Dynare v4 - User Guide
Public beta version

Tommaso Mancini Griffoli
tommaso.mancini@stanfordalumni.org

This draft: March 2007

.

Contents

List of Figures

Work in Progress!

This is the second version of the Dynare User Guide which is still work in progress! This means two things. First, please read this with a critical eye and send me comments! Are some areas unclear? Is anything plain wrong? Are some sections too wordy, are there enough examples, are these clear? On the contrary, are there certain parts that just click particularly well? How can others be improved? I'm very interested to get your feedback.

The second thing that a work in progress manuscript comes with is a few internal notes. These are mostly placeholders for future work, notes to myself or others of the Dynare development team, or at times notes to you - our readers - to highlight a feature not yet fully stable. Any such notes are marked with two stars (**).

Thanks very much for your patience and good ideas. Please write either direclty to myself: tommaso.mancini@stanfordalumni.org, or **preferably on the Dynare Documentation Forum** available in the Forum section of the Dynare website.

Contacts and Credits

Dynare was originally developed by Michel Juillard in Paris, France. Currently, the development team of Dynare is composed of

- Stéphane Adjemian (stephane.adjemian"AT"ens.fr)
- Michel Juillard (michel.juillard"AT"ens.fr)
- Ferhat Mihoubi (ferhat.mihoubi"AT"univ-evry.fr)
- Ondra Kamenik (ondra.kamenik"AT"volny.cz)
- Marco Ratto (marco.ratto"AT"jrc.it)
- Sébastien Villemot (sebastien.villemot"AT"ens.fr)

Several parts of Dynare use or have strongly benefitted from publicly available programs by F. Collard, L. Ingber, P. Klein, S. Sakata, F. Schorfheide, C. Sims, P. Soederlind and R. Wouters.

Finally, the development of Dynare could not have come such a long ways withough an active community of users who continually pose questions, report bugs and suggest new features. The help of this community is gratefully acknowledged.

The email addresses above are provided in case you wish to contact any one of the authors of Dynare directly. We nonetheless encourage you to first use the Dynare forums to ask your questions so that other users can benefit from them as well; remember, almost no question is specific enough to interest just one person, and yours is not the exception!

Chapter 1

Introduction

Welcome to Dynare!

1.1 About this Guide - approach and structure

This User Guide **aims to help you master Dynare**'s main functionalities, from getting started to implementing advanced features. To do so, this Guide is structured around examples and offers practical advice. To root this understanding more deeply, though, this Guide also gives some background on Dynare's algorithms, methodologies and underlying theory. Thus, a secondary function of this Guide is to **serve as a basic primer** on DSGE model solving and Bayesian estimation.

This Guide will focus on the most common or useful features of the program, thus emphasizing **depth over breadth**. The idea is to get you to use 90% of the program well and then tell you where else to look if you're interested in fine tuning or advanced customization.

This Guide is written mainly for an **advanced economist** - like a professor, graduate student or central banker - needing a powerful and flexible program to support and facilitate his or her research activities in a variety of fields. The sophisticated computer programmer, on the one hand, or the specialist of computational economics, on the other, may not find this Guide sufficiently detailed.

We recognize that the "advanced economist" may be either a beginning or intermediate user of Dynare. This Guide is written to accommodate both. If you're **new to Dynare**, we recommend starting with chapters 3 and 5, which introduce the program's basic features to solve (including running impulse response functions) and estimate DSGE models, respectively. To do

so, these chapters lead you through a complete hands-on example, which we recommend following from A to Z, in order to "**learn by doing**". Once you have read these two chapters, you will know the crux of Dynare's functionality and (hopefully!) feel comfortable using Dynare for your own work. At that point, though, you will probably find yourself coming back to the User Guide to skim over some of the content in the advanced chapters to iron out details and potential complications you may run into.

If you're instead an **intermediate user** of Dynare, you will most likely find the advanced chapters, 4 and 6, more appropriate. These chapters cover more advanced features of Dynare and more complicated usage scenarios. The presumption is that you would skip around these chapters to focus on the topics most applicable to your needs and curiosity. Examples are therefore more concise and specific to each feature; these chapters read a bit more like a reference manual.

We also recognize that you probably have had repeated if not active exposure to programming and are likely to have a strong economic background. Thus, a black box solution to your needs is inadequate. To hopefully address this issue, the User Guide goes into some depth in covering the **theoretical underpinnings and methodologies that Dynare follows** to solve and estimate DSGE models. These are available in the "behind the scenes of Dynare" chapters 7 and 8. These chapters can also serve as a **basic primer** if you are new to the practice of DSGE model solving and Bayesian estimation.

Finally, besides breaking up content into short chapters, we've introduced two different **markers** throughout the Guide to help streamline your reading.

- **TIP!** introduces advice to help you work more efficiently with Dynare or solve common problems.
- **NOTE!** is used to draw your attention to particularly important information you should keep in mind when using Dynare.

1.2 What is Dynare?

Before we dive into the thick of the "trees", let's have a look at the "forest" from the top . . . just what is Dynare?

Dynare is a powerful and highly customizable engine, with an intuitive front-end interface, to solve, simulate and estimate DSGE models.

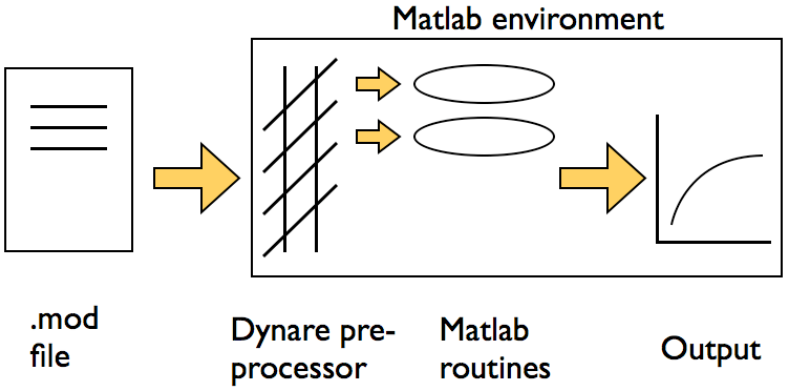

Figure 1.1: The .mod file being read by the Dynare pre-processor, which then calls the relevant Matlab routines to carry out the desired operations and display the results.

In slightly less flowery words, it is a pre-processor and a collection of Matlab routines that has the great advantages of reading DSGE model equations written almost as in an academic paper. This not only facilitates the inputting of a model, but also enables you to easily share your code as it is straightforward to read by anyone.

Figure 1.2 gives you an overview of the way Dynare works. Basically, the model and its related attributes, like a shock structure for instance, is written equation by equation in an editor of your choice. The resulting file will be called the .mod file. That file is then called from Matlab. This initiates the Dynare pre-processor which translates the .mod file into a suitable input for the Matlab routines (more precisely, it creates intermediary Matlab or C files which are then used by Matlab code) used to either solve or estimate the model. Finally, results are presented in Matlab. Some more details on the internal files generated by Dynare is given in section 4.2 in chapter 4.

Each of these steps will become clear as you read through the User Guide, but for now it may be helpful to summarize **what Dynare is able to do**:

- compute the steady state of a model
- compute the solution of deterministic models
- compute the first and second order approximation to solutions of stochastic models
- estimate parameters of DSGE models using either a maximum likelihood or a Bayesian approach
- compute optimal policies in linear-quadratic models

1.3 Additional sources of help

While this User Guide tries to be as complete and thorough as possible, you will certainly want to browse other material for help, as you learn about new features, struggle with adapting examples to your own work, and yearn to ask that one question whose answer seems to exist no-where. At your disposal, you have the following additional sources of help:

- **Reference Manual**: this manual covers all Dynare commands, giving a clear definition and explanation of usage for each. The User Guide will often introduce you to a command in a rather loose manner (mainly through examples); so reading corresponding command descriptions in the Reference Manual is a good idea to cover all relevant details.
- **Official online examples**: the Dynare website includes other examples - usually well documented - of .mod files covering models and methodologies introduced in recent papers.
- **Open online examples**: this page lists .mod files posted by users covering a wide variety of examples.
- **Dynare forums**: this lively online discussion forum allows you to ask your questions openly and read threads from others who might have run into similar difficulties.
- **Frequently Asked Questions** (FAQ): this section of the Dynare site emphasizes a few of the most popular questions in the forums.
- **DSGE.net**: this website, run my members of the Dynare team, is a resource for all scholars working in the field of DSGE modeling. Besides allowing you to stay up to date with the most recent papers and possibly make new contacts, it conveniently lists conferences, workshops and seminars that may be of interest.

1.4 Nomenclature

To end this introduction and avoid confusion in what follows, it is worthwhile to agree on a few **definitions of terms**. Many of these are shared with the Reference Manual.

- **Integer** indicates an integer number.
- **Double** indicates a double precision number. The following syntaxes are valid: 1.1e3, 1.1E3, 1.1E-3, 1.1d3, 1.1D3
- **Expression** indicates a mathematical expression valid in the underlying language (e.g. Matlab).
- **Variable name** indicates a variable name. **NOTE!** These must start with an alphabetical character and can only contain other alphabetical characters and digits, as well as underscores (_). All other characters, including accents, and **spaces**, are forbidden.
- **Parameter name** indicates a parameter name which must follow the same naming conventions as above.
- **Filename** indicates a file name valid in your operating system. Note that Matlab requires that names of files or functions start with alphabetical characters; this concerns your Dynare .mod files.
- **Command** is an instruction to Dynare or other program when specified.
- **Options** or optional arguments for a command are listed in square brackets [ ] unless otherwise noted. If, for instance, the option must be specified in parenthesis in Dynare, it will show up in the Guide as [(option)].
- `Typewritten text` indicates text as it should appear in Dynare code.

1.5 v4, what's new and backward compatibility

The current version of Dynare - for which this guide is written - is version 4. With respect to version 3, this new version introduces several important features, as well as improvements, optimizations of routines and bug fixes. The major new features are the following:

- Analytical derivatives are now used everywhere (for instance, in the Newton algorithm for deterministic models and in the linearizations necessary to solve stochastic models). This increases computational speed significantly. The drawback is that Dynare can now handle only a limited set of functions, although in nearly all economic applications this should not be a constraint.

- Variables and parameters are now kept in the order in which they are declared whenever displayed and when used internally by Dynare. Recall that in version 3, variables and parameters where at times in their order of declaration and at times in alphabetical order. **NOTE!** This may cause some problems of backward compatibility if you wrote programs to run off Dynare v3 output.

- The names of many internal variables and the organization of output variables has changed. These are enumerated in details in the relevant chapters. The names of the files internally generated by Dynare have also changed. (** more on this when explaining internal file structure - TBD)

- The syntax for the external steady state file has changed. This is covered in more details in chapter 3, in section 3.6.3. **NOTE!** You will unfortunately have to slightly amend any old steady state files you may have written.

- Speed. Several large-scale improvements have been implemented to speed up Dynare. This should be most noticeable when solving deterministic models, but also apparent in other functionality.

Chapter 2

Installing Dynare

2.1 Dynare versions

Three versions of Dynare exist: one for **Matlab**, one for **Scilab** and one for **Gauss**. The first benefits from ongoing development and is the most popular. Development of the Scilab version stopped after Dynare version 3.02 and that for Gauss after Dynare version 1.2.

This User Guide will exclusively **focus on the Matlab version of Dynare**. For the installation procedure for the Scilab or Gauss versions of the program, please see the Reference Manual. Note, though, that the Dynare syntax remains mostly unchanged across the Matlab, Scilab or Gauss versions, for those features common to the three versions.

You may also be interested by another version of Dynare, developed in parallel: **Dynare++**. This is a standalone C++ version of Dynare specialized in computing k-order approximations of dynamic stochastic general equilibrium models. See the Dynare++ webpage for more information.

2.2 System requirements

Dynare can run on Windows, as well as Unix-like operating systems, such as any Linux distribution, Solaris and, of course, Mac OS X. If you have questions about the support of a particular platform, feel free to write directly to Michel Juillard (michel.juillard"AT"ens.fr) or visit the **Dynare forums**.

To run Dynare, it is recommended to allocate at least 256MB of RAM to the platform running Dynare, although 512MB is preferred. Depending on the type of computations required, like the very processor intensive Metropolis Hastings algorithm, you may need up to 1GB of RAM to obtain acceptable

computational times.

2.3 Installing Dynare

2.3.1 Installing on Windows

The following assumes you have Matlab version 6.5.1 or later installed on your Windows system.[1] ** The current way to install Dynare version 4 may not yet be on par with the procedure described below. If a discrepancy exists, please follow downloading and installation instructions on the Dynare website.

1. Download the latest stable version of Dynare for Matlab (Windows) from the Dynare website.
2. You will now have on your computer a .zip file which you should unzip. This will create a folder called, by default, Dynare and its version number, for example: Dynare_v4.x (where x stands for any subsequent upgrades).
3. This directory contains several sub-directories, among which (i) matlab, (ii) doc and (iii) examples.
4. Place the Dynare folder (Dynare_v4.x in our example) in the c: directory and note that location. The easiest is probably to put it in the root of c: as in c:/dynare_v4.x.
5. Start Matlab and use the menu File/Set-Path to add the path to the Dynare matlab subdirectory. Following our example, this would correspond to c:/dynare_v4.x/matlab
6. Save these changes in Matlab and you're ready to go.

2.4 Matlab particularities

A question often comes up: what special Matlab toolboxes are necessary to run Dynare? In fact, no additional toolbox is necessary for running most of Dynare, except maybe for optimal simple rules (see chapter 9), but even then remedies exist (see the Dynare forums for discussions on this, or to ask your particular question). But if you do have the 'optimization toolbox' installed, you will have additional options for solving for the steady state (solve_algo option) and for searching for the posterior mode (mode_compute option), both of which are defined later.

[1] As of writing this Guide, Dynare is being developed on Matlab version 7. Nonetheless, great care is taken not to introduce features that would not work with reasonably recent versions of Matlab. However, Dynare requires at least the Matlab feature set of version 6.5.1, released September 22, 2003.

Chapter 3

Solving DSGE models - basics

This chapter covers everything that leads to, and stems from, the solution of DSGE models; a vast terrain. That is to say that the term "solution" in the title of the chapter is used rather broadly. You may be interested in simply finding the solution functions to a set of first order conditions stemming from your model, but you may also want to go a bit further. Typically, you may be interested in how this system behaves in response to shocks, whether temporary or permanent. Likewise, you may want to explore how the system comes back to its steady state or moves to a new one. This chapter covers all these topics. But instead of skipping to the topic closest to your needs, we recommend that you read this chapter chronologically, to learn basic Dynare commands and the process of writing a proper .mod file - this will serve as a base to carry out any of the above computations.

3.1 A fundamental distinction

Before speaking of Dynare, it is important to recognize a distinction in model types. This distinction will appear throughout the chapter; in fact, it is so fundamental, that we considered writing separate chapters altogether. But the amount of common material - Dynare commands and syntax - is notable and writing two chapters would have been overly repetitive. Enough suspense; here is the important question: **is your model stochastic or deterministic?**

The distinction hinges on **whether future shocks are known**. In deterministic models, the occurrence of all future shocks is known exactly at the time of computing the model's solution. In stochastic models, instead, only the distribution of future shocks is known. Let's consider a shock to a model's innovation only in period 1. In a deterministic context, agents will take their decisions knowing that future values of the innovations will be zero in all periods to come. In a stochastic context, agents will take their decisions

knowing that the future value of innovations are random but will have zero mean. This isn't the same thing because of Jensen's inequality. Of course, if you consider only a first order linear approximation of the stochastic model, or a linear model, the two cases become practically the same, due to certainty equivalence. A second order approximation will instead lead to very different results, as the variance of shocks will matter.

The solution method for each of these model types differs significantly. In deterministic models, a highly accurate solution can be found by numerical methods. The solution is nothing more than a series of numbers that match a given set of equations. Intuitively, if an agent has perfect foresight, she can specify today - at the time of making her decision - what each of her precise actions will be in the future. In a stochastic environment, instead, the best the agent can do is specify a decision, policy or feedback rule for the future: what will her optimal actions be contingent on each possible realization of shocks. In this case, we therefore search for a function satisfying the model's first order conditions. To complicate things, this function may be non-linear and thus needs to be approximated. In control theory, solutions to deterministic models are usually called "closed loop" solutions, and those to stochastic models are referred to as "open loop".

Because this distinction will resurface again and again throughout the chapter, but also because it has been a source of significant confusion in the past, the following gives some additional details.

3.1.1 NOTE! Deterministic vs stochastic models

Deterministic models have the following characteristics:

1. As the DSGE (read, "stochastic", i.e. not deterministic!) literature has gained attention in economics, deterministic models have become somewhat rare. Examples include OLG models without aggregate uncertainty.

2. These models are usually introduced to study the impact of a change in regime, as in the introduction of a new tax, for instance.

3. Models assume full information, perfect foresight and no uncertainty around shocks.

4. Shocks can hit the economy today or at any time in the future, in which case they would be expected with perfect foresight. They can also last one or several periods.

5. Most often, though, models introduce a positive shock today and zero shocks thereafter (with certainty).

6. The solution does not require linearization, in fact, it doesn't even really need a steady state. Instead, it involves numerical simulation to find the exact paths of endogenous variables that meet the model's first order conditions and shock structure.

7. This solution method can therefore be useful when the economy is far away from steady state (when linearization offers a poor approximation).

Stochastic models, instead, have the following characteristics:

1. These types of models tend to be more popular in the literature. Examples include most RBC models, or new keynesian monetary models.

2. In these models, shocks hit today (with a surprise), but thereafter their expected value is zero. Expected future shocks, or permanent changes in the exogenous variables cannot be handled due to the use of Taylor approximations around a steady state.

3. Note that when these models are linearized to the first order, agents behave as if future shocks where equal to zero (since their expectation is null), which is the **certainty equivalence property**. This is an often overlooked point in the literature which misleads readers in supposing their models may be deterministic.

3.2 Introducing an example

The goal of this first section is to introduce a simple example. Future sections will aim to code this example into Dynare and analyze its salient features under the influence of shocks - both in a stochastic and a deterministic environment. Note that as a general rule, the examples in the basic chapters, 3 and 5, are kept as bare as possible, with just enough features to help illustrate Dynare commands and functionalities. More complex examples are instead presented in the advanced chapters.

The model introduced here is a basic RBC model with monopolistic competition, used widely in the literature. Its particular notation adopted below is drawn mostly from notes available on Jesus Fernandez-Villaverde's very instructive website; this is a good place to look for additional information on any of the following model set-up and discussion. Note throughout this model description that the use of **expectation** signs is really only relevant in a stochastic setting, as per the earlier discussion. We will none-the-less illustrate both the stochastic and the deterministic settings on the basis of this example. Thus, when thinking of the latter, you'll have to use a bit of imagination (on top of that needed to think you have perfect foresight!) to ignore the expectation signs.

