CoRRAM-M: User Guide

Prof. Dr. Alfred Maußner

January 10, 2020

Contents

1 Introduction

1.1 Purpose

This document explains the use of my Matlab toolbox CoRRAM. The acronym stands for Computation of Recursive Representative Agent Models. The additional M indicates the Matlab version of this toolbox. Other versions are available in Gauss and Fortran.

The toolbox is designed to allow less experienced users of Matlab to quickly set-up, solve, and simulate dynamic, stochastic general equilibrium (DSGE) models. It is closely related to Chapter 3 of the forthcoming third edition of Heer and Maußner (2009). As a user, you must have sufficient knowledge of the Matlab command syntax to code the model's equations. The simulation tool frees you from writing program code that provides information about the mechanics of the model and about its time series implications. It provides you with plots of impulse responses and tables with second moments of simulated time series. Experienced Matlab programmers and developers of complex models, whose analysis requires programming beyond the standard simulation methods, can use the toolbox just to compute the model's solution.

CoRRAM provides perturbation solutions of DSGE models. First- and second-order accurate solutions require no additional Matlab toolboxes. Third-order accurate solutions require third-order partial derivatives for which numerical differentiation is too inaccurate. Therefore, CoRRAM resorts to the computer algebra system from the Matlab *symbolic toolbox*. The repeated use of the Jacobian function from this toolbox provides analytic formulas for the matrices of first-, second-, and third-order partial derivatives of the model's equations. As an alternative to the symbolic toolbox, CoRRAM integrates CasADi, an implementation of automatic differentiation developed by Andersson, Gillis, and Diehl (2018). The user must have installed CasADi, which is freely available from `http://casadi.sourceforge.net`, to use this functionality.

1.2 Licence

The source code of CoRRAM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but *without any warranty*; without even the implied warranty of *merchantability* or *fitness for a particular purpose*. See the GNU General Public License under `<https://www.gnu.org/licenses/>` for more details.

1.3 Notation

In this document I typeset CoRRAM commands and variables in a colored typewriter font. For instance

```
Mod=DSGE(nx,ny,nz,nu);
```

is the command which creates a new instance of the `DSGE` class in the object `Mod`. File names are set in the black typewriter font. The symbol (!) in the right margin indicates important information to which the user should pay special attention.

Keep in mind a few features of the Matlab programming language:

- Matlab passes the arguments of functions per copy and not by reference. As a consequence, a variable passed to the function and changed within this function does not change the variable with the same name in the calling programm.
- The arguments of Matlab functions can be either required, optional, or consist of name-value pairs. For instance, the function `DSGE` has four required arguments (shown above), two optional arguments, and the named argument `Names`, a cell array of character vectors (see Section 9).
- Required arguments must be passed in the order expected by the function's definition and must precede optional arguments as well as name-value pairs. For instance, if you would call the function `DSGE` with the optional argument `Symbol` in front of the required arguments

  ```
  Mod=DSGE(Symbols,nx,ny,nz,nu);
  ```

  Matlab would terminate with an error message.
- Matlab is case sensitive. Thus, `A` and `a` are different variables.

The next section explains the installation of the toolbox. Section 3 presents the canonical DSGE model. Section 4 gives an overview of the DSGE class that provides the interface between the toolbox and the user. Section 5 explains how to set-up and solve a model. Section 6 introduces the simulation tool, Section 7 considers the presentation of the simulation results, Section 8 deals with error messages, and the final Section 9 presents the function reference.

2 Installation

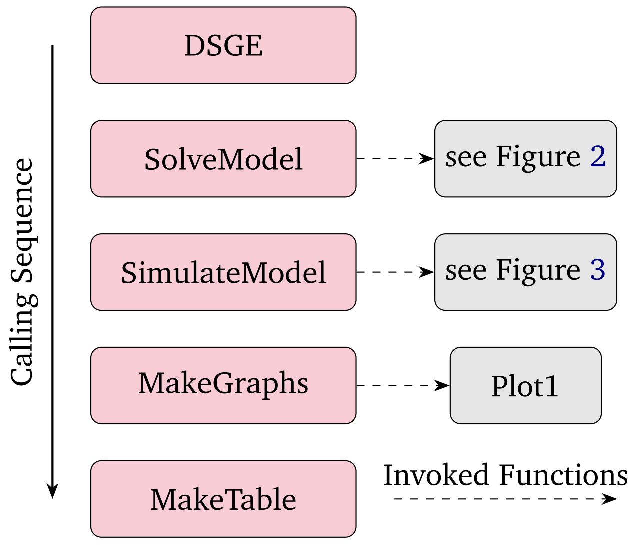
Extract all files from the archive `CorRRAM-M.rar` into a directory on your hard disk. Ensure that this directory is on Matlab's search path and check if the files listed in Table 1 are present. Run the script `Example_1_ND.m` to check the installation.

Table 1: Alphabetical list of files

baleig	GetFxt.m	Moments_FD.m
CCR.m	GetFxxt.m	Moments_TD.m
CDHesse.m	HamiltonFilter.m	Plot1.m
CDHesseRE.m	HPF.m	Quadratic.m
CDJac.m	Impulse.m	Simulate_NP.m
CDJacRE.m	lapack.c	Simulate_P2.m
Cubic.m	lapack.m	Simulate_P3.m
DSGE.m	lapackhelp.m	SimulateModel.m
Filtering.m	lapack.mexw64	SolveBA.m
First_Moments.m	Linear.m	SolveModel.m
GetFDH_ND.m	Lyapunov.m	Sylvester.m
GetFDHT_AD.m	MakeGraphs.m	tracem.m
GetFDHT_SD.m	MakeTable.m	

Figure 1 shows that the steps involved to set-up, solve, and simulate a model consist of calls of the functions `DSGE`, `SolveModel`, and `SimulateModel` in the sequence indicated by the vertical arrow. The function `MakeGraphs` graphs impulse response functions and `MakeTable` generates a table of second moments either written to a simple text file or to an Excel spread sheet. The functions called by `SolveModel` and `SimulateModel` are shown in Figure 2 and 3, respectively. The decision nodes in both figures show the role played by several switches.

Figure 1: Structure of the toolbox



The file lapack.mexw64 is an interface to a few routines from the linear algebra package Lapack (see http://www.netlib.org/lapack/lug/index.html). The CoRRAM functions HPF and Sylvester call functions from this package. If the file included in CoRRAM-M is not supported on your platform, delete it. The first time CoRRAM will try to use any of the Lapack routines, the function lapack.m will generate a new version of this file suitable for your environment.

