

DYNARE MANUAL

Version 3.0

Michel Juillard

DYNARE MANUAL: Version 3.0

Michel Juillard

Copyright © 1996, 2005 Michel Juillard

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions.

Table of Contents

Preface	iv
1. Changes	v
1. Introduction	1
1. Software requirements	2
2. Installation	3
2.1. Installing the Matlab version	3
2.2. Installing the Scilab version	3
2.3. Installing the Gauss version	4
2. Commands	5
1. Executing Dynare	6
2. General declarations	8
3. Model declaration	13
4. Initial and terminal conditions	15
5. Shocks on exogenous variables	19
6. Solving and simulating	23
7. Estimation	31
8. Displaying and saving results	40
3. Examples	44
Bibliography	45
Index	46

Preface

Dynare is a pre-processor and a collection of Matlab, Scilab or Gauss 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, and Juillard, 1996, Collard and Juillard 2001a and 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 used in Dynare can be found in Juillard (1996).

In a stochastic context, Dynare computes one or several simulations corresponding to a random draw of the shocks. Starting with version 2.3 (not available for Gauss), Dynare uses a second order Taylor approximation of the expectation functions (see Judd, 1996, Collard and Juillard, 2001a, 2001b, and Schmitt-Grohe and Uribe, 2002).

Starting with version 3.0, it is 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 (2002), Schorfheide (2000) or Smets and Wouters (2002).

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

1. Changes

July 20, 2005

- Expanded description of `unit_root_vars` statement
- changed the default for nonlinear solver in `steady`
- added a mention of the possibility to write explicitly a steady state function in `steady`, `stoch_simul`, `estimation` and ???
- added a brief *Output* section in `estimation`
- corrected misleading description of option **`prefilter`** in `estimation`
- added variance decomposition among the statistics computed with option **`moments_varendo`** in `estimation`
- `tex` option in `estimation` isn't yet implemented

May 3, 2005

- added option **`noprint`** in `stoch_simul`
- modified option **`irf`** in `stoch_simul`
- modified option **`simul_seed`** in `stoch_simul`

March 6, 2005

- corrected typos in equations for 1st and 2nd order approximation formulas in `stoch_simul`.
- temporarily removed description of output variables in `estimation` as old content was outdated and the new one isn't ready yet.
- added cross-references

Chapter 1. Introduction

In order to give instructions to Dynare, the user has to write a *model file* whose file name must terminate by ".mod". This file contains the description of the model and the computing tasks required by the user.

In practice, the handling of your 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 Gauss, Matlab or Scilab instructions are generated; in the second step, the program actually runs the computations. Both steps are triggered by a single keyword: **Dynare**.

1. Software requirements

This version of Dynare works only under Windows 98/NT/2000/XP. For a Unix version, please, write me.

The Matlab version has been written with Matlab 6.5.1.

The Scilab version has been tested with Scilab 3.0.

The Gauss version of Dynare has been written with Gauss version 3.2. It most likely doesn't work with previous versions.

2. Installation

In case of update from a previous version, it is a good idea to copy the old version in a backup directory so as to be able to revert to it in case of problems. None of the previous files are usefull anymore, so you are strongly encouraged to remove them from directory `c:\dynare`. The Matlab version of Dynare lets you now easily have different versions of Dynare on your computer.

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.

2.1. Installing the Matlab version

Starting with version 3.0, by default, Dynare is installed in a directory whose name contains the version number. For example

```
Dynare_v3.0
```

This directory contains several sub-directories, among which `matlab`, `doc` and `examples`.

After unpacking the archive, start the Matlab program and use the menu **File/Set path** to add the path to Dynare `matlab` subdirectory. For example

```
c:\dynare_v3.0\matlab
```

2.2. Installing the Scilab version

Unpack the zip file in the directory `c:\` (If you want to use another directory, see below). The Scilab version is automatically installed in `c:\dynare\scilab`.

Then, find the `scilab.star` file, in the top directory of your Scilab distribution. Edit this file and add the following line after similar statements:

```
load('c:/dynare/scilab/lib');
```

If you installed Dynare for Scilab in a directory different from `c:\dynare\scilab`, change the above instructions accordingly and edit the following line in `Dynare.sci`

```
command = 'c:\dynare\scilab\dynare_s '+fname;
```


Then, restart Scilab and run the command **uplib()**.

