Manual for CSD toolbox

a perturbation solver for Matlab,
based on CoRRAM-M

Sijmen Duineveld

CSD, version 2.4
September 1, 2021

Contents

Part I

Manual

1. Introduction

The toolbox `CSD` can solve Dynamic Stochastic General Equilibrium models using perturbation methods. The `CSD` toolbox is completely based on the `CoRRAM-M` toolkit (2018 version) developed by Professor Alfred Maussner[1]. `CSD` is in fact an acronym for Corram Simplified Design. The `CSD` toolbox does not have the same functionality as the original `CoRRAM` package, but is adjusted to make it easier to use[2]. One limitation is that it can currently only handle symbolic models.

The core of the `CSD` toolbox consists of two functions. The calculation of the policy function with the function `pert_ana_csd`, and the evaluation of the policy functions given the state variables with the function `eval_sol_csd`. This simple design of the `CSD` toolbox makes it especially useful for educational purposes, simulations and simple calibration procedures, and to obtain the initial guess for policy functions for projection methods. The simple design is also reflected in the small file size. The installed `CSD` toolbox takes up less than 1 MB of disk space, while `Dynare 4.6.4` takes up about 1 GB.

To solve a model with the `CSD` toolbox the modeler has to program a symbolic model file. This symbolic model is fed into the solver, similar to the popular `Dynare` software. There are some advantages over `Dynare`. In contrast to `Dynare` the `CSD` toolbox only uses functions, and no global variables, and does not save any files. This keeps your Matlab workspace and folders clean.

Another advantage of the `CSD` solver is that it can also handle saddle cycle models Galizia (forthcoming), ie. model exhibiting attracting limit cycles. For this purpose we have integrated the code `InvSubGen` developed by Galizia (forthcoming) into the the function `pert_ana_csd_lim`[3]. This function is similar to the function `pert_ana_csd`, and gives all the candidate solutions. One of the examples uses this function to replicate the saddle cycle model described in Galizia (forthcoming), which is a modified version of the model of Beaudry, Galizia, and Portier (2020).

We show with the program `compare_csd_dynare_rbc` in the folder 'Examples' that the solution to a standard RBC model of the `CSD` toolbox and `Dynare` are practically the same. After a simulation of a 1000 periods the differences in the variables (in logs) are smaller than 1e-12.

[1] The `CoRRAM-M` package can be found at https://www.uni-augsburg.de/de/fakultaet/wiwi/prof/vwl/maussner/dgebook/

[2] In addition, simulations will be faster than with the 2018 `CoRRAM` package, because `CSD` uses vectorization in simulations.

[3] It should be noted that if a stable manifold exists it is not guaranteed to be spanned by the third order pertubration approximation. Therefore, the third order approximation might not the satisfy the Transversality Condition, even if when the exact model is saddle cycle stable and satisfies the Transversality Condition.

1.1 Installation

For the installation download the 'CSD_v02.4.zip' file from the website https://www.saduineveld.com/tools, and unpack it in a folder. This will add the folder 'CSD_v02.4' to the destination folder. The folders and files of the `CSD` toolbox are shown in Figure 1.1.

In order to use the `CSD` toolbox in Matlab one needs to add the folder 'CSD_v02.4' and the subfolder 'subfun' to the searchpath. After unpacking the `zip` file in the folder 'C:\Myfolder' one can add 'CSD_v02.4' and all its subfolders to the searchpath with the Matlab command:

```
addpath(genpath('C:\Myfolder\CSD_v02.4'));
```

The folder 'CSD_v02.4' also has a subfolder 'Examples' which contains five example programs:

- `compare_csd_dynare_rbc` compares the perturbation solution of the `CSD` toolbox with the `Dynare` solution. This program shows that the solutions of the `CSD` toolbox and `Dynare` are practically the same (up to order 3). The file is not further documented, but self-explanatory. The model is the same model as in the example `main_stand_rbc_pert`. The `Dynare` solution is included, but it can recomputed by setting:

  ```
  run_dyn = 1;
  ```

  This requires that `Dynare` is installed. The file is tested with `Dynare 4.6.4`;

- `main_sgu` replicates the second order perturbation solution as in Schmitt-Grohé and Uribe (2004). This example is discussed in Chapter 2;

- `main_stand_rbc_pert` demonstrates a standard RBC model, including a simulation. This example is discussed in Chapter 3;

