

GENERAL EQUILIBRIUM ECONOMIC MODELLING LANGUAGE AND SOLUTION FRAMEWORK VERSION 1.2.3

WARSAW, APRIL 13, 2025

© Chancellery of the Prime Minister of the Republic of Poland 2012-2015
 © Grzegorz Klima, Karol Podemski, Kaja Retkiewicz-Wijtiwiak 2015-2018
 © Karol Podemski, Kaja Retkiewicz-Wijtiwiak 2019-2025

The views expressed herein are solely of the authors and do not necessarily reflect those of the Chancellery of the Prime Minister of the Republic of Poland or any other institution the authors have been/are employed by or affiliated with. DEVELOPMENT TEAM:

Karol Podemski (since 10.2012)

Kaja Retkiewicz-Wijtiwiak (since 10.2012)

PAST DEVELOPMENT TEAM MEMBERS:

Grzegorz Klima (lead developer 10.2012-03.2018)

Contents

In	TROD	DUCTION	3							
1	Get	TTING STARTED — YOUR FIRST MODEL IN gEcon	6							
	1.1	A SAMPLE MODEL ECONOMY	6							
	1.2	LANGUAGE	7							
	1.3	Reading model from R	10							
	1.4	FINDING THE STEADY STATE	11							
	1.5	Solving for dynamics	13							
	1.6	Results — correlations and IRFs	14							
	1.7	Automatic generation of model documentation in ${\rm I}\!\!\!{}^{\!\!A}T_{\!E}\!X$	16							
2	INST	TALLATION INSTRUCTIONS	18							
	2.1	Requirements	18							
	2.2	INSTALLATION	18							
	2.3	Syntax highlighting	18							
	2.4	Examples	19							
3	Moi	Model description language								
	3.1	Syntax basics	20							
	3.2	Organisation of gEcon input file	22							
	3.3	Options	22							
	3.4	VARIABLE REDUCTION	25							
	3.5	Model blocks	25							
4	TEM	Templates 3								
	4.1	INDEX SETS	30							
	4.2	INDEXED VARIABLES AND PARAMETERS	32							
	4.3	INDEXING EXPRESSIONS	33							
	4.4	The Kronecker delta and the rules of differentiation	35							
	4.5	An example — pure exchange model	37							
5	Moi	Model variants — using the preprocessor 34								
	5.1	Declaring model variants	39							
	5.2	Selecting model variants	40							
	5.3	An example — pure exchange model with different numéraires	40							
6	DER	DERIVATION OF FIRST ORDER CONDITIONS 4								
	6.1	The canonical problem	44							
	6.2	FIRST ORDER CONDITIONS	45							



General equilibrium economic modelling language and solution framework

	6.3	HANDLING LAGS GREATER THAN ONE	46
7	RC	LASSES	47
	7.1	CREATING gecon_model OBJECT	47
	7.2	INTERNAL REPRESENTATION	47
	7.3	FUNCTIONS OF gecon_model CLASS	48
	7.4	gecon_simulation CLASS	48
	7.5	INFORMATION ABOUT VARIABLES, PARAMETERS, AND SHOCKS	48
8	Det	ERMINISTIC STEADY STATE & CALIBRATION	50
	8.1	DETERMINISTIC STEADY STATE	50
	8.2	CALIBRATION OF PARAMETERS	50
	8.3	Implemented solvers	51
	8.4	How to improve the chances of finding solution?	51
	8.5	TROUBLESHOOTING	51
9	SOL	VING THE MODEL IN LINEARISED FORM	53
	9.1	Log-linearisation.	53
	9.2	CANONICAL FORM OF THE MODEL AND SOLUTION	54
	9.3	Solution procedure.	55
	9.4	TROUBLESHOOTING	56
10	Moi	DEL STATISTICS & SIMULATION	58
	10.1	Specification of shock distribution	58
	10.2	Computation of correlations	59
	10.3	SIMULATING THE MODEL	62
11	Wo	RKING WITH MODELS FROM R	64
	11.1	INFORMATION ABOUT PARAMETERS, VARIABLES & SHOCKS	64
	11.2	MODELS WRITTEN USING gEcon TEMPLATE MECHANISM	66
	11.3	MODEL EQUATIONS	67
	11.4	ACCESSING MODEL RESULTS.	67
	11.5	DOCUMENTING RESULTS IN LATEX	71
AF	PENI	DIX A. gEcon SOFTWARE LICENCE	72
AF	PENI	DIX B. ANTRL C++ TARGET SOFTWARE LICENSE	75
BIF	BLIOG	RAPHY	76
Int)EX		77
			•••

INTRODUCTION

gEcon is a framework for developing and solving large scale dynamic (stochastic) & static general equilibrium models. It consists of model description language and an interface with a set of solvers in R.

Publicly available toolboxes used in RBC/DSGE modelling require users to derive the first order conditions (FOCs) and linearisation equations by pen & paper (e.g. Uhlig's tool-kit, [Uhlig 1995]) or at least require manual derivation of the FOCs (e.g. Dynare, [Adjemian *et al.* 2013]). Derivation of FOCs is also required by GAMS [Brooke *et al.* 1996] and GEMPACK [Harrison *et al.* 2014] — probably the two most popular frameworks used in CGE modelling. Owing to the development of an algorithm for automatic derivation of first order conditions and implementation of a comprehensive symbolic computations library, gEcon allows users to describe their models in terms of optimisation problems of agents. To authors' best knowledge there is no other publicly available framework for writing and solving DSGE & CGE models in this natural way. Writing models in terms of optimisation problems instead of the FOCs is far more natural to an economist, takes off the burden of tedious differentiation, and reduces the risk of making a mistake. gEcon allows users to focus on economic aspects of the model and makes it possible to design large-scale (100+ variables) models. To this end, gEcon provides template mechanism (similar to those found in CGE modelling packages), which allows to declare similar agents (differentiated by parameters only) in a single block. Additionally, gEcon can automatically produce a draft of IAT_FX documentation for a model.

The model description language is simple and intuitive. Given optimisation problems, constraints and identities, computer derives the FOCs, steady state equations, and linearisation matrices automatically. Numerical solvers can be then employed to determine the steady state and approximate equilibrium laws of motion around it.

gEcon was initially (2012–2015) developed at the Department for Strategic Analyses at the Chancellery of the Prime Minister of the Republic of Poland as a part of a project aiming at construction of large scale DSGE & CGE models of the Polish economy. Since March 1, 2015 gEcon is no longer developed and maintained at the Chancellery of the Prime Minister, but it is still maintained and developed by its original authors.

About current release

gEcon 1.2.0 was released on September 8, 2019. This version comes with some changes in model solution procedures and new functionalities:

- The steady_state function uses by default initial values of variables and calibrated parameters specified by the user as starting values instead of their final values from the previous function call. However, both initial values and last solver iteration values are stored in the gecon_model, gecon_var_info and gecon_par_info classes objects even if the solver has not converged and can be used as needed. Additionally, two new functions get_init_calibr_par and get_init_val_var were added to facilitate the access to the initial values which are currently in use.
- The model preprocessor has been added. It allows to analyse different variants of the same model with a few equations (or blocks) modified instead of maintaining two (or more) versions of a model and keeping them synchronised.
- An option to generate model equations in C++, compile them and make them available to gecon_model objects through Rcpp has been added. This option may reduce the model solution time and, in case of larger models, their compilation time.

gEcon 1.2.3 was released on April 13, 2025. This is a patch that adjusts automatically generated parser and lexer codes to gcc version 14 standards and introduces a minor code optimization.

Why R?

All popular DSGE toolboxes work within Matlab/Octave environments. The decision to break up with this tradition was carefully weighted. Firstly, all vector programming languages/environments (Matlab, Octave, R, Ox) are built atop low level linear algebra and other numerical libraries like BLAS and LAPACK. The main differences between them fall into the following categories: language features, number of extensions (libraries/packages), support, and user base. Matlab and Octave offer much more functionality through their toolboxes in fields such as differential equations, optimisation etc. On the other hand, R language is more flexible (not everything has to be a matrix!) and it has many more packages intended for analysis of economic data. Flexibility of the language and natural synergies between economic modelling and econometric work have made R the environment of choice for this project.

Contact

Please send bugs, suggestions, and comments to gEcon.maintenance@gmail.com.

Acknowledgements

The authors wish to thank Dorota Poznańska, Director of the Department for Strategic Analyses for supporting and facilitating this project during our time at the Chancellery of the Prime Minister.

The authors also wish to thank Magdalena Krupa and Anna Sowińska for early attempts at R implementation of numerical solvers.

Anna Sowińska has significantly helped by extensively testing gEcon and suggesting improvements.

Marta Retkiewicz and Magdalena Krupa have helped by testing gEcon template support.

Michał Opuchlik has helped by testing gEcon reference mechanism and reporting the problems with compilation of large models by the JIT compiler.

The authors are grateful to Igor Łankiewicz for proof reading earlier version of this manual and R documentation. All mistakes remain ours.

Very special thanks are due to Maga Retkiewicz and Radosław Bala for their design of the gEcon logo.

The authors (Karol Podemski and Kaja Retkiewicz-Wijtiwiak) also wish to appreciate the support given to the project by Grzegorz Klima after he decided to leave it. Despite the decision, Grzegorz contributed a lot to the version 1.2.0 — he wrote the preprocessor for model files, came up with the idea of C++ code generation and its compilation with the help of the Rcpp package. Moreover, he still supported the team and actively tested all new features.

Last but not least, the authors would like to thank all gEcon users for support, bug reports, suggestions, and spreading the word.

A personal note from Grzegorz Klima

It was not an easy decision to make, but after five and a half years since gEcon development started I decided to leave the team and focus on other projects. I would like to take this opportunity to look back and say a very, very big thank you to Kaja and Karol.

The idea to start such a project within the Polish public sector, without external support, and with limited resources that we had, might have been considered crazy by some, but I firmly believed that we would achieve our goals. These were put pretty simply: we wanted to develop a unified framework for construction of both DSGE as well as

multi-sector (CGE) models that would be more flexible and user-friendly than Dynare (and derive FOCs for us!). We also knew what we needed: a symbolic library, a parser, an algorithm for deriving FOCs, R class design, interfaces to numerous numerical solvers, and... hundreds of overtime hours for testing models and debugging. We got all of these. I can still remember summer 2013, when we were working on the first release. It seemed that all the pieces started to fit together, yet every attempt at implementation of a new model led to disclosure of completely new bugs. We were working 12+ hours a day for a few weeks, but — thanks to Kaja and Karol — I think I have never had so much fun at work. Somewhere about that time we came up with the name for the project and the first version of gEcon was released and presented in September 2013. Since then, the code grew, new functionalities were added, and gEcon was complemented by gEcon.iosam and gEcon.estimation packages (by Marta Retkiewicz and Karol respectively) making it ready for applied work. We (together with Ania Sowińska) implemented Smets-Wouters '03 model and wrote a paper [Klima *et al.* 2015] showing advantages of gEcon. Three years later — after we had already left the Department for Strategic Analyses — version 1.0 of gEcon was released and we realised that we have users all around the world.

gEcon would not have become such a success without the unique talent and huge effort Kaja and Karol have put into this project. Each of us has chosen a different career path, but I know we can all be proud of what we have achieved and smile to our memories. I would like to thank you both very much and wish you all the best in your future endeavours!

Lódź, March 11, 2018

1 GETTING STARTED — YOUR FIRST MODEL IN gEcon

1.1 A SAMPLE MODEL ECONOMY

As an example we will solve a classical RBC model with capital adjustment costs. Our model economy is populated by a continuum of households (with an infinite planning horizon) with identical time-separable preferences. At time ta representative agent experiences instantaneous utility from consumption and leisure given by:

$$u(C_t, L_t^{(s)}) = \frac{\left(C_t^{\mu} (1 - L_t^{(s)})^{1 - \mu}\right)^{1 - \eta}}{1 - \eta},$$
(1.1)

where C_t is consumption, $L_t^{(s)}$ is labour input (labour supply), $\eta > 0$ the coefficient of relative risk aversion. Each period the representative agent is endowed with one unit of time, $N_t = 1$. $1 - L_t^{(s)}$ denotes leisure.

Households own production factors (capital and labour) and lend them to firms. Household's capital stock evolves according to:

$$K_t^{(s)} = (1 - \delta)K_{t-1}^{(s)} + I_t, \tag{1.2}$$

where $K_t^{(s)}$ is the supply of capital stock¹, I_t is the investment and δ is the depreciation rate.

They divide their income (from capital and labour) between consumption, investments, and capital installation costs. In each period they choose between labour and leisure and between consumption and investment. A representative household maximizes expected discounted utility at time 0:

$$U_0 = \mathcal{E}_0\left[\sum_{t=0}^{\infty} \beta^t u(C_t, L_t^{(s)})\right],$$

which is recursively given by the following equation:

$$U_t = u(C_t, L_t^{(s)}) + \beta \mathbf{E}_t [U_{t+1}].$$
(1.3)

Optimisation is done subject to the following budget constraint:

$$C_t + I_t + \chi(I_t, K_{t-1}^{(s)}) K_{t-1}^{(s)} = W_t L_t^{(s)} + r_t K_{t-1}^{(s)} + \pi_t$$
(1.4)

and the law of motion of capital described by the equation (1.2). Here W_t stands for real wages, r_t — real interest rate or cost of capital, π_t — profits generated by firms, $0 < \beta < 1$ is the discount factor and $\chi(I_t, K_{t-1})$ denotes capital's installation costs, where

$$\chi(I_t, K_{t-1}) = \psi \left(\frac{I_t}{K_{t-1}} - \delta\right)^2.$$
 (1.5)

In our model economy there is also a continuum of firms, each producing a homogeneous good using the same technology operating on competitive product and factor markets. Firms rent capital and labour from households and pay for it. Technology is available to them for free and is given by the Cobb-Douglas production function:

$$Y_t = Z_t \left(K_t^{(d)} \right)^{\alpha} \left(L_t^{(d)} \right)^{1-\alpha}, \tag{1.6}$$

¹At the end of period t. Timing convention is that the value of a control variable at time t is decided at time t. This means that $K_t^{(s)}$ is the capital stock at the end of period t (at the beginning of period t+1). Firms at time t rent capital from stock $K_{t-1}^{(s)}$.

where $K_t^{(d)}$ is the demand for capital stock at time t, $L_t^{(d)}$ is the demand for labour and $0 < \alpha < 1$ stands for the capital share. Z_t , the total factor productivity, is exogenously evolving according to:

$$\log Z_t = \phi \log Z_{t-1} + \epsilon_t, \quad \epsilon_t \sim i.i.d.N(0;\sigma^2), \tag{1.7}$$

where $0 < \phi < 1$ is an autocorrelation parameter.

Each period a representative firm maximises its profits π_t , treating production factors' prices as given:

$$\max_{\substack{K_{t-1}^{(d)}, L_t^{(d)}, \pi_t}} \pi_t, \tag{1.8}$$

where $\pi_t = Y_t - W_t L_t^{(d)} - r_t K_{t-1}^{(d)}$, subject to technology constraint given by (1.6). Labour, capital and goods markets clear:²

$$L_t^{(d)} = L_t^{(s)}, (1.9)$$

$$K_t^{(d)} = K_{t-1}^{(s)},$$

$$C_t + I_t = Y_t.$$
(1.10)

Our parameter choices are standard in literature. A list of calibrated parameter values is presented in the table 1.1.

Parameter	Value	Interpretation				
α	0.36	Share of physical capital in the final good technology				
ß	0.99	Subjective discount factor				
δ	0.025	Depreciation rate of physical capital				
η	2.0	Relative risk aversion parameter				
$\dot{\mu}$	0.3	Consumption weight in utility function				
ϕ	0.95	Persistence of Z				
Ŵ	0.8	Installation costs coefficient				

 Table 1.1: Benchmark parameter values

1.2 LANGUAGE

Now, let us see how easily and intuitively we can write the described model in the gEcon language, solve it, and analyse its behaviour.

An input model accepted by gEcon should be saved as a text file with the .gcn extension, which can be created in any text editor. In this section we will show how to write our example model in the gEcon language. A formal specification and further rules governing the gEcon syntax are presented in chapter 3.

An equilibrium model in the gEcon language is divided into blocks (usually corresponding to agents in the economy) which are consistent with the logic of the model. Each block begins with the keyword block followed by its name.

²For explanation of timing convention regarding capital stock (K) confront the footnote on the previous page.

Model blocks themselves are divided into several sections (definitions, controls, objective, constraints, identities, shocks, and calibration), each having a pretty natural interpretation to an economist.

Let us see how it works on the example from the previous section. There are two optimising agents: a representative consumer and a representative firm. The consumer's block will be called **Consumer** and it will contain information about her optimisation problem. Firstly, for clearer exposition we will provide the definition of instantaneous utility in **definitions** section. The consumer problem is described in three sections: **controls** (list of control variables), **objective** (objective function given in a recursive form), and **constraints** (budget constraint and the law of capital's motion). Calibration of the parameters relevant to this block may be set in the **calibration** section or omitted in a .gcn file and later set while solving the model in R. A correctly written consumer's block is presented below:

block CONSUMER

```
ł
    definitions
    {
        u[] = (C[] mu * (1 - L_s[])^{(1 - mu)}^{(1 - eta)} / (1 - eta);
    };
    controls
    {
        K_s[], C[], L_s[], I[];
    };
    objective
    {
       U[] = u[] + beta * E[][U[1]];
   };
    constraints
   {
        I[] + C[] = r[] * K_s[-1] + W[] * L_s[] -
                    psi * K_s[-1] * (I[] / K_s[-1] - delta)^2 + pi[] : lambda_c[];
        K_s[] = (1 - delta) * K_s[-1] + I[];
    };
   calibration
   {
        delta = 0.025;
        beta = 0.99;
        eta = 2;
        mu = 0.3;
        psi = 0.8;
    };
};
```

Basic rules governing the gEcon syntax can be easily noticed. The content of separate blocks and block sections should be enclosed in curly brackets ({}). All variables lists and equations should be ended with a semicolon (;). Such an ending is optional for sections and blocks³. Variable names are followed by square brackets ([]) containing a lead or a lag relative to time t with empty brackets standing for t. Parameters are denoted using their names only.

Lagrange multipliers are added to constraints and objective functions automatically. However, you can still declare your own multipliers (like lambda_c in the example above).⁴ A relevant equation should be followed then by a colon (:) and a corresponding Lagrange multiplier's name (followed by square brackets in the same way as other model variables).

Having constructed the first block of our model, let us now move on to the second optimising agent i.e. a representative firm. We will call its block Firm. Firm's block will consist of sections: controls, objective and constraints,

 $^{^3\}mathrm{Semicolons}$ after sections and blocks were mandatory up to the 0.4.0 version.