Figure 2: The Function SolveModel

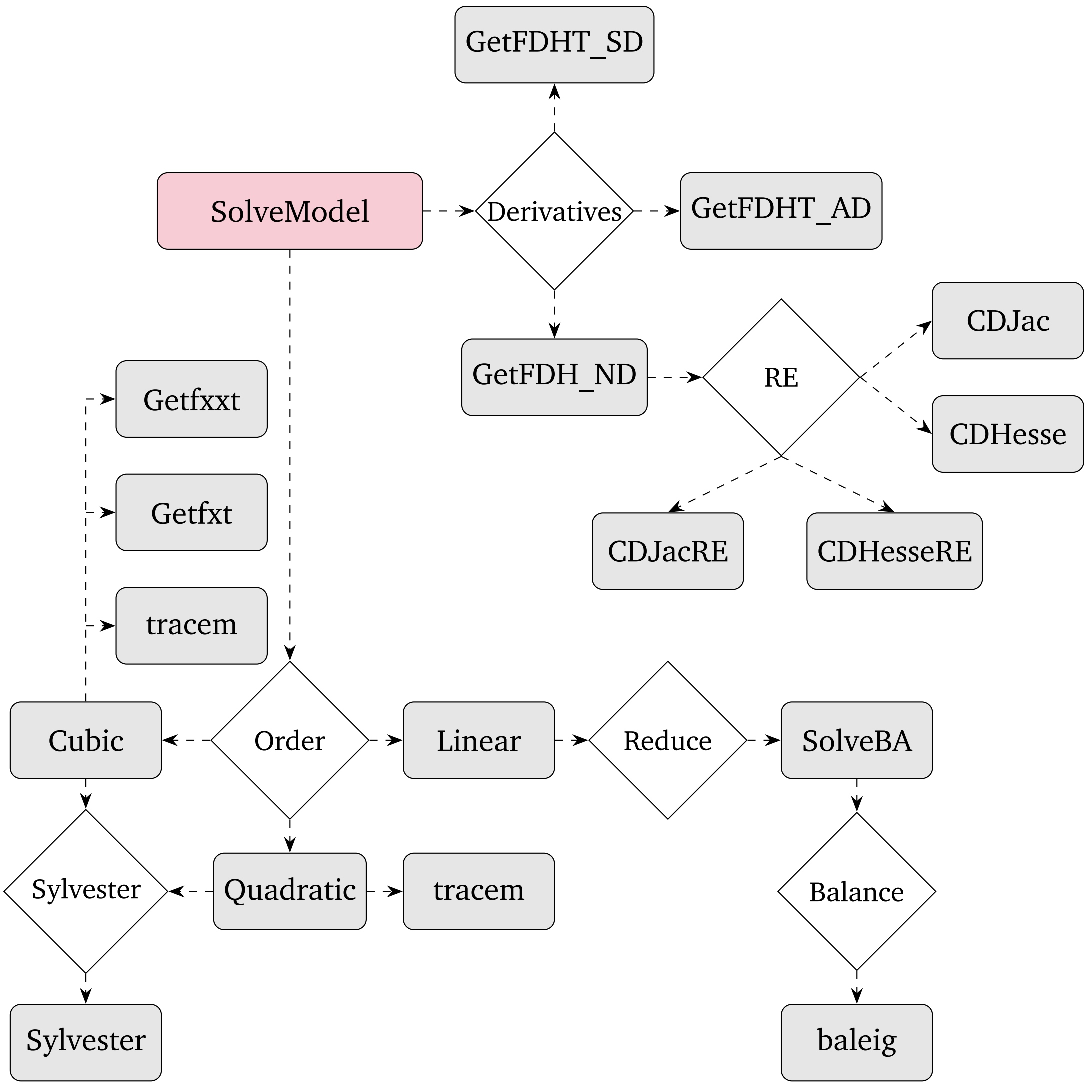
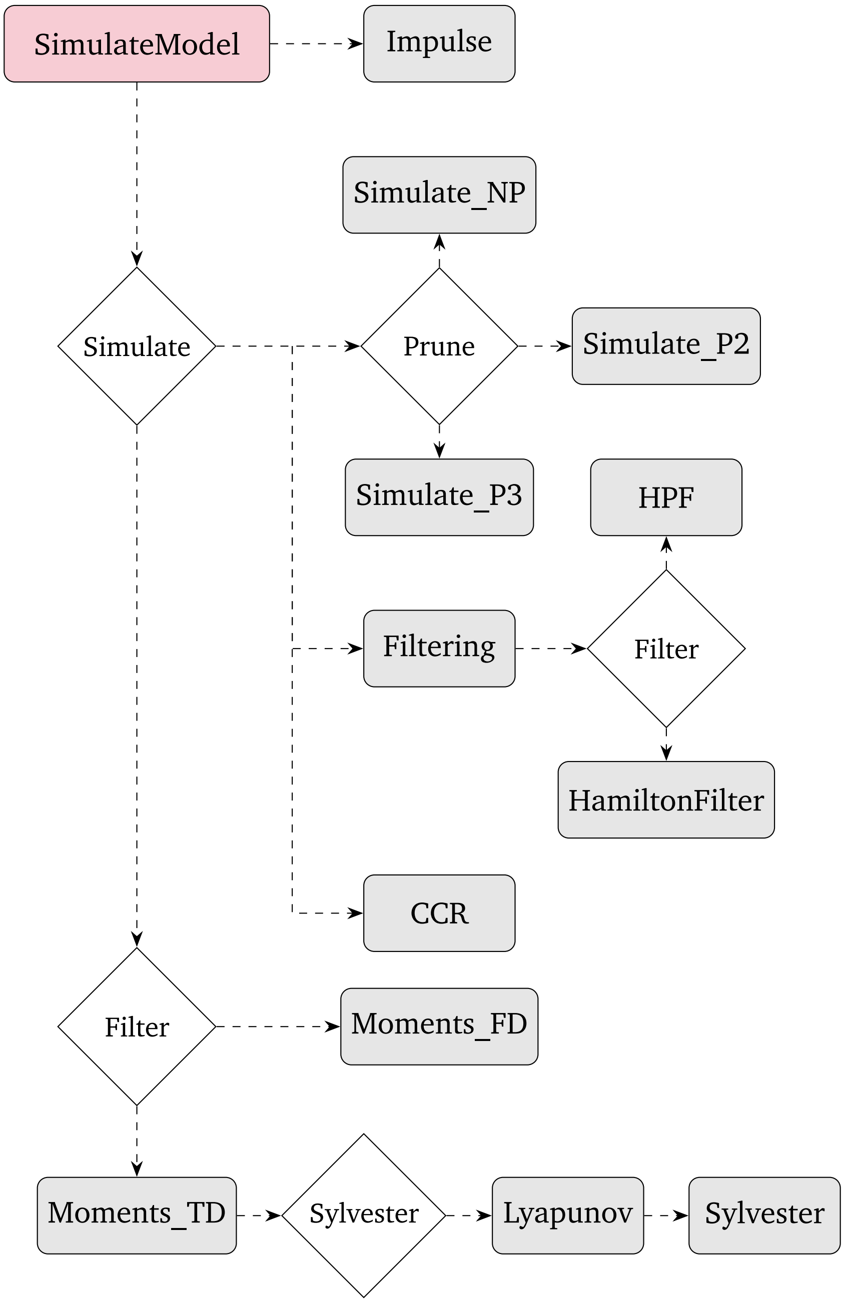


Figure 3: The Function SimulateModel



3 The canonical DSGE model

This section introduces the class of models which can be solved by the toolbox and explains the structure of the solution. A simple representative agent model will illustrate these concepts. In subsequent sections I will provide script listings that document various ways to solve and simulate this model with the CoRRAM toolbox.

I assume that you are sufficiently familiar with DSGE models so that I can restrict the presentation to the absolute minimum. For an introduction to DSGE models and to solution methods see Fernández-Villaverde et al. (2016) and Heer and Maußner (2009). Since this toolbox has been developed in parallel with the third edition of the latter book, I will frequently refer to Sections of this yet unpublished edition as Heer and Maußner (2021).

In the following $\mathbf{x} \in \mathbb{R}^{n(x)}$ denotes a column vector, whose $n(x) \in \mathbb{N}$ elements are real numbers, the superscript T denotes transposition, I_n denotes the n-dimensional identity matrix, and $A_{n \times m}$ denotes a matrix with n rows and m columns:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n(x)} \end{bmatrix} = \begin{bmatrix} x_1, & x_2, & \dots, & x_{n(x)} \end{bmatrix}^T, \; x_i \in \mathbb{R} \forall i = 1, \dots n(x),$$

$$I_n = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}_{n \times n}.$$

3.1 Variables

CoRRAM distinguishes three kinds of variables:

1. *state variables*, i.e., endogenous variables, whose values at the beginning of time t are predetermined (as, e.g., the stock of capital or the stock of nominal bonds),
2. *jump* or not predetermined variables, i.e., endogenous variables, whose values are determined within period t,
3. purely exogenous variables, the model's *shocks*,

Accordingly, there are three (column) vectors of variables:

- the vector of endogenous state variables $\mathbf{x}_t := \left[x_{1t}, x_{2t}, \dots, x_{n(x)t}\right]^T \in \mathbb{R}^{n(x)}$,
- the vector of jump variables $\mathbf{y}_t := \left[y_{1t}, y_{2t}, \dots, y_{n(y)t}\right]^T \in \mathbb{R}^{n(y)}$,
- the vector of shocks $\mathbf{z}_t := \left[z_{1t}, z_{2t}, \dots, z_{n(z)t}\right]^T \in \mathbb{R}^{n(z)}$.

It will be useful to define two additional vectors

$$\mathbf{w}_t := \begin{bmatrix} \mathbf{x}_t \\ \mathbf{z}_t \end{bmatrix} \in \mathbb{R}^{n(w)},\ n(w) = n(x) + n(z), \tag{1}$$

$$\mathbf{v}_t := \begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_t \\ \mathbf{z}_t \end{bmatrix} \in \mathbb{R}^{n(v)},\ n(v) = n(x) + n(z) + n(y). \tag{2}$$

3.2 Equations

The equilibrium law of motion of the model is given by:

$$\mathbf{0}_{(n(x)+n(y))\times 1} = \mathbb{E}_t \mathbf{g}(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t,) \tag{3a}$$

$$\mathbf{0}_{n(z)\times 1} = \mathbf{z}_{t+1} - R\mathbf{z}_t - \boldsymbol{\mu}(\sigma) - \boldsymbol{\eta}_{t+1}, \tag{3b}$$

$$\boldsymbol{\eta}_{t+1} := \sigma \Omega \boldsymbol{\epsilon}_{t+1}, \tag{3c}$$

$$\mathbf{0}_{n(z)\times 1} = \mathbb{E}_t \left(\boldsymbol{\epsilon}_{t+1}\right), \tag{3d}$$

$$I_{n(z)} = \mathbb{E}_t \left(\boldsymbol{\epsilon}_{t+1} \boldsymbol{\epsilon}_{t+1}^T\right), \tag{3e}$$

$$S_{n(z)\times n(z)^2} := \mathbb{E}_t \left[\boldsymbol{\eta}_{t+1} \left(\boldsymbol{\eta}_{t+1}^T \otimes \boldsymbol{\eta}_{t+1}^T\right)\right], \tag{3f}$$

$$\boldsymbol{\mu}(\sigma = 0) = \mathbf{0}_{n(z)\times 1},\ \boldsymbol{\mu}_\sigma(\sigma = 0) = \mathbf{0}_{n(z)\times 1},\ \boldsymbol{\mu}_{\sigma\sigma}(\sigma = 0) = -(I_{n(z)} - R)\text{diag}\left(\Sigma_z\right), \tag{3g}$$

where $\mathbb{E}_t$ denotes mathematical expectation as of period t and $\otimes$ the Kronecker product.

The process for the shocks in (3b) must be stationary. This requires that the eigenvalues of the matrix R must all be located within the unit circle. The covariance matrix Σ of the vector $\boldsymbol{\eta}_t$ is

$$\Sigma = \mathbb{E}\left(\boldsymbol{\eta}_t \boldsymbol{\eta}_t^T\right) = \mathbb{E}\left(\sigma \Omega \boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_t^T \Omega^T \sigma\right) = \sigma \Omega I_{nz} \Omega^T \sigma = \sigma^2 \Omega \Omega^T.$$

Thus, if the positive definite matrix Σ rather then Ω is given and the perturbation parameter σ is set equal to unity, Ω can be computed from

$$\Omega = C \Lambda^{1/2} C^T$$

where

- $\Lambda^{1/2}$ is the diagonal matrix with the square roots of the eigenvalues of the matrix Σ on the main diagonal, and
- C is the matrix of the normalized eigenvectors of Σ so that $CC^T = I_{n(z)}$.

The inclusion of the vector $\boldsymbol{\mu}$ allows to consider processes whose means are independent of their variance (mean preserving spread property).[1] The properties of the mean vector specified in (3g) are required for a correct first- and second-order perturbation solution as shown in Heiberger and Maußner (2019). The matrix Σ_z follows from the solution of the discrete Lyapunov equation

$$\Sigma_z = R\Sigma_z R^T + \Sigma.$$

The user who does not care about the mean preserving spread property can set both $\boldsymbol{\mu}$ and $\boldsymbol{\mu}_{\sigma\sigma}$ equal to the zero vector independent of the perturbation parameter σ.