Households maximize utility over consumption, c_t and leisure, $1-l_t$, where l_t is labor input, according to the following utility function

$$\mathbb{E}_t \sum_{t=0}^{\infty} \beta \left[\log c_t + \psi \log(1 - l_t)\right]$$

and subject to the following budget constraint

$$c_t + k_{t+1} = w_t l_t + r_t k_t + (1 - \delta)k_t, \qquad \forall t > 0$$

where k_t is capital stock, w_t real wages, r_t real interest rates or cost of capital and δ the depreciation rate.

The above equation can be seen as an accounting identity, with total expenditures on the left hand side and revenues - including the liquidation value of the capital stock - on the right hand side. Alternatively, with a little more imagination, the equation can also be interpreted as a capital accumulation equation after bringing c_t to the right hand side and noticing that $w_t l_t + r_t k_t$, total payments to factors, equals y_t, or aggregate output, by the zero profit condition. As a consequence, if we define investment as $i_t = y_t - c_t$, we obtain the intuitive result that $i_t = k_{t+1} - (1 - \delta)k_t$, or that investment replenishes the capital stock thereby countering the effects of depreciation. In any given period, the consumer therefore faces a tradeoff between consuming and investing in order to increase the capital stock and consuming more in following periods (as we will see later, production depends on capital).

Maximization of the household problem with respect to consumption, leisure and capital stock, yields the Euler equation in consumption, capturing the intertemporal tradeoff mentioned above, and the labor supply equation linking labor positively to wages and negatively to consumption (the wealthier, the more leisure due to the decreasing marginal utility of consumption). These equation are

$$\frac{1}{c_t} = \beta \mathbb{E}_t \left[\frac{1}{c_{t+1}} \left(1 + r_{t+1} - \delta\right)\right]$$

and

$$\psi \frac{c_t}{1 - l_t} = w_t$$

The firm side of the problem is slightly more involved, due to monopolistic competition, but is presented below in the simplest possible terms, with a little hand-waiving involved, as the derivations are relatively standard.

There are two ways to introduce monopolistic competition. We can either assume that firms sell differentiated varieties of a good to consumers who

aggregate these according to a CES index. Or we can postulate that there is a continuum of intermediate producers with market power who each sell a different variety to a competitive final goods producer whose production function is a CES aggregate of intermediate varieties.

If we follow the second route, the final goods producer chooses his or her optimal demand for each variety, yielding the Dixit-Stiglitz downward sloping demand curve. Intermediate producers, instead, face a two pronged decision: how much labor and capital to employ given these factors' perfectly competitive prices and how to price the variety they produce.

Production of intermediate goods follows a CRS production function defined as

$$y_{it} = k_{it}^{\alpha}(e^{z_t} l_{it})^{1-\alpha}$$

where the i subscript stands for firm i of a continuum of firms between zero and one and where α is the capital elasticity in the production function, with $0 < \alpha < 1$. Also, z_t captures technology which evolves according to

$$z_t = \rho z_{t-1} + e_t$$

where ρ is a parameter capturing the persistence of technological progress and $e_t \sim \mathcal{N}(0, \sigma)$.

The solution to the sourcing problem yields an optimal capital to labor ratio, or relationship between payments to factors:

$$k_{it} r_t = \frac{\alpha}{1-\alpha} w_t l_{it}$$

The solution to the pricing problem, instead, yields the well-known constant markup pricing condition of monopolistic competition:

$$p_{it} = \frac{\epsilon}{\epsilon - 1} mc_t p_t$$

where p_{it} is firm i's specific price, mc_t is real marginal cost and p_t is the aggregate CES price or average price. An additional step simplifies this expression: symmetric firms implies that all firms charge the same price and thus $p_{it} = p_t$; we therefore have: $mc_t = (\epsilon - 1)/\epsilon$

But what are marginal costs equal to? To find the answer, we combine the optimal capital to labor ratio into the production function and take advantage of its CRS property to solve for the amount of labor or capital required to produce one unit of output. The real cost of using this amount of any one factor is given by $w_t l_{it} + r_t k_{it}$ where we substitute out the payments to the

other factor using again the optimal capital to labor ratio. When solving for labor, for instance, we obtain

$$mc_t = \left(\frac{1}{1-\alpha}\right)^{1-\alpha}\left(\frac{1}{\alpha}\right)^{\alpha}\frac{1}{A_t}w_t^{1-\alpha}r_t^{\alpha}$$

which does not depend on i; it is thus the same for all firms.

Interestingly, the above can be worked out, by using the optimal capital to labor ratio, to yield $w_t[(1-\alpha)y_{it}/l_{it}]^{-1}$, or $w_t\frac{\partial l_{it}}{\partial y_{it}}$, which is the definition of marginal cost: the cost in terms of labor input of producing an additional unit of output. This should not be a surprise since the optimal capital to labor ratio follows from the maximization of the production function (minus real costs) with respect to its factors.

Combining this result for marginal cost, as well as its counterpart in terms of capital, with the optimal pricing condition yields the final two important equations of our model

$$w_t = (1-\alpha)\frac{y_{it}}{l_{it}}\frac{(\epsilon-1)}{\epsilon}$$

and

$$r_t = \alpha\frac{y_{it}}{k_{it}}\frac{(\epsilon-1)}{\epsilon}$$

To end, we aggregate the production of each individual firm to find an aggregate production function. On the supply side, we factor out the capital to labor ratio, k_t/l_t, which is the same for all firms and thus does not depend on i. On the other side, we have the Dixit-Stiglitz demand for each variety. By equating the two and integrating both side, and noting that price dispersion is null - or that, as hinted earlier, $p_{it} = p_t$ - we obtain aggregate production

$$y_t = A_t k_t^{\alpha} l_t^{1-\alpha}$$

which can be shown is equal to the aggregate amount of varieties bought by the final good producer (according to a CES aggregation index) and, in turn, equal to the aggregate output of final good, itself equal to household consumption. Note, to close, that because the ratio of output to each factor is the same for each intermediate firm and that firm specific as well as aggregate production is CRS, we can rewrite the above two equations for w_t and r_t without the i subscripts on the right hand side.

This ends the exposition of the example. Now, let's roll up our sleeves and see how we can input the model into Dynare and actually test how the model will respond to shocks.

3.3 Dynare .mod file structure

Input into Dynare involves the .mod file, as mentioned loosely in the introduction of this Guide. The .mod file can be written in any editor, external or internal to Matlab. It will then be read by Matlab by first navigating within Matlab to the directory where the .mod file is stored and then by typing in the Matlab command line `Dynare filename.mod;` (although actually typing the extension .mod is not necessary). But before we get into executing a .mod file, let's start by writing one!

It is convenient to think of the .mod file as containing four distinct blocks, illustrated in figure 3.3:

- **preamble**: lists variables and parameters
- **model**: spells out the model
- **steady state or initial value**: gives indications to find the steady state of a model, or the starting point for simulations or impulse response functions based on the model's solution.
- **shocks**: defines the shocks to the system
- **computation**: instructs Dynare to undertake specific operations (e.g. forecasting, estimating impulse response functions)

Our exposition below will structured according to each of these blocks.

3.4 Filling out the preamble

The preamble generally involves three commands that tell Dynare what are the model's variables, which are endogenous and what are the parameters. The **commands** are:

- `var` starts the list of endogenous variables, to be separated by commas.
- `varexo` starts the list of exogenous variables that will be shocked.
- `parameters` starts the list of parameters and assigns values to each.

In the case of our example, let's differentiate between the stochastic and deterministic cases. First, we lay these out, then we discuss them.

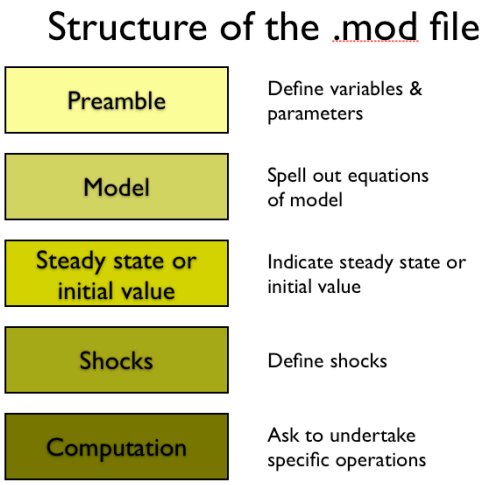

Figure 3.1: The .mod file contains five logically distinct parts.

3.4.1 The deterministic case

The model is inherited exactly as specified in the earlier description, except that we no longer need the e_t variable, as we can make z_t directly exogenous. Thus, the **preamble would look like**:

```
var y c k i l y_l w r;
varexo z;
parameters beta psi delta alpha sigma epsilon;
alpha = 0.33;
beta = 0.99;
delta = 0.023;
psi = 1.75;
sigma = (0.007/(1-alpha));
epsilon = 10;
```

3.4.2 The stochastic case

In this case, we go back to considering the law of motion for technology, consisting of an exogenous shock, e_t. With respect to the above, we therefore

adjust the list of endogenous and exogenous variables, and add the parameter ρ. Here's what the **preamble would look like**:

```
var y c k i l y_l w r z;
varexo e;
parameters beta psi delta alpha rho sigma epsilon;
alpha = 0.33;
beta = 0.99;
delta = 0.023;
psi = 1.75;
rho = 0.95;
sigma = (0.007/(1-alpha));
epsilon = 10;
```

3.4.3 Comments on your first lines of Dynare code

As you can tell, writing a .mod file is really quite straightforward. Two quick comments:

NOTE! Remember that each instruction of the .mod file must be terminated by a semicolon (;), although a single instruction can span two lines if you need extra space (just don't put a semicolon at the end of the first line).

TIP! You can also comment out any line by starting the line with two forward slashes (//), or comment out an entire section by starting the section with /* and ending with */. For example:

```
var y c k i l y_l w r z;
varexo e;
parameters beta psi delta
alpha rho sigma epsilon;
// the above instruction reads over two lines
/*
the following section lists
several parameters which were
calibrated by my co-author.  Ask
her all the difficult questions!
*/
alpha = 0.33;
beta = 0.99;
delta = 0.023;
psi = 1.75;
rho = 0.95;
```

```
sigma = (0.007/(1-alpha));
epsilon = 10;
```

3.5 Specifying the model

3.5.1 Model in Dynare notation

One of the beauties of Dynare is that you can **input your model's equations naturally**, almost as if you were writing them in an academic paper. This greatly facilitates the sharing of your Dynare files, as your colleagues will be able to understand your code in no-time. There are just a few conventions to follow. Let's first have a look at our **model in Dynare notation**, and then go through the various Dynare input conventions. What you can already try to do is glance at the model block below and see if you can recognize the equations from the earlier example. See how easy it is to read Dynare code?

```
model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_l = y/l;
z = rho*z(-1)+e;
end;
```

Just in case you need a hint or two to recognize these equations, here's a brief description: the first equation is the Euler equation in consumption. The second the labor supply function. The third the accounting identity. The fourth is the production function. The fifth and sixth are the marginal cost equal to markup equations. The seventh is the investment equality. The eighth an identity that may be useful and the last the equation of motion of technology.

NOTE! that the above model specification corresponds to the **stochastic case**; indeed, notice that the law of motion for technology is included, as per our discussion of the preamble. The corresponding model for the **deterministic casce** would simply loose the last equation.

3.5.2 General conventions

The above example illustrates the use of a few important commands and conventions to translate a model into a Dynare-readable .mod file.

- The first thing to notice, is that the model block of the .mod file begins with the command `model` and ends with the command `end`.
- Second, in between, there need to be as many equations as you declared endogenous variables (this is actually one of the first things that Dynare checks; it will immediately let you know if there are any problems).
- Third, as in the preamble and everywhere along the .mod file, each line of instruction ends with a semicolon (except when a line is too long and you want to break it across two lines. This is unlike Matlab where if you break a line you need to add ...).
- Fourth, equations are entered one after the other; no matrix representation is necessary. Note that variable and parameter names used in the model block must be the same as those declared in the preamble; **TIP!** remember that variable and parameter names are case sensitive.

3.5.3 Notational conventions

- Variables entering the system with a time t subscript are written plainly. For example, x_t would be written x.
- Variables entering the system with a time $t-n$ subscript are written with $(-n)$ following them. For example, x_{t-2} would be written $x(-2)$ (incidentally, this would count as two backward looking variables).
- In the same way, variables entering the system with a time $t+n$ subscript are written with $(+n)$ following them. For example, x_{t+2} would be written $x(+2)$. Writing $x(2)$ is also allowed, but this notation makes it slightly harder to count by hand the number of forward looking variables (a useful measure to check); more on this below ...

3.5.4 Timing conventions

- In Dynare, the timing of each variable reflects when that variable is decided. For instance, our capital stock is not decided today, but yesterday (recall that it is a function of yesterday's investment and capital stock); it is what we call in the jargon a **predetermined** variable. Thus, even-though in the example presented above we wrote $k_{t+1} = i_t + (1-\delta)k_t$, as in many papers, we would translate this equation into Dynare as `k=i+(1-delta)*k(-1)`.

- As another example, consider that in some wage negociation models, wages used during a period are set the period before. Thus, in the equation for wages, you can write wage in period t (when they are set), but in the labor demand equation, wages should appear with a one period lag.

- A slightly more roundabout way to explain the same thing is that for stock variables, you must use a "stock at the end of the period" concept. It is investment during period t that sets stock at the end of period t. Be careful, a lot of papers use the "stock at the beginning of the period" convention, as we did (on purpose to highlight this distinction!) in the setup of the example model above.

3.5.5 Conventions specifying non-predetermined variables

- A (+1) next to a variable tells Dynare to count the occurrence of that variable as a jumper or forward-looking or non-predetermined variable.

- **Blanchard-Kahn** conditions are met only if the number of non-predetermined variables equals the number of eigenvalues greater than one. If this condition is not met, Dynare will put up a warning.

- Note that a variable may occur both as predetermined and non-predetermined. For instance, consumption could appear with a lead in the Euler equation, but also with a lag in a habit formation equation, if you had one. In this case, the second order difference equation would have two eigenvalues, one needing to be greater and the other smaller than one for stability.

3.5.6 Linear and log-linearized models

There are two other variants of the system's equations which Dynare accommodates. First, the **linear model** and second, the **model in exp-logs**. In the first case, all that is necessary is to write the term `(linear)` next to the command `model`. Our example, with just the equation for y_l for illustration, would look like:

```
model (linear);
yy_l=yy - ll;
end;
```

when estimating models with unit roots, as we will see in chapter 5. If so, simply rewrite your equations by taking the exponential and logarithm of each variable. The Dynare input convention makes this very easy to do. Our example would need to be re-written as follows (just shown for the first two equations)

```
model;
(1/exp(cc)) = beta*(1/exp(cc(+1)))*(1+exp(rr(+1))-delta);
psi*exp(cc)/(1-exp(ll)) = exp(ww);
end;
```

where, this time, repeating a letter for a variable means log of that variable, so that the level of a variable is given by $exp(repeatedvariable)$.

3.6 Specifying steady states and/or initial values

Material in this section has created much confusion in the past. But with some attention to the explanations below, you should get through unscathed. Let's start by emphasizing the uses of this section of the .mod file. First, recall that stochastic models need to be linearized. Thus, they need to have a steady state. One of the functions of this section is indeed to provide these steady state values, or approximations of values. Second, irrespective of whether you're working with a stochastic or deterministic model, you may be interested to start your simulations or impulse response functions from either a steady state, or another given point. This section is also useful to specify this starting value. Let's see in more details how all this works.

In passing, though, note that the relevant commands in this section are `initval`, `endval` or, more rarely, `histval` which is covered only in the Reference Manual. The first two are instead covered in what follows.

3.6.1 Stochastic models and steady states

In a stochastic setting, your model will need to be linearized before it is solved. To do so, Dynare needs to know your model's steady state (more details on finding a steady state, as well as tips to do so more efficiently, are provided in section 3.6.3 below). You can either enter exact steady state values into your .mod file, or just approximations and let Dynare find the exact steady state (which it will do using numerical methods based on your approximations). In either case, these values are entered in the `initval` block, as in the following fashion:

```
initval;
```

```
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0;
e = 0;
end;
```

Then, by using the command `steady`, you can control whether you want to start your simulations or impulse response functions from the steady state, or from the exact values you specified in the `initval` block. Adding

3.6.2 Deterministic models and initial values

Deterministic models do not need to be linearized in order to be solved. Thus, technically, you do not need to provide a steady state for these model. But practically, most researchers are still interested to see how a model reacts to shocks when originally in steady state. In the deterministic case, the `initval` block serves very similar functions as described above. If you wanted to shock your model starting from a steady state value, you would enter approximate (or exact) steady state values in the `initval` block, followed by the command `steady`. Otherwise, if you wanted to begin your solution path from an arbitrary point, you would enter those values in your `initval` block and not use the `steady` command. An illustration of the `initval` block in the deterministic case appears further below.

3.6.3 Finding a steady state

The difficulty in the above, of course, is calculating actual steady state values. Doing so borders on a form of art, and luck is unfortunately part of the equation. Yet, the following **TIPS!** may help.

As mentioned above, Dynare can help in finding your model's steady state by calling the appropriate Matlab functions. But it is usually only successful if the initial values you entered are close to the true steady state. If you have trouble finding the steady state of your model, you can begin by playing with the **options following the `steady` command**. These are:

- `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.
- `solve_algo = 3`: uses the Sims solver. This is the default option if none are specified.

For complicated models, finding suitable 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.

But even for simpler models, you may still run into difficulties in finding your steady state. If so, another option is to **enter your model in linear terms**. In this case, variables would be expressed in percent deviations from steady state. Thus, their initial values would all be zero. Unfortunately, if any of your original (non-linear) equations involve sums (a likely fact), your

linearized equations will include ratios of steady state values, which you would still need to calculate. Yet, you may be left needing to calculate fewer steady state values than in the original, non-linear, model.

Alternatively, you could also use an **external program to calculate exact steady state values**. For instance, you could write an external **Maple** file and then enter the steady state solution by hand in Dynare. But of course, this procedure could be time consuming and bothersome, especially if you want to alter parameter values (and thus steady states) to undertake robustness checks.

The alternative is to write a **Matlab** program to find your model's steady state. Doing so has the clear advantages of being able to incorporate your Matlab program directly into your .mod file so that running loops with different parameter values, for instance, becomes seamless. **NOTE!** When doing so, your matlab (.m) file should have the same name as your .mod file, followed by `_steadystate` For instance, if your .mod file is called `example.mod`, your Matlab file should be called `example_steadystate.m` and should be saved in the same directory as your .mod file. Dynare will automatically check the directory where you've saved your .mod file to see if such a Matlab file exists. If so, it will use that file to find steady state values regardless of whether you've provided initial values in your .mod file.

Because Matlab does not work with analytical expressions, though (unless you're working with a particular toolbox), you need to do a little work to write your steady state program. It is not enough to simply input the equations as you've written them in your .mod file and ask Matlab to solve the system. You will instead need to write your steady state program as if you were solving for the steady state by hand. That is, you need to input your expressions sequentially, whereby each left-hand side variable is written in terms of known parameters or variables already solved in the lines above. For example, the steady state file corresponding to the above example, in the stochastic case, would be: (** example file to be added shortly)

3.6.4 Checking system stability

TIP! A handy command that you can add after the `initval` or `endval` block (following the `steady` command if you decide to add one) is the `check` command. This **computes and displays the eigenvalues of your system** which are used in the solution method. As mentioned earlier, 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. If this condition is not

met, Dynare will tell you that the Blanchard-Kahn conditions are not satisfied (whether or not you insert the `check` command).