2.3. Installing the Gauss version

Unpack the zip file in the directory `c:\` (If you want to use another directory, see below). The Gauss version is automatically installed in `c:\dynare\gauss`.

If you had any previous version of Dynare, use the Gauss editor or any text editor to remove all references to it from the library file `user.lcg`.

After unpacking the archive, start the Gauss program and type the following:

```
library pgraph
lib user c:\dynare\gauss\dynare.src
lib user c:\dynare\gauss\dynare1.src
lib user c:\dynare\gauss\dynare2.src
lib user c:\dynare\gauss\dynare3.src
```

If you installed Dynare for Gauss in a directory different from `c:\dynare\gauss`, change the above instructions accordingly and edit the following line in `Dynare.src`

```
declare string PARSER = "c:\\dynare\\gauss\\dynare_g ";
```

Chapter 2. Commands

Dynare commands are either single instructions or a block of instructions. Each single instructions or block elements are terminated by `;`. Block 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 []
- repeated arguments are indicated by ellipses ...
- **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 in the underlying language (Matlab, Scilab or Gauss)
- **VARIABLE_NAME** indicates a variable name starting with an alphabetical charcater and can't contain ()+-*/^=!:;@#. or accentuated characters
- **PARAMETER_NAME** indicates a parameter name starting with an alphabetical charcater and can't contain ()+-*/^=!:;@#. or accentuated characters
- **FILENAME** indicates a file name valid under your operating system (Windows, Linux or Unix)

1. Executing Dynare

Dynare

Dynare -- executes Dynare

Dynare

Dynare *FILENAME* [.mod]

Description

Dynare executes instruction included in `filename.mod`. `filename.mod` is the name of the model file containing the model and the processing instructions.

Details

In Matlab, **Dynare** creates three intermediary files:

- `filename.m` with the instructions for the simulations
- `filename_ff.m` with the dynamic model equations
- `filename_fff.m` with the long run static model equations

In Scilab, **Dynare** creates three intermediary files:

- `filename.sci` with the instructions for the simulations
- `filename_ff.sci` with the dynamic model equations
- `filename_fff.sci` with the long run static model equations

In Gauss, **Dynare** creates an intermediary file `filename.gau` with the instructions for the simulations. The Gauss version still accepts the former `.mdl` extension, but it is now deprecated.

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

Examples

Dynare ramst

or

Dynare ramst.mod

2. General declarations

General declarations of variables and parameters are made with the following commands:

- periods (deprecated)
- var
- varexo
- parameters

periods

periods -- specifies the number of simulation periods

```
periods  
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.

Set 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;
```

var

var -- declares endogenous variables

```
var  
var VARIABLE_NAME [ , ] [ VARIABLE_NAME ... ] ;
```

Description

This required command declares the endogenous variables in the model. The variable names must start with a letter and can't contain the following characters : ()+*/^=!,:@#. or accentuated characters.

In Gauss, setting `_longname = 1` allows the use of more than 8 characters in the variable names and makes a distinction between lower and upper case letters.

Example

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

varexo

varexo -- declares exogenous variables

varexo

varexo *VARIABLE_NAME* [,] [*VARIABLE_NAME* ...] ;

Description

This optional command declares the exogenous variables in the model. See command `var` for the syntax of *VARIABLE_NAME*.

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

Example

```
varexo m gov;
```


parameters

parameters -- declares parameters

parameters

```
parameters PARAMETER_NAME [ , ] [ PARAMETER_NAME ... ] ;
```

Description

This optional command declares parameters used in the model, in variable initialization or in shock declarations. The parameters must then be assigned values using standard syntax of underlying matrix programming language. Be carefull not to use names reserved by Dynare or the underlying language (Matlab, Scilab or Gauss).

Example

```
parameters alpha, bet;
```

```
alpha = 0.3;  
bet = sqrt(2);
```

3. Model declaration

The model is declared inside a model block.

model

`model` -- declares the model equations

```
model
model [ (linear) ] ;
(1) EQUATION;
[ (1) EQUATION; ...]
end;
(1) EXPRESSION [= EXPRESSION] ;
```

Description

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

There must be as many equations as there are endogenous variables in the model, except when used to compute the unconstrained optimal policy with **olr**. The lead and lag of the variables are written in parenthesis immediately after the variable name. Leads or lags of more than one period are allowed. All the functions available in Matlab, Scilab or Gauss, respectively, are recognized. 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.

