A Toolbox for SMM Estimation of DSGE Models Approximated by Extended Perturbation up to Fifth Order

Martin M. Andreasen
Aarhus University, the Danish Finance Institute, and CREATES

December 14, 2021

Contents

1 Introduction

The purpose of this note is to document a Matlab toolbox for estimating dynamic stochastic general equilibrium (DSGE) models by the Simulated method of moments (SMM) using the extended perturbation approximation up to fifth order. We illustrate the use of the toolbox by estimating on simulated data a stylized RBC model approximated up to fifth order. The toolbox also enables the user to evaluate the accuracy of the considered approximation by computing the Euler equation errors on either a simulated sample path or a grid.

The note is structured as follows. Section 2 presents the stylized RBC model. The required knowledge about SMM estimation to understand and use our toolbox is provided in Section 3. We formally present our toolbox in Section 4 by describing the steps required to carry out a SMM estimation of the RBC model considered.

It is worth noticing that the toolbox is currently only coded up for linear shocks when using the extended perturbation approximation. However, with an appropriate extension of the state variables, it is possible to accommodate nonlinear features such as stochastic volatility, GARCH, etc. as shown in Andreasen (2012). Note also that in this implementation, both lagged control variables and exogenous states are concentrated out of the fixed-point problem for the Extended Path. This ensures consistency between the states and the controls and it should also reduce the computational burden.

2 A Stylized RBC Model

This section presents a stylized RBC model similar to the one considered in King & Rebelo (1999).

2.1 Households

An infinitely lived representative household derives utility from consumption C_t and leisure L_t. The objective of this household is to maximize the utility function

$$\max \quad \sum_{j=0}^{\infty} \beta^j d_{t+j} E_t \left[\frac{(C_{t+j} - bC_{t-1+j})^{1-\eta_c} - 1}{1-\eta_c} + \theta \frac{L_{t+j}^{1-\eta_l} - 1}{1-\eta_l} \right],$$

where E_t is the conditional expectation operator. This utility function is described by the structural parameters $\{\beta, b, \eta_c, \eta_l, \theta\}$ where $\beta < 1$ is a discount factor. The variable d_t is a preference shock which evolves as

$$\log d_{t+1} = \rho_d \log d_t + \sigma_d \epsilon_{d,t+1}$$

with $\epsilon_{d,t+1} \sim \mathcal{NID}(0,1)$ and $|\rho_d| < 1$. The parameter $b \in [0,1]$ allows for internal consumption habits if $b > 0$. The household's time endowment is normalized to one and is allocated to leisure and hours worked N_t, i.e.

$$N_t + L_t = 1.$$

The capital stock K_t is assumed to be owned by the household who also conduct investments I_t. The law of motion for capital is given by

$$K_{t+1} = (1-\delta) K_t + I_t,$$

where K_t is the capital level at the beginning of period t and δ controlling depreciations.[1]

To derive the budget constraint for the household, let W_t denote the real wage in period t and let R_t^k refer to the real rental rate of capital in period t. The representative household also receives real dividends Π_t from firms, implying that its total real income in period t is given by

$$\text{income}_t = W_t N_t + R_t^k K_t + \Pi_t.$$

This income is used for either consumption or investments, meaning that the budget constraint reads

$$C_t + I_t \leq W_t N_t + R_t^k K_t + \Pi_t.$$

Thus the optimization problem faced by the representative household is

$$\max_{(C_{t+j}, N_{t+j}, K_{t+1+j}, I_{t+j})_{j=0}^{\infty}} \quad \sum_{j=0}^{\infty} \beta^j d_{t+j} E_t \left[\frac{(C_{t+j} - bC_{t-1+j})^{1-\eta_c} - 1}{1-\eta_c} + \theta \frac{(1-N_{t+j})^{1-\eta_l} - 1}{1-\eta_l} \right]$$

subject to

$$C_t + I_t \leq W_t N_t + R_t^k K_t + \Pi_t$$

$$K_{t+1} = (1-\delta) K_t + I_t$$

(and a no-Ponzi-game condition which we abstract from). The solution is a set of contingency plans for C_t, N_t, K_{t+1}, and I_t as a function of the state variables. It is easy to show that the optimization problem implies the following first-order conditions:

$$\frac{\partial \mathcal{L}}{\partial C_t} : d_t (C_t - bC_{t-1})^{-\eta_c} - \beta b E_t \left[d_{t+1} (C_{t+1} - bC_t)^{-\eta_c} \right] = \lambda_t$$

$$\frac{\partial \mathcal{L}}{\partial N_t} : \theta d_t (1 - N_t)^{-\eta_l} = \lambda_t W_t$$

$$\frac{\partial \mathcal{L}}{\partial K_{t+1}} : \lambda_t = E_t \left[\beta \lambda_{t+1} \left(R_{t+1}^k + (1-\delta) \right) \right]$$

where λ_t is the lagrange multiplier on the budget constraint.

[1] Note that capital is predetermined in period t (i.e. it was set in period $t-1$) and some therefore find it more natural to use the notation $K_t = (1-\delta) K_{t-1} + I_t$, where K_t is the capital level at the end of period t. This alternative notation is used in Dynare and may also be adopted in this toolbox.

2.2 Firms

Production Y_t is carried out using a Cobb-Douglas production function

$$Y_t = A_t K_t^{1-\alpha} N_t^{\alpha},$$

where A_t refers to productivity shocks that evolves as

$$\log A_{t+1} = \rho_A \log A_t + \sigma_A \epsilon_{A,t+1}$$

with $\epsilon_{A,t+1} \sim \mathcal{NID}(0,1)$ and $|\rho_A| < 1$. Firm profit is given by

$$\Pi_t = A_t K_t^{1-\alpha} N_t^{\alpha} - W_t N_t - R_t^k K_t,$$

and when optimized across capital and hours worked we get the standard first-order conditions:

$$\frac{\partial \Pi_t}{\partial K_t} : A_t (1-\alpha) K_t^{-\alpha} N_t^{\alpha} = R_t^k$$

$$\frac{\partial \Pi_t}{\partial N_t} : A_t \alpha K_t^{1-\alpha} N_t^{\alpha-1} = W_t.$$

2.3 Market clearing and Model Summary

Equilibrium in the goods market implies $C_t + I_t = Y_t$, which is satisfied via the budget constraint because $\Pi_t = 0$ and $W_t N_t + R_t^k K_t = Y_t$. Hence, our economy can be condensely expressed as:

1	$d_t (C_t - bC_{t-1})^{-\eta_c} - \beta b E_t \left[d_{t+1} (C_{t+1} - bC_t)^{-\eta_c} \right] = \lambda_t$
2	$\theta d_t (1 - N_t)^{-\eta_l} = \lambda_t W_t$
3	$\lambda_t = E_t \left[\beta \lambda_{t+1} \left(R_{t+1}^k + (1-\delta) \right) \right]$
4	$A_t (1-\alpha) K_t^{-\alpha} N_t^{\alpha} = R_t^k$
5	$A_t \alpha K_t^{1-\alpha} N_t^{\alpha-1} = W_t$
6	$C_t + I_t = Y_t$
7	$Y_t = A_t K_t^{1-\alpha} N_t^{\alpha}$
8	$K_{t+1} = (1-\delta) K_t + I_t$
9	$(C_{t-1})_{t+1} = C_t$
10	$\log A_{t+1} = \rho_A \log A_t + \sigma_A \epsilon_{A,t+1}$
11	$\log d_{t+1} = \rho_d \log d_t + \sigma_d \epsilon_{d,t+1}$

This economy has four state variables (K_t, C_{t-1}, A_t, d_t) and seven control variables $\left(C_t, \lambda_t, W_t, R_t^k, N_t, I_t, Y_t\right)$. Note that we in Eq 9 introduce a so-called link equation, that relates lagged consumption next period to the present consumption, i.e. $(C_{t-1})_{t+1} = C_t$.