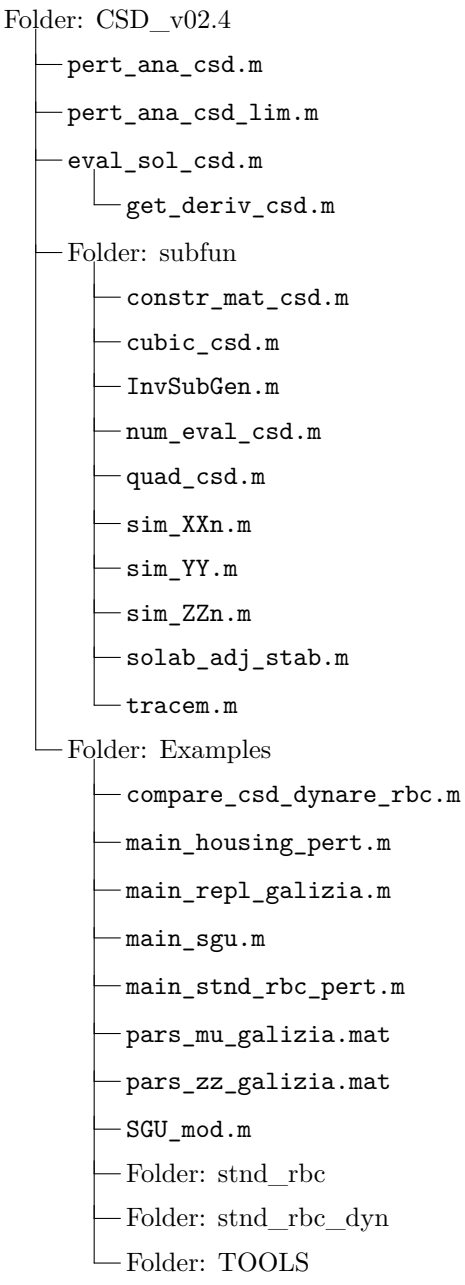
- `main_housing_pert` which solves an RBC model with two endogenous state variables, housing and capital. It also simulates the model. This example is not documented;

- `main_repl_galizia` which solves the saddle cycle model BGPe by Galizia (forthcoming). This model is described in Chapter 4, and uses the function `pert_ana_csd_lim` to solve the model.

1.2 Peculiarities

There are three peculiarities of the `CSD` solver.

Figure 1.1: `CSD` folders and files



Exogenous processes

The first is that exogenous processes should not be included in the model function. The exogenous process is defined as:

$$Z_{t+1} = \rho Z_t + \Omega \varepsilon_{t+1} \tag{1.1}$$

where Z is a vector of length nz, and both ρ and Ω are $nz \times nz$ matrices. Both ρ and Ω are inputs arguments when calling the solver `pert_ana_csd`.

Defining variables

The second peculiarity is that all variables need to have specific extensions. The extension $\_t$ is compulsory for current period variables, and the extension $\_n$ for period $t+1$ variables as shown in Section 2.1. For example assume there is a variable Q. In the model file Q_t has to be called Q_t, and Q_{t+1} has to be called Q_n. If the model also includes the lagged variable Q_{t-1}, then the modeler has to create an extra variable, for example $Q1\_t = Q_{t-1}$.

Policy functions compared to Dynare

The policy functions or decision rules in `Dynare` are reported differently. This is related to the treatment of the exogenous state variables. Assume we have a policy variable C_t that only depends on the state variable Z_t, which has exogenous shocks ϵ_t. The stochastic process is given by $Z_t = \rho Z_{t-1} + \sigma \epsilon_t$. The exact policy is $C_t = C(Z_t)$. For this model `Dynare` will use the following first order policy function:

$$\tilde{C}_t = a_z \tilde{Z}_{t-1} + a_\epsilon \epsilon_t$$

where the tilde indicates the deviation from the steady state. The `CSD` toolbox will use the first order policy function:

$$\tilde{C}_t = b_z \hat{Z}_t$$

The coefficients are then related as $a_z = \rho b_z$ and $a_\epsilon = \frac{b_z}{\sigma}$. Similar relations hold for the coefficients of higher order policy functions.

1.3 Other remarks

Technical details

This short manual does not include any details on the solution method.

Notes on typesetting

In this manual names in general are referred to by single quotations, like a folder name 'Myfolder'. Variables, cell arrays, structure names, fields of structures, objects, and properties of objects in Matlab are referred to in the text with the mathematical font of Latex, for example variable x, structure par or the field of a structure $par.alpha$. In general we use double letters in our programs such as xx, because this makes it easier to find them in a file. In this documentation we generally refer to variables by the single letter (x). Strings in Matlab code will be referred to in Matlab typesetting, for example `'thisstring'`. Names referring to toolboxes, code, functions or scripts are in Typewriter font, as in `myfunction`, where the `.m` extension of functions and scripts will be omitted for simplicity.

Scripts versus functions

There are two main differences between a function and a script in Matlab. The first is that one cannot define a subfunction in a script. The second is that a script will use the current workspace, while a function has its own workspace, which is empty unless input arguments are defined or global variables are used[4]. When using a function one can evaluate the variables in the workspace by placing a breakpoint, for example just before the end of the code.

The toolbox consists of functions, and most examples are also functions except for the file `main_sgu`. That example uses an external model file which is the function `SGU_mod`.

Feedback

All feedback is more than welcome at s.a.duineveld@outlook.com.

2. Basic Procedure

To solve and simulate a model using the `CSD` toolbox we have to take 5 steps:

1. Create the symbolic model function;
2. Set the parameters;
3. Set the steady state;
4. Solve the model, using the function `pert_ana_csd`;
5. Evaluate or simulate the model, using the function `eval_sol_csd`.

[4] Global variables are not recommended for Matlab.

We will demonstrate these steps by replicating the results of Schmitt-Grohé and Uribe (2004). The next sections describe these steps of the example code `main_sgu` and model file `SGU_mod`, both in the folder 'CSD_v02.4/Examples'.

The model, which we refer to as the SGU model, consists of three equations:

$$\begin{aligned} C_t^{-\nu} &= \beta E_t C_{t+1}^{-\nu} \left[\alpha A_{t+1} K_{t+1}^{\alpha-1} + 1 - \delta\right] \\ C_t + K_t &= A_t K_{t+1}^{\alpha} + (1-\delta) K_t \\ \log(A_{t+1}) &= \rho_a \log(A_t) + \sigma \varepsilon_{t+1} \end{aligned}$$

where C is consumption, A is Total Factor Productivity, K_t is the capital stock at the beginning of the period, and ε is a stochastic shock with a standard normal distribution.

