CoRRAM

Version 4 User Guide

Prof. Dr. Alfred Maußner

February 11, 2020

Contents

1 Introduction **4**
- 1.1 Purpose . . . 4
- 1.2 Licence . . . 4
- 1.3 Prerequisites . . . 4
- 1.4 Notation . . . 5
- 1.5 Differences to Previous Versions . . . 5

2 Installation **6**
- 2.1 Contents of the Archive . . . 6
- 2.2 Application Installer . . . 6
- 2.3 VBS Script . . . 6
- 2.4 Manually . . . 7

3 The DSGE Structure **8**

4 Structure of the Models **11**
- 4.1 Variables . . . 12
- 4.2 Equations . . . 13
- 4.3 Example . . . 13

5 Solutions **14**
- 5.1 Steady State . . . 14
- 5.2 Linearization . . . 16
- 5.3 Structure of Solutions . . . 19
- 5.4 Difference Stationary Growth . . . 19
- 5.5 Error Messages . . . 20

6 Providing a Model to CoRRAM **20**
- 6.1 Provide the Parameters and Compute the Stationary Solution . . . 20
- 6.2 Initialize the Model Envirionment . . . 21
- 6.3 Provide the Model's Equations . . . 22
- 6.4 Solve the Model . . . 24
- 6.5 Evaluate the Model . . . 25
- 6.6 Graphs and Tables . . . 26

7 Inside the Black Box **29**
- 7.1 Structure of the Toolbox . . . 30
- 7.2 Linear Solutions . . . 32
- 7.3 Quadratic Solution . . . 32
- 7.4 Simulation . . . 33

7.5 Stand Alone Procedures 35

References **36**

1 INTRODUCTION

1.1 Purpose

This document explains the use of the most recent version of my Gauss toolbox for the solution and simulation of a certain class of dynamic stochastic general equilibrium (DSGE) models. This class comprises models with a recursive structure and with representative agents. Recursive models have the property that the decisions of economic agents depend on a set of variables that fully describe the current state of their environment. These agents do not differ from each other with respect to assets, expectations, productivity or the like.[1] Or they can, by some trick, be treated as a homogenous group. The restriction to this class of models gave the program its name. The acronym CoRRAM stands for Computation of Recursive Representative Agent Models).

The aim of CoRRAM is to allow less experienced users of Gauss to solve and simulate their model using either linear or quadratic approximations of the model's equilibrium conditions.

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 Prerequisites

This manual accompanies my lecture notes Maußner (2019) on computational macroeconomics. Therefore, I assume that you are familiar with DSGE models and, in particular, with perturbation methods that obtain approximate solutions of these models. Other references to this subject are, among others, Schmitt-Grohé and Uribe (2004), Heer and Maußner (2009), and Fernández-Villaverde et al. (2016).

I also assume that you know how to start the Gauss software under the Windows operating system and that you have a basic knowledge of the Gauss syntax.

The toolbox will run under Gauss version 16 and above. ⓘ

[1] They can, however, differ with respect to age (for an example, see, e.g., Heer and Maußner (2009), Section 10.2.1.

1.4 Notation

In this document I use the `typewriter` font to print Gauss commands, program variables, and file names. Remember, the Gauss programming language is not case sensitive. Hence, if I use a mix of capital and small letters it is just for making the program code more readable. Vectors $\mathbf{x} \in \mathbb{R}^n$ are typeset in bold, where the symbol $\mathbb{R}^n$ stands for the Euclidean n-space. Thus, $\mathbf{x} := [x_1, x_2, \ldots, x_{n(x)}]^T$ is an ordered collection of $n(x)$ real numbers with indices i running from $i = 1$ to $i = n(x)$. The superscript T denotes transposition, i.e., the row vector $[x_1, x_2, \ldots, x_{n(x)}]$ is transposed into the column vector

$$\mathbf{x} := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n(x)} \end{bmatrix} = [x_1, x_2, \ldots, x_{n(x)}]^T.$$

1.5 Differences to Previous Versions

Version 4 of CoRRAM differs from the former versions in one important aspect: it dispenses with the use of global control variables and stores information about the model in a structure. In the parlance of object oriented programming, a model is an object with various properties, among them are the number, types, and names of variables, flags that control the solution and simulation of the model, and elements that store the solution and the results of simulations. This approach, although not fully object oriented, has at least two advantages vis-à-vis the use of global variables. First, there is no danger that the user unwittingly overwrites those variables. Second, since instances of structures can be saved in a file, the user can plot impulse responses or print tables with second moments at any time he desires to do so without the need to resolve the model. In addition, all information about the solution and simulation is kept.

Other changes were driven by my desire to shorten the set up of a model. It requires much less program code to specify a model, if the user just wants to solve and simulate his model and cares himself about the creation of graphs and tables.

Further changes include

- computation of second moments from the first-order solution of the model without the need to simulate time series,
- automatic reordering of the not predetermined variables so that the reduction of the model does not require a previous ordering by the user,

- the ability to handle models with shocks that have the mean preserving spread property.[2]

2 INSTALLATION

2.1 Contents of the Archive

The files of CoRRAM are in `app_CoRRAM_V4.xx.zip`, where `xx` is a two digit number that indicates the most current release. This archive includes several subdirectories:

- doc: the place where you find the user guide.
- examples: the place where you find examples.
- lib: the place with the CoRRAM library.
- src: the place with the CoRRAM source files (see Table 1).

In the root directory you find the licence file and two versions of a dynamic link library.

2.2 Application Installer

Gauss version 18 (I have not checked previous versions) is able to install CoRRAM automatically. Open the Tools menu and choose "Install Application". Gauss will create a subdirectory `pkgs\Corram6` and puts the files in the respective subdirectories of the distribution. In addition, it copies the correct version of the dynamic link library `Tools.dll` to its `dll` subdirectory and creates the Gauss library file, which is placed to Gauss' subdirectory `lib`. I cannot guarantee that this works properly. I know about instances where Gauss did not include the full list of procedures in the the library file `CoRRAM6.lcg`.

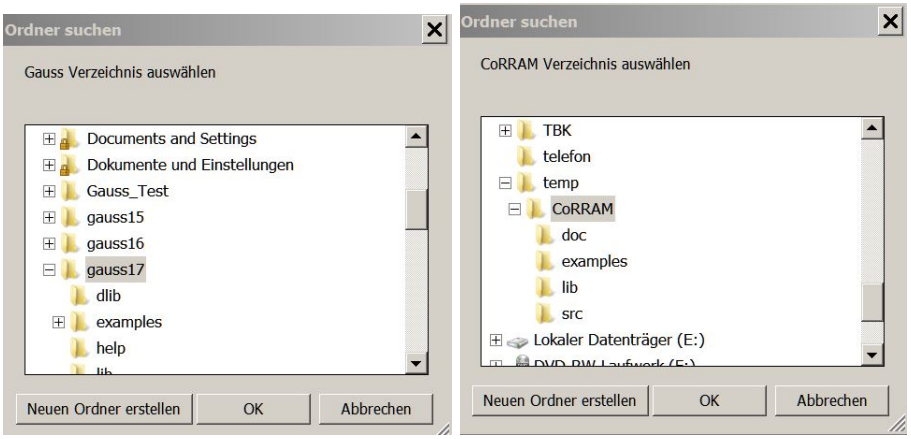
2.3 VBS Script

In the root directory of `app_CoRRAM_V4.xx.zip` you will find `CorramInstall.vbs`. This is a simple Windows script. Before you can run it, unpack all files to a place on your hard disc and ensure that the subdirectories are also unpacked. Then run `CorramInstall.vbs`. The script opens two windows from where you can choose the paths to the Gauss installation directory and the path to which you have unpacked `app_CoRRAM_V4.xx.zip` (see Figure 1). The script modifies the

[2] See Heiberger and Maußner (2020)

library file CoRRAM6.lcg and copies this version to the [GaussDir]\lib directory, where [GaussDir] refers to the installation directory of Gauss. In addition, it copies the required version of the dynamic link library (either Tools-win32.dll or Tools-win64.dll) to [GaussDir]\dlib.

Figure 1: Select Directories



2.4 Manually

You can also install CoRRAM manually. Unpack app_CoRRAM_V4.xx to a place on your hard disc and ensure that the directory structure of this archive is kept intact. Then, copy the files listed in Table 1 from the subdirectory src to the respective destination directory. If you run the 64 bit (32 bit) version of Gauss copy Tools_64.dll (Tools_32.dll) to [GaussDir]\dlib and rename this file to Tools.dll.

The files with the .sdf, and the .src extensions contain source code that you should not change. Gauss will know from the information in CoRRAM6.lcg where to find the required piece of code. I explain the role of these files in Section 7.1.

This dynamic link libraries Tools-32.dll and Tools-64.dll contain routines written in Fortran. If you do not want to use these routines, you can set the flag GaussOnly=1.

The files CoRRAM_Test_1.g, CoRRAM_Test_2.g, and CoRRAM_Test_3.g illustrate the use of the program with the benchmark business cycle model as it is described in Heer and Maußner (2009), Section 2.6.1 and in Maußner (2019), Section 8.1. You should run these files to check your installation.

You may also want to install the source files of CoRRAM to a different place than to [GaussDir]\src. To do this, follow these steps:

i. Unpack the app_CoRRAM_V4.xx.zip as explained above.