The matrix S supplies the third moments of the model's innovations $\boldsymbol{\eta}_{t+1}$. It is required only insofar as third-order accurate solutions shall be computed.

3.3 Solution

The solution of the model are time invariant functions of the vector of state variables $\mathbf{w}_t := [\mathbf{x}_t^T, \mathbf{z}_t^T]^T$ and the parameter σ. They determine the future endogenous state variables $\mathbf{x}_{t+1}$ and the jump variables $\mathbf{y}_t$:

$$\mathbf{x}_{t+1} = \mathbf{h}^x(\mathbf{x}_t, \mathbf{z}_t, \sigma) := \begin{bmatrix} h^{x_1}(\mathbf{w}_t, \sigma) \\ \vdots \\ h^{x_{n(x)}}(\mathbf{w}_t, \sigma) \end{bmatrix}, \tag{4a}$$

$$\mathbf{y}_t = \mathbf{h}^y(\mathbf{x}_t, \mathbf{z}_t, \sigma) := \begin{bmatrix} h^{y_1}(\mathbf{w}_t, \sigma) \\ \vdots \\ h^{y_{n(y)}}(\mathbf{w}_t, \sigma) \end{bmatrix}. \tag{4b}$$

For $\sigma = 0$ the model (3) reduces to a system of deterministic non-linear difference equations. Perturbation methods assume that this system is stable and converges to the point

$$\mathbf{s} := \begin{bmatrix} \mathbf{v} \\ \mathbf{v} \end{bmatrix} \equiv \left[\mathbf{x}^T, \mathbf{y}^T, \mathbf{0}_{1\times n(z)}, \mathbf{x}^T, \mathbf{y}^T, \mathbf{0}_{1\times n(z)}\right]^T \in \mathbb{R}^{2(n(x)+n(z)+n(y))}. \tag{5}$$

Let

$$\bar{\mathbf{w}}_t := \begin{bmatrix} \mathbf{x}_t - \mathbf{x} \\ \mathbf{z}_t \end{bmatrix}$$

[1] For instance, the mean of a log-normally distributed variable Z, $\ln Z \sim N(\mu, \sigma^2)$ is equal to

$$\mathbb{E}(Z) = e^{\mu + \sigma^2/2}$$

and, thus, depends on the variance σ^2. Hence, setting $\mu = -\sigma^2/2$ yields $\mathbb{E}(Z) = 1$ irrespective of the size of σ^2.

denote the vector of deviations of the model's endogenous and exogenous states $\mathbf{w}_t$ from their stationary values $\mathbf{x}$ and $\mathbf{0}_{n(z)\times 1}$. The perturbation solution of the model up to the third order is given by

$$\begin{aligned}
\mathbf{x}_{t+1} &\simeq \mathbf{x} + \mathbf{h}^x_w \bar{\mathbf{w}}_t + \tfrac{1}{2}\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{ww}\bar{\mathbf{w}}_t + \tfrac{1}{2}\mathbf{h}^x_{\sigma\sigma}\sigma^2 \qquad (6a)\\
&\quad + \tfrac{1}{6}\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{www}\bar{\mathbf{w}}_t + \tfrac{1}{2}\sigma^2\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{\sigma\sigma w} + \tfrac{1}{6}\mathbf{h}^x_{\sigma\sigma\sigma}\sigma^3,\\
\mathbf{y}_t &\simeq \mathbf{y} + \mathbf{h}^y_w \bar{\mathbf{w}}_t + \tfrac{1}{2}\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{ww}\bar{\mathbf{w}}_t + \tfrac{1}{2}\mathbf{h}^y_{\sigma\sigma}\sigma^2 \qquad (6b)\\
&\quad + \tfrac{1}{6}\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{www}\bar{\mathbf{w}}_t + \tfrac{1}{2}\sigma^2\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{\sigma\sigma w} + \tfrac{1}{6}\mathbf{h}^y_{\sigma\sigma\sigma}\sigma^3.
\end{aligned}$$

The symbols $\mathbf{h}^i_{jkl}$ with $i \in \{x, y\}$ and $j, k, l \in \{w, \sigma\}$ denote the matrices of partial derivatives of the functions (4) with respect to the vector $[\mathbf{w}_t^T, \sigma]^T$. The CoRRAM command `SolveModel` computes these matrices. The solution is employed by the function `SimulateModel` to compute impulse responses of the variables in the vectors $\mathbf{x}_{t+1}$ and $\mathbf{y}_t$ to the shocks and to characterize the dynamic properties of the model via second moments obtained from simulations of the model (see Section 6). *Note that CoRRAM sets the perturbation parameter σ equal to unity.* (!)

3.4 Example

A simple example model shall illustrate these concepts. Later sections will present Matlab scripts that demonstrate the various ways to solve and simulate this model.

Assume a benevolent planner seeks time sequences for consumption C_t, labor input N_t and capital accumulation K_{t+1} that maximize

$$U_t := \mathbb{E}_t \sum_{s=0}^{\infty} \beta^s \frac{C_{t+s}^{1-\eta}(1-N_{t+s})^{\theta(1-\eta)} - 1}{1-\eta}, \quad \beta \in (0,1) \qquad (7a)$$

subject to the resource restriction

$$\begin{aligned}
Y_{t+s} &= Z_{t+s}(A_{t+s}N_{t+s})^{1-\alpha}K^{\alpha}_{t+s},\ \alpha \in (0,1), && (7b)\\
Y_{t+1} &= C_{t+s} + I_{t+s}, && (7c)\\
K_{t+s+1} &= (1-\delta)K_{t+s} + I_{t+s}, \qquad \delta \in (0,1]. && (7d)
\end{aligned}$$

The first-order conditions of this problem are

$$\begin{aligned}
\theta\frac{C_t}{1-N_t} &= (1-\alpha)\frac{Y_t}{N_t}, && (8a)\\
C_t^{-\eta}(1-N_t)^{\theta(1-\eta)} &= \beta\mathbb{E}_t C_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}\left(1-\delta+\alpha\frac{Y_{t+1}}{K_{t+1}}\right). && (8b)
\end{aligned}$$

The simulation tool of CoRRAM is able to handle two different specifications of the exogenous processes $\{Z_{t+s}\}_{s=0}^{\infty}$ and $\{A_{t+s}\}_{s=0}^{\infty}$. Simulations of more complicated processes are left to the user.

3.4.1 Trend stationary economy

The first scenario is defined by

$$
\begin{aligned}
\ln Z_{t+s+1} &= \rho_z \ln Z_{t+s} + \epsilon_{t+s+1}, && \text{(9a)}\\
\epsilon_{t+s+1} &\text{ iid } N(0,\sigma_\epsilon), && \text{(9b)}\\
A_{t+s+1} &= aA_{t+s},\ a \geq 1. && \text{(9c)}
\end{aligned}
$$

It depicts an economy where labor augmenting technical progress A_t is either absent (i.e., $a = 1$) or grows at the deterministic rate $a - 1 > 0$. In the latter case, output Y_t, consumption C_t, and the stock of capital K_t will grow on average at the rate $a - 1$. Shocks to total factor productivity Z_t trigger fluctuations around this trend.[2] However, the canonical model requires variables that approach constant values if the shocks are shut down. The trick is to scale the respective variables by the level of labor augmenting technical progress. We define

$$
y_t := \frac{Y_t}{A_t}, c_t := \frac{C_t}{A_t}, k_t := \frac{K_t}{A_t}
$$

and leave hours N_t unscaled. The stock of capital K_t is determined from previous decisions. Since A_t is a deterministic variable, the scaled variable k_t is also an endogenous state of the model. Output Y_t, consumption C_t, and labor input N_t are chosen in every period t to satisfy the first-order conditions. Therefore, y_t, c_t, and N_t constitute the vector of jump variables. Finally, the single purely exogenous variable is equal to $\ln Z_t$. Hence, the model implied by the first-order conditions (8), the production function (7b), and the law of capital accumulation (7d) is given by

$$
\begin{aligned}
g^1(\mathbf{x}_{t+1},\mathbf{y}_{t+1},\mathbf{z}_{t+1},\mathbf{x}_t,\mathbf{y}_t,\mathbf{z}_t) &\equiv y_t - Z_t N_t^{1-\alpha} k_t^{\alpha}, && \text{(10a)}\\
g^2(\mathbf{x}_{t+1},\mathbf{y}_{t+1},\mathbf{z}_{t+1},\mathbf{x}_t,\mathbf{y}_t,\mathbf{z}_t) &\equiv \theta \frac{c_t}{1-N_t} - (1-\alpha)\frac{y_t}{N_t}, && \text{(10b)}\\
g^3(\mathbf{x}_{t+1},\mathbf{y}_{t+1},\mathbf{z}_{t+1},\mathbf{x}_t,\mathbf{y}_t,\mathbf{z}_t) &\equiv y_t - c_t - i_t, && \text{(10c)}\\
g^4(\mathbf{x}_{t+1},\mathbf{y}_{t+1},\mathbf{z}_{t+1},\mathbf{x}_t,\mathbf{y}_t,\mathbf{z}_t) &\equiv ak_{t+1} - (1-\delta)k_t + c_t, && \text{(10d)}\\
g^5(\mathbf{x}_{t+1},\mathbf{y}_{t+1},\mathbf{z}_{t+1},\mathbf{x}_t,\mathbf{y}_t,\mathbf{z}_t) &\equiv 1 - \beta a^{-\eta} \frac{c_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}}{c_t^{-\eta}(1-N_t)^{\theta(1-\eta)}} \left(1 - \delta + \alpha \frac{y_{t+1}}{k_{t+1}}\right), && \text{(10e)}
\end{aligned}
$$