3.7 Adding shocks

3.7.1 Deterministic models - temporary shocks

When working with a deterministic model, you have the choice of introducing both temporary and permanent shocks. The distinction is that under a temporary shock, the model eventually comes back to steady state, while under a permanent shock, the model reaches a new steady state. In both cases, though, the shocks are entirely expected, as explained in our original discussion on stochastic and deterministic models.

To work with a **temporary shock**, you are free to set the duration and level of the shock. To specify a shock that lasts 9 periods on z_t, for instance, you would write:

```
shocks;
var z;
periods 1:9;
values 0.1;
end;
```

Given the above instructions, Dynare would replace the value of z_t specified in the `initval` block with the value of 0.1 entered above. If variables were in logs, this would have corresponded to a 10% shock. Note that you can also use the `mshocks` command which multiplies the initial value of an exogenous variable by the `mshocks` value. Finally, note that we could have entered future periods in the shocks block, such as `periods 5:10`, in order to study the anticipatory behavior of agents in response to future shocks.

3.7.2 Deterministic models - permanent shocks

To study the effects of a **permanent shock** hitting the economy today, such as a structural change in your model, you would not specify actual "shocks", but would simply tell the system to which (steady state) values you would like it to move and let Dynare calculate the transition path. To do so, you would use the `endval` block following the usual `initval` block. For instance, you may specify all values to remain common between the two blocks, except for the value of technology which you may presume changes permanently. The corresponding instructions would be:

```
initval;
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0;
end;
steady;

endval;
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0.1;
end;
steady;
```

where `steady` can also be added to the `endval` block, and serves the same functionality as described earlier (namely, of telling Dynare to start and/ or end at a steady state close to the values you entered. If you do not use `steady` after `endval`, and the latter does not list exact steady state values, you may impose on your system that it does not return to steady state. This is unusual. In this case, your problem would become a so-called two boundary problem, which, when solved, requires that the path of your endogenous variables pass through the steady state closest to your `endval` values). In our example, we make use of the second `steady` since the actual terminal steady state values are bound to be somewhat different from those entered above, which are nothing but the initial values for all variables except for technology.

In the above example, the value of technology would move to 0.1 in period 1 (tomorrow) and thereafter. But of course, the other variables - the endogenous variables - will take longer to reach their new steady state values. **TIP!** If you instead wanted to study the effects of a permanent but future shock (anticipated as usual), you would have to add a `shocks` block after the `endval` block to "undo" the first several periods of the permanent shock. For instance, suppose you wanted the value of technology to move to 0.1, but only in period 10. Then you would follow the above `endval` block with:

```
shocks;
var z;
```

```
periods 1:9;
values 0;
end;
```

3.7.3 Stochastic models

Recall from our earlier description of stochastic models that shocks are only allowed to be temporary. A permanent shock cannot be accommodated due to the need to stationarize the model around a steady state. Furthermore, shocks can only hit the system today, as the expectation of future shocks must be zero. With that in mind, we can however make the effect of the shock propagate slowly throughout the economy by introducing a "latent shock variable" such as e_t in our example, that affects the model's true exogenous variable, z_t in our example, which is itself an $AR(1)$, exactly as in the model we introduced from the outset. In that case, though, we would declare z_t as an endogenous variable and e_t as an exogenous variable, as we did in the preamble of the .mod file in section 3.4. Supposing we wanted to add a shock with variance σ^2, where σ is determined in the preamble block, we would write:

```
shocks;
var e = sigma^2;
end;
```

TIP! You can actually **mix in deterministic shocks** in stochastic models by using the commands `varexo_det` and listing some shocks as lasting more than one period in the `shocks` block. For information on how to do so, please see the Reference Manual. This can be particularly useful if you're studying the effects of anticipated shocks in a stochastic model. For instance, you may be interested in what happens to your monetary model if agents began expecting higher inflation, or a depreciation of your currency.

3.8 Selecting a computation

So far, we have written an instructive .mod file, but what should Dynare do with it? What are we interested in? In most cases, it will be impulse response functions (IRFs) due to the external shocks. Let's see which are the appropriate commands to give to Dynare. Again, we will distinguish between deterministic and stochastic models.

3.8.1 For deterministic models

In the deterministic case, all you need to do is add the command `simul` at the bottom of your .mod file. Note that the command takes the option `[ (periods=INTEGER) ]` The command `simul` triggers the computation a numerical simulation of the trajectory of the model's solution for the number of periods set in the option. To do so, it uses a Newton method to solve simultaneously all the equations for every period (see Juillard (1996) for details). Note that unless you use the `endval` command, the algorithm makes the simplifying assumption that the system is back to equilibrium after the specified number of periods. Thus, you must specify a large enough number of periods such that increasing it further doesn't change the simulation for all practical purpose. In the case of a temporary shock, for instance, the trajectory will basicaly describe how the system gets back to equilibrium after being perturbed from the shocks you entered.

3.8.2 For stochastic models

In the more common case of stochastic models, the command `stoch_simul` is appropriate. This command instructs Dynare to compute a Taylor approximation of the decision and transition functions for the model (the equations listing current values of the endogenous variables of the model as a function of the previous state of the model and current shocks), impulse response functions and various descriptive statistics (moments, variance decomposition, correlation and autocorrelation coefficients).[1]

Impulse response functions are the expected future path of the endogenous variables conditional on a shock in period 1 of one standard deviation.**TIP!** If you linearize your model up to a first order, impulse response functions are simply the algebraic forward iteration of your model's policy or decision rule. If you instead linearize to a second order, impulse response functions will be the result of actual Monte Carlo simulations of future shocks. This is because in second order linear equations, you will have cross terms involving the shocks, so that the effects of the shocks depend on the state of the system when the shocks hit. Thus, it is impossible to get algebraic average values of all future shocks and their impact. The technique is instead to pull future shocks from their distribution and see how they impact your system, and repeat this procedure a multitude of times in order to draw out an average response. That said, note that future shocks will not have a significant impact

[1] 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.

on your results, since they get averaged between each Monte Carlo trial and in the limit should sum to zero, given their mean of zero. Note that in the case of a second order approximation, Dynare will return the actual sample moments from the simulations. For first order linearizations, Dynare will instead report theoretical moments. In both cases, the return to steady state is asymptotic, **TIP!** thus you should make sure to specify sufficient periods in your IRFs such that you actually see your graphs return to steady state. Details on implementing this appear below.

If you're interested to peer a little further into what exactly is going on behind the scenes of Dynare's computations, have a look at Chapter 7. Here instead, we focus on the application of the command and reproduce below the most common options that can be added to `stoch_simul`. For a complete list of options, please see the Reference Manual.

Options following the `stoch_simul` command:

- ar = INTEGER: Order of autocorrelation coefficients to compute and to print (default = 5).
- dr_algo = 0 or 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 (2004) (default) and 1 moves the point around which the Taylor expansion is computed toward the means of the distribution as in Collard and Juillard (2001b).
- 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 = INTEGER before computing moments (default: no filter).
- hp_ngrid = INTEGER: number of points in the grid for the discreet Inverse Fast Fourier Transform used in the HP filter computation. It may be necessary to increase it for highly autocorrelated processes (default = 512001-90.929-293.654cmBT-1284cm0g0G1001-90.929-T-1284cmBT/F3810.909Tf107.293T-1284T
(default =tingprssestheingof

- nomoments: doesn't print moments of the endogenous variables (printing them is the default).

- noprint: cancel any printing; usefull for loops.

- order = 1 or 2 : order of Taylor approximation (default = 2), unless you're working with a linear model in which case the order is automatically set to 1.

- periods = INTEGER: specifies the number of periods to use in simulations (default = 0). **TIP!** A simulation is similar to running impulse response functions with a model linearized to the second order, in the way that both sample shocks from their distribution to see how the system reacts, but a simulation only repeats the process once, whereas impulse response functions run a multitude of Monte Carlo trials in order to get an average response of your system.

- qz_criterium = INTEGER or 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 or DOUBLE or (EXPRESSION): specifies a seed for the random number 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). If you linearized to a second order, Dynare will actually undertake Monte Carlo simulations to generate moments of your variables. Because of the simulation, results are bound to be slightly different each time you run your program, except if you fix the seed for the random number generator. **TIP!** If you do decide to fix the seed, you should at least try to run your program without using `simul_seed`, just to check the robustness of your results.

Going back to our good old example, suppose we were interested in printing all the various measures of moments of our variables, want to see impulse response functions for all variables, are basically happy with all default options and want to carry out simulations over a good number of periods. We would then end our .mod file with the following command:

```
stoch_simul(periods=2100);
```

3.9 The complete .mod file

For completion's sake, and for the pleasure of seeing our work bear its fruits, here are the complete .mod files corresponding to our example for the deterministic and stochastic case. You can find the corresponding files in the *models* folder under *UserGuide* in your installation of Dynare. The files are called `RBC_Monop_JFV.mod` for stochastic models and `RBC_Monop_Det.mod` for deterministic models.

3.9.1 The stochastic model

```
var y c k i l y_l w r z;
varexo e;
parameters beta psi delta alpha rho sigma epsilon;
alpha = 0.33;
beta = 0.99;
delta = 0.023;
psi = 1.75;
rho = 0.95;
sigma = (0.007(1-alpha));
epsilon = 10;

model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_l = y/l;
z = rho*z(-1)+e;
end;

initval;
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0;
e = 0;
end;
```

```
steady;

check;

shocks;
var e = sigma^2;
end;

stoch_simul(periods=2100);
```

3.9.2 The deterministic model (case of temporary shock)

```
var y c k i l y_l w r ;
varexo z;
parameters beta psi delta alpha sigma epsilon;
alpha = 0.33;
beta = 0.99;
delta = 0.023;
psi = 1.75;
sigma = (0.007(1-alpha));
epsilon = 10;

model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_l = y/l;
end;

initval;
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0;
end;

steady;
```

```
check;

shocks;
var z; periods 1:9;
values 0.1;
end;

simul(periods=2100);
```

3.10 File execution and results

To see this all come to life, let's run our .mod file, which is conveniently installed by default in the Dynare "examples" directory (the .mod file corresponding to the stochastic model is called RBC_Monop_JFV.mod and that corresponding to the deterministic model is called RBC_Monop_Det.mod). (** note, this may not be the case when testing the beta version of Matlab version 4)

To run a .mod file, navigate within Matlab to the directory where the example .mod files are stored. You can do this by clicking in the "current directory" window of Matlab, or typing the path directly in the top white field of Matlab. Once there, all you need to do is place your cursor in the Matlab command window and type, for instance, `dynare ExSolStoch;` to execute your .mod file.

Running these .mod files should take at most 30 seconds. As a result, you should get two forms of output - tabular in the Matlab command window and graphical in one or more pop-up windows. Let's review these results.

3.10.1 Results - stochastic models

The tabular results can be summarized as follows:

1. **Model summary:** a count of the various variable types in your model (endogenous, jumpers, etc...).

2. **Eigenvalues** should be displayed, and you should see a confirmation of the Blanchard-Kahn conditions if you used the command `check` in your .mod file.

3. **Matrix of covariance of exogenous shocks:** this should square with the values of the shock variances and co-variances you provided in the .mod file.

4. **Policy and transition functions:** Solving the rational exectation model, $\mathbb{E}_t[f(y_{t+1}, y_t, y_{t-1}, u_t)] = 0$, means finding an unkown function, $y_t = g(y_{t-1}, u_t)$ that could be plugged into the original model and satisfy the implied restrictions (the first order conditions). A first order approximation of this function can be written as $y_t = \bar{y} + g_y \hat{y}_{t-1} + g_u u_t$, with $\hat{y}_t = y_t - \bar{y}$ and $\bar{y}$ being the steadystate value of y, and where g_x is the partial derivative of the g function with respect to variable x. In other words, the function g is a time recursive (approximated) representation of the model that can generate timeseries that will approximatively satisfy the rational expectation hypothesis contained in the original model. In Dynare, the table "Policy and Transition function" contains the elements of g_y and g_u. Details on the policy and transition function can be found in Chapter 6.

5. **Moments of simulated variables:** up to the fourth moments.

6. **Correlation of simulated variables:** these are the contemporaneous correlations, presented in a table.

7. **Autocorrelation of simulated variables:** up to the fifth lag, as specified in the options of `stoch_simul`.

The graphical results, instead, show the actual impulse response functions for each of the endogenous variables, given that they actually moved. These can be especially useful in visualizing the shape of the transition functions and the extent to which each variable is affected. **TIP!** If some variables do not return to their steady state, either check that you have included enough periods in your simulations, or make sure that your model is stationary, i.e. that your steady state actually exists and is stable. If not, you should detrend your variables and rewrite your model in terms of those variables.

3.10.2 Results - deterministic models

Automatically displayed results are much more scarce in the case of deterministic models. If you entered `steady`, you will get a list of your steady state results. If you entered `check`, eigenvalues will also be displayed and you should receive a statement that the rank condition has been satisfied, if all goes well! Finally, you will see some intermediate output: the errors at each iteration of the Newton solver used to estimate the solution to your model. **TIP!** You should see these errors decrease upon each iteration; if not, your model will probably not converge. If so, you may want to try to increase the periods for the transition to the new steady state (the number of simulations

periods). But more often, it may be a good idea to revise your equations. Of course, although Dynare does not display a rich set of statistics and graphs corresponding to the simulated output, it does not mean that you cannot create these by hand from Matlab. To do so, you should start by looking at section 4.1.3 of chapter 4 on finding, saving and viewing your output.

Chapter 4

Solving DSGE models - advanced topics

This chapter is a collection of topics - not all related to each other - that you will probably find interesting or at least understandable, if you have read, and/ or feel comfortable with, the earlier chapter 3 on the basics of solving DSGE models. To provide at least some consistency, this chapter is divided into three sections. **The first section** deals directly with features of Dynare, such as dealing with correlated shocks, finding and saving your output, using loops, referring to external files and dealing with infinite eigenvalues. **The second section** overviews some of the inner workings of Dynare. The goal is to provide a brief explanation of the files that are created by Dynare to help you in troubleshooting or provide a starting point in case you actually want to customize the way Dynare works. **The third section** of the chapter focusses on modeling tips optimized for Dynare, but possibly also helpful for other work.

4.1 Dynare features and functionality

4.1.1 Other examples

Other examples of .mod files used to generate impulse response functions are available on the Dynare website. In particular, Jesus Fernandez-Villaverde has provided a series of RBC model variants (from the most basic to some including variable capacity utilization, indivisible labor and investment specific technological change). You can find these, along with helpful notes and explanations, in the Official Examples section of the Dynare website.

Also, don't forget to check occasionally the Open Online Examples page to see if any other user has posted an example that could help you in your

work; or maybe you would like to post an example there yourself?

4.1.2 Alternative, complete example

The following example aims to give you an alternative example to the one in chapter 3, to learn the workings of Dynare. It also aims to give you exposure to dealing with **several correlated shocks**. Your model may have two or more shocks, and these may be correlated to each other. The example below illustrates how you would introduce this into Dynare. Actually, the example provided is somewhat more complete than strictly necessary. This is to give you an alternative, full-blown example to the one described in chapter 3.

The model

The model is a simplified standard RBC model taken from Collard and Juillard (2003) which served as the original User Guide for Dynare.

The economy consists of an infinitely living representative agent who values consumption c_t and labor services h_t according to the following utility function

$$\mathbb{E}_t \sum_{\tau=t}^{\infty} \beta^{\tau-t} \left(\log(c_t) - \theta \frac{h_t^{1+\psi}}{1+\psi} \right)$$

where, as usual, the discount factor $0 < \beta < 1$, the disutility of labor $\theta > 0$ and the labor supply elasticity $\psi \geq 0$.

A social planner maximizes this utility function subject to the resource constraint

$$c_t + i_t = y_t$$

where i_t is investment and y_t output. Consumers are therefore also owners of the firms. The economy is a real economy, where part of output can be consumed and part invested to form physical capital. As is standard, the law of motion of capital is given by

$$k_{t+1} = \exp(b_t) i_t + (1-\delta) k_t$$

with $0 < \delta < 1$, where δ is physical depreciation and b_t a shock affecting incorporated technological progress.

We assume output is produced according to a standard constant returns to scale technology of the form

$$y_t = \exp(a_t) k_t^{\alpha} h_t^{1-\alpha}$$

with α being the capital elasticity in the production function, with $0 < \alpha < 1$, and where a_t represents a stochastic technological shock (or Solow residual).

Finally, we specify a **shock structure** that allows for shocks to display persistence across time and correlation in the current period. That is

$$\begin{pmatrix} a_t \\ b_t \end{pmatrix} = \begin{pmatrix} \rho & \tau \\ \tau & \rho \end{pmatrix} \begin{pmatrix} a_{t-1} \\ b_{t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_t \\ \nu_t \end{pmatrix}$$

where $|\rho + \tau| < 1$ and $|\rho - \tau| < 1$ to ensure stationarity (we call ρ the coefficient of persistence and τ that of cross-persistence). Furthermore, we assume $\mathbb{E}_t(\epsilon_t) = 0$, $\mathbb{E}_t(\nu_t) = 0$ and that the contemporaneous variance-covariance matrix of the innovations ϵ_t and ν_t is given by

$$\begin{pmatrix} \sigma_\epsilon^2 & \psi\sigma_\epsilon\sigma_\nu \\ \psi\sigma_\epsilon\sigma_\nu & \sigma_\nu^2 \end{pmatrix}$$

and where $corr(\epsilon_t\nu_s) = 0$, $corr(\epsilon_t\epsilon_s) = 0$ and $corr(\nu_t\nu_s) = 0$ for all $t \neq s$.

This system - probably quite similar to standard RBC models you have run into - yields the following first order conditions (which are straightforward to reproduce in case you have doubts...) and equilibrium conditions drawn from the description above. Note that the first equation captures the labor supply function and the second the intertemporal consumption Euler equation.

$$\begin{aligned} c_t\theta h_t^{1+\psi} &= (1-\alpha)y_t \\ 1 = \beta\mathbb{E}_t\left[\left(\frac{\exp(b_t)c_t}{\exp(b_{t+1})c_{t+1}}\right)\left(\exp(b_{t+1})\alpha\frac{y_{t+1}}{k_{t+1}} + 1 - \delta\right)\right] \\ y_t &= \exp(a_t)k_t^\alpha h_t^{1-\alpha} \\ k_{t+1} &= \exp(b_t)i_t + (1-\delta)k_t \\ a_t &= \rho a_{t-1} + \tau b_{t-1} + \epsilon_t \\ b_t &= \tau a_{t-1} + \rho b_{t-1} + \nu_t \end{aligned}$$

The .mod file

To "translate" the model into a language understandable by Dynare, we would follow the steps outlined in chapter 3. We will assume that you're comfortable with these and simply present the final .mod file below. Fist, though, note that to introduce shocks into Dynare, we have two options (this was not discussed in the earlier chapter). Either write:

```
shocks;
var e; stderr 0.009;
var u; stderr 0.009;
```

```
var e, u = phi*0.009*0.009;
end;
```

where the last line specifies the contemporaneous correlation between our two exogenous variables.

