Paris School of Economics

Michał Miktus

11716114

Machine learning approach to DSGE estimation

Master dissertation
in
Analysis and Policy in Economics

Thesis advisorous:
Pablo Winant Ph.D.
Paris School of Economics

June 2019

Declaration of the supervisor

I declare the following thesis project was written under my supervision and I state that the project meets all submission criteria for the procedure of academic degree award.

Date Signature of the Supervisor

Declaration of the author of the project

Aware of legal responsibility, I declare that I am the sole author of the following thesis project and that the project I submit is entirely free from any content that constitutes copyright infringement or has been acquired contrary to applicable laws and regulations.

I also declare that the below project has never been subject of degree-awarding procedures in any school of higher education.

Moreover I declare that the attached version of the thesis project is identical with the enclosed electronic version.

Date Signature of the Author

Abstract

The estimation of Dynamic Stochastic General Equilibrium (DSGE) models through the Maximum Likelihood has a relatively protracted history. Elseways, the Machine Learning (ML) frameworks recently, yet profoundly entered the economics field. The following paper attempts to implement the advanced achievements of ML, not to mention the Automatic Differentiation (AD) or Stochastic Gradient Descent (SGD), to the Maximum Likelihood estimation of renowned Smets and Wouters (2003) DSGE model. The author identifies that the proposed framework can be considered as a valuable alternative to the Bayesian interference in case of highly-dimensional space parameter estimation.

Keywords

DSGE, Machine Learning, Maximum Likelihood, Automatic Differentation

Subject area (Socrates-Erasmus codes)

14.3 Economics

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Machine learning is an application of artificial intelligence (AI) that provides systems the capability to automatically lucubrate and progress from experience without being explicitly programmed. It concentrates on the development of computer programs being able to access data and then use it in their learning process. In the last decade, machine learning frameworks made the vast progress in plethora of fields, gradually entering into the economics applications as well (Mullainathan and Spiess (2017)).

On contrary, Dynamic Stochastic General Equilibrium (DSGE) models have become a standard tool in macroeconomics studies. Their attractiveness arises from the explicit specification of the objectives and constraints faced by households and firms, further employed in determination of the prices and allocations that emerge from their market interaction in an uncertain environment. Until recently, the calibration was the most common concept in the literature to explore the empirical properties of DSGE models. To be more specific, the calibration consists of fixing the structural parameters values to those estimated in preceding studies or those calculated using long-run averages of aggregate data.

In spite of the fact that the calibration is an appreciated utensil for understanding the dynamic properties of DSGE models, there is a plethora of advantages in their fully-fledged econometric estimation. To begin with, parameter estimates are obtained by imposing the restrictions of the model on the data, therefore they can overcome the inconsistency of DSGE assumptions with the calibration assumptions from the micro studies. Furthermore, the econometric estimation of the DSGE model permits to obtain estimates of parameters that may be difficult or even impossible to calculate from disaggregated data. Additionally, bootstrap techniques can be incorporated explicitly in impulse-response analysis, allowing to construct confidence intervals for the model's response to a shock, thus integrating the parameter uncertainty. Finally, classic tools of model selection and evaluation can be promptly enforced.

The Maximum Likelihood estimation procedure is relatively standard and its asymptotic properties are well known. Moreover, there are several studies implementing the aforementioned framework to the DSGE estimation, not to mention Ruge-Murcia (2007) or Andreasen (2010). The main contribution of the following paper is to capitalize the recent advancements in the machine learning field in order to enhance the Maximum Likelihood estimation, not to mention the tensor platforms in the form of deep-learning libraries, exploiting the Automatic Differentiation advantages.

Performed Monte Carlo analysis clearly indicates that the proposed framework can be deliberated as a profitable substitute to the Bayesian interference in case of highly-dimensional space parameter estimation. On the other hand, the implemented algorithms are surprisingly costly, in both time and memory allocation, in the case of solitary parameter estimation.

The paper is organized as follows, Chapter 1 exhaustively describes the machine learning, while the next one illustrates the DSGE model that will serve as backdrop for the estimation procedures.. Next section characterizes the solution techniques, while estimation procedures and their application to DSGE models are outlined in chapter 4. Section 5 presents the Monte Carlo design and report its results. The dissertation is completed with the concluding remarks with potential extensions, references and appendices with codes in Dynare and Python.

