General equilibrium economic modelling language and solution framework

version 0.4.2

Warsaw, April 7, 2014

Lead developer:

Grzegorz Klima

Development team:

Karol Podemski

Kaja Retkiewicz-Wijtiwiak

Contents

Introduction

`gEcon` is a framework for developing and solving large scale dynamic stochastic general equilibrium models. It consists of model description language and an interface with a set of solvers in R [R Development Core Team 2012]. It was 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 models of Polish economy.

Publicly available toolboxes/toolkits used in the RBC/DSGE modelling require users to derive first order conditions and linearisation equations by pen & pencil (e.g. Uhlig's toolkit, [Uhlig *et al.* 1995]) or at least they require manual derivation of FOCs (e.g. Dynare, [Adjemian *et al.* 2013]). Owing to the development and implementation of an algorithm for automatic derivation of first order conditions, `gEcon` allows users to describe their models in terms of optimisation problems of agents. There is no other publicly available framework for writing and solving DSGE models in this natural way to authors' best knowledge.[1] Writing models in terms of optimisation problems instead of FOCs is far more natural to an economist, it 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 also possible to design large-scale (100+ variables) models. In addition `gEcon` can automatically produce a draft of LaTeX documentation for a model.

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

About current release

`gEcon` 0.4.2 was released on April 7, 2014. This is the second public release of the software.

Few minor bugs have been removed in the current release. It is no longer necessary to put semicolons after the blocks and sections. Moreover, the documentation has been corrected.

Todo's

Current work is focused on the language. In particular, the next version will allow users to include variables in leads > 1, formulate and solve OLG models and provide template mechanism making it easy to work with many sectors/consumer classes/countries within the same structure, yet possibly different parameters.

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 build atop low level linear algebra and other numerical libraries like BLAS and LAPACK. The main differences between them fall into 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

[1] Authors know that there exists software developed at private institutions with similar functionality, but it has never been publicly released.

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. A 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 comment to Grzegorz Klima at `gklima@users.sourceforge.net` putting `gEcon` in the e-mail subject.

Acknowlegdements

The authors would like to thank Dorota Poznańska, director of the Department for Strategic Analyses and the supervisors of the department at the Chancellery of the Prime Minister for making this project possible and for support.

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

Anna Sowińska also significantly helped by testing `gEcon` and suggesting improvements.

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

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 t a representative agent experiences instantaneous utility from consumption and leisure given by:

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

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 stock[1], 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:

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

which is recursively given by the following equation:

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

Optimisation is done subject to the following budget constraint:

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

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

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

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:

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

[1] At 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:

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

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]

$$L_t^{(d)} = L_t^{(s)}$$
$$K_t^{(d)} = K_{t-1}^{(s)}$$
$$C_t + I_t = Y_t.$$

1.1.1 Calibration

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

Table 1.1: Benchmark parameter values

Parameter	Value	Interpretation
α	0.36	Share of physical capital in the final good technology
β	0.99	Subjective discount factor
δ	0.025	Depreciation rate of physical capital
η	2.0	Relative risk aversion parameter
μ	0.3	Consumption weight in utility function
ϕ	0.95	Persistence of Z
ψ	0.8	Installation costs coefficient

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.

[2] For 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
{
    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]]     : lambda_U[];
    };
    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[]     : lambda_k[];
    };
    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 (`{}`). Every equation making up the Lagrangian for a problem should be followed by a colon (`:`) and a corresponding Lagrange multiplier. All variables lists and equations should be ended with a semicolon (`;`). Such an ending is optional for sections and blocks[3]. Variables 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.

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

```
block Firm
{
    controls
    {
```

[3]Semicolons after sections and blocks were mandatory in the 0.4.0 version.

```
        K_d[], L_d[], Y[], pi[];
    };
    objective
    {
        PI[] = pi[]     : lambda_PI[];
    };
    constraints
    {
        Y[] = Z[] * K_d[]^alpha * L_d[]^(1 - alpha)     : lambda_y[];
        pi[] = Y[] - L_d[] * W[] - r[] * K_d[]     : lambda_pi[];
    };
    calibration
    {
        r[ss] * K_d[ss] = 0.36 * Y[ss] -> 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 6.

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.[4] 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[];
    };
};
```

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
```

[4] See e.g. [Mas-Colell *et al.* 1995].

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

This finishes the process of writing our model in the `gEcon` language. Now just put together four blocks, save it as a `.gcn` file, say `rbc.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`.

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:

   ```
   rbc <- make_model("PATH_TO_FILE/rbc.gcn")
   ```

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

The `make_model` function first calls dynamic library implemented in C++ which parses a `.gcn` model file. Then, first order conditions are derived, on the basis of an algorithm described in chapter 5. 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 an `.R` output file containing all the derived functions and matrices constituting a model. An `.R` output file is saved in the same directory in which the `.gcn` file has been saved. As a bonus `gEcon` can automatically produce a draft of LaTeXdocumentation of the model which allows 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 4 and 5. 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 6-9) 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.

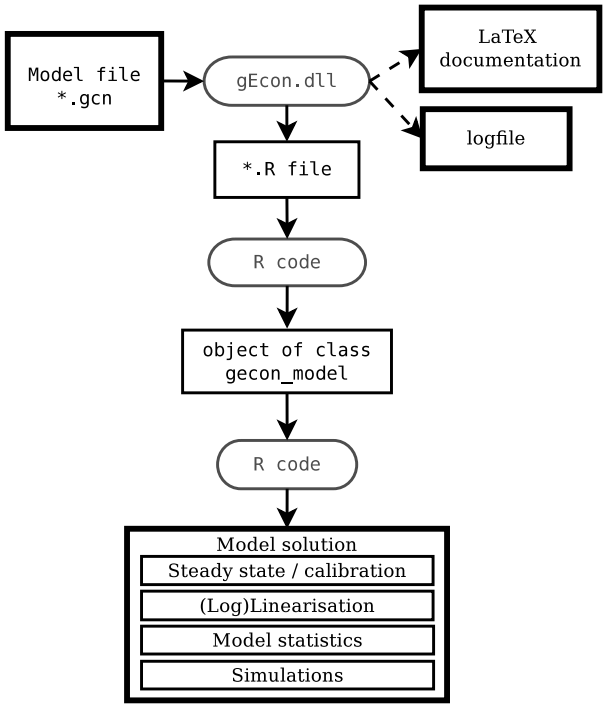

Figure 1.1: `gEcon` workflow

1.4 Finding 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.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 <- set_free_par(rbc, list(eta = 3, mu = 0.2))
```

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 <- set_free_par(model = rbc, reset = TRUE)
```

> KEEP IN MIND
>
> Most functions in the `gEcon` package in R have different options, which 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 <- steady_state(rbc)
```

After invoking this code line you should see `'Steady state 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:

```
get_ss_values(rbc)
```

and

```
get_parameter_vals(rbc)
```

and you will have them printed on the screen 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.