The option **linear** declares the model as being linear. It avoids to have to declare initial values for computing the steady state and it sets automatically **order=1** in **stoch_simul**.

Example 1

```
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

```
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 3

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

4. Initial and terminal conditions

In many contexts, it is necessary to compute the steady state of a non-linear model `initval` specifies then numerical initial values for the non-linear solver.

Used in perfect foresight mode, the types of forward-looking 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 equilibrium. To do that, one needs `initval` and `shocks`(see next section).

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 in different periods.

- `initval`
- `endval`
- `histval`

initval

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

```
initval
initval;
VARIABLE_NAME = EXPRESSION;
[ VARIABLE_NAME = EXPRESSION; ...]
end;
```

Description

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

The **initval**; ... **end**; block serves two purposes. It set the initial and, possibly, terminal conditions for the simulation and provides numerical initialization for various computation tasks (steady, simul, stoch_simul).

Theoretically, initial conditions are only necessary for lagged variables. However, as **initval** provides also numerical initialization, it is necessary to provide values for all variables in the model, except if the model is declared as linear.

For stochastic models, it isn't necessary to declare 0 as initial values for exogenous stochastic variables as it is the only possible value.

When the **initval** block is followed by the command `steady`, it is not necessary to provide exact initialization values for the endogenous variables. `steady` will use the values provided in the **initval** block as initial guess in the non-linear equation solver and computes exact values for the endogenous variables at the steady state. The steady state is defined by keeping constant the value of the exogenous variables.

Example

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

steady;
```

endval

endval -- specifies terminal values for deterministic simulations

```
endval
endval;
VARIABLE_NAME = EXPRESSION;
[ VARIABLE_NAME = EXPRESSION; ...]
end;
```

Description

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

The optional **endval**; ... **end**; block serves two purposes. It set the terminal conditions for the simulation with the LBJ algorithm, when those differ from the initial conditions. When it is the case, the **endval** block also provides the numerical initialization for various computation tasks (steady, simul), starting in period 1.

Theoretically, terminal conditions are required in the LBJ algorithm only for forward variables. However, as **endval** provides also numerical initialization, it is necessary to provide values for all variables in the model.

When the **endval** block is followed by the command steady, it is not necessary to provide exact values for the endogenous variables. steady will use the values provided in the **endval** block as initial guess in the non-linear equation solver and computes exact values for the endogenous variables at the steady state. The steady state is defined by keeping constant the value of the exogenous variables.

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.

histval

histval -- specifies historical values before the start of a simulation

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

5. 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 is 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 directly entered with `Sigma_e`. Note that, starting with version 2.5.2, the direct specification of the *internal* matrix `Sigma_e_`, prone to errors, is discouraged.

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`
- `Sigma_e`

shocks

shocks -- specifies shocks on deterministic or stochastic exogenous variables

```
shocks
shocks;
(1) DETERMINISTIC SHOCK STATEMENT | (3) STOCHASTIC SHOCK STATEMENT
[ (1) DETERMINISTIC SHOCK STATEMENT | (3) STOCHASTIC SHOCK STATEMENT ...]
end;
(1) var VARIABLE_NAME; periods (2) PERIOD STATEMENT; values EXPRESSION;
(2) INTEGER [: INTEGER] [INTEGER [: INTEGER] ...] ;
(3) (4) VARIANCE STATEMENT | (5) COVARIANCE STATEMENT | (6) STANDARD ERROR STATEMENT
(4) var VARIABLE_NAME = EXPRESSION;
(5) var VARIABLE_NAME , VARIABLE_NAME = EXPRESSION;
(6) var VARIABLE_NAME; stderr 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 (available only in the Matlab or Scilab versions), the **shocks** block specifies the non zero elements of the covariance matrix of the shocks.