Chapter 1

Machine learning

Machine learning (ML) permits to perform analysis of tremendous quantities of data. Moreover, it generally delivers faster and more accurate results, simultaneously requiring additional time and resources to be trained accurately. Hence, based on the assessment of its early contributions to economics, it is predicted to have a dramatic impact on the social sciences field within a foreseeable future (Athey (2018)). In the following section the definition of machine learning is granted, with a brief description of its main strengths, as well as fundamental weaknesses, contrasting them with traditional and widely implemented econometric and statistical tools. In addition, the automatic differentiation procedure is thoroughly expounded, with the emphasis imposed on its contributions to the ML optimization methods, further augmented with two elementary illustrations of forward and backward automatic differentiation techniques. Finally, as the ML framework requires a proper ecosystem of tensor computations such as PyTorch or TensorFlow, both of the above-mentioned deep learning platforms are briefly explained.

1.1. Definition

It has to be underlined at the beginning that it does not exist single and generally acceptable definition of machine learning. In the broad understanding, it refers to a collections of subfields of computer science, as well as a set of topics developed and used across computer science, engineering, statistics, and increasingly the social sciences. In the aforementioned meaning, ML is an application of artificial intelligence (AI) that provides systems able to automatically peruse and progress from experience without being explicitly programmed. On contrary, conferring to the narrower definition, machine learning is a field that develops algorithms designed to be applied to datasets, with the main areas of focus being optimization, prediction in both regression and classification instances, and clustering or grouping tasks.

Furthermore, machine learning tasks can be classified into several broad categories. As far as the supervised learning is concerned, the algorithm constructs a mathematical model from a data containing both the inputs and the desired outputs. Oppositely, in unsupervised learning, a mathematical model is built from a set of data which encloses

only inputs, without any output labels. In other words, it involves finding clusters of observations that are similar in terms of their covariates and thus can be viewed as *dimensionality reduction*. In addition, semi-supervised learning algorithms lay in-between and develop mathematical models from incomplete training data, where a chunk of the sample input lacks labels. What is more, the reinforcement machine learning algorithms interact with its environment by producing actions and discover errors or rewards, automatically determining the ideal behavior within a specific context in order to maximize its performance. Finally, the main study of the following paper is the last wide ML application: the optimization.

Machine learning intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples, which in turn expresses the divergence between the predictions of the trained model and the actual problem instances. Although it is true that machine learning optimization is a subset of optimization, there are some differences between classical optimization and machine learning (Goodfellow et al. (2016)). Namely, in most machine learning scenarios, the main concern is imposed on some performance measure, defined with respect to the test set and possibly intractable. In order to optimize it, a different cost function is introduced and then minimized, expecting that it will lead to the performance measure improvement.This stays in contrast to pure optimization, where minimizing a cost function is a goal in and of itself.

1.2. Characteristics

Machine learning and econometrics share purpose and a philosophical underpinning, incorporating data and background knowledge to induce new expertise. Such new, inferred proficiency can specify *in-sample* data generalizations (referred to as description or identification) or forecast and extrapolate into *out-of-sample* data generalizations, such as patterns or trends. To infer sound generalizations out of data both seek statistical support in the statistics body of knowledge, mainly statistic inference theory. Thus both are subject to bias, both are prone to overfitting, both suffer equally from bad quality input data. Therefore, data cleaning and preparation is essential for both to produce useful, actionable insights.

What distinguishes them is mainly scope. Econometric body of knowledge was developed and fitted to address economic (financial) data and questions. Machine learning is broader and from on-set has no commitment or attachment to any particular field of application. Hence, econometrics tends to incorporate more easily economic background knowledge, vide semi- and structured models, whereby economy theory (or expertise, or intuition) is explicitly represented and taken into account to estimate and to forecast.

On the other hand, most machine learning techniques, in particular, the most used and successful ones, cannot cater for background knowledge as swiftly and as neatly. In fact, background knowledge is more incorporated in the data preparation phase, as well as in heuristics and in technique specific parametrization such as structural topology,

thresholds or pruning decisions.

Finally, it is necessary to emphasis one particular gain for economic modeling and analysis that hails from machine learning application: a whole infrastructure to account for highly non-linear relationships. On the other hand, the conundrum between correlation and causation is more effectively unraveled in the econometric field (Boelaert and Ollion (2018)).

Before proceeding, it is useful to highlight two other contributions of the ML literature. The first one is computational rather than conceptual, but it has had such a large impact that it merits a short discussion. The technique is called stochastic gradient descent (SGD, Bottou (2010)) and it meticulously explained in the chapter 4. The second benefaction, the automatic differentiation, is the topic of the next section.

