous and endogenous variables in a deterministic simulation

Synopsis

```
initval_file(filename = FILENAME);
```

Description

In a deterministic setup, this command is used to specify a path for all endogenous and exogenous variables. The length of these paths must be equal to the number of simulation periods, plus the number of leads and the number of lags of the model (for example, with 50 simulation periods, in a model with 2 lags and 1 lead, the paths must have a length of 53). Note that these paths cover two different things:

- the constraints of the problem, which are given by the path for exogenous and the initial and terminal values for endogenous
- the initial guess for the non-linear solver, which is given by the path for endogenous variables for the simulation periods (excluding initial and terminal conditions)

The command accepts three file formats:

- M-file (extension `.m`): for each endogenous and exogenous variable, the file must contain a row vector of the same name
- MAT-file (extension `.mat`): same as for M-files
- Excel file (extension `.xls`): for each endogenous and exogenous, the file must contain a column of the same name

Warning
The extension must be omitted in the command argument. Dynare will automatically figure out the extension and select the appropriate file type.

4.6 Shocks on exogenous variables

In a deterministic context, when one wants to study the transition of one equilibrium position to another, it is equivalent to analyze the consequences of a permanent shock and this in done in Dynare through the proper use of initval and endval.

Another typical experiment is to study the effects of a temporary shock after which the system goes back to the original equilibrium (if the model is stable ...). A temporary shock is a temporary change of value of one or several exogenous variables in the model. Temporary shocks are specified with the command shocks.

In a stochastic framework, the exogenous variables take random values in each period. In Dynare, these random values follow a normal distribution with zero mean, but it belongs to the user to specify the variability of these shocks. The non-zero elements of the matrix of variance-covariance of the shocks can be entered with the shocks command. Or, the entire matrix can be direclty entered with Sigma_e (this use is however deprecated).

If the variance of an exogenous variable is set to zero, this variable will appear in the report on policy and transition functions, but isn't used in the computation of moments and of Impulse Response Functions. Setting a variance to zero is an easy way of removing an exogenous shock.

- shocks
- mshocks
- Sigma_e (deprecated)

4.6.1 shocks

shocks — specifies shocks on deterministic or stochastic exogenous variables

Synopsis

`shocks` ;
[[*(1)DETERMINISTIC SHOCK STATEMENT* | *(2)STOCHASTIC SHOCK STATEMENT*]...]

`end` ;

(1) var *VARIABLE_NAME*; periods *INTEGER* [:*INTEGER*] [[,]*INTEGER* [:*INTEGER*]...]; values *EXPRESSION* [[,]*EXPRESSION*...];

(2) var *VARIABLE_NAME*; stderr *EXPRESSION*; | var *VARIABLE_NAME* = *EXPRESSION*; | var *VARIABLE_NAME*, *VARIABLE_NAME* = *EXPRESSION*; | corr *VARIABLE_NAME*, *VARIABLE_NAME* = *EXPRESSION*;

Description

In deterministic context

For deterministic simulations, the **shocks** block specifies temporary changes in the value of an exogenous variables. For permanent shocks, use an endval block.

When specifying shocks on several periods, the **values** *EXPRESSION* must return either a scalar value common to all periods with a shock or a column vector with as many elements as there are periods in the **periods** statement just before it.

Example

```
shocks;
var e;
periods 1;
values 0.5;
var u;
periods 4:5;
values 0;
var v;
periods 4 5 6;
values 0;
var u;
periods 4  5   6;
values  1 1.1 0.9;
end;
```

In stochastic context

For stochastic simulations, the **shocks** block specifies the non zero elements of the covariance matrix of the shocks.

Example

```
shocks;
var e = 0.000081;
var u; stderr 0.009;
corr e, u = 0.8;
var v, w = 2;
end;
```

See also

Sigma_e

Mixing determininistic and stochastic shocks

It is possible to mix deterministic and stochastic shocks to build models where agents know from the start of the simulation about future exogenous changes. In that case stoch_simul will compute the rational expectation solution adding future information to the state space (nothing is shown in the output of stoch_simul) and forecast will compute a simulation conditional on initial conditions and future information.

Example

```
varexo_det tau;
varexo e;

...

shocks;
var e; stderr 0.01;
var tau;
periods 1:9;
values -0.15;
end;

stoch_simul(irf=0);
```

```
forecast;
```

4.6.2 mshocks

mshocks — specifies multiplicative deterministic shocks on exogenous variables

Synopsis

```
mshocks ;
[ var VARIABLE_NAME; periods INTEGER [:INTEGER] [[,]INTEGER [:INTEGER]...]; values EXPRESSION [[,]EXPRESSION...]; ...]

end ;
```

Description

The purpose of this command is similar to that of the shocks for deterministic shocks, except that the numeric values given will be interpreted in a multiplicative way. For example, if a value of `1.05` is given as shock value for some exogenous at some date, it means 5% above its steady state value (as given by the last initval or endval block).

This command is only meaningful in two situations:

- on exogenous variables with a non-zero steady state, in a deterministic setup,
- on deterministic exogenous variables with a non-zero steady state, in a stochastic setup.

4.6.3 Sigma_e

Sigma_e — specifies directly the covariance matrix of the stochastic shocks

Synopsis

```
Sigma_e = [ EXPRESSION [[,]EXPRESSION...] [; EXPRESSION [[,]EXPRESSION...]...] ];
```

Warning
The matrix elements are actually written between square brackets (`[]`). Here, the initial `[` and final `]` don't have the meaning of 'optional element' as elsewhere.

Description

The matrix of variance-covariance of the shocks can be directly specified as a upper (or lower) triangular matrix. Dynare builds the corresponding symmetrix matrix. Each row of the triangular matrix, except the last one, must be terminated by a semi-colon `;`. For a given element, an arbitrary *EXPRESSION* is allowed (instead of a simple constant), but in that case you need to enclose the expression in parentheses. *The order of the covariances in the matrix is the same as the one used in the varexo declaration.*

Example

```
varexo u, e;
...
Sigma_e = [ 0.81 (phi*0.9*0.009); 0.000081];
```

where the variance of `u` is 0.81, the variance of `e`, 0.000081, and the correlation between `e` and `u` is `phi`.

4.7 Other general declarations

- dsample
- periods (deprecated)

4.7.1 dsample

dsample — reduces the number of periods considered in subsequent output commands

Synopsis

`dsample` *INTEGER* [*INTEGER*] ;

Description

...

4.7.2 periods

periods — specifies the number of simulation periods

Synopsis

`periods` *INTEGER*;

Description

This command is now deprecated (but will still work for older model files). It is not necessary when no simulation is performed and is replaced by an option `periods` in simul and stoch_simul.

Sets the number of periods in the simulation. The periods are numbered from `1` to *INTEGER*. In perfect foresight simulations, it is assumed that all future events are perfectly known at the beginning of period `1`.

Example

```
periods 100;
```

4.8 Solving and simulating

Dynare has special commands for the computation of the static equilibrium of the model (steady), of the eigenvalues of the linearized model (check) for dynamics local analysis, of a deterministic simulation (simul) and for solving and/or simulating a stochastic model (stoch_simul).

- steady
- homotopy_setup
- check
- model_info
- simul
- stoch_simul

4.8.1 steady

steady — computes the steady state of a model

Synopsis

`steady` [(*OPTION* [, *OPTION*...])] ;

Options

`solve_algo` = ***INTEGER*** Determines the non-linear solver to use. Possible values for the option are:

- `0`: uses MATLAB® Optimization Toolbox FSOLVE
- `1`: uses Dynare's own nonlinear equation solver
- `2`: splits the model into recursive blocks and solves each block in turn
- `3`: Chris Sims' solver
- `4`: similar to value `2`, except that it deals differently with nearly singular Jacobian
- `5`: Newton algorithm with a sparse Gaussian elimination (SPE)

Default value is `2`.

`homotopy_mode` = ***INTEGER*** Use a homotopy (or divide-and-conquer) technique to solve for the steady state (see homotopy_setup for a short description of the technique). This option can take three possible values:

- `1`: in this mode, all the parameters are changed simultaneously, and the distance between the boudaries for each parameter is divided in as many intervals as there are steps (as defined by `homotopy_steps` option); the problem is solves as many times as there are steps
- `2`: same as mode `1`, except that only one parameter is changed at a time; the problem is solved as many times as steps times number of parameters
- `3`: Dynare tries first the most extreme values. If it fails to compute the steady state, the interval between initial and desired values is divided by two for all parameters. Every time that it is impossible to find a steady state, the previous interval is divided by two. When it succeeds to find a steady state, the previous interval is multiplied by two. In that last case `homotopy_steps` contains the maximum number of computations attempted before giving up.