2.1 Step 1: Symbolic model

To solve the model one has to write a symbolic model function. In our example this is the file `SGU_mod`.

Block 1: define parameters and variables

The model function first declares all the parameters and variables of the model as shown in Listing **??**. The parameters are abbreviated, with *aa* referring to α, *bb* to β, and *dd* to δ. The variables in our model are C_t, C_{t+1}, K_t, K_{t+1}, A_t and A_{t+1}. We use all variables in logs, so LC_t refers to $\log(C_t)$. Note that it is a requirement of the toolbox to use extension $\_t$ for this period variables, and $\_n$ for next period's variables. If a model includes a lagged value, for example K_{t-1}, one would have to declare an extra variable, $K1$, such that $K1\_t = K_{t-1}$, and $K1\_n = K_t$.

Listing 2.1: Model file `SGU_mod`, Blok 1

```
1 function [MOD] = SGU_mod %Schmidt-Grohe-Uribe (2004,
     JEDC) model
2
3 %% BLOCK 1: parameters and variables
4 %Parameters:
5 syms aa bb dd nu rho_a sigma_a;
6
7 %Variables:
8 syms LC_t LC_n LK_t LK_n LA_t LA_n;
```

Block 2: Model equations

Next we need to write down the model equations as shown in Listing **??**. As mentioned in Section 1.2 the stochastic process should not be included in the

model file. It is implicitly defined by the input arguments *Rho* and *Omega* of the solver `pert_ana_csd`. With these inputs the stochastic process is described by equation (1.1).

Listing 2.2: Model file `SGU_mod`, Blok 2

```
%% BLOCK 2:

%Euler equation:
f1 = bb *exp(-nu*LC_n)*...
  ( aa*exp((aa-1)*LK_n + LA_n) + (1-dd) ) -...
  exp(-nu*LC_t);

%Budget constraint:
f2 = exp(aa*LK_t + LA_t) + (1-dd)*exp(LK_t)...
  - exp(LK_n) -exp(LC_t);
```

Block 3: Assignments

The last block of the model file should assign the required properties to the structure *MOD* as explained in more detail in Chapter 5.

Note that contrary to `Dynare` the modeler has to define which variables are state variables, and which variables are control variables. The third block is shown in Listing 2.3.

Listing 2.3: Model file `SGU_mod`, blok 3

```
%% BLOCK 3: ASSIGNMENTS

%Model equations:
MOD.FS = [f1;f2];

%Endogenous state variables:
MOD.XX = [LK_t];

%Exogenous state variables:
MOD.ZZ = [LA_t];

%Control variables:
MOD.YY = LC_t;

MOD.XXn = [LK_n];
MOD.ZZn = [LA_n];
MOD.YYn = LC_n;

% Variable names (strings):
MOD.var_bs_nms  = {'LK','LA','LC'};
```

```
21
22 % Parameter names (strings):
23 MOD.par_nms      = {'aa','bb','dd','nu'};
24
25 end
```

Note that *var_bs_nms* and *par_nms* are cell arrays containing strings.

Having defined the symbolic model we turn to the file `main_sgu`, which runs the code. In that file we load the model with the call:

```
1 %% STEP 1: GET SYMBOLIC MODEL
2 [MOD]        = SGU_mod;
```

2.2 Step 2: Set parameters

Next we have to assign the numerical values to the parameters. The parameters of the model are listed in Table 2.1. We first assign all parameters values to the structure *par*. Next, we have to assign these values to the field *MOD.par_val*, which has to mimic the order of the parameter names in *MOD.par_nms*. We assign the numerical values with:

```
1 %% STEP 2: SET PARAMETERS
2
3 % Assign the numerical values:
4 MOD.par_val = [par.alpha,par.beta,par.delta,par.nu];
```

Table 2.1: Parameters of SGU model

Parameter	Value
α	0.3
β	0.95
δ	1
ν	1
ρ_a	0
σ_a	1

2.3 Step 3: Set steady state values

The steady state level where the model should be approximated is the non-stochastic or deterministic steady state. This is the steady state occurring in the absence of any shock, when the agents know no shock will ever occur.

The deterministic steady state levels of capital K and consumption C are:

$$K_{ss} = \left[\frac{1-\beta\left(1-\delta\right)}{\alpha\beta}\right]^{\frac{1}{\alpha-1}}$$
$$C_{ss} = K_{ss}^{\alpha} - \delta K_{ss}$$

We need to assign the numerical values of the steady state to the field *MOD.SS_vec*. We used all variables in logs, so the steady state values should also be in logs. The order in the vector *SS_vec* needs to be the same as in *MOD.var_bs_nms*:

```
%% STEP 3: SET STEADY STATE VALUE

MOD.SS_vec = [log(SS.Kss),log(SS.Ass),log(SS.Css)];
```

2.4 Step 4: Solve the model numerically

In Steps 1 to 3 we have assigned the symbolic model, the parameter values, and steady state values to the structure *MOD*. We can now call the perturbation solver with inputs *MOD*, *Rho*, *order* and *Omega*:

```
%% STEP 4: SOLVE MODEL NUMERICALLY

[SOL,NUM,MOD] = pert_ana_csd(MOD,par.rho_a,2,par.
    sigma_a);
```

As explained in Chapter 6 the stochastic process is determined by the input arguments *Rho* and *Omega* of `pert_ana_csd`. Note that one can also differentiate the model before assigning the numerical values. This is useful in a calibration loop and is explained in Section 3.2.