[2]The process (9a) has the unconditional mean

$$
\mathbb{E}(Z) = e^{\frac{\sigma_\epsilon^2}{2(1-\rho^2)}} > 1
$$

and increases with the variance of the innovations ϵ. Thus, the process (9a) is an example of a model which does not have the mean preserving spread property.

$$\mathbf{x}_t := k_t,\ \mathbf{y}_t := \begin{bmatrix} y_t, & c_t, & i_t, & N_t \end{bmatrix}^T,\ \mathbf{z}_t := \ln Z_t. \tag{10f}$$

We find the stationary solution of this model by setting $\ln Z_t$ equal to its stationary value of zero and by ignoring the expectations operator and as well as the time indices. Equation (10e), then, implies

$$\frac{y}{k} = \frac{a^{\eta} - \beta(1-\delta)}{\alpha\beta} \tag{11a}$$

so that equations (10c) and (10d) can be solved for c/k:

$$\frac{c}{k} = \frac{y}{k} - \frac{i}{k} = \frac{y}{k} - (a - 1 + \delta). \tag{11b}$$

Given y/k and c/k equation (10b) can be solved for N:

$$\frac{\theta}{1-\alpha}\frac{c/k}{y/k} = \frac{1-N}{N}. \tag{11c}$$

Finally, the solution for N can be used to solve equation (10a) for the level of the scaled capital stock k:

$$k = \left(\frac{y}{k}\right)^{\frac{1}{\alpha-1}} N \tag{11d}$$

so that y, c, and i can be recovered from y/k, c/k, and $i = (a - 1 + \delta)k$.

3.4.2 Difference stationary economy

The second growth scenario is a model where labor augmenting technical progress A_{t+s} is driven by a random walk with drift a while total factor productivity is a time invariant constant $Z \equiv 1$:

$$A_t = ae^{\zeta_t}A_{t-1}, \quad a \geq 0,\ \zeta_t \text{ iid } N(0, \sigma_\zeta). \tag{12a}$$

Again, we must scale output, consumption, and capital. Since A_t does change during period t, we use A_{t-1} as scaling factor so that $k_t := K_t/A_{t-1}$ remains an endogenous state variable. We employ the definitions

$$a_t := A_t/A_{t-1}, k_t := K_t/A_{t-1}, y_t := Y_t/A_{t-1}, c_t := C_t/A_{t-1}, i_t := I_t/A_{t-1}.$$

Note that these are an example of the more general definition

$$V_{it} = A_{t-1}^{\xi_i} v_{it} \tag{13}$$

with ξ_i being equal to one for the variables $V_{it} \in \{C_t, I_t, K_t, Y_t\}$ and zero for the variables $V_{it} \in \{a_t, N_t\}$. The model, thus, includes an additional variable, the growth factor a_t and its equations are

$$g^1(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv y_t - a_t^{1-\alpha} N_t^{1-\alpha} k_t^{\alpha}, \tag{14a}$$

$$g^2(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv \theta \frac{c_t}{1-N_t} - (1-\alpha)\frac{y_t}{N_t}, \tag{14b}$$

$$g^3(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv a_t - a e^{\zeta_t}, \tag{14c}$$

$$g^4(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv y_t - c_t - i_t, \tag{14d}$$

$$g^5(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv a_t k_{t+1} - y_t - (1-\delta)k_t + c_t, \tag{14e}$$

$$g^6(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv 1 - \beta a_t^{-\eta} \frac{c_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}}{c_t^{-\eta}(1-N_t)^{\theta(1-\eta)}} \left(1 - \delta + \alpha \frac{y_{t+1}}{k_{t+1}}\right). \tag{14f}$$

$$\mathbf{x}_t = k_t,\ \mathbf{y}_t := \begin{bmatrix} a_t, & y_t, & c_t, & i_t, & N_t \end{bmatrix},\ \mathbf{z}_t := \nu_t. \tag{14g}$$

The stationary versions of equations (14b)-(14f) imply the solutions for y/k, c/k, and N given in equations (11a)-(11c). The level of k follows from (10a) and is equal to

$$k = a\left(\frac{y}{k}\right)^{\frac{1}{\alpha-1}} N.$$

Note that the first three equations in the systems (10) and (14) contain only variables dated at period t. The solution algorithm of CoRRAM uses these equations to reduce the number of variables before it computes the solution from the remaining dynamic equations (10e)-(10f) and (14e)-(14f), respectively (see Section 5.4.2).

The next section will use this model to illustrate the usage of the toolbox. The parameters will be chosen as in Table 2.

4 Program interface

4.1 The DSGE class

CoRRAM stores the information about the model, its solution and simulation as well as the presentation of the simulation results in a Matlab class with name `DSGE`. The properties of this class provide the interface through which the user can interact with the toolbox. The `DSGE` class has two methods: the class construction method and the method which implements the policy functions (4).

Table 2: Calibration of the example model

Parameter	Description	Value
a	growth factor of output	1.004
α	capital share in output	0.36
β	discount factor	0.99
δ	rate of capital depreciation	0.025
η	coefficient of relative risk aversion	2.0
θ	utility weight of leisure	2.0
ρ_z	autocorrelation of TFP shock	0.95
σ_ϵ	standard deviation of TFP shock innovations	0.007
σ_ζ	standard deviation of growth factor shock	0.018

4.2 Class properties

The class properties fall into five categories:

i. properties of the model,

ii. properties that determine the solution

iii. properties that determine the simulation,

iv. properties that store the results of the solution and simulation,

v. properties that determine the presentation of the results in terms of graphs and tables.

Table 3 lists the properties of the `DSGE` class that store information about the model. These include the number of variables of each type, their stationary values, and – if desired – upper and lower bounds within which the variables are to be expected to remain in simulations of the model. In addition, the user must supply the matrices R and Ω from equation (3). For a third-order solution, the user must also provide the matrix S. The values of the model's parameters are required for the computation of the derivatives of the system of equations (3a). Users who choose to employ the Matlab symbolic toolbox for the computation of derivatives must also supply the symbolic definitions of the model's variables and parameters.

The class member `Var` is a structure that stores information about each of the model's variables. The elements of this structure that are shown in Table 4 are those through which the user can determine the content of tables and figures (see page 29). CoRRAM initializes all boolean members of `Var` with false (=0).

Tables 5 and 6 list those members of the `DSGE` class that provide CoRRAM with information about the solution and simulation of the model.

Table 3: Properties of the model

Property	Type	Purpose
nx	i	number of endogenous states
ny	i	number of jump variables
nz	i	number of shocks
nu	i	number of static equations
Rho	m	matrix R from equation (3b)
Omega	m	matrix Ω from equation (3c)
Mu	v	the vector μ from equation (3b),
Muss	v	the vector $\mu_{\sigma\sigma}$ defined in (3g),
Skew	m	matrix S from equation (3f)
Equations	str	name of the Matlab function that defines the system of equations (3a)
Var	s	structure, see Table 4
ParVal	v	vector with parameter values
ParSym	c	cell with strings that define the model's parameters
VarVal	v	vector that stores the stationary solution of the model
VarSym	c	cell with strings that define the model's variables
VarBnd	m	matrix with upper and lower bounds for the variables during simulations
VarXi	v	vector that stores the scaling of the variables.

Notes: Abbreviations: c=cell, i=integer, m=real matrix, s=structure, str=string, v=real vector.

Table 4: Elements of the Var structure

Element	Type	Default	Purpose
Name	str		name of the variable
Print	b	0	if true, include in table with second moments
Corr	b	0	if true, include a column with cross-correlations between this variable and all others
Rel	b	0	if true, include a column with standard deviations relative to the standard deviation of this variable
Plotno	i	0	number of the panel (out of eight) in which to plot the impulse response of this variable, see page 29

Notes: Abbreviations: b=boolean, i=integer, str=string.

Table 7 lists properties that store elements of the solution and simulation of the model. The entries in the first seven rows store intermediate results, whereas the remaining entries refer to the matrices of the approximate policy functions (6).

Table 8 lists those members of the DSGE class that provide CoRRAM with information about figures with and tables.

4.3 Class methods

An instance of the DSGE class is generated from the command

```
Mod=DSGE(nx,ny,nz,nu);
```

Table 5: Properties that control the solution

Element	Type	Range	Default	Purpose
`Derivatives`	str	'AD', 'ND', 'SD'	'ND'	determines the computation of derivatives
`etol`	r	$\mathbb{R}_{\geq 0}$	10^{-9}	see page 32
`itol`	r	$\mathbb{R}_{\geq 0}$	10^{-9}	see page 22
`order`	i	1,2,3	1	order of the solution
`Flags.Balance`	b	0,1	0	if true, the matrix pencil in equation (16) will be balanced. See page 26
`Flags.LoadDHT`	b	0,1	0	if true, use the results in `DS`, `HS`, and `TS`, see page 22
`Flags.RE`	b	0,1	0	if true, use Richardson extrapolation. See page 23
`Flags.Reduce`	b	0,1	1	if true, eliminate the static equations before solving the dynamic system, see Section 5.4.2
`Flags.Sylvester`	b	0,1	1	if true, use the function `Sylvester`

Notes: Abbreviations: b=boolean, i=integer, r=real, str=string.

Table 6: Properties that control the simulation

Element	Type	Range	Default	Purpose
`Flags.Ds`	b	0,1	0	if true, the model is driven by a difference stationary growth process, see page 27
`Flags.Log`	b	0,1	0	if true, the model's equations are in terms of logs of the variable instead of levels, see page 27
`Flags.Simulate`	b	0,1	1	if false, compute second moments from the matrices of the first-order solution
`Filter`	str	'None', 'HP', 'H'	'HP'	time series filter employed in the computation of second moments. See page 28
`hpl`	r	$\mathbb{R}$	1600	weight of the Hodrick-Prescott filter
`Prune`	i	0,2,3	0	for second (2)- or third-order (3) solutions, use pruning in simulations. See page 28
`maxlag`	i	$\mathbb{N}$	1	maximum lag considered in the computation of correlations and autocorrelations
`burnin`	i	$\mathbb{N}$	0	the number of periods used as burn in phase. See page 28
`nobs`	i	$\mathbb{N}$	120	length of simulated time series
`nofs`	i	$\mathbb{N}$	500	number of simulations
`inobs`	i	$\mathbb{N}$	10	length of impulse response functions
`seed`	i	$\mathbb{N}$	181191	seed for the random number generator. See page 28

Notes: Abbreviations: b=boolean, i=integer, r=real, str=string.