Alternatively, you can also write:

```
shocks;
var e = 0.009 ^ 2;
var u = 0.009 ^ 2;
var e, u = phi*0.009*0.009;
end;
```

So that you can gain experience by manipulating the entire model, here is the complete .mod file corresponding to the above example. You can find the corresponding file in the *models* folder under *UserGuide* in your installation of Dynare. The file is called *Alt_Ex1.mod*.

```
var y, c, k, a, h, b;
varexo e,u;
parameters beta, rho, alpha, delta, theta, psi, tau;
alpha = 0.36;
rho = 0.95;
tau = 0.025;
beta = 0.99;
delta = 0.025;
psi = 0;
theta = 2.95;
phi = 0.1;

model;
c*theta*h^(1+psi)=(1-alpha)*y;
k = beta*(((exp(b)*c)/(exp(b(+1))*c(+1)))
*(exp(b(+1))*alpha*y(+1)+(1-delta)*k));
y = exp(a)*(k(-1)^alpha)*(h^(1-alpha));
k = exp(b)*(y-c)+(1-delta)*k(-1);
a = rho*a(-1)+tau*b(-1) + e;
b = tau*a(-1)+rho*b(-1) + u;
end;

initval;
y = 1.08068253095672;
```

```
c = 0.80359242014163;
h = 0.29175631001732;
k = 5;
a = 0;
b = 0;
e = 0;
u = 0;
end;

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

stoch_simul(periods=2100);
```

4.1.3 Finding, saving and viewing your output

Where is output stored? Most of the moments of interest are stored in global variable `oo_` You can easily browse this global variable in Matlab by either calling it in the command line, or using the workspace interface. In global variable `oo_` you will find the following (**NOTE!** variables will always appear in the order in which you declared them in the preamble block of your .mod file):

- `steady_state`: the steady state of your variables
- `mean`: the mean of your variables
- `var`: the variance of your variables
- `autocorr`: the various autocorrelation matrices of your variables. Each row of these matrices will correspond to a variables in time t, and columns correspond to the variables lagged 1, for the first matrix, then lagged 2 for the second matrix, and so on. Thus, the matrix of autocorrelations that is automatically displayed in the results after running `stoch_simul` has, running down each column, the diagonal elements of each of the various autocorrelation matrices described here.
- `gamma_y`: the matrices of autocovariances. `gamma_y{1}` represents variances, while `gamma_y{2}` represents autocovariances where variables on each column are lagged by one period and so on. By default, Dynare will return autocovariances with a lag of 5. The last matrix (`gamma_y{7}` in

the default case) returns the **variance decomposition**, where each column captures the independent contribution of each shock to the variance of each variable.

Furthermore, if you decide to run impulse response functions, you will find a global variable `oo_.irfs` comprising of vectors named `endogenous variable_exogenous variable`, like `y_e`, reporting the values of the endogenous variables corresponding to the impulse response functions, as a result of the independent impulse of each exogenous shock.

To save your simulated variables, you can add the following command at the end of your .mod file: `dynasave (FILENAME) [variable names separated by commas]` If no variable names are specified in the optional field, Dynare will save all endogenous variables. In Matlab, variables saved with the `dynasave` command can be retrieved by using the Matlab command `load -mat FILENAME`.

4.1.4 Referring to external files

You may find it convenient to refer to an external file, either to compute the steady state of your model, or when specifying shocks in an external file. The former is described in section 3.6 of chapter 3 when discussing steady states. The advantage of using Matlab, say, to find your model's steady state was clear with respect to Dynare version 3, as the latter resorted to numerical approximations to find steady state values. But Dynare version 4 now uses the same analytical methods available in Matlab. For most usage scenarios, you should therefore do just as well to ask Dynare to compute your model's steady state (except, maybe, if you want to run loops, to vary your parameter values, for instance, in which case writing a Matlab program may be more handy).

But you may also be interested in the second possibility described above, namely of specifying shocks in an external file, to simulate a model based on

4.1.5 Infinite eigenvalues

If you use the command `check` in your .mod file, Dynare will report your system's eigenvalues and tell you if these meet the Blanchard-Kahn conditions. At that point, don't worry if you get infinite eigenvalues - these are are firmly grounded in the theory of generalized eigenvalues. They have no detrimental influence on the solution algorithm. As far as Blanchard-Kahn conditions are concerned infinite eigenvalues are counted as explosive roots of modulus larger than one.

4.2 Files created by Dynare

At times, you may get a message that there is an error in a file with a new name, or you may want to have a closer look at how Dynare actually solves your model - out of curiosity or maybe to do some customization of your own. You may therefore find it helpful to get a brief overview of the internal files that Dynare generates and the function of each one.

The dynare pre-processors essentially does three successive tasks: (i) parsing of the mod file (it checks that the mod file is syntactically correct), and its translation into internal machine representation (in particular, model equations are translated into expression trees), (ii) symbolic derivation of the model equations, up to the needed order (depending on the computing needs), (iii) outputting of several files, which are used from matlab. If the mod file is "filename.mod", then the pre-processor creates the following files:

- **filename.m**: a matlab file containing several instructions, notably the parameter initializations and the matlab calls corresponding to computing tasks

- **filename_dynamic.m**: a matlab file containing the model equations and their derivatives (first, second and maybe third). Endogenous variables (resp. exogenous variables, parameters) are contained in a "y" (resp. "x", "params") vector, with an index number depending on the declaration order. The "y" vector has as many entries as their are (variable, lag) pairs in the declared model. The model equations residuals are stored in a vector named "residuals". The model jacobian is put in "g1" matrix. Second (resp. third) derivatives are in "g2" matrix (resp. "g3"). If the "use_dll" option has been specified in the model declaration, the pre-processor will output a C file (with .c extension) rather than a matlab file. It is then compiled to create a library (DLL) file. Using a compiled C file is supposed to give better computing performance in model simulation/estimation.

- **filename_static.m**: a matlab file containing the stationarized version of the model (i.e. where lagged variables are replaced by current variables), with its jacobian. Used to compute the steady state. Same notations than the dynamic file. Replaced by a C file when "use_dll" option is specified.

4.3 Modeling tips

4.3.1 Stationarizing your model

Models in Dynare must be stationary, such that you can linearize them around a steady state and return to steady state after a shock. Thus, you must first stationarize your model, then linearize it, either by hand, or by letting Dynare do the work. You can then reconstruct ex-post the non-stationary simulated variables after running impulse response functions.

For deterministic models, the trick is to use only stationary variables in $t+1$. More generally, if y_t is $I(1)$, you can always write y_{t+1} as $y_t + dy_{t+1}$, where $dy_t = y_t - y_{t-1}$. Of course, you need to know the value of dy_t at the final equilibrium.

Note that in a stationary model, it is expected that variables will eventually go back to steady state after the initial shock. If you expect to see a growing curve for a variable, you are thinking about a growth model. Because growth models are nonstationary, it is easier to work with the stationarized version of such models. Again, if you know the trend, you can always add it back after the simulation of the stationary components of the variables.

4.3.2 Expectations taken in the past

For instance, to enter the term $\mathbb{E}_{t-1}y_t$, define $s_t = \mathbb{E}_t[y_{t+1}]$ and then use $s(-1)$ in your .mod file. Note that, because of Jensen's inequality, you cannot do this for terms that enter your equation in a non-linear fashion. If you do have non-linear terms on which you want to take expectations in the past, you would need to apply the above manipulation to the entire equation, as if y_t were an equation, not just a variable.

4.3.3 Infinite sums

Dealing with infinite sums is tricky in general, and needs particular care when working with Dynare. The trick is to use a **recursive representation** of the sum. For example, suppose your model included:

$$\sum_{j=0}^{\infty} \beta^j x_{t+j} = 0,$$

Note that the above can also be written by using an auxiliary variable S_t, defined as:

$$S_t \equiv \sum_{j=0}^{\infty} \beta^j x_{t+j},$$

which can also be written in the following recursive manner:

$$S_t \equiv \sum_{j=0}^{\infty} \beta^j x_{t+j} = x_t + \sum_{j=1}^{\infty} \beta^j x_{t+j} = x_t + \beta \sum_{j=0}^{\infty} \beta^j x_{t+1+j} \equiv x_t + S_{t+1}$$

This formulation turns out to be useful in problems of the following form:

$$\sum_{j=0}^{\infty} \beta^j x_{t+j} = p_t \sum_{j=0}^{\infty} \gamma^j y_{t+j},$$

which can be written as a recursive system of the form:

$$S1_t = x_t + \beta S1_{t+1},$$

$$S2_t = y_t + \gamma S2_{t+1},$$

$$S1 = p_t S2.$$

This is particularly helpful, for instance, in a **Calvo type setting**, as illustrated in the following brief example. The RBC model with monopolistic competition introduced in chapter 3 involved flexible prices. The extension with sticky prices, à la Calvo for instance, is instead typical of the new Keynesian monetary literature, exemplified by papers such as Clarida, Gali, and Gertler (1999).

The optimal price for a firm resetting its price in period t, given that it will be able to reset its price only with probability $1 - \theta$ each period, is

$$p_t^*(i) = \mu + (1 - \beta\theta) \sum_{k=0}^{\infty} (\beta\theta)^k \mathbb{E}_t[mc_{t+k}^n(i)]$$

where μ is the markup, β is a discount factor, i represents a firm of the continuum between 0 and 1, and mc_t is marginal cost as described in the example in chapter 3. The trouble, of course, is **how to input this infinite sum into Dynare**?

It turns out that the Calvo price setting implies that the aggregate price follows the equation of motion $p_t = \theta p_{t-1} + (1 - \theta)p_t^*$, thus implying the following inflation relationship $\pi_t = (1 - \theta)(p_t^* - p_{t-1})$. Finally, we can also rewrite the optimal price setting equation, after some algebraic manipulations, as

$$p_t^* - p_{t-1} = (1 - \beta\theta) \sum_{k=0}^{\infty} (\beta\theta)^k \mathbb{E}_t[\widehat{mc}_{t+k}] + \sum_{k=0}^{\infty} (\beta\theta)^k \mathbb{E}_t[\pi_{t+k}]$$

where $\widehat{mc}_{t+k} = mc_{t+k} + \mu$ is the deviation of the marginal cost from its natural rate, defined as the marginal cost when prices are perfectly flexible.

The trick now is to note that the above can be written recursively, by writing the right hand side as the first term of the sum (with $k = 0$) plus the remainder of the sum, which can be written as the left hand side term scrolled forward one period and appropriately discounted. Mathematically, this yields:

$$p_t^* - p_{t-1} = \ (1 - \beta\theta)\widehat{mc}_{t+k} + \pi_t + \beta\ \theta\mathbb{E}_t[p_{t+1}^* - p_t]$$

which has gotten rid of our infinite sum! That would be enough for Dynare, but for convenience, we can go one step further and write the above as

$$\pi_t = \beta\mathbb{E}_t[\pi_{t+1}] + \lambda\widehat{mc}_t$$

where $\lambda \equiv \frac{(1-\theta)(1-\beta\theta)}{\theta}$, which is the recognizable inflation equation in the new Keynesian (or new Neoclassical) monetary literature.

4.3.4 Infinite sums with changing timing of expectations

When you are not able to write an infinite sum recursively, as the index of the expectations changes with each element of the sum, as in the following example, a different approach than the one mentioned above is necessary.

Suppose your model included the following sum:

$$y_t = \sum_{j=0}^{\infty} \mathbb{E}_{t-j} x_t$$

where y_t and x_t are endogenous variables.

In Dynare, the best way to handle this is to write out the first k terms explicitly and enter each one in Dynare, such as: $\mathbb{E}_{t-1}x_t + \mathbb{E}_{t-2}x_t + \ldots + \mathbb{E}_{t-k}x_t$.

Chapter 5

Estimating DSGE models - basics

As in the chapter 3, this chapter is structured around an example. The goal of this chapter is to lead you through the basic functionality in Dynare to estimate models using Bayesian techniques, so that by the end of the chapter you should have the capacity to estimate a model of your own. This chapter is therefore very practically-oriented and abstracts from the underlying computations that Dynare undertakes to estimate a model; that subject is instead covered in some depth in chapter 8, while more advanced topics of practical appeal are discussed in chapter 6.

5.1 Introducing an example

The example introduced in this chapter is particularly basic. This is for two reasons. First, we did not want to introduce yet another example in this section; there's enough new material to keep you busy. Instead, we thought it would be easiest to simply continue working with the example introduced in chapter 3 with which you are probably already quite familiar. Second, the goal of the example in this chapter is really to explain features in context, but not necessarily to duplicate a "real life scenario" you would face when writing a paper. Once you feel comfortable with the content of this chapter, though, you can always move on to chapter 6 where you will find a full-fledged replication of a recent academic paper, featuring a non-stationary model.

Recall from chapter 3 that we are dealing with an RBC model with monopolistic competition. Suppose we had data on business cycle variations of output. Suppose also that we thought our little RBC model did a good job of reproducing reality. We could then use Bayesian methods to estimate the parameters of the model: α, the capital share of output, β, the discount factor,

δ, the depreciation rate, ψ, the weight on leisure in the utility function, ρ, the degree of persistence in productivity, and ϵ, the markup parameter. Note that in Bayesian estimation, the condition for undertaking estimation is that there be at least as many shocks as there are observables (a less stringent condition than for maximum likelihood estimation). It may be that this does not allow you to identify all your parameters - yielding posterior distributions identical to prior distributions - but the Bayesian estimation procedure would still run successfully. Let's see how to go about doing this.

5.2 Declaring variables and parameters

To input the above model into Dynare for estimation purposes, we must first declare the model's variables in the preamble of the .mod file. This is done exactly as described in chapter 3 on solving DSGE models. We thus begin the .mod file with:

```
var y c k i l y_l w r z;
varexo e;

parameters beta psi delta alpha rho epsilon;
```

5.3 Declaring the model

Suppose that the equation of motion of technology is a **stationary** AR(1) with an autoregressive parameter, ρ, less than one. The model's variables would therefore be stationary and we can proceed without complications. The alternative scenario with **non-stationary variables** is more complicated and dealt with in chapter 6 (in the additional example). In the stationary case, our model block would look exactly as in chater 3:

```
model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_l = y/l;
z = rho*z(-1)+e;
```

```
end;
```

5.4 Declaring observable variables

This should not come as a surprise. Dynare must know which variables are observable for the estimation procedure. **NOTE!** These variables must be available in the data file, as explained in section 5.7 below. For the moment, we write:

```
varobs Y;
```

5.5 Specifying the steady state

Before Dynare estimates a model, it first linearizes it around a steady state. Thus, a steady state must exist for the model and although Dynare can calculate it, we must give it a hand by declaring approximate values for the steady state. This is just as explained in details and according to the same syntax outlined in chapter 3, covering the `initval`, `steady` and `check` commands. In fact, as this chapter uses the same model as that outlined in chapter 3, the steady state block will look exactly the same.

TIP! During estimation, in finding the posterior mode, Dynare recalculates the steady state of the model at each iteration of the optimization routine (more on this later), based on the newest round of parameters available. Thus, by providing approximate initial values and relying solely on the built-in Dynare algorithm to find the steady state (a numerical procedure), you will significantly slow down the computation of the posterior mode. Dynare will end up spending 60 to 70% of the time recalculating steady states. It is much more efficient to write an external **Matlab steady state file** and let Dynare use that file to find the steady state of your model by algebraic procedure. For more details on writing an external Matlab file to find your model's steady state, please refer to section 3.6.3 of chapter 3.

5.6 Declaring priors

Priors play an important role in Bayesian estimation and consequently deserve a central role in the specification of the .mod file. Priors, in Bayesian estimation, are declared as a distribution. The general syntax to introduce priors in Dynare is the following:

```
estimated_params;
PARAMETER NAME, PRIOR_SHAPE, PRIOR_MEAN, PRIOR_STANDARD_ERROR [, PRIOR
3rd PARAMETER] [,PRIOR 4th PARAMETER] ;
end;
```

where the following table defines each term more clearly

PRIOR_SHAPE	**Corresponding distribution**	**Range**
NORMAL_PDF	$N(\mu, \sigma)$	$\mathbb{R}$
GAMMA_PDF	$G_2(\mu, \sigma, p_3)$	$[p_3, +\infty)$
BETA_PDF	$B(\mu, \sigma, p_3, p_4)$	$[p_3, p_4]$
INV_GAMMA_PDF	$IG_1(\mu, \sigma)$	$\mathbb{R}^+$
UNIFORM_PDF	$U(p_3, p_4)$	$[p_3, p_4]$

where μ is the `PRIOR_MEAN`, σ is the `PRIOR_STANDARD_ERROR`, p_3 is the `PRIOR` 3^{rd} `PARAMETER` (whose default is 0) and p_4 is the `PRIOR` 4^{th} `PARAMETER` (whose default is 1). **TIP!** When specifying a uniform distribution between 0 and 1 as a prior for a parameter, say α, you therefore have to put two empty spaces for parameters μ and σ, and then specify parameters $p3$ and $p4$, since the uniform distribution only takes $p3$ and $p4$ as arguments. For instance, you would write `alpha, uniform_pdf, , , 0,1;`

For a more complete review of all possible options for declaring priors, as well as the syntax to declare priors for maximum likelihood estimation (not Bayesian), see the Reference Manual. Note also that if some parameters in a model are calibrated and are not to be estimated, you should declare them as such, by using the `parameters` command and its related syntax, as explained in chapter 3.

TIP! Choosing the appropriate prior for your parameters is a tricky, yet very important endeavor. It is worth spending time on your choice of priors and to test the robustness of your results to your priors. Some considerations may prove helpful. First, think about the domain of your prior over each parameter. Should it be bounded? Should it be opened on either or both sides? Remember also that if you specify a probability of zero over a certain domain in your prior, you will necessarily also find a probability of zero in your posterior distribution. Then, think about the shape of your prior distribution. Should it be symmetric? Skewed? If so, on which side? You may also go one step further and build a distribution for each of your parameters in your mind. Ask yourself, for instance, what is the probability that your parameter is bigger than a certain value, and repeat the exercise by incrementally decreasing that value. You can then pick the standard distribution that best fits

your perceived distribution. Finally, instead of describing here the shapes and properties of each standard distribution available in Dynare, you are instead encouraged to visualize these distributions yourself, either in a statistics book or on the Web.

TIP! It may at times be desirable to estimate a transformation of a parameter appearing in the model, rather than the parameter itself. In such a

5.7 Launching the estimation

To ask Dynare to estimate a model, all that is necessary is to add the command `estimation` at the end of the .mod file. Easy enough. But the real complexity comes from the options available for the command (to be entered in parentheses and sequentially, separated by commas, after the command `estimation`). Below, we list the most common and useful options, and encourage you to view the Reference Manual for a complete list.