To verify the solution is the same as in Schmitt-Grohé and Uribe (2004) the fields in *SOL* should be evaluated. These fields are explained in Section 6.1.

2.5 Step 5: Simulate the model

In Step 5 we calculate a simple first order[5] Impulse Response Function (IRF), with a 1 standard deviation shock in period 3. Note that the size of the shock, σ_a, is also 1% in Schmitt-Grohé and Uribe (2004). The code to calculate the first order IRF is shown in Listing 2.4.

Listing 2.4: IRF for SHU model

[5] We use a first order IRF for simplicity as the steady state for the first order solution is the deterministic steady state. For higher order IRFs we would first have to calculate the stochastic steady state by simulating the model forward, without any shock.

```
%IRF: 1 standard deviation shock in period 3:
ini_T   = 3;%shock in period 3
TT      = 15;

LK = NaN(ini_T+TT+1,1);
LA = NaN(ini_T+TT+1,1);
LC = NaN(ini_T+TT,1);
%Starting values:
LK(1:3,1)    = log(SS.Kss);
LC(1:2,1)    = log(SS.Css);
LA(1:2,1)    = log(SS.Ass);
%Unexpected shock in period 3:
LA(3,1)      = get_ZZn(SOL,0,1);

for it = ini_T:ini_T+TT
  [LK(it+1,:),LC(it,:),LA(it+1,:)] = eval_sol_csd(SOL,
     LK(it,:),LA(it,:),0,1);
end
%Remove last row for LK and LA:
LK = LK(1:end-1,:);
LA = LA(1:end-1,:);

%Output:
LY = LA+par.alpha*LK;
```

3. Other aspects

In this Chapter we describe several aspects that were not discussed in the previous chapter. These aspects are demonstrated for the standard RBC model in the example `main_stnd_rbc_pert` in the folder 'CSD_v02.4/Examples'.

The first aspect is to differentiate the symbolic model before assigning any numerical values. This is useful in calibration procedures, because the symbolic differentiation only has to be carried out once, before assigning the numerical values of the steady state and parameters.

The second aspect that was not covered before is the use of multiple variables in one group of variables, either endogenous state variables, exogenous state variables or control variables. We use two control variables to demonstrate this. In addition we show how to use auxiliary variables in a model file. The third aspect we demonstrate is a stochastic simulation.

3.1 Standard RBC model

The model equations are:

$$C_t + K_{t+1} = Z_t K_t^{\alpha} H_t^{1-\alpha} + (1-\delta) K_t \tag{3.1}$$

$$\chi H_t^{\frac{1}{\eta}} = C_t^{-\nu} Z_t (1-\alpha) K_t^{\alpha} H_t^{-\alpha} \tag{3.2}$$

$$C_t^{-\nu} = \beta E_t \left\{ C_{t+1}^{-\nu} \left[Z_{t+1} \alpha K_{t+1}^{\alpha-1} H_{t+1}^{1-\alpha} + 1 - \delta \right] \right\} \tag{3.3}$$

$$\log(Z_t) = \rho_z \log(Z_{t-1}) + \sigma_z \epsilon_t \tag{3.4}$$

where C is consumption, K_t the capital stock at the beginning of the period, Z Total Factor Productivity, and H hours worked.

The first two blocks of the model function are shown in Listing 3.1. We use output LY as an auxiliary variable. This variable does not have to be declared, and is not part of the solution. We used hours worked LH as a control variable, which is defined by static equation $f3$. This equation only includes period t variables.

Listing 3.1: Model file `RBC_mod`, Blok 1 and 2

```
%% Perturbation model of Standard RBC model function
[MOD] = STND_RBC_mod()

%% BLOCK 1: Define parameters and variables
%Parameters:
syms aa bb dd ee nu ch;

%Variables:
syms LC_t LC_n LK_t LK_n LZ_t LZ_n LH_t LH_n;


%% BLOCK 2: MODEL EQUATIONS (excl. stochastic process)

%Auxiliary variables:
LY_t       = LZ_t + aa*LK_t + (1-aa)*LH_t;
LY_n       = LZ_n + aa*LK_n + (1-aa)*LH_n;

%Euler equation:
f1 = bb *exp(-nu*LC_n)*( aa*exp(LY_n-LK_n) + (1-dd) )
   - exp(-nu*LC_t);

% Capital accumulation:
f2 = exp(LY_t) + (1-dd)*exp(LK_t) - exp(LC_t) - exp(
   LK_n);

% Labour market:
```

```
f3 = log(1-aa) + LZ_t + aa*(LK_t-LH_t) + -nu*LC_t -
    log(ch) - 1/ee*LH_t;
```

The third block of the model function is shown in Listing 3.2. Note that the model equations need to be stacked in a column vector, using ';' as separating symbol, while variables need to stacked in row vectors using ',' as a separating symbol.