The command returns an instance of the class in the variable `Mod`, say. The four required arguments of the class constructor function are the number of endogenous state variables `nx`, the number of jump variables `ny`, the number of shocks `nz`, and the number of static equations `nu` (see page 14). The role of the first three arguments

Table 7: Results of the solution and simulation

Property	Type	Purpose
`FN`	v	left-hand sides of the system of equations (3a) evaluated at the stationary solution. See p. 22
`DN`	m	Jacobian matrix of (3a)
`DS`	str	symbolic Jacobian matrix of (3a). See page 22
`HN`	m	Hesse matrix of (3a)
`HS`	str	b Symbolic Hesse matrix of (3a). See page 22
`TN`	m	matrix of third-order derivatives of (3a)
`TS`	str	matrix of symbolic third-order derivatives of (3a). See page 22
`Hxw`	m	matrix $\mathbf{h}^x_w$ of (6a)
`Hxww`	m	matrix $\mathbf{h}^x_{ww}$ of (6a)
`Hxwww`	m	matrix $\mathbf{h}^x_{www}$ of (6a)
`Hxss`	v	vector $\mathbf{h}^x_{\sigma\sigma}$ of (6a)
`Hxsss`	v	vector $\mathbf{h}^x_{\sigma\sigma\sigma}$ of (6a)
`Hyw`	m	matrix $\mathbf{h}^y_w$ of (6b)
`Hyww`	m	matrix $\mathbf{h}^y_{ww}$ of (6b)
`Hywww`	m	matrix $\mathbf{h}^y_{www}$ of (6b)
`Hyss`	v	vector $\mathbf{h}^y_{\sigma\sigma}$ of (6b)
`Hysss`	v	vector $\mathbf{h}^y_{\sigma\sigma\sigma}$ of (6b)

Notes: Abbreviations: m=matrix, v=vector, str=string

Table 8: Properties that control graphs and tables

Element	Type	Range	Default	Purpose
`Flags.Excel`	b	0,1	0	if true, tables produced from the `MakeTable` command are also written to an Excel spreadsheet
`Flags.Grid`	b	0,1	1	if true, panels with impulse responses have grid. See page 30
`Flags.LegendBox`	b	0,1	1	if true, legends displayed in panels with impulse responses are placed in a box. See page 30
`Outfile`	str		Model	base name of files to which output is written, see pages 20 and 29.

Notes: Abbreviations: b=boolean, str=string.

is obvious. The fourth argument relates to the way CoRRAM computes the solution. Per default CoRRAM will reduce the model to a smaller dynamic model (see Section 5.4.2).

Once the model is solved, the policy functions (4) can be used. The respective command is

```
[x, y]=Mod.PFXY(w);
```

The command returns the left-hand side of equations (6) in the vectors `x` and `y`, respectively. Its argument `w` is the vector of state variables $\mathbf{w}_t$ as defined in (1).

5 Set-up and solution of a model

5.1 The main script

The first step in solving a model involves paper and pencil. As in the examples given in Section (3.4.1) you must write down the model's equations, determine the number and types of its variables, and develop a strategy to obtain the stationary solution of the model's deterministic version. In large scale models this may involve to solve a complex system of non-linear equations. This problem is outside the scope of this manual and Matlab provides resources to handle it. Furthermore, in many small and medium scale models the stationary solution can be found step by step as shown in (11).

The second step is to write a script that supplies CoRRAM with the information required to solve a model. This involves (i) the model's parameters, (ii) the stationary solution, and (iii) the model's equations. The code snippet in `Listing 1` shows the code for steps (i) and (ii) for the model in Section 3.4.1. It is taken from the file `Example_1_ND.m`:

Listing 1

```
% Parameters of the model: preferences
beta =0.99; % discount factor
eta  =2.0;  % CRRA
theta=2.0;  % utility weight of leisure

% production
a=1.004;    % growth factor
alpha=0.36; % capital share

% rate of capital depreciation
delta=0.025;n

% TFP shock
rhoz=0.95;
sigmaz=0.007;

% compute stationary solution
yk=(a^eta-beta*(1-delta))/(alpha*beta);
kn=yk^(1/(alpha-1));
ck=yk-(a-1+delta);
nstar=(theta/(1-alpha))*(ck/yk);
nstar=1/(1+nstar);
kstar=kn*nstar;
ystar=yk*kstar;
istar=(a-1+delta)*kstar;
cstar=ystar-istar;

% create an instance of the DSGE structure
nx=1;
ny=4;
nz=1;
nu=3;
BM=DSGE(nx,ny,nz,nu);

% supply information to the model
BM.Equations='Example_1_Eqs_ND';
```

```
37  BM.VarVal=[kstar;ystar;cstar;istar;nstar;0];
38  BM.ParVal=[a;alpha;beta;eta;delta;theta];
39  BM.Rho(1,1)=rhoz;
40  BM.Omega(1,1)=sigmaz;
41
42  BM.Derivatives='ND';
43  BM.order=2;
44  BM.Outfile='Example_1_ND';
45
46  BM=SolveModel(BM);
47  if BM.rc>0
48      error(BM.Messages{BM.rc})
49  end
```

The command in line 33 creates an instance of the DSGE class in the object BM. This object inherits the default properties of the class and can have an arbitrary name. The required arguments of the command are (in this order!) the number of endogenous states, the number of jump variables, the number of shocks, and the number of static equations (see Section 5.4.2).

Once an instance of the model has been created, you must supply the name of the function that encodes the model's equations, the stationary solution, the parameter values, and the parameters of the shock process (3b). For our example model the respective commands are in lines 36-40. The next lines provide CoRRAM with information about the solution algorithm. The solution requires the partial derivatives of the model's equations at the stationary solution. The settings in lines 42-44 instruct CoRRAM to use numeric differentiation and to compute a second-order approximate solution.

The command in line 46 solves the model. If CoRRAM fails to solve the model, an respective error message is printed to the screen (see Section 8) and additional information (for instance the eigenvalues) is written to a file. The name of this file is generated from the base name stored in the element Outfile by the command in line 44. CoRRAM appends _LogFile.txt to this name.

5.2 The system of equations

Depending on the setting of Derivatives (see Section 5.4.1), there are different requirements for the Matlab function that encodes the model's equations. For numeric derivatives an example of this function is given in the next listing.

Listing 2

```
function f = Example_1_Eqs_ND(s,eqno,Par)

% Parameters
a=Par(1);
alpha=Par(2);
beta=Par(3);
eta=Par(4);
delta=Par(5);
theta=Par(6);
```

```

f=ones(5,1);

% variables of the model, 1 refers to period t and 2 to period t+1 variables
k2=s(1);    k1=s(7);
y2=s(2);    y1=s(8);
c2=s(3);    c1=s(9);
i2=s(4);    i1=s(10);
n2=s(5);    n1=s(11);
z2=s(6);    z1=s(12);

% equations of the model
f(1)=y1-exp(z1)*(n1^(1-alpha))*(k1^alpha);
f(2)=theta*c1-(1-n1)*(1-alpha)*(y1/n1);
f(3)=y1-c1-i1;
f(4)=a*k2-(1-delta)*k1-i1;
f(5)=1-beta*(a^(-eta))*((c1/c2)^eta)*(((1-n2)/(1-n1))^(theta*(1-eta)))* ...
     (1-delta+alpha*(y2/k2));
if eqno~=0; f=f(eqno); end

return;
end
```

The function must have three mandatory arguments. The vector `s` contains the stationary solution as defined in (5), the integer `eqno` is required for the computation of second-order derivatives, and the vector `Par` provides the numeric values of the model's parameters. *The parameter values stored in this vector must be in the same order as in the command in line 38 of* `Listing 1`. In particular, note that the vector `s` represents the model's variables in this way:

$$\mathbf{s}_t := \begin{bmatrix} \mathbf{v}_{t+1} \\ \mathbf{v}_t \end{bmatrix}, \quad \mathbf{v}_t := \begin{bmatrix} \mathbf{x}_t \\ \mathbf{y}_t \\ \mathbf{z}_t \end{bmatrix} = \begin{bmatrix} x_{1t} \\ \vdots \\ x_{n(x)t} \\ y_{1t} \\ \vdots \\ y_{n(y)t} \\ z_{1t} \\ \vdots \\ z_{n(z)t} \end{bmatrix}, \tag{15}$$

i.e., the first $n(v) := n(x) + n(y) + n(z)$ elements hold the values of the model's variables as of date $t+1$ and the next $n(v)$ elements store the values as of date t. The ordering within the vectors $\mathbf{x}_t$ and $\mathbf{y}_t$ is taken from the ordering within the element `ValVar`. For instance, the command in line 37 of the listing `Listing 1` orders the elements of the vector $\mathbf{y}_t$ in the sequence: output, consumption, investment, and hours. Since CoRRAM sets up the vector $\mathbf{v}_t$ from the element `VarVal` *it is important to follow the ordering given in* (15) *when you set up* `VarVal`. Note that you are free to define the ordering of the variables within the vectors $\mathbf{x}_t$, $\mathbf{y}_t$, and $\mathbf{z}_t$. Since per definition the stationary values of elements of the vector $\mathbf{z}_t$ are equal to zero, the ordering within $\mathbf{z}_t$ is not taken from `VarVal` but from the order implied by the setting of `Omega` and `Rho`.