Table 1: File Installation

File	Destination	Remark
CoRRAM_1.src	[GaussDir]\src\CoRRAM_1.src	
CoRRAM_2.src	[GaussDir]\src\CoRRAM_2.src	
CoRRAM_3.src	[GaussDir]\src\CoRRAM_3.src	
CoRRAM_4.src	[GaussDir]\src\CoRRAM_4.src	
CoRRAM_5.src	[GaussDir]\src\CoRRAM_5.src	
CoRRAM_6.src	[GaussDir]\src\CoRRAM_6.src	
CoRRAM6.sdf	[GaussDir]\src\CoRRAM.sdf	
CoRRAM.lcg	[GaussDir]\lib\CoRRAM.lcg	
Tools-32.dll	[GaussDir]\dlib\Tools.dll	for Gauss 32 bit version
Tools-64.dll	[GaussDir]\dlib\Tools.dll	for Gauss 64 bit version

Notes: [GaussDir] refers to the directory where you have installed Gauss.

ii. Open the file MyLibTool.g in the examples subdirectory in the Gauss source editor.

iii. Edit line 20 in this file: replace the path with the path to the directory to which you have unpacked the file in src.

iv. Change the Gauss working directory to the directory with the source files.

v. Run MyLibTool.g from Gauss.

vi. In the directory with the source files you will now find the file CoRRAM6.lcg. Copy this file to the [GaussDir]\lib directory.

3 THE DSGE STRUCTURE

As noted in the introduction, CoRRAM stores a model in a structure. An instance of this structure, MyModel, say, is created by the command

```
struct DSGE MyModel;
```

Thereafter, members of the structure can be referenced using the dot syntax. For instance, the command

```
MyModel.nx=1;
```

assigns the number 1 to the model property `nx`, which stores the number of endogenous state variables.

The `DSGE` structure has many direct members and two substructures. The substructure `Flags` collects binary (or boolean) variables that are either *true* or *false*. In Gauss, the value true is coded as `1` and the false as `0`. The substructure `Var` stores information about a single variable. For instance, the command

```
MyModel.Var[1].name="Output";
```

assigns the string `Output` to the property `name` of the first variable of `MyModel`. After you have set up, solved, and simulated a model, you can store the entire structure to a file for later reuse with the Gauss command `savestruct`.

In the following sections I will introduce the members of the `DSGE` structure as needed. To highlight the difference between ordinary program variables and the members of the `DSGE` structure, I typeset the latter in red color. If I want to reference several members of the `DSGE` structure or its substructures, I print a star, `*`, in the place where either a number or a name is required. Table 2 lists the members of the `DSGE` structure, grouped according to their intended purpose. Table 3 lists the members of the `Var` substructure, and Table 4 displays the members of `Flags`.

Table 2: Members of the `DSGE` structure

Name	Purpose
	Properties of the Model
`nx`	Number of endogenous state variables. See Section 4.1.
`ny`	Number of jump variables. See Section 4.1.
`nz`	Number of stochastic processes. See Section 4.1.
`nu`	Number of static equations. See Section 4.2 and page 23.
`Rho`	Matrix. See Equation (1a).
`Omega`	Matrix, See Equation (1b).
`Var`	Substructure, see Table 3.
	Determinants of the Solution
`alpha`	Nonnegative scalar. If `confidenc=1`, 1-`alpha` percent confidence bounds will be computed. See pages 26 and 34. Default: 0.05.
`HPL`	Nonnegative scalar. The weight of the Hodrick-Prescott filter. If zero, no filter will be applied. See pages 26 and 34. Default: 1600
`maxlag`	Positive integer, the number of leads and lags used to compute auto- and cross-correlations. See pages 27, and 34. Default: 1
`nfreq`	Positive integer, the number of nodes used to compute analytic second moments in the frequency domain. See page 34. Default: 256
`nirf`	Positive integer, the number of periods for impulse responses. See page 25. Default: 15
`nobs`	Positive integer, the length of time series used in simulations of the model. See page 25. Default: 120
`nofs`	Positive integer, the number of simulations used to compute second moments. See page 25. Default: 500
`order`	Scalar, order of approximation. Possible values are $\{1, 2\}$. See pages 19 and 34. Default: 1
`QA_type`	Scalar, determines the algorithm of the second-order solution. Possible values are $\{1, 2, 3\}$. See Figure 4 and page 32. Default: 1.
`seed`	Scalar, the seed used for the random number generator. See pages 26 and 34. Default: 2018

continued on the next page

Name	Purpose
	Elements of the Solution
`Amat`	Matrix. See equations (8b) and (11c).
`Bmat`	Matrix. See equations (8c) and (11b).
`Cu`	Matrix. See equations (9) and (13).
`Cxv`	Matrix. See equations (9) and (13).
`Cz`	Matrix. See equations (9) and (13).
`Dxv`	Matrix. See equations (9) and (13).
`Fxv`	Matrix. See equations (9) and (13).
`Du`	Matrix. See equations (9) and (13).
`Fu`	Matrix. See equations (9) and (13).
`Dz`	Matrix. See equations (9) and (13).
`Fz`	Matrix. See equations (9) and (13).
`Gmat`	Matrix, the Jacobian matrix of the system of equations (2).
`Hxw`	Matrix. See equation (16a)
`Hyw`	Matrix. See equation (16b)
`Hxww`	Matrix. See equation (16a)
`Hyww`	Matrix. See equation (16b)
`Hxss`	Vector. See equation (16a).
`Hyss`	Vector. See equation (16b).
`wstar`	Vector. See pages 22 and 22.
`xivec`	Vector. Instead of passing the parameter ξ (see page 19) via the elements of the `Var` substructure (see Table 3), the user can provide this information through the elements of this vector. The ordering must be the same as in `wstar`.
	Elements of the Simulation Output
`Corr_m`	Three dimensional array with second moments. See pages 25 and 34.
`Corr_l`	Three dimensional array with lower bounds of second moments, determined by `alpha`. See pages 25 and 34.
`Corr_u`	Three dimensional array with upper bounds of second moments, determined by `alpha`. See pages 25 and 34.
`IRF`	Three dimensional array with impulse responses. See page 25.
`file`	String, the name of the file to which error messages and tables will be printed. The base name of `file` (i.e., the string without a file extension as for instance `.txt`), appended with the file extension `.xlsx` will be used as the name of the Excel spreadsheet to which tables will be written, if `Excel` is true.
`valid`	Integer, the number of valid solutions. See page 34.
	Tolerances
`e_tol`	Scalar, see page . Default: 10^{-10}. See page 24.
`i_tol`	Scalar, see page . Default: 10^{-10}. See page 20.

Table 3: Members of the `Var` substructure

Name	Function
`name`	String, name of the variable.
`type`	String, either `x`, `y`, or `z`, type of the variable.
`pos`	Scalar, the position of the variable in the respective vector.
`star`	Scalar, the stationary value of the variable.
`bounds`	2×1 vector, the upper and the lower bound of the variable. See page 34.
`xi`	The scaling factor of the variable. See Section 5.4
`print`	Boolean, if true, print second moments of the variable. See page 27.
`plotno`	Scalar, if >0, the impulse response of the variable will be plotted in panel number `plotno`. See page 27
`rel`	Boolean, if true, standard deviations of other variables relative to this variable will be printed. See page 27.
`cor`	Boolean, if true, correlations with other variables and this variable will be printed. See page 27.

Table 4: Members of the `Flags` Substructure

Name	Effect if true	Default
`bal`	The matrix pencil (A, B) from equation (8) is balanced before it is factored. See page 20.	0
`CheckEquatons`	Evaluate the model's equations at the stationary solution and print those equations that return a number larger in absolute value than `e_tol`. See page 24.	1.
`CheckBounds`	Check whether a variable exceeds its bounds during simulations. See page 26.	0
`color`	Colored impulse responses are plotted.	1
`confidence`	The program computes `1-MyModel.alpha` percent intervals for the computed second moments. See page 26.	0
`DS`	The program assumes a difference stationary growth process. See Section 5.4.	0.
`equations`	The program expects the equations of your model in procedure, see pages 16, 17, and 19.	1
`Excel`	Tables with second moments will also be written to an Excel spreadsheet. See page 30.	0
`GaussOnly`	No foreign language code will be loaded. Some program options may be unavailable. See Figure 4.	0
`logs`	CoRRAM assumes the model is defined in the logs of the variables. See page 18.	0
`overwrite`	If the log file of the model already exists, it will be overwritten.	0
`reduce`	The AB-model is solved. See Figure 4 and pages 16, 17, and 23.	0
`Simulate`	CoRRAM computes second moments from simulations of the model. See page 26.	1
`separate`	solve for the deterministic and the stochastic part of the linear policy functions in separate steps.	0
`Sylvester`	The procedures `SolveQA2` and `SolveQA3` employ the routine `sylvester` in `Tools.dll`. See Figure 4.	0

4 Structure of the Models

In this section, I describe the canonical form of the models that CoRRAM is able to solve.

4.1 Variables

Basically, the class of recursive representative agent models has three different kinds of variables:

1. Purely exogenous variables, the model's *shocks*,
2. *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),
3. endogenous variables, whose values are determined within period t. I will refer to these variables interchangeably as *control variable*, *jump variables* or *not predetermined variables*.

This document denotes the shocks of period t by z_{it} and collects them in the column vector $\mathbf{z}_t = \left[z_{1t}, z_{2t}, \ldots, z_{n(z)t}\right]^T$. Analogously, $\mathbf{x}_t = \left[x_{1t}, x_{2t}, \ldots, x_{n(x)t}\right]^T$ and $\mathbf{y}_t = \left[y_{1t}, y_{2t}, \ldots, y_{n(y)t}\right]^T$ collect the $n(x)$ state variables and the $n(y)$ control variables, respectively.

With respect to the model's shocks, CoRRAM assumes that either their absolute or their relative deviations from their deterministic means follow the first-order autoregressive vector process

$$\mathbf{z}_t = R\mathbf{z}_{t-1} + \boldsymbol{\epsilon}_t, \tag{1a}$$
$$\boldsymbol{\epsilon}_t = \sigma\Omega\boldsymbol{\nu}_t, \quad \sigma \geq 0, \tag{1b}$$
$$\boldsymbol{\nu}_t \sim N(\mathbf{0}_{n(z)\times 1}, I_{n(z)}). \tag{1c}$$

In order for this process to be stationary, the eigenvalues of the matrix R must all be located within the unit circle. Note that the covariance matrix Σ of the vector $\boldsymbol{\epsilon}_t$ is

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

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

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

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 $CC^T = I_{n(z)}$.