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. 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`.

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 two lines of code:

```
rbc <- initval_calibr_par(model = rbc,
                          calibr_par = c(alpha = 0.4))
```

```
rbc <- steady_state(model = rbc,
                    calibration = FALSE)
```

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. All of the options are described in detail in chapter 6 and documented in the `gEcon` package.

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 7 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 <- solve_pert(rbc)
```

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 SOLVED'` printed on the screen.

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 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 <- solve_pert(model = rbc,
                  loglin = FALSE)
```

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 <- solve_pert(model = rbc,
                  loglin = TRUE,
                  not_loglin_var = c("r"))
```

and that's it!

In order to see the results of the first order perturbation you should use the function:

```
get_pert_solution(rbc)
```

which prints (and returns if assigned to a variable) computed recursive laws of motion of the model's variables. 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 7 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 variance-covariance matrix to your `gecon_model` object and call a few `gEcon` functions.

In order to define the variance-covariance matrix of shocks occurring in a model you should use a `set_shocks` function. Since in our example model `rbc` there is only one shock, we will have a 1-element shock matrix containing only the shock's variance. If you wanted to set the variance to 0.01, then you would need to run the following code:

```
rbc <- set_shocks(model = rbc,
                  shock_matrix = matrix(c(0.01), 1, 1))
```

What you should keep in mind while using the `set_shocks` function in case of models with the number of shocks greater than 1 is that the order of rows and columns of shock matrix must agree with the order of shocks stored in an object of the `gecon_model` class. This order can be easily checked by using `shock_info` and `print` functions.

Having set the variance-covariance matrix of shocks, we can compute the moments of model's variables and correlation matrices using the function `compute_corr`, i.e. running the command:

```
rbc <- compute_corr(model = rbc,
                    ref_var = 'Y',
                    n_leadlags = 6)
```

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 8 and the `gEcon` package documentation.

The function `compute_corr` computes the following statistics:

- moments

  - means, standard deviations and variance of variables,
  - relative moments — means, standard deviations and variance of variables relative to a chosen reference variable,

- correlations
  - correlation matrix of all the model's variables,
  - relative correlations — correlations of variables with a 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. And if you want `gEcon` to compute variables' moments relative to corresponding values of a reference variable additionally, 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 in the table of relative correlations.

> 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.

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

```
get_moments(rbc)
```

for absolute values or:

```
get_moments(rbc, relative_to = TRUE)
```

for relative ones. 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_moments` 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 and shocks using the `var_info` and `shock_info` functions, respectively. The former 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 computation results concerning them. The latter gives you an option to choose the subset of shocks of interest and see in which equations they appear and how their variance-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` model, you should run the following:

```
rbc_irf <- compute_irf(model = rbc,
                       var_list = c('C', 'K_s', 'Z', 'Y', 'L_s', 'I'),
                       path_length = 40)
```

As `gEcon` stores information concerning IRF in another class, a newly created object — `rbc_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 8 and in the `gEcon` package documentation.

Now, if you call:

```
plot_simulation(rbc_irf)
```

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

```
plot_simulation(rbc_irf, to_eps = TRUE)
```

the IRFs will be not only plotted on the screen but also saved on your disk — in a `plots` subfolder created in the directory where the `rbc.gcn` file has been saved.

1.7 Automatic generation of model documentation in LaTeX

`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 `make_model` the function will produce a LaTeX document named just as your model file (with extension `.tex`).

Additionally, `gEcon` offers the facility of saving the results. All the so-called 'getters' in `gEcon`, i.e. `get_ss_values`, `get_parameter_vals`, `get_pert_solution`, `get_moments` functions, have a logical `to_tex` argument. If it is set to `TRUE` the output matrices or tables are written to a `.tex` file in the same directory in which the `.gcn` file has been saved. In fact, if a draft of model documentation is created while making the model, the desired information on the results is appended to it. The details of this process are described in chapter 9.

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 >=3.0.0 with the following packages: Matrix, MASS, nleqslv, Rcpp and methods. gEcon has been tested on Windows and Linux 32-bit and 64-bit platforms, but it should also run on other systems on which R works.

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 interface[1] — 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.x.zip
  ```

  or

  ```
  > R CMD INSTALL gEcon_x.x.x.tar.gz
  ```

- directly from R using installation options available in the GUI used.

Note: Windows users should use a precompiled binary package with the extension .zip by default. If you wish to build a package from source under Windows you have to install Rtools first.

Note: When installing a gEcon from a source code you might see (depending on compiler settings) some compiler warnings. If such appear, lease 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.

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++.

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

2.3.2 Kate

Go to the subdirectory `syntax` in the `gEcon` installation path. Copy the `gEcon_kate.xml` file to the directory: `~/.kde/share/apps/katepart/syntax` or `~/.kde4/share/apps/katepart/syntax`, where `~` denotes your home directory. Restart Kate.

Note: If directory `~/.kde/share/apps/` or `~/.kde4/share/apps/` exists, but it does not have subdirectory `katepart`, create it and then create `syntax` subdirectory.

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-Z0-9])*
```

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 $\delta^{K^{\text{home}}}$ 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 $\Pi_{t+1}^{\text{firm}^{\text{home}}}$ 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.

3.1.3 Reserved keywords

The following keywords are reserved in `gEcon` language:

```
E
options
```

```
block
definitions
controls
objective
constraints
identities
shocks
calibration
```

3.1.4 Comments

gEcon supports single line comments beginning with the: #, % 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.

Function names cannot be used as a variable or parameter names.

Function arguments should be enclosed in braces 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.

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

3.1.7 Conditional expectation operator

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 $\mathrm{E}_t\left[U_{t+1}\right]$.

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.

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.

Model block should begin with keyword block followed by a block name and an opening brace ({). Block names should follow the same convention as parameter/variable names. Every block should be closed by a brace (}) which can be followed by a semicolon (;).

Model file may begin with an options section. Options section should begin with the options keyword followed by an opening brace {. Options list should be closed by brace } which can be followed by a semicolon ;.

A typical gEcon model file would look as follows:

```
options {
    ...
};

block Name1 {
    ...
};

block Name2 {
    ...
};

...

block NameN {
    ...
};
```

Semicolons after the model blocks are not mandatory.

3.2.1 Options

A current set of options determines additional output files. Only an .R file is produced by the gEcon library by default.

Each output type setting should begin with the output keyword followed by the output type (currently either LaTeX/latex or logfile), operator = and a Boolean value[1]. Accepted as Boolean values are true, TRUE, false

[1] gEcon admits an option output R = but this option is silently ignored.

and `FALSE`.

The following code makes `gEcon` create both LaTeX documentation file and a text logfile:

```
options {
    output latex = TRUE;
    output logfile = TRUE;
};
```

The semicolon after the `options` block is not mandatory.

3.3 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 (`;`).

3.3.1 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]

[2] 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 {
    Y[] = K[]^alpha * EL[]^(1-alpha);
    EL[] = A[] * L[];
};
```

changing the order of the equations will cause an error.