1.3. Automatic differentiation

The maximization of the likelihood function can be performed through standard optimization procedures such as L-BFGS (Zhu et al. (1997)) or stochastic gradient descent (Bottou (2010)), both of them involving the calculation of derivatives. In fact, derivatives, predominantly in the form of gradients and Hessians of the objective function, are omnipresent in machine learning (Sra et al. (2011)). Techniques for the computation of derivatives in programming languages can be classified into four main categories: (1) manually deriving derivatives and coding them; (2) numerical differentiation using finite difference approximations; (3) symbolic differentiation encompassing the expression manipulation in specifically designed computer algebra systems such as Mathematica, Maxima, and Maple; (4) automatic differentiation (AD), also called algorithmic differentiation or autodiff, which is the fundamental subject of this dissertation.

Manual differentiation is a tedious, time consuming and prone to error procedure, requiring a model to be defined as a closed-form expression, eliminating or severely confining algorithmic control flow and expressivity. Alternatively, numerical differentiation is relatively straightforward to implement but due to round-off and truncation errors it can be eminently inaccurate (Jerrell (1997)) and unsatisfactorily scalable. On the other hand, symbolic differentiation sermonizes the defficiencies of both the manual and numerical methods, but often results in complex, cryptic and similarly to the manual technique closed-form expressions.

The fourth technique, automatic differentiation, surmounts all of the above-mentioned obstacles, with its basic form consisting of applying symbolic differentiation at the elementary operation level and keeping intermediate numerical results, in lockstep with the evaluation of the main function (Baydin et al. (2018)). It can be proved that all numerical computations are ultimately compositions of a finite set of elementary operations for which derivatives are known (Griewank and Walther (2008)). One can enumerate for instance the binary arithmetic operations, the unary sign switch or transcendental functions, not to mention the exponential, the logarithm or the trigonometric functions. Moreover, the derivative of the overall composition can be easily derived from the chain rule, imposed on the derivatives of the constituent operations.

A substantial point to emphasis is that AD can differentiate not only closed-form expressions in the classical sense, but the algorithms using control flow such as branching, loops, recursion and procedure calls as well. It will be essential for the looping structure of the Linear Time Iteration algorithm, discussed in section 3.3, for which the automatic differentiation seems like an indispensable tool in further maximization process.

1.3.1. Computational graph

The basic mathematical ideas behind AD have been known for a long period of time. However, interest in AD had definitely revived by 80s' due to improvements in programming techniques and the introduction of the efficient reverse mode. A useful tool in the development of AD following 1980 was the computational graph, which is a way of visualizing an algorithm or computer program different from formulas. For instance, Fig. 1.1 illustrates a computational graph, technically known as a directed acyclic graph (DAG). Transformations of the algorithm such as differentiation in forward or reverse mode can be thus expressed as modifications of the computational graph.

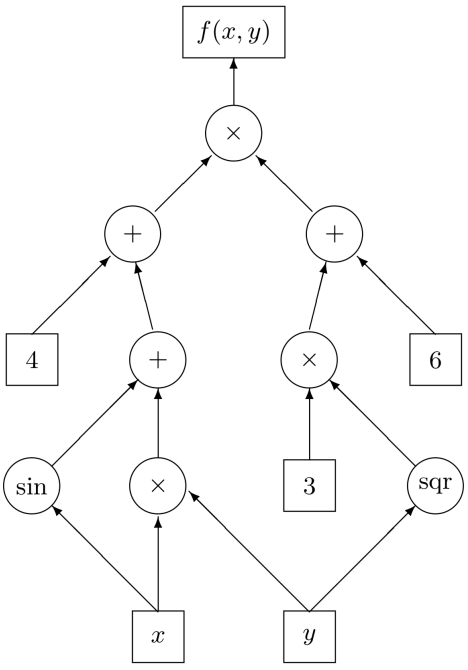

Figure 1.1: Computational graph[1]

Paraphrasing it, a computation graph is a DAG in which nodes represent variables, such as matrices, scalars or tensors, while the edges depict some mathematical operations, for instance summation or multiplication. Thus, during the optimization step, the chain

[1]Source: Bücker et al. (2006)

rule and the graph are combined in order to calculate the derivative of the output with respect to the learnable variable.

There are two different modes of automatic differentiation, based on the application of the chain rule. The next section will start with forward mode automatic differentiation, which often is find a little more intuitive, described on the simple example.