Example

```
shocks;  
var e = 0.000081;  
var e,u = phi*0.009*0.009;  
var u = 0.000081;  
var v; stderr 0.009;  
end;
```

See also

Sigma_e

Sigma_e

Sigma_e -- specifies directly the covariance matrix of the stochastic shocks

Sigma_e

```
Sigma_e = [(1) MATRIX ELEMENT [,](1) MATRIX ELEMENT...] [ ; (1) MATRIX ELEMENT...];  
(1) INTEGER | DOUBLE | (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 *EXPRESSION* using predefined parameters is allowed but must be placed between parentheses. THE ORDER OF THE COVARIANCES IN THE MATRIX IS THE SAME AS THE ONE USED IN THE VAREXO DECLARATION.

Note

In previous versions, it was possible to directly set Dynare's internal covariance matrix Sigma_e_. This is still possible for compatibility with older .mod files, but **STRONGLY DISCOURAGED** as too prone to error. When setting Sigma_e_ directly, the order of the exogenous shocks is the ALPHABETICAL order of their names.

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.

6. 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`
- `check`
- `simul`
- `stoch_simul`

steady

`steady` -- computes the steady state of a model

`steady`

```
steady [ (solve_algo = 0 | 1 | 2 ) ] ;
```

Options

- **solve_algo = 0**: uses Matlab Optimization Toolbox FSOLVE
- **solve_algo = 1**: uses Dynare's own nonlinear equation solver
- **solve_algo = 2**: splits the model into recursive blocks and solves each block in turn. (Thanks to Manfred Gilli for showing me Matlab's function DMPERM) (this is the default since Dynare version 3.046).

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 `ys_`. Endogenous variables are ordered alphabetically as in `lgy_`.

Examples

See `initval` and `endval`.

check

`check --` computes the eigenvalues of the (linearized) model

```
check  
check ;
```

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 `eigenvalues_`.

simul

simul -- simulates a deterministic model