You can use the procedure `MakeOmega` (documented in the file `CoRRAM_3.src`) to obtain Ω from Σ.

4.2 Equations

The dynamics of the models must be governed by the set of equations

$$\mathbb{E}_t g^i(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}) = 0, \quad i = 1, 2, \ldots, n(x) + n(y), \tag{2}$$

where $\mathbb{E}_t$ denotes mathematical expectations as of period t. Usually, a subset of $n(u)$ of the equations will only involve date t dated variables. In this case, the system (2) can be partitioned into

$$0 = g^i(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t), \qquad i = 1, 2, \ldots, n(u), \tag{3a}$$

$$0 = \mathbb{E}_t g^i(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1}), \quad i = n(u)+1, \ldots, n(x)+n(y). \tag{3b}$$

The first $n(u)$ equations are static ones, whereas the remaining $n(x)+n(y)-n(u)$ equations are dynamic one, because the relate variables at time t to those at time $t+1$.

4.3 Example

The stationary variables of the benchmark business cycle model from Section 1.5.1 of Heer and Maußner (2009) and Maußner (2019), Section 8.1 are the capital stock per efficiency unit of labor, k_t, output, consumption, and investment, all per efficiency unit of labor, y_t, c_t, and i_t, respectively, working hours N_t, the rental rate of capital r_t, the real wage per efficiency unit of labor w_t, the level of total factor productivity Z_t, and the scaled Lagrange multiplier of the household's budget constraint λ_t. Thus:

- $\mathbf{x}_t = k_t$,
- $\mathbf{y}_t = [y_t, c_t, i_t, N_t, w_t, r_t, \lambda_t]^T$,
- $\mathbf{z}_t = \ln Z_t$.

The model's equations are

$$0 = c_t^{-\eta}(1-N_t)^{\theta(1-\eta)} - \lambda_t, \tag{4a}$$

$$0 = \theta c_t^{1-\eta}(1-N_t)^{\theta(1-\eta)-1} - \lambda_t w_t, \tag{4b}$$

$$0 = r_t - \alpha\frac{y_t}{k_t}, \tag{4c}$$

$$0 = w_t - (1-\alpha)\frac{y_t}{N_t}, \tag{4d}$$

$$0 = y_t - Z_t N_t^{1-\alpha} k_t^{\alpha}, \tag{4e}$$

$$0 = y_t - i_t - c_t, \tag{4f}$$

$$0 = ak_{t+1} - (1-\delta)k_t - i_t, \tag{4g}$$

$$0 = \lambda_t - \beta a^{-\eta} \mathbb{E}_t \lambda_{t+1} (1 - \delta + r_{t+1}), \tag{4h}$$

where a, α, β, δ, η, and θ are parameters of the model. Equations (4a) through (4f) are static equations. Finally, with $z_t := \ln Z_t$ the shock process is given by

$$z_{t+1} = \varrho z_t + \sigma \epsilon_{t+1}, \quad \epsilon_{t+1} \text{ iid } \sim N(0, \sigma_\epsilon)$$

so that $\mathbf{z}_t \equiv z_t$, $R \equiv \varrho$, $\Omega \equiv \sigma_\epsilon$, and $\sigma \equiv 1$. Where σ_ϵ is the standard deviation of the innovations in the AR(1)-process of the technology shock $\ln Z_t$.

5 Solutions

Note that for $\sigma = 0$ the model in (1) and (2) collapses to a deterministic model. Iterating on equation (1a) for $\boldsymbol{\epsilon}_t = \mathbf{0}_{n(z)\times 1}$ drives the vector $\mathbf{z}_t$ to zero. Given that the deterministic model is stable, it will approach a stationary solution (or a balanced growth path) in which all variables remain constant. CoRRAM computes linear and quadratic approximations of the stochastic model's solution at this point.

Both the linear and the quadratic approximations build on the linearized equations (2). In order to understand the way you can input your model and to be able to respond to error messages reported by the program you need a basic understanding of the linearized model.

5.1 Steady State

The stationary solution is obtained from (2) in the following steps:

1. Replace $\mathbf{z}_t$ by $\mathbf{z} = \mathbf{0}$ and ignore the expectations operator $\mathbb{E}_t$.
2. Assume stationarity, i.e., drop the time index from all variables.
3. Solve

   $$g^i(\mathbf{x}, \mathbf{y}, \mathbf{0}, \mathbf{x}, \mathbf{y}, \mathbf{0}) = 0, \quad i = 1, 2, \ldots, n(x) + n(y),$$

   for $\mathbf{x}$ and $\mathbf{y}$.

There are two ways to compute this solution: a) Use paper and pencil to reduce the system so that it can be solved equation by equation, and b) employ a non-linear equations solver.

Step by Step. Consider example (4). In the steady state with $Z \equiv 1$ it reduces to

$$0 = c^{-\eta}(1-N)^{\theta(1-\eta)} - \lambda, \tag{5a}$$
$$0 = \theta c^{1-\eta}(1-N)^{\theta(1-\eta)-1} - \lambda w, \tag{5b}$$
$$0 = r - \alpha\frac{y}{k}, \tag{5c}$$
$$0 = w - (1-\alpha)\frac{y}{N}, \tag{5d}$$
$$0 = y - N^{1-\alpha}k^{\alpha}, \tag{5e}$$
$$0 = y - i - c, \tag{5f}$$
$$0 = ak - (1-\delta)k - i, \tag{5g}$$
$$0 = \lambda - \beta a^{-\eta}\lambda(1-\delta+r). \tag{5h}$$

Substituting equation (5c) into (5h) we can solve for the output-capital ratio

$$\frac{y}{k} = \frac{a^{\eta} - \beta(1-\delta)}{\alpha\beta}. \tag{6a}$$

The resource constraint (5f) and equation (5g) imply

$$\frac{c}{k} = \frac{y}{k} - (a - 1 + \delta). \tag{6b}$$

Substituting for λ in (5b) from (5a) and for w from (5d) yields

$$\frac{N}{1-N} = \frac{1-\alpha}{\theta}\frac{y}{c} \tag{6c}$$

which can be solved for N, since $y/c = (y/k)/(c/k)$. The production function (5e) implies

$$\frac{k}{N} = (y/k)^{\frac{1}{\alpha-1}}. \tag{6d}$$

Thus, given the solution for N we can back out the stationary values of k, y, c. Given these, equations (5a), (5c), and (5d) can be used to determine λ, r, and w, respectively.

Simultaneously. Instead of solving (5) stepwise you may want to input the system in Gauss and use either the Gauss command `EqSolve` or any other program that solves a system of non-linear equations. The Gauss non-linear equations solver needs an initial vector $\mathbf{x}_0 = [k_0, y_0, c_0, i_0, N_0, w_0, r_0, \lambda_0]^T$ to start the algorithm. If you do not have prior knowledge of the solution, you can use random numbers distributed

on a positive subset of $\mathbb{R}^8$. However, since solving non-linear equations is sometimes a tricky business, there is no guarantee that this approach works in all situations.

Given that you have managed to find the stationary solution of the model, you can proceed to give the program the information from which it will compute the linearized model.

5.2 Linearization

The CoRRAM program has two options that are chosen by the flag `equations`. If the flag is set to true (the default), numeric differentiation will be used, otherwise the user must provide the coefficient matrices of the linearized system. However, since the quadratic part of the solution always rests on second derivatives of the system (2), the option false restricts you to linear approximate solutions.

Via Numeric Differentiation. Consider the system (2). Let

$$\mathbf{g} := [g^1(\cdot), \ldots, g^{n(x)+n(y)}(\cdot)]'$$

denote the row vector of functions and $\mathbf{g}_j$, $j = 1, 2, \ldots, 6$ the matrices of derivatives with respect to the jth argument of these functions. Furthermore, let a bar over a variable denote deviations from the stationary solution, i.e., $\bar{\mathbf{x}}_t := \mathbf{x}_t - \mathbf{x}$ and $\bar{\mathbf{y}}_t := \mathbf{y}_t - \mathbf{y}$. Then, the linearized system (2) is given by

$$\mathbf{0}_{(n(x)+n(y))\times 1} = \mathbf{g}_1 \mathbb{E}_t \bar{\mathbf{x}}_{t+1} + \mathbf{g}_2 \mathbb{E}_t \mathbf{z}_{t+1} + \mathbf{g}_3 \mathbb{E}_t \bar{\mathbf{y}}_{t+1} + \mathbf{g}_4 \bar{\mathbf{x}}_t + \mathbf{g}_5 \mathbf{z}_t + \mathbf{g}_6 \bar{\mathbf{y}}_t. \tag{7}$$

CoRRAM computes the matrices $\mathbf{g}_j$ via numeric differentiation. In the next step, it checks the flag `reduce`. If this flag is false, CoRRAM transforms the previous system of equations together with the process (1a) into the system

$$A \mathbb{E}_t \begin{bmatrix} \bar{\mathbf{w}}_{t+1} \\ \bar{\mathbf{y}}_{t+1} \end{bmatrix} = B \begin{bmatrix} \bar{\mathbf{w}}_t \\ \bar{\mathbf{y}}_t \end{bmatrix}, \quad \bar{\mathbf{w}}_t \equiv \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix}, \tag{8a}$$

$$A = \begin{bmatrix} \mathbf{g}_3 & \mathbf{g}_6 & \mathbf{g}_5 \\ \mathbf{0}_{n(z)\times n(x)} & I_{n(z)} & \mathbf{0}_{n(z)\times n(y)} \end{bmatrix}, \tag{8b}$$

$$B = \begin{bmatrix} -\mathbf{g}_1 & -\mathbf{g}_3 & -\mathbf{g}_2 \\ \mathbf{0}_{n(z)\times n(x)} & R & \mathbf{0}_{n(z)\times n(y)} \end{bmatrix}. \tag{8c}$$

Since almost all models involve some static equations, the matrix B will have some zero columns, and, thus, is not invertible. Therefore, CoRRAM solves the system (8) via the generalized Schur factorization of the matrix pencil (A, B).

