CoRRAM-M: User Guide

Prof. Dr. Alfred Maußner

May 19, 2017

Contents

1 INTRODUCTION

This document explains the use of my Matlab toolbox CoRRAM. The acronym CoRRAM 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. 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. CoRRAM includes the Matlab functions jacobianest and hessianest written by D'Errico (2007) which employ numerical differentiation to compute first- and second-order partial derivatives of the model's equations. 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.

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

```
EM=DSGE(nx,ny,nz,nu,v);
```

is the command which creates a new instance of the `DSGE` class in the object `EM`. 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 five required arguments (shown above), one optional argument, and the named argument `Names`, a cell array with string elements.

- 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

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

  the command 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 your model. Section 6 introduces the simulation tool and Section 7 concludes with a list of error messages.

2 Installation

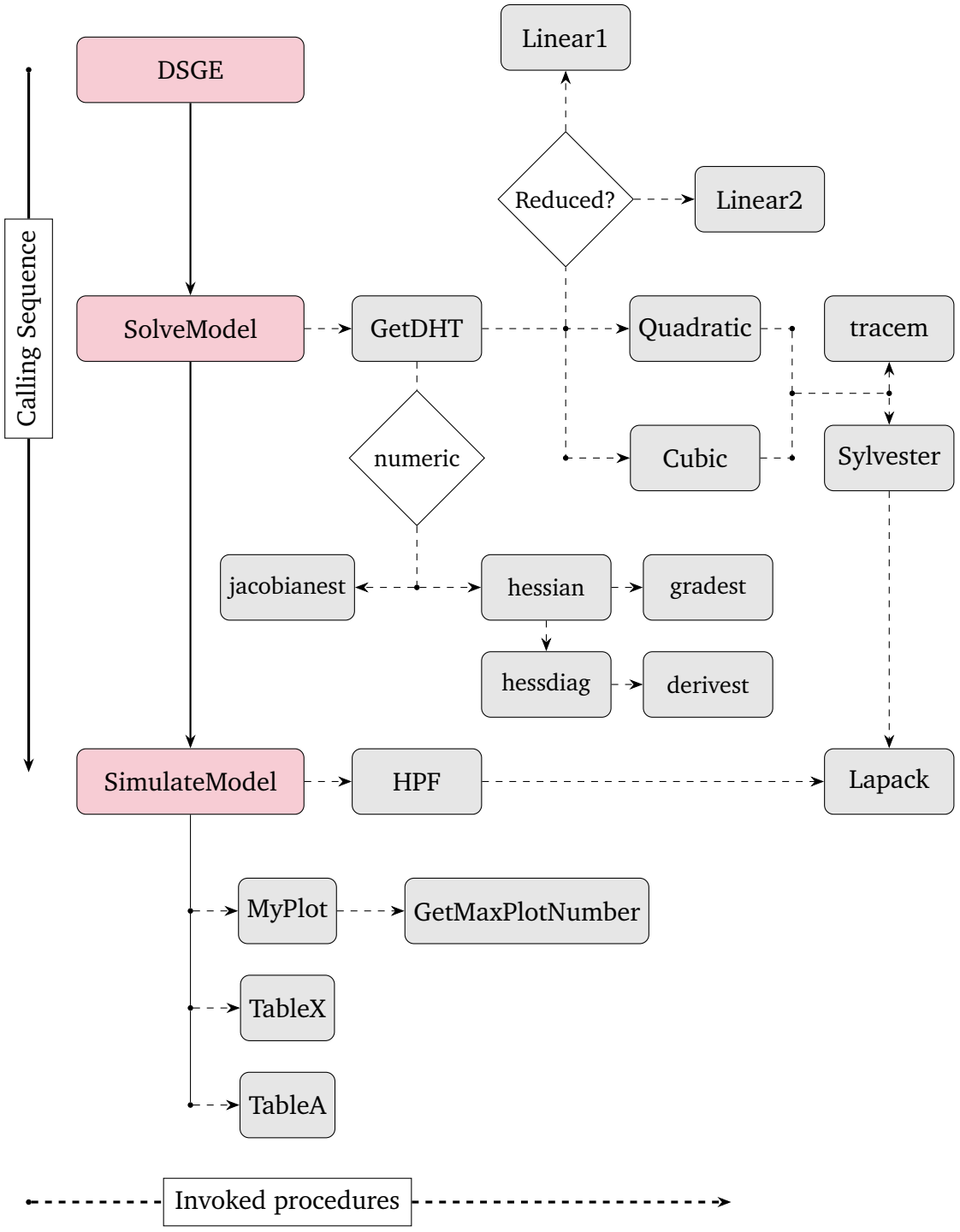
The toolbox consists of the Matlab files listed in Table 1. Simply copy the files on your hard disk and ensure that the directory that received the files is on Matlab's search path.

Table 1: Alphabetic list of files

Cubic.m	lapackhelp.m
derivest.m	Linear1.m
DSGE.m	Linear2.m
GetDHT.m	MyPlot.m
GetMaxPlotNumber	Quadratic.m
gradest.m	SimulateModel.m
hessdiag.m	SolveModel.m
hessian.m	Sylvester.m
HPF.m	TableA.m
jacobianest.m	TableX.m
lapack.c	tracem.m
lapack.m	

Figure 1 shows how the Matlab functions and scripts are interrelated. Basically, the steps involved to set up, solve, and simulate your model consist in calls of the functions `DSGE`, `SolveModel`, and `SimulateModel` in the order indicated by the

Figure 1: Structure of the toolbox



vertical arrow. These functions invoke other function or scripts as indicated by the horizontal arrows.

In addition to the core files, there are two scripts and the related Matlab functions listed in Table 2. The scripts solve and simulate the examples presented in Section 3.4. Run the script EM_Numeric.m to test your installation. If your Matlab includes the symbolic toolbox you can also run the script EM_Analytic.m.

Table 2: Example scripts and related functions

EM_Analytic.m	EM_Numeric.m
EM_Eqs.m	EM_Sys.m
EM_Eqs_ds.m	EM_Sys_ds.m
EM_Eqs_ds_log.m	EM_Sys_ds_log.m
EM_Eqs_log.m	EM_Sys_log.m

The first time you use the toolbox with the option `syl=true` (more on that latter in this document) the function `lapack` will create the file lapack.mexw64 which provides an interface to the linear algebra package Lapack (see `http://www.netlib.org/lapack/lug/index.html`). The CoRRAM functions `HPF` and `Sylvester` call functions from the package.

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). For the canonical model and the related algorithms to solve this class of models see my companion paper Maußner (2017).

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 $\mathbf{x}_t := \left[x_1, x_2, \ldots, x_{n(x)}\right]^T \in \mathbb{R}^{n(x)}$,
- the vector of jump variables $\mathbf{y}_t := \left[y_1, y_2, \ldots, y_{n(y)}\right]^T \in \mathbb{R}^{n(y)}$,
- the vector of shocks $\mathbf{z}_t := \left[z_1, z_2, \ldots, z_{n(z)}\right]^T \in \mathbb{R}^{n(z)}$.

It will be useful to define two additional vectors

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

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

3.2 Equations

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

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

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

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

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

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

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

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

The process for the shocks in (1b) 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 scaling factor σ is set equal to unity, Ω can be computed from

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

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 $CC' = I_{n(z)}$.