`homotopy_steps` = ***INTEGER*** Defines the number of steps when performing a homotopy. See `homotopy_mode` option for more details.

Description

Computes the equilibrium value of the endogenous variables for the value of the exogenous variables specified in the previous initval or endval block.

steady uses an iterative procedure and takes as initial guess the value of the endogenous variables set in the previous initval or endval block.

For complicated models, finding good numerical initial values for the endogenous variables is the trickiest part of finding the equilibrium of that model. Often, it is better to start with a smaller model and add new variables one by one.

If you know how to compute the steady state for your model, you can provide a MATLAB® function doing the computation instead of using **steady**. The function should be called with the name of the `.mod` file followed by `_steadystate`. See `fs2000a_steadystate.m` in `examples/fs2000` directory.

Output variables

The steady state is available in `oo_.steady_state`. Endogenous variables are ordered in order of declaration used in var command as in `M_.endo_names`.

Examples

See initval and endval.

4.8.2 homotopy_setup

homotopy_setup — declare initial and final values when using homotopy method

Synopsis

```
homotopy_setup;
VARIABLE_NAME, EXPRESSION [, EXPRESSION];...

end;
```

Description

The idea of homotopy (also called divide-and-conquer by some authors) is to subdivide the problem of finding the steady state into smaller problems. It assumes that you know how to compute the steady state for a given set of parameters, and it helps you finding the steady state for another set of parameters, by incrementally moving from one to another set of parameters.

The purpose of the **homotopy_setup** block is to declare the final (and possibly also the initial) values for the parameters or exogenous that will be changed during the homotopy. In the first syntax where only one value is specified for a given parameter/exogenous, then this value is interpreted as the final value, and the initial value is taken from the preceeding initval block. In the second syntax where two values are specified for a given parameter/exogenous, the first is the initial one, the second is the final one.

A necessary condition for a successful homotopy is that Dynare must be able to solve the steady state for the initial parameters/exogenous without additional help (using the guess values given in the initval block).

If the homotopy fails, a possible solution is to increase the number of steps (given in `homotopy_steps` option of steady).

Example

In the following example, Dynare will first compute the steady state for the initial values (`gam=0.5` and `x=1`), and then subdivide the problem into 50 smaller problems to find the steady state for the final values (`gam=2` and `x=2`).

```
var c k;
varexo x;

parameters alph gam delt bet aa;
alph=0.5;
delt=0.02;
aa=0.5;
bet=0.05;

model;
c + k - aa*x*k(-1)^alph - (1-delt)*k(-1);
c^(-gam) - (1+bet)^(-1)*(aa*alph*x(+1)*k^(alph-1) + 1 - delt)*c(+1)^(-gam);
end;

initval;
x = 1;
k = ((delt+bet)/(aa*x*alph))^(1/(alph-1));
c = aa*x*k^alph-delt*k;
end;
```

```
homotopy_setup;
gam, 0.5, 2;
x, 2;
end;

steady(homotopy_mode = 1, homotopy_steps = 50);
```

4.8.3 check

check — computes the eigenvalues of the (linearized) model

Synopsis

`check` [(`solve_algo` = *INTEGER*)] ;

Options

`solve_algo` = *`INTEGER`* See there for the possible values and their meaning

Description

Computes the eigenvalues of the model linearized around the values specified by the last initval, endval or steady statement. Generally, the eigenvalues are only meaningful if the linearization is done around a steady state of the model. It is a device for local analysis in the neighborhood of this steady state.

A necessary condition for the uniqueness of a stable equilibrium in the neighborhood of the steady state is that there are as many eigenvalues larger than one in modulus as there are forward looking variables in the system. An additional rank condition requires that the square submatrix of the right Schur vectors corresponding to the forward looking variables (jumpers) and to the explosive eigenvalues must have full rank.

Output variables

check returns the eigenvalues in the global variable `oo_.dr.eigval`.

4.8.4 model_info

model_info — Display the block structure of the model

Synopsis

`model_info` ;

Description

The **model_info** command provides information about:

- the normalization of the model: an endogenous variable is attributed to each equation of the model;
- the block structure of the model: for each block model_info indicates its type, the equations number and endogenous variables belonging to this block.

There are five different types of blocks depending on the simulation method used:

- EVALUATE FORWARD: in this case the block contains only equations where endogenous variable attributed to the equation appears currently on the left hand side and where no forward looking endogenous variables appear. $y_{j,t} = f_j(y_t, y_{t-1}, ..., y_{t-k})$
- EVALUATE BACKWARD: the block contains only equations where endogenous variable attributed to the equation appears currently on the left hand side and where no backward looking endogenous variables appear. $y_{j,t} = f_j(y_t, y_{t+1}, ..., y_{t+k})$
- SOLVE FORWARD *x*: the block contains only equations where endogenous variable attributed to the equation does not appear currently on the left hand side and where no forward looking endogenous variables appear. $g_j(y_{j,t}, y_t, y_{t-1}, ..., y_{t-k}) = 0$. *x* is equal to SIMPLE if the block has only one equation. If several equation appears in the block, *x* is equal to COMPLETE.
- SOLVE FORWARD *x*: the block contains only equations where endogenous variable attributed to the equation does not appear currently on the left hand side and where no backward looking endogenous variables appear. $g_j(y_{j,t}, y_t, y_{t+1}, ..., y_{t+k}) = 0$. *x* is equal to SIMPLE if the block has only one equation. If several equation appears in the block, *x* is equal to COMPLETE.
- SOLVE TWO BOUNDARIES *x*: the block contains equations depending on both forward and backward variables. $g_j(y_{j,t}, y_t, y_{t-1}, ..., y_{t-k}, y_t, y_{t+1}, ..., y_{t+k}) = 0$. *x* is equal to SIMPLE if the block has only one equation. If several equation appears in the block, *x* is equal to COMPLETE.

4.8.5 simul

simul — simulates a deterministic model

Synopsis

`simul` [(*OPTION* [, *OPTION*...])] ;

Description

Triggers the computation of a deterministic simulation of the model for the number of periods set in the option `periods`. **simul** uses a

Options

`periods` = ***INTEGER*** Number of periods of the simulation

`stack_solve_algo` = ***INTEGER*** Algorithm used for computing the solution. Possible values are:

- `0`: Newton method to solve simultaneously all the equations for every period, see Juillard (1996). (Default)
- `1`: use a Newton algorithm with a sparse LU solver at each iteration.
- `2`: use a Newton algorithm with a Generalized Minimal Residual (GMRES) solver at each iteration. This option is not available under Octave.
- `3`: use a Newton algorithm with a Stabilized Bi-Conjugate Gradient (BICGSTAB) solver at each iteration.
- `4`: use a Newton algorithm with a optimal path length at each iteration.
- `5`: use a Newton algorithm with a sparse Gaussian elimination (SPE) solver at each iteration.

`markowitz` = ***DOUBLE*** Value of the Markowitz criterion, used to select the pivot. Only used when `stack_solve_algo` = 5. Default: `0.5`

`minimal_solving_periods` = ***INTEGER*** Specify the minimal number of periods where the model has to be solved, before using a constant set of operations for the remaining periods. Only used when `stack_solve_algo` = 5. Default: `1`

`datafile` = ***FILENAME*** If the variables of the model are not constant over time, their initial values, stored in a text file, could be loaded, using that option, as initial values before a deteministic simulation.

Output variables

The simulated endogenous variables are available in global matrix `oo_.endo_simul`. The variables are arranged row by row, in order of declaration (as in `M_.endo_names`). Note that this variable also contains initial and terminal conditions, so it has more columns than the value of `periods` option.