2.4 The Steady State

The steady state solution is defined as the point where there are no shocks hitting the economy and all dynamics have played out, implying that we can replace all time-subscript by ss and solve for all variables as a function of the structural parameters. From Eq 10 and Eq 11 we have $A_{ss} = 1$ and $d_{ss} = 1$. Eq 3 implies $\lambda_{ss} = \beta\lambda_{ss}\left(R^k_{ss} + (1-\delta)\right)$, and therefore

$$R^k_{ss} = \frac{1}{\beta} - (1-\delta).$$

From Eq 4 we have $A_{ss}(1-\alpha)K^{-\alpha}_{ss}N^{\alpha}_{ss} = R^k_{ss}$, which is equivalent to

$$\frac{K_{ss}}{N_{ss}} = \left(\frac{R^k_{ss}}{A_{ss}(1-\alpha)}\right)^{-1/\alpha}.$$

From Eq 5 we directly get

$$W_{ss} = A_{ss}\alpha\left(\frac{K_{ss}}{N_{ss}}\right)^{1-\alpha}.$$

The law of motion for capital in Eq 8 implies

$$\frac{I_{ss}}{N_{ss}} = \delta\frac{K_{ss}}{N_{ss}}.$$

The production function in Eq 7 gives

$$\frac{Y_{ss}}{N_{ss}} = A_{ss}\left(\frac{K_{ss}}{N_{ss}}\right)^{1-\alpha}.$$

From Eq 6 we have

$$\frac{C_{ss}}{N_{ss}} = \frac{Y_{ss}}{N_{ss}} - \frac{I_{ss}}{N_{ss}}.$$

Combining Eq 1 and 2, we have

$$\theta(1-N_{ss})^{-\eta_l}\frac{1}{N^{-\eta_c}_{ss}} = \frac{(1-\beta b)(C_{ss}-bC_{ss})^{-\eta_c}}{N^{-\eta_c}_{ss}}W_{ss}$$

$\Updownarrow$

$$\theta(1-N_{ss})^{-\eta_l}N^{\eta_c}_{ss} = \left(\frac{C_{ss}}{N_{ss}}\right)^{-\eta_c}(1-\beta b)(1-b)^{-\eta_c}W_{ss}.$$

We can assume that N_{ss} is known (for instance $N_{ss} = 1/3$) and solve for the value of θ. Alternatively, and as done in the current implementation, condition on a value of θ and solve for N_{ss} by a numerical fixed-point algorithm. If $\eta^l = \eta^c = 1$ (i.e. log-log preferences) then we have the following closed-form solution

$$\theta\frac{N_{ss}}{1-N_{ss}} = \left(\frac{C_{ss}}{N_{ss}}\right)^{-1}(1-\beta b)(1-b)^{-1}W_{ss}$$

$$\Updownarrow$$

$$N_{ss} = (1 - N_{ss}) \left(\frac{C_{ss}}{N_{ss}} \right)^{-1} \frac{(1 - \beta b) W_{ss}}{\theta (1 - b)}$$

$$\Updownarrow$$

$$N_{ss} = \frac{\left(\frac{C_{ss}}{N_{ss}} \right)^{-1} \frac{(1-\beta b) W_{ss}}{\theta (1-b)}}{1 + \left(\frac{C_{ss}}{N_{ss}} \right)^{-1} \frac{(1-\beta b) W_{ss}}{\theta (1-b)}}.$$

Given a value of N_{ss}, we trivially have all the desired quantities using the previous expressions for $\frac{C_{ss}}{N_{ss}}$, $\frac{I_{ss}}{N_{ss}}$, $\frac{K_{ss}}{N_{ss}}$, and $\frac{Y_{ss}}{N_{ss}}$.

3 Theory on SMM Estimation

This section describes how our RBC model can be estimated by SMM. We proceed by introducing the SMM estimator in Section 3.1. The asymptotic distribution of this estimator is given in Section 3.2, and we finally discuss the considered moments for our SMM estimator in Section 3.3. For additional introduction to SMM, see for instance Ruge-Murcia (2012).

3.1 The SMM estimator

To present the SMM estimator, let $\mathbf{y}_t$ with dimension $n_y \times 1$ contain our observed variables in period t, with the entire sample denoted by $\mathbf{y}_{1:T} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_T\}$. The structural parameters in our DSGE model is denoted by $\boldsymbol{\theta}$ with dimension $n_\theta \times 1$. The objective in SMM is to estimate $\boldsymbol{\theta}$ using a set of moment conditions. We focus on first and second unconditional moments as typically done when estimating DSGE models. More formally, consider n_m unconditional moments $E^S[\mathbf{m}(\boldsymbol{\theta})] = \frac{1}{\tau T} \sum_{s=1}^{\tau T} \mathbf{m}(\boldsymbol{\theta}, \mathbf{y}_s)$ from our DSGE model, which we compute based on a simulated sample path of τT observations. Here, T is the length of our empirical sample and τ is an integer controlling the length of the simulated sample path. We next define the function $\mathbf{h}^S(\boldsymbol{\theta}, \mathbf{y}_t) \equiv \mathbf{m}(\mathbf{y}_t) - E^S[\mathbf{m}(\boldsymbol{\theta})]$. Let $\boldsymbol{\theta}_o$ denote the true value of $\boldsymbol{\theta}$, which implies $E^S[\mathbf{h}(\boldsymbol{\theta}_o, \mathbf{y}_t)] = \mathbf{0}$. The estimation applies the sample analogue

$$\mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T}) = \frac{1}{T} \sum_{t=1}^{T} \mathbf{h}^S(\boldsymbol{\theta}, \mathbf{y}_t)$$

as a substitute for the expectation operator, and the SMM estimator is then given by (see Duffie & Singleton (1993))

$$\boldsymbol{\theta}_{SMM} = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\arg\min} \ \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T})' \mathbf{W}_T \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T}). \tag{1}$$

Here, $\mathbf{W}_T$ is an $n_m \times n_m$ positive definite weighting matrix that may depend on $\mathbf{y}_{1:T}$. Note that $n_m \geq n_\theta$ is a necessary but not a sufficient condition for identification of $\boldsymbol{\theta}$. A weighting matrix $\mathbf{W}_T$ is thus required to make SMM operational. One commonly used approach is to determine $\mathbf{W}_T$ from

$\mathbf{y}_{1:T}$. This can be done by using the Newey-West estimator, i.e.

$$\hat{\mathbf{S}} = \hat{\Gamma}_0 + \sum_{\nu=1}^{q} \left(1 - \frac{\nu}{q+1}\right) \left(\hat{\Gamma}_\nu + \hat{\Gamma}'_\nu\right)$$

where

$$\begin{aligned} \hat{\Gamma}_\nu &= \frac{1}{T} \sum_{t=1}^{T} \mathbf{h}^S \left(\hat{\boldsymbol{\theta}}^{(1)}, \mathbf{y}_t\right) \mathbf{h}^S \left(\hat{\boldsymbol{\theta}}^{(1)}, \mathbf{y}_{t-\nu}\right)' \\ &= \frac{1}{T} \sum_{t=1}^{T} \left(\mathbf{m}\left(\mathbf{y}_t\right) - E^S\left[\mathbf{m}\left(\hat{\boldsymbol{\theta}}^{(1)}\right)\right]\right) \left(\mathbf{m}\left(\mathbf{y}_{t-\nu}\right) - E^S\left[\mathbf{m}\left(\hat{\boldsymbol{\theta}}^{(1)}\right)\right]\right)'. \end{aligned} \tag{2}$$

Here, $\hat{\boldsymbol{\theta}}^{(1)}$ denotes a preliminary first step estimate of $\boldsymbol{\theta}$ using (1) with $\mathbf{W}_T = \mathbf{I}$, whereas the optimal weighting matrix is given by $\hat{\mathbf{S}}^{-1}$. Instead of using $\mathbf{W}_T = \mathbf{I}$ in the first step, another possibility is to observe that for the true value of $\boldsymbol{\theta}$ we have $E\left[\mathbf{g}^S\left(\boldsymbol{\theta}_0, \mathbf{y}_{1:T}\right)\right] = \mathbf{0}$, implying

$$E\left[\frac{1}{T}\sum_{t=1}^{T} \mathbf{m}\left(\mathbf{y}_t\right)\right] = E^S\left[\mathbf{m}\left(\boldsymbol{\theta}_o\right)\right].$$