1.3.2. Forward mode

Given a function f, one can construct a computational graph (a directed acyclic one) representing it, for instance let:

$$f(x, y) = \cos(x)\sin(y) + \frac{x}{y}$$

As mentioned previously, each node of the graph expresses a primitive function, while the edges enact the flow of information:

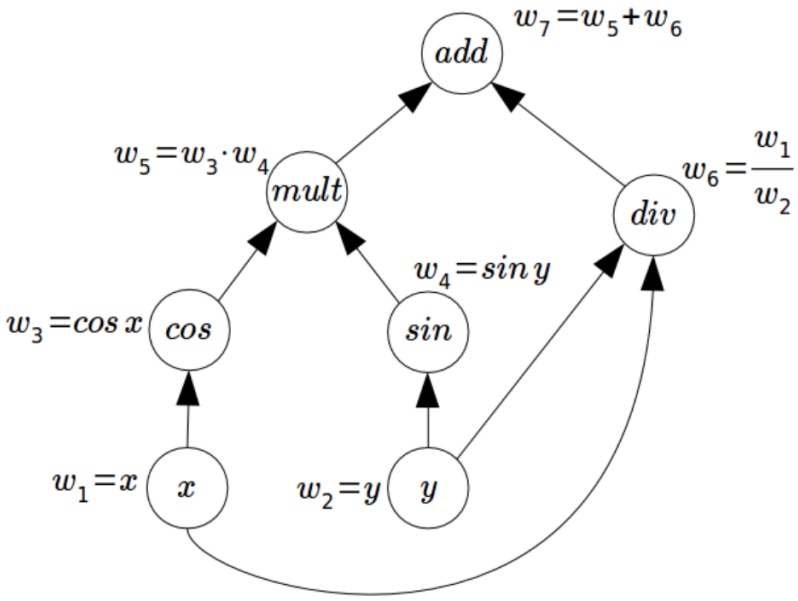

Figure 1.2: Forward propagation[2]

Forward differentiation operates by recursive computation of derivatives of nodes in terms of their parents. Suppose one wants to calculate the partial $\frac{\partial f}{\partial x}$, denoted by Df. Then:

[2]Source: http://www.columbia.edu/~ahd2125/post/2015/12/5/

$$
\begin{aligned}
Df &= Dw_7 = D(w_5 + w_6) = Dw_5 + Dw_6 \\
Dw_6 &= D\frac{w_1}{w_2} = \frac{w_1 Dw_2 - w_2 Dw_1}{w_2^2} \\
Dw_5 &= Dw_3 w_4 = w_3 Dw_4 + w_4 Dw_3 \\
Dw_4 &= D\sin(w_2) = \cos(w_2) Dw_2 \\
Dw_3 &= D\cos(w_1) = -\sin(w_1) Dw_1 \\
Dw_2 &= Dy \\
Dw_1 &= Dx
\end{aligned}
$$

The final value of Df depends only on x, y, Dw_1, Dw_2. In this case, $D = \frac{\partial f}{\partial x}$, so $Dx = 1$ and $Dy = 0$. Because information flows in the same direction as when one evaluates the expression (bottom-up), it is called the *forward mode*. Predictably, the information flows top-down in *backward mode*, also called *reverse mode*.

1.3.3. Backward mode

Reverse mode automatic differentiation lets calculate gradients much more efficiently than forward mode automatic differentiation. Suppose one wants to calculate the gradient of the f function described in the previous section:

$$
\nabla f = \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle = \left\langle \frac{\partial w_7}{\partial w_1}, \frac{\partial w_7}{\partial w_2} \right\rangle
$$

Then:

$$
\begin{aligned}
\frac{\partial w_7}{\partial w_1} &= \frac{\partial w_3}{\partial w_1}\frac{\partial w_7}{\partial w_3} + \frac{\partial w_6}{\partial w_1}\frac{\partial w_7}{\partial w_6} = -\sin w_1 \frac{\partial w_7}{\partial w_3} + \frac{1}{w_2}\frac{\partial w_7}{\partial w_6} \\
\frac{\partial w_7}{\partial w_2} &= \frac{\partial w_4}{\partial w_2}\frac{\partial w_7}{\partial w_4} + \frac{\partial w_6}{\partial w_2}\frac{\partial w_7}{\partial w_6} = \cos w_2 \frac{\partial w_7}{\partial w_4} - \frac{w_1}{w_2^2}\frac{\partial w_7}{\partial w_6} \\
\frac{\partial w_7}{\partial w_3} &= \frac{\partial w_5}{\partial w_3}\frac{\partial w_7}{\partial w_5} = w_4 \frac{\partial w_7}{\partial w_5} \\
\frac{\partial w_7}{\partial w_4} &= \frac{\partial w_5}{\partial w_4}\frac{\partial w_7}{\partial w_5} = w_3 \frac{\partial w_7}{\partial w_4} \\
\frac{\partial w_7}{\partial w_5} &= 1 \\
\frac{\partial w_7}{\partial w_6} &= 1
\end{aligned}
$$

Instead of implementing the chain rule to get derivatives of parents in terms of their children, one proceeds fully oppositely, computing derivatives of children nodes in terms

of their parents. Because information flows top-down, the reverse of normal evaluation, it is called the *reverse mode* automatic differentiation.