4.8.6 stoch_simul

stoch_simul — computes the solution and simulates the model

Synopsis

`stoch_simul` [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*...] ;

Options

ar = ***INTEGER*** Order of autocorrelation coefficients to compute and to print. Default: `5`

drop = ***INTEGER*** Number of points dropped at the beginning of simulation before computing the summary statistics. Default: `100`

hp_filter = ***INTEGER*** Uses HP filter with λ = *INTEGER* before computing moments. Note that this option is currently not available when computing empirical moments (see `periods` option). Default: no filter

hp_ngrid = ***INTEGER*** Number of points in the grid for the discrete Inverse Fast Fourier Transform used in the HP filter computation. It may be necessary to increase it for highly autocorrelated processes. Default: `512`

irf = ***INTEGER*** Number of periods on which to compute the IRFs. Setting `irf`=0, suppresses the plotting of IRF's. Default: `40`

relative_irf Requests the computation of normalized IRFs in percentage of the standard error of each shock

linear Indicates that the original model is linear (put it rather in the model command)

nocorr Don't print the correlation matrix (printing them is the default)

nofunctions Don't print the coefficients of the approximated solution (printing them is the default)

nomoments Don't print moments of the endogenous variables (printing them is the default)

nograph Doesn't do the graphs. Useful for loops

noprint Don't print anything. Useful for loops

print Print results (opposite of the above)

order = ***INTEGER*** Order of Taylor approximation. Acceptable values are `1`, `2` and `3`. Note that for third order, k_order_solver option is implied (in particular, you must specify the use_dll option on the model block and you need a working compilation environment, see Section 2.1 for more details), and only empirical moments are available (you must provide a value for `periods` option). Default: `2`

k_order_solver Use a k-order solver, implemented in C++, instead of the default Dynare solver. When using this option, you must specify the use_dll option on the model block, and you need a working compilation environment, *i.e.* a working `mex` command (see Section 2.1 for more details). Default: disabled for order 1 and 2, enabled otherwise

periods = ***INTEGER*** If different from zero, empirical moments will be computed instead of theoretical moments. The value of the option specifies the number of periods to use in the simulations. Values of the initval block, possibly recomputed by steady, will be used as starting point for the simulation. The simulated endogenous variables are made available to the user in a vector for each variable and in the global matrix `oo_.endo_simul`. The variables in the `oo_.endo_simul` matrix, in their order of declaration (as in `M_.endo_names`) Default: `0`

qz_criterium = *DOUBLE* Value used to split stable from unstable eigenvalues in reordering the Generalized Schur decomposition used for solving 1st order problems. Default: `1.000001`

replic = *INTEGER* Number of simulated series used to compute the IRFs. Default: `1` if `order=1`, and `50` otherwise

simul_seed = *INTEGER* Specifies a seed for the random generator so as to obtain the same random sample at each run of the program. Otherwise a different sample is used for each run. Default: seed not specified

simul_algo = *INTEGER* Obsolete. Use only the default = 0

solve_algo = *INTEGER* See there for the possible values and their meaning

aim_solver Use the Anderson-Moore Algorithm (AIM) to compute the decision rules, instead of using Dynare's default method based on a generalized Schur decomposition. This option is only valid for first order approximation. See AIM website for more details on the algorithm.

conditional_variance_decomposition = *INTEGER* See below

conditional_variance_decomposition = [*INTEGER1*:*INTEGER2*] See below

conditional_variance_decomposition = [*INTEGER1 INTEGER2* ...] Computes a conditional variance decomposition for the specified period(s). Conditional variances are given by $\mathrm{var}(y_{t+k}|t)$. For period 1, the conditional variance decomposition provides the decomposition of the effects of shocks upon impact.

Description

stoch_simul computes a Taylor approximation of the decision and transition functions for the model, impulse response functions and various descriptive statistics (moments, variance decomposition, correlation and autocorrelation coefficients). For correlated shocks, the variance decomposition is computed as in the VAR literature through a Cholesky decomposition of the covariance matrix of the exogenous variables. When the shocks are correlated, the variance decomposition depends upon the order of the variables in the varexo command.

The Taylor approximation is computed around the steady state. If you know how to compute the steady state for your model, you can provide a MATLAB®/Octave function doing the computation instead of using the nonlinear solver. The function should be called with the name of the `.mod` file followed by `_steadystate`. See `fs2000a_steadystate.m` in `examples/fs2000` directory.

The IRFs are computed as the difference between the trajectory of a variable following a shock at the beginning of period 1 and its steady state value.

Variance decomposition, correlation, autocorrelation are only displayed for variables with positive variance. Impulse response functions are only plotted for variables with response larger than 10^{-10}.

Variance decomposition is computed relative to the sum of the contribution of each shock. Normally, this is of course equal to aggregate variance, but if a model generates very large variances, it may happen that, due to numerical error, the two differ by a significant amount. Dynare issues a warning if the maximum relative difference between the sum of the contribution of each shock and aggregate variance is larger than 0.01 %.

Currently, the IRFs are only plotted for 12 variables. Select the ones you want to see, if your model contains more than 12 endogenous variables.

Currently, the HP filter is only available when computing theoretical moments, not for for moments of simulated variables.

The covariance matrix of the shocks is specified either with the shocks command or with the Sigma_e command.

When a list of *VARIABLE_NAME* is specified, results are displayed only for these variables.

Auxiliary variables for leads and lags

For a stochastic model, Dynare will perform a transformation of the model so that there is only one lead and one lag on endogenous, and no lead/lag on exogenous.

This transformation is achieved by the creation of auxiliary variables, and corresponding equations. For example, if `x(+2)` exists in the model, Dynare will create one auxiliary variable `AUX_ENDO_LEAD = x(+1)`, and replace `x(+2)` by `AUX_ENDO_LEAD(+1)`.

A similar transformation is done for lags greater than 2 on endogenous (auxiliary variables will have a name beginning with `AUX_ENDO_LAG`), and for exogenous with leads and lags (auxiliary variables will have a name beginning with `AUX_EXO_LEAG` or `AUX_EXO_LAG` respectively).

Once created, all auxiliary variables are included in the set of endogenous variables. The output of decision rules (see below) is such that auxiliary variable names are replaced by the original variables they refer to.

The number of endogenous variables before the creation of auxiliary variables is stored in `M_.orig_endo_nbr`, and the number of endogenous variables after the creation of auxiliary variables is stored in `M_.endo_nbr`.

Decision rules

The approximated solution of a model takes the form of a set of decision rules or transition equations expressing the current value of the endogenous variables of the model as function of the previous state of the model and shocks oberved at the beginning of the period. The decision rules are stored in the structure `oo_.dr` which is described below.

Typology and ordering of variables

Dynare distinguishes four types of endogenous variables:

Purely backward (or purely predetermined) variables Those that appear only at current and past period in the model, but not at future period (*i.e.* at `t` and `t-1` but not `t+1`). The number of such variables is equal to `oo_.dr.npred - oo_.dr.nboth`.

Purely forward variables Those that appear only at current and future period in the model, but not at past period (*i.e.* at `t` and `t+1` but not `t-1`). The number of such variables is stored in `oo_.dr.nfwrd`.

Mixed variables Those that appear at current, past and future period in the model (*i.e.* at `t`, `t+1` and `t-1`). The number of such variables is stored in `oo_.dr.nboth`.

Static variables Those that appear only at current, not past and future period in the model (*i.e.* only at `t`, not at `t+1` or `t-1`). The number of such variables is stored in `oo_.dr.nstatic`.

Note that all endogenous variables fall into one of these four categories, since after the creation of auxiliary variables, all endogenous have at most one lead and one lag. We therefore have the following identity: `oo_.dr.npred + oo_.dr.nfwrd + oo_.dr.nstatic = M_.endo_nbr`.

Internally, Dynare uses two orderings of the endogenous variables: the order of declaration (which is reflected in `M_.endo_names`), and an order based on the four types described above, which we will call the DR-order ("DR" stands for decision rules). Most of the time, the declaration order is used, but for elements of the decision rules, the DR-order is used.

The DR-order is the following: static variables appear first, then purely backward variables, then mixed variables, and finally purely forward variables. Inside each category, variables are arranged according to the declaration order.

Variable `oo_.dr.order_var` maps DR-order to declaration order, and variable `oo_.dr.inv_order_var` contains the inverse map. In other words, the k-th variable in the DR-order corresponds to the endogenous variable numbered `oo_.dr_order_var(k)` in declaration order. Conversely, k-th declared variable is numbered `oo_.dr.inv_order_var(k)` in DR-order.

Finally, the state variables of the model are the purely backward variables and the mixed variables. They are orderer in DR-order when they appear in decision rules elements. There are `oo_.dr.npred` such variables.