Now, consider the system (3). Its linearized version is[3]

$$C_y \bar{\mathbf{y}}_t = -C_x \bar{\mathbf{x}}_t - C_z \mathbf{z}_t, \tag{9a}$$

[3]The superscripts $i \in \{1, x\}$ of $\mathbf{g}_j^i$ refer to the first and the second set of equations, respectively.

$$D_x \mathbb{E}_t \bar{\mathbf{x}}_{t+1} + D_z \mathbb{E}_t \mathbf{z}_{t+1} + F_x \bar{\mathbf{x}}_t + F_z \mathbf{z}_t = -D_y \mathbb{E}_t \bar{\mathbf{y}}_{t+1} - F_y \bar{\mathbf{y}}_t, \tag{9b}$$

$$C_x := \mathbf{g}_1^1, C_y := \mathbf{g}_2^1, C_z := \mathbf{g}_3^1, \tag{9c}$$

$$F_x := \mathbf{g}_1^2, F_y := \mathbf{g}_2^2, F_z := \mathbf{g}_3^2, D_x := \mathbf{g}_4^2, D_y := \mathbf{g}_5^2, D_z := \mathbf{g}_6^2. \tag{9d}$$

CoRRAM Computes the matrices of this system, if the flag `reduce` is true. In a second step, it partitions the vector $\bar{\mathbf{y}}_t$ and the matrix C_y

$$\begin{aligned} \bar{\mathbf{y}}_t &= \begin{bmatrix} \bar{\mathbf{u}}_t \\ \bar{\mathbf{v}}_t \end{bmatrix}, \\ C_y &= \begin{bmatrix} C_u & C_v \end{bmatrix}, \end{aligned} \tag{10}$$

solves the first set of equations for $\bar{\mathbf{u}}_t$ as a function of the remaining variables $\bar{\mathbf{x}}_t$, $\bar{\mathbf{v}}_t$, and $\mathbf{z}_t$ so that the second set of equations can be reduced to a dynamical system in fewer variables. Together with the process (1a) this system is given by:

$$\tilde{A} \mathbb{E}_t \begin{bmatrix} \bar{\mathbf{w}}_{t+1} \\ \bar{\mathbf{v}}_{t+1} \end{bmatrix} = \tilde{B} \begin{bmatrix} \bar{\mathbf{w}}_t \\ \bar{\mathbf{v}}_t \end{bmatrix}, \quad \bar{\mathbf{w}}_t \equiv \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix}, \tag{11a}$$

$$\tilde{A} := \begin{bmatrix} D_x & D_z & D_v \\ \mathbf{0}_{n(z)\times n(x)} & I_{n(z)} & \mathbf{0}_{n(z)\times n(v)} \end{bmatrix} - \begin{bmatrix} D_u \\ \mathbf{0}_{n(z)\times n(u)} \end{bmatrix} C_u^{-1} \begin{bmatrix} C_x & C_z & C_v \end{bmatrix}, \tag{11b}$$

$$\tilde{B} := -\begin{bmatrix} F_x & F_z & F_v \\ \mathbf{0}_{n(z)\times n(x)} & -R & \mathbf{0}_{n(z)\times n(v)} \end{bmatrix} + \begin{bmatrix} F_u \\ \mathbf{0}_{n(z)\times n(u)} \end{bmatrix} C_u^{-1} \begin{bmatrix} C_x & C_z & C_v \end{bmatrix}. \tag{11c}$$

$$\tag{11d}$$

Since the static equations have been removed from this system, the matrix $\tilde{A}$ is usually non-singular so that the system can be further reduced to

$$\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{w}}_{t+1} \\ \bar{\mathbf{v}}_{t+1} \end{bmatrix} = W \begin{bmatrix} \bar{\mathbf{w}}_t \\ \bar{\mathbf{v}}_t \end{bmatrix}, \quad \bar{\mathbf{w}}_t \equiv \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix}, \; W := \tilde{A}^{-1} \tilde{B}. \tag{12}$$

This system can be solved via the Schur factorization of the matrix W. In cases where $\tilde{A}$ is not invertible, CoRRAM solves (11) via the generalized Schur factorization of the matrix pencil $(\tilde{A}, \tilde{B})$.

Linearization by Paper and Pencil. Numeric derivatives cannot be as precise as analytical ones. In addition, by writing the linearized model in terms of percentage instead of absolute deviations, the coefficient matrices usually involve only elasticities so that the numeric differences between their elements are small, making matrix factorization less imprecise. Furthermore, it is often not necessary to derive the stationary solutions of all variables.

If the flag `equations` is set to false, CoRRAM expects that you provide the coefficient matrices of the linearized system (9) and have also partitioned the jump

variables in the vector $\mathbf{y}_t$ as in (10). Hence, it expects the matrices C_u, C_v, C_x, C_z, D_u, D_v, D_x, D_z, F_u, F_v, F_x, F_z, R, and Ω from this model

$$C_u\bar{\mathbf{u}}_t = -C_x\bar{\mathbf{x}}_t - C_z\mathbf{z}_t - C_v\bar{\mathbf{v}}_t, \tag{13a}$$

$$\begin{bmatrix} D_x & D_v \end{bmatrix}\mathbb{E}_t\begin{bmatrix}\bar{\mathbf{x}}_{t+1}\\ \bar{\mathbf{v}}_{t+1}\end{bmatrix} + \begin{bmatrix} F_x & F_v \end{bmatrix}\begin{bmatrix}\bar{\mathbf{x}}_t\\ \bar{\mathbf{v}}_t\end{bmatrix} = -D_u\mathbb{E}_t\bar{\mathbf{u}}_{t+1} - F_u\bar{\mathbf{u}}_t - D_z\mathbb{E}_t\bar{\mathbf{z}}_{t+1} - F_z\bar{\mathbf{z}}_t, \tag{13b}$$

$$\mathbf{z}_t = R\mathbf{z}_{t-1} + \sigma\Omega\boldsymbol{\nu}_t,\ \boldsymbol{\nu}_t \sim N(\mathbf{0}_{n(z)\times 1}, I_{n(z)}). \tag{13c}$$

Now note, given these matrices, CoRRAM does not know whether they refer to a model linearized in terms of absolute or relative deviations. Since, approximately, $\ln(x_t/x) \approx (x_t - x)/x$, the user must set the flag `logs` to true, if the matrices in (13) refer to relative deviations. Analogously, if you formulate your system of equations (2) or (3) in the natural logs of the variables rather than in their levels, the matrices in (7) or (9) refer to a model in relative deviations. In this case too, you must set the flag `logs` to true.

As an example of a log-linearized system consider the model from Section 4.3 and let the hat over a variable denote relative instead of absolute deviations, i.e., $\hat{x}_t \equiv (x_t - x)/x$ for any variable x of the model including the shocks. The matrices from (13) for this model are

$$\hat{\mathbf{u}}_t \equiv [\hat{y}_t, \hat{c}_t, \hat{i}_t, \hat{N}_t, \hat{w}_t, \hat{r}_t]',\ \hat{\mathbf{x}}_t \equiv \hat{k}_t,\ \hat{\mathbf{v}}_t \equiv \hat{\lambda}_t, \hat{\mathbf{z}}_t \equiv \hat{Z}_t,$$

$$C_u = \begin{bmatrix} 0 & -\eta & 0 & -\theta(1-\eta)\frac{N}{1-N} & 0 & 0\\ 0 & (1-\eta) & 0 & -[\theta(1-\eta)-1]\frac{N}{1-N} & -1 & 0\\ -1 & 0 & 0 & 0 & 0 & 1\\ -1 & 0 & 0 & 1 & 1 & 0\\ 1 & 0 & 0 & \alpha - 1 & 0 & 0\\ 1 & -\frac{c}{y} & -\frac{i}{y} & 0 & 0 & 0 \end{bmatrix},$$

$$[C_x, C_v] = \begin{bmatrix} 0 & -1\\ 0 & -1\\ 1 & 0\\ 0 & 0\\ -\alpha & 0\\ 0 & 0 \end{bmatrix},\ C_z = \begin{bmatrix} 0\\ 0\\ 0\\ 0\\ -1\\ 0 \end{bmatrix},\ [D_x, D_v] = \begin{bmatrix} a & 0\\ 0 & 1 \end{bmatrix},\ [F_x, F_v] = \begin{bmatrix} \delta - 1 & 0\\ 0 & -1 \end{bmatrix},$$

$$D_u = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1-\beta a^{-\eta}(1-\delta) \end{bmatrix},$$

$$F_u = \begin{bmatrix} 0 & 0 & 1-a-\delta & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$D_z = F_z = \begin{bmatrix} 0\\ 0 \end{bmatrix}.$$

5.3 Structure of Solutions

The approximate solutions of a DSGE model are feed-back rules that determine the model's endogenous variables $\mathbf{x}_{t+1}$ and $\mathbf{y}_t$ as linear or quadratic functions of the model's current states $\mathbf{x}_t$ and realizations of the shocks $\mathbf{z}_t$.

If the flag `equations` is set to false, CoRRAM computes a first-order (linear) solution given by

$$\hat{\mathbf{x}}_{t+1} = H_w^x \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\mathbf{z}}_t \end{bmatrix}, \tag{15a}$$

$$\hat{\mathbf{y}}_t = H_w^y \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\mathbf{z}}_t \end{bmatrix}. \tag{15b}$$

If the flage `equations` is true and `order` has the value 2 the solutions are

$$\mathbf{x}_{t+1} = \mathbf{x} + H_w^x \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix} + \tfrac{1}{2}\left[I_{n(x)} \otimes \begin{bmatrix} \bar{\mathbf{x}}_t^T & \mathbf{z}_t^T \end{bmatrix}\right] H_{ww}^x \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix} + \tfrac{1}{2} H_\sigma^x \sigma^2 \tag{16a}$$

$$\mathbf{y}_t = \mathbf{y} + H_w^y \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\mathbf{z}}_t \end{bmatrix} + \tfrac{1}{2}\left[I_{n(x)} \otimes \begin{bmatrix} \bar{\mathbf{x}}_t^T & \mathbf{z}_t^T \end{bmatrix}\right] H_{ww}^y \begin{bmatrix} \bar{\mathbf{x}}_t \\ \mathbf{z}_t \end{bmatrix} + \tfrac{1}{2} H_\sigma^y \sigma^2. \tag{16b}$$

If `order` is set to 1, the program returns only the linear part of this solution.