1. datafile = FILENAME: the datafile (a .m file, a .mat file, or an .xls file). Note that observations do not need to show up in any order, but vectors of observations need to be named with the same names as those in `var_obs`. In Excel files, for instance, observations could be ordered in columns, and variable names would show up in the first cell of each column.

2. nobs = INTEGER: the number of observations to be used (default: all observations in the file)

3. first_obs = INTEGER: the number of the first observation to be used (default = 1). This is useful when running loops, or instance, to divide the observations into sub-periods.

4. prefilter = 1: the estimation procedure demeans the data (default=0, no prefiltering). This is useful if model variables are in deviations from steady state, for instance, and therefore have zero mean. Demeaning the observations would also impose a zero mean on the observed variables.

5. nograph: no graphs should be plotted.

6. conf_sig = {INTEGER — DOUBLE }: the level for the confidence intervals reported in the results (default = 0.90)

7. mh_replic = INTEGER: number of replication for Metropolis Hasting algorithm. For the time being, mh_replic should be larger than 1200 (default = 20000)

8. mh_nblocks = INTEGER: number of parallel chains for Metropolis Hasting algorithm (default = 2). Despite this low default value, it is advisable to work with a higher value, such as 5 or more. This improves the computation of between group variance of the parameter means, one of the key criteria to evaluate the efficiency of the Metropolis-Hastings to evaluate the posterior distribution. More details on this subject appear in Chapter 6.

9. mh_drop = DOUBLE: the fraction of initially generated parameter vectors to be dropped before using posterior simulations (default = 0.5; this means that the first half of the draws from the Metropolis-Hastings are discarded).

10. 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 tuned to obtain, ideally, an acceptance rate of 25% in the Metropolis- Hastings algorithm (default = 0.2). The idea is not to reject or accept too often a candidate parameter; the literature has settled on a value of between 0.2 and 0.4. If the acceptance rate were too high, your Metropolis-Hastings iterations would never visit the tails of a distribution, while if it were too low, the iterations would get stuck in a subspace of the parameter range. Note that the acceptance rate drops if you increase the scale used in the jumping distribution and vice a versa.

11. mh_init_scale=DOUBLE: the scale to be used for drawing the initial value of the Metropolis-Hastings chain (default=2*mh_jscale). The idea here is to draw initial values from a stretched out distribution in order to maximize the chances of these values not being too close together, which would defeat the purpose of running several blocks of Metropolis-Hastings chains.

12. 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. This is a particularly helpful option to speed up the estimation process if you have already undertaken initial estimations and have values of the posterior mode.

13. 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 (see the Reference Manual to set options for this command).

    2: uses Lester Ingber's Adaptive Simulated Annealing.

    3: uses Matlab fminunc.

    4 (default): uses Chris Sim's csminwel.

14. mode_check: when mode_check is set, Dynare plots the minus of the posterior density for values around the computed mode for each estimated parameter in turn. This is helpful to diagnose problems with the optimizer. A clear indication of a problem would be that the mode is not at the trough (bottom of the minus) of the posterior distribution.

15. load_mh_file: when load_mh_file is declared, Dynare adds to previous Metropolis-Hastings simulations instead of starting from scratch. Again, this is a useful option to speed up the process of estimation.

16. nodiagnostic: doesn't compute the convergence diagnostics for Metropolis-Hastings (default: diagnostics are computed and displayed). Actually seeing if the various blocks of Metropolis-Hastings runs converge is a powerful and useful option to build confidence in your model estimation. More details on these diagnostics are given in Chapter 6.

17. bayesian_irf: triggers the computation of the posterior distribution of impulse response functions (IRFs). The length of the IRFs are controlled by the irf option, as specified in chapter 3 when discussing the options for `stoch_simul`. To build the posterior distribution of the IRFs, Dynare pulls parameter and shock values from the corresponding estimated distributions and, for each set of draws, generates an IRF. Repeating this process often enough generates a distribution of IRFs. **TIP!** If you stop the estimation procedure after calculating the posterior mode, or carry out maximum likelihood estimation, only the corresponding parameter estimates will be used to generate the IRFs. If you instead carry out a full Metropolis-Hastings estimation, on the other hand, the IRFs will use the parameters the posterior distributions, including the variance of the shocks.

18. All options available for stoch_simul can simply be added to the above options, separated by commas. To view a list of these options, either see the Reference Manual or section 3.8 of chapter 3.

19. moments_varendo: triggers the computation of the posterior distribution of the theoretical moments of the endogenous variables as in `stoch_simul` (the posterior distribution of the variance decomposition is also included). ** will be implemented shortly - if not already - in Dynare version 4.

20. filtered_vars: triggers the computation of the posterior distribution of filtered endogenous variables and shocks. See the note below on the difference between filtered and smoothed shocks. ** will be implemented shortly - if not already - in Dynare version 4.

21. smoother: triggers the computation of the posterior distribution of smoothed endogenous variables and shocks. Smoothed shocks are a reconstruction of the values of unobserved shocks over the sample, using all the information contained in the sample of observations. Filtered shocks, instead, are built only based on knowing past information. To calculate one period ahead prediction errors, for instance, you should use filtered, not smoothed variables.

22. forecast = INTEGER: computes the posterior distribution of a forecast on INTEGER periods after the end of the sample used in estimation. The corresponding graph includes one confidence interval describing uncertainty due to parameters and one confidence interval describing uncertainty due to parameters and future shocks. Note that Dynare cannot forecast out of the posterior mode. You need to run Metropolis-Hastings iterations before being able to run forecasts on an estimated model. Finally, running a forecast is very similar to an IRF, as in `bayesian_irf`, except that the forecast does not begin at a steady state, but simply at the point corresponding to the last set of observations. The goal of undertaking a forecast is to see how the system returns to steady state from this starting point. Of course, as observation do not exist for all variables, those necessary are reconstructed by sampling out of the posterior distribution of parameters. Again, repeating this step often enough yields a posterior distribution of the forecast. ** will be implemented shortly - if not already - in Dynare version 4.

TIP! Before launching estimation it is a good idea to make sure that your model is correctly declared, that a steady state exists and that it can be simulated for at least one set of parameter values. You may therefore want to create a test version of your .mod file. In this test file, you would comment out or erase the commands related to estimation, remove the prior estimates for parameter values and replace them with actual parameter values in the preamble, remove any non-stationary variables from your model, add a `shocks` block, make sure you have `steady` and possibly `check` following the `initval` block if you do not have exact steady state values and run a simulation using `stoch_simul` at the end of your .mod file. Details on model solution and simulation can be found in Chapter 3.

Finally, coming back to our example, we could choose a standard option:

```
estimation(datafile=simuldataRBC,nobs=200,first_obs=500,
mh_replic=2000,mh_nblocks=2,mh_drop=0.45,mh_jscale=0.8);
```

This ends our description of the .mod file.

5.8 The complete .mod file

To summarize and to get a complete perspective on our work so far, here is the complete .mod file for the estimation of our very basic model. You can find the corresponding file in the *models* folder under *UserGuide* in your installation of Dynare. The file is called *RBC_Est.mod*.

```
var y c k i l y_l w r z;
```

```
varexo e;

parameters beta psi delta alpha rho epsilon;

model;
(1/c) = beta*(1/c(+1))*(1+r(+1)-delta);
psi*c/(1-l) = w;
c+i = y;
y = (k(-1)^alpha)*(exp(z)*l)^(1-alpha);
w = y*((epsilon-1)/epsilon)*(1-alpha)/l;
r = y*((epsilon-1)/epsilon)*alpha/k(-1);
i = k-(1-delta)*k(-1);
y_l = y/l;
z = rho*z(-1)+e;
end;

varobs Y;

initval;
k = 9;
c = 0.7;
l = 0.3;
w = 2.0;
r = 0;
z = 0;
e = 0;
end;

steady;

check;

estimated_params;
alpha, beta_pdf, 0.35, 0.02;
beta, beta_pdf, 0.99, 0.002;
delta, beta_pdf, 0.025, 0.003;
psi, gamma_pdf, 1.75, 0.02;
rho, beta_pdf, 0.95, 0.05;
epsilon, gamma_pdf, 10, 0.003;
stderr e, inv_gamma_pdf, 0.01, inf;
end;

estimation(datafile=simuldataRBC,nobs=200,first_obs=500,
mh_replic=2000,mh_nblocks=2,mh_drop=0.45,mh_jscale=0.8);
```

5.9 Interpreting output

As in the case of model solution and simulation, Dynare returns both tabular and graphical output. On the basis of the options entered in the example .mod file above, Dynare will display the following results.

5.9.1 Tabular results

The first results to be displayed (and calculated from a chronological standpoint) are the steady state results. Note the dummy values of 1 for the non-stationary variables Y_obs and P_obs. These results are followed by the eigenvalues of the system, presented in the order in which the endogenous variables are declared at the beginning of the .mod file. The table of eigenvalues is completed with a statement about the Blanchard-Kahn condition being met - hopefully!

The next set of results are for the numerical iterations necessary to find the posterior mode, as explained in more details in Chapter 6. The improvement from one iteration to the next reaches zero, Dynare give the value of the objective function (the posterior Kernel) at the mode and displays two important table summarizing results from posterior maximization.

The first table summarizes results for parameter values. It includes: prior means, posterior mode, standard deviation and t-stat of the mode (based on the assumption of a Normal, probably erroneous when undertaking Bayesian estimation, as opposed to standard maximum likelihood), as well as the prior distribution and standard deviation (pstdev). It is followed by a second table summarizing the same results for the shocks. It may be entirely possible that you get an infinite value for a standard deviation, this is simply the limit case of the inverse gamma distribution.

5.9.2 Graphical results

** corresponding graphs will be reproduced below.

The first figure comes up soon after launching Dynare as little computation is necessary to generate it. The figure returns a graphical representation of the priors for each parameter of interest.

The second set of figures displays "MCMC univariate diagnostics", where MCMC stands for Monte Carlo Markov Chains. This is the main source of feedback to gain confidence, or spot a problem, with results. Recall that Dynare completes several runs of Metropolis-Hastings simulations (as many as determined in the option `mh_nblocks`, each time starting from a different initial value). If the results from one chain are sensible, and the optimizer did not get stuck in an odd area of the parameter subspace, two things should happen. First, results within any of the however many iterations of Metropolis-Hastings simulation should be similar. And second, results between the various chains should be close. This is the idea of what the MCMC diagnostics track.

More specifically, the red and blue lines on the charts represent specific measures of the parameter vectors both within and between chains. For the results to be sensible, these should be relatively constant (although there will always be some variation) and they should converge. Dynare reports three measures: "interval", being constructed from an 80% confidence interval around the parameter mean, "m2", being a measure of the variance and "m3" based on third moments. In each case, Dynare reports both the within and the between chains measures. The figure entitled "multivariate diagnostic" presents results of the same nature, except that they reflect an aggregate measure based on the eigenvalues of the variance-covariance matrix of each parameter.

In our example above, you can tell that indeed, we obtain convergence and relative stability in all measures of the parameter moments. Note that the horizontal axis represents the number of Metropolis-Hastings iterations that have been undertaken, and the vertical axis the measure of the parameter moments, with the first, corresponding to the measure at the initial value of the Metropolis-Hastings iterations.

TIP! If the plotted moments are highly unstable or do not converge, you may have a problem of poor priors. It is advisable to redo the estimation with different priors. If you have trouble coming up with a new prior, try starting with a uniform and relatively wide prior and see where the data leads the posterior distribution. Another approach is to undertake a greater number of Metropolis-Hastings simulations.

The first to last figure - figure 6 in our example - displays the most interesting set of results, towards which most of the computations undertaken by Dynare are directed: the posterior distribution. In fact, the figure compares the posterior to the prior distribution (black vs. grey lines). In addition, on the posterior distribution, Dynare plots a green line which represents the posterior mode. These allow you to make statements about your data other than simply concerning the mean and variance of the parameters; you can also

discuss the probability that your parameter is larger or smaller than a certain value.

TIP! These graphs are of course especially relevant and present key results, but they can also serve as tools to detect problems or build additional confidence in your results. First, the prior and the posterior distributions should not be excessively different. Second, the posterior distributions should be close to normal, or at least not display a shape that is clearly non-normal. Third, the green mode (calculated from the numerical optimization of the posterior kernel) should not be too far away from the mode of the posterior distribution. If not, it is advisable to undertake a greater number of Metropolis-Hastings simulations.

The last figure returns the smoothed estimated shocks in a useful illustration to eye-ball the plausibility of the size and frequency of the shocks. The horizontal axis, in this case, represents the number of periods in the sample. One thing to check is the fact that shocks should be centered around zero. That is indeed the case for our example.

Chapter 6

Estimating DSGE models - advanced topics

This chapter focusses on advanced topics and features of Dynare in the area of model estimation. The chapter begins by presenting a more complex example than the one used for illustration purposes in chapter 5. The goal is to show how Dynare would be used in the more "realistic" setting of reproducing a recent academic paper. The chapter then follows with sections on comparing models to one another, and then to BVARs, and ends with a table summarizing where output series are stored and how these can be retrieved.

6.1 Alternative and non-stationary example

The example provided in chapter 5 is really only useful for illustration purposes. So we thought you would enjoy (and continue learning from!) a more realistic example which reproduces the work in a recent - and highly regarded - academic paper. The example shows how to use Dynare in a more realistic setting, while emphasizing techniques to deal with non-stationary observations and stochastic trends in dynamics.

6.1.1 Introducing the example

The example is drawn from Schorfheide (2000). This first section introduces the model, its basic intuitions and equations. We will then see in subsequent sections how to estimate it using Dynare. Note that the original paper by Schorfheide mainly focusses on estimation methodologies, difficulties and solutions, with a special interest in model comparison, while the mathematics and economic intuitions of the model it evaluates are drawn from Nason and Cogley (1994). That paper should serve as a helpful reference if anything is

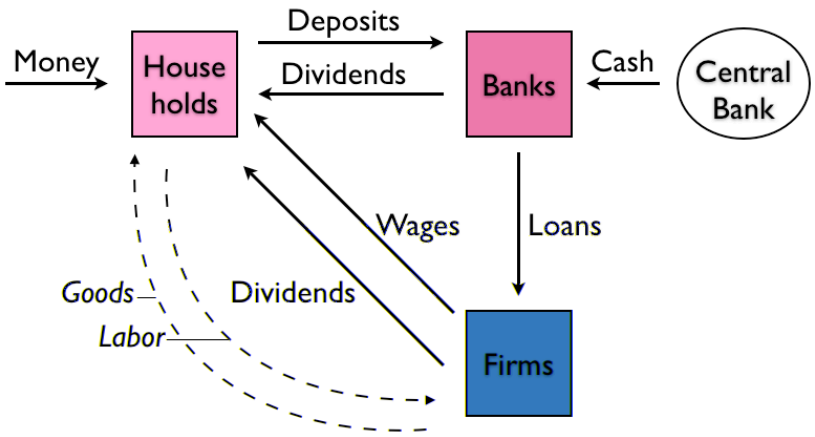

Figure 6.1: Continuous lines show the circulation of nominal funds, while dashed lines show the flow of real variables.

left unclear in the description below.

In essence, the model studied by Schorfheide (2000) is one of cash in advance (CIA). The goal of the paper is to estimate the model using Bayesian techniques, while observing only output and inflation. In the model, there are several markets and actors to keep track of. So to clarify things, figure 6.1.1 sketches the main dynamics of the model. You may want to refer back to the figure as you read through the following sections.

The economy is made up of three central agents and one secondary agent: households, firms and banks (representing the financial sector), and a monetary authority which plays a minor role. Households maximize their utility function which depends on consumption, C_t, and hours worked, H_t, while deciding how much money to hold next period in cash, M_{t+1} and how much to deposit at the bank, D_t, in order to earn $R_{H,t} - 1$ interest. Households

therefore solve the problem

$$\begin{aligned}\max_{\{C_t,H_t,M_{t+1},D_t\}} \quad & \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^t\left[(1-\phi)\ln C_t+\phi\ln(1-H_t)\right]\right]\\ \text{s.t.} \quad & P_tC_t \le M_t - D_t + W_tH_t\\ & 0 \le D_t\\ & M_{t+1} = (M_t - D_t + W_tH_t - P_tC_t) + R_{H,t}D_t + F_t + B_t\end{aligned}$$

where the second equation spells out the cash in advance constraint including wage revenues, the third the inability to borrow from the bank and the fourth the intertemporal budget constraint emphasizing that households accumulate the money that remains after bank deposits and purchases on goods are deducted from total inflows made up of the money they receive from last period's cash balances, wages, interests, as well as dividends from firms, F_t, and from banks, B_t, which in both cases are made up of net cash inflows defined below.

Banks, on their end, receive cash deposits from households and a cash injection, X_t from the central bank (which equals the net change in nominal money balances, $M_{t+1} - M_t$). It uses these funds to disburse loans to firms, L_t, on which they make a net return of $R_{F,t} - 1$. Of course, banks are constrained in their loans by a credit market equilibrium condition $L_t \le X_t + D_t$. Finally, bank dividends, B_t are simply equal to $D_t + R_{F,t}L_t - R_{H,t}D_t - L_t + X_t$.

Finally, firms maximize the net present value of future dividends (discounted by the marginal utility of consumption, since they are owned by households) by choosing dividends, next period's capital stock, K_{t+1}, labor demand, N_t, and loans. Its problem is summarized by

$$\begin{aligned}\max_{\{F_t,K_{t+1},N_t,L_t\}} \quad & \mathbb{E}_0\left[\sum_{t=0}^{\infty}\beta^{t+1}\frac{F_t}{C_{t+1}P_{t+1}}\right]\\ \text{s.t.} \quad & F_t \le L_t + P_t\left[K_t^{\alpha}(A_tN_t)^{1-\alpha} - K_{t+1} + (1-\delta)K_t\right] - W_tN_t - L_tR_{F,t}\\ & W_tN_t \le L_t\end{aligned}$$

where the second equation makes use of the production function $Y_t = K_t^{\alpha}(A_tN_t)^{1-\alpha}$ and the real aggregate accounting constraint (goods market equilibrium) $C_t + I_t = Y_t$, where $I_t = K_{t+1} - (1-\delta)K_t$, and where δ is the rate of depreciation. Note that it is the firms that engage in investment in this model, by trading off investment for dividends to consumers. The third equation simply specifies that bank loans are used to pay for wage costs.

To close the model, we add the usual labor and money market equilibrium equations, $H_t = N_t$ and $P_tC_t = M_t + X_t$, as well as the condition that $R_{H,t} = R_{F,t}$ due to the equal risk profiles of the loans.

More importantly, we add a stochastic elements to the model. The model allows for two sources of perturbations, one real, affecting technology and one nominal, affecting the money stock. These important equations are

$$\ln A_t = \gamma + \ln A_{t-1} + \epsilon_{A,t}, \qquad \epsilon_{A,t} \sim N(0, \sigma_A^2)$$

and

$$\ln m_t = (1-\rho)\ln m^* + \rho \ln m_{t-1} + \epsilon_{M,t}, \qquad \epsilon_{M,t} \sim N(0, \sigma_M^2)$$

where $m_t \equiv M_{T+1}/M_t$ is the growth rate of the money stock. Note that theses expressions for trends are not written in the most straightforward manner nor very consistently. But we reproduced them never-the-less to make it easier to compare this example to the original paper.