⁴Explicitly declaring Lagrange multipliers may prove useful in models in which multiplier of one agent appears in other model blocks (e.g. RBC model where the representative firm owns capital and there is no principal-agent problem). One can also explicitly declare Lagrange multipliers which have interesting economic interpretation (e.g. Tobin's q in the RBC model with capital's installation costs).



and calibration (pinning down parameter α). A properly written block for a representative firm looks as follows:

```
block FIRM
{
    controls
    {
        K_d[], L_d[], Y[], pi[];
    };
    objective
    {
        PI[] = pi[];
    };
    constraints
    {
        Y[] = Z[] * K_d[]^alpha * L_d[]^{(1 - alpha)};
        pi[] = Y[] - L_d[] * W[] - r[] * K_d[];
    };
    calibration
    {
        r[ss] * K_d[ss] = 0.36 * Y[ss] \rightarrow alpha;
    };
};
```

As one can infer from code snippets above, parameter values can be set in two ways in gEcon. In fact, gEcon distinguishes between two sorts of parameters: free and calibrated ones. While the first have their values assigned arbitrarily, the latter can be calibrated in process of solving for the steady state of the model — based on information about relations between parameters and steady-state values of variables. To grasp the difference, look at the code snippets above. The calibration section in the block Consumer contains free parameters only, while the parameter alpha in the block Firm is an example of a calibrated parameter. Its value will be determined in the process of solving the model based on a steady-state capital share in product.

How to include parameters in the model in gEcon depends on the type of parameters we are dealing with. gEcon gives flexibility with respect to free parameters, which may be either declared in calibration section in a .gcn file (like parameters in the block Consumer above) or omitted and set there while solving the model in R. However, even if set in a file, they can still be overwritten in R later. In contrast, calibrated parameters have to be declared in a .gcn file in the calibration section (like alpha in the block Firm above), however one may set their values later in gEcon, by switching off the calibration facility. The functionalities concerning parameters and variables available in R will be explained in detail in section 1.4 of this chapter and in the chapter 8.

Returning to our example model, in order to close it we need a block with market clearing conditions which we will call Equilibrium. Such a block will contain the identities section only. Although we have listed three equations for market clearing conditions in section 1.1, we need to put only two of them in the Equilibrium block. The third one, clearing goods markets, will be *implicite* taken into account by Walras law — it can be derived from other equations.⁵ The Equilibrium block of our model is presented in the following code snippet:

```
block EQUILIBRIUM
{
    identities
    {
        K_d[] = K_s[-1];
        L_d[] = L_s[];
    };
};
```

 $^{^5 \}mathrm{See}$ e.g. [Mas-Colell et~al. 1995].

Exogenous variables and shocks to the system should (but do not have to) be defined in gEcon in a separate block. Exogenous shocks will be listed in shocks section. As our model described above contains only one exogenous variable, one shock, and the relevant block — called here Exog — will be quite simple:

block EXOG

```
{
    identities
    {
        Z[] = exp(phi * log(Z[-1]) + epsilon_Z[]);
    };
    shocks
    {
        epsilon_Z[];
    };
        calibration
    {
            phi = 0.95;
    };
};
```

This completes formulation of our model. However, it contains some redundant variables, e.g. by market clearing conditions supply of production factors is equal to the demand for them. Additionally, we have explicitly named the Lagrange multiplier on the budget constraint, but it is not used anywhere else in the model. Moreover, because of the perfect competition assumption the profits of firms in the model will be 0. These remarks lead to a conclusion that five variables can be eliminated from the model: K_t^d , L_t^d , λ_t^c , π_t , and Π_t . **gEcon** offers automatic reduction of model variables. To use this feature you have to list the variables in question within the **tryreduce** section of the **.gcn** file, just before the first model block. This is shown in the following listing:

We have written all sections of our model in gEcon language. Now just put together the tryreduce section and the four blocks, save it as a .gcn file, say rbc_ic.gcn, and that's it! Once the whole model described in section 1.1 has been written properly in the gEcon language, it is ready to be loaded and solved from R by gEcon.

The entire code for this example can be found on the gEcon website at http://gecon.r-forge.r-project.org/.

1.3 Reading model from R

In order to read the model from R, assuming you have installed the gEcon R package (for instructions see chapter 2), you need to do just two things:

1. First of all, you have to load the gEcon package in R, running:

library(gEcon)

2. Secondly, you should use the make_model function, taking as an argument the path and the name of the .gcn file you have created. Assigning the return value of this function to a desired variable in R, you will obtain an object of the gecon_model class, which can be further processed with the functions from the gEcon package. To illustrate this, for our example model the command:



Figure 1.1: gEcon workflow

rbc_ic <- make_model("PATH_TO_FILE/rbc_ic.gcn")</pre>

will create an object named rbc_ic (of class gecon_model) in our workspace in R.

The make_model function first calls dynamic library implemented in C++ which parses the .gcn model file. Then, first order conditions are derived, on the basis of an algorithm described in chapter 6. Matrices are derived after collecting all model equations, steady state equations, and linearisation, which will be later used to determine steady state and approximate equilibrium laws of motion around it. It is worth mentioning that apart from saving appropriate information in a newly returned gecon_model object, the make_model function generates a .model.R output file containing all the derived functions and matrices constituting a model. A .model.R output file is saved in the same directory in which the .gcn file has been saved. In addition, gEcon can automatically produce a draft of LATEX documentation of the model and a logfile, which allow user to check model's automatically derived first order conditions as well as its equilibrium and steady-state relationships. This gEcon functionality is described in the section 1.7. gEcon workflow is presented in figure 1.1 below.

This is only a short description of the process of preparing a model for solving it in R. Further details concerning the class gecon_model and the derivation of FOCs can be found in chapters 7 and 6. In general, all models' elements are held in appropriate slots of gecon_model objects. Functions provided for solving and analysing models (described in detail in chapters 8-11) are methods of this class and usually change relevant slots for further use or retrieving information from them.

Having read our model into an object of the gecon_model class we can proceed to solve and analyze its static and dynamic properties.

1.4 FINDING THE STEADY STATE

As mentioned above, in the process of creating a model object in R steady-state relationships are derived.

A basic gEcon function for finding the steady state of a model offering users interface to non-linear solvers, is the steady_state. However, before using it, you should make sure that you have assigned your desired values to all free parameters in the model. If you skip this step and some free parameters remain unspecified, gEcon will produce an error message. As described in the section 1.2 you can assign values to free parameters either in a model file — just as we did declaring values of parameters beta, delta, eta, mu, psi and phi in our example rbc_ic.gcn file — or using the set_free_par function in R. If we had not declared values of free parameters in our .gcn file we could do this now in R using the function set_free_par. Doing both, you will overwrite the values from the file with the values passed to R. So, taking an example of our model, running now a command:

```
rbc_ic <- set_free_par(rbc_ic, list(eta = 3, mu = 0.2))</pre>
```

would change values of the parameters eta and mu. You could reverse this by setting a logical argument reset to TRUE in set_free_par, i.e. running:

```
rbc_ic <- set_free_par(model = rbc_ic, reset = TRUE)</pre>
```

KEEP IN MIND

Most functions in the gEcon package in R have many different parameters, whose values can be changed. In order to see a complete documentation with a full list of options available for any function, you should call its name preceded by ? or ??, e.g. ?set_free_par or ??steady_state.

Since the values of all free parameters appearing in our example model have been set in a .gcn file you may try to find its steady state, without invoking the set_free_par function. In order to do this, you should use the steady_state function and run:

rbc_ic <- steady_state(rbc_ic)</pre>

After invoking this code you should see **Steady state has been FOUND** printed on the console. The **steady_state** function has some additional arguments controlling non-linear solver behaviour (for a complete list of arguments available type **?steady_state**).

KEEP IN MIND

In order to make further use of computed results (e.g. obtained steady-state values) and information passed to the object of the gecon_model class (e.g. values assigned arbitrarily to parameters), it is crucial not only to run the functions but also to assign their return values to the model, i.e. the object of the gecon_model class. Only in this way a new information is stored and then you can proceed to further stages of solving and analysing the model.

Now, if you wish to see the results, i.e. computed steady-state values of our model's variables and calibrated parameters which have been computed, run use the get_ss_values and get_par_values functions:

get_ss_values(rbc_ic)
get_par_values(rbc_ic)

and you will have them printed on the console and returned as functions' values. The default option is to print (and return) the values of all the variables and parameters, unless you pass a list or a vector of a chosen subset to the functions. In case the steady-state solver has been started but has not converged, the functions return vectors of variables' and calibrated parameters' values from the last solver iteration. However, it is not the case of our example model.

It is worth mentioning that initial guesses of steady-state values which are close to final results usually improve the chance and speed of finding solution. You may pass initial values to model's variables and calibrated parameters by means of the initval_var and initval_calibr_par functions, respectively, and check them later with the get_init_val_var and get_init_calibr_par, respectively. However, a non-linear solver available in gEcon (through the steady_state function) often manages to find steady state for a model using values which are assigned to all variables and parameters by default — and this was the case of our example model rbc_ic.

So, we have computed the steady state for our model. Obviously, it is dependent on the values assigned to the free parameters, which you may change easily with the set_free_par function. However, gEcon offers you an additional functionality in terms of computing the steady state: an option to decide how you want to treat the parameters originally defined as calibrated ones without having to change a .gcn file.

In order to take advantage of this gEcon facility, a calibration option in the steady_state function should be used. It is a logical argument, which indicates if calibrating equations — provided they exist in a model should be taken into account in the computation of the steady state. If TRUE, which is its default value, calibrating parameters are treated analogously to variables and their value is determined based on calibration equations and this was the case with the alpha parameter in our example. However, if you set the calibration option to FALSE, alpha would be treated as a free parameter and its calibrating equation would be omitted while solving for the steady state. But you should not forget about assigning a desired value to it, say 0.4, which you can do by using the initval_calibr_par function — as switching off a calibration facility makes gEcon treat initial values of calibrated parameters as if they were the values of free parameters. In order to do this you should run the following code:

```
rbc_ic <- initval_calibr_par(model = rbc_ic, calibr_par = c(alpha = 0.4))
rbc_ic <- steady_state(model = rbc_ic, calibration = FALSE)</pre>
```

All the remaining options available in the **steady_state** function (except for **calibration**) refer to the process of solving the system of non-linear equations. Changing them may be especially useful while encountering troubles with finding the steady state. As we did not experience them with our example model, we do not devote more attention to this issue here. For more information see chapter 8 and in the gEcon package documentation.

1.5 Solving for dynamics

Now, having computed the steady state of our model, we can solve for its dynamics. As gEcon uses perturbation method for solving dynamic equilibrium models, your model will need to be linearised or log-linearised before it is solved.

However, with gEcon you will not have to do it by hand nor substitute natural logarithms for variables in your .gcn file. gEcon offers you the solve_pert function which, in short, linearises or log-linearises a model, transforms a model from its canonical form to a form accepted by solvers of linear rational expectations models, and solves the first order perturbation. For a detailed description of gEcon solution procedure see chapter 9 and the documentation of the gEcon package in R. To cut a long story short, all you need to solve our example model for its dynamic equilibrium is run the following line of code:

rbc_ic <- solve_pert(rbc_ic)</pre>

as we set all the function's argument but the first one to their default values. After invoking one of this code line you should see 'Model has been SOLVED' printed on the console.

You should note that we solved our example model in its log-linearised version, which is very convenient for further analyses as variables after log-linearisation may be interpreted as percent deviations from their steady-state

gÊcon

values. However, you may easily switch to solving the model in a linearised version — using the function's logical loglin argument (with a default value set to TRUE), which controls for the sort of perturbation's linearisation. If you set it to FALSE in the above function, i.e. run:

```
rbc_ic <- solve_pert(model = rbc_ic, loglin = FALSE)</pre>
```

then model would be linearised only. Apart from the option to choose or change the type of model's linearisation, gEcon offers you also the facility to diversify variables depending on the type of linearisation. After setting loglin = TRUE, you may declare a vector of variables that should be linearised only, by means of an not_loglin_var argument. So, if you wanted to have all the variables log-linearised in our example model except for, say, r, you should run:

```
rbc_ic <- solve_pert(model = rbc_ic, loglin = TRUE, not_loglin_var = c("r"))</pre>
```

and that's it!

In order to see the results of the first order perturbation you should use the get_pert_solution function, which prints (and returns if assigned to a variable) computed recursive laws of motion of the model's variables:

get_pert_solution(rbc_ic)

If you are interested in the eigenvalues of the system or checking Blanchard-Kahn conditions, which can be especially useful in debugging a model, you should make use of the check_bk function, which takes a model object as an argument. For more details on this function as well as the solve_pert function see chapter 9 and the documentation of gEcon package in R.

KEEP IN MIND

Invoking the following recap functions with a model object as an argument at every stage of solving and analysing your model with gEcon you will see:

- show basic information and diagnostics of the model and the stage of its solving process,
- print more detailed information and diagnostics of the model and the stage of solution process,
- summary the results of computations carried out so far.

1.6 Results — correlations and IRFs

Now, after we have solved the model, we can specify the structure of shocks, simulate it, and check if relationships between the variables or their reactions to shocks indicated by the model are consistent with the data. gEcon enables you to compute indicators most commonly used in RBC/DSGE literature, such as means, variances, correlations, or impulse response functions.

Once again, you do not have to change anything in the original .gcn file in order to perform stochastic simulations of the model and analyse its variables' properties. All you need to do is pass a shock covariance matrix to your gecon_model object and call a few gEcon functions.

In order to set the covariance matrix of shocks in a model you should use the set_shock_cov_mat function. Since in our example rbc_ic model there is only one shock, we will have an 1-element covariance matrix containing only the shock's variance. The following command sets covariance matrix to 0.01:



You can also set or change chosen elements in the covariance using the set_shock_distr_par function. This function allows to work with easily interpretable parameters, such as standard deviations and correlations instead of whole covariance matrix. In order to do the same as above by using the set_shock_distr_par function, you should run:

This function is described in detail in chapter 10.

Having set the covariance matrix of shocks, we can compute the statistics of model's variables and correlation matrices using the compute_model_stats function, by running the following command:

The function computes correlation matrices of variables' series, using spectral or simulation methods and, optionally, filtering series with the Hodrick-Prescott filter. The most of its options refer to the computation method chosen and its parameters which are described in detail in chapter 10 and the gEcon package documentation.

The function compute_model_stats computes the following statistics:

- basic statistics (means, standard deviations and variances of variables),
- correlations:
 - correlation matrix of all the model's variables,
 - correlations of variables with the reference variable and its lead and lagged values,
- autocorrelations correlations of variables with their own lagged values,
- variance decomposition ascription of variables' variability to different shocks,

from which we can subsequently choose only the information we are interested in. In order to have all the computation results stored for further use you should remember to assign the function return value to our object of the gecon_model class. If you want gEcon to compute additionally correlations of variables with the reference variable and relative standard deviations, you should pass a chosen variable's name through the ref_var argument, just as we did with Y in our example above. The n_leadlags option allows you to control for the number of lags in the autocorrelation table and leads and lags used for computation of correlations with the reference variable.

KEEP IN MIND

Changing values of any settings in an object of the gecon_model class that may impact results makes gEcon automatically clear the information which has already been stored and which could be affected by the changes. So, e.g. assigning new values to the parameters will clear all the information passed to the object after making the model, whereas changing values in a shock matrix will clear only the results of stochastic simulations. You should note that changing the values which could affect the steady-state results, e.g. free parameters, will force you to recompute the model but the steady-state values obtained prior to the change will be stored as new initial values of the variables.

gEcon

Now, if you wish to see the computed statistics for our example model, use the get_model_stats function and run:

get_model_stats(rbc_ic)

The function prints all the results by default. Naturally, you can choose for printing only some of the results available, setting the remaining ones to FALSE (see the gEcon package documentation in R for a complete list of arguments available). The function get_model_stats prints the results on the console and optionally returns them — if you assign its return value to any variable.

KEEP IN MIND

At every stage of analysing a model with gEcon you can get the information about its variables, parameters, and shocks by using the var_info, the par_info and the shock_info functions respectively. The first allows you to choose the subset of variables you are just interested in and see of which equations they are part of, whether they are state variables or not, as well as examine all the computation results concerning them. The second provides information about the incidence of parameters in the equations and calibrating equations, values, and types of the parameters. The third gives you an option to choose the subset of shocks of interest and see in which equations they appear and how their covariance matrix looks like.

Last but not least, you may want to analyse the impulse response functions (IRFs) of variables in your model. gEcon offers you this facility and in order to take advantage of it, you need to call the compute_irf function. It computes IRFs for requested sets of variables and shocks, returning an object of class gecon_simulation. It is important to assign the function to a new object, so as to have the results stored and make use of them. For example, if you want to compute IRFs for the variables C, K_s , Z, Y, L_s and I of our rbc_ic model, you should run the following:

As gEcon stores information concerning IRF in another class, a newly created object — rbc_ic_irf — will be of gecon_simulation class. The path_length argument allows you to specify the number of periods for which IRFs should be computed. All the options of this function are described thoroughly in chapter 10 and in the gEcon package documentation.

Now, if you call the plot_simulation function:

plot_simulation(rbc_ic_irf)

you will see the IRFs for the specified variables plotted. This function has a logical argument to_eps, and if you set it to TRUE, i.e. call:

plot_simulation(rbc_ic_irf, to_eps = TRUE)

the IRFs will be saved on your disk — in the plots subfolder created in the directory where the rbc_ic.gcn file has been saved.

1.7 Automatic generation of model documentation in IAT_{FX}

gEcon can automatically generate a draft of model documentation (optimisation problem, constraints, identities, FOCs, and steady-state equations). To use this feature you only have to include the following lines at the beginning



of your model file:

options
{
 output LaTeX = TRUE;
};

On successful call to make_model a LATEX document named just as your model file (with extension .tex) will be created. For details, see 3.3.2.

Additionally, gEcon offers the facility of saving the results. This functionality is described in section 11.5.

KEEP IN MIND

All the files created in the process of making, solving, and analysing a model in gEcon are saved in the same directory in which the original .gcn file has been saved.

2 INSTALLATION INSTRUCTIONS

2.1 Requirements

gEcon requires R version >=4.4.3 with the following packages: Matrix, MASS, nleqslv, Rcpp and methods. gEcon has been tested on Windows (32-bit and 64-bit), Linux (32-bit and 64-bit), but it should also run on other systems on which R works. Under Windows, Rtools may be necessary to compile model equations in C++/Rcpp if such an option was enabled.

2.2 INSTALLATION

In order to use gEcon you should install gEcon R package. You can do this in two ways:

- through the command line interface¹ after changing a current working directory to the folder where gEcon package has been saved, it is sufficient to run a command:
 > R CMD INSTALL gEcon_x.x.tar.gz (source code Un*x/Linux), or
 > R CMD INSTALL gEcon_x.x.zip (Windows binary)
- directly from R using installation options available in the GUI used.

Note: In general, Windows users should use a precompiled binary package (if made available) with the extensions .zip. If you wish to build a package from source under Windows or generate model equations in C++/Rcpp you have to install Rtools first.

Note: When installing gEcon from source you might see (depending on compiler settings) some compiler warnings. If such appear, please ignore them.

2.3 Syntax highlighting

Syntax highlighting is a very useful feature of many advanced text editors. Currently, gEcon provides users with highlighting configuration files for two editors: Notepad++ (under Windows) and Kate (under Linux with KDE). After installing the gEcon package, start R session, load the gEcon package and type:

> path.package("gEcon")

This will show you where gEcon has been installed. Syntax files will be found there, in the syntax subdirectory.

¹Under Windows you start the command line by executing cmd.exe.