5.4 Difference Stationary Growth

The computation of impulse responses and second moments depends on the interpretation of the model's variables. In the model of equations (4) the variables are scaled by the level of labor augmenting technical progress A_t which grows deterministically at the rate $a - 1$. Many DSGE models, however, assume that A_t grows according to the law

$$\begin{aligned} a_t &= \frac{A_{t+1}}{A_t}, \\ a_t &= e^{\ln a + z_t}, \\ z_t &= \rho z_{t-1} + \epsilon_t, \quad \rho \in (-1, 1),\ \epsilon_t \sim N(0, \sigma_\epsilon), \end{aligned} \tag{17}$$

so that $\ln A_t$ is a difference stationary growth process. In this case, the program assumes that you have defined stationary variables x_t according to $X_t = x_t A_{t-1}^{\xi}$, where X_t is the level of a variable, x_t is the solution for the scaled variable computed by the program, and the exponent ξ is either zero (if the variable needs not to be scaled, as for instance hours N_t), equal to $\xi = 1$ (if division by A_{t-1} makes a variable stationary), or equal to $\xi = -\eta$, as in the case of the Lagrange multiplier in the benchmark business cycle model, where η is coefficient of relative risk aversion.

To get the correct interpretation of impulse responses and second moments[4] you must

- set the flag `DS` to true,
- provide information on `xi` in the substructure `Var`,
- use the variable a_t as the first element in the vector $\mathbf{y}_t$.

5.5 Error Messages

DSGE models have determinate solutions if:

- $n(x)+n(z)$ eigenvalues of W in equation (12), or $n(x)+n(z)$ eigenvalues of the matrix pencil (B,A) in equation (8), or $n(x)+n(z)$ eigenvalues of the matrix pencil $(\tilde{A},\tilde{B})$ of the system (11) are inside the unit circle, and
- $n(v)$ eigenvalues of W, or $n(y)$ eigenvalues of (A,B), or $n(v)$ eigenvalues $(\tilde{A},\tilde{B})$ are outside of the unit circle.

The program stops with an error message, if these two conditions are not satisfied. CoRRAM stops with a second error message, if it is not able to factorize W, (A,B), or $(\tilde{A},\tilde{B})$. This message usually points to errors in your code.

When the program computes the linear part of the solution it uses complex arithmetic. If the matrices of your model are large and/or rather unbalanced, roundoff errors may cause the complex part of the solution matrices to be non-negligible. The respective tolerance is stored in `i_tol`. If this happens, the program issues a warning message. Please check, if errors in your code cause this problem. You may also try to set the flag `bal` to true. (see Heiberger et al. (2017) for the balancing of a matrix pencil).

6 Providing a Model to CoRRAM

A model consists of parameters, variables, and equations. The first step, therefore, is to provide CoRRAM with the model's parameters and to compute the stationary solution of the model.

6.1 Provide the Parameters and Compute the Stationary Solution

For the model in Section 4.3 the respective code is:[5]

[4] See Maußner (2019), Section 9.24 for the computation of impulse responses in such a model.

[5] These and the following code snippets are from the example program `CoRRAM_Test_2.g`. The lines before line 13 include comments.

```
13  new;
14  library CoRRAM6;
15
16  // Parameters
17  a=1.003;       // growth factor of per capita output
18  alpha=0.37;    // capital share in output
19  beta1=0.996;   // discount factor
20  eta=2;         // coefficient of relative risk aversion
21  delta=0.011;   // rate of capital depreciation
22  rhoZ=0.89;     // autocorrelation of AR(1) process for log TFP
23  sigmaz=0.0073; // standard deviation of innovations in AR(1) process for log TFP
24  nstar=0.123;   // stationary hours per capita as fraction of 16*90 hours per quarter
25
26  // Stationary solution
27  yk=(a^eta-beta1*(1-delta))/(beta1*alpha);
28  kn=yk^(1/(alpha-1));
29  kstar=kn*nstar;
30  ystar=yk*kstar;
31  istar=(a-1+delta)*kstar;
32  cstar=ystar-istar;
33  wstar=(1-alpha)*(ystar/nstar);
34  rstar=alpha*(ystar/kstar);
35  theta=(wstar/cstar)*(1-nstar);
36  lstar=(cstar^(-eta))*((1-nstar)^(theta*(1-eta)));
```

The first statement in line 13 clears the memory and the second statement provides Gauss with information about the location of the procedures of the CoRRAM toolbox. This statement *must* be the first executable statement in all programs that employ the toolbox. The statements in lines 16-36 should be self-explanatory.



6.2 Initialize the Model Envirionment

The second step is to initialize an instance of the DSGE structure with the following commands:

```
39   struct DSGE BM;
40   BM.nx=1;
41   BM.ny=7;
42   BM.nz=1;
43   BM.nu=6;
44   BM.file="Benchmark.txt";
45   NewModel(&BM);
46
47   // Store the solution
48  BM.wstar=(kstar|ystar|cstar|istar|nstar|wstar|rstar|lstar);
49
50  // Provide the model's shock process
51  BM.Rho[1,1]=RhoZ;
52  BM.Omega[1,1]=SigmaZ;
53
54  // optional: collect the parameters
55  Par=alpha|beta1|delta|eta|theta;
56
```

The command in line 39 creates an empty instance of the DSGE structure. Unfortunately, Gauss does not allow to set the members of this structure to default values

different from zero. Before we supply the new object with default values, we must inform CoRRAM about the number of the variables, the number of static equations, and the file to which CoRRAM will print error messages and results. The statements in lines 40-44 achieve this. The command in line 45 receives a pointer to the newly created model in BM and sets its elements to the default values presented in Tables 2 and 4. In addition, it initializes the Var substructure with default variables names, types, and positions.

The command in line 48 stores the vector of stationary solutions in the structure member wstar. A second way to inform CoRRAM about the stationary solution is to set the property star of each member of the substructure Var[*].star individually. If CoRRAM finds that wstar has less than $n(x)+n(y)$ elements, it screens the substructure Var and stores the stationary values in wstar in the order

$$(x_1,\ldots,x_{n(x)},y_1,\ldots,y_{n(y)})'.$$

Note that the NewModel command initializes the first $n(x)$ members of Var with variables of type x, the next $n(y)$ members with variables of type y, and the remaining $n(z)$ members with variables of type z. The latter have always zero stationary value so that there is no need to store their values in wstar. If you want to change the default ordering of the elements of Var, you must redefine the type and pos elements of *all* members of Var!

Lines 51 and 52 inform CoRRAM about the stochastic process (1a). The order of the elements of this vector follows from the order in which you store the parameters in the matrices Rho and Omega.

The command in line 55 stores the parameter values in the vector Par. This vector will be passed by CoRRAM to the procedure that encodes the model's equations. You can skip this step, if you use global variables for the model's parameters.

6.3 Provide the Model's Equations

In the third step you must provide either the model's equations or the matrices of the system (13). If you decide on the latter, note that the procedure NewModel has already initialized the respective members of the DSGE structure (see Table 2) to matrices of the correct size. Therefore, you can start to set the non-zero elements of the respective matrices to their corresponding values.[6] Also, if you input the log-linearized model, CoRRAM does not require the information in wstar and the commands in lines 48 and 55 of the previous listing are not necessary.

Usually, you will supply the equations instead of the matrices. In this case, you must write a procedure. This procedure has two mandatory arguments and one optional argument and must return a vector, fx, say, whose elements are equal to the

[6] See the program CoRRAM_Test_1.g.

left-hand sides of equations (2) or (3) evaluated at the values passed to the procedure in the vector `w`. This vector is the first mandatory argument of the procedure. The second argument is the scalar `neq`. The optional argument is `par`. `w` refers to the vector

$$\mathbf{w} = [\mathbf{x}^1, \mathbf{y}^1, \mathbf{z}^1, \mathbf{x}^2, \mathbf{y}^2, \mathbf{z}^2]', \tag{18}$$

where the superscript $i \in \{1, 2\}$ refers to variables at time t and $t + 1$. If the optional argument is not present, CoRRAM assumes that the parameters of the model are defined as global variables, otherwise you pass their values in the vector `par`. Furthermore, if the flag `reduce` is set to true, the program assumes that the first `nu` equations in the procedure are static equations.