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 current control variables $\mathbf{y}_t$:

$$\mathbf{x}_{t+1} = \mathbf{h}^x(\mathbf{x}_t, \mathbf{z}_t, \sigma) := \begin{bmatrix} h^{x_1}(\mathbf{w}_t, \sigma) \\ \vdots \\ h^{x_{n(x)}}(\mathbf{w}_t, \sigma) \end{bmatrix}, \tag{2a}$$

$$\mathbf{y}_t = \mathbf{h}^y(\mathbf{x}_t, \mathbf{z}_t, \sigma) := \begin{bmatrix} h^{y_1}(\mathbf{w}_t, \sigma) \\ \vdots \\ h^{y_{n(y)}}(\mathbf{w}_t, \sigma) \end{bmatrix}. \tag{2b}$$

For $\sigma = 0$ the model (1) reduces to a system of deterministic non-linear difference equations. Perturbation methods assume that this system is stable and converges to the point

$$\left[\mathbf{v}^T, \mathbf{v}^T\right]^T \equiv \left[\mathbf{x}^T, \mathbf{0}_{1\times n(z)}, \mathbf{y}^T, \mathbf{x}^T, , \mathbf{0}_{1\times n(z)}, \mathbf{y}^T\right]^T \in \mathbb{R}^{2(n(x)+n(z)+n(y))}. \tag{3}$$

Let

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

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

$$\begin{aligned} \mathbf{x}_{t+1} \simeq\ & \mathbf{x} + \mathbf{h}^x_w \bar{\mathbf{w}}_t + \tfrac{1}{2}\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{ww}\bar{\mathbf{w}}_t + \tfrac{1}{2}\mathbf{h}^x_{\sigma\sigma}\sigma^2 \\ & + \tfrac{1}{6}\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{www}\bar{\mathbf{w}}_t + \tfrac{1}{2}\sigma^2\left(I_{n(x)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^x_{\sigma\sigma w} + \tfrac{1}{6}\mathbf{h}^x_{\sigma\sigma\sigma}\sigma^3, \end{aligned} \tag{4a}$$

$$\begin{aligned} \mathbf{y}_t \simeq\ & \mathbf{y} + \mathbf{h}^y_w \bar{\mathbf{w}}_t + \tfrac{1}{2}\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{ww}\bar{\mathbf{w}}_t + \tfrac{1}{2}\mathbf{h}^y_{\sigma\sigma}\sigma^2 \\ & + \tfrac{1}{6}\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{www}\bar{\mathbf{w}}_t + \tfrac{1}{2}\sigma^2\left(I_{n(y)} \otimes \bar{\mathbf{w}}_t^T\right)\mathbf{h}^y_{\sigma\sigma w} + \tfrac{1}{6}\mathbf{h}^y_{\sigma\sigma\sigma}\sigma^3. \end{aligned} \tag{4b}$$

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 (2) with respect to the vector $[\mathbf{w}_t^T, \sigma]^T$. The CoRRAM command `SolveModel` computes these matrices. They are used as arguments of 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.

3.4 Example

A simple example model shall illustrate these concepts. Latter 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

$$U_t := \mathbb{E}_t \sum_{s=0}^{\infty} \beta^s \frac{C_{t+s}^{1-\eta}(1-N_{t+s})^{\theta(1-\eta)} - 1}{1-\eta}, \quad \beta \in (0,1) \tag{5a}$$

subject to the resource restriction

$$Y_{t+s} = Z_{t+s}(A_{t+s}N_{t+s})^{1-\alpha}K_{t+s}^{\alpha}, \quad \alpha \in (0,1), \tag{5b}$$

$$K_{t+s+1} = Y_{t+s} + (1-\delta)K_{t+s} - C_{t+s}, \; \delta \in (0,1]. \tag{5c}$$

The first-order conditions of this problem are

$$\theta \frac{C_t}{1-N_t} = (1-\alpha)\frac{Y_t}{N_t}, \tag{6a}$$

$$C_t^{-\eta}(1-N_t)^{\theta(1-\eta)} = \beta \mathbb{E}_t C_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}\left(1-\delta+\alpha\frac{Y_{t+1}}{K_{t+1}}\right). \tag{6b}$$

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

$$\ln Z_{t+s+1} = \rho \ln Z_{t+s} + \epsilon_{t+s+1}, \qquad \eta_{t+1} \sim \mathcal{N}(0,\sigma), \tag{7a}$$

$$A_{t+s+1} = aA_{t+s}, \; a \geq 1. \tag{7b}$$

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. 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 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 (6), the production function (5b), and the law of capital accumulation (5c) is given by

$$g^1(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv y_t - Z_t N_t^{1-\alpha} k_t^{\alpha}, \tag{8a}$$

$$g^2(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv \theta \frac{c_t}{1-N_t} - (1-\alpha)\frac{y_t}{N_t}, \tag{8b}$$

$$g^3(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv a k_{t+1} - Y_t - (1-\delta)k_t + c_t, \tag{8c}$$

$$g^4(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv 1 - \beta a^{-\eta} \frac{c_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}}{c_t^{-\eta}(1-N_t)^{\theta(1-\eta)}} \left(1 - \delta + \alpha \frac{y_{t+1}}{k_{t+1}}\right), \tag{8d}$$

$$\mathbf{x}_t := k_t,\ \mathbf{y}_t := \begin{bmatrix} y_t, & c_t, & N_t \end{bmatrix}^T,\ \mathbf{z}_t := \ln Z_t. \tag{8e}$$

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 (8d), then, implies

$$\frac{y}{k} = \frac{a^{\eta} - \beta(1-\delta)}{\alpha\beta} \tag{9a}$$

so that equation (8c) can be solved for c/k:

$$\frac{c}{k} = \frac{Y}{K} - (a - 1 + \delta). \tag{9b}$$

Given y/k and c/k equation (8b) can be solved for N:

$$\frac{\theta}{1-\alpha}\frac{c/k}{y/k} = \frac{1-N}{N}. \tag{9c}$$

Finally, the solution for N can be used to solve equation (8a) for the level of the scaled capital stock k:

$$k = \left(\frac{y}{k}\right)^{\frac{1}{\alpha-1}} N \tag{9d}$$

so that y and c can be recovered from y/k and c/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 $Z \equiv 1$:

$$A_t = ae^{\nu_t} A_{t-1}, \quad a \geq 0, \ \nu_t \sim \mathcal{N}(0, \sigma_\nu). \tag{10a}$$

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 $k_t :=\equiv K_t/A_{t-1}$ remains an endogenous state variable. We employ the definitions

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

The model, thus, includes an additional variable, the growth factor a_t and its equations are

$$g^1(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv y_t - a_t^{1-\alpha} N_t^{1-\alpha} k_t^\alpha, \tag{11a}$$

$$g^2(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv \theta \frac{c_t}{1-N_t} - (1-\alpha)\frac{y_t}{N_t}, \tag{11b}$$

$$g^3(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv a_t - ae^{\nu_t}, \tag{11c}$$

$$g^4(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv a_t k_{t+1} - y_t - (1-\delta)k_t + c_t, \tag{11d}$$

$$g^5(\mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}, \mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t) \equiv 1 - \beta a_t^{-\eta} \frac{c_{t+1}^{-\eta}(1-N_{t+1})^{\theta(1-\eta)}}{c_t^{-\eta}(1-N_t)^{\theta(1-\eta)}} \left(1 - \delta + \alpha \frac{y_{t+1}}{k_{t+1}}\right). \tag{11e}$$

$$\mathbf{x}_t = k_t, \ \mathbf{y}_t := \begin{bmatrix} a_t, & y_t, & c_t, & N_t \end{bmatrix}, \ \mathbf{z}_t := \nu_t. \tag{11f}$$

The stationary versions of equations (11b)-(11e) imply the solutions for y/k, c/k, and N given in equations (9a)-(9c). The level of k follows from (8a) and is equal to

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

The next section will use this model to illustrate the usage of the toolbox. The parameters will be chosen as in Table 3.