2.3.1 Notepad++

Start Notepad++ and go to the menu Language -> Define your language.... In the popup window choose Import. Go to the gEcon installation path, then subdirectory syntax, and choose the gEcon_notepadpp.xml file. Press the button Ok if import is successful and restart Notepad++.

2.3.2 Kate

Go to the subdirectory syntax in the gEcon installation path. Copy the gEcon_kate.xml file to the directory: ~/.local/share/katepart5/syntax where ~ denotes your home directory. Restart Kate.

Note: It may be necessary to create katepart5/syntax subdirectory within ~/.local/share/ directory.

2.4 Examples

Sample models (.gcn files) are distributed with gEcon. First check where gEcon was installed by typing (after loading the gEcon package):

> path.package("gEcon")

In the examples subdirectory you will find some sample models.

3 MODEL DESCRIPTION LANGUAGE

3.1 SYNTAX BASICS

3.1.1 Numbers

gEcon supports both integers and floating point numbers. Integers other than 0 may not begin with the digit 0. Valid integer token should be 0 or match the following pattern:

[1-9][0-9]*

Floating point numbers have decimal sign (.) preceded or followed by digit(s). Notation with exponents is also allowed as in 2.e-2. Valid floating point number token should match any of the three patterns:

[0-9]\.[0-9]+([eE][+-]?[0-9]+)? [0-9]+\.([eE][+-]?[0-9]+)? \.[0-9]+([eE][+-]?[0-9]+)?

3.1.2 VARIABLES AND PARAMETERS

Each variable and parameter name should begin with a letter. gEcon is case sensitive and supports Latin alphabet only. Digits are allowed in names (after initial letter). Underscores are allowed only inside variable/parameter name and should not be doubled. The regular expression for a valid parameter/variable name is:

[a-zA-Z](_?[a-zA-ZO-9])*

Underscores are used to divide variable/parameter names into sections which in LATEX output are put as upper indices. Greek letters are parsed in LATEX output properly. For instance delta_K_home is a valid gEcon parameter name and becomes $\delta^{K^{home}}$ in LATEX output.

Variables are represented by their names followed by square brackets ([]) possibly with an integer inside and parameters are represented solely by the names. Using the same name for a parameter and a variable is an error. Empty brackets denote value of a variable at time t, any index inside brackets is relative to t. For example Pi_firm_home[1] is a valid gEcon variable name and becomes $\Pi_{t+1}^{\text{firm}home}$ in LATEX output. However, while current version of gEcon allows to include all the variables in any lags, it does not allow to declare variables in leads > 1. Only variables denoting the objective functions as well as the exogenous variables are allowed to appear in models in leads (of a maximal value equal to 1, though).

A name followed by time index [ss], [SS], [-inf], [-Inf] or [-INF] denotes the steady-state value of a variable.

Since the introduction of template mechanism to the language (see chapter 4) variables and parameters can be indexed, for details see section 4.2.

3.1.3 Reserved Keywords

The following keywords are reserved in gEcon language:

General equilibrium economic modelling language and solution framework



3.1.4 Comments

gEcon supports single line comments beginning with #, %, or //.

3.1.5 Functions

Currently, the following functions are available in gEcon:

sqrt (square root)		
exp (exponential function)	log (natural logarithm)	
sin (sine)	cos (cosine)	tan (tangent)
asin (arc sine)	acos (arc cosine)	atan (arc tangent)
sinh (hyperbolic sine)	cosh (hyperbolic cosine)	tanh (hyperbolic tangent)
pnorm (cdf of the standard normal distr.)		

Function names cannot be used as variable or parameter names.

Function arguments should be enclosed in parentheses as in exp(Z[]).

3.1.6 ARITHMETICAL OPERATIONS

gEcon supports four basic arithmetical operations (+, -, *, /) as well as powers (^). Please note that power operator is right associative (as in R but not Matlab), i.e. 2^3^2 is equal to 512 and not 64.

The natural precedence of arithmetical operators can be changed by parentheses as in 2 * (3 + 4).

3.1.7 CONDITIONAL EXPECTATION OPERATOR

 $\verb|gEcon|$ uses the following convention for conditional expectation operator:

E[lag][expression],

where lag may be an integer or empty field implying expectation conditional on information at time t. For example, E[][U[1]] is understood as $E_t[U_{t+1}]$.

It should be noted that in gEcon all of the leading variables (in relation to time t) appearing in stochastic models have to be put under a conditional expectation operator.

gEcon

3.2 ORGANISATION OF gEcon INPUT FILE

A model file should be divided into block(s) corresponding to optimising agents in the model and block(s) describing equilibrium relationships. Each model file must have at least one model block. Additionally, any .gcn file may begin with the options block, determining the behaviour of the gEcon library. The options block can be followed by the indexsets block (for details see section 4.1) and tryreduce block containing a list of variables selected for symbolic reduction.

In model blocks describing economic agents and equilibrium relationships the **block** keyword has to be followed by a block name, which should obey the same naming rules as parameter/variable names. The contents of any block should begin with an opening brace ({) and be closed by a brace (}) which can be followed by a semicolon (;). Semicolons after closing braces are not mandatory.

A typical gEcon model file would look as follows:

```
options {
     . . .
};
indexsets {
     . . .
};
tryreduce {
     . . .
};
block Name1 {
     . . .
};
. . .
block NameN {
     . . .
};
```

3.3 Options

A set of options determines the behaviour of the gEcon package. The general form of (un)setting an option is: option = Boolean value ;

Accepted as Boolean values are true, TRUE, false, and FALSE.

The following set of options is currently supported:¹

silent	output	logfile	output	R	(ignored)	output	LaTeX	
verbose			output	R	long	output	LaTeX	long
warnings			output	R	Jacobian	output	LaTeX	landscape
			output	R	Rcpp			

¹Option backwardcomp, which was present, but ignored in recent gEcon releases, was removed in version 1.1.



3.3.1 GENERAL OPTIONS

The silent option (set to false by default) makes gEcon suppress display of messages related to model construction. On the other hand, the verbose option (set to false by default) makes gEcon print detailed information about the model construction process. If both are set to true, verbose option overrides silent.

The warnings option (true by default) controls display of warning messages.

3.3.2 CONTROLLING gEcon OUTPUT

Aside from an .model.R file produced by a call to the make_model function, gEcon can generate LATEX documentation and text logfiles for the model. A particular type of output can be turned on/off by:

output output_type = Boolean value ;

Admissible output types are: logfile and LaTeX (or latex). gEcon actually admits an option output R but this option is ignored. By default additional output files are not created.

Each type of output can have some additional settings (detailed in the following paragraphs). Specific properties of a given output type can be set via:

output output_type output_property = Boolean value ;

R OUTPUT FILE

gEcon writes the steady state and perturbation equations to a .model.R file without using the names of variables, but using their indices instead. In this way, the output file size as well as the time needed by R to parse the model are reduced. However, the user may force gEcon to print full names by setting output R long = true, which makes the generated .model.R file larger, yet human readable.

By default, gEcon derives Jacobian of steady-state/equilibrium equations to be used by the steady_state function (see section 8.3). In case of large models (say, 1000+ variables and calibrated parameters), derivation of Jacobian can be time consuming and use significant memory resources. Users may disable Jacobian derivation by setting output R Jacobian = false.

Since version 1.2.0, gEcon offers an option to generate model equations in C++, compile them and make them available to gecon_model objects through Rcpp. The users may enable this option by setting output R Rcpp = true. The generated C++ code is compiled when the make_model function is called. When either the steady state or the perturbation is solved, the gecon_model objects call the compiled C++ functions.

Note: The effect of the **output R Rcpp** option on total computation time may vary depending on a model size. Two factors play a role here: compilation time and function call time. Since version 3.5.0, R has the JIT compiler enabled (hence even R functions are compiled). This compiler, written in R, works effectively for smaller functions but lacks the scalability especially for very large functions. Models consisting of many thousands of equations may not be solved because JIT compilation uses up all available RAM memory leading to a crash. Compilation of C++ code works much faster and more effectively for large models. Moreover, when function call times are compared, C++ functions are generally 2-10 times faster than their R counterparts compiled by the JIT compiler. On the other hand C++ compilers introduce some overhead for smaller models. Consequently, the compilation of C++ functions may lengthen creation of small models. This will negatively affect total computation time as small models can be solved quickly with the R compiler. This is one of the reasons why gEcon by default generates R rather C++/Rcpp functions. For medium size and especially large models, generation of C++ functions should speed up both compilation and execution processes.

Logfile

When the option ouput logfile is set to true, gEcon generates (on successful call to the make_model function) a text logfile containing all information about the model (optimisation problems, FOCs, derived equilibrium and steady-state equations, information about variables and parameters).

Note: A (partial) logfile is always written (irrespective of the **ouput logfile** setting) on errors that occurred during derivation and collection of model equations. Logfile is also generated when all warnings cannot be displayed. Inspection of this file may prove helpful for debugging purposes.

$\mathbb{L}\!\!\!^{A}\!T_{E}\!X \text{ output file}$

When LATEX output is turned on (via output LaTeX = true setting), then on a call to the make_model function three .tex files are created (in the same directory in which the .gcn file is located):

- model_name.tex the main LATEX file taking the two remaining files as inputs,
- model_name.model.tex a draft of the model documentation (optimisation problems, FOCs, final model equations, etc.),
- model_name.results.tex the file to which model results are written from R interface level (for details see section 11.5).

In some models (especially those with complicated consumer's utility function) equations can become pretty long and will not fit within the page in the IAT_EX output. In such cases setting the output LaTeX landscape option to true (false is the default) will make gEcon create IAT_EX document with equations printed on pages oriented horizontally.

Given models written using the gEcon template mechanism (for details see chapter 4), LATEX documentation files may turn out to be quite lengthy. An output file can be shortened by setting the option output LaTeX long to false (by default it is set to true) — a model is documented then in a "templated" form instead of an expanded one.

3.3.3 AN EXAMPLE

The following code makes gEcon print diagnostic messages, create IAT_EX documentation file, a text logfile, and a .R file with full names of variables.

```
options {
    verbose = true;
    output latex = true;
    output logfile = true;
    output R long = true;
};
```

The second example turns on C++ code generation while preventing gEcon from generating Jacobian. This is the recommended setting for large (500+ variables) CGE models, for which R JIT compilation takes too long and Jacobian storage may consume too much memory.

```
options {
    verbose = false;
    output latex = false;
    output R Rcpp = true;
    output R Jacobian = false;
};
```



3.4 VARIABLE REDUCTION

Handbook formulation of general equilibrium model may introduce many variables which are redundant from the computational point of view, e.g. Lagrange multipliers, both supply and demand for a particular good, which are equal by the market clearing conditions. In the example from chapter 1, the demand and supply for production factors are equal (equations (1.9) and (1.10)). One of the variables for each factor can always be eliminated from equilibrium conditions. Reduction of the number of variables (and equations) improves the chance of finding the steady state and reduces computational complexity of steady-state and perturbation solution.

The standard approach is to reduce variables manually, but gEcon can assist the user in this task. It employs symbolic reduction algorithm in order to try and eliminate the variables requested by the user. This takes off the burden of reformulating equilibrium equations and reduces the risk of making a mistake.

gEcon tries to reduce all the internally generated variables (Lagrange multipliers and lagged variables). This is done in a two-stage process. Internally generated Lagrange multipliers are checked for the possibly of reduction just after the FOCs are determined. At this stage Lagrange multipliers are reduced only if they can be substituted with an expression without any variables in leads or lags. The second stage reduction takes place after all the equations in the model have been collected.

User-declared variables can be selected for reduction within the tryreduce block. Variables listed in this block have to be separated by a comma (,) and the list must be closed with a semicolon (;). The reduction of user-declared variables takes place in the second stage, i.e. after all equations have been collected. The following code, added to the code of the model from chapter 1, generates output with variables $L_t^{(d)}$ and $K_t^{(d)}$ substituted with $L_t^{(s)}$ and $K_t^{(s)}$ in the equilibrium conditions:

tryreduce {
 L_d[], K_d[];
};

Note: In general, objective functions cannot be reduced symbolically as they involve recursive formulation. On the other hand, in most cases they appear in one equation only and make finding the steady state more difficult. Because of that, gEcon also allows for reduction of variables appearing in one equation only (even if they cannot be solved for) — by simply removing the variable and the equation from the model.

Note: When using gEcon template mechanism, variables listed for reduction have to be properly indexed, for details please refer to section 4.3.1.

Note: When specifying the variables for reduction, the user has to be cautious about the timing of variables. In the example above, one should not specify the $K_t^{(s)}$ for reduction, as the model generated in this way would have a sunspot solution. This is because the new state variable representing the capital stock $(K_t^{(d)})$ would not appear in the model equations in lag, but in lead. Such formulation of the model is not compatible with the rational expectations solver used in gEcon.

3.5 Model blocks

Each model block describes one type of optimising agent in a model economy or set of equilibrium identities.

Model blocks are divided into sections. Each block must at least have a pair of controls and objective sections or identities section. All other sections are optional. Sections must be ordered as follows: definitions, controls, objective, constraints, identities, shocks, calibration.

Each section begins with a keyword (from the list above) and an opening brace ({). Sections are closed by a closing brace (}) and optionally a semicolon (;).

3.5.1 Definitions

This section is optional. Every definition should be of the form:

variable = expression;

or

parameter = constant expression;

Expressions on the right hand side are substituted for variables/parameters on the left hand side in all the remaining sections within a block. It may be useful for example to use u[] for instantaneous utility of a consumer or pi[] for firms profit in a given period after previously defining them in the **definitions** section. Variables or parameters that are 'defined' in this way in one block will not be substituted in other blocks. Thay cannot be declared as controls or shocks within a given block.

A sample definitions section might look as follows:²

definitions {
 u[] = b / e * log(a * C_m[]^e + (1 - a) * C_h[]^e) + (1 - b) * log(1 - N_m[] - N_h[]);
};

If more than one variable/parameter are defined in the section, relevant expressions are substituted in order of their definitions. It should be noted that a variable which has been already 'defined' cannot appear on the right hand side of the consecutive definitions.³

3.5.2 Optimizing agent sections

Each block describing optimisation problem should have controls section with a list of control variables and objective section with agent's objective function. Constraints on the problem should be listed in an optional constraints section.

gEcon forms the Lagrangian for each agent, based on objective function and constraints. Lagrange multipliers are created automatically⁴ and then reduced after first order conditions derivation (if possible). Users can still declare multipliers on their own and use them in model equations. There is only one restriction: multipliers cannot be declared on time aggregator in static optimisation problems (e.g. in the firm's problem from section 1).

Note: It is recommended not to specify Lagrange multipliers manually (if it is not necessary). If a model file contains multipliers specified by the user, a larger system of equations determining steady state will be generated and numerical solver may not find solution of the system given initial values used so far. Explicitly declared multipliers will not be reduced unless listed in tryreduce block (see section 3.4).

CONTROL VARIABLES

Control variables list must contain at least one variable and be finished with a semicolon (;). Variables should be separated by a comma (,). Time index of all control variables in a list must equal 0.

A sample controls section may look as follows:

Y[] = K[]^alpha * EL[]^(1-alpha); EL[] = A[] * L[]; };

changing the order of the equations will cause an error.

⁴Naming convention for automatically generated multipliers is $\lambda^{\text{NAME}_OF_BLOCK^i}$, where *i* stands for the number of constraint in a given block.

 $^{^2\}mathrm{This}$ is an example from consumer's block in a RBC model with home production.

 $^{^{3}}$ E.g. whereas it is allowed to define variables Y[] and EL[] as follows:

definitions {



```
controls {
    K[], L[], Y[];
};
```

gEcon does not allow objective variables and Lagrange multipliers of agents to be control variables of other agents. It also warns when the same variable is a control variable in two different problems. However, in some cases (optimisation subject to optimal actions of other agents, eg. in Nash bargaining or Ramsey problems) such constructs are necessary. To this end, gEcon provides the so-called "reference mechanism". In general, reference to a control, objective variable, or Lagrange multiplier of other agent has the following form:

variable @ referenced_block_name

Only previously declared blocks can be referenced.

OBJECTIVE FUNCTION

gEcon automatically derives first order conditions for dynamic problems with objective function given in a recursive manner and for static ones (see chapter 6).

Objective function should be provided in the following way:

objective_variable = time_aggregator_expression;

or alternatively with a Lagrange multiplier explicitly specified by the user:

objective_variable = time_aggregator_expression : Lagrange_multiplier;

Multipliers should not be declared in static optimisation problems. The time index of both the objective function and the Lagrange multiplier should be 0. Time aggregator expression may contain expected value of some variables and objective function in lead 1 conditional on information at time t. Objective function may appear on the right hand side only in lead 1. Objective variable of one agent cannot be objective or control variable of any other agent.

A sample objective section may look as follows:⁵

```
objective {
    U[] = log(c[]) + beta * E[][U[1]];
};
```

Constraints

Economic problems involve different sorts of constraints on optimisation problems of agents. Constraints are expressed in the gEcon language in the following fashion:

expression = expression;

or alternatively with a Lagrange multiplier explicitly named by the user:

expression = expression : Lagrange_multiplier;

A sample constraints block might look as below:⁶

```
constraints {
    I[] + C[] = r[] * K_s[-1] + W[] * L_s[] + pi[] : lambda_C[];
    K_s[] = (1 - delta) * K_s[-1] + I[];
};
```

 $^{^{5}}$ This is consumer's problem with exponential discounting and logarithmic utility from consumption.

⁶This is the budget constraint of a representative consumer/household owning and supplying capital and labour.

In some cases, optimisation is done subject to optimal actions of other agents or subject to identities from other block, eg. in Nash bargaining or Ramsey problems. Referencing other agent's (block's) objective variable declaration, constraints, identities and first order conditions is necessary in these situations. In order to automatically add the aforementioned equations to the constraint list one has to use the "referencing mechanism" as follows:

objective @ referenced_block_name ;

constraints @ referenced_block_name ;

identities @ referenced_block_name ;

focs @ referenced_block_name ;

The **focs** keyword stands for first order conditions.

Lagrange multipliers are inserted by gEcon automatically and cannot be explicitly named by the user in this case. Only previously declared blocks can be referenced.

3.5.3 Identities

If controls and objective sections are not present in a block this section becomes mandatory. Identities are simply equations that hold in any time at any state. This block is especially useful for market clearing conditions or description of exogenous (to the agents) processes. For instance, first order conditions derived manually may be entered into the model as identities.

Identities are given in a simple way:

expression = expression ;

A very simple identities block with a market clearing condition is given below:

```
identities {
    L_d[] = L_s[];
};
```

3.5.4 Shocks

Shocks are exogenous random variables. Since it is technically impossible to infer from model equations which variables are exogenous, shocks have to be declared by the user. The **shocks** section serves this purpose. Shocks should have 0 time index and must be separated by a comma (,). A complete shock list must be closed with a semicolon (;).

When shocks are used in expressions they should also have 0 time index.

A sample declaration of two shocks $\epsilon_t^1, \epsilon_t^2$ is listed below:

```
shocks {
    epsilon_1[], epsilon_2[];
};
```

gEcon assumes shocks to have a joint normal distribution with zero expected value. Shock distribution parameters can be set at the R interface level as described in section 10.1.