The next lines of program code define the procedure that supplies the equations of the the model from Section 4.3.

```
103  proc(1)=Sys1(w,eqno,Par);
104
105    local fx, c1, y1, i1, r1, w1, n1, k1, l1, z1, c2, y2, i2, r2, w2, n2, k2, l2, z2;
106    local alpha, beta1, eta, delta, theta;
107
108    alpha=Par[1]; beta1=Par[2]; delta=Par[3]; eta=Par[4]; theta=Par[5];
109
110    k1=w[1];              k2=w[10];
111    y1=w[2];              y2=w[11];
112    c1=w[3];              c2=w[12];
113    i1=w[4];              i2=w[13];
114    n1=w[5];              n2=w[14];
115    w1=w[6];              w2=w[15];
116    r1=w[7];              r2=w[16];
117    l1=w[8];              l2=w[17];
118    z1=exp(w[9]);         z2=exp(w[18]);
119
120    fx=ones(9,1);
121
122    fx[1]=1-(c1^(-eta))*((1-n1)^(theta*(1-eta)))/l1;
123    fx[2]=theta*c1-(1-n1)*w1;
124    fx[3]=w1-(1-alpha)*(y1/n1);
125    fx[4]=r1-alpha*(y1/k1);
126    fx[5]=y1-z1*(n1^(1-alpha))*(k1^alpha);
127    fx[6]=y1-c1-i1;
128
129    fx[7]=a*k2-(1-delta)*k1 - i1;
130    fx[8]=1-beta1*(a^(-eta))*(l2/l1)*(1-delta+r2);
131
132    if eqno==0; retp(fx); else; retp(fx[eqno]); endif;
133
134  endp;
```

The name of the procedure can be any acceptable name for Gauss procedures. Here, the procedure has the name `Sys1`. The statements in lines 105 and 106 define local variables. The command in line 108 reads the values of the model's parameters from the vector `Par` into the local parameters `alpha`, etc. For ease of coding, I also prefer

to write the values in the vector `w` into variables with names similar to those used in (4).[7] This is done in lines 110-118. It is important for this step to note that

i. the variables in `w` are ordered as defined in (18), i.e.,
   - the first $n(x)$ elements refer to state variables at time t,
   - the next $n(y)$ elements refer to jump variables at time t,
   - the next $n(z)$ elements refer to the shocks at time t,
   - the next $n(x)$ elements refer to state variables at time $t+1$,
   - the next $n(y)$ elements refer to jump variables at time $t+1$,
   - and the remaining $n(z)$ elements refer to the shocks at time $t+1$.

ii. The ranking within the vectors $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ is determined by the ranking provided by the user,
   - either explicitly by setting the `Var[*].pos` for each variable and assigning the respective stationary value to `Var[*].star`
   - or implicitly, as in line 48 of the previous listing, where the stationary value of output is stored in the first element of the vector $\mathbf{y}_t$, etc.

The code in line 120 initializes the vector `fx`. The commands in lines 122-130 overwrite this vector. Lines 122-127 evaluate the static equations (4a)-(4f), lines 129 and 130 the dynamic equations (4g) and (4h). Line 132 returns the vector `fx` to the calling program. The `if-else-endif` construct is required to compute second derivatives (which is done equation by equation) and must not be changed!

CoRRAM checks whether or not the model's equations are satisfied at the stationary solution. The user can turn off this check by setting `CheckEquations` to false. However, I do not recommend this setting. If `CheckEquations` is true, any element in the vector `fx` larger than the tolerance in `e_tol` will produce an error. CoRRAM writes the respective equation number and the associated value to the file whose name is specified in the string `file`.

6.4 Solve the Model

Having specified the model's equations or the respective matrices, the command

```
58  rc=SolveModel(&BM,&Sys1,Par);
```

solves the model. The procedure `SolveModel` has two required arguments and the optional argument `Par`. The latter is required, if you want local instead of global parameter in the procedure `Sys1`. The first required argument is the pointer to

[7] Of course, you can use `w[1]`-`w[18]` instead of the auxiliary variables.

the model in BM and the second is the pointer to the model's equations in Sys1. If CoRRAM fails to compute the solution, rc is set to false and you can use this information to terminate the program. Otherwise, depending on the setting of order the matrices Hxw, Hyw, Hxww, Hyww, Hxss, and Hyss – corresponding to the solution (16) – are available.

6.5 Evaluate the Model

The next step is to evaluate the performance of the model and to get an understanding of its internal mechanics. The latter is aided by the study of impulse responses of the model's variables to one of its shocks and the former by tables of second moments of simulated time series. CoRRAM is able to provide valid impulse responses and second moments for the following models:

- models with trend stationary growth, formulated either in levels or logs of the variables,
- models with one difference stationary growth process formulated in the levels of the variables.

Other models, as, e.g., a model with two different difference stationary growth processes, must be handled by the user.

The command

```
61  if rc; rc=SimulateModel(&BM); endif;
```

invokes the procedures that compute impulse responses and second moments, if CoRRAM successfully solved the model (so that rc has the value true). The single argument of the command is again the pointer to the model. SimulateModel resets rc to false, if it fails to simulate the model. Without any further settings, CoRRAM uses the default values of simulate, nirf, nobs, nofs and

- computes impulse responses with nirf periods for all variables and all shocks and stores the results in the three dimensional array IRF,
- and computes a set of second moments that includes standard deviations, autocorrleations, and correlations between all variables of the model. These results are saved in the structure members Corr_m, Corr_l, and Corr_u.

The array IRF has three dimensions: IRF[s,t,i] stores the impulse response of variable i at period t to the shock s. The array Corr_m (as well as Corr_l and Corr_u) also has three dimensions: Corr_m[k,i,j] stores the correlation between the variable i at period t and variable j at period $t-(k-1)$ in the elements above the main diagonal and between variable i at $t+(k-1)k$ and variable j at time t in

the elements below the main diagonal. The diagonal elements in page $k+1$ hold the autocorrelations at lag k.

There are several options that determine the evaluation of the model:

- The flag `simulate` determines whether second moments are from simulated time series or are computed analytically from the the matrices H_w^x, H_w^y, R, and Ω. However, sensible results require that the model is formulated in the logs of its variables.
- The value of `HPL` determines whether second moments refer to data passed through the Hodrick-Prescott filter (see Hodrick and Prescott (1997)). If `HPL` is set to the weight of this filter, second moments refer to filtered data, otherwise the refer to raw data. This holds for both settings of `simulate`. .
- Simulations of time series require random numbers (the variables in the vector ν_t in equation (1c)). The setting of `seed` determines the seed of Gauss' random number generator.
- Simulations of time series can produce upper and lower bounds of the second moments so that $1-\alpha$ percent of the respective moments are within these bounds. If `confidence` is set to true, CoRRAM computes bounds corresponding to 1-`alpha` percent confidence intervals.
- Finally, you might want to see whether during simulations some of your variables exceed certain bounds. In this case, you must set the flag `CheckBounds` to true. and provide upper and lower bounds for each variable through the `bounds` elements of the `Var` substructure. If CoRRAM detects a variable outside the given bounds, it prints an error message to the file whose name is given in `file`.

6.6 Graphs and Tables

The experienced user will produce graphs of impulse responses and tables with second moments from the simulation output of CoRRAM himself. For users who want CoRRAM to perform this task the commands `Graphs` and `Tables` are available. However, before you can invoke these commands, you must inform CoRRAM about the desired results. You can do this either via editing the substructure `Var` or via optional arguments.

Configure Tables and Graphs. The member `plotno` of the substructure `Var` determines whether the impulse response of the respective variable will be plotted. CoRRAM displays one graph for each of the $n(z)$ shocks. Each graph can be split

into at most eight panels. The number assigned to `plotno` determines into which panel the respective response will be plotted. If `plotno` is set to 0, no response of the respective variable is drawn. The appearance of the plots is determined by the settings of `color` (see Table 4).

The members `print`, `rel`, `cor` of the substructure `Var` determine the output of the the `Tables` command. If `print` is true, second moments of the respective variable are taken out of `Corr_m`. These include

- the standard deviation of this variable,
- the standard deviation of this variable relative to any other variable for which both flags `print` and `rel` are set to true,
- correlations at `maxlag` leads and lags of this variable with any other variable for which the flag `print` is true,
- and autocorrelations of this variable at `maxlag`.

For instance, the next lines of program code produce Figure 2 and Table 5 for the model from Section 4.3:

```
63  // provide information for graphs and tables
64  BM.Var[1].name="Capital";
65  BM.Var[1].plotno=4;
66
67  BM.Var[2].name="Output";
68  BM.Var[2].plotno=1;
69  BM.Var[2].print=1;
70  BM.Var[2].rel=1;
71  BM.Var[2].cor=1;
72
73  BM.Var[3].name="Consumption";
74  BM.Var[3].plotno=2;
75  BM.Var[3].print=1;
76
77  BM.Var[4].name="Investment";
78  BM.Var[4].plotno=2;
79  BM.Var[4].print=1;
80
81  BM.Var[5].name="Hours";
82  BM.Var[5].plotno=3;
83  BM.Var[5].print=1;
84
85  BM.Var[6].name="Real Wage";
86  BM.Var[6].plotno=3;
87  BM.Var[6].print=1;
88  BM.Var[6].cor=1;
89
90  BM.Var[9].name="TFP Shock";
91  BM.Var[9].plotno=1;
92
93  if rc; Graphs(&BM); rc=Tables(&BM); endif;
```

Figure 2: Impulse Responses from the Benchmark Business Cycle Model

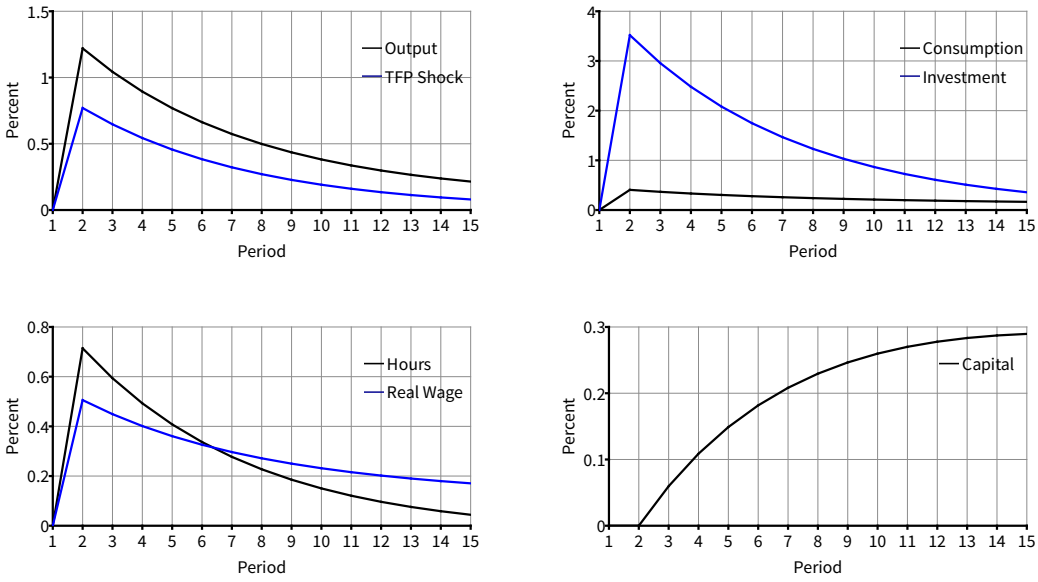


Table 5: Sample output of CoRRAM

```
Table of second moments:
from   500 valid simulations of length  120
Order of approximation: 1
HP-filter weight: 1600

Output      1.46 1.00 1.00 1.00 0.62
Consumption 0.49 0.33 0.99 1.00 0.64
Investment  4.24 2.90 1.00 0.99 0.62
Hours       0.86 0.59 1.00 0.99 0.62
Real Wage   0.61 0.41 1.00 1.00 0.63

Column  1: Variable
Column  2: Standard deviation
Column  3: Standard deviation relative to variable Output
Column  4: Contemporaneous correlation with variable Output
Column  5: Contemporaneous correlation with variable Real Wage
Column  6: First order autocorrelation
```

Optional Arguments. The commands `Table` and `Graphs` have optional arguments that allow for a more direct approach to print tables and graphs with impulse re-

sponses. The optional arguments of `Table` are

- a string vector with the names of the variables to be included in the table,
- a corresponding vector with the indices of these variables,
- and a vector with indices for additional columns with correlations.