```
simul  
simul [ (periods=INTEGER) ] ;
```

Description

Triggers the computation of a deterministic simulation of the model for the number of periods set in the option **periods=**. **simul** uses a Newton method to solve simultaneously all the equations for every period (see Juillard, 1996).

Output variables

the simulated variables are available in global matrix `y_`. The variables are arranged row by row, in alphabetical order.

stoch_simul

stoch_simul -- computes the solution and simulates the model

stoch_simul

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

Options

- **ar** = *INTEGER*: Order of autocorrelation coefficients to compute and to print (default = 5) n
- **dr_algo** = 0 | 1: specifies the algorithm used for computing the quadratic approximation of the decision rules:
 - 0: uses a *pure* perturbation approach as in Schmitt-Grohe and Uribe (2002) (default)
 - 1: moves the point around which the Taylor expansion is computed toward the means of the distribution as in Collard and Juillard (2001)
- **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 $\lambda = \text{INTEGER}$ before computing moments (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 (default = 40). Setting IRF=0, suppresses the plotting of IRF's.
- **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**: doesn't print the correlation matrix (printing them is the default)
- **nofunctions**: doesn't print the coefficients of the approximated solution (printing them is the default)
- **nomoments**: doesn't print moments of the endogenous variables (printing them is the default)
- **noprint**: cancel any printing. Useful for loops.
- **order** = 1 | 2 : order of Taylor approximation (default = 2)
- **periods** = *INTEGER*: specifies the number of periods to use in simulations. At order=1, no simulation is necessary to compute theoretical moments and IRFs. A number of periods larger than one triggers automatically option **simul** (default = 0).
- **qz_criterium** = *INTEGER* | *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**: computes a stochastic simulation of the model for the number of periods specified in the **periods** statement. Uses initial values, possibly recomputed by steady, as initial values for the simulation. The simulated endogenous variables are made available to the user in a vector for each variable and in the global matrix *y_*. The variables are ordered alphabetically in the *y_* matrix (default: no simulation)
- **simul_seed** = *INTEGER*|*DOUBLE*| (*EXPRESSION*): 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). Note that if you use an *EXPRESSION* rather than an *INTEGER* or a *DOUBLE*, the *EXPRESSION* must be in parenthesis.
- all **steady** options (see steady)

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

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 (except with option **dr_algo=1**). If you know how to compute the steady state for your model, you can provide a Matlab 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.

Variance decomposition, correlation, autocorrelation are only displayed for variables with positive variance. Impulse response functions are only plotted for variables with response larger than $1e-10$.

Currently, the IRF's 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 moments of simulated variables.

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

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 observed at the beginning of the period.

First order approximation

$$y_t = y_s + A y_{h,t-1} + B u_t$$

where y_s is the steady state value of y and $y_h = y_t - y_s$.

Second order approximation

$$y_t = y_s + 0.5\Delta^2 + A y_{h,t-1} + B u_t + 0.5C(y_{h,t-1} \otimes y_{h,t-1}) + 0.5D(u_t \otimes u_t) + E(y_{h,t-1} \otimes u_t)$$

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

Output variables

stoch_simul sets several 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 coefficients of the decision rules are stored in global structure `dr_`. Here is the correspondance with the symbols used in the above description of the decision rules:

Decision rule coefficients

- y_s : `dr_.ys`. The vector rows correspond to variables in alphabetical order of the variable names.
- Δ^2 : `dr_.ghs2`. The vector rows correspond to re-ordered variables (see below).
- A : `dr_.ghx`. The matrix rows correspond to re-ordered variables. The matrix columns correspond to state variables (see below).
- B : `dr_.ghu`. The matrix rows correspond to re-ordered variables (see below). The matrix columns correspond to exogenous variables in alphabetical order.

- `C: dr_.ghxx`. The matrix rows correspond to re-ordered variables. The matrix columns correspond to the Kronecker product of the vector of state variables (see below).
- `D: dr_.ghuu`. The matrix rows correspond to re-ordered variables (see below). The matrix columns correspond to the Kronecker product of exogenous variables in alphabetical order.
- `E: dr_.ghxu`. The matrix rows correspond to re-ordered variables. The matrix columns correspond to the Kronecker product of the vector of state variables (see below) by the vector of exogenous variables in alphabetical order.

When reordered, the variables are stored in the following order: static variables, purely predetermined variables (variables that appear only at the current and lagged periods in the model), variables that are both predetermined and forward-looking (variables that appear at the current, future and lagged periods in the model), purely forward-looking variables (variables that appear only at the current and future periods in the model). In each category, the variables are arranged alphabetically.

The state variables of the model are purely predetermined variables and variables that are both predetermined and forward-looking. They are ordered in that order. When there are lags on more than one period, the state variables are ordered first according to their lag: first variables from the previous period, then variables from two periods before and so on. Note also that when a variable appears in the model at a lag larger than one period, it is automatically included at all inferior lags.

- The mean of the endogenous variables is available in the vector `oo_.mean`. The variables are arranged in alphabetical order.
- The matrix of variance-covariance of the endogenous variables in the matrix `oo_.var`. The variables are arranged in alphabetical 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` (default `ar=5`) specifies the number of autocorrelation matrices available.
- Simulated variables, when they have been computed, are available in Matlab vectors with the same name as the endogenous variables.
- Impulse responses, when they have been computed, are available in Matlab vectors with the following naming convention `VARIABLE_NAME_shock name`.

`gnp_ea` contains the effect on `gnp` of a one standard deviation shock on `ea`.

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`.

7. 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
- estimated_params_init
- estimation
- unit_root_vars

varobs

varobs -- lists the observed variables

varobs

varobs *VARIABLE_NAME* ... [*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).

Example

```
varobs C y rr;
```

observation_trends

observation_trends -- specifies linear trends for observed variables

```
observation_trends
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;
```

estimated_params

`estimated_params` -- specifies the estimated parameters and their prior

`estimated_params`

Syntax I (maximum likelihood estimation)

```
estimated_params;  
{ stderr VARIABLE_NAME | PARAMETER_NAME } , INITIAL_VALUE [ , LOWER_BOUND ] [ ,  
UPPER_BOUND ] ;  
...  
end;
```

Syntax II (Bayesian estimation)