First order approximation

The approximation has the form:

$$y_t = ys + A\ yh_{t-1} + B\ u_t$$

where ys is the steady state value of y and yh_t=y_t-ys.

The coefficients of the decision rules are stored as follows:

- ys is stored in `oo_.dr.ys`. The vector rows correspond to all endogenous in the declaration order.
- A is stored in `oo_.dr.ghx`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to state variables in DR-order.
- B is stored `oo_.dr.ghu`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to exogenous variables in declaration order.

Second order approximation

The approximation has the form:

$$y_t = ys + 0.5\ \Delta^2 + A\ yh_{t-1} + B\ u_t + 0.5\ C\ (yh_{t-1} \otimes yh_{t-1}) + 0.5\ D\ (u_t \otimes u_t) + E\ (yh_{t-1} \otimes u_t)$$

where ys is the steady state value of y, yh_t=y_t-ys, and Δ^2 is the shift effect of the variance of future shocks.

The coefficients of the decision rules are stored in the variables described for first order approximation, plus the following variables:

- Δ^2 is stored in `oo_.dr.ghs2`. The vector rows correspond to all endogenous in DR-order.
- C is stored in `oo_.dr.ghxx`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to the Kronecker product of the vector of state variables in DR-order.
- D is stored in `oo_.dr.ghuu`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to the Kronecker product of exogenous variables in declaration order.
- E is stored in `oo_.dr.ghxu`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to the Kronecker product of the vector of state variables (in DR-order) by the vector of exogenous variables (in declaration order).

Third order approximation

The approximation has the form:

$$y_t = ys + G_0 + G_1\ z_t + G_2\ (z_t \otimes z_t) + G_3\ (z_t \otimes z_t \otimes z_t)$$

where ys is the steady state value of y, and z_t is a vector consisting of the deviation from the steady state of the state variables (in DR-order) at date t-1 followed by the exogenous variables at date t (in declaration order). The vector z_t is therefore of size n_z = `oo_.dr.npred + M_.exo_nbr`.

The coefficients of the decision rules are stored as follows:

- ys is stored in `oo_.dr.ys`. The vector rows correspond to all endogenous in the declaration order.
- G_0 is stored in `oo_.dr.g_0`. The vector rows correspond to all endogenous in DR-order.

- G_1 is stored in `oo_.dr.g_1`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to state variables in DR-order, followed by exogenous in declaration order.
- G_2 is stored in `oo_.dr.g_2`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to the Kronecker product of state variables (in DR-order), followed by exogenous (in declaration order). Note that the Kronecker product is stored in a folded way, *i.e.* symmetric elements are stored only once, which implies that the matrix has $n_z(n_z+1)/2$ columns. More precisely, each column of this matrix corresponds to a pair (i_1, i_2) where each index represents an element of z_t and is therefore between 1 and n_z. Only non-decreasing pairs are stored, *i.e.* those for which $i_1 \leq i_2$. The columns are arranged in the lexicographical order of non-decreasing pairs. Also note that for those pairs where $i_1 \neq i_2$, since the element is stored only once but appears two times in the unfolded G_2 matrix, it must be multiplied by 2 when computing the decision rules.
- G_3 is stored in `oo_.dr.g_3`. The matrix rows correspond to all endogenous in DR-order. The matrix columns correspond to the third Kronecker power of state variables (in DR-order), followed by exogenous (in declaration order). Note that the third Kronecker power is stored in a folded way, *i.e.* symmetric elements are stored only once, which implies that the matrix has $n_z(n_z+1)(n_z+2)/6$ columns. More precisely, each column of this matrix corresponds to a tuple (i_1, i_2, i_3) where each index represents an element of z_t and is therefore between 1 and n_z. Only non-decreasing tuples are stored, *i.e.* those for which $i_1 \leq i_2 \leq i_3$. The columns are arranged in the lexicographical order of non-decreasing tuples. Also note that for tuples that have three distinct indices (*i.e.* $i_1 \neq i_2$ and $i_1 \neq i_3$ and $i_2 \neq i_3$), since these elements are stored only once but appears six times in the unfolded G_3 matrix, they must be multiplied by 6 when computing the decision rules. Similarly, for those tuples that have two equal indices (*i.e.* of the form (a,a,b) or (a,b,a) or (b,a,a)), since these elements are stored only once but appears three times in the unfolded G_3 matrix, they must be multiplied by 3 when computing the decision rules.

Other output variables

stoch_simul sets other fields in global variable `oo_`. The descriptive statistics are theoretical moments when no simulation is requested and otherwise represent the moments of the simulated variables.

- The mean of the endogenous variables is available in the vector `oo_.mean`. The variables are arranged in declaration order.
- The matrix of variance-covariance of the endogenous variables in the matrix `oo_.var`. The variables are arranged in declaration order.
- The matrix of autocorrelation of the endogenous variables are made available in cell array `oo_.autocorr`. The element number of the matrix in the cell array corresponds to the order of autocorrelation. The option `ar` specifies the number of autocorrelation matrices available.
- Simulated variables, when they have been computed, are available in MATLAB®/Octave vectors in the global workspace with the same name as the endogenous variables. They are also available in the `oo_.endo_simul` matrix. The series are arranged by row, in declaration order of the variable names
- Impulse responses, when they have been computed, are available in `oo_.irfs`, with the following naming convention: *VARIABLE_NAME_SHOCK_NAME*.

  (DEPRECATED) They are currently also available in MATLAB®/Octave vectors in the global workspace, however they will disappear there in a future version.

  Example: `oo_.irfs.gnp_ea` contains the effect on `gnp` of a one standard deviation shock on `ea`.

Examples

Example 1

```
shocks;
var e;
stderr 0.0348;
end;

stoch_simul;
```

Performs the simulation of the 2nd order approximation of a model with a single stochastic shock `e`, with a standard error of 0.0348.

Example 2

```
stoch_simul(linear,irf=60) y k;
```

Performs the simulation of a linear model and displays impulse response functions on 60 periods for variables `y` and `k`.

4.9 Estimation

Provided that you have observations on some endogenous variables, it is possible to use Dynare to estimate some or all parameters. Both maximum likelihood and Bayesian techniques are available.

Note that in order to avoid stochastic singularity, you must have at least as many shocks or measurement errors in your model as you have observed variables.

- varobs
- observation_trends
- estimated_params
- estimated_params_init
- estimated_params_bounds
- estimation
- model_comparison
- shock_decomposition
- unit_root_vars (deprecated)

4.9.1 varobs

varobs — lists the observed variables

Synopsis

`varobs` *VARIABLE_NAME*... ;

Description

varobs lists the name of observed endogenous variables for the estimation procedure. These variables must be available in the data file (see estimation).

Successive instances of **varobs** overwrite the list of observed variables. They don't add to the list.

Example

```
varobs C y rr;
```

4.9.2 observation_trends

observation_trends — specifies linear trends for observed variables

Synopsis

```
observation_trends ;
VARIABLE_NAME(EXPRESSION); ...

end ;
```

Description

observation_trends specifies trends for observed variables as functions of model parameters. In most cases, variables shouldn't be centered when **observation_trends** is used.

Example

```
observation_trends;
Y (eta);
P (mu/eta);
end;
```

4.9.3 estimated_params

estimated_params — specifies the estimated parameters and their prior

Synopsis

Syntax I (Maximum likelihood estimation)

```
estimated_params ;
stderrVARIABLE_NAME | corrVARIABLE_NAME_1, VARIABLE_NAME_2 | PARAMETER_NAME , INITIAL_VALUE [ , LOWER_BOUND , UPPER_BOUND] ; ...

end ;
```

Syntax II (Bayesian estimation)

```
estimated_params ;
stderrVARIABLE_NAME | corrVARIABLE_NAME_1, VARIABLE_NAME_2 | PARAMETER_NAME [ , INITIAL_VALUE [ , LOWER_BOUND , UPPER_BOUND]] , (1)PRIOR_SHAPE , PRIOR_MEAN , PRIOR_STANDARD_ERROR [ , PRIOR_3RD_PARAMETER [ , PRIOR_4TH_PARAMETER [ , SCALE_PARAMETER]]] ; ...

end ;
```

(1) `beta_pdf | gamma_pdf | normal_pdf | uniform_pdf | inv_gamma_pdf | inv_gamma1_pdf | inv_gamma2_pdf`

Description

The **estimated_params** block lists all parameters to be estimated and specifies bounds and priors as necessary.