For instance, the next lines of code also produce Table 5:

```
VarNames="Capital"$|"Output"$|"Consumption"$|"Investment"$|
         "Hours"$|"Real Wage"$|"TFP-Shock";
printIndex=(1|2|3|4|5|6|9);
corIndex=(2|6);
rc=Tables(&BM,VarNames,printIndex,corIndex);
```

Note that `corIndex` must be a subset of the indices in `printIndex`.

The procedure `Graphs` has two optional inputs:

- a string vector with the names of the variables for which CoRRAM shall plot impulse responses
- and a corresponding matrix whose first column holds the indices and whose second column stores the panel number.

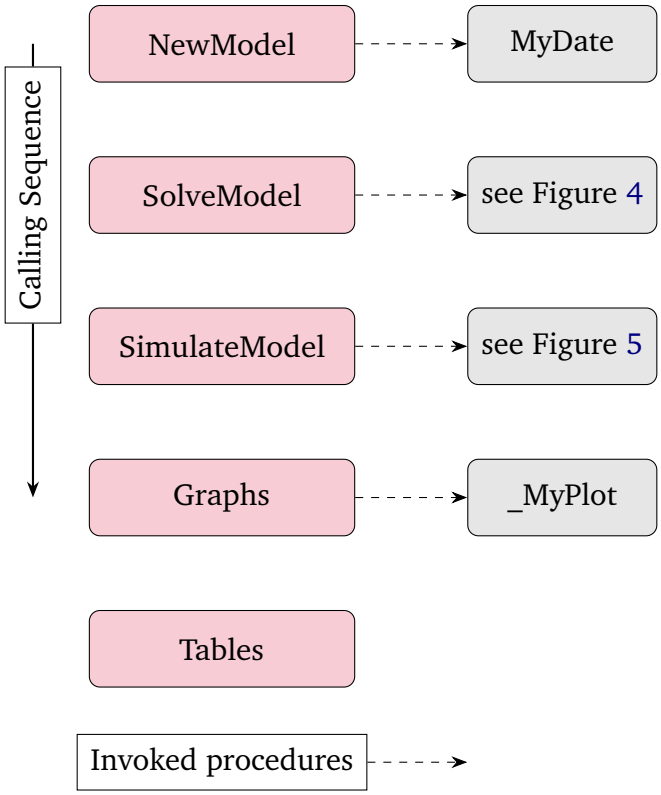
Thus, a more direct way to obtain Figure 2 are the next lines of code:

```
VarNames="Capital"$|"Output"$|"Consumption"$|"Investment"$|
         "Hours"$|"Real Wage"$|"TFP-Shock";
plotInfo=(1~4)|(2~1)|(3~2)|(4~2)|(5~3)|(6~3)|(9~1);
Graphs(&BM,VarNames,plotInfo);
```

7 Inside the Black Box

The default options for solving and simulating a model suffice for many standard models. However, more advanced models require additional programming by the user. This section discloses the structure of the CoRRAM toolbox and provides hints for using its subprograms. This will give the user more flexibility in solving and simulating DSGE models. If you feel that you can get along with the standard options you can skip this section.

Figure 3: Structure of the toolbox



7.1 Structure of the Toolbox

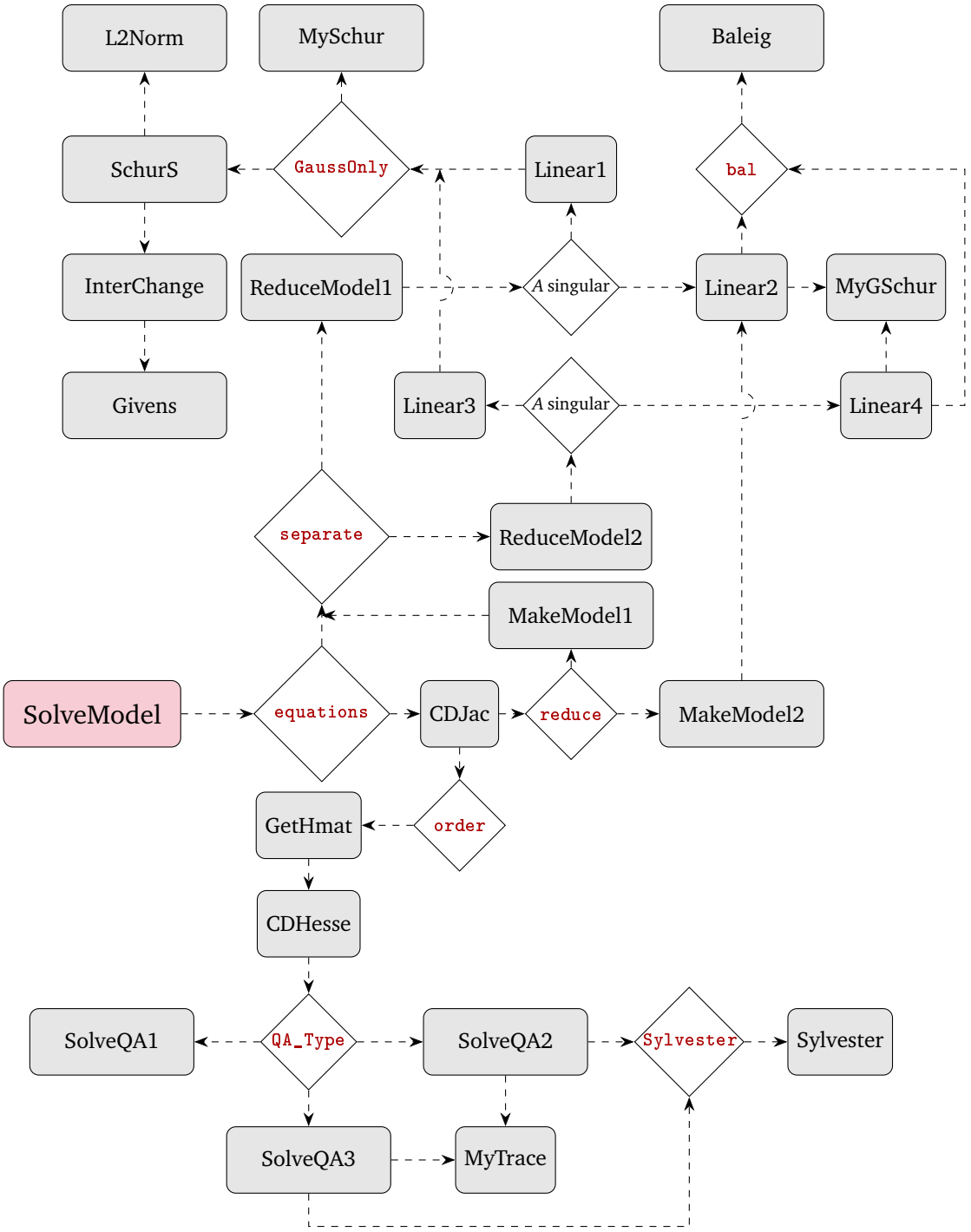
Figure 3 displays the basic structure of the toolbox. The three main procedures are `NewModel`, `SolveModel`, and `SimulateModel`. They must be called in the sequence shown in Figure 3. Afterwards, the procedures `Graphs` and `Tables` can be called in arbitrary order. The latter procedure writes its results to the file whose name is provided by the string `file`. If the the flag `Excel` is true, the same set of tables is also written to an Excel spreadsheet. The name of this spreadsheet is also taken from the string `file` and appended with the file ending `.xlsx`.

The procedures `SolveModel` and `SimulateModel` call themselves several other procedures that make up the toolbox. Some of the procedures will only be called, if the respective flag is true. The procedures called by `SolveModel` are displayed in Figure 4 and those called by `SimulateModel` are shown in Figure 5.

The procedures of CoRRAM are scattered over various source files. The library file `CoRRAM6.lcg` lists the source files and the names of the procedures that are stored in these file. Just open this file with Gauss and search for the name of a procedure. Next open the respective source file and search again for the name of the procedure.

Each procedure has a header that informs about its usage.

Figure 4: Procedures Called by SolveModel



I will now explain the core routines of CoRRAM in more detail. Consider, first, the linear part of the solution (15).