Hence, if we let $\mathbf{m}_T \equiv \frac{1}{T}\sum_{t=1}^{T} \mathbf{m}\left(\mathbf{y}_t\right)$, we can use the sample mean of the moments to estimate $E^S\left[\mathbf{m}\left(\boldsymbol{\theta}_o\right)\right]$ instead of $E\left[\mathbf{m}\left(\hat{\boldsymbol{\theta}}^{(1)}\right)\right]$. This leads to the alternative estimator

$$\hat{\Gamma}_{\nu,mean} = \frac{1}{T}\sum_{t=1}^{T} \left(\mathbf{m}\left(\mathbf{y}_t\right) - \mathbf{m}_T\right)\left(\mathbf{m}\left(\mathbf{y}_t\right) - \mathbf{m}_T\right)',$$

which does not require a preliminary estimate of $\boldsymbol{\theta}$. Hence, the weighting matrix may be obtain using

$$\hat{\mathbf{S}}_{mean} = \hat{\Gamma}_{0,mean} + \sum_{\nu=1}^{q} \left(1 - \frac{\nu}{q+1}\right) \left(\hat{\Gamma}_{\nu,mean} + \hat{\Gamma}'_{\nu,mean}\right).$$

Given $\hat{\mathbf{S}}_{mean}$, we may either let $\mathbf{W}_T = \left(\hat{\mathbf{S}}_{mean}\right)^{-1}$ or only use the diagonal elements of $\hat{\mathbf{S}}_{mean}$ and let $\mathbf{W}_T = \left(diag\left(\hat{\mathbf{S}}_{mean}\right)\right)^{-1}$. We prefer the latter option as it implies that all moments in the estimation are scaled according to how precise they are estimated in the sample.

To summarize, the estimation procedure implemented in our toolbox by SMM is as follows:

1. Step: Let $\mathbf{W}_T = \left(diag\left(\hat{\mathbf{S}}_{mean}\right)\right)^{-1}$ and obtain $\hat{\boldsymbol{\theta}}^{(1)}$ from (1).
2. Step: Use $\hat{\boldsymbol{\theta}}^{(1)}$ to compute $\hat{\mathbf{W}}_T^{(1)} = \hat{\mathbf{S}}^{-1}$, and obtain $\hat{\boldsymbol{\theta}}^{(2)}$ from (1)

3.2 Asymptotic Distribution of the SMM Estimator

Given standard regularity conditions, Duffie & Singleton (1993) show that the SMM estimator is asymptotically normally distributed, i.e.

$$\sqrt{T}\left(\hat{\boldsymbol{\theta}}-\boldsymbol{\theta}_o\right) \xrightarrow{d} \mathcal{N}\left(\mathbf{0}, \left(1+\frac{1}{\tau}\right)\mathbf{M}_o\mathbf{S}_o\mathbf{M}_o'\right),$$

where

$$\mathbf{M}_o = \left(\mathbf{D}_o'\mathbf{W}_T\mathbf{D}_o\right)^{-1}\mathbf{D}_o'\mathbf{W}_T$$

and

$$\mathbf{D}_o = \left.\frac{\partial \mathbf{g}^S\left(\boldsymbol{\theta}, \mathbf{y}_{1:T}\right)}{\partial \boldsymbol{\theta}'}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_o}.$$

We estimate these matrices by replacing $\boldsymbol{\theta}_o$ by $\hat{\boldsymbol{\theta}}$. If we use the optimal weighting matrix $\mathbf{W}_T = \mathbf{S}_o^{-1}$, the expression for the asymptotic variance simplifies to

$$\sqrt{T}\left(\hat{\boldsymbol{\theta}}-\boldsymbol{\theta}_o\right) \xrightarrow{d} \mathcal{N}\left(\mathbf{0}, \left(1+\frac{1}{\tau}\right)\left(\mathbf{D}_o'\mathbf{S}_o^{-1}\mathbf{D}_o\right)^{-1}\right).$$

In this case a model specification test is given by (see Ruge-Murcia (2012))

$$J = T\left(1+\frac{1}{\tau}\right)\mathbf{g}\left(\hat{\boldsymbol{\theta}}, \mathbf{y}_{1:T}\right)'\hat{\mathbf{S}}^{-1}\mathbf{g}\left(\hat{\boldsymbol{\theta}}, \mathbf{y}_{1:T}\right) \xrightarrow{d} \chi^2_{n_m-n_\theta}.$$

3.3 Moments Applied for the SMM Estimator

We allow three types of unconditional moments to be considered for the SMM estimation:

- Sample means, i.e. $\mathbf{m}_1\left(\mathbf{y}_t\right) = \mathbf{y}_t$
- Contemporaneous covariances, i.e. $\mathbf{m}_2\left(\mathbf{y}_t\right) = vech\left(\mathbf{y}_t\mathbf{y}_t'\right)$
- The own auto-covariances, i.e. $\mathbf{m}_3\left(\mathbf{y}_t\right) = \left\{y_{i,t}y_{i,t-k}\right\}_{i=1}^{n_y}$ for various values of k

Hence, the total set of moments considered for the estimation is given by

$$\mathbf{m}\left(\mathbf{y}_t\right) \equiv \begin{bmatrix} \mathbf{m}_1\left(\mathbf{y}_t\right) \\ \mathbf{m}_2\left(\mathbf{y}_t\right) \\ \mathbf{m}_3\left(\mathbf{y}_t\right) \end{bmatrix}.$$

Finally, the user can then freely decide which of the moments in $\mathbf{m}\left(\mathbf{y}_t\right)$ that are included in the estimation.

4 Implementing SMM Estimation in the Toolbox

This section explains the steps required to carry out a SMM estimation of a DSGE model when approximated up to fifth order by the extended perturbation method. The most demanding aspect for the user relates to constructing two model specific m-files containing i) the model equations and ii) the steady state solution. The requirements we impose on these two files are mainly due to our desire to concentrate out lagged control variables and the exogenous states when solving for the Extended Path.

We proceed as follows. Section 4.1 provides a brief overview of the toolbox. Section 4.2 outlines how the model equations should be coded up, and Section 4.3 provide guidance on how to implement the steady state solution. The case with non-standard transformations of any variable in the model is discussed in Section 4.4. We then show in Section 4.5 how to automatically generate functions that efficiently compute the numerical derivatives of the model up to fifth as needed for the perturbation approximation, two additional files needed to run the Extended Path, and finally files for evaluating the Euler equation errors. Section 4.6 illustrates how to estimate the stylized RBC model by SMM, and Section 4.7 shows how to evaluate accuracy.

4.1 Overview of the Toolbox

The toolbox has the following folders:

- **AccuracyEuler:** Codes needed for computing Euler equation errors.
- **DisplayModelDeriv**: This folder contains a set of general m-files that is used to generate functions to compute the DSGE model at the steady state, compute the Extended Path, and compute the Euler equation errors. These files are called by the executable scripts "createFiles_-Derivatives_model.m" and "createFiles_ExtendedPathAndAccuracy.m".
- **Documentation**: The documentation of the toolbox is saved here.
- **ExtendedPer**: Codes accompanying the paper by Andreasen & Kronborg (2020) on how to compute the Extended Path efficiently and hence compute the extended perturbation approximation.
- **files:** Contains the required files for using the codes by Levintal (2017) to compute the standard perturbation approximation up to fifth order.
- **ModelSpecification**: The content of this folder is reserved to the user and should contain the following files:
  - An m-file with the equations of the DSGE model
  - An m-file computing the steady state of the DSGE model

The remaining files in this folder will be generated automatically when running "createFiles_-Derivatives_model.m" and "createFiles_ExtendedPathAndAccuracy.m".

- **Perturbation_Levintal**: This folder contains the code accompanying Levintal (2017).
- **SMMtoolbox**: This folder contains a set of general m-files needed to carry out the SMM estimation of a DSGE model solved by the extended perturbation method and standard perturbation (with pruning) up to fifth order. The latter is included as a useful benchmark or to obtain good starting values for an estimation based on extended perturbation.

In addition to these folders, the toolbox contains five executable scripts:

- "createFiles_ExtendedPathAndAccuracy.m" uses the m-file for the DSGE model and its steady state solution to automatically generate "DSGEforesight_N.m", "getModelDeriv_EP.m", "EulerEqError.m", and "EulerEqErrorPruning.m"
- "createFiles_Derivatives_model.m" computes the required derivatives for a standard fifth order perturbation approximation.
- "Run_accuracyEuler.m" a script to evaluate the Euler equation errors
- "Run_solveAndSimulateModel.m" a script to simulate a sample path from the DSGE model by extended perturbation.
- "SunGMM.m" is the main script for carrying out the SMM estimation.