The first equation is therefore a unit root with drift in the log of technology, and the second an autoregressive stationary process in the growth rate of money, but an AR(2) with a unit root in the log of the level of money. This can be seen from the definition of m_t which can be rewritten as $\ln M_{t+1} = \ln M_t + \ln m_t$.[1]

When the above functions are maximized, we obtain the following set of first order and equilibrium conditions. We will not dwell on the derivations here, to save space, but encourage you to browse Nason and Cogley (1994) for additional details. We nonetheless give a brief intuitive explanation of each

[1] Alternatively, we could have written the AR(2) process in state space form and realized that the system has an eigenvalue of one. Otherwise said, one is a root of the second order autoregressive lag polynomial. As usual, if the logs of a variable are specified to follow a unit root process, the rate of growth of the series is a stationary stochastic process; see Hamilton (1994), chapter 15, for details.

equation. The system comes down to

$$
\begin{aligned}
\mathbb{E}_t\Big\{ -\widehat{P}_t/\left[\widehat{C}_{t+1}\widehat{P}_{t+1}m_t\right]\Big\} &= \beta e^{-\alpha(\gamma+\epsilon_{A,t+1})}P_{t+1}\Big[\alpha\widehat{K}_t^{\alpha-1}N_{t+1}^{1-\alpha}+(1-\delta)\Big] \\
&\quad /\left[\widehat{c}_{t+2}\widehat{P}_{t+2}m_{t+1}\right]\Big\} \\
\widehat{W}_t &= \widehat{L}_t/N_t \\
\frac{\phi}{1-\phi}\left[\widehat{C}_t\widehat{P}_t/\left(1-N_t\right)\right] &= \widehat{L}_t/N_t \\
R_t &= (1-\alpha)\widehat{P}_t e^{-\alpha(\gamma+\epsilon_{A,t+1})}\widehat{K}_{t-1}^{\alpha}N_t^{-\alpha}/\widehat{W}_t \\
\left[\widehat{C}_t\widehat{P}_t\right]^{-1} &= \beta\left[\left(1-\alpha\right)\widehat{P}_t e^{-\alpha(\gamma+\epsilon_{A,t+1})}\widehat{K}_{t-1}^{\alpha}N_t^{1-\alpha}\right] \\
&\quad \times\mathbb{E}_t\left[\widehat{L}_t m_t\widehat{C}_{t+1}\widehat{P}_{t+1}\right]^{-1} \\
\widehat{C}_t+\widehat{K}_t &= e^{-\alpha(\gamma+\epsilon_{A,t})}\widehat{K}_{t-1}^{\alpha}N^{1-\alpha}+(1-\delta)e^{-(\gamma+\epsilon_{A,t})}\widehat{K}_{t-1} \\
\widehat{P}_t\widehat{C} &= m_t \\
m_t-1+\widehat{D}_t &= \widehat{L}_t \\
\widehat{Y}_t &= \widehat{K}_{t-1}^{\alpha}N^{1-\alpha}e^{-\alpha(\gamma+\epsilon_{A,t})} \\
\ln(m_t) &= (1-\rho)\ln(m^{\star})+\rho\ln(m_{t-1})+\epsilon_{M,t} \\
\frac{A_t}{A_{t-1}}\equiv dA_t &= \exp(\gamma+\epsilon_{A,t}) \\
Y_t/Y_{t-1} &= e^{\gamma+\epsilon_{A,t}}\widehat{Y}_t/\widehat{Y}_{t-1} \\
P_t/P_{t-1} &= (\widehat{P}_t/\widehat{P}_{t-1})(m_{t-1}/e^{\gamma+\epsilon_{A,t}})
\end{aligned}
$$

where, importantly, hats over variables no longer mean deviations from steady state, but instead represent variables that have been made stationary. We come back to this important topic in details in section 6.1.3 below. For now, we pause a moment to give some intuition for the above equations. In order, these equations correspond to:

1. The Euler equation in the goods market, representing the tradeoff to the economy of moving consumption goods across time.

2. The firms' borrowing constraint, also affecting labor demand, as firms use borrowed funds to pay for labor input.

3. The intertemporal labor market optimality condition, linking labor supply, labor demand, and the marginal rate of substitution between consumption and leisure.

4. The equilibrium interest rate in which the marginal revenue product of labor equals the cost of borrowing to pay for that additional unit of labor.

5. The Euler equation in the credit market, which ensures that giving up one unit of consumption today for additional savings equals the net present value of future consumption.

6. The aggregate resource constraint.

7. The money market equilibrium condition equating nominal consumption demand to money demand to money supply to current nominal balances plus money injection.

8. The credit market equilibrium condition.

9. The production function.

10. The stochastic process for money growth.

11. The stochastic process for technology.

12. The relationship between observable variables and stationary variables; more details on these last two equations appear in the following section.

6.1.2 Declaring variables and parameters

This block of the .mod file follows the usual conventions and would look like:

```
var m P c e W R k d n l Y_obs P_obs y dA;
varexo e_a e_m;

parameters alp, bet, gam, mst, rho, psi, del;
```

where the choice of upper and lower case letters is not significant, the first set of endogenous variables, up to l, are as specified in the model setup above, and where the last five variables are defined and explained in more details in the section below on declaring the model in Dynare. The exogenous variables are as expected and concern the shocks to the evolution of technology and money balances.

6.1.3 The origin of non-stationarity

The problem of non-stationarity comes from having stochastic trends in technology and money. The non-stationarity comes out clearly when attempting to solve the model for a steady state and realizing it does not have one. It can be shown that when shocks are null, real variables grow with A_t (except for labor, N_t, which is stationary as there is no population growth), nominal variables grow with M_t and prices with M_t/A_t. **Detrending** therefore involves

the following operations (where hats over variables represent stationary variables): for real variables, $\hat{q}_t = q_t/A_t$, where $q_t = [y_t, c_t, i_t, k_{t+1}]$. For nominal variables, $\hat{Q}_t = Q_t/M_t$, where $Q_t = [d_t, l_t, W_t]$. And for prices, $\hat{P}_t = P_t \cdot A_t/M_t$.

6.1.4 Stationarizing variables

Let's illustrate this transformation on output, and leave the transformations of the remaining equations as an exercise, if you wish (Nason and Cogley (1994) includes more details on the transformations of each equation). We stationarize output by dividing its real variables (except for labor) by A_t. We define $\widehat{Y}_t$ to equal Y_t/A_t and $\widehat{K}_t$ as K_t/A_t. **NOTE!** Recall from section 3.5 in chapter 3), that in Dynare variables take the time subscript of the period in which they are decided (in the case of the capital stock, today's capital stock is a result of yesterday's decision). Thus, in the output equation, we should actually work with $\widehat{K}_{t-1} = K_{t-1}/A_{t-1}$. The resulting equation made up of stationary variables is

$$\begin{aligned}
\frac{Y_t}{A_t} &= \left(\frac{K_{t-1}}{A_{t-1}}\right)^{\alpha} A_t^{1-\alpha} N_t^{1-\alpha} A_t^{-1} A_{t-1}^{\alpha} \\
\widehat{Y}_t &= \widehat{K}_{t-1}^{\alpha} N_t^{1-\alpha} \left(\frac{A_t}{A_{t-1}}\right)^{-\alpha} \\
&= \widehat{K}_{t-1}^{\alpha} N_t^{1-\alpha} \exp(-\alpha(\gamma + \epsilon_{A,t}))
\end{aligned}$$

where we go from the second to the third line by taking the exponential of both sides of the equation of motion of technology.

The above is the equation we retain for the .mod file of Dynare into which we enter:

```
y=k(-1)^alp*n^(1-alp)*exp(-alp*(gam+e_a))
```

The other equations are entered into the .mod file after transforming them in exactly the same way as the one above. A final transformation to consider, that turns out to be useful since we often deal with the growth rate of technology, is to define

```
dA = exp(gam+e_a)
```

by simply taking the exponential of both sides of the stochastic process of technology defined in the model setup above.

6.1.5 Linking stationary variables to the data

And finally, we must make a decision as to our **non-stationary observations**. We could simply stationarize them by **working with rates of growth** (which we know are constant). In the case of output, the observable variable would become Y_t/Y_{t-1}. We would then have to relate this observable, call it *gy_obs*, to our (stationary) model's variables $\widehat{Y}_t$ by using the definition that $\widehat{Y}_t \equiv Y_t/A_t$. Thus, we add to the model block of the .mod file:

```
gy_obs = dA*y/y(-1);
```

where, the y of the .mod file are the stationary $\widehat{Y}_t$.

But, we could also **work with non-stationary data in levels**. This complicates things somewhat, but illustrates several features of Dynare worth highlighting; we therefore follow this path in the remainder of the example. The result is not very different, though, from what we just saw above. The goal is to add a line to the model block of our .mod file that relates the non stationary observables, call them Y_{obs}, to our stationary output, $\widehat{Y}_t$. We could simply write $Y_{obs} = \widehat{Y}_t A_t$. But since we don't have an A_t variable, but just a dA_t, we we-write the above relationship in ratios. To the .mod file, we therefore add:

```
Y_obs/Y_obs(-1) = dA*y/y(-1);
```

We of course do the same for prices, our other observable variable, except that we use the relationship $P_{obs} = \widehat{P}_t M_t/A_t$ as noted earlier. The details of the correct transformations for prices are left as an exercise and can be checked against the results below.

6.1.6 The resulting model block of the .mod file

```
model;
dA = exp(gam+e_a);
log(m) = (1-rho)*log(mst) + rho*log(m(-1))+e_m;
-P/(c(+1)*P(+1)*m)+bet*P(+1)*(alp*exp(-alp*(gam+log(e(+1))))*k^(alp-1)
*n(+1)^(1-alp)+(1-del)*exp(-(gam+log(e(+1)))))/(c(+2)*P(+2)*m(+1))=0;
W = l/n;
-(psi/(1-psi))*(c*P/(1-n))+l/n = 0;
R = P*(1-alp)*exp(-alp*(gam+e_a))*k(-1)^alp*n^(-alp)/W;
1/(c*P)-bet*P*(1-alp)*exp(-alp*(gam+e_a))*k(-1)^alp*n^(1-alp)/
(m*l*c(+1)*P(+1)) = 0;
c+k = exp(-alp*(gam+e_a))*k(-1)^alp*n^(1-alp)+(1-del)
```

```
*exp(-(gam+e_a))*k(-1);
P*c = m;
m-1+d = l;
e = exp(e_a);
y = k(-1)^alp*n^(1-alp)*exp(-alp*(gam+e_a));
Y_obs/Y_obs(-1) = dA*y/y(-1);
P_obs/P_obs(-1) = (p/p(-1))*m(-1)/dA;
end;
```

where, of course, the input conventions, such as ending lines with semicolons and indicating the timing of variables in parentheses, are the same as those listed in chapter 3.

TIP! In the above model block, notice that what we have done is in fact relegated the non-stationarity of the model to just the last two equations, concerning the observables which are, after all, non-stationary. The problem that arises, though, is that we cannot linearize the above system in levels, as the last two equations don't have a steady state. If we first take logs, though, they become linear and it doesn't matter anymore where we calculate their derivative when taking a Taylor expansion of all the equations in the system. Thus, **when dealing with non-stationary observations, you must log-linearize your model** (and not just linearize it); this is a point to which we will return later.

6.1.7 Declaring observable variables

We begin by declaring which of our model's variables are observables. In our .mod file we write

```
varobs P_obs Y_obs;
```

to specify that our observable variables are indeed *P_obs* and *Y_obs* as noted in the section above. **NOTE!** Recall from earlier that the number of observed variables must be smaller or equal to the number of shocks such that the model be estimated. If this is not the case, you should add measurement shocks to your model where you deem most appropriate.

6.1.8 Declaring trends in observable variables

Recall that we decided to work with the non-stationary observable variables in levels. Both output and prices exhibit stochastic trends. This can be seen explicitly by taking the difference of logs of output and prices to compute growth rates. In the case of output, we make use of the usual (by now!)

relationship $Y_t = \widehat{Y}_t \cdot A_t$. Taking logs of both sides and subtracting the same equation scrolled back one period, we find:

$$\Delta \ln Y_t = \Delta \ln \widehat{Y}_t + \gamma + \epsilon_{A,t}$$

emphasizing clearly the drift term γ, whereas we know $\Delta \ln \widehat{Y}_t$ is stationary in steady state.

In the case of prices, we apply the same manipulations to show that:

$$\Delta \ln P_t = \Delta \ln \widehat{P}_t + \ln m_{t-1} - \gamma - \epsilon_{A,t}$$

Note from the original equation of motion of $\ln m_t$ that in steady state, $\ln m_t = \ln m^*$, so that the drift terms in the above equation are $\ln m^* - \gamma$.[2]

In Dynare, any trends, whether deterministic or stochastic (the drift term) must be declared up front. In the case of our example, we therefore write (in a somewhat cumbersome manner)

```
observation_trends;
P_obs (log(mst)-gam);
Y_obs (gam);
end;
```

In general, the command `observation_trends` specifies linear trends as a function of model parameters for the observed variables in the model.

6.1.9 Declaring unit roots in observable variables

And finally, since P_obs and Y_obs inherit the unit root characteristics of their driving variables, technology and money, we must tell Dynare to use a diffuse prior (infinite variance) for their initialization in the Kalman filter. Note that 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). To give these instructions to Dynare, we write in the .mod

```
unit_root_vars P_obs Y_obs;
```

NOTE! You don't need to declare unit roots for any non-stationary model. Unit roots are only related to stochastic trends. You don't need to use a diffuse

[2] This can also be see from substituting for $\ln m_{t-1}$ in the above equation with the equation of motion of $\ln m_t$ to yield: $\Delta \ln P_t = \Delta \ln \widehat{P}_t + \ln m^* + \rho(\ln m_{t-2} - \ln m^*) + \epsilon_{M,t} - \gamma - \epsilon_{A,t}$ where all terms on the right hand side are constant, except for $\ln m^*$ and γ.

initial condition in the case of a deterministic trend, since the variance is finite.

6.1.10 Specifying the steady state

Declaring the steady state is just as explained in details and according to the same syntax explained in chapter 3, covering the `initval`, `steady` and `check` commands. In chapter 5, section 5.5, we also discussed the usefulness of providing an external Matlab file to solve for the steady state. In this case, you can find the corresponding steady state file in the *models* folder under *UserGuide*. The file is called *fs2000ns_steadystate.m*. There are some things to notice. First, the output of the function is the endogenous variables at steady state, the `ys` vector. The `check=0` limits steady state values to real numbers. Second, notice the declaration of parameters at the beginning; intuitive, but tedious... This functionality may be updated in later versions of Dynare. Third, note that the file is really only a sequential set of equalities, defining each variable in terms of parameters or variables solved in the lines above. So far, nothing has changed with respect to the equivalent file of chapter 5. The only novelty is the declaration of the non-stationary variables, *P_obs* and *Y_obs* which take the value of 1. This is Dynare convention and must be the case for all your non-stationary variables.

6.1.11 Declaring priors

We expand our .mod file with the following information:

```
estimated_params;
alp, beta_pdf, 0.356, 0.02;
bet, beta_pdf, 0.993, 0.002;
gam, normal_pdf, 0.0085, 0.003;
mst, normal_pdf, 1.0002, 0.007;
rho, beta_pdf, 0.129, 0.223;
psi, beta_pdf, 0.65, 0.05;
del, beta_pdf, 0.01, 0.005;
stderr e_a, inv_gamma_pdf, 0.035449, inf;
stderr e_m, inv_gamma_pdf, 0.008862, inf;
end;
```

6.1.12 Launching the estimation

We add the following commands to ask Dynare to run a basic estimation of our model:

```
estimation(datafile=fsdat,nobs=192,loglinear,mh_replic=2000,
mode_compute=4,mh_nblocks=2,mh_drop=0.45,mh_jscale=0.65);
```

NOTE! As mentioned earlier, we need to instruct Dynare to log-linearize our model, since it contains non-linear equations in non-stationary variables. A simple linearization would fail as these variables do not have a steady state. Fortunately, taking the log of the equations involving non-stationary variables does the job of linearizing them.

6.1.13 The complete .mod file

We have seen each part of the .mod separately; it's now time to get a picture of what the complete file looks like. For convenience, the file also appears in the *models* folder under *UserGuide* in your Dynare installation. The file is called `fs2000ns.mod`.

```
var m P c e W R k d n l Y_obs P_obs y dA;
varexo e_a e_m;

parameters alp, bet, gam, mst, rho, psi, del;
model;
dA = exp(gam+e_a);
log(m) = (1-rho)*log(mst) + rho*log(m(-1))+e_m;
-P/(c(+1)*P(+1)*m)+bet*P(+1)*(alp*exp(-alp*(gam+log(e(+1))))*k^(alp-1)
*n(+1)^(1-alp)+(1-del)*exp(-(gam+log(e(+1)))))/(c(+2)*P(+2)*m(+1))=0;
W = l/n;
-(psi/(1-psi))*(c*P/(1-n))+l/n = 0;
R = P*(1-alp)*exp(-alp*(gam+e_a))*k(-1)^alp*n^(-alp)/W;
1/(c*P)-bet*P*(1-alp)*exp(-alp*(gam+e_a))*k(-1)^alp*n^(1-alp)/(m*l*c(+1)*P(+1))
= 0;
c+k = exp(-alp*(gam+e_a))*k(-1)^alp*n^(1-alp)+(1-del)*exp(-(gam+e_a))*k(-1);
P*c = m;
m-1+d = l;
e = exp(e_a);
y = k(-1)^alp*n^(1-alp)*exp(-alp*(gam+e_a));
Y_obs/Y_obs(-1) = dA*y/y(-1);
P_obs/P_obs(-1) = (p/p(-1))*m(-1)/dA;
end;

varobs P_obs Y_obs;

observation_trends;
```

```
P_obs (log(mst)-gam);
Y_obs (gam);
end;

unit_root_vars = P_obs Y_obs;

initval;
k = 6;
m = mst;
P = 2.25;
c = 0.45;
e = 1;
W = 4;
R = 1.02;
d = 0.85;
n = 0.19;
l = 0.86;
y = 0.6;
dA = exp(gam);
end;

// the above is really only useful if you want to do a stoch_simul
// of your model, since the estimation will use the Matlab
// steady state file also provided and discussed above.

steady;

estimated_params;
alp, beta_pdf, 0.356, 0.02;
bet, beta_pdf, 0.993, 0.002;
gam, normal_pdf, 0.0085, 0.003;
mst, normal_pdf, 1.0002, 0.007;
rho, beta_pdf, 0.129, 0.223;
psi, beta_pdf, 0.65, 0.05;
del, beta_pdf, 0.01, 0.005;
stderr e_a, inv_gamma_pdf, 0.035449, inf;
stderr e_m, inv_gamma_pdf, 0.008862, inf;
end;

estimation(datafile=fsdat,nobs=192,loglinear,mh_replic=2000,
mode_compute=4,mh_nblocks=2,mh_drop=0.45,mh_jscale=0.65);
```

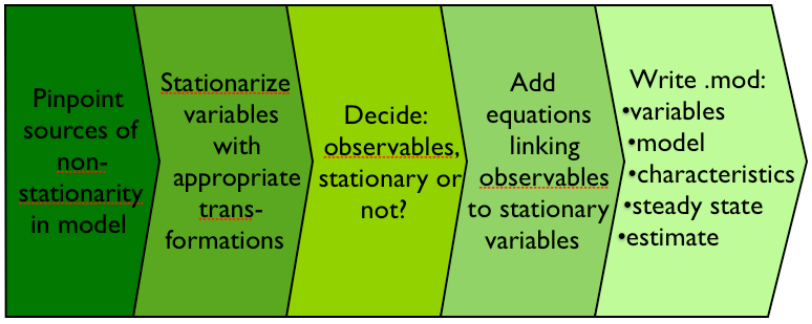

Figure 6.2: At a high level, there are five basic steps to translate a model into Dynare for successful estimation.