7.2 Linear Solutions

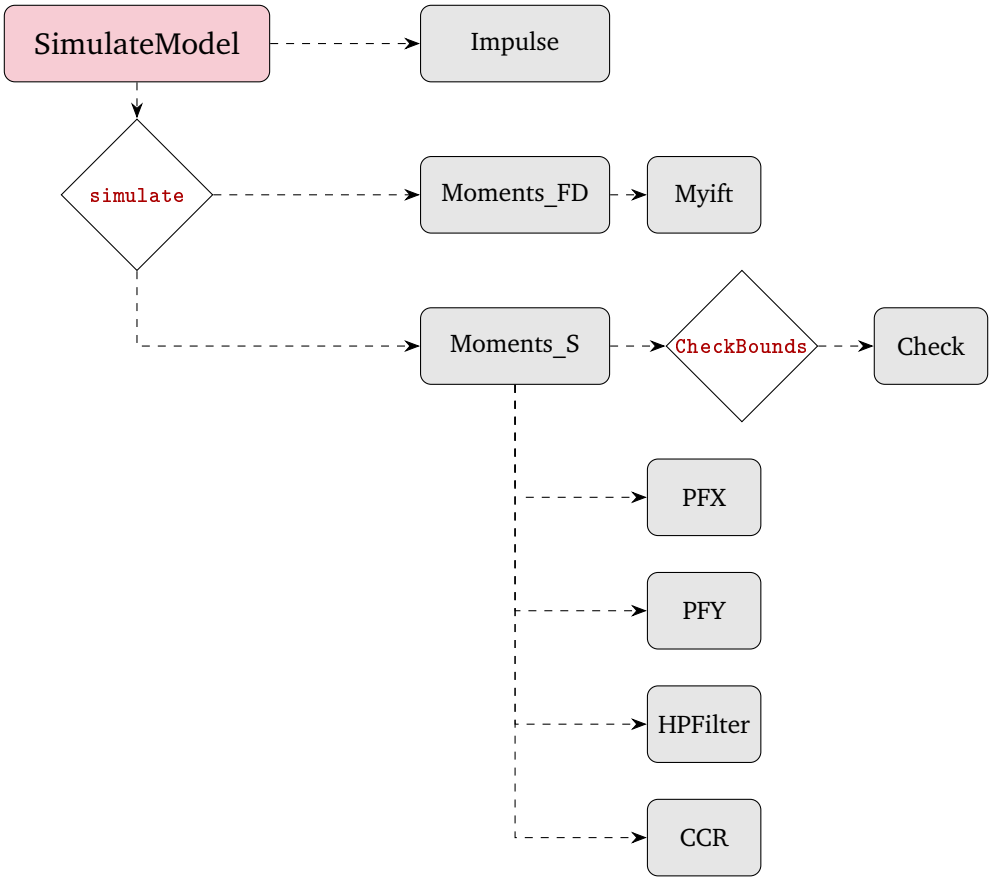
In Figure 4 the dashed arrows indicate the sequence of procedure calls. `SolveModel` first checks the flag `equations`. If this flag is false, it expects that the matrices from equation (13) are available. Otherwise the procedure `CDJac` computes the matrices from equation (7). The setting of the flag `reduce` determines the next steps. If `reduce` is true, the procedure `MakeModel1` builds the matrices of the system (11); otherwise the procedure `MakeModel2` builds the matrices of the system (8). The solution of the reduced model depends on the setting of the flag `separate`. If this flag is set to true, CoRRAM solves in separate steps for the deterministic and the stochastic part of the linear policy function. If the model must be solved repeatedly for unchanged matrices A and B or $\tilde{A}$ and $\tilde{B}$ but for different values of the matrix R this approach saves time. If `separate` is set to false, CoRRAM computes the matrices from equation (15) in one step. Both the procedure `ReduceModel1` and the procedure `ReduceModel2` check whether the matrix $\tilde{A}$ from equations (11) is invertible. If this test fails, they either call `Linear2` or `Linear4` depending on the flag `separate`; otherwise `SolveModel` either calls the procedure `Linear1` or the procedure `Linear3`. Both solve the model via the Schur factorization of the matrix W from equation (12) and employ two different procedures to compute this factorization. If the flag `GaussOnly` is true, only Gauss commands are used to obtain the complex Schur form of W. Gauss, however, cannot order the eigenvalues. `SchurS` orders the eigenvalues on the main diagonal of S via repeated calls to `L2Norm`, `InterChange` and `Givens`. The procedure `MySchur` returns an already ordered Schur form. However, it requires that the dynamic link library `Tools.dll` is available.

7.3 Quadratic Solution

`SolveModel` also computes the quadratic part of the solution (16), if the structure member `order` has a value of 2 and if the computation of the linear part was successful. The quadratic part requires the second-order derivatives of the system (2). Note that each of the $n(x)+n(y)$ equations has a $2(n(x)+n(y)) \times 2(n(x)+n(y))$ Hesse matrix of second-order derivatives. Depending on the respective algorithm, these matrices are either stored in a three-dimensional array (if `QA_Type` has the value 1) or in a large matrix. The procedure `GetHmat` returns the required form. For each of the $n(x)+n(y)$ equations it calls the procedure `CDHesse`. This procedure returns central difference approximations of the second-order derivatives of the system (2).

The procedure `SolveQA1` implements the algorithm from Section 2.5.4 of Heer and Maußner (2009). The procedure `SolveQA2` implements the algorithm proposed by Gomme and Klein (2011). One step in this algorithm is to solve a generalized Sylvester equation. If the flag `Sylvester` is true, `SolveModel` calls the procedure `Sylvester` in `Tools.dll`; otherwise it transforms this equation to a large linear system that can be solved without foreign language code. Finally, the procedure `SolveQA3` solves models with shocks that have the mean preserving spread property (see Heiberger and Maußner (2020) for the respective algorithm).

Figure 5: Procedures Called by SimulateModel



7.4 Simulation

Impulse Responses. The procedure `Impulse` computes the impulse responses and writes its results to the array `IRF`. The number of periods is determined by `nirf`. The procedure employs the linear part of the solution and computes percentage deviations of the model's variables from their respective trend.

The time path $\{z_{st}\}_{t=1}^{T}$ of each shock $s = 1, \ldots, n(z)$ is computed from (1) given a one-standard deviation shock $\nu_{s2} = \Omega \mathbf{e}_s$, where $\mathbf{e}_s$ is the row vector with its s-th element equal to one and zeros elsewhere.

Second Moments. CoRRAM is able to compute second moments either from simulations of the model or analytically from the linear part of the solution. If the flag `simulate` is set to false, CoRRAM computes second moments as proposed by Uhlig (1999). The number of frequency points used by this algorithm is stored in `nfreq`. Meaningful results require that your model is formulated in the logs of the variables and does not include difference stationary growth processes. The computations use my implementation of the inverse Fourier transform in `Myift`.

To handle other kinds of models, you must simulate time series from either the linear or quadratic solution of the model. Simulations are enabled by setting the flag `simulate` to true. `SimulateModel` then invokes the procedure `Moments_S`. Depending on the flag `order`, this procedure either uses the linear or the quadratic solution of the model. These solutions are encoded in the functions `PFX` for the vector of states $\mathbf{x}_{t+1}$ and in `PFY` for the vector of jump variables $\mathbf{y}_t$. If the number stored in `HPL` is larger than zero, the results of the simulation are passed through the Hodrick-Prescott filter. Finally, the procedure `CCR` computes correlation coefficients between all variables at all lags between $k = 0$ and $k =$`maxlag`. The mean of these correlations over `nofs` simulations is stored in the three-dimensional array `Corr_m`. If the flag `confidence` is true, upper and lower bounds, corresponding to a confidence interval of size `1-alpha` percent, are provided in the arrays `Corr_l` and `Corr_u`.

Economic variables are usually bounded from below and above. For instance, output cannot be negative and working hours per day cannot exceed 24 hours. If you want to check whether your model's variables remain within their bounds during simulations, you can do this by setting the flag `"CheckBounds"` to true. Before you start the simulations, however, you must define these bounds for each variable in the substructure element `Var[*].bounds`. This is a vector with two elements, the first stores the lower bound and the second the upper bound. Then, the procedure `Check` tests at each simulation step whether any of the variables lies outside the bounds. The respective simulation is ignored. You find the number of valid simulations in the structure member `valid`. If you do not change the value of `seed`, different simulations of the same model or simulations with different models will always employ the same set of random numbers. Therefore, differences in the results of different models will stem from systematic and not random differences between the considered models.

7.5 Stand Alone Procedures

In addition to the procedures referenced in Figures 3-5 CoRRAM has a few additional procedures that may be helpful for writing a program to solve and simulate a model. Table 6 list these programs together with a short description and the place where they are stored.

Table 6: Stand Alone Procedures

Name	Place	Purpose
BiSec	CoRRAM_3.src	Finds the zero of a single valued function in the interval $[a, b]$.
GetMom	CoRRAM_2.src	Extracts a table of second moments from Corr_m
GetOmega	CoRRAM_3.src	Computes the matrix Ω in (1) from the covariance matrix Σ.
Lyapunov	CorRAM_2.src	Solves the Lyapunov equation $X = AXA' + B$ for the matrix X.
MachEps	CoRRAM_3.src	Gets the machine epsilon
Moments_TD	CoRRAM_2.src	Computes analytic second moments for unfiltered data and unconditional first moments.
PF	CoRRAM_2.src	The policy function of a solved model.
QuickSolve_S1	CoRRAM_4.src	Given a previous solution, solves the same model but for a different R matrix in equation (1a) if the matrix A in equation (8)is invertible.
QuickSolve_S2	CoRRAM_4.src	Given a previous solution, solves the same model but for a different R matrix in equation (1a) if the matrix A in equation (8) is not invertible.
WSS	CoRRAM_1.scr	Prints the list of variables and their stationary solution.

REFERENCES

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

Gomme, P. and P. Klein (2011). Second-order approximation of dynamic models without the use of tensors. *Journal of Economic Dynamics and Control 35*, 604–615.

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

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 (2020). Perturbation solution and welfare costs of business cycles in dsge models. *Journal of Economic Dynamics and Control*. https://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*, 1–16.

Maußner, A. (2019). Computational macroeconomics. Lecture Notes, University of Augsburg.

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

Uhlig, H. (1999). A toolkit for analysing nonlinear dynamic stochastic models easily. In R. Marimon and A. Scott (Eds.), *Computational Methods for the Study of Dynamic Economies*, pp. 30–61. Oxford: Oxford University Press.