3.3.2 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.

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:

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

Objective function

`gEcon` automatically derives first order conditions for dynamic problems with objective function given in a recursive manner and static problems (see chapter 5).

Objective function should be provided in the following way:

objective_variable `=` *time_aggregator_expression* `:` *Lagrange_multiplier*`;`

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:[4]

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

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* `:` *Lagrange_multiplier*`;`

A sample `constraints` block might look as below:[5]

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

};
```

[4] This is consumer's problem with exponential discounting and logarithmic utility from consumption.
[5] This is the budget constraint of a representative consumer/household owning and supplying capital and labour.

3.3.3 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[];
};
```

3.3.4 Shocks

Shocks are exogenous random variables. Since it is technically impossible to infer from model equations which variables are exogenous, the shocks have to be declared by the user. The `shocks` section serves this purpose. Each shock listed should have 0 time index and be followed by 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[];
};
```

3.3.5 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* -> *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 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 through invoking functions operating on objects of gecon_model class or generic functions[1].

4.1 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 called 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 .R extension. It can be later on loaded without parsing gEcon file again, using load_model("PATH_TO_FILE/NAME_OF_FILE.R") command. The user may create a new model without building it from a .gcn input file — calling the constructor of the gecon_model class. However it is cumbersome, error-prone and against gEcon spirit.

It is worth mentioning that the dynamic linked library may create new variables or substitute some of the user-defined 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 defined in chapter 5.3 will appear as the y__lag_1 in R interface.

4.2 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. The usage of gecon_model and gecon_simulation as names of class instances is not recommended 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. Although slots of a gecon_model class object can be accessed using @ followed by the slot's name (e.g. model_name@steady), it is strongly recommended not to modify slots directly, i.e. without the use of gEcon functions.

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 up-

[1] Generic 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].

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

dated parameters or settings. For instance, when a variance-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,
- stochastic or deterministic.

`gEcon` neither allows to compute statistics of the deterministic models, nor to solve the perturbation in case of the static ones. The information concerning the type of the model can be easily printed with the `show` or `print` functions. Any static model is in fact a computable general equilibrium model (CGE) — since `gEcon` accepts such models, it may be treated as a tool for formulating, calibrating, and solving CGE models.

4.3 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` gradually. This allows users to control subsequently obtained results and facilitates debugging 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, 6),
- steady state and perturbation solvers (see chapters 1.4, 1.5, 6, 7),
- tools for IRFs and statistics computations (see chapters 1.6, 8),
- debugging utilities (see chapters 9),
- functions for retrieving computation results (see chapters 1, 9).

4.4 `gecon_simulation` class

The `compute_irf`, `simulate_model` and `random_path` functions (for details see chapter 8) create an object of `gecon_simulation` class. 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 enables simulations' visualization in a convenient way.

5 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.

5.1 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 $t = 1, 2, \ldots$ 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 $(\Omega, \mathcal{F}, \mathrm{P})$ 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 $x(s_t) = \left(x^1(s_t), \ldots, x^N(s_t)\right)$ 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:

$$U_t(s_t) = F\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), \mathrm{E_t}H^1(x_{t-1}, x_t, U_{t+1}, z_{t-1}, z_t, z_{t+1}), \ldots, \mathrm{E_t}H^J(\ldots)\right), \tag{5.1}$$

with constraints satisfying:

$$\begin{aligned} &G^i\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), \mathrm{E_t}H^1(x_{t-1}, x_t, U_{t+1}, z_{t-1}, z_t, z_{t+1}), \ldots, \mathrm{E_t}H^J(\ldots)\right) = 0, \\ &x_{-1} \text{ given.} \end{aligned} \tag{5.2}$$

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 $\mathrm{E_t}H^j(x_{t-1}, x_t, U_{t+1}, z_{t-1}, z_t, z_{t+1})$ compactly as $\mathrm{E_t}H^j_{t+1}$ with $j = 1, \ldots, J$. We have:

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

Let us now modify the problem by substituting $q^j_t(s_t)$ for $\mathrm{E_t}H^j_{t+1}$ and adding constraints of the form $q^j_t(s_t) = \mathrm{E_t}H^j_{t+1}$. We shall also use $F_t(s_t)$ and $G^i_t(s_t)$ to denote expressions $F\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), q^1_t(s_t), \ldots, q^j_t(s_t)\right)$ and $G^i\left(x_{t-1}(s_{t-1}), x_t(s_t), z_{t-1}(s_{t-1}), z_t(s_t), q^1_t(s_t), \ldots, q^j_t(s_t)\right)$ respectively.

Then the agent's problem may be written as:

$$\begin{aligned} &\max_{(x_t)_{t=0}^{\infty},(U_t)_{t=0}^{\infty}} U_0 \\ &\text{s.t. :} \\ &U_t(s_t) = F_t(s_t), \\ &G_t^i(s_t) = 0, \\ &q_t^j(s_t) = \mathrm{E_t} H_{t+1}^j, \\ &x_{-1} \text{ given.} \end{aligned} \tag{5.3}$$

5.2 First Order Conditions

After formulating the Lagrangian for the problem (5.3) 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:

$$\lambda_{t+1}(s_{t+1}) = \sum_{j=1}^{J} \left(F_{t,4+j}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,4+j}^i(s_t) \right) H_{t+1,3}^j(s_{t+1}) \tag{5.4}$$

$$\begin{aligned} 0 =& F_{t,2}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,2}^i(s_t) \\ &+ \sum_{j=1}^{J} \left(F_{t,4+j}(s_t) + \sum_{i=1}^{I} \mu_t^i(s_t) G_{t,4+j}^i(s_t) \right) H_{t+1,2}^j(s_{t+1}) \\ &+ \mathrm{E_t} \lambda_{t+1} \left[F_{t+1,1} + \sum_{i=1}^{I} \mu_{t+1}^i(s_{t+1}) G_{t+1,1}^i \right. \\ &\left. + \sum_{j=1}^{J} \left(F_{t+1,4+j}(s_{t+1}) + \sum_{i=1}^{I} \mu_{t+1}^i(s_{t+1}) G_{t+1,4+j}^i(s_{t+1}) \right) H_{t+2,1}^j(s_{t+2}) \right] \end{aligned} \tag{5.5}$$

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 (5.4) and N w.r.t. x_t^n (5.5). 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.

[1] This 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.

5.3 Incorporating the 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 mth 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:

$$\begin{aligned} y_t^{\text{lag}^1} &= y_{t-1} \\ y_t^{\text{lag}^2} &= y_{t-1}^{\text{lag}^1} \\ &\ldots \\ y_t^{\text{lag}^{m-1}} &= y_{t-1}^{\text{lag}^{m-2}} \end{aligned}$$

If y is a control variable, these equations are added to the `constraints` block, each one accompanied by a Lagrange multiplier ($\lambda^{\text{NAME\_OF\_BLOCK}^{y^{\text{lag}^i}}}$). 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.

6 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.

6.1 Deterministic steady state

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

$$\mathrm{E}_t F(y_{t-1}, y_t, y_{t+1}, \epsilon_t; \theta) = 0, \tag{6.1}$$

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:

$$F(y^\star, y^\star, y^\star, 0; \theta) = 0. \tag{6.2}$$

6.2 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 the 6.2 equation 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 $y^\star$ and the vector of calibrated parameters θ_{calibr} satisfy a system of $(n+m)$ equations:

$$\bar{F}\left(y^\star, y^\star, y^\star, 0, \theta_{calibr}; \theta_{fixed}, \gamma\right) = 0. \tag{6.3}$$

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.

6.3 Implemented solvers

The `steady_state` function calls the `nleqslv` package of non-linear solvers interactively.

In the `nleqslv` package the two solvers based on Broyden and Newton methods have been implemented. 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 most important solver's settings can be accessed and changed using the `options` argument (a list) 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 taken into account. Options which may prove especially useful for 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 and stored in one of the model's slots. Solver status is printed on the screen and stored in the object of the `gecon_model` class.

6.4 How to improve the chance of finding solution?

Good initial guesses of steady-state values 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 respectively. However, our experience indicates that most solvers manage to find the steady state of models, at least medium-size ones, based only on the default initial values (0.9 for variables and 0.5 for parameters). While looking for the steady state of the model one has to remember about the function's domain — 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 these variables to 0.2 solves the problem.

Solvers based on the Newton method require the Jacobian matrix of the system. Solvers available in the `nleqslv` package are able to compute its numerical approximation. Additionally, `gEcon` symbolic library differentiates the system of steady-state equations creating its own Jacobian matrix, which can be passed to the steady-state solver instead of a numerically computed one. The use of a "true" Jacobian (by setting the `use_jac` option to `TRUE` in the function `steady_state`) may help to avoid numerical problems in case the Jacobian is ill-conditioned or nearly singular and speed up convergence.

The experience of the authors indicates that the most robust method is the Newton-based one with a quadratic or geometric line search.

[1] For details see the package's CRAN R site with the documentation [Hasselman 2009].

7 Solving 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.

7.1 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 absolute values). `gEcon` log-linearises equations automatically, right before solving perturbation.

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