```
estimated_params;  
{ stderr VARIABLE_NAME | PARAMETER_NAME } , PRIOR_SHAPE , PRIOR_MEAN , PRIOR_STANDARD_ERROR  
[ , PRIOR_3RD_PARAMETER ] [ , PRIOR_4TH_PARAMETER ] [ , SCALE_PARAMETER ] ;  
...  
end;
```

Description

The `estimated_params;...end;` 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** is a keyword indicating 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
- *PARAMETER_NAME* is 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* is prior density among **beta_pdf**, **gamma_pdf**, **normal_pdf**, **inv_gamma_pdf**, **inv_gamma1_pdf**, **inv_gamma2_pdf**, **uniform_pdf**
- *PRIOR_MEAN* is the mean of the prior distribution
- *PRIOR_STANDARD_ERROR* is the standard error of the prior distribution
- *PRIOR_3RD_PARAMETER* is a third parameter of the prior used for generalized beta distribution, generalized gamma and for the uniform distribution (default 0)
- *PRIOR_4TH_PARAMETER* is a fourth parameter of the prior used for generalized beta distribution, generalized gamma and for the uniform distribution (default 1)
- *SCALE_PARAMETER* is the scale parameter to be used for the jump distribution of the Metropolis-Hasting algorithm

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: if you want to specify *jscale*, you must specify *prior_p3* and *prior_p4*. Use default values, if these parameters don't apply.

estimated_params_init

estimated_params_init -- specifies initial values for optimization

```
estimated_params_init
estimated_params_init;
{ stderr VARIABLE_NAME | PARAMETER_NAME }, INITIAL_VALUE;
...
end;
```

Description

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

Estimated parameter initial value specification

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

- **stderr** is a keyword indicating 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
- *PARAMETER_NAME* is the name of a model parameter to be estimated
- *INITIAL_VALUE* specifies a starting value for maximum likelihood estimation

estimated_params_bounds

`estimated_params_bounds` -- specifies lower and upper bounds for the estimated parameters

```
estimated_params_bounds
estimated_params_bounds;
{ stderr VARIABLE_NAME | PARAMETER_NAME } , LOWER_BOUND , UPPER_BOUND ;
...
end;
```

Description

The **estimated_params;...end;** block lists all parameter to be estimated and specifies bounds and priors when required.

Estimated parameter specification

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

- **stderr** is a keyword indicating 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
- *PARAMETER_NAME* is the name of a model parameter to be estimated
- *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

estimation

estimation -- computes estimation.

estimation

estimation [(OPTIONS)] ;

OPTIONS

- **datafile** = *FILENAME*: the datafile (a .m file or a .mat file)
- **nobs** = *INTEGER*: the number of observations to be used (default: all observations in the file)
- **first_obs** = *INTEGER*: the number of the first observation to be used (default = 1)
- **prefilter** = 1: the estimation procedure demeanes the data (default=0, no prefiltering)
- **presample** = *INTEGER*: the number of observations to be skipped before evaluating the likelihood (default = 1)
- **loglinear**: computes a log--linear approximation of the model instead of a linear (default) approximation. The data must correspond to the definition of the variables used in the modelx.
- **nograph**: no graphs should be plotted
- **lik_init**: *INTEGER*: type of initialization of Kalman filter.
 - 1 (default): 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.
- **conf_sig** = {*INTEGER* | *DOUBLE*}: the level for the confidence intervals reported in the results (default = 0.90)
- **mh_replic** = *INTEGER*: number of replication for Metropolis Hasting algorithm. For the time being, mh_replic should be larger than 1200 (default = 20000.)
- **mh_nblocks** = *INTEGER*: number of parallel chains for Metropolis Hasting 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 MH algorithm. The default value is rarely satisfactory. This option must be tune to obtain, ideally, an accpetation 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).
- **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 NAME_mode*.
- **mode_compute**=*INTEGER*: specifies the optimizer for the mode computation.
 - 0: the mode isn't computed. mode_file must be specified
 - 1: uses Matlab **fmincon**.
 - 2: uses Lester Ingber's Adaptive Simulated Annealing.
 - 3: uses Matlab **fminunc**.
 - 4 (default): uses Chris Sim's **csmminwel**.
- **mode_check**: when **mode_check** is set, Dynare plots 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**: when **load_mh_file** is declared, Dynare adds to previous Metropolis-Hastings simulations instead of starting from scratch.
- **optim**=(*fmincon options*): can be used to set options for fmincon, the optimizing function of Matlab Optimizaition toolbox. Use Matlab 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 (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 as well as variance decomposition
- **filtered_vars** triggers the computation of the posterior distribution of filtered endogenous variables and shocks
- **smoother** triggers the computation of the posterior distribution of smoothened 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)
- All options for `stoch_simul`

Note

If no **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 variables

Many estimation results are made available in Matlab structure `oo_`. It is a structure of structures and the field names are self-explanatory (to be expanded).

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.

unit_root_vars

`unit_root_vars` -- declares unit-root variables for estimation