Listing 3.2: Model file `RBC_mod`, blok 3

```
%% BLOCK 3: ASSIGNMENTS

%Model equations:
MOD.FS = [f1;f2;f3];

%Endogenous state variables:
MOD.XX = LK_t;
%Exogenous state variables:
MOD.ZZ = LZ_t;
%Control variables:
MOD.YY = [LC_t,LH_t];

MOD.XXn = LK_n;
MOD.ZZn = LZ_n;
MOD.YYn = [LC_n,LH_n];

% Variable names (strings):
MOD.var_bs_nms  = {'LK','LZ','LC','LH'};

% Parameter names (strings):
MOD.par_nms     = {'aa','bb','dd','ee','nu','ch'};
```

3.2 Symbolic differentiation

We have defined the model in the previous section. We can differentiate this model using the function `get_deriv_csd`, before assigning any numerical values:

```
%% STEP 1: GET SYMBOLIC MODEL & DIFFERENTIATE
MOD          = STND_RBC_mod;

% Differentiate symbolic model:
par.opt.order = 3;
[MOD] = get_deriv_csd(MOD,par.opt.order);
```

After differentiating the model we assign the parameters and steady state values in Step 2 and Step 3, as in Chapter 2. When we solve the model we

set the optional input $excl\_der = 1$ such that the solver does not calculate the derivatives again:

```
%% STEP 4: SOLVE MODEL NUMERICALLY

%exclude calculation of derivates (see STEP 1)
excl_der = 1;

% Solve model numerically:
SOL  = pert_ana_csd(MOD,par.rho_z,par.opt.order,par.
   sigma_z,excl_der);
```

3.3 Simulation

To simulate the model we use the function `eval_sol_csd`. This takes the period t state variables as input. We use the function `stnd_rbc_sim_pert` to simulate the model. The crucial part of the simulation is:

```
% loop over time (from T_ini +1 till T_ini + TT):

for it = T_ini + 1: T_ini + TT

  % Get XXn, YY, ZZn using eval_sol_csd:
  [LK(it+1,:),YY_t,LZ(it+1,:)] = eval_sol_csd(SOL,LK(
     it,:),LZ(it,:),epsilon(it+1,:),order);

  %Cons. is first control variable:
  LC(it,:) = YY_t(1,:);

  %Hours is second control variable
  LH(it,:) = YY_t(2,:);
end
```

Note how the control variables are stacked vertically in the output YY_t. Similarly if there are two endogenous state variables they have to be stacked vertically in the input XX, and will be stacked in the same way in the output XXn of `eval_sol_csd`. This is shown in the undocumented example `main_housing_pert`, in the function `housing_sim_pert`.

3.4 Non-stochastic models

To solve models without any exogenous shocks the following aspects should be taken care of:

1. In the symbolic model of Step 3 set the fields $MOD.ZZn$ and $MOD.ZZ$ to empty:

```
MOD.ZZn = [];
```

and

```
MOD.ZZ = [];
```

2. When solving the model in Step 5 set inputs fields *Rho* and *Omega* of the function `pert_ana_csd` to empty ('[]'):

```
[SOL] = pert_ana_csd(MOD,[],order,[]);
```

3. When evaluating the model with `eval_sol_csd` do not use the third output argument[6]:

```
[XXn,YY] = eval_sol_csd(SOL,XX,ZZ,epsilon,order);
```

4. Saddle cycle example

In this chapter describe and replicate the BGPe model by Galizia (forthcoming). The program that solves this model is `main_repl_galizia`[7]. To solve saddle cycle models we call `pert_ana_csd_lim` instead of the usual `pert_ana_csd`. The output of `pert_ana_csd_lim` is similar, except when multiple candidate solutions exists. If multiple candidate solutions exists the output *SOL* will be a cell array with each cell containing a structure with the same fields as the standard solution of `pert_ana_csd`.

4.1 Model summary

$$\begin{aligned}
\mu_t \lambda_t &= Q\left(e_t\right) E_t\left[e_{t+1}^{\varphi_e} \lambda_{t+1}\right] \\
\lambda_t &= \left(Y_{t+1} + \frac{1-\delta-\gamma}{1-\delta} X_t - \frac{1-\delta-\psi}{1-\delta}\gamma Y_t\right)^{-\omega} \\
X_{t+1} &= (1-\delta) X_t + \psi Y_{t+1} \\
Y_{t+1} &= z_t e_t^{\alpha}
\end{aligned}$$

With μ and z AR(1) processes:

[6] One can specify the output argument *ZZn* but it will be assigned the value empty ('[]').

[7] We replicate the results almost exactly. The difference is that we use logs in both the model and the simulation, while Galizia uses logs to solve the model, but uses percentage deviations for the simulation. This results in small differences in the simulation.

Table 4.1: Parameters

Symbol	Code	z shocks	μ shocks
δ	del	0.05	0.05
α	al	0.67	0.67
$\overline{e}$	e	0.9455	0.9430
ω	om	0.2596	0.2736
γ	gam	0.6489	0.6259
ψ	psi	0.3929	0.3905
φ_e	phie	0.0454	0.0460
ϕ	phi0	0.9871	0.9108
$\overline{\Phi}$	Phi0	0.0430	0.0470
Φ_2	Phi2	2.6092	1.7103
Φ_3	Phi3	325.8463	186.8312
ρ_z	rhoz	0.6254	-
σ_z	sigz	0.0027	-
ρ_μ	rhomu	-	0.0671
σ_μ	sigmu	-	1.3577e-4

$$\log(\mu_t) = \rho_\mu \log(\mu_{t-1}) + \sigma_\mu \varepsilon_{\mu,t}$$
$$\log(z_t) = \rho_z \log(z_{t-1}) + \sigma_z \varepsilon_{z,t}$$

Furthermore,

$$Q(e_t) \equiv \Theta\left[1 + (1 - e_t)\,\phi\Phi(e_t)\right]$$
$$\Phi(e_t) = \overline{\Phi} \exp\left\{\frac{1}{2}\frac{\Phi_2}{\overline{\Phi}}(e_t - \overline{e})^2 + \frac{1}{6}\frac{\Phi_3}{\overline{\Phi}}(e_t - \overline{e})^3\right\}$$

with $\Theta = \frac{\overline{e}^{-\varphi_e}}{1+(1-\overline{e})\phi\overline{\Phi}}$.