Note that it is not necessary to assign the elements of `s` to new local variables before you code the model's equations. Instead, you could also refer to the elements of this vector. For example, instead of line 22 in Listing 2 you could also write

```
22    f(1)=s(8)-exp(s(12))*(s(11)^(1-alpha))*(s(7)^alpha);
```

However, for me this is more error prone than the additional lines of code and to work with variable names that resemble the symbols in equations (10).

Finally, once you have assigned the right-hand sides of equations (10) to the vector `f`, do not forget to add the commands in line 28 of Listing 2. CoRRAM requires it.

5.3 Elements of the solution

The first step of the solution procedure is a check of the consistency of the model's equations. CoRRAM returns the left-hand sides of the model's equations evaluated at the stationary solution in the element `FN`. Note that at this point the left-hand side of (3a) should be equal to zero. Since floating point arithmetic is necessarily imprecise, CoRRAM returns an error, if at least one of the $2[n(x)+n(y)]$ equations has a left-hand side that exceeds the threshold `etol` in absolute value. You can increase this threshold, if you are convinced that your equations are correct.

CoRRAM also checks the matrix of first-order derivatives, returned in `DN` for over- or underflows. Depending on the setting of `order`, `HN` and `TN` store the matrix of second- and third-order partial derivatives of the system (3a).

If `Derivatives='DS'` is used, the matrices `DS`, `HS`, and `TS` store the symbolic Jacobian, Hessian, and matrix of third-order derivatives of the system (3a). If the same equations are repeatedly solved for different values of its parameters setting the flag `Flags.LoadDHT=1` instructs CoRRAM to reuse the symbolic information in these strings. This will speed up the solution.

Depending on the order of the solution the matrices $\mathbf{h}^x_w$, $\mathbf{h}^y_w$, $\mathbf{h}^x_{ww}$, $\mathbf{h}^y_{ww}$, $\mathbf{h}^x_{\sigma\sigma}$, $\mathbf{h}^y_{\sigma\sigma}$, $\mathbf{h}^x_{www}$, $\mathbf{h}^y_{www}$, $\mathbf{h}^x_{\sigma\sigma\sigma}$ and $\mathbf{h}^y_{\sigma\sigma\sigma}$ of (6) are returned in `Hxw`, `Hyw`, `Hxww`, `Hyww`, `Hxss`, `Hyss`, `Hxwww`, `Hywww`, `Hxsss`, and `Hysss`.

Note that with the choice of numeric derivatives the maximum order of approximation is equal to 2.

5.4 Solution options

There are several options that determine the solution of a model. They control the computation of derivatives and the computation of the linear part of the policy functions.

5.4.1 Derivatives

The settings of Derivatives determines the computation of the matrices of first-, second-, and third-order partial derivatives. There are three possible strings that can be assigned to Derivatives:

- 'AD' for algorithmic differentiation,
- 'ND' for numeric differentiation, and
- 'SD' for symbolic differentiation.

The example in Listing 1 employs numeric differentiation. In this case the setting of the flag Flags.RE determines whether CoRRAM employs central difference formulas (Flags.RE=0) to compute the derivatives or a more precise iterative scheme, known as Richardson extrapolation (see, e.g., Burden and Faires (2016), p. 180).

Algorithmic differentiation. Algorithmic differentiation requires that you have installed CasADi (see Andersson et al. (2018)), a freely available toolbox, which, among other things, implements algorithmic differentiation in Matlab. The next listing provides an example of the function that implements the model's equations suitable for algorithmic differentiation:

Listing 3

```
function fx=Example_1_Eqs_AD(Par);

import casadi.*;

% Parameters
a=Par(1);
alpha=Par(2);
beta=Par(3);
eta=Par(4);
delta=Par(5);
theta=Par(6);

v=SX.sym('s',12);

% variables of the model, 1 refers to period t and 2 to period t+1 variables
k2=s(1);    k1=s(7);
y2=s(2);    y1=s(8);
c2=s(3);    c1=s(9);
i2=s(4);    i1=s(10);
n2=s(5);    n1=s(11);
z2=s(6);    z1=s(12);

% equations of the model
f=[y1-exp(z1)*(n1^(1-alpha))*(k1^alpha);
   theta*c1-(1-n1)*(1-alpha)*(y1/n1);
   y1-c1-i1;
   a*k2-(1-delta)*k1-i1;
   1-beta*(a^(-eta))*((c1/c2)^eta)*(((1-n2)/(1-n1))^(theta*(1-eta)))*...
    (1-delta+alpha*(y2/k2));
    ];
```

```
31  fx=Function('BM',{s},{f});
32
33 return;
34 end
```

Vis-à-vis Listing 2 there are four important differences

1. The function must have only one argument: the vector of parameters Par.
2. The command in line 3, which makes the toolbox available.
3. The command in line 13, which creates a CasADi symbol. This symbol receives the elements of the vector $\mathbf{s}_t$ from (15).
4. The function must return a CasADi function object. The respective command is given in line 31. The name given to the function object, BM in the example, can be arbitrary and is not used by CoRRAM.

In Listing 1 you just have to change two lines of code: Instead of line 36 you would write

```
36 BM.Equations='Example_1_Eqs_AD';
```

and instead of line 42

```
42 BM.Derivatives='AD';
```

Symbolic differentiation. In order to use this functionality of CoRRAM you must have the Matlab symbolic toolbox installed. Here is the programm code that returns the model's equations for the example in Section 3.4.1

Listing 4

```
function [fx, s] = Example_1_Eqs_SD()

syms a eta delta theta;
beta=sym('beta');
alpha=sym('alpha');

syms k2 y2 c2 i2 n2 z2 k1 y1 c1 i1 n1 z1;

s=[k2 y2 c2 i2 n2 z2 k1 y1 c1 i1 n1 z1];

% equations of the model
fx=[y1-exp(z1)*(n1^(1-alpha))*(k1^alpha);
   theta*c1-(1-n1)*(1-alpha)*(y1/n1);
   y1-c1-i1;
   a*k2-(1-delta)*k1-i1;
   1-beta*(a^(-eta))*((c1/c2)^eta)*(((1-n2)/(1-n1))^(theta*(1-eta)))*...
    (1-delta+alpha*(y2/k2));
    ];

return;
end
```

Note the following differences vis-à-vis numeric differentiation:

1. The function has no arguments.
2. It returns both the symbolic equations in `fx` and the symbolic vector of variables in `s`.
3. You must define both the symbols used for the parameters and those used for the model's variables.

Since both `beta` and `alpha` are Matlab commands, you must use the syntax shown in lines 4 and 5 to override their meaning. Of course, you could also employ different names that could be included in the command in line 3. Note, you must return in `s` the symbols that make up the elements of the vector $\mathbf{s}_t$ defined in (15). Hence, what I have said above about the ordering of these elements does also apply here.

There are two additional lines of code that inform CoRRAM about the symbols used for parameters and variables. Instead of lines 36-42 in Listing 1 you would now write:

Listing 5

```
36  BM.Equations='Example_1_Eqs_SD';
37  BM.VarSym={'k';'y';'c';'i';'n';'z'};
38  BM.VarVal=[kstar;ystar;cstar;istar;nstar;0];
39  BM.ParSym={'a';'alpha';'beta';'eta';'delta';'theta'};
40  BM.ParVal=[a;alpha;beta;eta;delta;theta];
41  BM.Rho(1,1)=rhoz;
42  BM.Omega(1,1)=sigmaz;
43
44  BM.Derivatives='SD';
```

Note from line 37 that it is sufficient to provide the symbols of the variables without the additional 1 and 2 used in line 7 of Listing 4.

5.4.2 Model reduction

CoRRAM can compute the linear part of the policy function (6) in two different ways. The most straight forward approach is to linearize the system (3a) at the deterministic stationary solution. This yields the linear rational expectations model (see Heiberger et al. (2017) and Heer and Maußner (2021), Section 3.2.2):

$$B\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{w}}_{t+1} \\ \bar{\mathbf{y}}_{t+1} \end{bmatrix} = A \begin{bmatrix} \bar{\mathbf{w}}_t \\ \bar{\mathbf{y}}_t \end{bmatrix}, \tag{16}$$

where the matrices A and B consist of the first-order partial derivatives of (3a) and of the matrix R. The function `SolveBA` solves this system via the QZ factorization of the matrix pencil (B,A). The flag `Flags.Balance` determines whether this pencil will balanced as proposed by Lemonnier and Van Dooren (2006) to increase the numerical precision (see Heiberger et al. (2017), Section 3.3)

A more sensible procedure (see Heiberger et al. (2017) and Heer and Maußner (2021) Section 3.2.3) is to use the model's static equations and eliminate $n(u)$ of the variables. The resulting model has the same structure as the system (16) but only $n(x)+n(z)+n(y)-n(u)$ variables. Furthermore, in most cases the matrix B of the reduced model is invertible and the Schur factorization of $B^{-1}A$ can be employed to solve the model. For this reason, the default approach of CoRRAM is to reduce the model. If the matrix B is invertible, CoRRAM employs the Schur factorization, otherwise it solves the reduced model via the QZ factorization. The default setting of `Flags.Reduce=1` requires the user

i. to supply the number of equations, which involve only variables as of date t

ii. and *to present these $n(u)$ equations first in the function that encodes the model's equations* (3a).

ⓘ

This behavior can be overturned by setting the flag `Flags.Reduce=0`. In this case CoRRAM will not reference the value of `nu` and it can be set to an arbitrary value.

6 Simulation of a model

6.1 Simulation output

CoRRAM provides two tools to evaluate a model: impulse responses and second moments (see Heer and Maußner (2021), Chapter 4). The command

```
BM=SimulateModel(BM);
```

returns in `BM.IRF` the impulse responses and in `BM.Corr_m` tables with second-moments. If both the flag `Flags.Confidence` and the flag `Flags.Simulate` are set to true, CoRRAM also returns lower and upper bounds of the moments from simulations in `Corr_l` and `Corr_u`.