$$\mathrm{E}_t F(y_{t-1}, y_t, y_{t+1}, \epsilon_t) = 0 \tag{7.1}$$

whose steady state is described as:

$$F(y^\star, y^\star, y^\star, 0) = 0. \tag{7.2}$$

Differentiating (7.1), the model can be expanded around its steady state:

$$F_1|_{(y^\star,y^\star,y^\star)}(y^\star - y_{t-1}) + F_2|_{(y^\star,y^\star,y^\star)}(y^\star - y_t) + F_3|_{(y^\star,y^\star,y^\star)}(y^\star - \mathrm{E}_t y_{t+1}) = 0, \tag{7.3}$$

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

$$y^i(\tilde{y}) = y^{\star i} + \tilde{y}^i \tag{7.4}$$

while in case of the log-linearisation:

$$y^i(\tilde{y}) = y^{\star i} e^{\tilde{y}^i}. \tag{7.5}$$

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

$$\frac{\partial y}{\partial \tilde{y}}|_0 = I, \tag{7.6}$$

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

$$\frac{\partial y}{\partial \tilde{y}}|_0 = \begin{pmatrix} y^{\star 1} & 0 & \dots & 0 \\ 0 & y^{\star 2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & y^{\star n} \end{pmatrix}. \tag{7.7}$$

This matrix is denoted as T. Using $y^i(\tilde{y})$ enables us to write (7.1) in a more general form:

$$\mathrm{E}_t F(y(\tilde{y}_{t-1}), y(\tilde{y}_t), y(\tilde{y}_{t+1}), \epsilon_t) = 0. \tag{7.8}$$

Linearising (7.8) and using a chain rule for composite functions we obtain:

$$F_1|_{(y^\star,y^\star,y^\star)} T\tilde{y}_{t-1} + F_2|_{(y^\star,y^\star,y^\star)} T\tilde{y}_t + F_3|_{(y^\star,y^\star,y^\star)} T\mathrm{E}_t\tilde{y}_{t+1} = 0 \tag{7.9}$$

The $\tilde{F}_i = F_i|_{(y^\star,y^\star,y^\star)}T$ 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-linearized. 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.

7.2 Canonical form of the model and solution

`gEcon` canonical form of the model in linearised form is:

$$Ay_{t-1} + By_t + C\mathrm{E}_t y_{t+1} + D\epsilon_t = 0. \tag{7.10}$$

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:

$$y_t^{(s)} = Py_{t-1}^{(s)} + Q\epsilon_t \tag{7.11}$$

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

To verify whether the set of P, Q, R, S matrices solves the (7.10) problem, permute y and columns of matrices yielding $\tilde{y}_t = \begin{pmatrix} y_t^{(s)} \\ y_t^{(j)} \end{pmatrix}$ and:

$$\tilde{A}\tilde{y}_{t-1} + \tilde{B}\tilde{y}_t + \tilde{C}\mathrm{E}_t\tilde{y}_{t+1} + \tilde{D}\tilde{\epsilon}_t = 0. \tag{7.12}$$

Using $\tilde{y}_t$ the (7.11) equations can be rewritten in a more compact way:

$$\tilde{y}_t = \underbrace{\begin{pmatrix} P & 0 \\ R & 0 \end{pmatrix}}_{R'} \tilde{y}_{t-1} + \underbrace{\begin{pmatrix} Q \\ S \end{pmatrix}}_{S'} \epsilon_t \tag{7.13}$$

The solution should satisfy the (7.12) equation, so after using (7.13), the following condition is obtained:

$$(A + BR' + CR'R')\tilde{y}_{t-1} + (BS' + CR'S' + D)\epsilon_t + CS'E_t\epsilon_{t+1} = 0. \tag{7.14}$$

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

$$A + BR' + CR'R' = 0 \text{ (deterministic part condition)} \tag{7.15}$$

$$BS' + CR'S' + D = 0 \text{ (stochastic part condition).}$$

`gEcon` checks these conditions and accepts the solution obtained by the solver only if they are satisfied. To specify a more or less strict accuracy of this check, the `norm_tol` option of the `solve_pert` function can be used. It specifies a maximum tolerable 1-norm of the left sides of the equations (7.15).

7.3 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:

$$\Gamma_0 g_t = \Gamma_1 g_{t-1} + C + \Psi\eta_t + \Pi\epsilon_t, \tag{7.16}$$

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:

$$g_t = \begin{pmatrix} y_{1,t} \\ y_{2,t} \\ \vdots \\ y_{n,t} \\ \mathrm{E}_t y_{1,t+1} \\ \mathrm{E}_t y_{2,t+1} \\ \vdots \\ \mathrm{E}_t y_{k,t+1} \end{pmatrix}, \tag{7.17}$$

the vector η_t denotes expectational errors:

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

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:

$$g_t = \Theta_1 g_{t-1} + \Theta_c + \Theta_0 \epsilon_t. \tag{7.18}$$

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 (7.10) and (7.17) definitions, the transformation can be written as:

$$\underbrace{\left(\begin{array}{c|c} B & C \\ \hline -I & 0 \end{array}\right)}_{\Gamma_0} g_t = \underbrace{\left(\begin{array}{c|c} A & 0 \\ \hline 0 & I \end{array}\right)}_{\Gamma_1} g_{t-1} + C + \underbrace{\left(\begin{array}{c} 0 \\ \hline I \end{array}\right)}_{\Psi} \eta_t + \underbrace{\left(\begin{array}{c} D \\ \hline 0 \end{array}\right)}_{\Pi} \epsilon_t. \tag{7.19}$$

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 (7.13).

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 (infinite number of solutions). The `gEcon` `check_bk` function allows to print the eigenvalues and compare them with the number of non-predetermined variables.

8 Model analysis

The results obtained, i.e. the recursive equilibrium laws of motion in the form of (7.11) equations can be used to examine model implications. `gEcon` offers the computation of statistics most commonly used in literature, using spectral or simulation methods.

8.1 Computation of correlations

In order to compute the second moment properties of variables in `gEcon`, such as variances, autocorrelations, or correlation matrices, the `compute_corr` function should be used.

8.1.1 Spectral analysis

If the `sim` option is set to `FALSE`, then frequency-domain techniques will be applied to compute variables' moments. As far as the methodology is concerned, `gEcon` uses mainly the framework proposed by Uhlig [Uhlig *et al.* 1995].

In chapter 7 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:

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

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

The total number of variables y_t is assumed to be equal to n and $\mathrm{E}(y_t) = \mu$ 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 jth autocovariance matrix is defined to be the following $(n \times n)$ matrix:

$$\Gamma_j = \mathrm{E}\left[(y_t - \mu)(y_{t-j} - \mu)^T\right] \tag{8.1}$$

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

$$G_Y(z) \equiv \sum_{j=-\infty}^{\infty} \Gamma_j z^j, \tag{8.2}$$

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:

$$f_Y(\omega) = (2\pi)^{-1} G_Y(e^{-i\omega}) = (2\pi)^{-1} \sum_{j=-\infty}^{\infty} \Gamma_j e^{-i\omega j} \tag{8.3}$$

When any element of $f_Y(\omega)$ defined by (8.3) 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 jth autocovariance matrix of y:

$$\int_{-\infty}^{\infty} f_Y(\omega)e^{i\omega j}d\omega = \Gamma_j. \tag{8.4}$$

The area under the population spectrum is the unconditional variance-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 jth 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:

$$f(\omega) = \frac{1}{2\pi}(I_m - P'e^{-i\omega})^{-1}Q'NQ'^T((I_m - P'^T e^{i\omega})^{-1}), \tag{8.5}$$

where I_m is the identity matrix of dimension m denoting the number of state variables and N is a variance-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 variables in the literature on RBC modelling are detrended with the Hodrick-Prescott filter (HP-filter). `gEcon` offers the possibility to remove a trend form the series with the HP-filter, irrespective of the moments' computation method chosen, so that the series could be analysed in this way.

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

$$\min_{\tau_t} \sum_{t=1}^{T} \left((y_t - \tau_t)^2 + \lambda((\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1}))^2 \right), \tag{8.6}$$

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:

$$\tilde{h}(\omega) = \frac{4\lambda(1 - cos(\omega))^2}{1 + 4\lambda(1 - cos(\lambda))^2}. \tag{8.7}$$

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

$$g_{HP}(\omega) = \tilde{h}(\omega)g(\omega). \tag{8.8}$$

Taking advantage of (8.3) and (8.4), we derive autocorrelations of r_t by means of an inverse Fourier transformation:

$$\int_{-\pi}^{\pi} g_{HP}(\omega)e^{i\omega k}d\omega = \mathrm{E}[r_t r_{t-k}^T]. \tag{8.9}$$

Subsequently, this is used to derive a variance-covariance matrix and — after relevant transformations — variances, standard deviations of the model's variables and their correlation matrix as well as variables' moments relative to a chosen reference variable (e.g. GDP).

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

8.1.2 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 variance-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 remove a trend from the series, irrespective of the moments' computation method chosen. Using simulation methods to analyse the model simulated paths can be filtered too, upon the choice of a relevant option in the `compute_corr` function, and it is done by means of the HP-filter, like in case of the spectral analysis (a sparse HP-filter is used so as to allow for computations based on a greater number of simulated observations).

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

However, choosing simulation methods one has to remember that MC simulations used with large-scale models may significantly extend the computation time.

8.1.3 Decomposition of variance

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

- the total variance of the model 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 (8.5) for spectral density computation, with one exception. N_i equal to:

$$N_i = (Ae_i)(Ae_i)' \tag{8.10}$$

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

8.2 Model simulation

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

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

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

The command random_path simulates the behaviour of the economy, drawing a path of shocks based on their variance-covariance matrix and computing their impact on specified set of variables based on the solution of the model.

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

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

As far as the implementation issues are concerned, the functions random_path and compute_irf create shock paths which are passed to simulate_model function — the main simulation engine. This function is implemented in a standard way, basing on the perturbation results, i.e. recursive equilibrium laws of motion of variables represented in gEcon by the matrices P, Q, R, S. The simulation is computed for all the state variables, shock values and specified non-state variables.

All the simulation results are passed to and stored in an object of class gecon_simulation. The results can be plotted with the function plot_simulation invoked on an object of this class. The user may also see the simulation results printed after calling the method summary and retrieve them invoking the function get_simulation_results. The visualisation of different types of simulation for chosen variables of the example model from the chapter 1 is depicted below.

The user defined shocks have been simulated with the command:

```
irf_rbc_ic <- simulate_model(rbc_ic, shock_m=matrix(c(-0.05, -0.05), nrow=1, ncol=2),
                             periods=c(1, 4),  var_list=c('K_s', 'C', 'Z', 'I', 'Y'))
```

The analysed scenario consisted of two negative shocks influencing productivity in the first and fourth period.

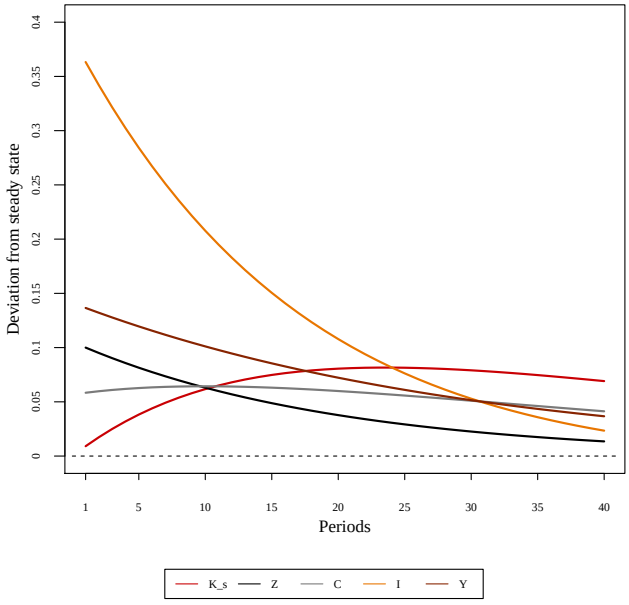

Figure 8.1: Impulse response function for ϵ^Z

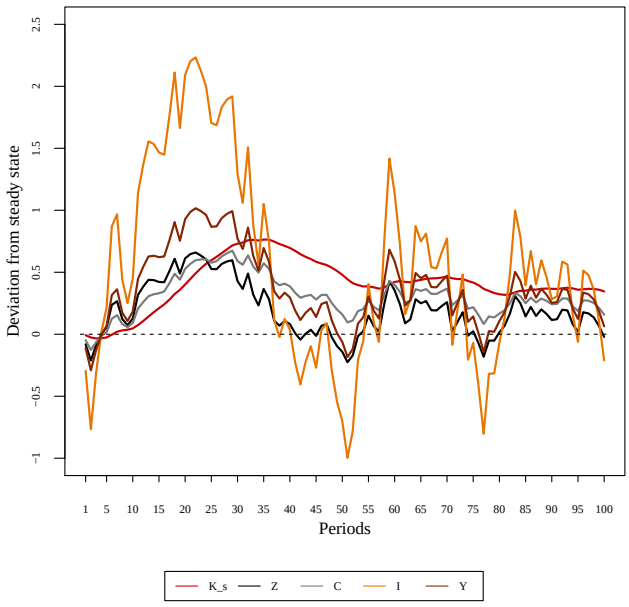

Figure 8.2: Random path for 100 periods

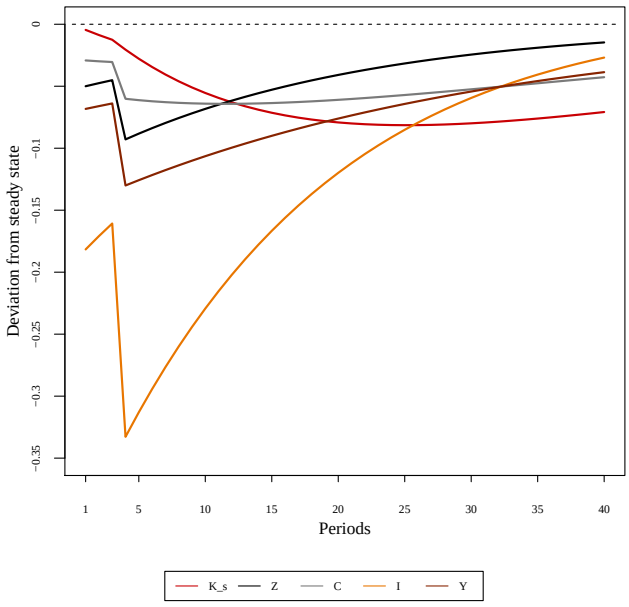

Figure 8.3: Simulation with the user defined shocks

9 Retrieving information about the model

gEcon has been created in order to simplify the process of creating and solving DSGE models. Apart from tools useful in debugging models and checking solution status, it offers co-called 'getters' that allow users to easily extract model characteristics and use them for further analysis in the R environment.

9.1 Debugging facilities

The var_info and shock_info functions are particularly useful in verifying the correctness of the model's specification.

For instance, in case of a simple RBC model which steady state has been found but for which problems with the perturbation solution have been encountered, the check_bk command will show that there are more forward looking variables than eigenvalues larger than 1:

```
Eigenvalues of system:
              Mod            Re            Im
[1,] 9.500000e-01  9.500000e-01  0.000000e+00
[2,] 9.658471e-01  9.658471e-01 -1.555702e-18
[3,] 1.010101e+00  1.010101e+00  0.000000e+00
[4,] 1.045819e+00  1.045819e+00 -1.470342e-17
[5,] 3.087408e+14  3.087408e+14  0.000000e+00
[6,] 4.701462e+16  4.696996e+16  2.048699e+15
[7,] 1.061755e+17 -1.061755e+17  0.000000e+00

There are: 6 forward looking variables. There are: 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. As far as the incidence of the model's variables is concerned, it can be easily checked by using the var_info function. For instance, the information generated by this function for a slightly modified RBC model described in chapter 1 is depicted below:

```
Incidence info:
            C       I    K_d     K_s    L_d     L_s     PI      U     W     Y       Z     pi     r
Eq. 1       0       0      t     t-1      0       0      0      0     0     0       0      0     0
Eq. 2       0       0      0       0      t       t      0      0     0     0       0      0     0
Eq. 3       0       0      0       0      0       0      t      0     0     0       0      t     0
Eq. 4       0       0      t       0      t       0      0      0     t     0       t      0     0
Eq. 5       0       0      t       0      t       0      0      0     0     t       t      0     0
Eq. 6       0       0      0       0      0       0      0      0     0     0  t, t+1      0     0
Eq. 7       0       0      t       0      t       0      0      0     0     0       t      0     t
Eq. 8       t       0      0       0      0       t      0      0     t     0       0      0     0
Eq. 9  t, t+1  t, t+1      0  t-1, t      0  t, t+1      0      0     0     0       0      0   t+1
Eq. 10      0       t      0  t-1, t      0       0      0      0     0     0       0      0     0
```

```
Eq. 11      t       0      0       0      0       t      0  t, t+1     0      0       0      0     0
Eq. 12      0       0      t       0      t       0      0       0      t      t       0      t     t
Eq. 13      t       t      0     t-1      0       t      0       0      t      0       0      t     t
```

It can be inferred from this output that 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 its lagged and current values. After changing the timing convention, the model will be solved without troubles. A similar output referring to the model's shocks can be produced using the `shock_info` function.