Estimated parameter specification

Each line corresponds to an estimated parameter and follows this syntax:

stderr *VARIABLE_NAME* Indicates that the standard error of the exogenous variable *VARIABLE_NAME*, or of the observation error associated with endogenous observed variable *VARIABLE_NAME*, is to be estimated

corr *VARIABLE_NAME_1*, *VARIABLE_NAME_2* Indicates that the correlation between the exogenous variables *VARIABLE_NAME_1* and *VARIABLE_NAME_2*, or the correlation of the observation errors associated with endogenous observed variables *VARIABLE_NAME_1* and *VARIABLE_NAME_2*, is to be estimated

PARAMETER_NAME The name of a model parameter to be estimated

INITIAL_VALUE Specifies a starting value for maximum likelihood estimation

LOWER_BOUND Specifies a lower bound for the parameter value in maximum likelihood estimation

UPPER_BOUND Specifies an upper bound for the parameter value in maximum likelihood estimation

PRIOR_SHAPE A keyword specifying the shape of the prior density. See the list of possible values. Note that `inv_gamma_pdf` is equivalent to `inv_gamma1_pdf`

PRIOR_MEAN The mean of the prior distribution

PRIOR_STANDARD_ERROR The standard error of the prior distribution

PRIOR_3RD_PARAMETER A third parameter of the prior used for generalized beta distribution, generalized gamma and for the uniform distribution. Default: `0`

PRIOR_4TH_PARAMETER A fourth parameter of the prior used for generalized beta distribution and for the uniform distribution. Default: `1`

SCALE_PARAMETER The scale parameter to be used for the jump distribution of the Metropolis-Hasting algorithm

---

Note

INITIAL_VALUE, *LOWER_BOUND*, *UPPER_BOUND*, *PRIOR_MEAN*, *PRIOR_STANDARD_ERROR*, *PRIOR_3RD_PARAMETER*, *PRIOR_4TH_PARAMETER* and *SCALE_PARAMETER* can be any valid *EXPRESSION*. Some of them can be empty, in which Dynare will select a default value depending on the context and the prior shape.

Note

At minimum, one must specify the name of the parameter and an initial guess. That will trigger unconstrained maximum likelihood estimation.

Note

As one uses options more towards the end of the list, all previous options must be filled: for example, if you want to specify *SCALE_PARAMETER*, you must specify *PRIOR_3RD_PARAMETER* and *PRIOR_4TH_PARAMETER*. Use empty values, if these parameters don't apply.

Parameter transformation

Sometimes, it is desirable to estimate a transformation of a parameter appearing in the model, rather than the parameter itself. It is of course possible to replace the original parameter by a function of the estimated parameter everywhere is the model, but it is often unpractical.

In such a case, it is possible to declare the parameter to be estimated in the parameters statement and to define the transformation, using a pound sign (#) expression (see model).

Example

```
parameters bet;

model;
# sig = 1/bet;
c = sig*c(+1)*mpk;
end;

estimated_params;
bet, normal_pdf, 1, 0.05;
end;
```

4.9.4 estimated_params_init

estimated_params_init — specifies initial values for optimization

Synopsis

```
estimated_params_init ;
stderr VARIABLE_NAME | corr VARIABLE_NAME_1 , VARIABLE_NAME_2 | PARAMETER_NAME , INITIAL_VALUE ; ...

end ;
```

Description

The **estimated_params_init** block declares numerical initial values for the optimizer when these ones are different from the prior mean.

Estimated parameter initial value specification

See estimated_params for the meaning and syntax of the various components.

4.9.5 estimated_params_bounds

estimated_params_bounds — specifies lower and upper bounds for the estimated parameters

Synopsis

```
estimated_params_bounds ;
stderr VARIABLE_NAME | corr VARIABLE_NAME_1 , VARIABLE_NAME_2 | PARAMETER_NAME , LOWER_BOUND , UPPER_BOUND ; ...

end ;
```

Description

The **estimated_params_bounds** block declares lower and upper bounds for parameters in maximum likelihood estimation.

Estimated parameter bounds specification

See estimated_params for the meaning and syntax of the various components.

4.9.6 estimation

estimation — computes estimation

Synopsis

`estimation` [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*...] ;

Options

datafile = ***FILENAME*** The datafile (a `.m` file, a `.mat` file or a `.xls` file)

xls_sheet = ***NAME*** The name of the sheet with the data in an Excel file

xls_range = ***RANGE*** The range with the data in an Excel file

nobs = ***INTEGER*** The number of observations to be used. Default: all observations in the file

nobs = [***INTEGER_1*:*INTEGER_2***] Runs a recursive estimation and forecast for samples of size ranging of *INTEGER_1* to *INTEGER_2*. Option `forecast` must also be specified

first_obs = ***INTEGER*** The number of the first observation to be used. Default: `1`

prefilter = ***INTEGER*** A value of `1` means that the estimation procedure will demean the data. Default: `0`, *i.e.* no prefiltering

presample = ***INTEGER*** The number of observations to be skipped before evaluating the likelihood. Default: `0`

loglinear Computes a log--linear approximation of the model instead of a linear approximation. The data must correspond to the definition of the variables used in the model. Default: computes a linear approximation

nograph No graphs should be plotted

lik_init = ***INTEGER*** Type of initialization of Kalman filter:

- `1`: for stationary models, the initial matrix of variance of the error of forecast is set equal to the unconditional variance of the state variables
- `2`: for nonstationary models: a wide prior is used with an initial matrix of variance of the error of forecast diagonal with 10 on the diagonal

Default value is `1`.

lik_algo = ***INTEGER*** ...

conf_sig = ***DOUBLE*** See there

mh_replic = ***INTEGER*** Number of replications for Metropolis-Hastings algorithm. For the time being, `mh_replic` should be larger than `1200`. Default: `20000`

mh_nblocks = ***INTEGER*** Number of parallel chains for Metropolis-Hastings algorithm. Default: `2`

mh_drop = ***DOUBLE*** The fraction of initially generated parameter vectors to be dropped before using posterior simulations. Default: `0.5`

mh_jscale = ***DOUBLE*** The scale to be used for the jumping distribution in Metropolis-Hastings algorithm. The default value is rarely satisfactory. This option must be tuned to obtain, ideally, an acceptation rate of 25% in the Metropolis-Hastings algorithm. Default: `0.2`

mh_init_scale = *DOUBLE* The scale to be used for drawing the initial value of the Metropolis-Hastings chain. Default: 2*mh_scale

mh_recover Attempts to recover a Metropolis-Hastings simulation that crashed prematurely. Shouldn't be used together with load_mh_file

mh_mode = *INTEGER* ...

mode_file = *FILENAME* Name of the file containing previous value for the mode. When computing the mode, Dynare stores the mode (xparam1) and the hessian (hh) in a file called MODEL_FILENAME_mode.mat

mode_compute = *INTEGER* | *FUNCTION_NAME* Specifies the optimizer for the mode computation:

- 0: the mode isn't computed. mode_file option must be specified
- 1: uses MATLAB®'s **fmincon**
- 2: value no longer used
- 3: uses MATLAB®'s **fminunc**
- 4: uses Chris Sims's **csminwel**
- 5: uses Marco Ratto's **newrat**
- 6: uses a Monte-Carlo based optimization routine (see Dynare wiki for more details)
- 7: uses MATLAB®'s **fminsearch** (a simplex based routine)
- It is also possible to give a *FUNCTION_NAME* to this option, instead of an *INTEGER*. In that case, Dynare takes the return value of that function as the posterior mode.