6.1.1 Impulse responses

`IRF` is a (potentially) three-dimensional array. The element `IRF(t,i,p)` stores the percentage deviation of variable v_{it} from its stationary value v_i at time t due to a shock in variable z_{pt}. The columns of each page $p=1,\dots,n(z)$ of `IRF` are ordered as the elements $i=1,2,\dots,n(x)+n(y)$ in the vector $\mathbf{v}_t$ in (15). Column $n(x)+n(y)+1$ stores the time path of shock z_{pt}. The number of periods is taken from the number stored in the element `inobs`.

6.1.2 Second moments

Corr_m (as well as Corr_l and Corr_u) is a three-dimensional array. Its page $l = 1$ stores the contemporaneous correlation coefficients between the model's variables. The ordering of the columns and rows of this matrix follows the ordering of the variables in the vector $\mathbf{v}_t$ in (15). The main diagonal stores the standard deviations of the model's variables. Pages $l = 2$ to $l = m$, where the value of m is taken from the value provided in maxlag, return the correlation between variable v_i at period t and variable v_j at period $t - l$. The respective main diagonals store the autocorrelations at lag l.

6.2 Simulation options

CoRRAM must know how it has to interpret the solved model. It reads this information from two flags. If the flag Flags.Log is false (the default), CoRRAM assumes that the solution pertains to the (scaled) levels of the variables. Otherwise CoRRAM assumes that the model was specified in terms of the natural logarithms of the variables. If the flag Flags.Ds is false (the default), CoRRAM assumes that the model either describes an economy without growth or a model with deterministic growth. Otherwise CoRRAM assumes that the economy is driven by a difference stationary growth process. In this case it employs the information given in VarXi to compute compute impulse responses and second-moments. Each element of the vector VarXi stores the scaling parameter ξ_i from equation (13).[3]

CoRRAM can compute second moments in two different ways:

i. from the matrices of the first-order approximate solution or

ii. from simulated time series.

The choice between i. and ii. is controlled by the flag Flags.Simulate. If set to false, the entries in Corr_m are computed analytically from the matrices of the first-order solution (see Heer and Maußner (2021), Sections 4.2.2 and Section 4.2.4). This will be done in the time domain, if no filter is specified, Filter='None', or in the frequency domain, if the Hodrick-Prescott filter (see Hodrick and Prescott (1997)) is specified, i.e., Filter='HP'. In this case, the filter weight is taken from hpl.

If the flag Flags.Simulate is set to true, the second moments are obtained as averages from simulated time series. The length of the simulated series is taken from nobs and the number of simulations from nofs. If the flag Flags.Checkbounds is set to true and upper *and* lower simulations bounds are provided in VarBnd, CoRRAM checks at each step whether the simulated values fall outside the specified bounds. If

[3]For more details on scaling see Heer and Maußner (2021), Sections 4.2 and 4.4. See also the example script Example_2_ND.m for the solution and simulation of the model of Section 3.4.2

this is true for at least one of the model's variables at time t, the simulation is stopped and the respective time series ignored in the computation of averages. Furthermore, if `Flags.Confidence` is true, the arrays `Corr_l` and `Corr_u` store for each element of `Corr_m` the lower and upper bound of the interval to which 95 percent of all simulated second-moments belong. The second-moments will be computed from filtered simulated time series depending on the setting of `Filter`. Besides the well-known Hodrick-Prescott filter the filter recently proposed by Hamilton (2017) is available.

Simulations are further controlled by the settings of `Prune`, `burnin`, and `seed`. `Prune` can take on three values:[4]

0 time paths are simulated using the policy functions (6) up to the order given in the element `order`,

2 time paths are simulated using the second-order pruned solution,

3 time path are simulated using the third-order pruned solution.

Pruning eliminates explosive behavior that might emerge from second- and third-order approximate solutions.

CoRRAM always starts simulations at the stationary solution of the model. To eliminate the effect of the model's transient behavior, CoRRAM increases the number of periods in each simulation by the number provided in the element `burnin` and discards the same number of periods before it computes second-moments from the simulated time series.

CoRRAM uses the Matlab random number generator `randn` to simulate draws from a standard normal distribution for the innovations $\boldsymbol{\epsilon}_{t+1}$ from equation (3d). CoRRAM seeds the random-number generator with the integer value provided in the element `seed`. Unless you change this value in different runs of a model, the results in `Corr_m` (as well as in `Corr_l` and `Corr_u`) will not change. You can also use the same value of `seed` in different models. This will guarantee that any differences in simulated second-moments are not random but are the result of differences in the model's parameters and/or equations.

7 Graphs and tables

CoRRAM is able to produce graphs and tables from the results stored in the elements `IRF`, `Corr_m`, `Corr_l`, and `Corr_u`. The `MakeGraphs` function generates one figure with impulse responses for each of the model's shock. The `MakeTable` function generates a table with second-moments. This table is written to an ASCII file. CoRRAM

[4]See Andreasen et al. (2018) on pruning.

generates the name of this file from the string in Outfile and appends the file extension .txt. If the flag Flags.Excel is set to true, MakeTable also writes its output to an Excel spread sheet. The name of the respective file is equal to the string in Outfile with file extension .xlsx.

There are two ways to instruct CoRRAM which kind of output the user desires: via the structure Var or via additional arguments of the function MakeGraphs and MakeTable.

Using Var. The first and probably more cumbersome way is to use the Var structure (see Table 4). For each variable you can specify its name in Name. This name will be used in plots of impulse responses and in tables with second-moments.

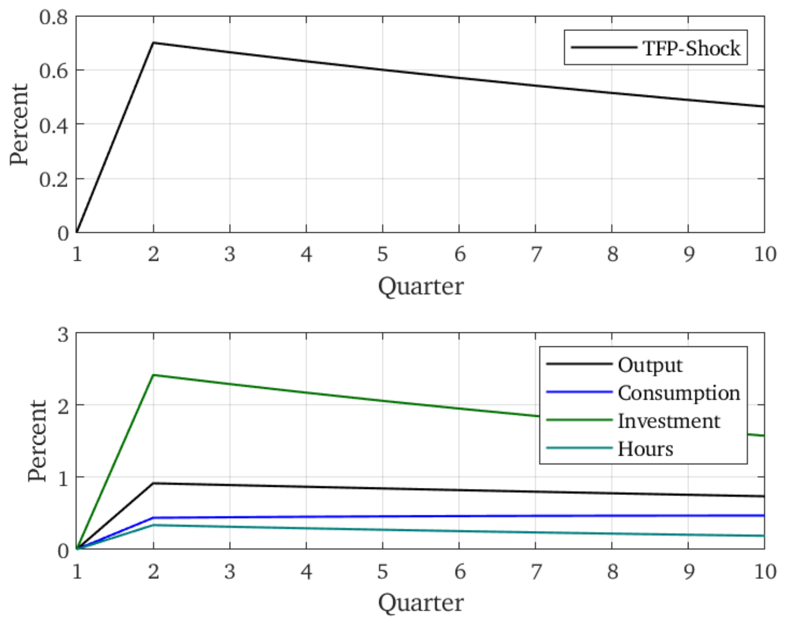
Whether or not a variable will be included in a graph depends on the setting of Plotno. If set to zero (the default), the respective variable will be excluded. If set to a number between 1 and 8, the impulse response of the variable will be plotted in the respective panel of the figure. *CoRRAM accommodates up to eight panels.* ⓘ

The setting of Print determines whether CoRRAM includes the respective variable in a table. If Print is true, CoRRAM prints the standard deviation of the respective variable. If in addition Corr is set to true, CoRRAM includes a column with correlations between the respective variable and all other variables for which Print is true. You can also use the boolean variable Rel and tell CoRRAM to produce a column of standard deviations between the other variables relative to the standard deviation of the variable for which Rel is true.

As an example, consider the code snippet in the next listing. It generates Figure 4 and Table 9 for the model of Section 3.4.1.

Listing 6
```
BM=SimulateModel(BM)

BM.Var(2).Name='Output';
BM.Var(2).Print=1;
BM.Var(2).Corr=1;
BM.Var(2).Plotno=2;

BM.Var(3).Name='Consumption';
BM.Var(3).Print=1;
BM.Var(3).Plotno=2;

BM.Var(4).Name='Investment';
BM.Var(4).Print=1;
BM.Var(4).Plotno=2;

BM.Var(5).Name='Hours';
BM.Var(5).Print=1;
BM.Var(5).Plotno=2;

BM.Var(6).Name='TFP-Shock';
BM.Var(6).Plotno=1;

MakeGraphs(BM,'Charter');
MakeTable(BM);
```

Figure 4: Impulse responses



The figure shows in its upper panel the time path of the TFP-shock relative to its deterministic value of one in percent. The panel below displays the percentage deviations of output, consumption, investment, and hours from their respective stationary values. The typeface employed for axes number, labels, and the legend is determined by the second argument of the `MakeGraphs` command. In the example, it is the Charter font, in which this manual is typeset. The user can switch on or off the grid in all panels through the flag `Flags.Grid`. Analogously, if `Flags.LegendBox` is true, CoRRAM places the legends in boxes. If you want more control over the appearance of the graphs, you are free to adapt the program code in the function `Plot1.m`. There you can change colors, the positioning of the legends, the fonts and font sizes employed in setting axis numbers and labels, etc.

Table 9 shows the output of `MakeTable`. The first five lines present information about the simulation procedure, the employed filter, and the order of solution. The next four lines display the table of second-moments. The columns of this table are explained in the last four lines of text.

Optional arguments. The second way to graph impulse responses and to print tables are optional arguments of the functions `MakeGraphs` and `MakeTable` as shown in the next `Listing 7`. The Matlab commands presented there yield the same output as the commands in `Listing 6` (see Figure 4 and Table 5).