4.2 Implementing The Model Equations

When implementing the model equations describing the DSGE model, we adopt the notation and framework of Schmitt-Grohé & Uribe (2004). That is, we use the symbolic toolbox in Matlab to set up a function that reports the model equations along with the state and control vectors. Our implementation of the RBC model is available in "RBCmodel.m" which appears in the folder "ModelSpecification".

The first section of the file "RBCmodel.m" is given below:

```
function [f,x,xp,y,yp,symparams,Phi] = RBCmodel(unitFree)

%% Section 1: Declaring coefficients and variables
%Define the structureal parameters
syms DELTA BETTA B ETAl ETAc THETA ALFA RHOA RHOD
symparams=[DELTA,BETTA,B,ETAl,ETAc,THETA,ALFA,RHOA,RHOD];

%Define the state variables in this period (_cu) and the next
period
%(_cup). Variables lagged by one period appear as "_ba1"
syms k_cu  c_ba1  a_cu  d_cu
syms k_cup c_ba1p a_cup d_cup

%Define the control variables in this period (_cu) and the next
period (_cup)
syms c_cu  iv_cu  y_cu  la_cu  n_cu  rk_cu  w_cu
syms c_cup iv_cup y_cup la_cup n_cup rk_cup w_cup
```

We start by using the Matlab option "syms" to define the structural parameters of the model as symbolic objects. All these symbolic variables are then stored in "symparams." Then we define the state variables as symbolic objects and then the control variables. When defining these variables, we adopt the following notation (explained using a few examples)

- K_t is coded as k_cu where "_cu" is an abbreviation for "current"
- K_{t+1} is coded as k_cup where "_cup" is an abbreviation for "current, plus one"
- C_{t-1} is coded as c_ba1 where "_ba" is for "back" and 1 refers to one lag

The second section of the file "RBCmodel.m" contains Eq 1 to 8 of our model. That is

```
%% Section 2: Endogenous model equations
% Eq 1: FOC for consumption
if unitFree == 1
    eq1 = -1 + d_cu*((c_cu-B*c_ba1)^(-ETAc) -
BETTA*B*d_cup*(c_cup-B*c_cu)^(-ETAc))/la_cu;
else
    eq1 = -la_cu +d_cu*(c_cu-B*c_ba1)^(-ETAc) -
BETTA*B*d_cup*(c_cup-B*c_cu)^(-ETAc);
end

% Eq 2: Household's FOC for labor
if unitFree == 1
    eq2 = (-THETA*d_cu*(1-n_cu)^-ETAl)/(la_cu*w_cu) + 1;
else
    eq2 = -THETA*d_cu*(1-n_cu)^-ETAl + la_cu*w_cu;
end

% Eq 3: FOC for capital
if unitFree == 1
    eq3 = -1 + (BETTA*la_cup*(rk_cup + (1-DELTA)))/la_cu;
else
    eq3 = -la_cu + BETTA*la_cup*(rk_cup + (1-DELTA));
end

% EQ 4: Firm's FOC for capital
if unitFree == 1
    eq4 = -a_cu*(1-ALFA)*k_cu^(-ALFA)*n_cu^(ALFA) + rk_cu;
else
    eq4 = (-a_cu*(1-ALFA)*k_cu^(-ALFA)*n_cu^(ALFA))/rk_cu + 1;
end
```

and

```
% EQ 5: Firm's FOC for labor
if unitFree == 1
    eq5 = (-a_cu*ALFA*k_cu^(1-ALFA)*n_cu^(ALFA-1))/w_cu + 1;
else
    eq5 = -a_cu*ALFA*k_cu^(1-ALFA)*n_cu^(ALFA-1) + w_cu;
end

% EQ 6: National income identity
if unitFree == 1
    eq6 = (-c_cu - iv_cu)/y_cu + 1;
else
    eq6 = -c_cu - iv_cu + y_cu;
end

% EQ 7: Production function
if unitFree == 1
    eq7 = -1 + (a_cu*k_cu^(1-ALFA)*n_cu^(ALFA))/y_cu;
else
    eq7 = -y_cu + a_cu*k_cu^(1-ALFA)*n_cu^(ALFA);
end

% EQ 8: Law of motion for capital
if unitFree == 1
    eq8 = (-k_cup + (1-DELTA)*k_cu)/iv_cu + 1;
else
    eq8 = -k_cup + (1-DELTA)*k_cu + iv_cu;
end
```

Note that we do not explicitly write the conditional expectation operator, as the perturbation codes automatically takes the conditional expectation at time t to all equations of the model. For the Extended Path, it is useful to also accommodate the case where all the model equations (and hence their errors) are expressed in unit free terms to make them mutually comparable. This is done by scaling each of the equilibrium equations such that we equal 0 or 1. For instance, the equation $C_t + I_t = Y_t$ is expressed as $C_t/Y_t + I_t/Y_t = 1$.

The third section in the file "RBCmodel.m" is reserved for link-equations. Our RBC model has only one link-equation for consumption, i.e.

```
%% Section 3: Link equations
% EQ 9: Link for lagged consumption
if unitFree == 1
    eq9 = -c_ba1p/c_cu + 1;
else
    eq9 = -c_ba1p + c_cu;
end
```

Note if we have additional lagged control variables in the model, then they appear as additional state variables and lead to additional link-equations. It is important to note that the link equations must appear after the main equations of the model in order to exploit the fact that we do not separately need to solve for C_{t-1} in the Extended Path.

The fourth section in the file "RBCmodel.m" specifies the exogenous shocks to the model using the representation in Eq 10 and Eq 11, i.e.

```
%% Section 4: The exogenous shocks
% EQ 10: Law of motion for technology
eq10 = -log(a_cup) + RHOA*log(a_cu);

% EQ 11: Law of motion for technology
eq11 = -log(d_cup) + RHOD*log(d_cu);
```

It is important that the equations for the exogenous shocks appear last in the **f**-function when computing the certainty equivalent solution by the Extended Path. Note also that we do not explicitly type in the structural innovations, i.e. we do not need to write $\sigma_A \epsilon_{A,t+1}$. Note also that the adopted timing convention for the exogenous shocks differs from the one used in Dynare, where the law of motion for technology typically would be represented as

$$\log A_t = \rho_A \log A_{t-1} + \sigma_A \epsilon_{A,t}.$$

This is because Dynare considers both A_{t-1} and $\epsilon_{A,t}$ as state variables. By using the representation in Eq 10, i.e. $\log A_{t+1} = \rho_A \log A_t + \sigma_A \epsilon_{A,t+1}$, only A_t appears as a state variable, which reduces the computational burden when solving the DSGE model. We finally stack all equations in section 4 and construct the vector function **f**.

The final section of the file "RBCmodel.m" defines the state vector

$$\mathbf{x}_t = \left[\begin{array}{cccc} K_t & C_{t-1} & A_t & d_t \end{array} \right]$$

and the control vector

$$\mathbf{y}_t = \left[\begin{array}{cccccc} C_t & I_t & Y_t & \lambda_t & N_t & R_t^k & W_t \end{array} \right]$$

in the following way:

```
%% Section 5: Defining x,xp,y,yp
% Define the vector of states, x and xp
x  = [k_cu  c_ba1  a_cu  d_cu];
xp = [k_cup c_ba1p a_cup d_cup];
% Define the vector of controls, y and yp
y =  [c_cu  iv_cu  y_cu  la_cu  n_cu  rk_cu  w_cu ];
yp = [c_cup iv_cup y_cup la_cup n_cup rk_cup w_cup];

% For the log-approximation
f   = subs(f, [x,y,xp,yp], exp([x,y,xp,yp]));

% Phi: the expected value for the exogenous shocks, else use Phi =
[]
Phi = [RHOA*log(a_cu);RHOD*log(d_cu)];
% For the log-approximation
Phi = subs(Phi, [x,y,xp,yp], exp([x,y,xp,yp]));
```

Note the that $\mathbf{x}_t$ is coded as **x** and $\mathbf{x}_{t+1}$ is coded as **xp** following the convention introduced by Schmitt-Grohé & Uribe (2004), and similarly for the control vector. In order to only solve for the

control variables and the *endogenous* states in the Extended Path, we impose that the state and control vector must be defined as follows:

$$\underbrace{\mathbf{x}_t}_{nx} = \left[\underbrace{\text{endogenous states}}_{mx} \quad \underbrace{\text{controls appearing in } \mathbf{x}_t}_{myx} \quad \underbrace{\text{exogenous states}}_{ne} \right] \tag{3}$$

$$\underbrace{\mathbf{y}_t}_{ny} = \left[\underbrace{\text{controls appearing in } \mathbf{x}_t}_{myx} \quad \text{remaining controls} \right] \tag{4}$$

Note that the size of the vectors are given below, i.e. $\mathbf{x}_t$ has nx elements and so on. Applying these rules to our RBC model, this explains why the state vector is defined as K_t (the endogenous state variable), C_{t-1} (the lagged control variable), and (A_t, d_t) (the exogenous states). Note also that the order of the variables "controls appearing in $\mathbf{x}_t$" must be the same in $\mathbf{x}_t$ and $\mathbf{y}_t$. This explains why C_t appears as the first control variable in $\mathbf{y}_t$.