Default value is 4.

mode_check Tells Dynare to plot the posterior density for values around the computed mode for each estimated parameter in turn. This is helpful to diagnose problems with the optimizer

prior_trunc = *DOUBLE* Probability of extreme values of the prior density that is ignored when computing bounds for the parameters. Default: 1e-32

load_mh_file Tells Dynare to add to previous Metropolis-Hastings simulations instead of starting from scratch. Shouldn't be used together with mh_recover

optim = (*fmincon options*) Can be used to set options for **fmincon**, the optimizing function of MATLAB® Optimizaiton toolbox. Use MATLAB®'s syntax for these options. Default: ('display','iter','LargeScale','off','MaxFunEvals',100000,'TolFun',1e-8,'TolX',1e-6)

nodiagnostic Doesn't compute the convergence diagnostics for Metropolis-Hastings. Default: diagnostics are computed and displayed

bayesian_irf Triggers the computation of the posterior distribution of IRFs. The length of the IRFs are controlled by the irf option

moments_varendo Triggers the computation of the posterior distribution of the theoretical moments of the endogenous variables

filtered_vars Triggers the computation of the posterior distribution of filtered endogenous variables and shocks

smoother Triggers the computation of the posterior distribution of smoothered endogenous variables and shocks

forecast = *INTEGER* Computes the posterior distribution of a forecast on *INTEGER* periods after the end of the sample used in estimation

tex Requests the printing of results and graphs in TeX tables and graphics that can be later directly included in LaTeX files (not yet implemented)

kalman_algo = *INTEGER* ...

kalman_tol = *INTEGER* ...

filter_step_ahead = [*INTEGER_1*:*INTEGER_2*] ...

constant ...

noconstant ...

diffuse_filter ...

solve_algo = *INTEGER* See there

order = *INTEGER* See there

irf = *INTEGER* See there

aim_solver See there

Note
If no `mh_jscale` parameter is used in estimated_params, the procedure uses `mh_jscale` for all parameters. If `mh_jscale` option isn't set, the procedure uses `0.2` for all parameters.

Results

- results from posterior optimization (also for maximum likelihood)
- marginal log density
- mean and shortest confidence interval from posterior simulation
- Metropolis-Hastings convergence graphs that still need to be documented
- graphs with prior, posterior and mode
- graphs of smoothed shocks, smoothed observation errors, smoothed and historical variables

Output

After running **estimation**, the parameters and the variance matrix of the shocks are set to the mode for maximum likelihood estimation or posterior mode computation without Metropolis iterations.

After **estimation** with Metropolis iterations (option `mh_replic` > 0 or option `load_mh_file` set) the parameters and the variance matrix of the shocks are set to the posterior mean.

Depending on the options, **estimation** stores results in the following fields of structure `oo_`:

Field 1	Field 2	Field 3	Field 4	Field 5	Required options
Forecast	See Moments of forecasts	*Variable name*			forecast
MarginalDensity	LaplaceApproximation				Always provided
	ModifiedHarmonicMean				mh_replic> 0 or load_mh_file
FilteredVariables	See Moments Names	*Variable name*			filtered_vars
PosteriorIRF	Dsge	See Moments Names	IRF name: name of endogenous variable '_' name of shock		bayesian_irf
SmoothedMeasurementErrors	See Moments Names	*Variable name*			smoother
SmoothedShocks	See Moments Names	*Variable name*			smoother
SmoothedVariables	See Moments Names	*Variable name*			smoother
PosteriorTheoreticalMoments	See Theoretical Moments	See Estimated Objects	See Moments Names	*Variable name*	moments_varendo
posterior_density	*Parameter name*				mh_replic> 0 or load_mh_file
posterior_hpdinf	See Estimated Objects	*Variable name*			mh_replic> 0 or load_mh_file
posterior_hpdsup	See Estimated Objects	*Variable name*			mh_replic> 0 or load_mh_file
posterior_mean	See Estimated Objects	*Variable name*			mh_replic> 0 or load_mh_file
posterior_mode	See Estimated Objects	*Variable name*			mh_replic> 0 or load_mh_file
posterior_std	See Estimated Objects	*Variable name*			mh_replic> 0 or load_mh_file

Table 4.1: Content of oo_

Field name	Description
HPDinf	Lower bound of a 90% HPD interval[4] of forecast due to parameter uncertainty
HPDsup	Lower bound of a 90% HPD interval [4] due to parameter uncertainty
HPDTotalinf	Lower bound of a 90% HPD interval of forecast [4] due to parameter uncertainty and future shocks
HPDTotalsup	Lower bound of a 90% HPD interval [4] due to parameter uncertainty and future shocks
Mean	Mean of the posterior distribution of forecasts
Median	Median of the posterior distribution of forecasts
Std	Standard deviation of the posterior distribution of forecasts

Table 4.2: Moments of forecasts

Field name	Description
HPDinf	Lower bound of a 90% HPD interval[5]
HPDsup	Upper bound of a 90% HPD interval [5]
Mean	Mean of the posterior distribution
Median	Median of the posterior distribution
Std	Standard deviation of the posterior distribution

Table 4.3: Moments Names

Field name	Description
Autocorrelation	Autocorrelation of endogenous variables[6]
Correlation	Correlation between two endogenous variables
Decomp	Decomposition of variance[7]
Expectation	Expectation of endogenous variables
Variance	(co-)variance of endogenous variables

Table 4.4: Theoretical Moments

Examples

```
oo_.posterior_mode.parameters.alp
oo_.posterior_mean.shocks_std.ex
oo_.posterior_hpdsup.measurement_errors_corr.gdp_conso
```

Note on steady state computation

If you know how to compute the steady state for your model, you can provide a MATLAB® function doing the computation instead of using **steady**. The function should be called with the name of the `.mod` file followed by `_steadystate`. See `fs2000a_steadystate.m` in `examples/fs2000` directory.

4.9.7 model_comparison

model_comparison — Bayesian model comparison

[4] See option conf_sig to change the size of the HPD interval
[5] See option conf_sig to change the size of the HPD interval
[6] The autocorrlation coefficients are computed for the number of periods specified in option ar.
[7] When the shocks are correlated, it is the decomposition of orthogonalized shocks via Cholesky decompostion according to the order of declaration of shocks (see varexo).

Field name	Description
measurement_errors_corr	Correlation between two measurement errors
measurement_errors_std	Standard deviation of measurement errors
parameters	Parameters
shocks_corr	Correlation between two structural shocks
shocks_std	Standard deviation of structural shocks

Table 4.5: Estimated objects

Synopsis

model_comparison [(marginal_density = laplace | modifiedharmonicmean)] [*FILENAME* [(*DOUBLE*)] [[,]*FILENAME* [(*DOUBLE*)]...]] ;

Description

This function computes odds ratios and estimate a posterior density over a colletion of models. The priors over models can be specified as the *DOUBLE* values, otherwise a uniform prior is assumed.

4.9.8 shock_decomposition

shock_decomposition — computes and displays shock decomposition according to the model for a given sample

Synopsis

shock_decomposition [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*...] ;

Options

parameters = ***PARAMETER_NAME*** ...

shocks = (***VARIABLE_NAME*** [***VARIABLE_NAME*** **...**] [**;** ***VARIABLE_NAME*** [***VARIABLE_NAME*** **...**] **...**]) ...

labels = (***VARIABLE_NAME*** [***VARIABLE_NAME*** **...**]) ...

Description

...

4.9.9 unit_root_vars

unit_root_vars — declares unit-root variables for estimation

Synopsis

unit_root_vars *VARIABLE_NAME* [[,]*VARIABLE_NAME*...] ;

Description

unit_root_vars is now deprecated and will result in no action, It was used to declare unit-root variables of a model so that a diffuse prior can be used in the initialization of the Kalman filter for these variables only. For stationary variables, the unconditional covariance matrix of these variables is used for initialization. The algorithm to compute a true diffuse prior is taken from Durbin and Koopman (2001) and Koopman and Durbin (2003).

When **unit_root_vars** is used the lik_init option of estimation has no effect.

When there are nonstationary variables in a model, there is no unique deterministic steady state. The user must supply a MATLAB® function that computes the steady state values of the stationary variables in the model and returns dummy values for the nonstationary ones. The function should be called with the name of the `.mod` file followed by `_steadystate`. See `fs2000a_steadystate.m` in `examples/fs2000` directory.

Note that the nonstationary variables in the model must be integrated processes(their first difference or k-difference must be stationary).

4.10 Forecasting

On a calibrated model, forecasting is done using the forecast command. On an estimated command, use the forecast option of estimation command.

It is also possible to compute forecasts on a calibrated or estimated model for a given constrained path of the future endogenous variables. This is done, from the reduced form representation of the DSGE model, by finding the structural shocks that are needed to match the restricted paths. Use conditional_forecast, conditional_forecast_paths and plot_conditional_forecast for that purpose.

4.10.1 forecast

forecast — computes a simulation of a stochastic model from a given state

Synopsis

`forecast` [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*] [[,]*VARIABLE_NAME*...] ;

Options

periods = ***INTEGER*** Number of periods of the forecast. Default: `40`

conf_sig = ***DOUBLE*** Level of significance for confidence interval. Default: `0.90`

nograph Don't display graphics.