6.1.14 Summing it up

The explanations given above of each step necessary to translate the Schorfheide (2000) example into language that Dynare can understand and process was quite lengthy and involved a slew of new commands and information. It may therefore be useful, to gain a "bird's eyeview" on what we have just accomplished, and summarize the most important steps at a high level. This is done in figure 6.1.14.

6.2 Comparing models based on their posterior distributions

** TBD

6.3 Where is your output stored?

The output from estimation can be extremely varied, depending on the instructions you give Dynare. The Reference Manual overviews the complete set of potential output files and describes where you can find each one.

Chapter 7

Solving DSGE models - Behind the scenes of Dynare

7.1 Introduction

The aim of this chapter is to peer behind the scenes of Dynare, or under its hood, to get an idea of the methodologies and algorithms used in its computations. Going into details would be beyond the scope of this User Guide which will instead remain at a high level. What you will find below will either comfort you in realizing that Dynare does what you expected of it - and what you would have also done if you had had to code it all yourself (with a little extra time on your hands!), or will spur your curiosity to have a look at more detailed material. If so, you may want to go through Michel Juillard's presentation on solving DSGE models to a first and second order (available on Michel Juillard's website), or read Collard and Juillard (2001a) or Schmitt-Grohe and Uribe (2004) which gives a good overview of the most recent solution techniques based on perturbation methods. Finally, note that in this chapter we will focus on stochastic models - which is where the major complication lies, as explained in section 3.1.1 of chapter 3. For more details on the Newton-Raphson algorithm used in Dynare to solve deterministic models, see Juillard (1996).

7.2 What is the advantage of a second order

unconditional expectations of the endogenous variables are equal to their non-stochastic steady state values.

This may be an acceptable simplification to make. But depending on the context, it may instead be quite misleading. For instance, when using second order welfare functions to compare policies, you also need second order approximations of the policy function. Yet more clearly, in the case of asset pricing models, linearizing to the second order enables you to take risk (or the variance of shocks) into consideration - a highly desirable modeling feature. It is therefore very convenient that Dynare allows you to choose between a first or second order linearization of your model in the option of the `stoch_simul` command.

7.3 How does dynare solve stochastic DSGE models?

In this section, we shall briefly overview the perturbation methods employed by Dynare to solve DSGE models to a first order approximation. The second order follows very much the same approach, although at a higher level of complexity. The summary below is taken mainly from Michel Juillard's presentation "Computing first order approximations of DSGE models with Dynare", which you should read if interested in particular details, especially regarding second order approximations (available on Michel Juillard's website).

To summarize, a DSGE model is a collection of first order and equilibrium conditions that take the general form:

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

$$\begin{aligned} \mathbb{E}(u_t) &= 0 \\ \mathbb{E}(u_t u_t') &= \Sigma_u \end{aligned}$$

and where:

y : vector of endogenous variables of any dimension

u : vector of exogenous stochastic shocks of any dimension

The solution to this system is a set of equations relating variables in the current period to the past state of the system and current shocks, that satisfy the original system. This is what we call the policy function. Sticking to the above notation, we can write this function as:

$$y_t = g(y_{t-1}, u_t)$$

Then, it is straightforward to re-write y_{t+1} as

$$\begin{aligned} y_{t+1} &= g(y_t, u_{t+1}) \\ &= g(g(y_{t-1}, u_t), u_{t+1}) \end{aligned}$$

We can then define a new function F, such that:

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

which enables us to rewrite our system in terms of past variables, and current and future shocks:

$$\mathbb{E}_t \left[F(y_{t-1}, u_t, u_{t+1}) \right] = 0$$

We then venture to linearize this model around a steady state defined as:

$$f(\bar{y}, \bar{y}, \bar{y}, 0) = 0$$

having the property that:

$$\bar{y} = g(\bar{y}, 0)$$

The first order Taylor expansion around $\bar{y}$ yields:

$$\begin{aligned} &\mathbb{E}_t \left\{ F^{(1)}(y_{t-1}, u_t, u_{t+1}) \right\} = \\ &\quad \mathbb{E}_t \Big[f(\bar{y}, \bar{y}, \bar{y}) + f_{y_+} \left(g_y \left(g_y \hat{y} + g_u u \right) + g_u u' \right) \\ &\quad + f_{y_0} \left(g_y \hat{y} + g_u u \right) + f_{y_-} \hat{y} + f_u u \Big] \\ &\quad = 0 \end{aligned}$$

with $\hat{y} = y_{t-1} - \bar{y}$, $u = u_t$, $u' = u_{t+1}$, $f_{y_+} = \frac{\partial f}{\partial y_{t+1}}$, $f_{y_0} = \frac{\partial f}{\partial y_t}$, $f_{y_-} = \frac{\partial f}{\partial y_{t-1}}$, $f_u = \frac{\partial f}{\partial u_t}$, $g_y = \frac{\partial g}{\partial y_{t-1}}$, $g_u = \frac{\partial g}{\partial u_t}$.

Taking expectations (we're almost there!):

$$\begin{aligned} &\mathbb{E}_t \left\{ F^{(1)}(y_{t-1}, u_t, u_{t+1}) \right\} = \\ &\quad f(\bar{y}, \bar{y}, \bar{y}) + f_{y_+} \left(g_y \left(g_y \hat{y} + g_u u \right) \right) \\ &\quad + f_{y_0} \left(g_y \hat{y} + g_u u \right) + f_{y_-} \hat{y} + f_u u \Big\} \\ &= \left(f_{y_+} g_y g_y + f_{y_0} g_y + f_{y_-} \right) \hat{y} + \left(f_{y_+} g_y g_u + f_{y_0} g_u + f_u \right) u \\ &= 0 \end{aligned}$$

As you can see, since future shocks only enter with their first moments (which are zero in expectations), they drop out when taking expectations of the linearized equations. This is technically why certainty equivalence holds

in a system linearized to its first order. The second thing to note is that we have two unknown variables in the above equation: g_y and g_u each of which will help us recover the policy function g.

Since the above equation holds for any $\hat{y}$ and any u, each parenthesis must be null and we can solve each at a time. The first, yields a quadratic equation in g_y, which we can solve with a series of algebraic trics that are not all immediately apparent (but detailed in Michel Juillard's presentation). Incidentally, one of the conditions that comes out of the solution of this equation is the Blanchard-Kahn condition: there must be as many roots larger than one in modulus as there are forward-looking variables in the model. Having recovered g_y, recovering g_u is then straightforward from the second parenthesis.

Finally, notice that a first order linearization of the function g yields:

$$y_t = \bar{y} + g_y\hat{y} + g_u u$$

And now that we have g_y and g_u, we have solved for the (approximate) policy (or decision) function and have succeeded in solving our DSGE model. If we were interested in impulse response functions, for instance, we would simply iterate the policy function starting from an initial value given by the steady state.

The second order solution uses the same "perturbation methods" as above (the notion of starting from a function you can solve - like a steady state - and iterating forward), yet applies more complex algebraic techniques to recover the various partial derivatives of the policy function. But the general approach is perfectly isomorphic. Note that in the case of a second order approximation of a DSGE model, the variance of future shocks remains after taking expectations of the linearized equations and therefore affects the level of the resulting policy function.

Chapter 8

Estimating DSGE models - Behind the scenes of Dynare

This chapter focuses on the theory of Bayesian estimation. It begins by motivating Bayesian estimation by suggesting some arguments in favor of it as opposed to other forms of model estimation. It then attempts to shed some light on what goes on in Dynare's machinery when it estimates DSGE models. To do so, this section surveys the methodologies adopted for Bayesian estimation, including defining what are prior and posterior distributions, using the Kalman filter to find the likelihood function, estimating the posterior function thanks to the Metropolis-Hastings algorithm, and comparing models based on posterior distributions.

8.1 Advantages of Bayesian estimation

Bayesian estimation is becoming increasingly popular in the field of macroeconomics. Recent papers have attracted significant attention; some of these include: Schorfheide (2000) which uses Bayesian methods to compare the fit of two competing DSGE models of consumption, Lubik and Schorfheide (2003) which investigates whether central banks in small open economies respond to exchange rate movements, Smets and Wouters (2003) which applies Bayesian estimation techniques to a model of the Eurozone, Ireland (2004) which emphasizes instead maximum likelihood estimation, Fernandez-Villaverde and Rubio-Ramirez (2004) which reviews the econometric properties of Bayesian estimators and compare estimation results with maximum likelihood and BVAR methodologies, Lubik and Schorfheide (2005) which applies Bayesian estimation methods to an open macro model focussing on issues of misspecification and identification, and finally Rabanal and Rubio-Ramirez (2005) which compares the fit, based on posterior distributions, of four competing specifications of New Keynesian monetary models with nominal rigidi-

ties.

There are a multitude of advantages of using Bayesian methods to estimate a model, but five of these stand out as particularly important and general enough to mention here.

First, Bayesian estimation fits the complete, solved DSGE model, as opposed to GMM estimation which is based on particular equilibrium relationships such as the Euler equation in consumption. Likewise, estimation in the Bayesian case is based on the likelihood generated by the DSGE system, rather than the more indirect discrepancy between the implied DSGE and VAR impulse response functions. Of course, if your model is entirely mis-specified, estimating it using Bayesian techniques could be a disadvantage.

Second, Bayesian techniques allow the consideration of priors which work as weights in the estimation process so that the posterior distribution avoids peaking at strange points where the likelihood peaks. Indeed, due to the stylized and often misspecified nature of DSGE models, the likelihood often peaks in regions of the parameter space that are contradictory with common observations, leading to the "dilemma of absurd parameter estimates".

Third, the inclusion of priors also helps identifying parameters. Unfortunately, when estimating a model, the problem of identification often arises. It can be summarized by different values of structural parameters leading to the same joint distribution for observables. More technically, the problem arises when the posterior distribution is flat over a subspace of parameter values. But the weighting of the likelihood with prior densities often leads to adding just enough curvature in the posterior distribution to facilitate numerical maximization.

Fourth, Bayesian estimation explicitly addresses model misspecification by including shocks, which can be interpreted as observation errors, in the structural equations.

Sixth, Bayesian estimation naturally leads to the comparison of models based on fit. Indeed, the posterior distribution corresponding to competing models can easily be used to determine which model best fits the data. This procedure, as other topics mentioned above, is discussed more technically in the subsection below.

8.2 The basic mechanics of Bayesian estimation

This and the following subsections are based in great part on work by, and discussions with, Stéphane Adjemian, a member of the Dynare development team. Some of this work, although summarized in presentation format, is available in the "conferences and workshops" page of the Dynare website. Other helpful material includes An and Schorfheide (2006), which includes a clear and quite complete introduction to Bayesian estimation, illustrated by the application of a simple DSGE model. Also, the appendix of Schorfheide (2000) contains details as to the exact methodology and possible difficulties encountered in Bayesian estimation. You may also want to take a glance at Hamilton (1994), chapter 12, which provides a very clear, although somewhat outdated, introduction to the basic mechanics of Bayesian estimation. Finally, the websites of Frank Schorfheide and Jesus Fernandez-Villaverde contain a wide variety of very helpful material, from example files to lecture notes to related papers. Finally, remember to also check the open online examples of the Dynare website for examples of .mod files touching on Bayesian estimation.

At its most basic level, Bayesian estimation is a bridge between calibration and maximum likelihood. The tradition of calibrating models is inherited through the specification of priors. And the maximum likelihood approach enters through the estimation process based on confronting the model with data. Together, priors can be seen as weights on the likelihood function in order to give more importance to certain areas of the parameter subspace. More technically, these two building blocks - priors and likelihood functions - are tied together by Bayes' rule. Let's see how.

First, priors are described by a density function of the form

$$p(\boldsymbol{\theta}_{\mathcal{A}}|\mathcal{A})$$

where $\mathcal{A}$ stands for a specific model, $\boldsymbol{\theta}_{\mathcal{A}}$ represents the parameters of model $\mathcal{A}$, $p(\bullet)$ stands for a probability density function (pdf) such as a normal, gamma, shifted gamma, inverse gamma, beta, generalized beta, or uniform function.

Second, the likelihood function describes the density of the observed data, given the model and its parameters:

$$\mathcal{L}(\boldsymbol{\theta}_{\mathcal{A}}|\mathbf{Y}_T, \mathcal{A}) \equiv p(\mathbf{Y}_T|\boldsymbol{\theta}_{\mathcal{A}}, \mathcal{A})$$

where $\mathbf{Y}_T$ are the observations until period T, and where in our case the likelihood is recursive and can be written as:

$$p(\mathbf{Y}_T|\boldsymbol{\theta}_{\mathcal{A}}, \mathcal{A}) = p(y_0|\boldsymbol{\theta}_{\mathcal{A}}, \mathcal{A}) \prod_{t=1}^{T} p(y_t|\mathbf{Y}_{t-1}, \boldsymbol{\theta}_{\mathcal{A}}, \mathcal{A})$$

We now take a step back. Generally speaking, we have a prior density $p(\boldsymbol{\theta})$ on the one hand, and on the other, a likelihood $p(\mathbf{Y}_T|\boldsymbol{\theta})$. In the end, we are interested in $p(\boldsymbol{\theta}|\mathbf{Y}_T)$, the **posterior density**. Using the **Bayes theorem** twice we obtain this density of parameters knowing the data. Generally, we have

$$p(\boldsymbol{\theta}|\mathbf{Y}_T) = \frac{p(\boldsymbol{\theta};\mathbf{Y}_T)}{p(\mathbf{Y}_T)}$$

We also know that

$$p(\mathbf{Y}_T|\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta};\mathbf{Y}_T)}{p(\boldsymbol{\theta})} \Leftrightarrow p(\boldsymbol{\theta};\mathbf{Y}_T) = p(\mathbf{Y}_T|\boldsymbol{\theta}) \times p(\boldsymbol{\theta})$$

By using these identities, we can combine the **prior density** and the **likelihood function** discussed above to get the posterior density:

$$p(\boldsymbol{\theta}_{\mathcal{A}}|\mathbf{Y}_T,\mathcal{A}) = \frac{p(\mathbf{Y}_T|\boldsymbol{\theta}_{\mathcal{A}},\mathcal{A})p(\boldsymbol{\theta}_{\mathcal{A}}|\mathcal{A})}{p(\mathbf{Y}_T|\mathcal{A})}$$

where $p(\mathbf{Y}_T|\mathcal{A})$ is the **marginal density** of the data conditional on the model:

$$p(\mathbf{Y}_T|\mathcal{A}) = \int_{\Theta_{\mathcal{A}}} p(\boldsymbol{\theta}_{\mathcal{A}};\mathbf{Y}_T|\mathcal{A})d\boldsymbol{\theta}_{\mathcal{A}}$$

Finally, the **posterior kernel** (or un-normalized posterior density, given that the marginal density above is a constant or equal for any parameter), corresponds to the numerator of the posterior density:

$$p(\boldsymbol{\theta}_{\mathcal{A}}|\mathbf{Y}_T,\mathcal{A}) \propto p(\mathbf{Y}_T|\boldsymbol{\theta}_{\mathcal{A}},\mathcal{A})p(\boldsymbol{\theta}_{\mathcal{A}}|\mathcal{A}) \equiv \mathcal{K}(\boldsymbol{\theta}_{\mathcal{A}}|\mathbf{Y}_T,\mathcal{A})$$

This is the fundamental equation that will allow us to rebuild all posterior moments of interest. The trick will be to estimate the likelihood function with the help of the **Kalman filter** and then simulate the posterior kernel using a sampling-like or Monte Carlo method such as the **Metropolis-Hastings**. These topics are covered in more details below. Before moving on, though, the subsection below gives a simple example based on the above reasoning of what we mean when we say that Bayesian estimation is "somewhere in between calibration and maximum likelihood estimation". The example is drawn from Zellner (1971), although other similar examples can be found in Hamilton (1994), chapter 12.

8.2.1 Bayesian estimation: somewhere between calibration and maximum likelihood estimation - an example

Suppose a data generating process $y_t = \mu + \varepsilon_t$ for $t = 1, ..., T$, where $\varepsilon_t \sim \mathcal{N}(0,1)$ is gaussian white noise. Then, the likelihood is given by

$$p(\mathbf{Y}_T|\mu) = (2\pi)^{-\frac{T}{2}} e^{-\frac{1}{2}\sum_{t=1}^{T}(y_t-\mu)^2}$$

We know from the above that $\widehat{\mu}_{ML,T} = \frac{1}{T}\sum_{t=1}^{T} y_t \equiv \overline{y}$ and that $\mathbb{V}[\widehat{\mu}_{ML,T}] = \frac{1}{T}$.

In addition, let our prior be a gaussian distribution with expectation μ_0 and variance σ_μ^2. Then, the posterior density is defined, up to a constant, by:

$$p\left(\mu|\mathbf{Y}_T\right) \propto (2\pi\sigma_\mu^2)^{-\frac{1}{2}} e^{-\frac{1}{2}\frac{(\mu-\mu_0)^2}{\sigma_\mu^2}} \times (2\pi)^{-\frac{T}{2}} e^{-\frac{1}{2}\sum_{t=1}^{T}(y_t-\mu)^2}$$

Or equivalently, $p\left(\mu|\mathbf{Y}_T\right) \propto e^{-\frac{(\mu-\mathbb{E}[\mu])^2}{\mathbb{V}[\mu]}}$, with

$$\mathbb{V}[\mu] = \frac{1}{\left(\frac{1}{T}\right)^{-1} + \sigma_\mu^{-2}}$$

and

$$\mathbb{E}[\mu] = \frac{\left(\frac{1}{T}\right)^{-1}\widehat{\mu}_{ML,T} + \sigma_\mu^{-2}\mu_0}{\left(\frac{1}{T}\right)^{-1} + \sigma_\mu^{-2}}$$

From this, we can tell that the posterior mean is a convex combination of the prior mean and the ML estimate. In particular, if $\sigma_\mu^2 \to \infty$ (ie, we have no prior information, so we just estimate the model) then $\mathbb{E}[\mu] \to \widehat{\mu}_{ML,T}$, the maximum likelihood estimator. But if $\sigma_\mu^2 \to 0$ (ie, we're sure of ourselves and we calibrate the parameter of interest, thus leaving no room for estimation) then $\mathbb{E}[\mu] \to \mu_0$, the prior mean. Most of the time, we're somewhere in the middle of these two extremes.