Furthermore, `gEcon` supports the user in the process of finding the steady state of the model. The `get_residuals` function allows to check which equations have the highest residuals after the initial evaluation of `steady_state` function (based on starting values) and after its final evaluation (when the solver stalls). For example, the following output indicates that the process is converging towards the solution but after the default number of iterations it is still too far from satisfying the convergence criterion.

```
Initial residuals:
       1        2        3        4        5        6        7        8        9       10
   0.000    0.000    0.000   -0.184 -957.711    0.000    0.006   -0.001    0.069    0.000
      11       12       13 1 calibr
   0.029  957.812   -0.060 -344.884

Equations with highest errors in initial residuals:
12, 5, 1 calibr, 4, 9

Final residuals:
       1        2        3        4        5        6        7        8        9       10
   0.000    0.000    0.000    2.666   27.292    0.000    0.007    0.000   -0.434    0.000
      11       12       13 1 calibr
   0.071  -22.889   22.889   20.740

Equations with highest errors in final residuals:
5, 13, 12, 1 calibr, 4
```

The equations 5 and 12 may be displayed by means of the `find_eq` function:

```
Eq. 5:   "-Y[] + Z[] * K_d[]^alpha * L_d[]^(1 - alpha) = 0"
Eq. 12:  "Y[] - pi[] - K_d[] * r[] - L_d[] * W[] = 0"
```

and a calibrating equation can be retrieved using the `find_calibr_eq` function:

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

As the Y, K_d and L_d variables appear in all equations above, their initial values may be suspected for causing troubles with convergence to the steady state. However, since K_d and L_d also appear in other equations, it is Y which seems to prevent the model from converging. Indeed, while solving the example model, its initial value was set deliberately to 1000, i.e. far from a true value.

When the norm of final residuals is greater then the norm of initial residuals, it may indicate one of several possible problems. It can suggest that variables' initial values are far from the solution. Moreover, it may hint that the model has been incorrectly formulated and does not allow for the existence of the deterministic steady state — an input `.gcn` file should be reformulated then. However, it could also be caused by improper values of free parameters — and those can be changed without resorting to the `.gcn` file, by means of the `set_free_par` function.

9.2 Logfile

gEcon can automatically generate a logfile containing all the information characterising the model (equations declared in a .gcn file, FOCs, derived equilibrium and steady-state equations, information on variables, and parameters). To use this feature the following lines should be included *at the beginning* of a .gcn file:

```
options
{
    output logfile = TRUE;
};
```

On successful make_model call the function will produce a logfile named just as the model file (with .model.log extension).

In case of errors encountered during derivation and collection of model equations, a logfile is written irrespective of the options chosen in a .gcn file, which may prove useful for debugging the model.

9.3 Functions get_*

The results of computations made in gEcon can be further analysed and presented by using specially designed R functions. The retrieval of the desired information is much more intuitive then in other DSGE packages, which print the outcomes, however store them in complex objects which make the access by the users difficult. gEcon implements a set of functions (so-called 'getters') allowing to retrieve the computed results in a user-friendly way.

The get_parameter_vals function prints and returns the vector of parameters. For instance, the call of this function for the example model presented in chapter 1 (named 'rbc'), i.e.:

```
get_parameter_vals(rbc)
```

will print the following output:

```
Parameters of the model:

      Parameters
alpha      0.360
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) which values are to be returned with this function. Running:

```
get_parameter_vals(rbc, var_names = c('alpha'))
```

for the example model, it 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 the example model 'rbc', the call of the command:

```
get_ss_values(rbc)
```

will print:

```
Steady state values:

    Steady state
C         0.7422
I         0.2559
K_d      10.2368
K_s      10.2368
L_d       0.2695
L_s       0.2695
U      -136.2372
W         2.3706
Y         0.9981
Z         1.0000
pi        0.0000
r         0.0351
```

One should note that presented results may be assigned to any R variable and compared with the results of models with different parametrisation or distinct agent policies later(comparative statics analysis).

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

```
Matrix P:

    K_s[-1]  Z[-1]
K_s  0.9658 0.0863
Z    0.0000 0.9500


Matrix Q:

    epsilon_Z
K_s    0.0908
Z      1.0000


Matrix R:

    K_s[-1]   Z[-1]
C    0.4748  0.5545
I   -0.3661  3.4511
K_d  1.0000  0.0000
L_d -0.1575  0.5426
L_s -0.1575  0.5426
U   -0.0418 -0.0644
W    0.4167  0.7547
Y    0.2592  1.2972
pi   0.0000  0.0000
```

```
r   -0.7408  1.2972


Matrix S:

    epsilon_Z
C      0.5837
I      3.6328
K_d    0.0000
L_d    0.5711
L_s    0.5711
U     -0.0678
W      0.7944
Y      1.3655
pi     0.0000
r      1.3655
```

After assigning such a list to any variable in R, the user may define initial deviations of the system from the steady state and analyse the impact of specific sets of shocks (or a specific path of shocks) on it.

The `get_moments` function prints and returns the statistics of the model. The user may choose statistics which should be returned.

After invoking the following command for our example model:

```
get_moments(model = rbc, relative_to = FALSE, moments = TRUE, correlations = TRUE,
            autocorrelations = TRUE, var_dec = TRUE)
```

absolute values of statistics are returned, as the `relative_to` option is set to `FALSE`. The output is presented below:

```
Moments of variables:

    Steady state value Std. dev.  Variance Loglinear
C               0.7422    0.0783    0.0061     Y
I               0.2559    0.4741    0.2248     Y
K_d            10.2368    0.0422    0.0018     Y
K_s            10.2368    0.0422    0.0018     Y
L_d             0.2695    0.0749    0.0056     Y
L_s             0.2695    0.0749    0.0056     Y
PI                   0         0         0     Y
U            -136.2372     0.009     1e-04     Y
W               2.3706    0.1047     0.011     Y
Y               0.9981    0.1781    0.0317     Y
Z                    1    0.1303     0.017     Y
pi                   0         0         0     Y
r               0.0351    0.1814    0.0329     Y

Correlations of variables:

          C       I     K_d     K_s     L_d     L_s PI       U       W       Y       Z pi
C    1.0000  0.9579  0.2350  0.4983  0.9402  0.9402  0 -0.9981  0.9960  0.9806  0.9667  0
I    0.9579  1.0000 -0.0540  0.2284  0.9984  0.9984  0 -0.9736  0.9798  0.9956  0.9995  0
K_d  0.2350 -0.0540  1.0000  0.9598 -0.1101 -0.1101  0 -0.1753  0.1467  0.0399 -0.0215  0
```

```
K_s  0.4983  0.2284  0.9598  1.0000  0.1733  0.1733  0 -0.4445  0.4184  0.3187  0.2599  0
L_d  0.9402  0.9984 -0.1101  0.1733  1.0000  1.0000  0 -0.9592  0.9670  0.9887  0.9961  0
L_s  0.9402  0.9984 -0.1101  0.1733  1.0000  1.0000  0 -0.9592  0.9670  0.9887  0.9961  0
PI   0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0  0.0000  0.0000  0.0000  0.0000  0
U   -0.9981 -0.9736 -0.1753 -0.4445 -0.9592 -0.9592  0  1.0000 -0.9996 -0.9907 -0.9805  0
W    0.9960  0.9798  0.1467  0.4184  0.9670  0.9670  0 -0.9996  1.0000  0.9942  0.9858  0
Y    0.9806  0.9956  0.0399  0.3187  0.9887  0.9887  0 -0.9907  0.9942  1.0000  0.9981  0
Z    0.9667  0.9995 -0.0215  0.2599  0.9961  0.9961  0 -0.9805  0.9858  0.9981  1.0000  0
pi   0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0  0.0000  0.0000  0.0000  0.0000  0
r    0.9082  0.9901 -0.1933  0.0897  0.9965  0.9965  0 -0.9321  0.9422  0.9726  0.9851  0
          r
C    0.9082
I    0.9901
K_d -0.1933
K_s  0.0897
L_d  0.9965
L_s  0.9965
PI   0.0000
U   -0.9321
W    0.9422
Y    0.9726
Z    0.9851
pi   0.0000
r    1.0000