Table 3: Calibration of the example model

Parameter	Description	Value
a	growth factor of output	1.004
α	capital share in output	0.3
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	2.07
ρ	autocorrelation of TFP shock	0.95
σ	standard deviation of TFP shock innovations	0.007
σ_ν	standard deviation of growth factor shock	0.018

4 PROGRAM INTERFACE

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. Table 4 lists the class properties, its default settings, and a short description of its purpose.

Several of the properties of this class will be set at the time when the user creates an instance of the class. Afterwards he can modify the new model either via the Matlab variables interface or via Matlab commands.

The class property `Var` is a structure with several fields. It stores information about the model's variables. Table 5 displays the field names, types, and default settings. The command `CreateModel` generates an instance of the `Var` structure with $n(x) + n(z) + n(y)$ elements. The fields `Name` and `Symbol` of the first $n(x)$ elements receive the strings `X1`, `X2`, etc., the next $n(z)$ elements the strings `Z1`, `Z2`, etc., and the last $n(y)$ elements the strings `Y1`, `Y2` etc. As you will learn below, you can supply more meaningful names and symbols which replace the defaults. *You must, however, stick to the succession of types.* CORRAM assumes that the first $n(x)$ variables are endogenous states followed by $n(z)$ exogenous states preceding the remaining $n(y)$ jump variables.

You will learn more about the features of the interface as we proceed and find out how to set up and solve a model.

Table 4: Alphabetic list of properties of the DSGE class

Property	Type	Default	Purpose
CheckBounds	b	f	if true, check whether a variable exceeds its bounds during simulations
ds	b	f	if true, the model is driven by a difference stationary growth process
Equations	s/fh		if numeric=false the name else the handle to the Matlab function defining the models equations
etol	d	10^{-9}	tolerance applied in checking the model's equations
Grid	b	t	if true, turn grid on in plots of impulse responses
hp	d	1600	the weight of the Hodrick-Prescott filter, if equal to zero, no filtering takes place
itol	d	10^{-10}	tolerance applied to check for complex coefficients
inobs	i	10	number of periods for impulse responses
LegendBox	b	t	if true, legends in plots of impulse responses are set within a box
loadpath	s	cd	path to the directory where Matlab looks for files to be loaded by the program
Messages	s		cell string with error messages, see Table 9
nu	i		the number of static equations
numeric	b	f	if true, numeric differentiation will be used
nx	i		the number of endogenous state variabes
ny	i		the number of not predetermined variables
nz	i		the number of shocks
Omega	d	0	$n(z) \times n(z)$ matrix of doubles, see Ω in (1c)
order	i	1	the order of the solution, must be equal to 1, 2, or 3
outfile	s	Model	the name of the file (without file extension) to which results are stored
Print	b	f	if true, print second moments from simulation
Plot	b	f	if true, plot impulse responses
Rho	d	0	$n(z) \times n(z)$ matrix of doubles, see R in (1b)
Skew	d	0	$n(z) \times n(z)^2$ matrix of doubles, see S in (1f)
syl	b	t	if true, solve the generalized Sylvester equation
Table	s	B	if equal to A: write results to ASCII file, if equal to X: write results to Excel table, if equal to B: write to both file types
Trendline	b	f	if true, plots of impulse responses display a trend line
Var	struct		structure that stores information used to print and plot simulation results

Notes: Abbreviations: b:=Boolean, either t:=true or f:=false, cd:=current working directory, d:=floating point, any real number, fh:=a Matlab file handle, i:=integer, including 0, s:= character string, struct:=Matlab structure. An empty field in the third column indicates that the property will be set when an instance of the class is created.

5 SET UP AND SOLVE A MODEL

The commands which create a model depend on whether numeric or analytic derivatives will be used. I begin with the numeric case.

5.1 Numeric derivatives

The script EM_numeric displays a worked out example of the solution and simulation of the example from Section 3.4.1.[1]

[1]The program code presented here puts together code segments from the script EM_Numeric.m.

Table 5: Fields of the `Var` structure

Field	Type	Default	Effect
`Bound(1)`	d	0	lower bound
`Bound(2)`	d	0	upper bound
`Corr`	b	f	compute correlations with this variable
`Name`	s		variable name for plots and tables
`Plotno`	i	0	plot to panel i∈{1, 2, ..., 6}
`Pos`	i		position of variable in $\mathbf{x}_t$, $\mathbf{y}_t$, or $\mathbf{z}_t$
`Print`	b	f	if true, print second moments for this variable to table
`Rel`	b	f	if true, compute standard deviations relative to the standard deviation of this variable
`Type`	s		type of variable, either x, y, or z
`Star`	d	0	stationary value of variable
`Symbol`	s		symbol used to represent variable in the model's equations
`Xi`	d	0	scaling factor

Notes: Abbreviations: b:=Boolean swich, either t:=true or f:=false, d:=floating point, any real number, i:=integer (including 0), s:=string. An empty field in the third column indicates that the property will be set when an instance of the class is created.

5.1.1 Example script

Lines 2 to 9 assign values to the model's parameters. Lines 12 to 18 compute the stationary solution from equations (9). This part of the script is model specific and outside of the CoRRAM environment.

EM_numeric

```
1  % Calibration
2  a=1;
3  alpha=0.3;
4  beta=0.99;
5  delta=0.025;
6  eta=2.0;
7  theta=2.07;
8  rho=0.95;
9  sigma=0.007;
10
11 % Stationary solution
12 YK=(a^eta-beta*(1-delta))/(alpha*beta);
13 CK=YK-(a-1+delta);
14 temp=(theta/(1-alpha))*(CK/YK);
15 Nstar=1/(1+temp);
16 Kstar=(YK^(1/(alpha-1)))*Nstar;
17 Ystar=YK*Kstar;
18 Cstar=CK*Kstar;
19
20 % Create instance of the DSGE class
21 nx=1;
22 nz=1;
23 ny=3;
24 nu=2;
25 v=[Kstar;0;Ystar;Cstar;Nstar];
26 Par=[a;alpha;beta;delta;eta;theta];
27 Names={'Capital','log of TFP','Output','Consumption','Hours'};
28 EM=DSGE(nx,ny,nz,nu,v,'Names',Names);
29
30 % Change default settings
31 EM.numeric=true;
```

```
32 EM.reduced=true;
33 EM.outfile='EM_numeric';
34 EM.order=2;
35
36 % Supply equations
37 EM.Equations=@(x,en)EM_Sys(x,en,Par);
38
39 % Transition of shocks
40 EM.Rho=rho;
41 EM.Omega=sigma;
42
43 % Solve Model
44 [Hmat,rc]=SolveModel(EM);
45
46 % Simulate Model
47 if rc>0;
48         EM.Messages{rc}
49 else
50         EM.Var(2).Plotno=1;
51         EM.Var(3).Plotno=2;EM.Var(3).Print=1; EM.Var(3).Rel=1; EM.Var(3).Corr=1;
52         EM.Var(4).Plotno=2;EM.Var(4).Print=1;
53         EM.Var(5).Plotno=2;EM.Var(5).Print=1;
54         EM.Print=true;
55         EM.log=false;
56         EM.Plot=true;
57         [irf,sx,rx]=SimulateModel(Hmat,EM);
58 end;
```

5.1.2 Create a model

The commands in lines 21 to 27 define the arguments of the function DSGE. This function has five required arguments, one optional argument, and a name-value pair. The required arguments are:

1. nx, the number of endogenous state variables $n(x)$,
2. ny, the number of jump variables $n(y)$,
3. nz, the number of shocks $n(z)$,
4. nu, the number of static equations,
5. v, the stationary solution of the model $\mathbf{v} \equiv [\mathbf{x}^T, \mathbf{0}_{1\times n(z)}, \mathbf{y}^T]^T$.