Following Galizia, instead of using e_t as policy variable we use the transformation etr_t:

$$e_t = \frac{1}{1 + \exp(-\text{etr}_t)}$$

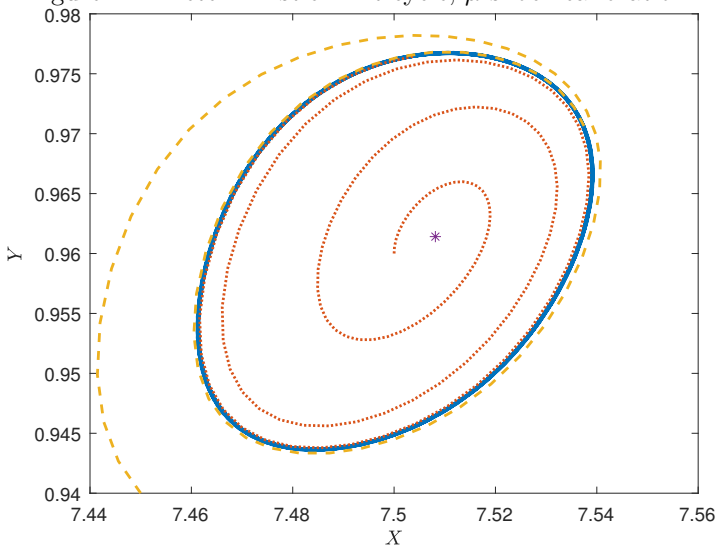
4.2 Calibration

The parameters are listed in Table 4.2.

4.3 Steady state

The steady state is determined from the parameters including $\overline{e}$:

Figure 4.1: Deterministic limit cycle, μ shock calibration

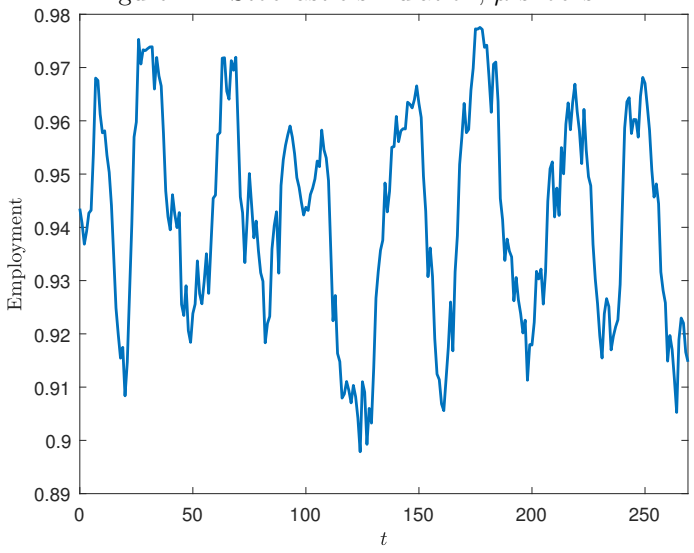


$$\overline{Y} = \overline{z}\overline{e}^{\alpha}$$
$$\overline{X} = \frac{1}{\delta}\psi\overline{Y}$$
$$\overline{\lambda} = \left(\frac{1-\delta-\gamma}{1-\delta}\overline{X} + \left(1 - \frac{1-\delta-\psi}{1-\delta}\gamma\right)\overline{Y}\right)^{-\omega}$$

4.4 Results

The results are plotted in Figure 4.1 and Figure 4.2.

Figure 4.2: Stochastic simulation, μ shocks



Part II

Technical descriptions

5. Symbolic model function

The symbolic model should be a function which assigns several fields to the structure MOD. The model should consists of three blocks:

1. Define parameters and variables;
2. Model equations;
3. Assignment of fields.

Each block is best understood from the examples in Chapter 2 and Chapter 3.

The third block should assign the following fields to the output MOD structure:

- FS: column vector of all relevant equations, excluding the stochastic process. The stochastic process is implicitly defined through the input arguments Rho and $Omega$ of the function `pert_ana_csd` as explained in Chapter 6;
- XX: all period t endogenous state variables, including the extension $\_t$ ($1 \times nx$ vector);
- ZZ: list all period t exogenous state variables, including the extension $\_t$ ($1 \times nz$ vector);
- YY: list all period t control or jump variables variables, including the extension $\_t$ ($1 \times ny$ vector);
- XXn: list all period $t+1$ endogenous state variables, including the extension $\_n$, in the same order as XX ($1 \times nx$ vector);
- ZZn: list all period $t+1$ exogenous state variables, including the extension $\_n$, and in the same order as ZZ ($1 \times nz$ vector);
- YYn: list all period $t+1$ control or jump variables variables, including the extension $\_n$, and in the same order as YY ($1 \times ny$ vector);
- var_bs_nms: cell array of strings with the names of all variables, without the extension $\_t$ or $\_n$. The cell array has size $1 \times (nx + ny + nz)$. Each cell has to contain the name of a variable occurring in XX, ZZ, and YY. For example if there is a variable LC_t in YY and LC_n in YYn, then $MOD.var\_bs\_nms$ should contain a cell with the name `'LC'`. The order in which the variables occur can be random, but the steady state values assigned later to $MOD.SS\_vec$ have to be in the same order (see Section 5.1);