Notice that all operations in reverse mode differentiation are scalar operations, even though one calculates the (vector) gradient. Computing gradients via reverse mode can be thus significantly faster if f contains a vast number of input variables.

1.4. Tensor platform

PyTorch is a relatively new deep learning library, backing dynamic computation graphs and tensors which can tap into the resources of a GPU to significantly speed up matrix operations. The dynamic graph paradigm allows to make changes to the model architecture during runtime, as a graph is created only when a piece of code is run. This means a graph may be redefined during the lifetime for a program. This, however, is not possible with static graphs, used by another popular deep learning libraryTensorFlow, where graphs are created before running the program, and merely executed later.

Finally, PyTorch has in-built several optimization algorithms, such as Stochastic Gradient Descent or ADAM, which solves the likelihood maximization problem relatively costlessly.

1.4.1. GPU accelaration

A central processing unit (CPU) is essentially the brain of any computing device, effectuating the instructions of a program by enforcing control, logical, and input/output (I/O) operations. Today's CPUs have just a handful of cores, and its basic design and purpose—to process complex computations—has not really changed over the years. Essentially, they are mostly applicable for problems that require parsing through or interpreting complex logic in code. A graphical processing unit (GPU), on the other hand, has smaller-sized but many more logical cores (arithmetic logic units or ALUs, control units and memory cache) whose basic design is to process a set of simpler and more identical computations in parallel. As a result, GPU can highly accelerate the Automatic Differentiation, in fact composed of simple matrix math calculations.

Both aforementioned deep learning libraries enable the GPU computation, therefore it can considered as valuable extension.

Chapter 2

Dynamic Stochastic General Equilibrium (DSGE) model

It is generally acknowledged that a satisfactory macroeconomic model should be grounded on rational intertemporal optimizing behavior of microagents, satisfying the rational expectations assumption. Furthermore, it should reflect the imperfectly flexible wages and prices, associated with deficiencies on labor and product markets. One of the most renowned New Keynesian model satisfying the above-mentioned presumptions was developed by Smets and Wouters (2003), as the elaboration of previous work of Christiano et al. (2001).

Succinctly, it assumes that in the closed economy framework there exists a continuum of households, who supply household-specific labor in monopolistically competitive environment. In addition, they set Calvo-sticky wages, deriving utility from leisure and consumption in relation to their habits. Households save and part of their savings is transformed into capital through a household technology, which in turn has its own adjustment costs. Moreover, they rent out their capital to the intermediate-good sector and choose total amount of investment with variable capital utilization rate. The rest of their savings is spent for lending and borrowing decisions, not to mention the bond holding settlements.

On the firm side, a continuum of intermediate goods produced by firms in monopolistic competition is assumed. Their production requires various kinds of differentiated labor and (homogeneous) capital, leading to the Calvo-sticky prices setting. On the other hand, final goods are produced in perfect competition framework through intermediate goods and they are consumed by households and government, allowing for the fiscal policy and Taylor-type monetary policy rule.

Finally, there is a plethora of sources of shocks, such as two preference shocks, aggregate productivity shock or monetary policy rule shock (similarly to the **?**).

After introduction of the main essence of the Smets and Wouters (2003) model, in the following sections it will be derived meticulously.

2.1. Households

In the examined economy, there is a continuum of mass one households indexed by $\tau \in (0,1)$, maximizing an intertemporal utility function over an endless horizon:

$$\mathbb{E}_0 \left(\sum_{t=0}^{\infty} \beta^t U_t^\tau \right) \tag{2.1.1}$$

where β is the discount factor and U_t^τ stands for the contemporaneous utility function depending on consumption C_t^τ, and labor l_t^τ.