3.5.5 Calibration

There are two types of parameters in gEcon: so-called free parameters, which value may be changed by the user at R level and do not have to be set in the model file, and calibrated parameters. The values of calibrated parameters are

determined alongside the steady-state values of variables based on steady-state relationships. Calibrating equations and free parameter values are provided by the user in the calibration section.

Free parameter values are set as follows:

parameter = numeric_expression ;

The calibrating equation should contain parameters and/or the steady-state values of variables. As gEcon is unable to infer which parameters should be determined based on a given relationship, calibrating equation should be followed by the -> operator and a list of parameters. The syntax for calibrating equations is the following:

 $parameter_or_steady_state_expression = parameter_or_steady_state_expression \rightarrow parameter_1, \ ..., \ parameter_N \ ;$

A sample calibration block is presented below. Parameter β is a free parameter set to $(1.01)^{-1}$ and technology parameter α is calibrated based on the steady-state capital share in product:

```
calibration {
    beta = 1 / 1.01;
    r[ss] * K_d[ss] = 0.36 * Y[ss] -> alpha;
};
```

4 TEMPLATES

Most economic models used in applied work (especially CGE models) have many similar agents (firms, consumers) that solve problems of the same type and differ only in the value of some parameters. Writing such models using language features described in the previous chapter only is a tedious process and subject to high risk of making a mistake.

The problem of automatically replicating part of a code with different parametrisations has a long history in programming language design. Two solutions have been proposed and successfully implemented in many languages: preprocessor macros and templates (generic programming). The first approach, with the most prominent example of C language preprocessor, allows users to declare so-called macros which are expanded *before* (hence name) actual code is compiled (analysed). Such a two-stage process is easier to implement, but has fundamental flaws: code compiled (analysed) is different from what the user has actually written, any error in the initial code is multiplicated as many times as macro is expanded, which makes debugging difficult. In the context of gEcon language, the fact that expansion of the code takes place before analysis would mean deriving FOCs as many times as optimising agent block is replicated. Still, preprocessor macros offer a valid solution to an important problem and in the economic modelling software have been implemented e.g. in Dynare [Adjemian *et al.* 2013]. Templates (generic programming) have been introduced later (e.g. in C++ programming language) and address the aforementioned issues by extending the language in question instead of building on top of it. This means that "templated" code is analysed in the same way as regular code is and the process of "expansion" takes place at the end of compilation (code analysis). Such approach has been taken in the gEcon project. "Templated" (indexed) blocks are analysed only once and equations are expanded after equilibrium relationships (e.g. FOCs) have been derived.

4.1 INDEX SETS

4.1.1 Declarations

Before using template mechanism, the user has to properly declare sets over which the variables and parameters will be indexed. Such declarations should be placed in the indexsets section of the .gcn file. Each set declaration should be followed by a semicolon (;).

The syntax for standard declaration of a set is as follows:

set_name = { elements_list } ;

Valid set names should obey the same rule as valid variables and parameters names. The *elements_list* is a list of indices in a set quoted using single quotation marks (') and separated by commas (,). Valid index values may be formed by any combination of numbers and letters and single underscores (except for the beginning and the end), i.e. should match the following regular expression:

[a-zA-ZO-9](_?[a-zA-ZO-9])*

gEcon allows to generate sequences of letters or numbers that can be used for indexing. The sequences can be created by one of the following expressions:

{ number .. number }
{ capital_letter .. capital_letter }
{ small_letter .. small_letter }

gEcon

gEcon allows to create ascending sequences only. Numbers and letters should be quoted using single quotation marks ('). The sequences of numbers or letters can be concatenated with prefixes and suffixes using tilde (\sim) operator to form more meaningful names.

A sample indexsets block, in which a set of three sectors is declared in two ways described in this section, may look as follows:

indexsets

```
{
    SECTORS = {'sector_a', 'sector_b', 'sector_c'};
    SECTORS_STAR = 'sector_' ~ {'a' .. 'c'};
}
```

Sets cannot be redeclared. Redeclaration of any set will cause an error.

4.1.2 Set operations

New sets can be also created by performing set operations on other sets. gEcon supports three set operations: union (operator |), intersection (operator &), and asymmetric difference (operator \backslash).

The intersection operator has precedence over the union and difference operators. The latter operators are evaluated from left to right. The parentheses may override the default precedence.

Note: Tilde operator has precedence over set algebraic operators. The following expression presents this rule:

SECTORS_BAR = 'sector_' ~ {'a' .. 'e'} & 'sector_' ~ {'b' .. 'f'};

Two sets (from sector_a to sector_e and from sector_b to sector_f) are created initially and then gEcon finds their intersection (a set of indices from sector_b to sector_e).

4.1.3 Set validation

gEcon allows user to verify if the required sets have been declared properly by imposing some relation between two sets. All the validating expressions should be written in the indexsets section of .gcn file and be followed by a question mark (?).

gEcon evaluates the validating expressions and prints an error message if any of expressions turns out to be false.

A relation between two sets can be imposed by writing:

$set_A relation set_B$?

Three types of relations are supported by gEcon: equality (==), inequality (!=) and improper set inclusion (<=). In writing validating expressions an empty set (denoted by 0) may become useful.

An example

Consider a large-scale model of an open economy with two types of goods (sectors): tradables and non-tradables. These goods (sectors) will be indexed over two sets TRADABLE and NONTRADABLE. All goods (sectors) in the economy are indexed over set ECONOMY. One expects the following relations to hold for sets of tradable, non-tradable sectors, and the set of all the sectors in the economy:

- the tradable and non-tradable sectors are non-empty,
- the tradable and non-tradable sectors should be subsets of the set of all sectors in the economy,



- no sector can be both in tradable and non-tradable sets,
- sum of tradable and non-tradable sectors should yield the set of all sectors in economy.

These conditions may be stated in gEcon language as follows:

```
TRADABLE != 0? # tradables are non-empty
NONTRADABLE != 0? # non-tradables are non-empty
TRADABLE <= ECONOMY? # tradables are subset of all sectors
NONTRADABLE <= ECONOMY? # non-tradables are subset of all sectors
TRADABLE & NON_TRADABLE == 0? # empty intersection (intersection equal to the empty set)
TRADABLE | NON_TRADABLE == ECONOMY? # sum equal to all sectors in the economy</pre>
```

Note: It is highly recommended to make use of validation option whenever possible. If validation was not used in the example above and any of tradable sectors were misspelled, the number of equations in the model would be different than the number of variables, which would lead to an error. Such an error would be difficult to debug. If validation was used, gEcon would return information about the relations that are not satisfied.

4.2 INDEXED VARIABLES AND PARAMETERS

Indices of parameters and variables should be provided in gEcon within angle brackets (< and >). Multiple indices should be separated by commas (,). Indexed parameters are referred to as follows:

parameter_name<index_list>

Variables should be indexed as:

variable_name<index_list>[time_index]

gEcon makes distinction between fixed and free indices. This distinctions is pretty natural. Suppose x is a vector, then in expression denoting the *i*th element (x_i) *i* is a free index and 7 in expression denoting the 7th element (x_7) is a fixed one. The names of free indices should obey the same rules as the names of variables, parameters, and sets. Fixed indices should be quoted using single quotation marks ('). The examples below should make these rules clear:

```
alpha<s>  # parameter alpha indexed with free index s
alpha<'AGR'>  # parameter alpha indexed with fixed index 'AGR'
Y<c>[]  # variable Y (at time 0) indexed with free index c
Y<'PL'>[]  # variable Y (at time 0) indexed with fixed index 'PL'
EX<'PL',c>[]  # variable EX (at time 0) indexed with fixed index 'PL' and free index c
eta<'PL','DE'>  # parameter eta indexed with fixed index 'PL' and fixed index 'DE'
```

These expressions will be displayed in LATEX output as: $\alpha^{\langle s \rangle}$, $\alpha^{\langle AGR \rangle}$, $Y_t^{\langle c \rangle}$, $Y_t^{\langle PL \rangle}$, $EX_t^{\langle PL, c \rangle}$, $\eta^{\langle PL, DE \rangle}$.

In an object of gecon_model class (created through a call to make_model) the names of indexed parameters and variables are transformed so that they can be used in R easily (when setting initial values for steady state / equilibrium solvers, retrieving information about variables and parameters etc.). The general rule is that each (fixed) index is appended to the parameter / variable name after a double underscore (__). For instance:

alpha<'AGR'>, Y<'PL'>[], eta<'PL','DE'>

become:

alpha__AGR, Y__PL, eta__PL__DE

gEcon supports up to 4 indices for both parameters and variables.

4.3 INDEXING EXPRESSIONS

4.3.1 INDEXING VARIABLES AND EQUATIONS

Any expression involving free indices cannot be properly processed without knowing the sets to which the free indices belong. To connect a free index with an index set a so-called indexing expression should be used. The general form of such an expression is:

<index_name::set_name>

Suppose consumer chooses between many goods. Let us denote her consumption of good g as $C \leq []$. Goods belong to set GOODS. To list consumption of all the goods (e.g. in controls section) one should use the following expression:

<g::GOODS> C<g>[]

This is understood by gEcon as $\left(C_t^{\langle g \rangle}\right)_{g \in GOODS}$

Indexing expressions should also be used in equations that are supposed to hold for all indices belonging to some set. Again, let GOODS denote the set of all goods in the economy, C < g > [] consumption of good g. Let Y < g > [] denote production of good g. Market clearing condition (in closed economy) should then be written (somewhere in the identities section) in the form:

<g::GOODS> C<g>[] = Y<g>[];

This is understood by gEcon as $g \in GOODS: C_t^{\langle g \rangle} = Y_t^{\langle g \rangle}$.

gEcon currently accepts up to two indexing expressions preceding a variable or an equation except for the tryreduce block, where up to four indexing expressions are allowed.

In many applications, an index should run over all elements of a set but one. For example, total export of the *i*-th country is a sum of exports from the *i*-th country to all countries in the model but not to itself — the *i*-th country. Forcing the index not to take the value of another may be achieved in gEcon by using indexed expression with backslash operator (\backslash) followed by a free or fixed index:

<index_name::set_name\free_index> <index_name::set_name\'fixed_index'>

4.3.2 Sums and products

gEcon supports sums and products over indices belonging to some set. These operations are written in a natural way using indexing expressions:

SUM<index_name::set_name>(expression)
PROD<index_name::set_name>(expression)

The frequently used Cobb-Douglas and CES (constant elasticity of substitution) functions may be written using SUM and PROD as follows:

gEcon will understand these expression as:

$$CD_t = \prod_{f \in FACTORS} C_t^{\langle f \rangle^{\alpha^{\langle f \rangle}}}$$

and

$$C\!E\!S_t = \left(\sum_{g \in G\!OODS} share^{\langle g \rangle} D_t^{\langle g \rangle^{(\eta-1)}/\eta}\right)^{\eta/(\eta-1)}$$

Indexing expressions used in sums and products can involve skipping some index as in:

SUM<index_name::set_name\free_index>(expression)
SUM<index_name::set_name\'fixed_index'>(expression)
PROD<index_name::set_name\ free_index>(expression)
PROD<index_name::set_name\'fixed_index'>(expression)

As an example, recall the definition of total exports from the ith country as a sum of exports to all the countries except to itself (assuming COUNTRIES were declared as an index set):

<i:::COUNTRIES> EX<i> = SUM<j:::COUNTRIES\i>EX<i,j>

Sums and products follow standard rules. In particular, product over an empty set is taken to be 1, and sum over an empty set equals zero.

Note: The double sums, double products, and sum of products can be written without taking the argument into parentheses. However, one has to be vary about product of sums. In that case one has to use internal and external parentheses:

PROD<i::SET>(SUM<j::SET>(a<i,j>[]))

Otherwise gEcon will not parse the expression properly and an error will occur.

4.3.3 BLOCK TEMPLATES

The template mechanism in gEcon allows the user to write down general form of maximisation problems for similar agents, which are expanded automatically.

Blocks are expanded over the sets of indices. The indexing expressions must be placed after the **block** keyword but before the name of a block. The syntax is as follows:

```
block <i::SET_1><j::SET_2> name
{
    # sections
}
```

As a rule, indices from block declaration must be used for indexing variables in the definitions section, controls and objective variables (but do not have to be used in constraints nor identities). In the example above indices i and j must appear in the variables on left hand side in the definitions sections, objective variable and all control variables.

Index exclusion can be applied in block templates declarations just like in equations, sums and products.

The maximum number of indices in a block declaration is two.

4.3.4 POTENTIAL SOURCES OF ERRORS

STRAY INDICES

Consider the following equation:


SUM<i::SET\k>SUM<j::SET> X<i,j>[] = Y<i>[];

The equation above cannot be properly expanded without knowledge about the index k. If it is not assigned to any index set (in front of the equation or in the block declaration), gEcon will call it a "stray" index and will stop on error. All indices are checked before any further computations are performed.

MISSING INDICES

Consider the following example:

```
block <c::country> Consumer
{
    # other sections
    objective
    {
        U<c>[] = u<c>[] + beta * E[][U[1]];
    }
}
```

Here the objective (U[]) on the right hand side is missing the index c. gEcon treats U[] as a different variable than U < c > [] and the problem as *static* (no time aggregation, since U < c > [] does not appear on the right hand side). Errors of this type cannot be automatically diagnosed by gEcon. In most cases they will lead to different numbers of variables and equations or the inability to find steady state / equilibrium.

DUPLICATED INDICES IN NESTED INDEXING EXPRESSIONS

Another potential mistake when using template mechanism in gEcon can be made by using the same index twice in nested indexing expressions. Consider a bit contrived example:

```
block <a::SET_A> FOO
{
    # other sections
    identities
    {
        SUM<a::SET_B> B<a>[] = 0;
    }
}
```

Here **a** is the index in a sum within a block template parametrised with the same index. **gEcon** cannot tell whether **a** corresponds to the set **SET_B** in the sum or the set **SET_A** from the block declaration. Such code will cause an error.

4.4 The Kronecker delta and the rules of differentiation

The Kronecker delta is a double-indexed symbol returning one if indices coincide and zero otherwise:

$$\delta^{i,j} = \begin{cases} 1: i=j\\ 0: i \neq j \end{cases}$$
(4.1)

In what follows we will write the Kronecker delta using the standard indexing convention of gEcon, i.e. putting indices in angle brackets.

The Kronecker delta in gEcon is written as:

KRONECKER_DELTA<index_1, index_2>

The Kronecker delta of two fixed indices is automatically evaluated to 0 or 1.

The Kronecker delta makes writing rules of differentiation very simple. Suppose $x^{\langle i \rangle}$ is differentiated with respect to $x^{\langle j \rangle}$. We have:

$$\frac{\partial x^{\langle i \rangle}}{\partial x^{\langle j \rangle}} = \delta^{\langle i,j \rangle},$$

i.e. the derivative is 1 if indices are the same and zero otherwise. Given variables with two indices the derivative is equal to 1 if both indices coincide:

$$\frac{\partial x^{\langle i,j\rangle}}{\partial x^{\langle k,l\rangle}} = \delta^{\langle i,k\rangle} \delta^{\langle j,l\rangle}.$$

These rules can be generalised for arbitrary number of indices.

The rules of differentiation of sums and products are obvious:

$$\begin{split} \frac{\partial}{\partial x^{\langle j \rangle}} \sum_{i \in I} y^{\langle i \rangle} &= \sum_{i \in I} \frac{\partial y^{\langle i \rangle}}{\partial x^{\langle j \rangle}}, \\ \frac{\partial}{\partial x^{\langle j \rangle}} \prod_{i \in I} y^{\langle i \rangle} &= \left(\prod_{i \in I} y^{\langle i \rangle}\right) \left(\sum_{i \in I} \frac{1}{y^{\langle i \rangle}} \frac{\partial y^{\langle i \rangle}}{\partial x^{\langle j \rangle}}\right). \end{split}$$

Although rules are pretty clear, care should be taken when differentiating sums and products. Consider the following trivial example:

$$\frac{\partial}{\partial x^{\langle i \rangle}} \sum_{i \in I} x^{\langle i \rangle} y^{\langle i \rangle} = ?$$

Here the index i is used as an index of the variable with respect to which the derivative is taken but also as an index in the summation. Since summation is not changed with the change of indices and the derivative of a sum is the sum of derivatives, we can restate and solve our problem as follows:

$$\frac{\partial}{\partial x^{\langle i \rangle}} \sum_{i \in I} x^{\langle i \rangle} y^{\langle i \rangle} = \frac{\partial}{\partial x^{\langle i \rangle}} \sum_{i' \in I} x^{\langle i' \rangle} y^{\langle i' \rangle} = \sum_{i' \in I} \frac{\partial x^{\langle i' \rangle} y^{\langle i' \rangle}}{\partial x^{\langle i \rangle}} = \sum_{i' \in I} \delta^{\langle i', i \rangle} y^{\langle i' \rangle}.$$

The strategy outlined above is the one that gEcon uses when differentiating sums and products — sums and products are reindexed before differentiation whenever the indices collide. Indices created by gEcon have underscore appended (in LATEX the prime symbol '), so they will never coincide with any user-declared index.

In our example the result of differentiation was $\sum_{i' \in I} \delta^{\langle i', i \rangle} y^{\langle i' \rangle}$. If we knew that the index *i* runs over the same set *I*, the complicated expression could be reduced to just $y^{\langle i \rangle}$.¹ This type of "Kronecker delta reduction" is automatically done by gEcon and allows to obtain legible FOCs in problems written using sums and products.