Description

forecast computes a simulation of a stochastic model from an arbitrary initial point.

When the model also contains deterministic exogenous shocks, the simulation is computed conditionaly to the agents knowing the future values of the deterministic exogenous variables.

forecast must be called after stoch_simul.

forecast plots the trajectory of endogenous variables. When a list of variable names follows the command, only those variables are plotted. A 90% confidence interval is plotted around the mean trajectory. Use option `conf_sig` to change the level of the confidence interval.

Output variables

The following variables are set in structure `oo_`:

- `oo_.forecast.Mean.`*`VARIABLE_NAME`*: mean forecast of endogenous variables
- `oo_.forecast.HPDinf.`*`VARIABLE_NAME`*: lower bound of a confidence interval around the forecast
- `oo_.forecast.HPDsup.`*`VARIABLE_NAME`*: upper bound of a confidence interval around the forecast
- `oo_.forecast.Exogenous.`*`VARIABLE_NAME`*: trajectory of the deterministic exogenous variables

Example

```
varexo_det tau;
varexo e;

...

shocks;
var e; stderr 0.01;
var tau;
periods 1:9;
values -0.15;
end;

stoch_simul(irf=0);

forecast;
```

4.10.2 conditional_forecast

conditional_forecast — computes a simulation of a stochastic model conditionally to a specified future path for some endogenous variables.

Synopsis

`conditional_forecast` [(*`OPTION`* [, *`OPTION`*...])] *`VARIABLE_NAME`* [[,]*`VARIABLE_NAME`*...] ;

Options

`parameter_set` = **`prior_mode`** | **`prior_mean`** | **`posterior_mode`** | **`posterior_mean`** | **`posterior_median`** Specify the parameter set to use for the forecasting. No default value, mandatory option.

`controlled_varexo` = (***`VARIABLE_NAME`*** [[,] ***`VARIABLE_NAME`*** ...]) Specify the exogenous variables to use as control variables. No default value, mandatory option.

`periods` = ***`INTEGER`*** Number of periods of the forecast. Default: `40`. `periods` cannot be less than the number of constrained periods.

`replic` = ***`INTEGER`*** Number of simulations. Default: `5000`.

`conf_sig` = ***`DOUBLE`*** Level of significance for confidence interval. Default: `0.80`

Description

conditional_forecast computes forecasts on an estimated model for a given constrained path of some future endogenous variables. This is done, from the reduced form representation of the DSGE model, by finding the structural shocks that are needed to match the restricted paths. This command has to be called after estimation.

Use conditional_forecast_paths to give the list of constrained endogenous, and their constrained future path. Option `controlled_varexo` is used to specify the structural shocks which will be matched to generate the constrained path.

Use plot_conditional_forecast to graph the results.

Example

```
var y a
varexo e u;

...

estimation(...);

conditional_forecast_paths;
var y;
periods 1:3, 4:5;
values 2, 5;
var a;
periods 1:5;
values 3;
end;

conditional_forecast(parameter_set = calibration, controlled_varexo = (e, u), replic =  ↩
    3000);

plot_conditional_forecast(periods = 10) e u;
```

4.10.3 conditional_forecast_paths

conditional_forecast_paths — in a conditional forecast, gives the list of constrained endogenous and their path

Synopsis

`conditional_forecast_paths` ;
[var *VARIABLE_NAME*; periods *INTEGER* [:*INTEGER*] [[,]*INTEGER* [:*INTEGER*]...]; values *EXPRESSION* [[,]*EXPRESSION*...]; ...]

`end` ;

Description

Describes the path of constrained endogenous, before calling conditional_forecast. The syntax is similar to deterministic shocks in shocks, see conditional_forecast for an example.

4.10.4 plot_conditional_forecast

plot_conditional_forecast — plots the conditional forecasts

Synopsis

`plot_conditional_forecast` [(`periods` = *INTEGER*)] ;

Options

`periods` = *INTEGER* Number of periods to be plotted. Default: equal to `periods` in **conditional_forecast**. The number of periods declared in **plot_conditional_forecast** cannot be greater than the one declared in **conditional_forecast**.

Description

To be used after conditional_forecast.

4.11 Optimal policy

Dynare has tools to compute optimal policies for quadratic objectives. You can either solve for optimal policy under commitment with **planner_objective** or for optimal simple rule with osr.

- optim_weights
- osr
- osr_params
- planner_objective
- ramsey_policy

4.11.1 optim_weights

optim_weights — specifies quadratic objectives for optimal policy problems

Synopsis

`optim_weights` ;
[*VARIABLE_NAMEEXPRESSION*; | *VARIABLE_NAME*, *VARIABLE_NAMEEXPRESSION*; ...] `end` ;

Description

optim_weights secifies the nonzero elements of the quadratic weight matrices for the objectives in osr.

4.11.2 osr

osr — computes optimal simple policy rules

Synopsis

`osr` [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*...] ;

Options

All options for stoch_simul.

Description

osr computes optimal simple policy rules for linear--quadratic problems of the form:

$\max_{\gamma} E(y'_t W y_t)$
s.t.
$A_1 E_t(y_{t+1}) + A_2 y_t + A_3 y_{t-1} + C e_t = 0$

with:

- γ: parameters to be optimized. They must be elements of matrices A_1, A_2, A_3.
- y: endogenous variables
- e: exogenous stochastic shocks

The parameters to be optimized must be listed with osr_params.
The quadratic objectives must be listed with optim_weights.
This problem is solved using a numerical optimizer.

4.11.3 osr_params

osr_params — declares the parameters to be optimized for optimal simple rules

Synopsis

`osr_params` *PARAMETER_NAME* [*PARAMETER_NAME*...] ;

Description

osr_params declares parameters to be optimized by osr.

4.11.4 planner_objective

planner_objective — declares the policy maker objective, for use with ramsey_policy

Synopsis

`planner_objective` *MODEL_EXPRESSION* ;

Description

...

4.11.5 ramsey_policy

ramsey_policy — computes the first order approximation of the policy that maximizes the policy maker objective function (see planner_objective) submitted to the constraints provided by the equilibrium path of the economy

Synopsis

`ramsey_policy` [(*OPTION* [, *OPTION*...])] [*VARIABLE_NAME*...] ;

Options

All options for stoch_simul, plus:

planner_discount = ***DOUBLE*** Declares the discount factor of the central planner. Default: `1.0`

Note that only first order approximation is available (*i.e.* `order=1` must be specified).

4.12 Sensitivity and identification analysis

4.12.1 dynare_sensitivity

dynare_sensitivity — interface to the global sensitivity analysis (GSA) toolbox

Synopsis

`dynare_sensitivity` [(*OPTION* [, *OPTION*...])] ;

Description

This function is an interface to the global sensitivity analysis (GSA) toolbox developed by the Joint Research Center (JRC) of the European Commission. The GSA toolbox needs to be downloaded separately from the JRC web site.

Options

Please refer to the documentation of the GSA toolbox on the official website.

4.12.2 identification

identification — triggers identification analysis

Synopsis

`identification` [(*OPTION* [, *OPTION*...])] ;

Options

ar = ***INTEGER*** Number of lags of computed autocorrelations (theoretical moments). Default: `3`

useautocorr = ***INTEGER*** If equal to `1`, compute derivatives of autocorrelation. If equal to `0`, compute derivatives of autocovariances. Default: `1`

load_ident_files = ***INTEGER*** If equal to `1`, allow Dynare to load previously computed analyzes. Default: `0`

prior_mc = ***INTEGER*** Size of Monte Carlo sample. Default: `2000`

4.13 Displaying and saving results

Dynare has comments to plot the results of a simulation and to save the results.

- rplot
- dynatype
- dynasave

4.13.1 rplot

rplot — plot variables

Synopsis

`rplot` *VARIABLE_NAME* [*VARIABLE_NAME*...] ;

Description

Plots the simulated path of one or several variables.

4.13.2 dynatype

dynatype — print simulated variables

Synopsis

`dynatype` (*FILENAME*) [*VARIABLE_NAME*...] ;

Description

dynatype prints the listed variables in a text file named *FILENAME*. If no *VARIABLE_NAME* is listed, all endogenous variables are printed.