The utility function is separable in its two arguments:

$$U_t^\tau \left(C_t^\tau, l_t^\tau \right) = \varepsilon_t^B \left[\frac{1}{1-\sigma_c} \left(C_t^\tau - H_t \right)^{1-\sigma_c} - \frac{\varepsilon_t^L}{1+\sigma_l} \left(l_t^\tau \right)^{1+\sigma_l} \right] \tag{2.1.2}$$

where σ_c is the measure of relative risk aversion or the inverse of the intertemporal elasticity of substitution, while $\frac{1}{\sigma_l}$ denotes the elasticity of labor with respect to the real wage.

The above-mentioned utility function includes two shocks, namely a general preference shock ε_t^B affecting intertemporal substitution of households and a labor supply shock ε_t^L. All of them follow autoregressive processes of the first order with normally and independently distributed error terms:

$$\varepsilon_t^B = \rho_B \cdot \varepsilon_{t-1}^B + \eta_t^B, \quad \eta_t^B \sim \mathcal{N}(0,1) \tag{2.1.3}$$

$$\varepsilon_t^L = \rho_L \cdot \varepsilon_{t-1}^L + \eta_t^L, \quad \eta_t^L \sim \mathcal{N}(0,1) \tag{2.1.4}$$

The utility function includes as well a habit H_t that is proportional to the aggregate past consumption:

$$H_t = h \cdot C_{t-1} \tag{2.1.5}$$

which allows the optimal behavior of households to respond to shocks with an hump-shaped and lagged path of the optimal consumption. It has to be underlined that the habit variable depends on lagged aggregate consumption that is unaffected by any one agent's decisions.

Furthermore, the income of the households Y_t^τ takes the form of:

$$Y_t^\tau = \frac{W_t^\tau}{P_t} \cdot l_t^\tau + A_t^\tau + D_t^\tau + \left(r_t^k z_t^\tau - \Psi(z_t^\tau) \right) K_{t-1}^\tau - T_t \tag{2.1.6}$$

thus depending on real labor income $l_t^\tau \cdot \frac{W_t^\tau}{P_t}$, net cash inflow from participating in state-contingent securities A_t^τ, the dividends derived from the imperfect competitive intermediate firms D_t^τ, taxes T_t and the return on the real capital stock depreciated by the cost associated with variations in the degree of capital utilization z_t^τ. It is assumed that the cost of capital utilization is zero when capital utilization is one ($\Psi(1) = 0$) as it

would in the steady state. Framing differently, households rent capital services to firms and decide about the amount of capital to accumulate given certain capital adjustment costs. With the augmentation of the rental price of capital, the capital stock can be used more intensively according to a specific cost schedule.

Households hold their financial wealth in the form of bonds B_t, which are one-period securities with market price b_t and gross return:

$$R_t = 1 + i_t = \frac{1}{b_t} \tag{2.1.7}$$

Current income and financial wealth can be used for consumption and investment in physical capital, thus households maximize their objective function subject to an intertemporal budget constraint:

$$C_t^\tau + I_t^\tau + b_t \frac{B_t^\tau}{P_t} = Y_t^\tau + \frac{B_{t-1}^\tau}{P_t} \tag{2.1.8}$$

Finally, following **?**, it is assumed that there exist state-contingent securities that perfectly insure the households against variations in household specific labor income. As a result, the labor income plus the net cash-inflow from participating in state-contingent securities component $\frac{W_t^\tau}{P_t} \cdot l_t^\tau + A_t^\tau$ in the household's income will be equal to aggregate labor income. Analogously the consumption of an individual household will equal the aggregate consumption and the marginal utility of consumption will be identical across different types of households. Consequence, capital holdings $K_t^\tau \equiv K_t$, bond holdings $B_t^\tau \equiv B_t$ as well as firm dividends $D_t^\tau \equiv D_t$ will be indistinguishable across different types of households.

2.1.1. Consumption and savings behavior

The maximization of the objective function of the household subject to the budget constraint with respect to consumption and holdings of bonds, may be performed through the Lagrangian-based technique, yielding the first-order condition for consumption growth in the existence of external habit formation:

$$\mathbb{E}_t \left[\beta \frac{\varepsilon_{t+1}^B (C_{t+1}^\tau - h \cdot C_t)^{-\sigma_c}}{\varepsilon_t^B (C_t^\tau - h \cdot C_{t-1})^{-\sigma_c}} R_t \frac{P_t}{P_{t+1}} \right] = 1 \tag{2.1.9}$$

or equivalently:

$$\mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} R_t \frac{P_t}{P_{t+1}} \right] = 1 \tag{2.1.10}$$

where:

$$\lambda_t = \frac{\partial U_t^\tau}{\partial C_t^\tau} = \varepsilon_t^B (C_t^\tau - H_t)^{-\sigma_c} = \varepsilon_t^B (C_t^\tau - h \cdot C_{t-1})^{-\sigma_c} = \varepsilon_t^B (C_t - h \cdot C_{t-1})^{-\sigma_c} \tag{2.1.11}$$

The exact calculations for the whole chapter's derivations are performed in the Appendix A.