¹In order to convince yourself that this really is the case, consider a simple example:

$$\frac{\partial}{\partial x^{\langle 1 \rangle}} \sum_{i' \in \{1,2,3\}} x^{\langle i' \rangle} y^{\langle i \rangle} = \frac{\partial}{\partial x^{\langle 1 \rangle}} (x^{\langle 1 \rangle} y^{\langle 1 \rangle} + x^{\langle 2 \rangle} y^{\langle 2 \rangle} + x^{\langle 3 \rangle} y^{\langle 3 \rangle}) = \frac{\partial x^{\langle 1 \rangle} y^{\langle 1 \rangle}}{\partial x^{\langle 1 \rangle}} + \frac{\partial x^{\langle 2 \rangle} y^{\langle 2 \rangle}}{\partial x^{\langle 1 \rangle}} + \frac{\partial x^{\langle 3 \rangle} y^{\langle 3 \rangle}}{\partial x^{\langle 1 \rangle}} = y^{\langle 1 \rangle} + 0 + 0.$$

$$\frac{\partial}{\partial x^{\langle 2 \rangle}} \sum_{i' \in \{1,2,3\}} x^{\langle i' \rangle} y^{\langle i \rangle} = \frac{\partial}{\partial x^{\langle 2 \rangle}} (x^{\langle 1 \rangle} y^{\langle 1 \rangle} + x^{\langle 2 \rangle} y^{\langle 2 \rangle} + x^{\langle 3 \rangle} y^{\langle 3 \rangle}) = \frac{\partial x^{\langle 1 \rangle} y^{\langle 1 \rangle}}{\partial x^{\langle 2 \rangle}} + \frac{\partial x^{\langle 3 \rangle} y^{\langle 3 \rangle}}{\partial x^{\langle 2 \rangle}} = 0 + y^{\langle 2 \rangle} + 0.$$

$$\frac{\partial}{\partial x^{\langle 3 \rangle}} \sum_{i' \in \{1,2,3\}} x^{\langle i' \rangle} y^{\langle i \rangle} = \frac{\partial}{\partial x^{\langle 3 \rangle}} (x^{\langle 1 \rangle} y^{\langle 1 \rangle} + x^{\langle 2 \rangle} y^{\langle 2 \rangle} + x^{\langle 3 \rangle} y^{\langle 3 \rangle}) = \frac{\partial x^{\langle 1 \rangle} y^{\langle 1 \rangle}}{\partial x^{\langle 3 \rangle}} + \frac{\partial x^{\langle 3 \rangle} y^{\langle 3 \rangle}}{\partial x^{\langle 3 \rangle}} = 0 + 0 + y^{\langle 3 \rangle}.$$

gEcon

4.5 An example — pure exchange model

A pure exchange model is a basic example of general equilibrium model. In our example there will be two agents (denoted by $a \in \{A, B\}$) and three goods (denoted by $g \in \{1, 2, 3\}$) in the economy. Each consumer is endowed with some amounts of three different goods (agent's *a* endowment of good *g* is denoted by $e^{\langle a, g \rangle}$). There are no production opportunities but the agents can freely trade with their endowments maximising utility $(U^{\langle a \rangle})$ from consumption $(C^{\langle a, g \rangle})$ denotes the consumption of good *g* by agent *a*) given by the Cobb-Douglas function (with parameters $\alpha^{\langle a, g \rangle}$):

$$U^{\langle a \rangle} = C^{\langle a,1 \rangle} \alpha^{\langle a,1 \rangle} C^{\langle a,2 \rangle} \alpha^{\langle a,2 \rangle} C^{\langle a,3 \rangle} \alpha^{\langle a,3 \rangle} = \prod_{g=1}^{3} C^{\langle a,g \rangle} \alpha^{\langle a,g \rangle}.$$
(4.2)

Each agent faces budget constraint $(p^{\langle g \rangle})$ is the price of good g:

$$p^{\langle 1 \rangle} C^{\langle a,1 \rangle} + p^{\langle 2 \rangle} C^{\langle a,2 \rangle} + p^{\langle 3 \rangle} C^{\langle a,3 \rangle} = p^{\langle 1 \rangle} e^{\langle a,1 \rangle} + p^{\langle 2 \rangle} e^{\langle a,2 \rangle} + p^{\langle 3 \rangle} e^{\langle a,3 \rangle}, \tag{4.3}$$

or equivalently:

$$\sum_{g=1}^{3} p^{\langle g \rangle} C^{\langle a,g \rangle} = \sum_{g=1}^{3} p^{\langle g \rangle} e^{\langle a,g \rangle}.$$

$$\tag{4.4}$$

All markets clear:

$$\sum_{a \in \{A,B\}} C^{\langle a,g \rangle} = \sum_{a \in \{A,B\}} C^{\langle a,g \rangle}, \ \forall g \in \{1,2,3\}.$$

$$(4.5)$$

The equilibrium for this economy is a set of prices $(p^{\langle g \rangle})_{g \in \{1,2,3\}}$ and allocations $(C^{\langle a,g \rangle})_{a \in \{A,B\},g \in \{1,2,3\}}$ such that the allocations maximise agents' utilities under budget constraints and markets clear. In equilibrium only relative prices are determined. For numerical solution, one of the prices has to be set as a *numeraire* (let us assume $p^{\langle 1 \rangle} = 1$). By the Walras law one of the market clearing conditions is redundant and will be omitted when writing model using gEcon.

The code snippet below presents the implementation of this model in gEcon. The naming convention for the variables, parameters, and indices corresponds to the model description above. Additionally, e_calibr<a,g> are parameters determining the initial endowments of agents. In EQUILIBRIUM section, the price of the first good (numeraire good) is set to 1. The market clearing conditions are given for all goods but first.

indexsets

```
{
    goods = { '1 ' .. '3 '};
agents = { 'A' , 'B' };
};
block <a::agents> AGENTS
{
    controls
    {
         <g::goods> C<a,g>[];
    };
    objective
    {
         U < a>[] = PROD < g:: goods>(C < a, g>[] ^ alpha < a, g>);
    };
    constraints
    {
         SUM \le :: goods > (p \le p) = SUM \le :: goods > (p \le p) 
    };
    identities
```



The formulation of the model is very compact but general. In fact, if the index sets were modified appropriately, one could obtain a pure exchange model for arbitrary n agents and m goods without any additional effort.

5 Model variants — Using the preprocessor

As described in chapter 4, gEcon provides users with a powerful template mechanism, which in the context of working with heterogenous-agents models renders preprocessor redundant. However, in some applications (e.g. CGE model calibration and, later on, policy experiment) one might want to analyse different variants of the same model with a few equations (or blocks) modified. Maintaining two (or more) versions of a model and keeping them synchronised is burdensome and risky, especially for more complicated models. In order to assist users in such situations in gEcon 1.2.0 preprocessor was introduced.

5.1 Declaring model variants

5.1.1 BASIC USE

Preprocessor directives, that is marks defining start and end of a model variant, have to be stated in a separate line beginning with two hash characters (##) followed at some point by two opening or closing curly braces ({{ or }}). In the simplest form, a variant beginning is marked as

variant_number {{

and variant end marked by

}}

gEcon supports up to 100 model variants with indices in range 0-99.

Variants can be commented using double slashes at the beginning-of-variant mark as in:

variant_number {{ // my variant description

If one wished to start a new variant just after another instead of using two lines (one to finish one variant and another to start a new one) like in:

0 {{ // my variant 0
 ... # code parsed when variant 0 is selected
}}
1 {{ // my variant 1
 ... # code parsed when variant 1 is selected
}}

one can use more compact notation as in:

```
## 0 {{ // my variant 0
    ... # code parsed when variant 0 is selected
## }} ## 1 {{ // my variant 1
    ... # code parsed when variant 1 is selected
## }}
```

5.1.2 Multiple nested or overlapping model variants

gEcon does not allow nested or overlapping variants. Any attempt to start a code variant before the previous

one was closed will lead to an error. However, one piece of code can belong to multiple model variants, effectively providing users with the same functionality as nested or overlapping variants.

As an example consider 3 model equations and 3 model variants. We would like the second equation to be present in variants 1 and 2 but not the 3rd one. Equation 1 is present in variant 1 only and equation 3 in variants 2 and 3. This can be achieved as follows:

```
## 1 {{ // variant 1
    ... # equation 1
## }} ## 1,2 {{ // variants 1 and 2
    ... # equation 2
## }} ## 2,3 {{ // variants 2 and 3
    ... # equation 3
## }}
```

Multiple variants can be listed either separated by comma (as in our example) or given as ranges using dash (-). The following declares a piece of code that will appear in model variants 1,3,4,5:

5.2 Selecting model variants

Particular model variant is selected using optional argument variant when calling make_model as in:

```
mymodel3 <- make_model("PATH_TO_MYMODEL/mymodel", variant = 3)</pre>
```

make_model function calls the preprocessor function and generates a new .gcn file (in this example mymodel_3.gcn), which is then parsed as described in section 1.3. The newly created .gcn file will have all code that belongs to variants other than 3 commented.

By default, names of generated .gcn files will have the underscore (_) and the variant numbers appended to their names. One can change this behaviour by setting the optional argument variant_name. In the following example variant 2 is selected and the generated file name will be mymodel_two.gcn.

```
mymodel2 <- make_model("PATH_TO_MYMODEL/mymodel", variant = 3, variant_name = "two")</pre>
```

5.3 An example — pure exchange model with different numéraires

Consider the simple pure exchange model from section 4.5. In our formulation we selected good 1 to be numéraire. What if we wanted to check 3 alternatives in which different good is a numéraire? The following code achieves this.

```
indexsets
{
    goods = { '1 ' .. '3 '};
    agents = { 'A', 'B' };
};
block <a::agents> AGENIS
{
    controls
```

General equilibrium economic modelling language and solution framework

```
{
                                            <g::goods> C<a,g>[];
                       };
                        objective
                        {
                                             U \leq a > [] = PROD \leq g :: goods > (C \leq a, g > [] \cap alpha \leq a, g > );
                       };
                       constraints
                       {
                                           SUM \leq :: goods > (p \leq g > [] * C \leq a, g > []) = SUM \leq :: goods > (p \leq g > [] * e \leq a, g > []);
                        };
                       identities
                       {
                                             <g::goods> e<a,g>[] = e_calibr<a,g>;
                       };
                       calibration
                       {
                                             \langle g::goods \rangle alpha\langle a, g \rangle = 1 / 3;
                        };
};
block EQUILIBRIUM
 {
                       identities
                       {
\# 1 { // 1 as numeraire
\begin{array}{l} \# & 1 \\ p < 1' > [] = 1; \\ \# & \} & \# & 2 \\ \# & 2 \\ p < 2' > [] = 1; \\ p < 2' > [] 
\# }} \# 3 {{ // 3 as numeraire
                                             p<'3'>[] = 1;
## }}
                                             # goods market clearing
                                             \langle g:: goods \setminus '1' \rangle SUM\langle a:: agents \rangle (C \langle a, g \rangle []) = SUM\langle a:: agents \rangle (e \langle a, g \rangle []);
                       };
};
```

The following code (assuming model file is pe_var.gcn) checks the results of three variants:

```
get_ss_values(pe2, silent = TRUE)[pnames]
get_ss_values(pe2, silent = TRUE)[cnames]
get_ss_values(pe3, silent = TRUE)[pnames]
get_ss_values(pe3, silent = TRUE)[cnames]
```

The results are as expected — consumption levels and relative prices are the same, the only difference being the index of good which price is equal 1.

```
> get_ss_values(pe1, silent = TRUE)[pnames]
p__1 p__2 p__3
1.00 0.75 0.60
> get_ss_values(pe1, silent = TRUE)[cnames]
C_A_1 C_A_2 C_A_3 C_B_1 C_B_2 C_B_3
1.233333 1.644444 2.055556 1.766667 2.355556 2.944444
> get_ss_values(pe2, silent = TRUE)[pnames]
        p__2
                p__3
p__1
1.333333 1.000000 0.800000
> get_ss_values(pe2, silent = TRUE)[cnames]
C_A_1 C_A_2 C_A_3 C_B_1 C_B_2 C_B_3
1.233333 1.644444 2.055556 1.766667 2.355556 2.944444
> get_ss_values(pe3, silent = TRUE)[pnames]
p__1
        p__2
                р__З
1.666667 1.250000 1.000000
> get_ss_values(pe3, silent = TRUE)[cnames]
C A 1 C A 2 C A 3 C B 1 C B 2 C B 3
1.233333 1.644444 2.055556 1.766667 2.355556 2.944444
```

The relevant parts of code (equilibrium identities) of the three generated variants look as follows.

```
# 1 as numeraire
p<'1'>[] = 1;
# # 2 as numeraire
# p<'2'>[] = 1;
# # 3 as numeraire
# p<'3'>[] = 1;
# goods market clearing
<g::goods\'1'> SUM<a::agents>(C<a,g>[]) = SUM<a::agents>(e<a,g>[]);
```

```
# # 1 as numeraire
# p<'1'>[] = 1;
# 2 as numeraire
p<'2'>[] = 1;
# # 3 as numeraire
# p<'3'>[] = 1;
# goods market clearing
<g::goods\'1'> SUM<a::agents>(C<a,g>[]) = SUM<a::agents>(e<a,g>[]);
```

```
# # 1 as numeraire
# p<'1'>[] = 1;
# # 2 as numeraire
```



p<'2'>[] = 1; # 3 as numeraire p<'3'>[] = 1; # goods market clearing <g::goods\'1'> SUM<a::agents>(C<a,g>[]) = SUM<a::agents>(e<a,g>[]);

6 DERIVATION OF FIRST ORDER CONDITIONS

First order conditions for optimisation problems are derived automatically in gEcon by means of an algorithm developed and implemented for this purpose. The algorithm is applicable to most common optimisation problems encountered in dynamic stochastic models. It is fairly general and can be extended to handle more complicated problems. The detailed exposition can be found in [Klima & Retkiewicz-Wijtiwiak 2014].

6.1 The canonical problem

The algorithm presented here is applicable to a general dynamic (or static) stochastic optimisation problem with objective function given by a recursive forward-looking equation. The setup presented here is standard in economic textbooks. For example a detailed exposition can be found in [Ljungqvist & Sargent 2004] or [LeRoy *et al.* 1997].

Time is discrete, infinite and begins at t = 0. In each period t = 1, 2, ... a realisation of stochastic event ξ_t is observed. A history of events up to time t is denoted by s_t . More formally, let $(\Omega, \mathcal{F}, \mathbf{P})$ be a discrete probabilistic space with filtration $\{\emptyset, \Omega\} = \mathcal{F}_0 \subset \mathcal{F}_1 \subset \cdots \mathcal{F}_t \subset \mathcal{F}_{t+1} \cdots \subset \Omega$. Each event at date t (ξ_t) and every history up to time t (s_t) is \mathcal{F}_t -measurable. Let $\pi(s_t)$ denote the probability of history s_t up to time t. The conditional probability $\pi(s_{t+1}|s_t)$ is the probability of an event ξ_{t+1} such that $s_{t+1} = s_t \cap \xi_{t+1}$.

In what follows it is assumed that variable with time index t is \mathcal{F}_t -measurable.

In t = 0 period an agent determines vectors of control variables $x(s_t) = (x^1(s_t), \ldots, x^N(s_t))$ at all possible events s_t as a solution to her optimisation problem. The objective U_0 (lifetime utility) function is recursively given by the following equation:

$$U_t(s_t) = F\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), \mathcal{E}_t H^1(x_{t-1}, x_t, U_{t+1}, z_{t-1}, z_t, z_{t+1}), \dots, \mathcal{E}_t H^J(\dots)\right),$$
(6.1)

with constraints satisfying:

$$G^{i}(x_{t-1}(s_{t-1}), x_{t}(s_{t}), z_{t-1}(s_{t-1}), z_{t}(s_{t}), \mathbb{E}_{t}H^{1}(x_{t-1}, x_{t}, U_{t+1}, z_{t-1}, z_{t}, z_{t+1}), \dots, \mathbb{E}_{t}H^{J}(\dots)) = 0,$$

$$x_{-1} \text{ given.}$$
(6.2)

where $x_t(s_t)$ are decision variables and $z_t(s_t)$ are exogenous variables and $i = 1, \ldots, I$ indexes constraints. We shall denote expression $E_t H^j(x_{t-1}, x_t, U_{t+1}, z_{t-1}, z_t, z_{t+1})$ compactly as $E_t H^j_{t+1}$ with $j = 1, \ldots, J$. We have:

$$E_t H_{t+1}^j = \sum_{s_{t+1} \subset s_t} \pi(s_{t+1}|s_t) H^j(x_{t-1}(s_{t-1}), x_t(s_t), U_{t+1}(s_{t+1}), z_{t-1}(s_{t-1}), z_t(s_t), z_{t+1}(s_{t+1})) .$$

Let us now modify the problem by substituting $q_t^j(s_t)$ for $\mathbf{E}_t H_{t+1}^j$ and adding constraints of the form $q_t^j(s_t) = \mathbf{E}_t H_{t+1}^j$. We shall also use $F_t(s_t)$ and $G_t^i(s_t)$ to denote expressions $F\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), q_t^1(s_t), \dots, q_t^j(s_t)\right)$ and $G^i\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), q_t^1(s_t), \dots, q_t^j(s_t)\right)$ respectively. Then the agent's problem may be written as:

$$\max_{\substack{(x_t)_{t=0}^{\infty}, (U_t)_{t=0}^{\infty}}} U_0$$

s.t.: (6.3)
$$U_t(s_t) = F_t(s_t),$$

$$G_t^i(s_t) = 0,$$

$$q_t^j(s_t) = E_t H_{t+1}^j,$$

$$x_{-1} \text{ given.}$$

6.2 FIRST ORDER CONDITIONS

After formulating the Lagrangian for the problem (6.3) one arrives at first order conditions for maximizing it with respect to $U_t(s_t)$, $x_t(s_t)$ and $q_t^j(s_t)$. After some transformations and setting $\lambda_t(s_t) = 1^1$, first order conditions take the following form:

$$\lambda_{t+1}(s_{t+1}) = \sum_{j=1}^{J} \left(F_{t,4+j}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,4+j}^i(s_t) \right) H_{t+1,3}^j(s_{t+1}), \tag{6.4}$$

$$0 = F_{t,2}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,2}^i(s_t)$$

$$+ \sum_{j=1}^{J} \left(F_{t,4+j}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,4+j}^i(s_t) \right) H_{t+1,2}^j(s_{t+1})$$

$$+ E_t \lambda_{t+1} \left[F_{t+1,1} + \sum_{i=1}^{I} \mu_{t+1}^i(s_{t+1}) G_{t+1,1}^i \right]$$

$$+ \sum_{j=1}^{J} \left(F_{t+1,4+j}(s_{t+1}) + \sum_{i=1}^{I} \mu_{t+1}^i(s_{t+1}) G_{t+1,4+j}^i(s_{t+1}) \right) H_{t+2,1}^j(s_{t+2}) \right],$$

$$(6.5)$$

where e.g. 3 in $H_{t+1,3}^{j}(s_{t+1})$ stands for a partial derivative of $H_{t}^{j}(s_{t})$ with respect to its third argument, i.e. $U_{t+1}(s_{t+1})$ (we shall adopt such notation throughout this chapter).

There are N + 1 first order conditions: one w.r.t. to U_t (6.4) and N w.r.t. x_t^n (6.5). There are also I conditions $G_t^i = 0$, equation $F\left(x_{t-1}, x_t, z_{t-1}, z_t, q_t^1, \ldots, q_t^j\right) = U_t$ and J equations defining q_t^j . The overall number of equations (N+I+J+2) equals the number of variables: N decision variables x_t^n , the variable U_t , J variables q_t^j , the Lagrange multiplier λ_t and I Lagrange multipliers μ_t^i (which gives N + I + J + 2 variables).

FOCs are derived similarly for models formulated in a deterministic settings — based on the appropriately modified problem.

¹This is equivalent to reinterpreting $\lambda_{t+1}(s_{t+1})$ as $\frac{\lambda_{t+1}(s_{t+1})}{\lambda_t(s_t)}$ in all equations.

6.3 HANDLING LAGS GREATER THAN ONE

When the lags greater than one appear in the model formulation, the problem is transformed into canonical form. For this purpose, for each y_{t-m} variable appearing in the *m*th lag, m-1 artificial variables $(y_t^{\text{lag}^1}, y_t^{\text{lag}^2}, \ldots, y_t^{\text{lag}^{m-1}})$ and m-1 additional equations are added:

$$y_t^{\text{lag}^1} = y_{t-1},$$

$$y_t^{\text{lag}^2} = y_{t-1}^{\text{lag}^1},$$

...

$$y_t^{\text{lag}^{m-1}} = y_{t-1}^{\text{lag}^{m-2}}.$$

If y is a control variable, these equations are added to the constraints block, each one accompanied by a Lagrange multiplier

Artificial variables are added to the list of control variables. In case of exogenous variables appearing in lags > 1, additional equations are added only to the identities block.