The optional argument is not used (we will encounter it in the script EM_analytic). The argument 'Name' passes a cell array with the names of the variables to the function. CoRRAM invokes the names, if the user requests plots and tables. The 'Name' argument is optional. If it is not supplied, CoRRAM labels the state variables as explained in Section 4.

5.1.3 Change default settings

After the model has been created the user must change the default settings of those properties of the model that constitute a particular solution algorithm. They are summarized in Section 5.3 and their effect will become clear in the example scripts below.

5.1.4 Supply equations

The command in line 37 defines an anonymous function `@(x,en)` with arguments `x` and `en` and assigns the handle to this function to the `Equations` property of the model. The anonymous function is defined from the function `EM_Sys(x,en,Par)`. This function must be written by the user (its name is arbitrary) and obey the following rules:

- It has two required inputs `v` and `eqn` and one optional argument `Par`.
- The function must return a vector `fx` with $n(x)+n(y)$ elements.
- The elements of the vector are equal to the left-hand sides of the model's equations (1a).
- The function must end with the command line

  ```
  if eqn>0; fx=fx(eqn); end;
  ```

The script `EM_Sys` presents the function that implements the model from Section 3.4.1:

EM_Sys

```
1  function fx = EM_Sys(v,eqn,Par)
2  % Example model from CoRRAM-M user guide. Equations for numeric differentiation
3
4  % assign values to parameters
5  a=Par(1);
6  alpha=Par(2);
7  beta=Par(3);
8  delta=Par(4);
9  eta=Par(5);
10 theta=Par(6);
11
12 % variables of the model, 1 refers to period t and 2 to period t+1 variables
13 K2=v(1);    K1=v(6);
14 z2=v(2);    z1=v(7);
15 Y2=v(3);    Y1=v(8);
16 C2=v(4);    C1=v(9);
17 N2=v(5);    N1=v(10);
18
19 % equations of the model
20 fx=ones(4,1);
21
22 fx(1)=Y1-exp(z1)*(N1^(1-alpha))*(K1^alpha);
23 fx(2)=theta*(C1/(1-N1))-(1-alpha)*(Y1/N1);
```

```
24  fx(3)=a*K2-Y1-(1-delta)*K1+C1;
25  fx(4)=1-beta*(a^(-eta))*(C1/C2)^(eta)*(((1-N2)/(1-N1))^(theta*(1-eta)))*...
26        (1-delta+alpha*(Y2/K2));
27
28  if eqn>0; fx=fx(eqn); end;
29
30  return;
31  end
```

The optional argument `Par` allows the user to pass the values of the model's parameters a, α, β, η, δ, and θ to the function. Alternatively, you could hard-code the values of these parameters:

```
5   a=1.0;
6   alpha=0.3;
7   beta=0.99;
8   eta=2.0;
9   delta=0.025;
10  theta=2.07;
```

The second argument of the function, `eqn` is used by the program to compute the Hessian matrix and must be placed before the `return` command.

The first argument `v` represents the vector

$$\begin{bmatrix} \mathbf{v}_{t+1} \\ \mathbf{v}_t \end{bmatrix} := \begin{bmatrix} \mathbf{x}_{t+1} \\ \mathbf{z}_{t+1} \\ \mathbf{y}_{t+1}, \\ \mathbf{x}_t \\ \mathbf{z}_t \\ \mathbf{y}_t \end{bmatrix}.$$

Lines 12 through 16 write the elements of this vector into new variables which makes it easier to code the equations (8a)-(8d) in lines 21 through 24. This step builds on the definition of the vectors in equation (8e). Without this step, the first equation in line 21 of `EM_Sys` would have to be written as

```
22  fx(1)=v(8)-exp(v(7))*(v(10)^(1-\alpha))*(v(6)^alpha);
```

In large models it will be hard to track the numbering of the variables in `v` and, thus, it is less error prone to assign the elements of this vector to new variables with names akin to those used in the model's analytic formulation.

5.1.5 Solve model

The command in line 44 of the script `EM_numeric` solves the model. The function `SolveModel` receives the model object in `EM` and returns two arguments. The variable `rc` stores the return code. The integer value 0 indicates success. Otherwise the

program prints a short error message to the Matlab command window (see line 48 in the script `EM_numeric`). Table 9 lists the error number and the corresponding message. The structure `Hmat` stores in its fields the matrices of the solution (4). For instance, the field `Hmat.Hx_w` holds the matrix $\mathbf{h}_w^x$, the field `Hmat.Hy_w` the matrix $\mathbf{h}_w^y$, the field `Hmat.Hx_ss` the vector $\mathbf{h}_{\sigma\sigma}^x$, and so fourth.

5.2 Analytic derivatives

The script `EM_analytic` provides a worked out example for the solution of the example from Section 3.4 with analytic derivatives.[2] I will explain the differences vis-à-vis the script `EM_numeric` in the next subsections.

5.2.1 Example script

EM_analytic

```
 % Calibration
Par=struct('Symbol','','Value',0);
Par(1).Symbol='a';     Par(1).Value=1.0;
Par(2).Symbol='alpha'; Par(2).Value=0.3;
Par(3).Symbol='beta';  Par(3).Value=0.99;
Par(4).Symbol='delta'; Par(4).Value=0.025;
Par(5).Symbol='eta';   Par(5).Value=2.0;
Par(6).Symbol='theta'; Par(6).Value=2.07;

rho=0.95;
sigma=0.007;

% Stationary solution
for i=1:1:6;
    assignin('base',Par(i).Symbol,Par(i).Value);
end;

YK=(a^eta-beta*(1-delta))/(alpha*beta);
CK=YK-(a-1+delta);
temp=(theta/(1-alpha))*(CK/YK);
Nstar=1/(1+temp);
Kstar=(YK^(1/(alpha-1)))*Nstar;
Ystar=YK*Kstar;
Cstar=CK*Kstar;

% Create instance of the DSGE class
nx=1;
nz=1;
ny=3;
nu=2;
v=[Kstar;0;Ystar;Cstar;Nstar];
Names={'Capital','log of TFP','Output','Consumption','Hours'};
Symbols={'K','z','Y','C','N'};
EM=DSGE(nx,ny,nz,nu,v,Symbols,'Names',Names);

% Modify default settings
EM.reduced=true;
EM.outfile='EM_Analytic';
EM.order=3;
```

[2] The program code presented here puts together code segments from the script EM_Analytic.m.

```
40
41  % Supply equations
42  EM.Equations='EM_Eqs';
43
44  % Transition of shocks
45  EM.Rho=rho;
46  EM.Omega=sigma;
47  EM.Skew=sigma^3;
48
49  % Solve Model
50  [Hmat,rc]=SolveModel_N(EM,Par);
51
52  % Simulate Model
53  if rc>0;
54          EM.Messages{rc}
55  else
56          EM.Var(2).Plotno=1;
57          EM.Var(3).Plotno=2;EM.Var(3).Print=1; EM.Var(3).Rel=1; EM.Var(3).Corr=1;
58          EM.Var(4).Plotno=2;EM.Var(4).Print=1;
59          EM.Var(5).Plotno=2;EM.Var(5).Print=1;
60          EM.Print=true;
61          EM.log=false;
62          EM.Plot=true;
63          [irf,sx,rx]=SimulateModel_N(Hmat,EM);
64  end;
```

5.2.2 Create model

The script `EM_analytic` assumes that you want to pass the model's parameter as symbols to the function which will define the model's equations. Alternatively, and similar to the case of numeric derivatives, you can hard code the parameters in the latter function. The disadvantage of this procedure is that you must change the function's code when you want to use different parameter values. If you supply the parameters in a structure you just have to change the code in lines 3 to 8. The field names `Symbol` and `Value` are mandatory, the name of the structure is arbitrary. The code in lines 14 to 16 assigns the values to the parameters so that you are able to use the same commands as in `EM_numeric` to compute the stationary solution in lines 18 to 24.