2.1.2. Labor supply decisions and the wage setting equation

Households differ, supply ing a differentiated type of labor. Thus, each household has a monopoly power over the supply of its labor. In other words, labor is a differentiated good, supplied by households in a monopolistic competition to the production sector. Consequently, firms consider labor supplies l_t^τ as imperfect substitutes. The aggregate labor demand L_t and the aggregate nominal wage W_t are given by the following Dixit-Stiglitz type aggregator, where $\lambda_{w,t} > 0$ is the substitution parameter:

$$L_t = \left[\int_0^1 (l_t^\tau)^{\frac{1}{1+\lambda_{w,t}}} \, d\tau \right]^{1+\lambda_{w,t}} \tag{2.1.12}$$

It can be observed that the condition $\lambda_{w,t} = 0$ would correspond to the perfect substitution of labor.

Therefore, firms minimize the cost function $\int_0^1 W_t^\tau l_t^\tau d\tau$ under the aggregate labor demand, which in turn yields:

$$\begin{aligned} L_t &= \left[\int_0^1 \left(\frac{W_t^\tau}{W_t} \right)^{\frac{-1}{\lambda_{w,t}}} L_t^{\frac{1}{1+\lambda_{w,t}}} d\tau \right]^{1+\lambda_{w,t}} \\ \iff W_t &= \left[\int_0^1 (W_t^\tau)^{\frac{-1}{\lambda_{w,t}}} \, d\tau \right]^{-\lambda_{w,t}} \end{aligned} \tag{2.1.13}$$

where the Lagrange multiplier W_t can be interpreted as well as the price of a working hour and thus a wage index.

In addition, the determination of wages follows a Calvo rule. In other words, each household τ has a (constant) probability of $1 - \varsigma_w$ in every period t to be able to set its nominal wage and therefore in each period t a fraction of $1 - \varsigma_w$ households adapts its wage. Moreover, wages of the fraction of households ς_w that cannot reoptimize are partially anchored to the inflation rate:

$$W_t^\tau = (\Pi_{t-1})^{\gamma_w} W_{t-1}^\tau = \left(\frac{P_{t-1}}{P_{t-2}} \right)^{\gamma_w} W_{t-1}^\tau \tag{2.1.14}$$

where γ_w is the degree of wage indexation, such that $\gamma_w = 0$ entails no indexation ($W_t^\tau = W_{t-1}^\tau$), while $\gamma_w = 1$ suggests a complete indexation ($W_t^\tau = \Pi_{t-1} W_{t-1}^\tau$).

After denoting by $\widetilde{W}_t$ the wage of the household which is able to reoptimize in period t, it follows:

$$W_t^{\tau} = \begin{cases} \widetilde{W}_t & \text{with probability } 1-\varsigma_w \\ (\Pi_{t-1})^{\gamma_w} W_{t-1}^{\tau} & \text{with probability } \varsigma_w \end{cases} \tag{2.1.15}$$

Therefore, the motion for the aggregate wage index W_t equation can be computed through 2.1.13:

$$\begin{aligned} (W_t)^{\frac{-1}{\lambda_{w,t}}} &= \varsigma_w \cdot \int_0^1 \left(W_{t-1} \left(\frac{P_{t-1}}{P_{t-2}} \right)^{\gamma_w} \right)^{\frac{-1}{\lambda_{w,t}}} d\tau + (1-\varsigma_w) \cdot \int_0^1 \widetilde{W}_t^{\frac{-1}{\lambda_{w,t}}} d\tau = \\ &= \varsigma_w \cdot \left[W_{t-1} \left(\frac{P_{t-1}}{P_{t-2}} \right)^{\gamma_w} \right]^{\frac{-1}{\lambda_{w,t}}} + (1-\varsigma_w) \cdot \widetilde{W}_t^{\frac{-1}{\lambda_{w,t}}} \end{aligned} \tag{2.1.16}$$