7 R CLASSES

The R-part of gEcon implementation is object-based. All the information characterizing a model (parameter values, steady state, solution, information about variables and stochastic structure) is stored in objects of the gecon_model class. The outputs of stochastic simulations of the model are, on the other hand, stored in a gecon_simulation class. Models are solved and analysed by invoking functions operating on the objects of gecon_model class or generic functions¹. The information retrieved about the model variables, parameters, and shocks is stored in gecon_var_info, gecon_par_info, and gecon_shock_info classes, respectively.

7.1 CREATING gecon_model OBJECT

.gcn input files containing agent problems, identities, and market clearing conditions are processed by a shared library invoked from R level. As a result, an R file is created which comprises the gecon_model class constructor with functions and data to initialize slots. The command invoking the whole process of parsing an input file and constructing the gecon_model class object is (the .gcn extension is optional):

make_model("PATH_TO_FILE/NAME_OF_FILE.gcn")

The R file created by gEcon has exactly the same name as the input file followed by an .model.R extension. It can be later on loaded without parsing the .gcn file again, using the load_model function (the .model.R extension is optional):

load_model("PATH_TO_FILE/NAME_OF_FILE.model.R")

It is worth mentioning that the dynamic linked library may create new variables or substitute some of the userdefined variables. In particular, the variables defined in the **definitions** section are substituted for and no longer used (for details see chapter 3). gEcon may also create artificial variables to handle models with lags > 1 or models with time aggregators more complicated than in the case of exponential discounting. For example, the $y_t^{\text{lag}^1}$ variable described in section 6.3 will appear as the y_lag_1.

7.2 INTERNAL REPRESENTATION

All gEcon models are represented by the objects of the gecon_model class. The name of the class has been chosen to avoid errors caused by overwriting the class definition. If the class was named model and the user called one's model model, too, the model would load once but in the process, the class constructor would be overwritten by the instance of class.² Taking this into consideration, it has been decided to use gecon_ prefixes in class definitions. Using names of objects beginning with gecon_ is discouraged for the same reason.

All the model's elements are stored in **gecon_model** class slots, each of them containing objects of a specific class/type. Although slots of a **gecon_model** class object can be accessed using **@** followed by the slot's name

¹Generic functions are functions that behave differently depending on class of arguments on which they are invoked. Usually they are used for performing standard operations on models like printing results or plotting. Generic functions make computations with gEcon intuitive for R users [Chambers 2010].

²Therefore further use of gEcon would not be possible until the workspace is cleared.

(e.g. model_name@steady), it is strongly recommended not to modify slots directly, i.e. without the use of gEcon functions.

The user may create a new model object without building it from a .gcn input file, by calling the constructor of the gecon_model class. Again, using this constructor directly is discouraged.

The so-called 'setters', i.e. the functions which allow to set the class slots to values specified by the user, use hash tables to check if the input variables' names comply with the list of model variables. Whenever a 'setter' is used, relevant slots are updated. gEcon clears the values of slots that may no longer be in compliance with the updated parameters or settings. For instance, when a covariance matrix of shocks is passed to the object of gecon_model class, the steady state values and solution matrices are preserved but the model's statistics are cleared. Any changes in free parameters' values remove all the results from the model's slots, forcing the user to recompute the model. However, steady-state values computed prior to the change which could affect them will be stored as new initial values of the variables.

During the construction of an object of gecon_model class, all the models are classified based on the information passed to the constructor. The model's shocks, lead, and lagged values are examined which allows to classify the model as dynamic or static, and stochastic or deterministic.

gEcon neither allows to compute statistics of the deterministic models, nor to solve the perturbation in case of static ones. The information concerning the type of the model can be easily printed with the **show** or **print** functions.

7.3 FUNCTIONS OF gecon_model CLASS

One of gEcon's greatest advantages is the possibility to solve models interactively, i.e. by invoking functions available for class gecon_model sequentially. This allows users to control subsequently obtained results and facilitates debugging of models. Nevertheless, all the functions used may still be invoked altogether as one R script.

The user can solve and analyse models using implemented in gEcon:

- calibration utilities (see chapters 1.4, 8),
- steady state and perturbation solvers (see chapters 1.4, 1.5, 8, 9),
- tools for IRFs and statistics computations (see chapters 1.6, 10),
- debugging utilities (see chapters 11),
- functions for retrieving computation results (see chapters 1, 11).

7.4 gecon_simulation CLASS

The compute_irf, simulate_model, and random_path functions create an object of gecon_simulation class (for details see section 10.3). This class was designed in order to store the information about the simulations' settings and results. Standard generic functions such as — show, print, and summary — may be used with it. It is worth noting that the get_simulation_results function allows to retrieve the simulated series. Additionally, the plot_simulation function allows for simulations' visualization in a convenient way.

7.5 INFORMATION ABOUT VARIABLES, PARAMETERS, AND SHOCKS

gEcon makes it easy for users to retrieve information about specified model elements by using commands ending with _info suffix: var_info, par_info, and shock_info. This option becomes very useful, when dealing with large-scale



models, for example it allows to easily identify the equations in which the variables/parameters of interest appear. The aforementioned functions return objects of classes: gecon_var_info, gecon_par_info, and gecon_shock_info respectively. These classes store the information in a structured way, and have the print, show, and summary methods defined, allowing to print information in an aesthetic manner.

8 DETERMINISTIC STEADY STATE & CALIBRATION

First order conditions, identities, and market clearing conditions determine the behaviour of agents in the model. If a long run equilibrium exists, one can find a set of variables' values that solves the system under the assumption that shocks are equal to zero and variables values do not change over time. This static equilibrium or the steady state can be a subject of separate analyses (e.g. comparative statics) but it is also a prerequisite of (log-)linearising the model and finding solution of the perturbation.

8.1 Deterministic steady state

All gEcon models can be written as a system of n equations of the form:

$$E_t F(y_{t-1}, y_t, y_{t+1}, \epsilon_t; \theta) = 0,$$
(8.1)

where y is a vector of n variables (consisting of control and exogenous variables: $y_t = (x_t, z_t)$) and θ is a vector of k parameters. In this setting a vector of deterministic steady-state values \bar{y} satisfies:

$$F(y^{\star}, y^{\star}, y^{\star}, 0; \theta) = 0.$$
(8.2)

8.2 Calibration of parameters

It is a common practice to calibrate model parameters in a way that assures consistency of chosen variables' steady-state values with the values observed empirically (e.g. the technology parameter calibrated based on capital share in GDP). Such calibration can be done by gEcon automatically — the gEcon language allows the user to specify which parameters are calibrated parameters and set relevant variables' steady-state values in accordance with the real world data (these quantities are denoted as γ). The system of equations (8.2) is modified for this purpose by adding m equations which describe the relationships between the chosen steady-state values where m parameters are treated as variables. Denote free parameters as θ_{fixed} and calibrated parameters as θ_{calibr} . The vector of variables' steady-state values y^* and the vector of calibrated parameters θ_{calibr} satisfy a system of (n + m) equations:

$$F(y^{\star}, y^{\star}, y^{\star}, 0, \theta_{calibr}; \theta_{fixed}, \gamma) = 0.$$
(8.3)

The calibration equations are specified in a .gcn file. The initial values of calibrated parameters may be set in R by means of the initval_calibr_par function. Deterministic steady state is computed using the steady_state function. A logical argument calibration of the steady_state function specifies whether calibration equations should be taken into account or not. When it is set to FALSE, calibrated parameters, as set with the initval_calibr_par function, are treated as free ones and calibration equations declared in a .gcn file are ignored. Therefore the user has to be careful when using this option and specify reasonable values using the initval_calibr_par function. Initial values of calibrated parameters which are currently used can be checked with the get_init_calibr_par function. Additionally, they are stored in objects of gecon_par_info class and can be printed using methods relevant for this class.

8.3 IMPLEMENTED SOLVERS

The steady_state function calls numerical non-linear solvers from the nleqslv package. The nleqslv package implements two solvers based on Broyden's and Newton's methods. The effectiveness of these methods can be influenced by a choice of a global search strategy: quadratic or geometric line search, the Powell single dogleg method or the double dogleg method.¹ The default solver employed by gEcon is Newton's method with quadratic line search.²

The most important solver settings can be accessed and changed using the options argument of the steady_state function. A list of options may contain one or more elements — if some options are not specified, the default values are assumed. Options that may prove especially useful to users are: global which specifies the search strategy, max_iter which determines the maximal number of iterations carried out in search of the solution and tol which specifies tolerance for the solution. gEcon checks if solution indicated by the solver satisfies the model's equations. If the 1-norm of residuals is less then the specified tolerance, the solution is saved as steady-state values of variables and calibrated parameters if calibration is used. Otherwise, it is saved with the information that it represents values from the last solver iteration. Solver status is printed on the console and stored in the object of the gecon_model class.

8.4 How to improve the chances of finding solution?

Our experience shows that using symbolic reduction algorithm implemented in gEcon significantly improves chances of finding the steady state by reducing the problem dimension. You should not explicitly name Lagrange multipliers if not necessary (internally generated Lagrange multipliers are automatically selected for reduction) and always try to reduce as many variables as possible by listing candidates for reduction in the tryreduce block of the .gcn file (see section 3.4).

Although our experience indicates that most solvers manage to find the steady state of models, at least medium-size ones, based on the default initial values only,³ good initial guesses of steady-state values always improve the chance of finding the solution. The initial values of variables and calibrated parameters are passed to the gecon_model class using the initval_var and initval_calibr_par functions and can be checked by invoking the get_init_val_var and get_init_calibr_par functions, respectively. When setting the initial values one has to remember about functions' domains — solver will not find a solution if it encounters an undefined expression in an initial iteration. E.g. the solver will not be able to compute the expression: log(1 - a - b) when a + b > 1 — setting the initial values of both variables a and b to 0.2 solves the problem. As an alternative, steady-state solver can start searching for a solution from the values of the last saved iteration of the former search process — the user has to set to TRUE the last_solver_iter option of the steady_state function (set to FALSE by default).

8.5 TROUBLESHOOTING

In situation when gEcon fails to find the solution, the following warning message will appear:

> model <- steady_state(model)
Warning message:
In steady_state(model) :
 The steady state has not been found, 1-norm of residuals is: 582.637797660505.
It is more than requested precision.
Change initial values and check if the steady state can be found.</pre>

¹For details see the package documentation [Hasselman 2013].

 $^{^{2}}$ It uses Jacobian matrix automatically derived by gEcon, if Jacobian derivation was not turned off, see section 3.3.

³These are 0.9 for variables and 0.5 for parameters.

The get_residuals function allows to check which equations had the largest initial residuals and the largest residuals after the solver has stopped. For example, the following output indicates that the solver is converging but after the default number of iterations it is still too far from solution.

> get_residual:	s(model)									
Initial residua	als:									
Eq. 1	Eq. 2	Eq. 3	Eq. 4	Eq. 5	Eq. 6					
-0.887	-999.100	0.000	14.100	26.100	0.005					
Eq. 7	Eq. 8	Eq. 9	Eq. 10	Eq. 11 Ca	libr Eq. 1					
2.748	-34.658	0.878	5.182	-900.774	899.676					
Equations with	Equations with the largest initial residuals:									
Eq. 2, Eq. 11,	Calibr Eq. 1,	Eq. 8, Eq. 5								
Final residuals	5:									
Eq. 1	Eq. 2	Eq. 3	Eq. 4	Eq. 5	Eq. 6					
-0.167	0.000	0.000	582.638	378.930	0.000					
Eq. 7	Eq. 8	Eq. 9	Eq. 10	Eq. 11 Ca	libr Eq. 1					
0.041	-219.188	0.000	2.550	-397.172	233.009					
Equations with	the largest fi	nal residual	s:							

Eq. 4, Eq. 11, Eq. 5, Calibr Eq. 1, Eq. 8

The equations 4 and 11 may be displayed through a call to the list_eq function:

> list_eq(model, eq_idx = c(4, 11))

Eq. 4: "-W[] + Z[] * (1 - alpha) * K_d[]^alpha * L_d[]^(-alpha) = 0" Eq. 11: "-C[] - I[] + Y[] + K_s[-1] * r[] - r[] * K_d[] + L_s[] * W[] - L_d[] * W[] - psi * K_s[-1] * (-delta + K_s[-1]^-1 * I[])^2 = 0"

The 1st calibrating equation can be viewed using the list_calibr_eq function:

> list_calibr_eq(model, c(1))

Calibr. Eq. 1: "-0.36 * Y[ss] + r[ss] * K_d[ss] = 0"

As the Y, K^d and r variables appear in all the 1st calibrating equation, their initial values may be suspected for causing troubles with convergence to the steady state. However, K^d appears in all equations and seems to prevent the model from converging. Indeed, in this a bit contrived example, its initial value was set deliberately to 1000, i.e. far from the true value. One can get more information about variables using the var_info function, see section 11.1.

When the norm of final residuals is greater than the norm of initial residuals, it may indicate one of several possible problems. It may suggest that variables' initial values are far from the solution. It may also hint that the model has been incorrectly formulated and does not allow for existence of equilibrium. Improper values of free parameters in an otherwise correct model can also cause this problem, e.g. discount factor greater than 1 or negative depreciation rate. Free parameters' values can be checked using the get_par_values function and set by the set_free_par function.

9 SOLVING THE MODEL IN LINEARISED FORM

gEcon solves dynamic equilibrium models using the first order perturbation method, which is most popular among researches, especially when dealing with larger scale models. The perturbation method requires linearisation of the model around its steady state. Log-linearising models instead of only linearising them is a common practice among researchers, since variables after log-linearisation can be interpreted as percent relative deviations from their steady-state values.

9.1 LOG-LINEARISATION

Currently most models have to be log-linearised manually or written down using natural logarithms of variables in order to be log-linearised (the latter is required e.g. by Dynare). The first approach is quite tedious, while the latter makes interpretation of steady-state values difficult (one have to exponentiate obtained steady-state results manually as they appear as natural logarithms of the model's variables instead of their values). gEcon log-linearises equations automatically, right before solving the perturbation.

First order conditions and identities describing a model can be written as the following system:

$$E_t F(y_{t-1}, y_t, y_{t+1}, \epsilon_t) = 0.$$
(9.1)

The steady state satisfies:

$$F(y^{\star}, y^{\star}, y^{\star}, 0) = 0. \tag{9.2}$$

Differentiating (9.1), the model can be expanded around its steady state:

$$F_1|_{(y^\star, y^\star, y^\star, 0)}(y_{t-1} - y^\star) + F_2|_{(y^\star, y^\star, y^\star, 0)}(y_t - y^\star) + F_3|_{(y^\star, y^\star, y^\star, 0)}(\mathcal{E}_t y_{t+1} - y^\star) + F_4|_{(y^\star, y^\star, y^\star, 0)}\epsilon_t = 0,$$
(9.3)

where $F_n|_{(y^*, y^*, y^*, 0)}$ denotes the derivative of F with respect to the *n*th argument at the deterministic steady state. Let us define \tilde{y}^i as the measure of the *i*th variable's deviation from its steady-state value. In case of linearisation one has:

$$y^{i}(\tilde{y}) = y^{\star i} + \tilde{y}^{i}, \tag{9.4}$$

while in case of the log-linearisation:

$$y^i(\tilde{y}) = y^{\star i} e^{\tilde{y}^i}.\tag{9.5}$$

Linearising the model around its steady state in levels (where deviations are equal to zero), one obtains:

$$\frac{\partial y}{\partial \tilde{y}}\Big|_{0} = I, \tag{9.6}$$

where I denotes identity matrix. Linearising it in logarithms, one arrives at:

$$\frac{\partial y}{\partial \tilde{y}}\Big|_{0} = \begin{pmatrix} y^{\star 1} & 0 & \dots & 0\\ 0 & y^{\star 2} & & 0\\ \vdots & & \ddots & \vdots\\ 0 & 0 & \dots & y^{\star n} \end{pmatrix}.$$
 (9.7)

Let us denote this matrix by T. Using $y^i(\tilde{y})$ enables us to rewrite (9.1) as:

$$E_t F(y(\tilde{y}_{t-1}), y(\tilde{y}_t), y(\tilde{y}_{t+1}), \epsilon_t) = 0.$$
(9.8)

Linearising (9.8) and using the chain rule we obtain:

$$F_1|_{(y^\star, y^\star, y^\star, 0)}T\tilde{y}_{t-1} + F_2|_{(y^\star, y^\star, y^\star, 0)}T\tilde{y}_t + F_3|_{(y^\star, y^\star, y^\star, 0)}TE_t\tilde{y}_{t+1} + F_4|_{(y^\star, y^\star, y^\star, 0)}\epsilon_t = 0.$$
(9.9)

The $\tilde{F}_i = F_i|_{(y^*, y^*, y^*)}T$ matrices for *i* in 1, 2, 3 are further used in solving the perturbation. In case of each variable the user can decide whether it should be linearised or log-linearised — the *T* matrix diagonal's elements will be set accordingly either to 1 or relevant steady-state values.

Variables with a zero steady-state value are not log-linearised. A logical loglin argument of the solve_pert function specifies whether variables should be log-linearised. If it is set to TRUE, one can specify — using the not_loglin_var option — which variables should be omitted in this process, i.e. which ones are to be linearised only. gEcon does not log-linearise variables having zero steady-state values.

9.2 CANONICAL FORM OF THE MODEL AND SOLUTION

gEcon canonical form of the model in linearised form is:

$$Ay_{t-1} + By_t + CE_t y_{t+1} + D\epsilon_t = 0. (9.10)$$

A, B, C, D matrices depend both on parameters and steady-state values. y_t are (percentage) deviations of variables from the steady state in case of (log-)linearisation. Let $y_t^{(s)}$ be state variables, i.e. those variables that appear in the model in lagged values (variables corresponding to non-zero rows in A matrix). Variables that are neither state variables nor exogenous shocks (ϵ_t) are called jumpers ($y_t^{(j)}$). The solution of the model in terms of state variables $y_t^{(s)}$ and exogenous shocks ϵ_t looks as follows:

$$y_t^{(s)} = P y_{t-1}^{(s)} + Q \epsilon_t,$$

$$y_t^{(j)} = R y_{t-1}^{(s)} + S \epsilon_t.$$
(9.11)

To verify whether the set of P, Q, R, S matrices solves the (9.10) problem, permute y and columns of matrices yielding $\tilde{y}_t = \begin{pmatrix} y_t^{(s)} \\ y_t^{(j)} \end{pmatrix}$ and:

$$\tilde{A}\tilde{y}_{t-1} + \tilde{B}\tilde{y}_t + \tilde{C}\mathbf{E}_t\tilde{y}_{t+1} + \tilde{D}\tilde{\epsilon}_t = 0.$$
(9.12)

Using \tilde{y}_t the (9.11) equations can be rewritten in a more compact way:

$$\tilde{y}_t = \underbrace{\begin{pmatrix} P & 0 \\ R & 0 \end{pmatrix}}_{R'} \tilde{y}_{t-1} + \underbrace{\begin{pmatrix} Q \\ S \end{pmatrix}}_{S'} \epsilon_t.$$
(9.13)

The solution should satisfy the (9.12) equation, so after using (9.13), the following condition is obtained:

$$(A + BR' + CR'R')\tilde{y}_{t-1} + (BS' + CR'S' + D)\epsilon_t + CS'E_t\epsilon_{t+1} = 0.$$
(9.14)

This condition can be satisfied for all the y and ϵ values only if:

$$A + BR' + CR'R' = 0$$
 (deterministic part condition), (9.15)

BS' + CR'S' + D = 0 (stochastic part condition).

gEcon checks these conditions and displays a warning if they are not satisfied.¹ In order to specify the accuracy of this check, the norm_tol option of the solve_pert function can be used. It contends the maximum tolerable 1-norm of the residuals of the equations (9.15).