8.3 DSGE models and Bayesian estimation

8.3.1 Rewriting the solution to the DSGE model

Recall from chapter 7 that any DSGE model, which is really a collection of first order and equilibrium conditions, can be written in the form $\mathbb{E}_t\left\{f(y_{t+1}, y_t, y_{t-1}, u_t)\right\} = 0$, taking as a solution equations of the type $y_t = g(y_{t-1}, u_t)$, which we call the decision rule. In more appropriate terms for what follows, we can rewrite the solution to a DSGE model as a system in the following manner:

$$\begin{aligned} y_t^* &= M\bar{y}(\theta) + M\hat{y}_t + N(\theta)x_t + \eta_t \\ \hat{y}_t &= g_y(\theta)\hat{y}_{t-1} + g_u(\theta)u_t \\ E(\eta_t\eta_t') &= V(\theta) \\ E(u_t u_t') &= Q(\theta) \end{aligned}$$

where $\hat{y}_t$ are variables in deviations from steady state, $\bar{y}$ is the vector of steady state values and θ the vector of deep (or structural) parameters to be estimated. Other variables are described below.

The second equation is the familiar decision rule mentioned above. But the equation expresses a relationship among true endogenous variables that are not directly observed. Only y_t^* is observable, and it is related to the true variables with an error η_t. Furthermore, it may have a trend, which is captured with $N(\theta)x_t$ to allow for the most general case in which the trend depends on the deep parameters. The first and second equations above therefore naturally make up a system of measurement and transition or state equations, respectively, as is typical for a Kalman filter (you guessed it, it's not a coincidence!).

8.3.2 Estimating the likelihood function of the DSGE model

The next logical step is to estimate the likelihood of the DSGE solution system mentioned above. The first apparent problem, though, is that the equations are non linear in the deep parameters. Yet, they are linear in the endogenous and exogenous variables so that the likelihood may be evaluated with a linear prediction error algorithm like the Kalman filter. This is exactly what Dynare does. As a reminder, here's what the Kalman filter recursion does.

For $t = 1, \ldots, T$ and with initial values y_1 and P_1 given, the recursion follows

$$\begin{aligned} v_t &= y_t^* - \bar{y}^* - M\hat{y}_t - Nx_t \\ F_t &= MP_tM' + V \\ K_t &= g_y P_t g_y' F_t^{-1} \\ \hat{y}_{t+1} &= g_y \hat{y}_t + K_t v_t \\ P_{t+1} &= g_y P_t (g_y - K_t M)' + g_u Q g_u' \end{aligned}$$

For more details on the Kalman filter, see Hamilton (1994), chapter 13.

From the Kalman filter recursion, it is possible to derive the **log-likelihood** given by

$$\ln \mathcal{L}\left(\boldsymbol{\theta}|\mathbf{Y}_T^*\right) = -\frac{Tk}{2}\ln(2\pi) - \frac{1}{2}\sum_{t=1}^{T}|F_t| - \frac{1}{2}v_t' F_t^{-1} v_t$$

where the vector $\boldsymbol{\theta}$ contains the parameters we have to estimate: θ, $V(\theta)$ and $Q(\theta)$ and where Y_T^* expresses the set of observable endogenous variables y_t^* found in the measurement equation.

The log-likelihood above gets us one step closer to our goal of finding the posterior distribution of our parameters. Indeed, the **log posterior kernel** can be expressed as

$$\ln \mathcal{K}(\boldsymbol{\theta}|\mathbf{Y}_T^*) = \ln \mathcal{L}\left(\boldsymbol{\theta}|\mathbf{Y}_T^*\right) + \ln p(\boldsymbol{\theta})$$

where the first term on the right hand side is now known after carrying out the Kalman filter recursion. The second, recall, are the priors, and are also known.

8.3.3 Finding the mode of the posterior distribution

Next, to find the mode of the posterior distribution - a key parameter and an important output of Dynare - we simply maximize the above log posterior kernel with respect to θ. This is done in Dynare using numerical methods. Recall that the likelihood function is not Gaussian with respect to θ but to functions of θ as they appear in the state equation. Thus, this maximization problem is not completely straightforward, but fortunately doable with modern computers.

8.3.4 Estimating the posterior distribution

Finally, we are now in a position to find the posterior distribution of our parameters. The distribution will be given by the kernel equation above, but again, it is a nonlinear and complicated function of the deep parameters θ. Thus, we cannot obtain an explicit form for it. We resort, instead, to sampling-like methods, of which the Metropolis-Hastings has been retained in the literature as particularly efficient. This is indeed the method adopted by Dynare.

The general idea of the Metropolis-Hastings algorithm is to simulate the posterior distribution. It is a "rejection sampling algorithm" used to generate a sequence of samples (also known as a "Markov Chain" for reasons that will become apparent later) from a distribution that is unknown at the outset. Remember that all we have is the posterior mode; we are instead more often interested in the mean and variance of the estimators of θ. To do so, the algorithm builds on the fact that under general conditions the distribution of the deep parameters will be asymptotically normal. The algorithm, in the words of An and Shorfheide, "constructs a Gaussian approximation around the posterior mode and uses a scaled version of the asymptotic covariance matrix as the covariance matrix for the proposal distribution. This allows for an efficient exploration of the posterior distribution at least in the neighborhood of the mode" (An and Schorfheide (2006), p. 18). More precisely, the **Metropolis-Hastings algorithm implements the following steps**:

1. Choose a starting point $\boldsymbol{\theta}^{\circ}$, where this is typically the posterior mode, and run a loop over 2-3-4.

2. Draw a *proposal* $\boldsymbol{\theta}^{*}$ from a *jumping* distribution

$$J(\boldsymbol{\theta}^{*}|\boldsymbol{\theta}^{t-1}) = \mathcal{N}(\boldsymbol{\theta}^{t-1}, c\Sigma_{m})$$

where Σ_m is the inverse of the Hessian computed at the posterior mode.

3. Compute the acceptance ratio

$$r = \frac{p(\boldsymbol{\theta}^*|\mathbf{Y}_T)}{p(\boldsymbol{\theta}^{t-1}|\mathbf{Y}_T)} = \frac{\mathcal{K}(\boldsymbol{\theta}^*|\mathbf{Y}_T)}{\mathcal{K}(\boldsymbol{\theta}^{t-1}|\mathbf{Y}_T)}$$

4. Finally accept or discard the proposal $\boldsymbol{\theta}^*$ according to the following rule, and update, if necessary, the jumping distribution:

$$\boldsymbol{\theta}^t = \begin{cases} \boldsymbol{\theta}^* & \text{with probability } \min(r, 1) \\ \boldsymbol{\theta}^{t-1} & \text{otherwise.} \end{cases}$$

Figure 8.3.4 tries to clarify the above. In step 1, choose a candidate paramter, θ^* from a Normal distribution, whose mean has been set to θ^{t-1} (this will become clear in just a moment). In step 2, compute the value of the posterior kernel for that candidate parameter, and compare it to the value of the kernel from the mean of the drawing distribution. In step 3, decide whether or not to hold on to your candidate parameter. If the acceptance ratio is greater than one, then definitely keep your candidate. Otherwise, go back to the candidate of last period (this is true in very coarse terms, notice that in fact you would keep your candidate only with a probability less than one). Then, do two things. Update the mean of your drawing distribution, and note the value of the parameter your retain. After having repeated these steps often enough, in the final step, build a histogram of those retained values. Of course, the point is for each "bucket" of the histogram to shrink to zero. This "smoothed histogram" will eventually be the posterior distribution after sufficient iterations of the above steps.

But why have such a complicated acceptance rule? The point is to be able to visit the entire domain of the posterior distribution. We should not be too quick to simply throw out the candidate giving a lower value of the posterior kernel, just in case using that candidate for the mean of the drawing distribution allows us to to leave a local maximum and travel towards the global maximum. Metaphorically, the idea is to allow the search to turn away from taking a small step up, and instead take a few small steps down in the hope of being able to take a big step up in the near future. Of course, an important parameter in this searching procedure is the variance of the jumping distribution and in particular the **scale factor**. If the scale factor is too small, the **acceptance rate** (the fraction of candidate parameters that are accepted in a window of time) will be too high and the Markov Chain of candidate parameters will "mix slowly", meaning that the distribution will take a long time to converge to the posterior distribution since the chain is likely to get "stuck" around a local maximum. On the other hand, if the scale factor is too large, the acceptance rate will be very low (as the candidates are likely to

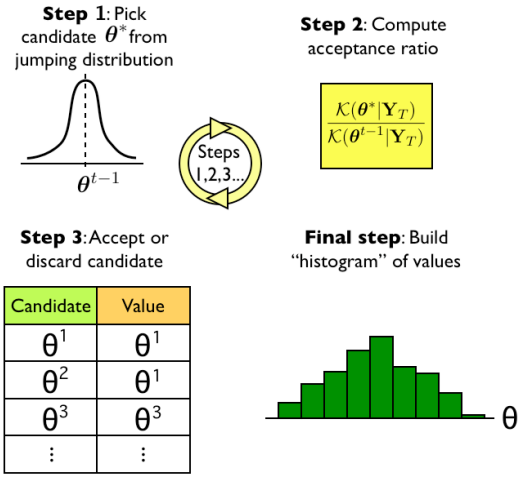

Figure 8.1: The above sketches the Metropolis-Hastings algorithm, used to build the posterior distribution function. Imagine repeating these steps a large number of times, and smoothing the "histogram" such that each "bucket" has infinitely small width.

land in regions of low probability density) and the chain will spend too much time in the tails of the posterior distribution.

While these steps are mathematically clear, at least to a machine needing to undertake the above calculations, several practical questions arise when carrying out Bayesian estimation. These include: How should we choose the scale factor c (variance of the jumping distribution)? What is a satisfactory acceptance rate? How many draws are ideal? How is convergence of the Metropolis-Hastings iterations assessed? These are all important questions that will come up in your usage of Dynare. They are addressed as clearly as possible in section 5.7 of Chapter 5.

8.4 Comparing models using posterior distributions

As mentioned earlier, while touting the advantages of Bayesian estimation, the posterior distribution offers a particularly natural method of comparing models. Let's look at an illustration.

Suppose we have a prior distribution over two competing models: $p(\mathcal{A})$ and $p(\mathcal{B})$. Using Bayes' rule, we can compute the posterior distribution over models, where $\mathcal{I} = \mathcal{A}, \mathcal{B}$

$$p(\mathcal{I}|\mathbf{Y}_T) = \frac{p(\mathcal{I})p(\mathbf{Y}_T|\mathcal{I})}{\sum_{\mathcal{I}=\mathcal{A},\mathcal{B}} p(\mathcal{I})p(\mathbf{Y}_T|\mathcal{I})}$$

where this formula may easily be generalized to a collection of N models.

Then, the comparison of the two models is done very naturally through the ratio of the posterior model distributions. We call this the **posterior odds ratio**:

$$\frac{p(\mathcal{A}|\mathbf{Y}_T)}{p(\mathcal{B}|\mathbf{Y}_T)} = \frac{p(\mathcal{A})}{p(\mathcal{B})}\frac{p(\mathbf{Y}_T|\mathcal{A})}{p(\mathbf{Y}_T|\mathcal{B})}$$

The only complication is finding the magrinal density of the data conditional on the model, $p(\mathbf{Y}_T|\mathcal{I})$, which is also the denominator of the posterior density $p(\boldsymbol{\theta}|\mathbf{Y}_T)$ discussed earlier. This requires some detailed explanations of their own.

For each model $\mathcal{I} = \mathcal{A}, \mathcal{B}$ we can evaluate, at least theoretically, the marginal density of the data conditional on the model by integrating out the deep parameters $\boldsymbol{\theta}_\mathcal{I}$ from the posterior kernel:

$$p(\mathbf{Y}_T|\mathcal{I}) = \int_{\Theta_\mathcal{I}} p(\boldsymbol{\theta}_\mathcal{I}; \mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I}, \mathcal{I})d\boldsymbol{\theta}_\mathcal{I} = \int_{\Theta_\mathcal{I}} p(\boldsymbol{\theta}_\mathcal{I}|\mathcal{I}) \times p(\mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I}, \mathcal{I})d\boldsymbol{\theta}_\mathcal{I}$$

Note that the expression inside the integral sign is exactly the posterior kernel. To remind you of this, you may want to glance back at the first subsection above, specifying the basic mechanics of Bayesian estimation.

To obtain the marginal density of the data conditional on the model, there are two options. The first is to assume a functional form of the posterior kernel that we can integrate. The most straightforward and the best approximation, especially for large samples, is the Gaussian (called a **Laplace approximation**). In this case, we would have the following estimator:

$$\widehat{p}(\mathbf{Y}_T|\mathcal{I}) = (2\pi)^{\frac{k}{2}}|\Sigma_{\boldsymbol{\theta}^m_\mathcal{I}}|^{\frac{1}{2}}p(\boldsymbol{\theta}^m_\mathcal{I}|\mathbf{Y}_T, \mathcal{I})p(\boldsymbol{\theta}^m_\mathcal{I}|\mathcal{I})$$

where $\boldsymbol{\theta}^m_\mathcal{I}$ is the posterior mode. The advantage of this technique is its computational efficiency: time consuming Metropolis-Hastings iterations are not

necessary, only the numerically calculated posterior mode is required.

The second option is instead to use information from the Metropolis-Hastings runs and is typically referred to as the **Harmonic Mean Estimator**. The idea is to simulate the marginal density of interest and to simply take an average of these simulated values. To start, note that

$$p(\mathbf{Y}_T|\mathcal{I}) = \mathbb{E}\left[\frac{f(\boldsymbol{\theta}_\mathcal{I})}{p(\boldsymbol{\theta}_\mathcal{I}|\mathcal{I})p(\mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I},\mathcal{I})}\middle|\boldsymbol{\theta}_\mathcal{I},\mathcal{I}\right]^{-1}$$

where f is a probability density function, since

$$\mathbb{E}\left[\frac{f(\boldsymbol{\theta}_\mathcal{I})}{p(\boldsymbol{\theta}_\mathcal{I}|\mathcal{I})p(\mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I},\mathcal{I})}\middle|\boldsymbol{\theta}_\mathcal{I},\mathcal{I}\right] = \frac{\int_{\Theta_\mathcal{I}} f(\boldsymbol{\theta})d\boldsymbol{\theta}}{\int_{\Theta_\mathcal{I}} p(\boldsymbol{\theta}_\mathcal{I}|I)p(\mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I},\mathcal{I})d\boldsymbol{\theta}_\mathcal{I}}$$

and the numerator integrates out to one (seeGeweke (1999) for more details).

This suggests the following estimator of the marginal density

$$\widehat{p}(\mathbf{Y}_T|\mathcal{I}) = \left[\frac{1}{B}\sum_{b=1}^{B}\frac{f(\boldsymbol{\theta}_\mathcal{I}^{(b)})}{p(\boldsymbol{\theta}_\mathcal{I}^{(b)}|\mathcal{I})p(\mathbf{Y}_T|\boldsymbol{\theta}_\mathcal{I}^{(b)},\mathcal{I})}\right]^{-1}$$

where each drawn vector $\boldsymbol{\theta}_\mathcal{I}^{(b)}$ comes from the Metropolis-Hastings iterations and where the probability density function f can be viewed as a weights on the posterior kernel in order to downplay the importance of extreme values of $\boldsymbol{\theta}$. Geweke (1999) suggests to use a truncated Gaussian function, leading to what is typically referred to as the **Modified Harmonic Mean Estimator**.

Chapter 9

Optimal policy under commitment

Chapter 10

Troubleshooting

To make sure this section is as user friendly as possible, the best is to compile what users have to say! Please let me know what your most common problem is with Dynare, how Dynare tells you about it and how you solve it. Thanks for your precious help!

Bibliography

An, S., and F. Schorfheide (2006): "Bayesian Analysis of DSGE Models," *Econometric Review*, Forthcoming.

Clarida, R., J. Gali, and M. Gertler (1999): "The Science of Monetary Policy: A New Keynesian Perspective," *Journal of Economic Literature*, XXXVII, 1661–1707.

Collard, F., and M. Juillard (2001a): "Accuracy of stochastic perturbation methods: The case of asset pricing models," *Journal of Economic Dynamics and Control*, 25(6-7), 979–999.

——— (2001b): "A Higher-Order Taylor Expansion Approach to Simulation of Stochastic Forward-Looking Models with an Application to a Nonlinear Phillips Curve Model," *Computational Economics*, 17(2-3), 125–39.

——— (2003): "Stochastic simulations with DYNARE. A practical guide.," CEPREMAP mimeo.

Durbin, J., and S. Koopman (2001): *Time Series Analysis by State Space Methods*. Oxford University Press, Oxford, U.K.

Fernandez-Villaverde, J., and J. F. Rubio-Ramirez (2004): "Comparing dynamic equilibrium models to data: a Bayesian approach," *Journal of Econometrics*, 123(1), 153–187.

Geweke, J. (1999): "Using Simulation Methods for Bayesian Econometric Models: In- ference, Development and Communication," *Econometric Review*, 18(1), 1–126.

Hamilton, J. D. (1994): *Time Series Analysis*. Princeton University Press, Princeton, NJ.

Ireland, P. N. (2004): "A method for taking models to the data," *Journal*

Lubik, T., and F. Schorfheide (2003): "Do Central Banks Respond to Exchange Rate Movements? A Structural Investigation," Economics Working Paper Archive 505, The Johns Hopkins University,Department of Economics.

——— (2005): "A Bayesian Look at New Open Economy Macroeconomics," Economics Working Paper Archive 521, The Johns Hopkins University,Department of Economics.

Nason, J. M., and T. Cogley (1994): "Testing the Implications of Long-Run Neutrality for Monetary Business Cycle Models," *Journal of Applied Econometrics*, 9(S), S37–70.

Rabanal, P., and J. F. Rubio-Ramirez (2005): "Comparing New Keynesian models of the business cycle: A Bayesian approach," *Journal of Monetary Economics*, 52(6), 1151–1166.

Schmitt-Grohe, S., and M. Uribe (2004): "Solving dynamic general equilibrium models using a second-order approximation to the policy function," *Journal of Economic Dynamics and Control*, 28(4), 755–775.

Schorfheide, F. (2000): "Loss function-based evaluation of DSGE models," *Journal of Applied Econometrics*, 15(6), 645–670.

Smets, F., and R. Wouters (2003): "An Estimated Dynamic Stochastic General Equilibrium Model of the Euro Area," *Journal of the European Economic Association*, 1(5), 1123–1175.

Zellner, A. (1971): *An Introduction to Bayesian Inference in Econometrics*. John Wiley & Sons, Inc., New York.