The command in line 33 defines a Matalb cell variable `Symbols`. CoRRAM assumes that you employ these symbols in the function which defines the model's equations (see Section 5.2.4). If you do not supply symbols for the model's variables the program assigns the symbol `X1` to the state variable, `Z1` to the shock, `Y1`, `Y2` and `Y3` to the three not predetermined variables. In this case, you are bound to formulate the model's equations with these symbolic names.

The model object is created in line 34 with a call to the function `DSGE`. Here you must pass the required arguments `nx` through `v`. `Symbols` and `Names` are optional.

5.2.3 Change default settings

Lines 37 to 39 change the default values of the model's properties. Since the order is set to 3, you must also supply the model with the matrix S from (1f). This is accomplished in line 47. Note, now you must not pass a function handle to the `Equations` property but a string with the name of the Matlab function where you have defined the model's equations.

5.2.4 Supply equations

The definition of the model's equations for use with the Matlab symbolic toolsbox is shown in the script `EM_Eqs`.

EM_Eqs

```
1  function [g,v] = EM_Eqs();
2  % Example model from CoRRAM-M user guide. Equations for symbolic differentiation
3
4  % Parameters
5  a=sym('a');
6  alpha=sym('alpha');
7  beta=sym('beta');
8  eta=sym('eta');
9  delta=sym('delta');
10 theta=sym('theta');
11
12 % variables of the model, 1 refers to period t and 2 to period t+1 variables
13 syms K1 z1 Y1 C1 N1 K2 z2 Y2 C2 N2;
14
15 v=[K2 z2 Y2 C2 N2 K1 z1 Y1 C1 N1];
16
17 % equations of the model
18 g=[ Y1-exp(z1)*(N1^(1-alpha))*(K1^alpha);
19     theta*(C1/(1-N1))-(1-alpha)*(Y1/N1);
20     a*K2-Y1-(1-delta)*K1+C1;
21     1-beta*(a^(-eta))*((C1/C2)^(eta))*(((1-N2)/(1-N1))^(theta*(1-eta)))*...
22     (1-delta+alpha*(Y2/K2));
23     ];
24 return;
25 end
```

The function `EM_Eqs` has no arguments and returns in the vector `v` the symbols which define the vector $[\mathbf{v}_{t+1}^T, \mathbf{v}_t]^T$ of the vector valued function $\mathbf{g}$ in (1a). In the variable `g` the function returns the analytic expressions for the four equations of the model. Lines 5 to 10 declare the symbols for the model's parameters. They must agree with the definitions in lines 3 to 7 of `EM_analytic`. Alternatively, you could define the parameters as numeric variables rather than as symbolic variables by assign numeric values to them:

```
5  a=1.0;
6  alpha=0.3;
7  beta=0.99;
8  eta=2.0;
9  delta=0.025;
10 theta=2.07;
```

The definition of the model's parameters as symbols has the advantage that you do not need to recompute the analytic expressions for the derivatives if you change the model's calibration. CoRRAM saves the symbolic code to a file and loads this information so that the model can be quickly recomputed with different parameters settings.

Line 13 uses the `syms` command to declare the symbolic names of the variables K_{t+1}, $\ln Z_{t+1}$, etc. Different from `EM_Sys` you *must* use 1 and 2 to distinguish between period t and $t+1$ variables.

5.2.5 Solve model

The model is solved via a call to the function `SolveModel` which receives both the model object in `EM` and the parameter structure in `Par`.

5.3 Solve options

CoRRAM provides several options for the solution of a model. The user can access these options by changing the respective properties of the model object. The next paragraphs document the properties in alphabetical order.

`etol` CoRRAM always checks whether the user has correctly solved for the stationary solution of the model. The solution algorithm evaluates the model's equations at the stationary solution. For each equation i it compares the left-hand side $g(i)$ to the tolerance `etol`. If $g(i)$ exceeds this value the program stops with the error message INVALID STATIONARY SOLUTION. In addition, it writes the equation number and the respective left-hand side returned by the call to the model's function to the file LOGFILE.TXT in Matlab's current directory. The default value of `etol` is 10^{-9}.

`itol` CoRRAM's solution algorithm involves complex valued matrices. The solution, however, are real valued matrices. The finite precision arithmetic of computers may entail small imaginary elements. If these are larger than the value of `itol` CoRRAM prints a warning message and the user should check for correct input and formulation of his or her model. The default value of `itol` is 10^{-10}.

`lm` This is a Boolean switch. When you solve your model for the first time, CoRRAM saves the code for the derivatives of (1b) to a file with extension `.mat`. It takes the base name of this file from the string in the model property `outfile`. The next time you solve your model, you can set `ls` to `true` and CoRRAM will look for this file an load the code. For models with many variables this saves considerable time.

`numeric` You have learned about the Boolean switch `numeric` in the previous

subsections. If `numeric=true` (or `numeric=1`), the solution algorithm employs numeric differentiation and the user must provide the adequate Matlab function with the model's equations. In addition, only first- and second-order accurate solutions can be computed. The default value of `numeric` is false.

`order` The `order` property is an integer value between 1 and 3 and determines the accuracy of the solution. The default value is 1 (i.e., linear solution).

`reduced` The Boolean switch `reduced` tells CoRRAM whether is should reduce the linearized model to a smaller model before the algorithm solves the dynamic part of the model. The default value is false. If you set `reduced=true` you must ensure two things:

1. The static equations of the model must precede the dynamic equations. The latter are equations where variables dated at period t and $t+1$ occur simultaneously.

2. The `nu` property of the model must be set to the number of static equations.

The case `reduced=false` produces many eigenvalues which are reported as either infinite or very large. They stem from the model's static equations. Also, be aware that `reduced=true` occasionally results in the error message CU MATRIX SINGULAR. You can respond to this message either by setting `reduced=false` or by changing the ordering of the variables in the vector $\mathbf{y}_t$.

`syl` Second- and third-order accurate solutions are obtained from solving generalized Sylvester equations (see Maußner (2017)). There is no Matlab intrinsic command to solve these equations. One way to solve this equation is to employ the vec operator. If the Boolean switch `syl` is set to `true`, CoRRAM uses an interface to the Lapack subroutine `tgsyl` to solve the Sylvester equation. This procedure requires smaller matrices that the vec-operator solution. Therefore, the default for `syl` is `true`

6 SIMULATE A MODEL

The function `SimulateModel` returns impulse responses and second moments of simulated time series. In order to compute these objects, the function must know how to interpret the model's variables and in what form it should provide the results to the user.

6.1 Interpretation of variables

The program can handle four different cases displayed in Table 6. They are invoked via the setting of two Boolean switches, `log` and `ds`.