9.3 Solution procedure

In order to obtain the solution, gEcon uses the gensys solver written by Christopher Sims [Sims 2002]. The canonical form accepted by this solver differs from the gEcon's form described above. It is as follows:

$$\Gamma_0 g_t = \Gamma_1 g_{t-1} + C + \Psi \eta_t + \Pi \epsilon_t, \tag{9.16}$$

where the vector g_t consists of the model's variables sorted so that the first k variables are variables that appear in leads in any of the equations:

$$g_{t} = \begin{pmatrix} y_{1,t} \\ y_{2,t} \\ \vdots \\ y_{n,t} \\ E_{t}y_{1,t+1} \\ E_{t}y_{2,t+1} \\ \vdots \\ E_{t}y_{k,t+1} \end{pmatrix},$$
(9.17)

the vector η_t denotes expectational errors:

$$\eta_t = \begin{pmatrix} \eta_{1,t} = y_{1,t} - \mathbf{E}_{t-1}y_{1,t} \\ \eta_{2,t} = y_{2,t} - \mathbf{E}_{t-1}y_{2,t} \\ \vdots \\ \eta_{k,t} = y_{k,t} - \mathbf{E}_{t-1}y_{k,t} \end{pmatrix}$$

 ϵ_t is a vector of stochastic shocks at time t with dimension s equal to the number of shocks, Γ_0 and Γ_1 are matrices with dimensions $(n + k) \times (n + k)$ and Ψ and Π have dimensions of $(n + k) \times (k)$ and $(n + k) \times s$, respectively. C denotes a constant term. The solver uses qz decomposition (based on Lapack implementation) with qzdiv and qzswitch routines to order decomposition results, dividing the system into stable and non-stable parts. After solving the each part, the solution is written in the following form:

 $^{^{1}}$ This may occur when the model is incorrectly specified or due to numerical roundoff errors. In the latter case, the user should consider log-linearising variables with large steady-state values or changing model's parametrisation.

$$g_t = \Theta_1 g_{t-1} + \Theta_c + \Theta_0 \epsilon_t. \tag{9.18}$$

See [Sims 2002] for the detailed description of the procedure.

The transformation of gEcon's canonical form into Sims' form requires sorting matrices' columns so that they could correspond to the order of variables in the g vector and adding equations for expectational errors. In matrix notation, using the naming convention applied in the (9.10) and (9.17) definitions, the transformation can be written as:

$$\underbrace{\left(\begin{array}{c|c} B & C \\ \hline -I & 0 \end{array}\right)}_{\Gamma_0} g_t = \underbrace{\left(\begin{array}{c|c} A & 0 \\ \hline 0 & I \end{array}\right)}_{\Gamma_1} g_{t-1} + C + \underbrace{\left(\begin{array}{c} 0 \\ \hline I \end{array}\right)}_{\Psi} \eta_t + \underbrace{\left(\begin{array}{c} D \\ \hline 0 \\ \hline \end{array}\right)}_{\Pi} \epsilon_t.$$
(9.19)

The gensys output is transformed into gEcon solution's form by picking indices of non-zero columns in Θ_1 and then adjusting it to the similar form as (9.13).

The solution can be found only if the number of the non-predetermined variables is equal to the number of eigenvalues outside the unit circle ([Blanchard O. J. 1980]). If the number of eigenvalues greater than 1 exceeds (is less than) the number of non-predetermined variables, there is no solution (an infinite number of solutions). The gEcon check_bk function allows to print the eigenvalues and compare them with the number of non-predetermined variables.

9.4 TROUBLESHOOTING

Consider a case of a simple RBC model whose steady state has been found but problems with the perturbation solution occurred. The check_bk command shows that there are more forward looking variables than eigenvalues larger than 1:

```
> check_bk(model)
```

```
Eigenvalues of system:
```

Mod	Re	Im
9.50000e-01	9.500000e-01	0.000000e+00
9.658471e-01	9.658471e-01	-1.555702e-18
1.010101e+00	1.010101e+00	0.000000e+00
1.045819e+00	1.045819e+00	-1.470342e-17
3.087408e+14	3.087408e+14	0.000000e+00
4.701462e+16	4.696996e+16	2.048699e+15
1.061755e+17	-1.061755e+17	0.000000e+00
	Mod 9.50000e-01 9.658471e-01 1.010101e+00 1.045819e+00 3.087408e+14 4.701462e+16 1.061755e+17	ModRe9.500000e-019.500000e-019.658471e-019.658471e-011.010101e+001.010101e+001.045819e+003.087408e+143.087408e+143.087408e+144.701462e+164.696996e+161.061755e+17-1.061755e+17

The model has 6 forward looking variables and 5 eigenvalues larger than 1 in modulus. BK conditions have NOT been SATISFIED.

Such an output indicates that either timing convention, parametrisation, or the model formulation is wrong. The timing of variables in all equations in which they appear can be easily checked by using the var_info function. In our example, the information generated by this function is as follows:

```
> var_info(model, all = T)
```

Incidence	info:

		С	I	K_d	K_s	L_d	L_s	PI	U	W	Y	Z	pi	r
Equation	1		•	t	t-1		•	•	•	•	•			



Equation	2					t	t							
Equation	3		•					t					t	
Equation	4		•	t		t				t		t		
Equation	5		•	t		t					t	t		
Equation	6		•								. t.	, t+1		
Equation	7		•	t		t						t		t
Equation	8	t	•				t			t				
Equation	9 t	:, t+1 t,	t+1		t-1, t		t, t+1							t+1
Equation	10		t		t-1, t									
Equation	11	t	•				t	. t	, t+1					
Equation	12			t		t				t	t		t	t
Equation	13	t	t		t-1		t			t			t	t

It can be inferred from this output that the following variables: C (consumption), r (interest rate), U (aggregate utility), Z (technology level), L_s (labour supply), and I (investments) appear in leads. While in case of variables C, r, U, L_s and I such a timing convention is accepted in RBC models, Z — technology level — should appear only in lagged and current values. After changing the timing convention, the model will be solved without trouble.

10 Model statistics & simulation

Model solution, i.e. the recursive equilibrium laws of motion (9.11) can be used to examine model implications. gEcon offers the computation of statistics most commonly found in literature, using spectral or simulation methods. In addition, gEcon allows users to easily determine the Impulse Response Functions and simulate the model.

10.1 Specification of shock distribution

Stochastic innovations in gEcon models are assumed to follow a multivariate normal distribution with zero mean. By default, the covariance matrix of shocks is assumed to be an identity matrix, i.e. shocks are assumed to be uncorrelated with one another, with variance of each equal to 1. The entire covariance matrix as well as its individual elements can be set or changed by one of two functions: set_shock_cov_mat and set_shock_distr_par.

The entire covariance matrix can be passed to a gecon_model object using the set_shock_cov_mat function. It is assumed that the order of rows and columns in the supplied matrix is consistent with the order of shocks stored in an object of the gecon_model class. The order of shocks in the supplied matrix can be altered using the shock_order argument. As an example, the following command has to be executed to set the covariance matrix for a model with three shocks: epsilon_1, epsilon_2, and epsilon_3:

The set_shock_distr_par function gives an alternative method of setting and modifying the covariance matrix modification. It accepts single entries, updating a current covariance matrix in a coherent way. Distribution parameters can be specified as standard deviations (sd), variances (var), covariances (cov) or correlations (cor). Correlations between shocks are preserved even if the user subsequently modifies variance or standard deviation of any shock.

The naming convention for parameters accepted by this function is as follows:

```
"sd( SHOCK_NAME )"
"var( SHOCK_NAME )"
"cov( SHOCK_NAME_1, SHOCK_NAME_2 )"
"cor( SHOCK_NAME_1, SHOCK_NAME_2 )"
```

General equilibrium economic modelling language and solution framework

The following command:

should assign the same parameters to the covariance matrix of model shocks as the **set_shock_cov_mat** command above.

Note: There are two issues which the user should be careful about while using the set_shock_distr_par function. First, in contrast to other parameters, shock distribution parameters require quotation marks to be assigned properly. If quotation marks are omitted, R parser treats elements of the distr_par list or vector as functions and attempts to evaluate them, producing errors. Second, parameters passed to the distr_par argument should not be specified twice. The following code snippets present commands leading to syntax errors discussed above:

If variance or standard deviation of any shock is set to zero using any of two functions discussed in this section, this shock is not taken into account when the model is simulated.

10.2 Computation of correlations

In order to compute the statistics of model variables in gEcon, such as variances, correlations, autocorrelations, and variance decomposition the compute_model_stats function should be used.

10.2.1 Spectral analysis

If the sim option is set to FALSE, then frequency-domain techniques will be applied to compute variables' statistics. As far as the computational approach is concerned, gEcon uses mainly the framework proposed by Uhlig [Uhlig 1995].

In chapter 9 state variables were defined as $y_t^{(s)}$ and jumpers, i.e. variables that are neither state variables nor exogenous shocks (ϵ) as $y_t^{(j)}$. Using this notation the solution of the model in terms of state variables $y_t^{(s)}$ and exogenous shocks ϵ was formulated as the system of P, Q, R, S matrices such that:

$$y_t^{(s)} = Py_{t-1}^{(s)} + Q\epsilon_t,$$

 $y_t^{(j)} = Ry_{t-1}^{(s)} + S\epsilon_t.$

The total number of variables y_t is assumed to be equal to n and $E(y_t) = \mu$ is the unconditional mean of the vector. Following Hamilton (see [Hamilton 1994], chapter 10), for a covariance-stationary *n*-dimensional vector process y_t the *j*th autocovariance matrix is defined to be the following $(n \times n)$ matrix:

$$\Gamma_{j} = \mathbf{E} \left[(y_{t} - \mu)(y_{t-j} - \mu)^{T} \right].$$
(10.1)

For the process y_t with an absolute summable sequence of autocovariance matrices, the matrix-valued autocovariancegenerating function $G_Y(z)$ is defined as:

$$G_Y(z) \equiv \sum_{j=-\infty}^{\infty} \Gamma_j z^j, \qquad (10.2)$$

where z is a complex scalar.

The function $G_Y(z)$ associates $(n \times n)$ matrix of complex numbers with the complex scalar z. If it is divided by 2π and evaluated at $z = e^{-i\omega}$, where ω is a real scalar and $i = \sqrt{-1}$, the result is the *population spectrum* of the vector y:

$$f_Y(\omega) = (2\pi)^{-1} G_Y(e^{-i\omega}) = (2\pi)^{-1} \sum_{j=-\infty}^{\infty} \Gamma_j e^{-i\omega j}.$$
 (10.3)

When any element of $f_Y(\omega)$ defined by (10.3) the equation is multiplied by $e^{-i\omega j}$ and the resulting function of ω is integrated from $-\pi$ to π , the result is the corresponding element of the *j*th autocovariance matrix of *y*:

$$\int_{-\infty}^{\infty} f_Y(\omega) e^{i\omega j} d\omega = \Gamma_j.$$
(10.4)

The area under the population spectrum is the unconditional covariance matrix of y. So, knowing the value of the spectral density for the vector of model's variables y for all ω in a real scalar $[0, \pi]$, the value of the *j*th autocovariance matrix for y can be calculated.

If we combine the matrices P and R into $P' = \begin{pmatrix} P \\ R \end{pmatrix}$ and Q and S into $Q' = \begin{pmatrix} Q \\ S \end{pmatrix}$, then the matrix-valued spectral density for the entire vector of variables y_t is given by:

$$f(\omega) = \frac{1}{2\pi} (I_m - P'e^{-i\omega})^{-1} Q' N Q'^T ((I_m - P'^T e^{i\omega})^{-1}),$$
(10.5)

where I_m is the identity matrix of dimension m denoting the number of state variables and N is a covariance matrix of shocks existing in the model. In order to approximate the spectrum, the grid of points is constructed (the grid's density can be controlled using **ngrid** option — the experience of the authors indicates that it should be at least 256 so that correlations do not diverge significantly from the simulation results for ordinary RBC models).¹

Most statistics in the DSGE/RBC literature are computed for HP-filtered data. gEcon offers this functionality, too.

The HP-filter removes the trend τ_t from the data given by y_t by solving:

$$\min_{\tau_t} \sum_{t=1}^T \left((y_t - \tau_t)^2 + \lambda ((\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1}))^2 \right),$$
(10.6)

¹For details of estimating the population spectrum see [Hamilton 1994], pp. 276-278.

where λ is a HP-filter parameter determining the smoothness of the trend component. The transfer function for the solution, i.e. a linear lag polynomial $r_t = y_t - \tau_t = h(L)x_t$, is:

$$\tilde{h}(\omega) = \frac{4\lambda(1 - \cos(\omega))^2}{1 + 4\lambda(1 - \cos(\lambda))^2}.$$
(10.7)

We obtain the matrix spectral density of the HP-filtered vector of the form:

$$g_{HP}(\omega) = \hat{h}(\omega)g(\omega). \tag{10.8}$$

Taking advantage of (10.3) and (10.4), we derive autocorrelations of r_t by means of an inverse Fourier transformation:

$$\int_{-\pi}^{\pi} g_{HP}(\omega) e^{i\omega k} d\omega = \mathbf{E}[r_t r_{t-k}^T].$$
(10.9)

Subsequently, this is used to derive a covariance matrix and — after relevant transformations — variances, standard deviations of the model's variables and correlations, including correlations with the reference variable (e.g. GDP) in leads and lags.

10.2.2 Simulations

As mentioned above, models may be analysed in gEcon based on the Monte Carlo simulations.

Depending on the number of simulation runs which are to be executed (with the default of 100 000), random shock vectors for multivariate normal distribution are generated. Every simulation run proceeds according to the algorithm:

- 1. First, the Cholesky decomposition (factorization) of the covariance matrix of model's shockse Σ is computed, so as to obtain a matrix A for which there is: $AA^T = \Sigma$.
- 2. Second, a vector Z consisting of n independent random variables (model's shocks) with standard normal distribution is generated.
- 3. Assuming a mean vector equal to 0, a random shock vector X is equal to: X = AZ.
- 4. Using the matrices containing the variables' equilibrium laws of motion, i.e. the impact of lagged state variables (matrices P and Q) and shocks (matrices R and S) on all the variables in the model, consecutive values of the variable series are computed based on random shock vectors.

In this way the series for all the model's variables are simulated. As mentioned above, gEcon allows to filter the series using the HP-filter.

Finally, based on the simulated and optionally HP-filtered series, the covariance matrix of the model's variables and autocorrelations are computed.

Please note that MC simulations for large-scale models may take significant amount of time.

10.2.3 Decomposition of variance

In order to obtain the decomposition of variance a three-step procedure is carried out:

- the total variance of each model variable is computed,
- the amount of variance each shock accounts for is determined,

• the share of variance caused by each shock relative to the total variance is calculated.

The amount of variance each shock accounts for is computed analogously to the total variance, i.e. using (10.5) for spectral density computation, with one exception. N_i equal to:

$$N_i = (Ae_i) \left(Ae_i\right)' \tag{10.10}$$

is used for the *i*th shock instead of N (where N = AA' and e_i is a column vector with 1 on the *i*th place and zeros elsewhere).

10.3 SIMULATING THE MODEL

gEcon allows users to perform model simulations in three different ways:

- computation of standard impulse response functions for all model shocks,
- simulation using random path of shocks drawn from distribution with a given covariance matrix,
- simulation using a user-defined path of shocks.

It should be noted that all the simulations available in gEcon are performed under the assumption that agents in the model do not know shocks' realisations in advance.

The function compute_irf computes the IRFs based on uncorrelated shocks when the option cholesky is set to FALSE. The IRFs based on correlated shocks are computed when this option is set to TRUE, i.e. when the Cholesky decomposition of a covariance matrix of the model's shocks is used.

The command random_path simulates the behaviour of the economy. It draws a path of shocks based on their covariance matrix and computes the implied dynamics of chosen variables.

The user may also specify her own path of shocks and verify its impact on the economy using the function simulate_model. E.g. the IRFs for negative shocks can be generated in this way.

The functions random_path and compute_irf create shock paths which are passed to the simulate_model function — the main simulation engine. Based on the state-space representation (the matrices P, Q, R, and S) the simulation is performed for all state variables and specified non-state variables.

Simulation results are returned in an object of class gecon_simulation. The user may see the simulation results after calling the summary method and retrieve them by using the get_simulation_results function.

In the following example the user-defined shocks have been set with the command:

In the analysed scenario two negative shocks affect productivity in the first and fourth period.

Simulation results stored in an object of gecon_simulation class can be plotted by using the plot_simulation function taking an object of this class as an argument. Sample plots for the model from chapter 1 are presented below.



Figure 10.1: Impulse response function for ϵ^{Z}



Figure 10.2: Random path for 100 periods



Figure 10.3: Simulation with the user defined shocks

11 WORKING WITH MODELS FROM R

gEcon has been designed with a goal to simplify the process of creating and solving DSGE & CGE models and to make it interactive, so that models can be analysed and debugged more easily. This is reflected both in the language and the R interface, which provides users with functions that allow to easily extract model characteristics, check solution status and help with debugging.

11.1 INFORMATION ABOUT PARAMETERS, VARIABLES & SHOCKS

Parameters, variables, and shocks in a model can be listed by the get_par_names, get_var_names, and get_shock_names functions, which return vectors of character strings. Using the get_par_names logical arguments free_par and calibr_par one can select free or calibrated parameters only. By default all parameters are returned.

In our rbc_ic example from chapter 1 these functions return:

```
> get_par_names(rbc_ic, free_par = TRUE, calibr_par = FALSE)
[1] "beta" "delta" "eta"
                           "mu"
                                    "phi"
                                            "psi"
> get_par_names(rbc_ic, free_par = FALSE, calibr_par = TRUE)
[1] "alpha"
> get_var_names(rbc_ic)
[1] "r" "C"
                      "K s" "L s" "U"
                                        "W"
              "I"
                                              "Y"
                                                     "Z"
> get_shock_names(rbc_ic)
[1] "epsilon_Z"
```

gEcon provides users with three functions: par_info, var_info and shock_info, which collect information about (selected) model parameters, variables and shocks and return objects of classes gecon_par_info, gecon_var_info, and gecon_shock_info respectively. If the return value of these functions is not assigned to a variable, it is printed on the R console. They can be used for both model analysis as well as diagnosing problems. In order to select parameters, variables and shocks of interest use these functions' arguments par_names, var_names, and shock_names respectively.