The final line in the codes above is needed for a log-approximation as it replaces all elements in $[\mathbf{x}_t, \mathbf{y}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}]$ by $\exp\{[\mathbf{x}_t, \mathbf{y}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}]\}$ in the **f-**function. That is, the **f-**function changes from

```
f =

 d_cu/(c_cu - B*c_ba1)^ETAc - la_cu - (B*BETTA*d_cup)/(c_cup - B*c_cu)^ETAc
                                    la_cu*w_cu - (THETA*d_cu)/(1 - n_cu)^ETAl
                                    BETTA*la_cup*(rk_cup - DELTA + 1) - la_cu
                             (a_cu*n_cu^ALFA*(ALFA - 1))/(k_cu^ALFA*rk_cu) + 1
                              w_cu - ALFA*a_cu*k_cu^(1 - ALFA)*n_cu^(ALFA - 1)
                                                          y_cu - iv_cu - c_cu
                                        a_cu*k_cu^(1 - ALFA)*n_cu^ALFA - y_cu
                                             iv_cu - k_cup - k_cu*(DELTA - 1)
                                                               c_cu - c_ba1p
                                                  RHOA*log(a_cu) - log(a_cup)
                                                  RHOD*log(d_cu) - log(d_cup)
```

to

```
f =

 exp(d_cu)/(exp(c_cu) - B*exp(c_ba1))^ETAc - exp(la_cu) - (B*BETTA*exp(d_cup))/(exp(c_cup) - B*exp(c_cu))^ETAc
                                                 exp(la_cu)*exp(w_cu) - (THETA*exp(d_cu))/(1 - exp(n_cu))^ETAl
                                                      BETTA*exp(la_cup)*(exp(rk_cup) - DELTA + 1) - exp(la_cu)
                                           (exp(-rk_cu)*exp(a_cu)*exp(n_cu)^ALFA*(ALFA - 1))/exp(k_cu)^ALFA + 1
                                           exp(w_cu) - ALFA*exp(a_cu)*exp(k_cu)^(1 - ALFA)*exp(n_cu)^(ALFA - 1)
                                                                          exp(y_cu) - exp(iv_cu) - exp(c_cu)
                                                    exp(a_cu)*exp(k_cu)^(1 - ALFA)*exp(n_cu)^ALFA - exp(y_cu)
                                                             exp(iv_cu) - exp(k_cup) - exp(k_cu)*(DELTA - 1)
                                                                                       exp(c_cu) - exp(c_ba1p)
                                                                        RHOA*log(exp(a_cu)) - log(exp(a_cup))
                                                                        RHOD*log(exp(d_cu)) - log(exp(d_cup))
```

Effectively, this transformation implies that the perturbation approximation is carried out for a log-transformation of the state and control variables, i.e. for

$$\mathbf{x}_t = \left[\log K_t \quad \log C_{t-1} \quad \log A_t \quad \log d_t \right]$$

$$\mathbf{y}_t = \left[\begin{array}{ccccccc} \log C_t & \log I_t & \log Y_t & \log \lambda_t & \log N_t & \log R_t^k & \log W_t \end{array} \right]. \tag{5}$$

However, the package also accommodates the possibility of using a simple 'level' approximation, which in this case corresponds to omitting "f = subs(f, ...)". Another possibility which is also accommodated is to use a logistic transformation of a variable. This may be a convenient transformation to consider if a variable lies within the unit interval. For instance, for some variable X_t in the model, we may let

$$X_t = \frac{1}{1 + e^{-\tilde{X}_t}}$$

and do the approximation in $\tilde{X}_t$.

Finally, in "Phi" the user has the opportunity to report the conditional expectation of the exogenous shocks, which may be used by the codes of Levintal (2017) when computing the standard perturbation approximation.

We acknowledge that the required ordering of the model equations and the state and control variables perhaps may seen a bit strange. So let us briefly explain why this ordering is required. Firstly, in the Extended Path we do not need to separately solve for control variables and lagged control variables (i.e. C_{t-1} and C_t in our case). Instead, we exploits that we only need to solve for the control variables once and this increases the numerical efficiency of the codes. To exploit this numerical trick, the codes require that the link equations appear after the main model equations and that the state and control vectors are ordered as outlined in (3) and (4). Secondly, we also exploit that for linear shock processes, we can easily solve for these variables in the Extended Path independently of any of the endogenous variables in the model. This explains why the exogenous shocks must appear last in the **f**-function and the exogenous shocks must appear last in the state vector. We should finally note that the user may "turn off" both of these optimized features in the codes by letting $mx = nx$ and $myx = 0$. This may for instance be useful when debugging the codes.

4.3 Implementing the Steady State

Our implementation of the steady state solution for our RBC model is available in "RBCmodel_ss.m" appearing in the folder "ModelSpecification". The first part of this function defines the output arguments as follows:

```
function [auxOut,errorMes] = RBCmodel_ss(params)

%% Section 1:
% The size of the model
ny    = 7;      %Number of control variables
nx    = 4;      %Number of state variables
ne    = 2;      %Number of shocks
mx    = 1;      %Number of endogenous state variables (must come first in x)
myx   = 1;      %Number of lagged control variables appearing as states.
                %These lagged control variables must come first in y AND must
                %appear in x after the endogenous states.
```

All output are stored in "auxOut" (auxiliary output) and "errorMes", which is a flag for reporting errors. We then set the size of the model.

The next step is to unfold the structural parameters in the structure "params". We also define the eta-matrix $\boldsymbol{\eta}$ which in our framework is the square-root of the covariance matrix to the structural shocks. Note that technology A_t appears at the third position in $\mathbf{x}_t$, and σ_A therefore appears at position $\eta(3,1)$ and so on. Finally, we set the higher order moments, which we specify to be Gaussian (using the codes provided by Levintal (2017)).

```
%Unfold the params vector
DELTA = params.DELTA;
BETTA = params.BETTA;
B     = params.B;
ETAl  = params.ETAl;
ETAc  = params.ETAc;
THETA = params.THETA;
ALFA  = params.ALFA;
RHOA  = params.RHOA;
RHOD  = params.RHOD;
STDA  = params.STDA;
STDD  = params.STDD;

% The eta matrix: nx*ne
eta      = zeros(nx,ne);
eta(3,1) = STDA;
eta(4,2) = STDD;

% The higher order moments
% M.M2 is the expected value of kron(eps,eps).
% M.M3 is the expected value of kron(eps,eps,eps) and so on.
% Here I assume that the shock is standard normal.
% M.M2=1; M.M3=0; M.M4=3; M.M5=0;
% If the shocks are independent standard normal you can use the
command:
momEps =gaussian_moments(ne);
```