Table 6: Variable transformations

	levels	logs
trend stationary	I	II
difference stationary	III	IV

6.1.1 Levels versus logs

The Boolean switch `log` tells the program whether is has to interpret all variables (except the shocks) either as (possibly scaled) levels of the variables or as natural logs of the original variables. If you employ the the log-specification, you must (i) pass the logs of **x** and **y** to the function `DSGE` and (ii) write adequate functions for the model's equations. This is easy if you employ numeric derivatives. Consider again our example from Section 3.4. Change line 25 of `EM_numeric` to

```
25    v=[log(Kstar);0;log(Ystar);log(Cstar);log(Nstar)];
```

and replace `EM.log=false` in line 55 by `EM.log=true`. Finally, change the function `EM_Sys` as shown below:

EM_Sys_Log

```
1   function fx = EM_Sys_Log(v,eq_n,Par)
2   % Example model from CoRRAM-M user guide. Equations for numeric differentiation in logs
3
4   %Parameters
5   a=Par(1);
6   alpha=Par(2);
7   beta=Par(3);
8   delta=Par(4);
9   eta=Par(5);
10  theta=Par(6);
11
12  % variables of the model, 1 refers to period t and 2 to period t+1 variables
13  v=exp(v);
14
15  K2=v(1);    K1=v(6);
16  z2=v(2);    z1=v(7);
17  Y2=v(3);    Y1=v(8);
18  C2=v(4);    C1=v(9);
19  N2=v(5);    N1=v(10);
20
21  % equations of the model
22  fx=ones(4,1);
23
```

```
24  fx(1)=Y1-z1*(N1^(1-alpha))*(K1^alpha);
25  fx(2)=theta*(C1/(1-N1))-(1-alpha)*(Y1/N1);
26  fx(3)=a*K2-Y1-(1-delta)*K1+C1;
27  fx(4)=1-beta*(a^(-eta))*(C1/C2)^(eta)*(1-delta+alpha*(Y2/K2));
28
29  if eq_n>0; fx=fx(eq_n); end;
30
31  return;
32  end
```

As you see, there are just two changes vis-à-vis the script `EM_Sys`. The code in line 13 transforms the logs back to levels, including the TFP-shock. As a consequence, `exp(z1)` in line 24 of `EM_Sys` is replaced by `z1` in line 24 of `EM_Sys_Log`.

If you want to solve the example with analytic derivatives, you would define the model's equations as shown in the next script:

EM_Eqs_Log

```
1   function [f,x] = EM_Eqs_Log();
2   % Example model from CoRRAM-M user guide. Equations for symbolic differentiation in logs
3
4   % Parameters
5   a=sym('a');
6   alpha=sym('alpha');
7   beta=sym('beta');
8   eta=sym('eta');
9   delta=sym('delta');
10  theta=sym('theta');
11
12  % variables of the model, 1 refers to period t and 2 to period t+1 variables
13  syms k1 z1 y1 c1 n1 k2 z2 y2 c2 n2;
14
15  x=[k2 z2 y2 c2 n2 k1 z1 y1 c1 n1];
16
17  % equations of the model
18  f=[ exp(y1)-exp(z1+n1*(1-alpha)+alpha*k1);
19      theta*(exp(c1)/(1-exp(n1)))-(1-alpha)*exp(y1-n1);
20      a*exp(k2)-exp(y1)-(1-delta)*exp(k1)+exp(c1);
21      1-beta*(a^(-eta))*exp(eta*(c1-c2))*(1-delta+alpha*exp(y2-k2));
22      ];
23  return;
24  end
```

6.1.2 Trend versus difference stationary

The Boolean switch `ds` determines whether growth is deterministic as in the model of Section 3.4.1 or stochastic as in the model of Section 3.4.2. The `DSGE` class assumes the former as default. To change to the latter set the class property `ds` to `true`.

Trend stationary growth. CoRRAM solves this model in the scaled variables. The function `SimulateModel` computes impulse responses and time series which must be interpreted as percentage deviations from the growth path of the model. This is

the default behavior implied by the class property `ds=false`. You may also want to pass the simulated time series through the Hodrick-Prescott (HP) filter by setting Verb"hp" to the desired value.[3]

Difference stationary growth. In difference stationary models even a one-time shock has lasting effects, since it triggers a permanent deviation from the trend path (see Maußner (2017), Section 9.2). To take account of this effect you must set `ds=true` and include the variable a_t as the first variable in the vector of not predetermined variables $\mathbf{y}_t$. For difference stationary models the function `SimulateModel` computes time series for the levels, employs the Hodrick-Prescott filter to the logged results, and computes second moments from the filtered series. In order to convert the stationary variables back to levels the program must know the transformation

$$X_t = A_{t-1}^{\xi} x_t$$

where x_t is the scaled variable. In the example (11), $\xi = 1$ for the variables k_t, y_t, and c_t and $\xi = 0$ for the variable N_t. If `ds=true`, CoRRAM gets the value of the parameter ξ from the information in the field `Xi` of the structure `Var`.

Here is a worked out example script for the model in (11). Compare this to the script `EM_analtyic` on page 18. Lines 8 and 9 define two additional parameters which are required to define equation (11c) (see the script `EM_Eqs_ds` below). Lines 20 to 26 compute the stationary solution. Importantly, lines 33 and 36 ensure that the variable a_t is placed at the top of the vector of not-predetermined variables $\mathbf{y}_t$ while lines 61, 64, and 65 change the default setting of the field `Xi` from 0 to 1.

EM_analytic_ds

```
1  % Calibration
2  Par=struct('Symbol','','Value',0);
3  Par(1).Symbol='alpha'; Par(1).Value=0.3;
4  Par(2).Symbol='beta';  Par(2).Value=0.99;
5  Par(3).Symbol='delta'; Par(3).Value=0.025;
6  Par(4).Symbol='eta';   Par(4).Value=2.0;
7  Par(5).Symbol='theta'; Par(5).Value=5.79;;
8  Par(6).Symbol='astar';
9
10 rho=0.0;
11 sigma=0.018;
12
13 % Stationary solution
14 for i=1:1:5;
15     assignin('base',Par(i).Symbol,Par(i).Value);
16 end;
17 astar=exp(a);
18 Par(6).Value=astar;
```

[3] See Hodrick and Prescott (1997). The cyclical component of this filter remains unchanged, if a linear time trend is added to a time series. Therefore, filtered data from the model are comparable to log-filtered empirical data, if the latter display exponential growth. See, e.g., Heer and Maußner (2009) for this property of the HP filter.

```
yk=(astar^eta-beta*(1-delta))/(alpha*beta);
ck=yk-(astar-1+delta);
temp=(theta/(1-alpha))*(ck/yk);
Nstar=1/(1+temp);
kstar=astar*(yk^(1/(alpha-1)))*Nstar;
ystar=yk*kstar;
cstar=ck*kstar;

% Create instance of the DSGE class
nx=1;
nz=1;
ny=4;
nu=3;
v=[kstar;0;astar;ystar;cstar;Nstar];

Names={'Capital','Growth Factor Shock','Growth Factor','Output','Consumption','Hours'};
Symbols={'k','z','a','y','c','N'};
EM=DSGE(nx,ny,nz,nu,v,Symbols,'Names',Names);

% Modify default settings
EM.reduced=true;
EM.outfile='EM_Analytic_ds';
EM.order=3;
EM.Plot=true;
EM.lm=false;
EM.ds=true;

% Supply equations
EM.Equations='EM_Eqs_ds';

% Transition of shocks
EM.Rho=rho;
EM.Omega=sigma;

% Solve Model
[Hmat,rc]=SolveModel_N(EM,Par);

% Simulate Model
if rc>0;
        EM.Messages{rc}
else
        EM.Var(1).Xi=1;
        EM.Var(2).Plotno=1;
        EM.Var(4).Plotno=2;EM.Var(4).Print=1; EM.Var(4).Rel=1; EM.Var(4).Corr=1;
        EM.Var(4).Xi=1;
        EM.Var(5).Plotno=2;EM.Var(5).Print=1; EM.Var(5).Xi=1;
        EM.Var(6).Plotno=2;EM.Var(6).Print=1;
        EM.Print=true;
        EM.log=false;
        [irf,sx,rx]=SimulateModel_N(Hmat,EM);
end;
```

The related function EM_Eqs_ds defines the model's equations and (hopefully) requires no additional comments. Just compare it to the script EM_Eqs on page 20.