- *par_nms*: cell array of strings with the names of the parameters as they occur in the model function. The order can be random, but the numerical values assigned later to the field *MOD.par_val* have to be in the same order (see Section 5.1).

In addition there is the requirement that:

- All period t variables should have the extension $\_t$ and all period $t+1$ variables should have extension $\_n$.

5.1 Assign numerical values

After assigning the symbolic model to the structure *MOD* the numerical values of the steady state and the parameters have to be set in order to solve the model. These values should be assigned to the fields:

- *MOD.SS_vec*: numerical values of the deterministic or non-stochastic steady state, in a $1 \times (nx + ny + nz)$ vector. The order has to be the same as in *MOD.var_bs_nms* (see Chapter 5);
- *MOD.par_val*: numerical values of the parameters in a row vector. The order has to be the same as in *MOD.par_nms* (see Chapter 5).

6. Function `pert_ana_csd`

The function `pert_ana_csd` solves the model and takes the following inputs:

- *MOD*: structure with the required fields discussed in Chapter 5;
- *Rho*: $nz \times nz$ autocorrelation matrix of the exogenous variables as demonstrated in equation (6.1);
- *order*: the order of the perturbation solution. Note that higher order solutions do not affect lower order coefficients. When solving the third order solution the function `pert_ana_csd` will assign the first, second and third order coefficients in separate matrices. This allows one to solve the model once using the third order approximation, and then simulate either first, second or third order solutions.
- *Omega*: $nz \times nz$ variance-covariance matrix of the exogenous shocks as demonstrated in equation (6.1);
- *excl_der* (optional): set this input to *excl_der* = 1 when the structure *MOD* already includes the symbolic derivatives as explained in Section 6.2. This will omit the call to `get_deriv_csd` inside `pert_ana_csd`. This option can save computation time when calibrating a model.

The exogenous process is defined as:

$$Z_{t+1} = \rho Z_t + \Omega \varepsilon_{t+1} \tag{6.1}$$

where Z is an vector of length nz, and both ρ and Ω are $nz \times nz$ matrices. The symbol ρ is the input argument *Rho*, and Ω is the input argument *Omega*.

The function `pert_ana_csd` has the following output arguments:

- *SOL*: a structure that contains the solution in various fields. These are briefly described in Section 6.1;
- *NUM*: a structure that contains the numerical values of first, second and third order derivatives of the model (in the fields *DD*, *TT*, and *HH*). This structure also contains the matrices *AA* and *BB* used as input arguments into the function `solab_adj_stab`. This function is a slightly modified version of `solab` developed by Paul Klein to obtain the first order perturbation solution;
- *MOD*: the structure with all the fields described in Chapter 5 plus the symbolic derivatives;
- *stab*: a scalar which is 1 if the model is saddle path stable, and either -1 or 0 if it is locally explosive or indeterminate, respectively;

6.1 Solution *SOL*

The solver assigns various fields to the output *SOL*. The policy functions all start with H, following by either x or y, referring to the endogenous state variables or the control variables, respectively. The policies H_x together determine next period's endogenous state variables, while the policies H_y together determine period t control variables.

The policy function are all calculated using $\tilde{W}_t$, which we call *W_dev* in our code. $\tilde{W}_t$ is an $(nx + nz) \times cols$ matrix of the state variables in period t, in deviation of their steady state values. Each row is a state variable, and each column reflects a datapoint. The state variables are ordered vertically as $[XX; ZZ]$. The order within XX and ZZ is defined by the Assignment Block in the model function as explained in Chapter 5.

The first order policy function for the endogenous variables is:

$$X^1_{t+1} = \overline{X} + H_{x.w}\tilde{W}_t \tag{6.2}$$

where the superscript 1 refers to the order, $\overline{X}$ is $nx \times 1$ vector, $H_{x.w}$ is $nx \times (nx + nz)$ matrix, and $\tilde{W}$ is $(nx + nz) \times cols$ matrix with data points. The output X_{t+1} is an $nx \times cols$ matrix of the endogenous state variables.
The second order policy function for the endogenous variables is:

$$X_{t+1}^2 = X_{t+1}^1 + \frac{1}{2}H_{x,ss} + \frac{1}{2}\left(I_{nx} \otimes \tilde{W}_t^{\intercal}\right) H_{x,ww}\tilde{W}_t$$

The third order policy function for the endogenous variables is:

$$\begin{aligned} X_{t+1}^3 = X_{t+1}^2 &+ \frac{1}{2}\left(I_{nx} \otimes \tilde{W}_t^{\intercal}\right) H_{x,ssw} \\ &+ \frac{1}{6}\left(I_{nx} \otimes \tilde{W}_t^{\intercal} \otimes \tilde{W}_t^{\intercal}\right) H_{x,ww}\tilde{W}_t + \frac{1}{6}H_{x,sss} \end{aligned} \tag{6.3}$$