4.13.3 dynasave

dynasave — save simulated variables in a binary file

Synopsis

`dynasave` (*FILENAME*) [*VARIABLE_NAME*...] ;

Description

dynasave saves the listed variables in a binary file named *FILENAME*. If no *VARIABLE_NAME* are listed, all endogenous variables are saved.

In MATLAB®, variables saved with the **dynasave** command can be retrieved by the MATLAB® command **load -mat *FILENAME***.

4.14 Macro-processing language

It is possible to use "macro" commands in the `.mod` file for doing the following tasks: source file inclusion, replicating blocks of equations through loops, conditional inclusion of code...

Technically, this macro language is totally independent of the basic Dynare language, and is processed by a separate component of the Dynare pre-processor. The macro processor transforms a `.mod` file with macros into a `.mod` file without macros (doing expansions/inclusions), and then feeds it to the Dynare parser.

- @#include
- @#define
- @#if ... @#else ... @#endif
- @#for ... @#endfor
- @#echo
- @#error

4.14.1 @#include

@#include — includes another file

Description

...

4.14.2 @#define

@#define — defines a macro-variable

Description

...

4.14.3 @#if ... @#else ... @#endif

@#if ... @#else ... @#endif — conditional inclusion of some part of the `.mod` file

Description

...

4.14.4 @#for ... @#endfor

@#for ... @#endfor — loop for replications of portions of the `.mod` file

Description

...

4.14.5 @#echo

@#echo — asks the preprocessor to display some message on standard output

Description

...

4.14.6 @#error

@#error — asks the preprocessor to display some error message on standard output and to abort

Description

...

4.15 Misc commands

- save_params_and_steady_state
- load_params_and_steady_state
- bvar_density
- bvar_forecast

4.15.1 save_params_and_steady_state

save_params_and_steady_state — saves the values of the parameters and of the computed steady-state in a file

Synopsis

`save_params_and_steady_state` *`FILENAME`* `;`

Description

For all parameters, endogenous and exogenous variables, stores their value in a text file, using a simple name/value associative table.

- for parameters, the value is taken from the last parameter initialization
- for exogenous, the value is taken from the last initval block
- for endogenous, the value is taken from the last steady state computation (or, if no steady state has been computed, from the last initval block)

Note that no variable type is stored in the file, so that the values can be reloaded (with load_params_and_steady_state) in a setup where the variable types are different.

The typical usage of this function is to compute the steady-state of a model by calibrating the steady-state value of some endogenous variables (which implies that some parameters must be endogeneized during the steady-state computation).

You would then write a first `.mod` file which computes the steady state and saves the result of the computation at the end of the file, using **save_params_and_steady_state**.

In a second file designed to perform the actual simulations, you would use load_params_and_steady_state just after your variable declarations, in order to load the steady state previously computed (including the parameters which had been endogeneized during the steady state computation).

The need for two separate `.mod` files arises from the fact that the variable declarations differ between the files for steady state calibration and for simulation (the set of endogenous and parameters differ between the two); this leads to different var and parameters statements.

Also note that you can take advantage of the @#include directive to share the model equations between the two files.

4.15.2 load_params_and_steady_state

load_params_and_steady_state — loads the values of the parameters and of the steady-state from a file

Synopsis

`load_params_and_steady_state` *FILENAME* ;

Description

For all parameters, endogenous and exogenous variables, loads their value from a file created with **save_params_and_steady_state**.

- for parameters, their value will be initialized as if they had been calibrated in the `.mod` file
- for endogenous and exogenous, their value will be initialized as they would have been from an initval block

This function is used in conjunction with save_params_and_steady_state; see the documentation of that function for more information.

4.15.3 bvar_density

bvar_density — computes the marginal density of an estimated BVAR model, using Minnesota priors

Description

...

4.15.4 bvar_forecast

bvar_forecast — computes in-sample or out-sample forecasts for an estimated BVAR model, using Minnesota priors

Description

...

Chapter 5

Examples

Dynare comes with a database of example `.mod` files, which are designed to show a broad range of Dynare features, and are taken from academic papers for most of them. You should have these files in the `examples` subdirectory of your distribution.

Here is a short list of the examples included. For a more complete description, please refer to the comments inside the files themselves.

`ramst.mod` An elementary real business cycle (RBC) model, simulated in a deterministic setup.

`example1.mod`, `example2.mod` Two examples of a small RBC model in a stochastic setup, presented in Collard (2001) (see the file `guide.pdf` which comes with Dynare).

`fs2000.mod` A cash in advance model, estimated by Schorfheide (2000).

`bkk.mod` Multi-country RBC model with time to build, presented in Backus, Kehoe and Kydland (1992).

Chapter 6

Bibliography

[Backus, Kehoe and Kydland (1992)] David K. Backus, Patrick J. Kehoe, and Finn E. Kydland, "International Real Business Cycles", 1992.

Journal of Political Economy, 100, 745-775.

[Boucekkine (1995)] Raouf Boucekkine, "An alternative methodology for solving nonlinear forward-looking models", 1995.

Journal of Economic Dynamics and Control, 19, 711-734.

[Collard (2001)] Fabrice Collard, "Stochastic simulations with Dynare: A practical guide", 2001.

[Collard and Juillard (2001a)] Fabrice Collard and Michel Juillard, "Accuracy of stochastic perturbation methods: The case of asset pricing models", 2001.

Journal of Economic Dynamics and Control, 25, 979-999.

[Collard and Juillard (2001b)] Fabrice Collard and Michel Juillard, "A Higher-Order Taylor Expansion Approach to Simulation of Stochastic Forward-Looking Models with an Application to a Non-Linear Phillips Curve", 2001.

Computational Economics, 17, 125-139.

[Durbin and Koopman (2001)] J. Durbin and S.J. Koopman, Time Series Analysis by State Space Methods, 2001.

[Fair and Taylor (1983)] Ray Fair and John Taylor, "Solution and Maximum Likelihood Estimation of Dynamic Nonlinear Rational Expectation Models", 1983.

Econometrica, 51, 1169-1185.

[Fernandez-Villaverde and Rubio-Ramirez (2004)] Jesus Fernandez-Villaverde and Juan Rubio-Ramirez, "Comparing Dynamic Equilibrium Economies to Data: A Bayesian Approach", 2004.

Journal of Econometrics, 123, 153-187.

[Ireland (2004)] Peter Ireland, "A Method for Taking Models to the Data", 2004.

Journal of Economic Dynamics and Control, 28, 1205-26.

[Judd (1996)] Kenneth Judd, "Approximation, Perturbation, and Projection Methods in Economic Analysis", 1996.

Hans Amman, David Kendrick, and John Rust, Handbook of Computational Economics, 1996, 511-585.

[Juillard (1996)] Michel Juillard, "Dynare: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm", 1996.

Couverture Orange, 9602.

[Koopman and Durbin (2003)] S.J. Koopman and J. Durbin, "Filtering and Smoothing of State Vector for Diffuse State Space Models", 2003.

Journal of Time Series Analysis, 24, 85-98.

[Laffargue (1990)] Jean-Pierre Laffargue, "Résolution d'un modèle macroéconomique avec anticipations rationnelles", 1990.

Annales d'Économie et Statistique, 17, 97-119.

[Lubik and Schorfheide (2007)] Thomas Lubik and Frank Schorfheide, "Do Central Banks Respond to Exchange Rate Movements? A Structural Investigation", 2007.

Journal of Monetary Economics, 54, 1069-1087.

[Mancini-Griffoli (2007)] Tommaso Mancini-Griffoli, *Dynare User Guide*, 2007, An introduction to the solution and estimation of DSGE models.

[Rabanal and Rubio-Ramirez (2003)] Pau Rabanal and Juan Rubio-Ramirez, "Comparing New Keynesian Models of the Business Cycle: A Bayesian Approach", 2003.

Working Paper Series, 2003-30.

[Schmitt-Grohe and Uribe (2002)] Stephanie Schmitt-Grohe and Martin Uribe, "Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function", 2002.

NBER Technical Working Papers, 0282.

[Schorfheide (2000)] Frank Schorfheide, "Loss Function-based evaluation of DSGE models", 2000.

Journal of Applied Econometrics, 15, 645-670.

[Smets and Wouters (2003)] Frank Smets and Rafael Wouters, "An Estimated Dynamic Stochastic General Equilibrium Model of the Euro Area", 2003.

Journal of the European Economic Association, 1, 1123-1175.