An example using these functions is presented below:

```
> par_info(rbc_ic, parameters = c("alpha", "eta", "psi"))
Incidence info:
```

	alpha	eta	psi
1	Х		
2	Х		
3	Х		
5		Х	Х
6		Х	
8		Х	
9			Х
	1 2 5 6 9	alpha 1 X 2 X 3 X 5 . 6 . 8 . 9 .	alpha eta 1 X . 2 X . 3 X . 5 . X 6 . X 8 . X 9 . .



Parameter info:

.gcn fi	ile value (Current va	alue Calibr	.initial	value	Parameter type
alpha					0.4	Calibrated
eta	2		2			Free
psi	0.8		0.8			Free
> var_info(rt	oc_ic, var:	iables = 0	c("Y", "C",	, "I"))		
Incidence inf	fo:					
	Y	С	I			
Equation	3 t					
Equation	5.	t, t+1	t, t+1			
Equation	6.	t				
Equation	7.		t			
Equation	8.	t	•			
Equation	9 t	t	t			
Calibr. Eq.	1 ss					
-						
Steady-state	values:					
U U						
Steady stat	te					
Y 1.339	93					
C 0.957	78					
I 0.381	16					
Initial value	es for stea	adv state	computatio	on:		
		j	r			
Initial val	lues					
Y	0.9					
C	0.9					
T	0.9					
-	0.0					
Variable info	. .					
Variable into						
Te a state	variable?	Is log-l	inearised?			
v rs a state	variabie.	15 10g 1. V	mear ibeu.			
C C		v				
т		I V				
T		I				
D				_		
Recursive law	ws or motio	on ior the	e variables	3		
a						
State variabl	Les impact	:				

Ζ

K_s[-1] Z[-1] Y 0.5150 0.5269 C -0.2073 3.1849 0.3092 1.2842 Ι Shocks impact: epsilon_Z 0.5547 Y С 3.3526 Ι 1.3518 _____ Basic statistics: Steady-state value Std. dev. Variance Loglin Y 1.3393 0.1762 0.031 Y С 0.9578 0.0744 0.0055 Y Ι 0.3816 0.4373 0.1913 Y Correlations: С Ι K_s L s U W Y r Y 0.9760 0.9791 0.9962 0.3119 0.9895 0.9909 0.9939 1.0000 0.9977 C 0.9113 1.0000 0.9577 0.4987 0.9393 0.9976 0.9956 0.9791 0.9629 I 0.9912 0.9577 1.0000 0.2282 0.9983 0.9755 0.9805 0.9962 0.9998 > shock_info(rbc_ic, all = TRUE) Incidence info: epsilon Z Eq. 4 Х Covariance matrix of shocks: epsilon_Z

0.01 epsilon_Z

An application of the var_info function to debugging first order perturbation is presented in section 9.4.

11.2MODELS WRITTEN USING gEcon TEMPLATE MECHANISM

The gEcon template mechanism allows to create models consisting of hundreds or even thousands of variables without much effort (see chapter 4). However, calibration and analysis of such models may get tedious. To facilitate these processes, two types of functions were added to the gEcon's R interface.

The get_index_sets function allows to retrieve a list of index sets used in a model. For instance, the call to this function for the example model presented in chapter 4 (named 'pure_exchange'), i.e.:



get_index_sets(pure_exchange)

will print the following list:

\$agents [1] "A" "B"

\$goods [1] "1" "2" "3"

Each of the list elements contains vector of the indices names in a given set.

The get_var_names_by_index, get_par_names_by_index, and get_shock_names_by_index functions allow to retrieve names of variables, parameters, and shocks with a given index. The following syntax could be used in the example model from chapter 4 in order to retrieve the names of variables related to the agent A:

```
get_var_names_by_index(pure_exchange, index_names = c('A'))
```

The output will be as follows:

[1]	"eA1"	"eA2"	"eA3"	"lambdaAGENTS_1A"	"CA1"
[6]	"CA2"	"CA3"	"UA"		

11.3 MODEL EQUATIONS

As described in section 3.3.2, the model equations derived by gEcon can be written to a logfile and IATEX documentation. For debugging purposes it can also be useful to list them from the R interface level. To this end functions list_eq and list_calibr_eq are provided. Some examples of their use can be found in section 8.5.

11.4 Accessing model results

The results of computations performed in gEcon can be further analysed and presented by using specially designed R functions. Contrary to other DSGE packages, which print the outcomes, but internally store them in complex and difficult-to-access structures, gEcon implements a set of functions allowing to retrieve the computed results in a user-friendly way.

The get_model_info returns a character vector containing the input file name, the input file path, and the date of model creation.

The get_par_values function prints and returns the vector of parameters. A call to this function for the example model presented in chapter 1 (called rbc_ic), i.e.:

get_par_values(rbc_ic)

will print the following output:

Model parameters:

Value alpha 0.400 beta 0.990 delta 0.025 eta 2.000 mu 0.300 phi 0.950 psi 0.800

It is worth mentioning that one can choose parameters (e.g. calibrated parameters only) whose values are to be returned with this function. In our example the call:

get_par_values(rbc_ic, parameters = c('alpha'))

will only print the value of the selected calibrated α parameter. Most gEcon "getters" have an option allowing to specify the set of variables (parameters) of interest.

The get_ss_values function prints and returns the vector of steady-state values. Going on with our example rbc_ic, the call:

get_ss_values(rbc_ic)

will print:

Steady-state values:

	Steady-state value
r	0.0351
С	0.9578
I	0.3816
K_s	15.2627
L_s	0.2645
U	-125.6048
W	3.0384
Y	1.3393
Z	1.0000

The presented results may be assigned to any R variable. For example, they could be later used for comparison with the results of model with different parametrisation (comparative statics). It is worth mentioning that in case the steady-state solver has been started but has not converged, the function will return a vector of variables' values from the last solver iteration.

The get_init_val_var and get_init_calibr_par functions return and print initial values of variables and calibrated parameters, respectively, used for the steady-state computation.

The get_pert_solution function prints and returns a list of four matrices containing variables' recursive laws of motion. The output for the example from chapter 1 is:

Matrix P:

K_s[-1] Z[-1] K_s[] 0.9698 0.0796 Z[] 0.0000 0.9500

Matrix Q:



	epsilon_Z
K_s	0.0838
Z	1.0000

Matrix R:

	K_s[-1]	Z[-1]
r[]	-0.0242	0.0451
C[]	0.5150	0.5269
I[]	-0.2073	3.1849
L_s[]	-0.1513	0.5570
U[]	0.0483	0.0670
W[]	0.4605	0.7272
Y[]	0.3092	1.2842

Matrix S:

epsilon_Z r 0.0474 C 0.5547 I 3.3526 L_s 0.5863 U 0.0705 W 0.7655 Y 1.3518

Again, the returned list can be assigned to a variable for future use. Both get_pert_solution and get_ss_values functions have option silent, which suppresses console output when set to TRUE.

The solution status can be verified by using the ss_solved and re_solved functions. They return TRUE if the steady state and perturbation solution, respectively, have been found and FALSE otherwise.

The get_shock_cov_mat returns the covariance matrix of model shocks.

The get_model_stats function prints and returns the statistics of the model. The user may choose statistics which to be returned. This function should be called after a call to compute_model_stats.

The following commands in our example:

rbc_ic <- compute_model_stats(model = rbc_ic, n_leadlags = 6)
get_model_stats(rbc_ic)</pre>

produce:

Basic statistics:

	Steady-state value	Std. dev.	Variance	Loglin
r	0.0351	0.0063	0.0000	Ν
С	0.9578	0.0744	0.0055	Y
I	0.3816	0.4373	0.1913	Y
K_s	15.2627	0.0390	0.0015	Y
L_s	0.2645	0.0769	0.0059	Y
U	-125.6048	0.0093	0.0001	Y

W	3.0384	0.1008	0.0102	Y
Y	1.3393	0.1762	0.0310	Y
Z	1.0000	0.1303	0.0170	Y

Correlation matrix:

r С I K_s L_s U W Y Ζ 1 0.911 0.991 0.098 0.997 0.938 0.946 0.976 0.989 r С 1 0.958 0.499 0.939 0.998 0.996 0.979 0.963 Ι 1 0.228 0.998 0.975 0.981 0.996 1.000 K_s 1 0.171 0.437 0.415 0.312 0.246 1 0.961 0.967 0.989 0.997 L_s 1 0.991 0.979 U 1 W 1 0.994 0.984 Y 1 0.998 Ζ 1.000

Autocorrelations:

Lag 1 Lag 2 Lag 3 Lag 4 Lag 5 Lag 6 0.711 0.467 0.267 0.105 0.000 -0.115 r С 0.745 0.522 0.331 0.170 0.039 -0.064 Ι 0.712 0.470 0.269 0.108 -0.018 -0.113 K_s 0.960 0.863 0.729 0.574 0.411 0.249 L_s 0.711 0.467 0.266 0.105 -0.021 -0.115 0.734 0.504 0.310 0.149 0.020 -0.081 U 0.731 0.499 0.303 0.143 0.014 -0.086 W Y 0.718 0.479 0.280 0.119 -0.008 -0.104 Ζ 0.713 0.471 0.271 0.110 -0.016 -0.111

Variance decomposition:

	epsilon_Z	
r	1	
С	1	
I	1	
K_s	1	
L_s	1	
U	1	
W	1	
Y	1	
Z	1	

It is a common practice to relate variables' standard deviations to a chosen reference variable (GDP) and to compute correlations with its leads and lags. This can be achieved by setting the **ref_var** argument of the **compute_model_stats** function to the name of the reference variable. In our example the call:

```
rbc_ic <- compute_model_stats(model = rbc_ic, ref_var = 'Y', n_leadlags = 5)
get_model_stats(rbc_ic, basic_stats = F, corr = T, autocorr = F, var_dec = F)</pre>
```

will produce:

Correlation matrix:


r	C	I	K_s	L_s	U	W	Y	Z
1	0.911	0.991	0.098	0.997	0.938	0.946	0.976	0.989
	1	0.958	0.499	0.939	0.998	0.996	0.979	0.963
		1	0.228	0.998	0.975	0.981	0.996	1.000
			1	0.171	0.437	0.415	0.312	0.246
				1	0.961	0.967	0.989	0.997
					1	1	0.991	0.979
						1	0.994	0.984
							1	0.998
								1.000
	r 1	r C 1 0.911 1	r C I 1 0.911 0.991 1 0.958 1	r C I K_s 1 0.911 0.991 0.098 1 0.958 0.499 1 0.228 1	r C I K_s L_s 1 0.911 0.991 0.098 0.997 1 0.958 0.499 0.939 1 0.228 0.998 1 0.171 1	r C I K_s L_s U 1 0.911 0.991 0.098 0.997 0.938 1 0.958 0.499 0.939 0.998 1 0.228 0.998 0.975 1 0.171 0.437 1 0.961 1	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	$ \begin{array}{cccccccccccccccccccccccccccccccccccc$

Cross correlations with the reference variable (Y):

	Std.	dev.	rel. to Y	Y[-5]	Y[-4]	Y[-3]	Y[-2]	Y[-1]	Y[]	Y[1]	Y[2]	Y[3]	Y[4]	Y[5]
r[]			0.036	0.102	0.222	0.368	0.542	0.745	0.976	0.638	0.363	0.143	-0.026	-0.151
C[]			0.422	-0.111	0.017	0.186	0.399	0.662	0.979	0.762	0.567	0.396	0.249	0.126
I[]			2.482	0.036	0.161	0.317	0.507	0.733	0.996	0.690	0.435	0.227	0.062	-0.065
K_s[]			0.221	-0.483	-0.426	-0.326	-0.176	0.034	0.312	0.498	0.607	0.656	0.657	0.622
L_s[]			0.436	0.065	0.188	0.340	0.523	0.740	0.989	0.669	0.404	0.191	0.023	-0.103
U[]			0.053	-0.076	0.052	0.219	0.428	0.685	0.991	0.750	0.539	0.358	0.206	0.081
W[]			0.572	-0.064	0.064	0.230	0.438	0.692	0.994	0.746	0.529	0.344	0.190	0.065
Y[]			1.000	-0.008	0.119	0.280	0.479	0.718	1.000	0.718	0.479	0.280	0.119	-0.008
Z[]			0.740	0.027	0.152	0.309	0.501	0.730	0.998	0.697	0.445	0.239	0.074	-0.053

11.5 Documenting results in IAT_{EX}

All functions described in the previous section (get_par_values, get_ss_values, get_pert_solution, get_model_stats) have a logical argument to_tex. If it is set to TRUE, results are written to a LATEX file model_name.results.tex. If this file does not exist (LATEX model output has not been turned on, see 3.3.2) it will be created on the first call to any of the aforementioned functions (with to_tex = TRUE argument).

Plots created by a call to plot_simulation function (see 10.3) can be stored on disk (as encapsulated Postscript files) after setting the to_eps argument to TRUE.

APPENDIX A. gEcon SOFTWARE LICENCE

Copyright (c) 2012-2015 The Chancellery of the Prime Minister of the Republic of Poland. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted free of charge provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. This software and its possible modifications may be used in the Republic of Poland and outside its borders solely for the purpose of carrying out economic, financial, demographic, sociological analyses and forecasts, and assessing impact of regulation or economic policy. The use of this software in its original or modified form for other purposes or against the law is a violation of this license.
- 4. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed at the Department for Strategic Analyses at the Chancellery of the Prime Minister of the Republic of Poland.

5. Neither the name of the Chancellery of the Prime Minister of the Republic of Poland nor the names of its employees may be used to endorse or promote products derived from this software or results of analyses conducted using this software in its original or modified form without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE CHANCELLERY OF THE PRIME MINISTER OF THE REPUBLIC OF POLAND ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CHANCELLERY OF THE PRIME MINISTER General equilibrium economic modelling language and solution framework



OF THE REPUBLIC OF POLAND BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2015-2018 Grzegorz Klima, Karol Podemski, Kaja Retkiewicz-Wijtiwiak (authors) Copyright (c) 2018-2025 Karol Podemski, Kaja Retkiewicz-Wijtiwiak (authors) All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted free of charge provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. This software and its possible modifications may be used in the Republic of Poland and outside its borders solely for the purpose of carrying out economic, financial, demographic, sociological analyses and forecasts, and assessing impact of regulation or economic policy. The use of this software in its original or modified form for other purposes or against the law is a violation of this license.
- 4. All advertising materials mentioning features or use of this software must display the following acknowledgement:

This product includes software developed by Grzegorz Klima, Karol Podemski, and Kaja Retkiewicz-Wijtiwiak.

5. The names of the authors may not be used to endorse or promote products derived from this software or results of analyses conducted using this software in its original or modified form without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Appendix B. ANTRL C++ target software license

gEcon uses ANTLR parser generator and its C++ output.

[The "BSD licence"] Copyright (c) 2005-2009 Gokulakannan Somasundaram, ElectronDB

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

BIBLIOGRAPHY

- [Adjemian et al. 2013] Adjemian, Stéphane, Bastani, Houtan, Karamé, Frédéric, Juillard, Michel, Maih, Junior, Mihoubi, Ferhat, Perendia, George, Ratto, Marco, & Villemot, Sébastien. 2013. Dynare: Reference Manual Version 4. Dynare Working Papers 1. CEPREMAP.
- [Blanchard O. J. 1980] Blanchard O. J., Kahn Ch. M. 1980. The Solution of Linear Difference Models under Rational Expectations. *Econometrica*.
- [Brooke et al. 1996] Brooke, Anthony, Kendrick, David, & Meeraus, Alexander. 1996. GAMS: A User's Guide. Tech. rept.
- [Chambers 2010] Chambers, J. M. 2010. Software for Data Analysis. Programming with R. Springer.
- [Hamilton 1994] Hamilton, James Douglas. 1994. Time series analysis. Princeton, NJ: Princeton Univ. Press.
- [Harrison *et al.* 2014] Harrison, Horridge, Jerie, & Pearson. 2014. *GEMPACK manual*. Tech. rept. GEMPACK Software.
- [Hasselman 2013] Hasselman, Berend. 2013. nleqslv: Solve systems of non linear equations. R package version 2.0.
- [Klima & Retkiewicz-Wijtiwiak 2014] Klima, Grzegorz, & Retkiewicz-Wijtiwiak, Kaja. 2014 (Apr.). On automatic derivation of first order conditions in dynamic stochastic optimisation problems. MPRA Paper 55612. University Library of Munich, Germany.
- [Klima et al. 2015] Klima, Grzegorz, Podemski, Karol, Retkiewicz-Wijtiwiak, Kaja, & Sowińska, Anna E. 2015 (Feb.). Smets-Wouters '03 model revisited - an implementation in gEcon. MPRA Paper 64440. University Library of Munich, Germany.
- [LeRoy et al. 1997] LeRoy, S.F., Werner, J., & Ross (Foreword), S.A. 1997. Principles of Financial Economics. Cambridge University Press.
- [Ljungqvist & Sargent 2004] Ljungqvist, L., & Sargent, T.J. 2004. Recursive macroeconomic theory. MIT press.
- [Mas-Colell et al. 1995] Mas-Colell, Andreu, Whinston, Michael D., & Green, Jerry R. 1995. Microeconomic Theory. Oxford University Press.
- [Sims 2002] Sims, Christopher A. 2002. Solving Linear Rational Expectations Models. Computational Economics.
- [Uhlig 1995] Uhlig, H. 1995. A toolkit for analyzing nonlinear dynamic stochastic models easily. Discussion Paper 1995-97. Tilburg University, Center for Economic Research.

INDEX

check_bk function, 13, 49 compute_irf function, 15, 41, 55 compute_model_stats function, 14, 52, 62, 63 gecon_model class, 9-11, 13, 14, 40, 41, 44, 51 gecon_par_info class, 40, 42 gecon_shock_info class, 40, 42 gecon_simulation class, 15, 40, 41, 55 gecon_var_info class, 40, 42 get index sets function, 59 get_model_info function, 60 get_model_stats function, 14, 15, 62, 64 get_par_names_by_index function, 60 get_par_names function, 57 get par values function, 11, 45, 60, 64 get_pert_solution function, 13, 61, 62, 64 get_residuals function, 44 get_shock_cov_mat function, 62 get_shock_names_by_index function, 60 get_shock_names function, 57 get simulation results function, 41, 55 get_ss_values function, 11, 61, 62, 64 get_var_names_by_index function, 60 get_var_names function, 57 initval_calibr_par function, 12, 43, 44 initval_var function, 12, 44 list_calibr_eq function, 45, 60 list_eq function, 45, 60 load model function, 40make_model function, 9, 10, 16, 22 par_info function, 15, 41, 57 plot_simulation function, 15, 41, 55, 64 print function, 13, 41, 42 random_path function, 41, 55 re_solved function, 62 set_free_par function, 11, 12, 45 set_shock_cov_mat function, 13, 51, 52 set_shock_distr_par function, 14, 51 shock_info function, 15, 41, 57 show function, 13, 41, 42 simulate_model function, 41, 55 solve_pert function, 12, 13, 48 ss_solved function, 62 steady state function, 11, 12, 22, 43, 44 summary function, 13, 41, 42var_info function, 15, 41, 45, 49, 57, 59