The second section of "RBCmodel_ss.m" reported on the next page simply computes the steady state as a function of the structural parameters.

```
%% Section 2: Solving for the steady state
% Initializing errorMes: 0 for no errros, else 1
errorMes = 0;

% The value of technology
A = 1;

% The value of the preference shock
D = 1;

% The rental rate of capital
RK = 1/BETTA - (1-DELTA);

% Capital divided by labor
K_O_N = (RK/(A*(1-ALFA)))^(-1/ALFA);
if K_O_N <= 0
    errorMes = 1;
end

% The wage level
W = A*ALFA*(K_O_N)^(1-ALFA);

% Investment over labor
IV_O_N = DELTA*K_O_N;

% Output over labor
Y_O_N = A*K_O_N^(1-ALFA);

% Consumption over labor
C_O_N = Y_O_N - IV_O_N;
if C_O_N <= 0
    errorMes = 1;
end
```

and

```

% The labor level
if ETAc == 1 && ETAl == 1
    % Closed-form solution for N
    N = (1-BETTA*B)*(C_O_N*(1-B))^-1*W/THETA/(1+(1-BETTA*B)*(C_O_N*(1-B))^-1*W/THETA);
else
    % No closed-form solution and we therefore use a fixed-point algorithm
    if errorMes == 0
        options = optimset('Display','off','TolX',1e-12,'TolFun',1e-12);
        N0 = 1/3;
        [N,~,exitflag] = fsolve(@findN,N0,options);
        if exitflag <= 0
            errorMes = 1;
        end
    else
        N = NaN;
    end
end

% Value of remaining variables
C  = C_O_N*N;
Y  = Y_O_N*N;
IV = IV_O_N*N;
K  = K_O_N*N;
LA = (C-B*C)^(-ETAc)-BETTA*B*(C-B*C)^(-ETAc);
```

To find the steady state value of hours worked, i.e. N_{ss}, the function "findN" is defined in the bottom of the file "RBCmodel_ss.m" as follows

```
%% Section 4: auxiliary function for the steady state
function error = findN(N)
error = THETA*(1-N)^(-ETAl)*N^ETAc - (1-BETTA*B)*(C_O_N*(1-B))^(-ETAc)*W;
end
```

The third section in "RBCmodel_ss.m" simply assigns the steady state values to all elements in the states (in "X_{ss}") and the controls (in "Y_{ss}") in the appropriate order. Then we also provide the name of the variables in "labelx" and "labely", respectively. Finally, we indicate in "transformX" and "transformY", which transformation that is applied for each of the variables in the model. Note that we use a log-transformation of the steady state because all variables in "RBCmodel.m" were transformed by the exp-function.

```
%% Section 3: Reporting the output for the perturbation approximation
% For level approximation: k_cu = K;
% For log transformation: k_cu = log(K);
% For logistic transformation: k_cu = -log(1/Vss-1)

%The level of the states
auxOut.Xss        = [K C A D]';

%The level of the controls
auxOut.Yss        = [C IV Y LA N RK W]';

auxOut.labelx     = [{'k_t'},{'c_{t-1}'},{'a_t'},{'d_t'}];
auxOut.labely     = [{'c_t'},{'i_t'},{'y_t'},{'la_t'},{'n_t'},{'rk_t'},{'w_t'}];

%1 for a log-transformation, 2 for logistic transformation, 0 for a level approx
auxOut.transformX = ones(1,nx);
auxOut.transformY = ones(1,ny);
```

Within the third section of "RBCmodel_ss.m" we finally save some auxiliary output in the structure.

4.4 Non-Standard Transformations in The Model

In some cases it may also be necessary to "untransform" the solution from the approximation to get the model output to match with the corresponding output in the data. This should currently only be the case if a logistic function is used, but it may also be the case for a log-transformation if the empirical data is not log-transformed. In any case, it is recommended that one briefly checks the content of the file "untransformYandX.m" which currently reads:

```
function [Ysim,Xsim] = untransformYandX(Ysim,model,Xsim)
ny = size(Ysim,1);
% undo the transformation
for i=1:ny
    if model.transformY(i) == 1
        % log-transformation

    elseif model.transformY(i) == 2
        % logistic transformation
        Ysim(i,:) = 1./(1+exp(-Ysim(i,:)));
    elseif model.transformY(i) == 0
        % Level approximation

    end
end
if exist('Xsim','var')
    nx          = size(Xsim,1);
    for i=1:nx
        if model.transformX(i) == 1
            % log-transformation

        elseif model.transformX(i) == 2
            % logistic transformation
            Xsim(i,:) = 1./(1+exp(-Xsim(i,:)));
        elseif model.transformX(i) == 0
            % Level approximation
        end
    end
end

end
```

4.5 Generating Functions for the Numerical Derivatives

Based on the two model specific m-files, we are now able to generate functions that compute the required numerical derivatives of the model.

To generate these functions for standard perturbation, open the executable script called "createFiles_Derivatives_model.m". The user specific part of this script is given below:

```
orderApp=5;
unitFree = 0;

%% Model specific part
[f,x,xp,y,yp,symparams,Phi] = RBCmodel(unitFree);

% A string with the name of the steady state file
nameSteadyStateFile = '[auxOut,errorMes] = RBCmodel_ss(params);';
```

We first specify the order of the approximation using "orderApp". Then we call the m-file containing the model equations - i.e. "RBCmodel.m" in our case. Finally, we need to provide the name of the file computing the DSGE model in the steady state. Here, the name of the file (i.e. RBCmodel_ss) is optional but the output arguments must be as indicated above. Based on these inputs, running the script then uses the codes of Levintal (2017) to compute the required derivatives.

To generate the functions for the Extended Path (as needed for the extended perturbation method) and to generate codes to compute the Euler equation errors, open the executable script called "createFiles_ExtendedPathAndAccuracy.m". The user specific part of this script is given below:

```
%% Model specific part
unitFree = 1;
[f,x,xp,y,yp,symparams] = RBCmodel(unitFree);

% The number of endogenous states (mx) and number of lagged controls (myx)
mx  = 1;
myx = 1;
```

For the Extended Path, it is useful to express all the model equations (and hence their errors) in unit free terms to make the errors comparable across the various equations in the model. We account for this in the current implementation by letting "unitFree = 1". Finally, we need to specify the number of endogenous states "mx=1" and the number of lagged controls "myx = 1" in our case.

Following the construction of these files, we are now ready to carry out the SMM estimation.

4.6 Executing the SMM Estimation

To start the SMM estimation, open the file called "RunSMM.m", which simulates a sample path from our RBC model and then estimates the RBC model on this sample path. The first section of this file lists the main user settings:

```
%% Section 1: User settings for the SMM estimation
appMethod     = 2;          % 1 for pruning and 2 for extended perturbation
tau           = 5;          % model moments computed from a sample of tau*T observations
orderApp      = 2;          % Approximation order of DSGE model
T             = 200;        % Length of the simulated sample path

autoLagsIdx   = [1 5];      % Number of lagged covariances included in the GMM estimation
selectY       = [1,2,5]'; % The control variables selected for the estimation
qLag          = 20;         % Number of lags for the Newey-West estimator of W
epsValue      = 1e-6;       % stepsize for computing standard errors

% Among the variables in selectY, we include the following moments in the
% estimation, where 1 indicates inclusion and 0 otherwise.
inclMoms_Ey       = [1 1 1]';   %for first moments
inclMoms_Eyy      = [1 1 1 ;    %for contemporaous second moments
                     0 1 1 ;
                     0 0 1 ];
inclMoms_autoEyy  = repmat([1 1 1 ]',1,length(autoLagsIdx)); %for auto-cov
```

That is, we select the approximation method ("appMethod") and the length of the simulated sample path ("tau") used to compute the model implied moments. We also set the order of the approximation ("orderApp"), and the length of the sample path ("T"). Then we specify the number of moments which we can select from in the estimation. Here

- "autoLagsIdx" specifies the considered lags for the own auto-covariances of $\mathbf{y}_t$ which appear in the moment conditions. In this case, we include $\{y_{i,t}y_{i,t-1}\}_{i=1}^{n_y}$ and $\{y_{i,t}y_{i,t-5}\}_{i=1}^{n_y}$ in $\mathbf{m}_3(\mathbf{y}_t)$

- "selectY" selects the elements in the control vector which we consider for the SMM estimation. In our case, we use position 1,2, and 5 in $\mathbf{y}_t$, implying that we use moments for $\begin{bmatrix} \log C_t & \log I_t & \log N_t \end{bmatrix}$ in the estimation (see (5))

Below in the "inclMoms_Ey", etc. we then specify which of the moments among the ones considered for the estimation that we really want to include. Here, a 1 indicates that a moment is include, and zero otherwise. Finally, "qLag" is the number of lags when computing the Newey-West estimate of the weighting matrix, and "epsValue" is the step size used when computing standard errors for the SMM estimator.