EM_Eqs_ds

```
function [f,x] = EM_Eqs_ds();
% Example model from CoRRAM-M user guide. Equations for symbolic differentiation in levels

% Parameters
alpha=sym('alpha');
```

```
6   beta=sym('beta');
7   eta=sym('eta');
8   delta=sym('delta');
9   theta=sym('theta');
10  astar=sym('astar');
11
12  % variables of the model, 1 refers to period t and 2 to period t+1 variables
13  syms k1 z1 a1 y1 c1 N1 k2 z2 a2 y2 c2 N2;
14
15  x=[k2 z2 a2 y2 c2 N2 k1 z1 a1 y1 c1 N1];
16
17  % equations of the model
18  f=[ y1-(a1^(1-alpha))*(N1^(1-alpha))*(k1^alpha);
19      theta*(c1/(1-N1))-(1-alpha)*(y1/N1);
20      a1-astar*exp(z1);
21      a1*k2-y1-(1-delta)*k1+c1;
22      1-beta*(a1^(-eta))*((c1/c2)^(eta))*(1-delta+alpha*(y2/k2));
23      ];
24  return;
25  end
```

6.2 Simulation output

CoRRAM provides two tools to analyze a DSGE model: impulse responses and second moments of simulated time series. The command in line 57 of the script `EM_numeric` or in line 63 of the script `EM_analytic` passes the model's solution in `Hmat` and the model object `EM` to the function `SolveModel`. This function returns in `irf` the impulse responses, in `sx` the standard deviations of simulated time series, and in `rc` the matrix of correlations between the model's variables. If the `Print` and `Plot` switches are set to `true` the function also writes formatted output to a text file and/or an Excel file and plots selected impulse responses.

6.2.1 Impulse responses

Impulse responses show the dynamics of the model after a one-time shock to variable z_{it} in period $t = 2$, if the system was in its stationary equilibrium at time $t = 1$. The reaction of the model's variables to the shock provide a good means to explore the model's mechanics.

Computation. CoRRAM computes impulse responses only from the linear part of the solution (4). The iterations start at time $t = 1$ at the stationary solution

$[\mathbf{x}', \mathbf{0}_{1\times n(z)}, \mathbf{y}']'$ and proceed according to

$$\mathbf{z}_{t+1} = R\mathbf{z}_t + \begin{cases} \Omega\mathbf{e}_i & \text{for } t = 1, \\ \mathbf{0}_{n(z)\times 1} & \text{for } t = 2, 3, \ldots, T \end{cases} \tag{12a}$$

$$\mathbf{x}_{t+1} - \mathbf{x} = \mathbf{h}_w^x \begin{bmatrix} \mathbf{x}_t - \mathbf{x} \\ \mathbf{z}_t \end{bmatrix} \tag{12b}$$

$$\mathbf{y}_t - \mathbf{y} = \mathbf{h}_w^y \begin{bmatrix} \mathbf{x}_t - \mathbf{x} \\ \mathbf{z}_t \end{bmatrix}, \tag{12c}$$

for $t = 1, 2, \ldots, T$, where $\mathbf{e}_i$ is the vector with 1 in place i and zeros elsewhere.

The number of periods T is stored in the class property `inobs`. The variable `irf` is a three-dimensional array, each page $i = 1, \ldots, n(z)$ stores in its $n(x) + n(y) + 1$ columns the responses of the state variables $\mathbf{x}_{t+1}$ and the non-predetermined variables $\mathbf{y}_t$ to the shock z_{it}. The time path of the shock is returned in the right-most column $n(x) + n(y) + 1$ of each page. For instance, `irf(:,1,1)` stores the impulse response of the first variable in the vector $\mathbf{x}_{t+1}$ to the first-numbered shock of the model. The number of rows is equal to `inobs`.

Impulse responses are returned by `SimulateModel` in terms of percentage deviations from the stationary solution of the model or from the unshocked growth path, depending on the setting of the switch `ds`. For variables whose stationary value is equal to zero the program returns the absolute deviation from the stationary value.

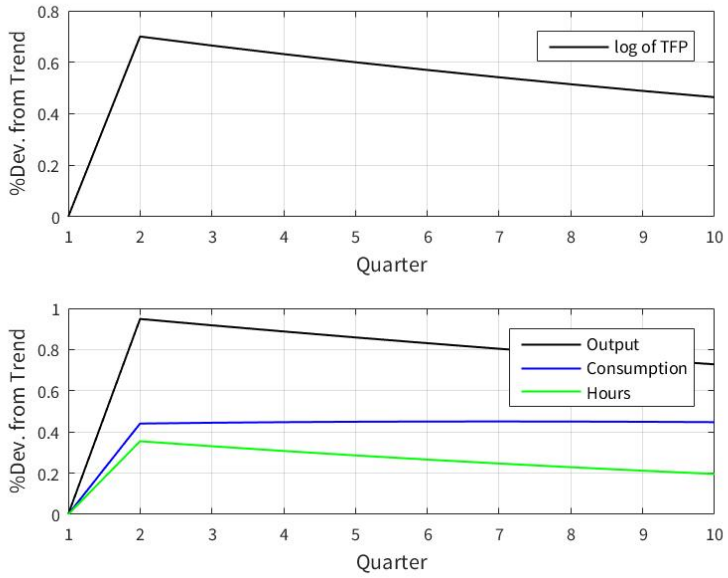
Output options. The default setting of the property `Plot` is `false` and the command `SimulateModel` returns the impulse responses in the array `irf`. It is up to user to visualize the results. If you want CoRRAM to do this job, set `Plot` to `true` and change the setting in the field `Plotno` of the `Var` structure. For instance, the commands in lines 49 to 52 of the script `EM_numeric` instruct CoRRAM to plot the time path of the TFP shock in panel 1 and the impulse responses of output, consumption, and hours to panel 2 as shown in Figure 2. CoRRAM supports up to eight panels each of which can display any number of impulse responses. For each shock, it provides one figure.

6.2.2 Second Moments

Second moments reflect the time series properties of the model (for more details see Maußner (2017), Section 7.6).

Computation. Their computation involves several steps. The first step is to trace out time paths of the model's variables. Towards this purpose CoRRAM draws normally distributed random numbers which represent the innovations $\boldsymbol{\epsilon}_{t+1}$ in (1d).

Figure 2: Impulse responses



You may also employ random numbers saved in a previous run of the model to the file `eps_array.mat`. In this case, you must set the class property `ls` to `true`. . Then, it iterates over

$$\mathbf{z}_{t+1} = R\mathbf{z}_t + \sigma\Omega\boldsymbol{\epsilon}_{t+1}, \tag{13a}$$

$$\mathbf{x}_{t+1} - \mathbf{x} = \hat{\mathbf{h}}^x(\bar{\mathbf{w}}_t, \sigma), \tag{13b}$$

$$\mathbf{y}_t - \mathbf{y} = \hat{\mathbf{h}}^y(\bar{\mathbf{w}}_t, \sigma) \tag{13c}$$

where the right-hand sides of (13b) and (13c) abbreviate the right-hand sides of the policy functions presented in equation (4). CoRRAM sets the perturbation parameter σ equal to one and starts the iterations at the stationary solution. In this way it traces out time paths of length equal to the `nobs` property of the `DSGE` class.

The user can instruct the program to check whether or not the variables exceed economically meaningful bounds during the simulation. This may happen if you use a second- or even third-order solution (see, e.g., Fernández-Villaverde et al. (2016)). To perform the check, set `Checkbounds=true` and assign lower and upper bound for all variables.

For the example from Section 3.4 it is reasonable to assume non-negative lower bounds and upper bounds of two times the stationary value. You can accomplish this via the following loop:

```
nvar=nx+ny+nz;
for i=1:nvar
    EM.Var(i).Bound(2)=2*EM.Var(i).Star(i);
end
```

CoRRAM saves the random numbers to the file `eps_array.mat` in the directory `loadpath`. If you want to reuse these numbers for repeated simulations set `ls` to `true`.