Autocorrelations of variables:

       t-1    t-2    t-3    t-4     t-5
C   0.7446 0.5209 0.3292 0.1686  0.0376
I   0.7115 0.4684 0.2679 0.1066 -0.0193
K_d 0.9598 0.8626 0.7281 0.5723  0.4082
K_s 0.9598 0.8626 0.7281 0.5723  0.4082
L_d 0.7098 0.4657 0.2647 0.1034 -0.0223
L_s 0.7098 0.4657 0.2647 0.1034 -0.0223
PI     NaN    NaN    NaN    NaN     NaN
U   0.7346 0.5050 0.3106 0.1498  0.0204
W   0.7304 0.4983 0.3028 0.1419  0.0131
Y   0.7179 0.4786 0.2798 0.1186 -0.0083
Z   0.7133 0.4711 0.2711 0.1098 -0.0163
pi     NaN    NaN    NaN    NaN     NaN
r   0.7103 0.4664 0.2655 0.1042 -0.0215

Decomposition of variance:

     epsilon_Z
C            1
I            1
K_d          1
K_s          1
L_d          1
L_s          1
PI         NaN
```

```
U            1
W            1
Y            1
Z            1
pi         NaN
r            1
```

After setting the `relative_to` argument to `TRUE`, relative statistics of variables will be returned. Relative moments available in `gEcon` include variables' means, variances and standard deviations relative to the relevant values of a chosen reference variable as well as the correlations of variables with lead and lagged values of the reference variable. Most papers on RBC modelling examine correlations with lead and lagged values of GDP. In order to obtain such a table in `gEcon` for the example model, it is sufficient to invoke the following command:

```
get_moments(model = rbc, relative_to = TRUE, moments = TRUE, correlations = TRUE)
```

and the following output will be printed on the screen:

```
Moments of variables relative to Y :

    Steady state value relative to Y Std. dev. relative to Y Variance relative to Y Loglinear
C                             0.7436                 0.4395                 0.1931     Y
I                             0.2564                 2.6621                 7.0869     Y
K_d                          10.2561                 0.2368                 0.0561     Y
K_s                          10.2561                 0.2368                 0.0561     Y
L_d                             0.27                 0.4205                 0.1768     Y
L_s                             0.27                 0.4205                 0.1768     Y
PI                                 0                      0                      0     Y
U                           -136.4937                 0.0504                 0.0025     Y
W                              2.3751                 0.5877                 0.3453     Y
Y                                   1                      1                      1     Y
Z                              1.0019                 0.7319                 0.5357     Y
pi                                 0                      0                      0     Y
r                             0.0352                 1.0184                 1.0372     Y

Correlations of variables with lead and lagged Y :

     Y_[-5]  Y_[-4]  Y_[-3]  Y_[-2]  Y_[-1]   Y_[0]   Y_[1]   Y_[2]   Y_[3]   Y_[4]   Y_[5]
C   -0.1067  0.0213  0.1894  0.4025  0.6650  0.9806  0.7609  0.5644  0.3923  0.2448  0.1212
I    0.0390  0.1636  0.3192  0.5084  0.7335  0.9956  0.6875  0.4309  0.2220  0.0566 -0.0702
K_d -0.5030 -0.4795 -0.4216 -0.3213 -0.1704  0.0399  0.3187  0.5039  0.6124  0.6595  0.6589
K_s -0.4795 -0.4216 -0.3213 -0.1704  0.0399  0.3187  0.5039  0.6124  0.6595  0.6589  0.6227
L_d  0.0671  0.1898  0.3414  0.5242  0.7397  0.9887  0.6664  0.4006  0.1865  0.0192 -0.1069
L_s  0.0671  0.1898  0.3414  0.5242  0.7397  0.9887  0.6664  0.4006  0.1865  0.0192 -0.1069
PI      NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
U    0.0765 -0.0517 -0.2183 -0.4279 -0.6842 -0.9907 -0.7507 -0.5400 -0.3589 -0.2065 -0.0814
W   -0.0621  0.0660  0.2318  0.4393  0.6925  0.9942  0.7449  0.5278  0.3426  0.1881  0.0624
Y   -0.0083  0.1186  0.2798  0.4786  0.7179  1.0000  0.7179  0.4786  0.2798  0.1186 -0.0083
Z    0.0226  0.1481  0.3058  0.4986  0.7288  0.9981  0.6988  0.4479  0.2423  0.0782 -0.0488
pi      NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
r    0.1089  0.2280  0.3727  0.5446  0.7446  0.9726  0.6308  0.3527  0.1323 -0.0369 -0.1614
```

9.4 Documenting results in LaTeX

All of the functions described in this section (`get_parameter_vals`, `get_ss_values`, `get_pert_solution`, `get_moments`) have a logical argument `to_tex`. It it is set to `TRUE`, output matrices or tables are written to a LaTeX document. In fact, if an option of LaTeX documentation is set to `TRUE` in a `.gcn` file, then on a call of the `make_model` function three `.tex` files are created in the same directory in which the `.gcn` file has been saved, each of them storing different parts of the model's documentation:

- `model_name.tex` — contains the commands specifying the layout of the document and takes two following files as inputs,
- `model_name.model.tex` — an input file containing a draft of model documentation, i.e. elements specified in or derived based on a `.gcn` file (optimisation problems, contraints, identities, FOCs, stead-state and equilibrium relationships, parameter settings),
- `model_name.results.tex` — an input file containing chosen output matrices or tables, i.e. those returned by the functions with an option `to_tex` set to `TRUE`; chosen results are appended to the file.

If the generation of LaTeX documentation was not enabled (a relevaant option has not been set to `TRUE` in a `.gcn` file) but an option `to_tex` is set to `TRUE` while invoking any of the 'getters', then only a `.results.tex` file is created.

Appendix A. gEcon software licence

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*.

[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.

[Hasselman 2009] Hasselman, B. 2009. `nleqslv`: Solve systems of non linear equations.

[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.

[R Development Core Team 2012] R Development Core Team. 2012. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

[Sims 2002] Sims, Christopher A. 2002. Solving Linear Rational Expectations Models. *Computational Economics*.

[Uhlig *et al.* 1995] Uhlig, H., *et al.* 1995. *A toolkit for analyzing nonlinear dynamic stochastic models easily*. Institute for Empirical Macroeconomics, Federal Reserve Bank of Minneapolis.

Index