The next part of the control settting relates to the optimizer

```
% The optimizer
optim         = 2;          % 1 for the CMAES
                            % 2 for a gradient based optimizer
                            % 3 for the simple algorithm by Nelder-Mead.
numOptimStep1 = 2;          % Number of optimizations in step 1
numOptimStep2 = 2;          % Number of optimizations in step 1
MaxIter       = 5000;       % Max number of iterations for the optimizer
MaxEvals      = 5000;       % Max number of function evaluations for the optimizer
TolFun        = 1D-6;       % Function tolerance at the objective function
TolX          = 1D-6;       % Function tolerance for the coefficients
PopSize       = 50;         % Number of draws per generation i.e. lambda in the CMAES
```

- "optim" = 1 for the CMAES routine considered in Andreasen (2010)
- "optim" = 2 for the Levenberg-Marquard algorithm in Matlab as the objective function in the SMM estimation can be expressed as

$$\begin{aligned} Q &= \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T})' \mathbf{W}_T \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T}) \\ &= \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T})' \mathbf{S}_T' \mathbf{S}_T \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T}) \\ &= \left(\mathbf{S}_T \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T})\right)' \left(\mathbf{S}_T \mathbf{g}^S(\boldsymbol{\theta}, \mathbf{y}_{1:T})\right) \\ &= \tilde{\mathbf{g}}(\boldsymbol{\theta}, \mathbf{y}_{1:T})' \tilde{\mathbf{g}}(\boldsymbol{\theta}, \mathbf{y}_{1:T}) \end{aligned}$$

  where $\mathbf{W}_T = \mathbf{S}_T' \mathbf{S}_T$ is the Cholesky decomposition. Hence, the structure of the SMM estimator is equivalent to a non-linear least squares regression problem for which the Levenberg-Marquard algorithm applies.
- "optim" = 3 for the Nelder-Mead simplex algorithm
- "numOptimStep1" and "numOptimStep2" is the number of times we restart the optimizer in step 1 and step 2, respectively.
- The following five options specified in section 1 of "RunSMM.m" relate to the optimizers and should be obvious.

```
% Configurations for the Extended Perturnation
setupEPer.Nmax            = 200;       %Max steps in the Extended Path solution
setupEPer.Nmin            = 50;        %Min steps in the Extended Path solution
setupEPer.maxDistSS       = 0.001;     %Max distance to SS when determining N
setupEPer.orderAppStart   = 4;         %Order of approximation for the starting values
                                       %in the fixed point solver (max 4th order)
setupEPer.fixedPointSolver= 1;         %1 for the Newton-Raphson solver,
                                       %2 Newton-Raphson solver with optimal delta,
                                       %3 for an LM algorithm that minimizes the residuals
                                       %4 for an LM algorithm that minimizes the weighted residuals
setupEPer.JacobianOption  = 3;         %1 for using numerical J and numerically
                                       %  solving the system J*x = -f
                                       %2 for using analytical J to solve the system J*x = -f
                                       %3 for using analytical J computed recursively to solve
                                       %  the system J*x = -f
setupEPer.lambda0         = 1e-6;      %Tuining parameter for fixedPointSolver = 3
setupEPer.lambdaBackup    = 1e-2;      %Tuining parameter for the LM algorithm when used as backup
setupEPer.tolf            = 1e-6;      %Tolerance level for optimization problem
setupEPer.MaxIter         = 1D4;       %Maximum number of iterations allowed in the Extended Path
algorithm
setupEPer.residualMax     = 0.0001;    %Max allowed value of a residual - for the hybrid simulator
setupEPer.MexOn           = 1;         %1 For using MEX-files, else 0
```

For extended perturbation, there is an additional set of options which the user needs to set. Let us briefly go through these options and explain how they are related to Andreasen & Kronborg (2020). These options are stored in the struct setupEPer as follows:

- "Nmax" denotes the maximal horizon for the Extended Path and is denoted by $N_{\max}$ in Andreasen & Kronborg (2020).
- "Nmin" denotes the shortest allowed horizon for the Extended Path and is denoted by $N_{\min}$ in Andreasen & Kronborg (2020).
- "maxDistSS" is the maximal allowed distance to the steady state, which is allowed when setting the horizon N in the Extended Path. Hence, "maxDistSS" corresponds to "D_{ss}" in Andreasen & Kronborg (2020).
- "orderAppStart" is the approximation order used in standard perturbation to obtain good starting values for the Extended Path.
- "fixedPointSolver" determines which fixed-point solver is used in the Extended Path. The two first options are standard fixed-point algorithms, whereas the latter two simply minimizes the euler-equation errors by a type of Levenberg-Marquard (LM) algorithm. When "fixedPointSolver=4", then we put decaying weights on euler equation errors from period "t" and until period "t+N". Note also if these algorithms are unable to solve for the certainty equivalence solution, then we restart the Extended Path using a "backup" LM algorithm.
- "JacobianOption" specifies how we solve a linear equation system within each iteration of the Newton-Rapson solver, i.e. when "fixedPointSolver = 1" or "fixedPointSolver = 2". Here, "JacobianOption = 2" uses analytical derivatives of the model to formulate this linear problem,

which is solved by standard methods (although exploiting the sparcity of the system). When "JacobianOption = 3", then we recursively solve this linear system as outlined in Boucekkine (1995). Normally, the fastest option is to let "JacobianOption = 3", whereas "JacobianOption = 1" is only for debugging.

- "lambda0" and "lambdaBackup" are tuning coefficients for the LM algorithm.
- "tolf" is the tolerance used when solving for the Extended Path.
- "MaxIter" is the max number of iterations allowed for when solving for the Extended Path.
- "ResidualMax" is "EE" in Andreasen & Kronborg (2020).
- "MexOn" allows you to use an Mex implementation of "CondMoments_4th_levelCE.m". BUT, here you probably need to recompile the Mex-file unless you can use my compiled version (applies to 64bit computer windows machine)

In the second section of "RunSMM.m" the user first needs to list in "allModelParams" all the names of the structural parameters appearing in the DSGE model. We use this list of coefficients to check that a given structural coefficient is either calibrated or estimated.

```
%% Section 2: Model parameters
allModelParams =
{'DELTA','BETTA','B','ETAl','ETAc','THETA','ALFA','RHOA','RHOD','STDA','STDD'};

% The variables which we do not select for estimation are calibrated to the
assigned values
calibrateParams.ETAl  = 1.00;
calibrateParams.THETA = 3.48;
calibrateParams.RHOD  = 0.95;
calibrateParams.STDD  = 0.01;
calibrateParams.DELTA = 0.025;

% Starting values
params0.BETTA           = 0.984;
params0.B               = 0.5;
params0.ETAc            = 5;
params0.ALFA            = 0.667;
params0.RHOA            = 0.979;
params0.STDA            = 0.0072;
```

In the structure "calibrateParams", we denote the parameters which are calibrated and their calibrated values. In the structure "params0", we denote the parameters which are estimated and their starting value for the SMM estimation.