The policy functions for the control variables have exactly the same format:

$$Y_t^1 = \overline{Y} + H_{y.w}\tilde{W}_t$$

$$Y_t^2 = Y_t^1 + \frac{1}{2}H_{y,ss} + \frac{1}{2}\left(I_{nx} \otimes \tilde{W}_t^{\intercal}\right) H_{y,ww}\tilde{W}_t$$

$$Y_t^3 = Y_t^2 + \frac{1}{2}\left(I_{nx} \otimes \tilde{W}_t^{\intercal}\right) H_{y,ssw} \tag{6.4}$$

$$+ \frac{1}{6}\left(I_{nx} \otimes \tilde{W}_t^{\intercal} \otimes \tilde{W}_t^{\intercal}\right) H_{y,www}\tilde{W}_t + \frac{1}{6}H_{y,sss} \tag{6.5}$$

The first order terms are:

- Hx_w: first order coefficients for next period's endogenous variables. The coefficients are ordered as $[XX, ZZ]$. Dimensions: $nx \times (nx + nz)$ matrix. Corresponds to $h_x\,(1,:)$ in Schmitt-Grohé and Uribe (2004);
- Hy_w: first order coefficients for period t control variables. The coefficients are ordered as $[XX, ZZ]$. Dimensions: $nv \times (nx + nz)$ matrix. Corresponds to g_x in Schmitt-Grohé and Uribe (2004);

The second order terms are:

- Hx_ww: second order coefficients for next period's endogenous variables. Corresponds to $[h_{xx}\,(1,:,1)\,; h_{xx}\,(2,:,2)]$ in Schmitt-Grohé and Uribe (2004);
- Hx_ss: the second order correction term for the endogenous state variables. Corresponds to $h_{\sigma\sigma}\,(1,1)$ in Schmitt-Grohé and Uribe (2004);
- Hy_ww: second order coefficients for period t control variables. Corresponds to $[g_{xx}\,(:,:,1)\,; g_{xx}\,(:,:,2)]$ in Schmitt-Grohé and Uribe (2004);
- Hy_ss: the second order correction term for the control variables. Corresponds to $g_{\sigma\sigma}$ in Schmitt-Grohé and Uribe (2004);

The third order terms are best understood from the policy functions (6.3) and (6.4). The third order terms are:

- Hx_www;
- Hx_ssw;
- Hx_sss.

6.2 Function `get_deriv_csd`

The function `get_deriv_csd` constructs the analytical derivatives of the model. It takes as inputs:

- MOD: the structure with the symbolic model with the properties explained in Chapter 5;
- $order$: the order of the derivatives, which is either 1, 2 or 3.

The function assigns the following fields to the structure MOD:

- nx, nz, ny: the number of endogenous state variables, the number of exogenous state variables, and the number of control variables, respectively;
- qq: array with the symbolic variables, dimension $1 \times 2\,(nx + nz + ny)$;
- DS, HS, TS: the first, second and third order derivatives, respectively, of the model equations;
- ord: the order of the symbolic derivatives, set in the input field $order$.

7. Function `eval_sol_csd`

To evaluate the policy function for given state variables we use `eval_sol_csd`. This function calculates the endogenous variables, the control variables and the exogenous state variables. The policy functions are described in Section 6.1[8]. The input and output matrices can contain multiple time series. Each series is represented by a column, with the total number of series being $cols$. The function takes as input:

- SOL: the structure with the policy functions (see Chapter 6);
- XX: the endogenous state variables in period t ($nx \times cols$ matrix). The vertical order of the endogenous state variables should be the same as the horizontal order in $MOD.XX$ (see Chapter 5);

[8]The policy functions are calculated using vectorization, such that Matlab can evaluate multiple datapoints simultaneously.

- ZZ: the exogenous state variables in period t ($nz \times cols$ matrix). The vertical order of the exogenous state variables should be the same as the horizontal order in $MOD.ZZ$ (see Chapter 5);
- *epsilon*: shocks in period t ($nz \times cols$ matrix). The vertical order should match the order of ZZ;
- *order*: the order of the solution to evaluate the policy function. The order for the evaluation should not be higher than the order of the solution;

The output of the function is:

- XXn: the endogenous state variables in period $t+1$ ($nx \times cols$ matrix). The order of the endogenous state variables is the same as in $MOD.XX$ (see Chapter 5);
- YY: the control variables in period t ($ny \times cols$ matrix). The order of the control variables is the same as in $MOD.YY$ (see Chapter 5);
- ZZn: the exogenous state variables in period $t+1$ ($nz \times cols$ matrix). The order of the exogenous state variables is the same as in $MOD.ZZ$ (see Chapter 5).

Bibliography

Beaudry, Paul, Dana Galizia, and Franck Portier (2020). "Putting the cycle back into business cycle analysis". In: *American Economic Review* 110.1, pp. 1–47.

Galizia, Dana (forthcoming). "Saddle Cycles: Solving Rational Expectations Models Featuring Limit Cycles (or Chaos) Using Perturbation Methods". In: *Quantitative Economics*.

Schmitt-Grohé, Stephanie and Martın Uribe (2004). "Solving dynamic general equilibrium models using a second-order approximation to the policy function". In: *Journal of economic dynamics and control* 28.4, pp. 755–775.