Table 9: Second Moments

```
Second moments from 500 simulations of length=120 and burnin=50.
Bounds were not checked
Filter: HP
Filter weight=  1600
Quadratic policy functions were used.

     Output  1.16  1.00  0.68
Consumption  0.56  0.99  0.70
 Investment  3.06  1.00  0.68
      Hours  0.43  0.99  0.68

Column  1 : Variable
Column  2 : Standard Deviation
Column  3 : Correlation with Variable  Output
Column  4 : First Order Autocorrelation
```

Listing 7

```
76 Names={'Output','Consumption','Investment','Hours','TFP-Shock'};
77 plotinfo=[[2 2];[3 2];[4 2];[5 2];[6 1]];
78 MakeGraphs(BM,'Charter',Namestr,plotinfo);
79
80 printidx=[2;3;4;5];
81 corridx=[2];
82 MakeTable(BM,Namestr(1:4),printidx,corridx);
```

The function `MakeGraph` has two optional arguments. The first argument `Namestr` is a Matlab cell array of character vectors that supply the names of those variables for which the function shall graph impulse responses. *Note that this list must include names for all shocks of the model at its end.* The names must be sorted according to the ordering of the variables in the vector $\mathbf{v}_t$ defined in (15). The second argument `plotinfo` is a matrix. Its first column stores the position numbers of the variables whose impulse responses shall be displayed, the second column receives the number of the panel, in which the respective graph shall be plotted. The command in line 77 instructs CoRRAM to plot the impulse responses of output, consumption, investment, and hours in panel two and the impulse response of the TFP-shock in panel one.

The function `MakeTable` has three optional arguments. The argument `Namestr` supplies the names of all variables that shall be included in the table. The vector `printidx` stores the respective positions of these variables in the vector $\mathbf{v}_t$. For each element of the vector `printidx` CoRRAM includes a column with correlations between the respective variable and the variables specified in `corridx`.

8 Error messages

There are two kinds of error messages: messages generated by Matlab and by CoRRAM, respectively.

(i) Matlab error messages will usually indicate that something is wrong with your script. For instance, if you call `DSGE` without arguments, Matlab will terminate with the message “Not enough input arguments”. However, I cannot exclude that error messages originate from a bug in my code. I provide CoRRAM “as is” without warranty of any kind. The core of this program has been in use by myself for quite some time and I have used either the programs or the results of others to check the correctness of the solution algorithm.

(ii) There are a few instances where CoRRAM issues either a warning message or terminates with an error message. For instance CoRRAM warns, if the first-order solution involves complex numbers whose imaginary part exceeds `itol`. In some instances CoRRAM will stop execution, for instance, if you want to simulate a model before you have solved it. The respective warning and error messages are hopefully sufficiently instructive to eliminate the source of the problem.

The function `SolveModel` CoRRAM returns the following error codes and messages displayed in Table 10.

Table 10: Error messages

#	Message text
1	Model does not exist
2	Invalid stationary solution
3	Jacobian includes NaNs and/or Infs
4	Schur failed
5	Instable model
6	Indeterminate model

Error code 1 simply means that Matlab was not able to find the function that encodes the model’s equations. Error code 2 indicates that the stationary solution supplied in `VarVal` does not solve the system of equations. Error code 3 may occur, if one of the first-order partial derivative could not be computed. Error code 4 occurs, if one of the matrices from the QZ or the Schur factorization is not invertible. In this case, a solution does not exist (see, e.g., Heiberger et al. (2015)). Error code 5 indicates that at least one out of $n(x)$ eigenvalues is outside the unit circle so that the solution is instable. If not all of the $n(y)$ or - in the case of a reduced model - $n(y)-n(u)$ eigenvalues are outside the unit circle, the solution is indeterminate, indicating sunspot equilibria (see, e.g. Burnside (1999), Section 5 or Farmer (1999), Chapter 3).

9 Function reference

This section summarizes the syntax of the main functions from Figure 1.

DSGE. There are four versions of this command:

i. `Mod=DSGE(nx,ny,nz,nu);`

ii. `Mod=DSGE(nx,ny,nz,nu,xy);`

iii. `Mod=DSGE(nx,ny,nz,nu,xy,Symbols);`

iv. `Mod=DSGE(nx,ny,nz,nu,xy,Symbols,'Names',Namestr);`

Each of the four commands creates an instance of the `DSGE` class in the variable `Mod`. The required arguments are the number of endogenous states `nx`, jump variables `ny`, shocks `nz`, and static equations `nu`. Optionally you can supply the stationary solution $[\mathbf{x}^T,\mathbf{y}^T]^T$ (see (5)) in the $n(x)+n(y)$ vector `xy`, the symbols of the elements of the vector $\mathbf{v}_t$ (see (15)) in the cell array of character vectors `Symbols`, and the names of the elements of the vector $\mathbf{v}_t$ in the cell array of character vectors `Namestr`. The second version of the command is a substitute for the command in line 37 of `Listing 1`. The command in line 37 of `Listing 5` becomes redundant, if you employ version iii. Finally, you can supply names for all elements of the `Var` structure with version iv. In this case statements as in lines 53, 58, 62, 66, and 70 of `Listing 6` are redundant.

SolveModel. This command has only the version

```
Mod=SolveModel(Mod);
```

The argument of the function `SolveModel` is an instance of the `DSGE` class and the function returns a new instance of this class. Thus, you could also type

```
Mod_new=SolveModel(Mod);
```

SimulateModel. The single version of this command is

```
Mod=SimulateModel(Mod);
```

As the `SolveModel` function the argument is an instance of the `DSGE` class. The function checks whether the model has been solved, simulates the model, and returns its results in the elements of an new instance of the `DSGE` class. Hence

```
Mod_new=SimulateModel(Mod);
```

is also valid.

MakeGraphs. As explained in Section 7, there are two versions of this command:

i. `MakeGraphs(Mod,Fontstr);`

ii. `MakeGraphs(Mod,Fontstr,Namestr,plotinfo);`

The first version expects that the information required to plot impulse responses is supplied via the `Var` structure. The second required input is a string that holds the name of font used for axis numbers and labels as well as for the legend. The second version expects the names of all variables for which impulse responses shall be graphed and the names of all shocks in the cell array of character vectors `Namestr`. The matrix `plotinfo` must supply the integers that indicate the position numbers of the variables for which impulse responses shall be graphed in its first column and the panel number in which these shall be plotted in its second column.

MakeTable. The two versions of this command are:

i. `MakeTable(Mod);`

ii. `MakeTable(Mod,Namestr,printidx,corridx);`

The first version accepts an instance of the `DSGE` class as single argument, checks if the model has been simulated before, employs the information supplied in the structure `Var`, and generates the desired table of second-moments. The second version draws this information from the three optional arguments. The function employs the names supplied in the cell array of character vectors `Namestr` for the first column of the table. Its second column shows the standard deviations of these variables. The respective positions are drawn from the vector `printidx`. For each element of the vector `corridx` the function produces an additional column with correlations between the respective variable and the variables whose indices appear in `printidx`.

First_Moments. The command

```
[Ex,Ey,Ez]=First_Moments(Mod);
```

returns the deviations of the unconditional first moments of the variables of a DSGE model from their stationary values as explained in Heiberger and Maußner (2019). The function receives an instance of the DSGE class and returns in `Ex`, `Ey`, and `Ez` the first moments of the vector of endogenous states $\mathbf{x}_\mathbf{t}$, the vector of jumps $\mathbf{y}_t$, and the vector of shocks $\mathbf{z}_t$, respectively.

References

Andersson, J., J. Gillis, and M. Diehl (2018). User documentation for casadi v3.44. Manuscript. http://casadi.sourceforge.net/v3.4.4/users_guide/casadi-users_guide.pdf.

Andreasen, M. M., J. Fernández-Villaverde, and J. F. Rubion-Ramírez (2018). The pruned state-space system for non-linear DSGE models: Theory and empirical applications. *Review of Economic Studies 85*, 1–49.

Burden, R. L. and D. Faires (2016). *Numerical Analysis, 10th Edition*. Pacific Grove, CA: Brooks/Cole, Cengage Learning.

Burnside, C. (1999). Real business cycle models: Linear approximation and gmm estimation. https://ideas.repec.org/c/dge/qmrbcd/76.html.

Farmer, R. E. (1999). *The Macroeconomics of Self-Fulfilling Prophecies, 2nd Ed.* Cambridge, MA and London: MIT Press.

Fernández-Villaverde, J., R. Ramírez, and F. Schorfheide (2016). Solution and estimation methods for DSGE models. Working Paper W21862, NBER.

Hamilton, J. D. (2017). Why you should never use the hodrick-prescott filter. Working Paper W23429, National Bureau of Economic Research (NBER).

Heer, B. and A. Maußner (2009). *Dynamic General Equilibrium Modelling. 2nd Edition*. Berlin: Springer.

Heer, B. and A. Maußner (2021). *Dynamic General Equilibrium Modelling. 3rd Edition*. Berlin: Springer. Forthcoming.

Heiberger, C., T. Klarl, and A. Maußner (2015). On the uniqueness of solutions to rational expectations models. *Economic Letters 128*, 14–16.

Heiberger, C., T. Klarl, and A. Maußner (2017). On the numerical accuracy of first-order approximate solutions to DSGE models. *Macroeconomic Dynamics 21*, 1811–1826.

Heiberger, C. and A. Maußner (2019). Perturbation solutions and welfare costs of business cycles in DSGE models. *Journal of Economic Dynamics and Control*. http://doi.org/10.1016/j.jedc.2019.103819.

Hodrick, R. J. and E. C. Prescott (1997). Postwar U.S. business cycles: An empirical investigation. *Journal of Money, Credit and Banking 29*(1), 1–16.

Lemonnier, D. and P. Van Dooren (2006). Balancing regular matrix pencils. *SIAM Journal on Matrix Analysis and Applications 28(1)*, 253–263.