Section 3 of "RunSMM.m" reads:

```
%% Section 3: We load the data or simulate the model using params0
load([pwd,'\ModelSpecification\setupModel'],'setupModel')
data =
dataForSMM(params0,calibrateParams,orderApp,T,selectY,setupModel,setupEPer,appMethod);

% Compute empirical moments
inclMoms = collectMoments(inclMoms_Ey,inclMoms_Eyy,inclMoms_autoEyy,autoLagsIdx);
dataInfo = momentsGMMData(data,autoLagsIdx,inclMoms);

% Transforming params0 for estimation
selectParams  = fieldnames(params0);
params0Values = struc2values(params0,selectParams);

% Test of enought moments for estimation
numMom    = sum(inclMoms);
numParams = size(selectParams,1);
disp(['Parameters to estimate = ', num2str(numParams) ,'. Moments for estimation = ',
num2str(sum(inclMoms))]);
if numMom < numParams
    error('We must have at least as many moments as parameters for GMM')
end

% For CMAES, the standard deviations for the search
Insigma.DELTA           = 0.01;
Insigma.BETTA           = 0.01;
Insigma.B               = 0.1;
Insigma.ETAl            = 1;
Insigma.ETAc            = 1;
Insigma.ALFA            = 0.05;
Insigma.RHOA            = 0.05;
Insigma.RHOD            = 0.05;
Insigma.STDA            = 0.001;
Insigma.STDD            = 0.001;

% For CMAES, the lower and upper bounds for the parameters
lowerBounds.DELTA       = 0;
lowerBounds.BETTA       = 0;
lowerBounds.B           = 0;
lowerBounds.ETAl        = 0;
lowerBounds.ETAc        = 0;
lowerBounds.ALFA        = 0;
lowerBounds.RHOA        = 0;
lowerBounds.RHOD        = 0;
lowerBounds.STDA        = 0;
lowerBounds.STDD        = 0;

upperBounds.DELTA       = 1;
upperBounds.BETTA       = 1;
upperBounds.B           = 1;
upperBounds.ETAl        = 10;
upperBounds.ETAc        = 10;
upperBounds.ALFA        = 1;
upperBounds.RHOA        = 1;
upperBounds.RHOD        = 1;
upperBounds.STDA        = 1;
upperBounds.STDD        = 1;

% Test of all variables names are either estimated or calibrated
paramsTest(allModelParams,params0,calibrateParams)
```

That is, we load our empirical data, which in our case is a simulated sample path. Then we compute the chosen moments on the empirical data and test if we have sufficient moments for the estimation. We finally set the search distribution in the CMAES routine ("Insigma") and the lower and upper bounds for all parameters in the model.

In section 4 of "Run_SMM.m", we simply save all relevant information for the SMM estimation in the structure "setupStep1".

```
%% Section 4: Constructing the struct setupStep1
setupStep1.setupModel        = setupModel;
setupStep1.selectParams      = selectParams;
setupStep1.calibrateParams   = calibrateParams;
setupStep1.data              = data;
setupStep1.dataInfo          = dataInfo;
setupStep1.autoLagsIdx       = autoLagsIdx;
setupStep1.inclMoms          = inclMoms;
setupStep1.optim             = optim;
setupStep1.MaxIter           = MaxIter;
setupStep1.MaxEvals          = MaxEvals;
setupStep1.TolFun            = TolFun;
setupStep1.TolX              = TolX;
setupStep1.PopSize           = PopSize;
setupStep1.selectY           = selectY;
setupStep1.orderApp          = orderApp;
setupStep1.Insigma           = Insigma;
setupStep1.lowerBounds       = lowerBounds;
setupStep1.upperBounds       = upperBounds;
setupStep1.optimWeightMat    = 0;
setupStep1.qLag              = qLag;
setupStep1.tau               = tau;
setupStep1.appMethod         = appMethod;
setupStep1.setupEPer         = setupEPer;
```

The first step of the SMM estimation is carried out in section 5 of "RunSMM.m":

```
%% Section 5: Step 1 of the SMM estimation
Wstep1                   = getOptimalWeighting(qLag,dataMoments,setupStep1);
setupStep1.Sw            = chol(diag(diag(Wstep1)));
for i=1:numOptimStep1
    [paramsStep1,setupStep1] = runOptimization(setupStep1,params0);
    params0 = paramsStep1;
end
resultsStep1             = getSESMM(struct2array(paramsStep1)',epsValue,setupStep1);
```

We start by computing the weighting matrix using the value of the moments in the empirical sample, that is we compute $\mathbf{W}_T = \left(\hat{\mathbf{S}}_{mean}\right)^{-1}$ using the notation introduced in Section 3. Then we use the diagonal of $\mathbf{W}_T$ and save its square-root in "setupStep1.Sw". We then run the first step of the SMM estimation and compute standard errors. All results from the first estimation step are available in the structure "resultsStep1" which takes the following form:

```
resultsStep1 =

               params: [1x1 struct]
            paramsSE: [1x1 struct]
            Jacobian: [15x8 double]
                   Q: 0.4853
               model: [1x1 struct]
        modelMoments: [15x1 double]
         dataMoments: [15x1 double]
         nameMoments: {1x15 cell}
   modelMomentsScaled: [15x1 double]
    nameMomentsScaled: {1x15 cell}
    dataMomentsScaled: [15x1 double]
```

That is, we report the estimated parameters ("params"), the standard errors ("paramsSE"), the jacobian to test for local identification ("Jacobian"), the value of the objective function at optimum

("Q"), the perturbation approximation in the structure "model", the moments in the RBC model ("modelMoments"), the moments in the empirical sample ("dataMoments"), and the type of the i'th moments considered in "nameMoments". Finally, we also report the scaled version of these moments.

The second step of the SMM estimation is carried out in section 6 of "RunSMM.m":

```
%% Section 6: Step 2 of the SMM estimation
[~,modelMoments]          = objectFunc(struct2array(paramsStep1)',setupStep1);
Wopt                      = getOptimalWeighting(qLag,modelMoments,setupStep1);
setupStep2                = setupStep1;
setupStep2.Sw             = chol(Wopt);
setupStep2.optimWeightMat = 1;
disp(['Rank of Wopt = ', num2str(rank(Wopt))])
params0Step2              = paramsStep1;
for i=1:numOptimStep2
    [paramsStep2,setupStep2]  = runOptimization(setupStep2,params0Step2);
    params0Step2 = paramsStep2;
end

% The standard errors
resultsStep2 = getSESMM(struct2array(paramsStep2)',epsValue,setupStep2);
```

We start by getting the model moments from the first step and use these moments to compute the optimal weighting matrix ("Wopt"). Then we run the second estimation step and obtain standard errors. All results from the second estimation step are available in the structure "resultsStep2" which has the following form:

```
resultsStep2 =

                Jtest: 5.4194
              JtestDf: 7
            ProbJtest: 0.6089
               params: [1x1 struct]
             paramsSE: [1x1 struct]
             Jacobian: [15x8 double]
                    Q: 0.0090
                model: [1x1 struct]
          modelMoments: [15x1 double]
           dataMoments: [15x1 double]
           nameMoments: {1x15 cell}
    modelMomentsScaled: [15x1 double]
     nameMomentsScaled: {1x15 cell}
     dataMomentsScaled: [15x1 double]
```

Compared to resultsStep1, the only new items are the reported test statistic for the J-test ("Jtest"), the number of degrees of freedom for the J-test ("JtestDf"), and the related P-value ("ProbJtest").

4.7 Evaluating Accuracy

The script "Run_accuracyEuler.m" allows you to evaluate the accuracy of the approximation by computing the Euler equation errors along a simulated sample path or an a grid. When computing the Euler equation errors, Gauss-Hermite polynomials are used to carry out the numerical integration impled by the conditional expectation operator.

References

Andreasen, M. M. (2010), 'How to maximize the likelihood function for a DSGE model', *Computational Economics* **35**(2), 127–154.

Andreasen, M. M. (2012), 'On the Effects of Rare Disasters and Uncertainty Shocks for Risk Premia in Non-Linear DSGE Models', *Review of Economic Dynamics* **15**(3), 295–316.

Andreasen, M. M. & Kronborg, A. (2020), 'The Extended Perturbation Method: With Applications to the New Keynesian Model and the Zero Lower Bound', *Aarhus University: Working Paper* .

Boucekkine, R. (1995), 'An alternative methodology for solving nonlinear forward-looking models', *Journal of Economic Dynamics & Control* **19**, 711–734.

Duffie, D. & Singleton, K. J. (1993), 'Simulated Moments Estimation of Markov Models of Asset Prices', *Econometrica* **61**(4), 929–952.

King, R. G. & Rebelo, S. T. (1999), 'Resuscitating Real Business Cycles', *Handbook of Macroeconomics* **1**, 927–1007.

Levintal, O. (2017), 'Fifth-Order Perturbation Solution to DSGE Models', *Journal of Economics Dynamic and Control* **80**, 1–16.

Ruge-Murcia, F. (2012), 'Estimating Nonlinear DSGE Models by the Simulated Method of Moments: With an Application to Business Cycles', *Journal of Economics Dynamic and Control* **36**(6), 914–938.

Schmitt-Grohé, S. & Uribe, M. (2004), 'Solving Dynamic General Equilibrium Models Using a Second-Order Approximation to the Policy Function', *Journal of Economic Dynamics and Control* **28**(4), 755–775.