In the second step the program transforms the hypothetical time series to stationary data. If the weight of the Hodrick-Prescott filter in `hp` is equal to zero and the model is trend stationary (i.e,. `ds` is `false`) the program computes percentage deviations of all variables from either their stationary solution or from their trend growth path. If `hp` is positive, CoRRAM passes the logs of the time series (or the percentage deviations from trend if the levels are non-positive) to the Hodrick-Prescott filter. Difference stationary growth requires the Hodrick-Prescott filter and the program will stop if `hp=0` and `ds=true`. Otherwise it log filters the time series.

In the third step the program computes standard deviations, correlations between all variables, and first-order autocorrelations.

The three steps are repeated many times as determined by the value of the `snobs` property. The simulation routine then returns in the elements of the vector `sx` the average standard deviations of the variables ordered according to

$$x_1 \dots x_{n(x)}, z_1 \dots, z_{n(z)}, y_1 \dots y_{n(y)}.$$

The elements of the $2(n(x)+n(y)+n(z)) \times 2(n(x)+n(y)+n(z))$ matrix `rc` holds the contemporaneous correlations and the correlations between current and one-period lagged variables. The structure of this matrix is sketched in Table 7. The ordering within the vectors depends on the user's input in the call to `DSGE`.

Table 7: Structure of the correlation matrix `rc`

	$\mathbf{x}_t$	$\mathbf{z}_t$	$\mathbf{y}_t$	$\mathbf{x}_{t-1}$	$\mathbf{z}_{t-1}$	$\mathbf{y}_{t-1}$
$\mathbf{x}_t$						
$\mathbf{z}_t$						
$\mathbf{y}_t$						
$\mathbf{x}_{t-1}$						
$\mathbf{z}_{t-1}$						
$\mathbf{y}_{t-1}$						

Output options. The user must set the property `Print` to `true` if he wants selected moments from `sx` and `rc` printed to a a file. The setting in the property `Table`

determines where this information is written to. `Table='A'` writes to an ASCII file, `Table='X'` to an Excel file, and `Table='B'` to both.

To request output for a singe variable, set the field `Print` of this variable in the structure `Var` to `true`. If you want the contemporaneous correlations between this variable and other variables with `Print=true` to be displayed set the field `Corr` of this variable to `true`. Finally, if you want standard deviations of other variables relative to the standard deviation of variable i set the field `Rel` to `true`.

For instance, the command in lines 50 to 57 of the script `EM_numeric` produce the output displayed in Table 8.

Table 8: Second moments

```
18-Jan-2017 14:16:32

Second moments. Bounds were not checked.
Quadratic policy functions were used.

Output        1.19   1.00   1.00   0.70
Consumption   0.56   0.47   0.99   0.71
Hours         0.45   0.38   0.99   0.69

Column 1: Variable
Column 2: Standard Deviation
Column 3: Standard deviation relative to variable Output
Column 4: Cross correlation with variable Output
Column 5: First order autocorrelation
```

6.3 Simulation options

The next paragraphs summarize in alphabetical order those properties of the DSGE class that determine the simulation of model.

`CheckBounds` Set this Boolean switch to `true` if you want to check whether the model's variables remain within reasonable bounds during simulations. You must provide bounds through the `Bound` field of the `Var` structure. See page 29.

`ds` Set this Boolean switch to `true` if your model belongs to the class of difference stationary growth models. You must provide scaling factors through the `Xi` field of the `Var` structure. See page 24.

`Grid` Set this Boolean switch to `true` if you want to see grid lines on your plots of

impulse responses.

`hp` Set this to the desired value of the Hodrick-Prescott filter. Your simulated time series will not be filtered if `hp=0`. See page 25.

`inobs` This number determines the length of the impulse responses. See page 28.

`LegendBox` Set this Boolean switch to `true` and the legends in the plots of impulse will be put in box.

`ls` Set this Boolean switch to `true` if you want to employ random numbers saved in in previous run in the file `eps_array.mat`. See page 29.

`nobs` Set this to the desired length of the simulated time series. See page 29.

`nofs` Set this to the desired number of simulations. See page 29.

`outfile` The string in this field supplies the base name of the files to which CoRRAM writes results. The following extensions apply:

- mat: stores the analytic expressions of the Jacobian (`D`), the Hessian (`H`) and the matrix of third-order derivatives (`T`).
- txt: stores the ASCII table with second moments.
- xlsx: stores the Excel table with second moments.

`Print` Set this Boolean switch to `true` if you want CoRRAM to write information about selected second moments to a file. You determine the amount of information via the fields of the `Var` structure.

`Plot` Set this Boolean switch to `true` if you want plots of selected impulse responses. CoRRAM

`Table` Set this field to `A`, `X` or `B`, respectively, if you want second moments to be written to a simple text file, an Excel spreadsheet, or to both. See 30.

`Trendline` Set this Boolean switch to `true` if you want your plots of impulse responses to display a trend line.

7 Error messages

There are two kinds of error messages: (i) messages produced by Matlab and (ii) messages issued by CoRRAM.

(i) Matlab error messages usually will 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 will terminate with an error message printed to the Matlab command window. Both the `DSGE` function and the `SimulateModel` test for proper input. For instance, `SimulateModel` checks the switch `ds` and the value of `hp` and terminates if the former is true and the latter equal to zero because difference stationary time series must be filtered (see page 30). The error messages should be sufficiently instructive to eliminate the source of the problem.

The `SolveModel` function records 8 errors that may occur while it tries to solve the model. The are presented in Table 9.

Table 9: Error messages from `SolveModel`

Number	Message
1	Model does not exist
2	Invalid stationary solution
3	Illconditioned Jacobian
4	Not able to reduce model
5	Not able to solve reduced model
6	Schur failed
7	Instable model
8	Indetermined model

Error no. 1 is harmless. `SolveModel` searches for the file with the model's equations. If it is not able to locate this file, the function stops. Check the string in `Equations` and ensure that the directory is in the Matlab search paths.

In the next step, `SovleModel` checks the model's equations. If the left-hand side of at least one equation is not approximately equal to zero, it terminates with error no 2 (see page 21). The file Logfile.txt will then show a list with equation numbers and left-hand side values. There are (at least) three possible sources for this error. (1) errors in coding the model's equations, (2) errors in computing the stationary

solution, and (3) errors in passing the stationary solution to the function. The latter error occurs, if the positions of the variables in the vector **v** passed to `DSGE` do not coincide with the positions assigned to the variables in the function (1a).

Next, `SolveModel` evaluates the Jacobian of the model. It expects a real-valued matrix and stops if this is not the case.

In the next step, `SolveModel` computes the linear part of the model's solution. At this stage errors 4 to 8 may occur. Error 4 indicates that the function is not able to reduce the linearized model (see page 22). The matrix labeled C_u in equation (29a) of Maußner (2017) is singular. A reordering of the jump variables may resolve the problem. Error 5 occurs, if more there are more than $n(u)$ static equations in the model. Switching from `reduced=true` to `reduced=false` should solve this problem.

Error no 6 indicates an ill-conditioned model that has not been detected by the previous checks.

In the final step, `SolveModel` checks the stability and uniqueness of the first-order accurate solution. This requires $n(x) + n(z)$ eigenvalues of the linearized model to be less than one in absolute value and $n(y)$ to be larger than one. Error no 7 (8) indicates that the first (second) requirement is violated. The file Logfile.txt displays a list of all eigenvalues.

REFERENCES

D'Errico, J. R. (2007). Derivest. mimeo.

Fernández-Villaverde, J., R. Ramírez, and F. Schorfheide (2016). Solution and estimation methods for dsge models. Working Paper W21862, National Bureau of Economic Research (NBER).

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

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

Maußner, A. (2017). Perturbation solution of dsge models. mimeo, Universität Augsburg.