```
unit_root_vars  
unit_root_vars VARIABLE_NAME [ VARIABLE_NAME ...] ;
```

Description

unit_root_vars is 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, 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).

8. Displaying and saving results

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

- `rplot`
- `dynatype`
- `dynasave`

rplot

rplot -- plot variables

rplot

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

Description

Plots one or several variables

dynatype

`dynatype -- print simulated variables`

`dynatype`

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

Description

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

dynasave

dynasave -- save simulated variables in a binary file

dynasave

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

Chapter 3. Examples

Fabrice Collard (GREMAQ, University of Toulouse) has written a guide to stochastic simulations with Dynare entitled "Dynare in Practice" which is in `guide.pdf`.

Bibliography

- Raouf Boucekkine, 1995, "An alternative methodology for solving nonlinear forward-looking models", *Journal of Economic Dynamics and Control*, 19, 711-734,
- Fabrice Collard and Michel Juillard, 2001, "Accuracy of stochastic perturbation methods: The case of asset pricing models", *Journal of Economic Dynamics and Control*, 25, 979-999,
- Fabrice Collard and Michel Juillard, 2001, "A Higher-Order Taylor Expansion Approach to Simulation of Stochastic Forward-Looking Models with an Application to a Non-Linear Phillips Curve", *Computational Economics*, 17, 125-139,
- J. Durbin and S.J. Koopman, 2001, *Time Series Analysis by State Space Methods*, Oxford University Press,
- Ray Fair and John Taylor, 1983, "Solution and Maximum Likelihood Estimation of Dynamic Nonlinear Rational Expectation Models", *Econometrica*, 51, 1169-1185,
- Jesus Fernandez-Villaverde and Juan Rubio-Ramirez, 2004, "Comparing Dynamic Equilibrium Economies to Data: A Bayesian Approach", *Journal of Econometrics*, 123, 153-187,
- Peter Ireland, 2004, "A Method for Taking Models to the Data", *Journal of Economic Dynamics and Control*, 28, 1205-26,
- Kenneth Judd, 1996, "Approximation, Perturbation, and Projection Methods in Economic Analysis", Hans Amman, David Kendrick, and John Rust, *Handbook of Computational Economics*, 1996, North Holland Press, 511-585,
- Michel Juillard, 1996, *Dynare: A program for the resolution and simulation of dynamic models with forward variables through the use of a relaxation algorithm*, CEPREMAP, Couverture Orange, 9602,
- S.J. Koopman and J. Durbin, 2003, "Filtering and Smoothing of State Vector for Diffuse State Space Models", *Journal of Time Series Analysis*, 24, 85-98,
- Jean-Pierre Laffargue, "Résolution d'un modèle macroéconomique avec anticipations rationnelles", 1990, *Annales d'Economie et Statistique*, 17, 97-119,
- Thomas Lubik and Frank Schorfheide, 2003, *Do Central Banks Target Exchange Rates? A Structural Investigation*, University of Pennsylvania,
- Pau Rabanal and Juan Rubio-Ramirez, 2003, *Comparing New Keynesian Models of the Business Cycle: A Bayesian Approach*, Atlanta Fed, *Working Paper*, 2001-22a, rev 2003,
- Frank Schorfheide, 2000, "Loss Function-based evaluation of DSGE models", *Journal of Applied Econometrics*, 15, 645-70,
- Stephanie Schmitt-Grohe and Martin Uribe, 2002, *Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function*, Rutgers University,
- Frank Smets and Rafael Wouters, 2002, *An Estimated Stochastic Dynamic General Equilibrium Model of the Euro Area*, European Central Bank, *ECB Working Paper*, 171,

Index

C

check25

D

Dynare7

dynasave43

dynatype42

E

endval17

estimated_params34

estimated_params_bounds36

estimated_params_init35

estimation37

H

histval18

I

initval16

M

model14

O

observation_trends33

P

parameters12

periods9

R

rplot41

S

shocks20

simul26

stoch_simul27

V

var10

varexo11

varobs32