The probability for $\widetilde{W}_t$ not to re-set until period i is $(\varsigma_w)^i$, thus through the indexation the not reoptimized wage in period $t+i$ becomes:

$$\begin{aligned} W_t^{\tau} &= \widetilde{W}_t \\ W_{t+1}^{\tau} &= \left(\frac{P_t}{P_{t-1}} \right)^{\gamma_w} W_t^{\tau} = \left(\frac{P_t}{P_{t-1}} \right)^{\gamma_w} \widetilde{W}_t \\ W_{t+2}^{\tau} &= \left(\frac{P_{t+1}}{P_t} \right)^{\gamma_w} W_{t+1}^{\tau} = \left(\frac{P_{t+1}}{P_t} \right)^{\gamma_w} \left(\frac{P_t}{P_{t-1}} \right)^{\gamma_w} W_t^{\tau} = \left(\frac{P_{t+1}}{P_{t-1}} \right)^{\gamma_w} \widetilde{W}_t \\ &\vdots \\ W_{t+i}^{\tau} &= \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \widetilde{W}_t \end{aligned} \tag{2.1.17}$$

Resetting-wages households maximize their objective function subject to their budget constraint and the demand of labor, taking into consideration the fact that wages remain fixed until period i with a probability $(\varsigma_w)^i$, which yields:

$$\begin{aligned} \frac{\widetilde{W}_t}{P_t} \mathbb{E}_t \left\{ \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{P_t}{P_{t+i}} \left(\frac{l_{t+1}^{\tau} U_{t+i}^C}{1+\lambda_{w,t+i}} \right) \right\} = \\ = \mathbb{E}_t \left\{ \sum_{i=0}^{\infty} \beta^i \varsigma_w^i l_{t+1}^{\tau} U_{t+i}^l \right\} \end{aligned} \tag{2.1.18}$$

Assuming perfect flexibility of wages ($\varsigma_w = 0$), the above-mentioned equation becomes:

$$\frac{\widetilde{W}_t}{P_t} = (1+\lambda_{w,t}) \frac{U_t^L}{U_t^C} \tag{2.1.19}$$

It will be moreover assumed that the degree of substitutability $\lambda_{w,t}$ is affected by a normally and independently distributed shocks:

$$\lambda_{w,t} = \lambda_w + \eta_t^w, \quad \eta_t^w \sim \mathcal{N}\left(0, \sigma_{\eta_w}^2\right) \tag{2.1.20}$$

In the flexible-wage economy, $1 + \lambda_{w,t}$ will be the markup of real wages over the classical ratio of the marginal disutility of labor to the marginal utility of consumption. Thus, η_t^w can be regarded as a wage markup shock.

2.1.3. Capital accumulation and investment decision

Households are the proprietors of the stock of capital of the economy and rent it at the rate r_t^k to the firms of the intermediate sector. Moreover, investment in period $t-1$ increases the stock of capital in period t and households are assumed to choose the utilization rate z_t of their stock of capital. The equation of motion of capital becomes:

$$K_t = K_{t-1}(1-\tau) + I_t \left[1 - S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right)\right] \tag{2.1.21}$$

where τ is the depreciation rate, while the function S introduces in the model the adjustment costs for investment, which in turn are contingent on its rate of growth, proposed in order to avoid strong oscillations of capital. It will be furthermore assumed that for the constant growth rate of investment, as in the steady state, $S(1) = 0$ and similarly in the neighborhood of the steady state $S'(1) = 0$. Thus, adjustment costs depend solely on the second derivative such that $S''(1) > 0$. Finally, the function S is subject to a shock ε_t^I following an $AR(1)$ process:

$$\varepsilon_t^I = \rho_I \varepsilon_{t-1}^I + \eta_t^I, \quad \eta_t^I \sim \mathcal{N}(0,1) \tag{2.1.22}$$

Solving the respective maximization problem with the Lagrange multiplier in the form of $\lambda_t Q_t$ yields:

$$r_t^k = \Psi'(z_t) \tag{2.1.23}$$

$$Q_t = \mathbb{E}_t\left[\beta \frac{\lambda_{t+1}}{\lambda_t}\left(Q_{t+1}(1-\tau) + z_{t+1} r_{t+1}^k - \Psi(z_{t+1})\right)\right] \tag{2.1.24}$$

$$\begin{aligned}
&\mathbb{E}_t\left[-\beta^t \lambda_t - \beta^t \lambda_t Q_t\left(-1 + S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right) + I_t \cdot S'\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right) \cdot \frac{\varepsilon_t^I}{I_{t-1}}\right)\right] = \\
&= \mathbb{E}_t\left[\beta^{t+1} \lambda_{t+1} Q_{t+1}\left(I_{t+1} \cdot S'\left(\frac{\varepsilon_{t+1}^I I_{t+1}}{I_t}\right) \cdot \frac{-\varepsilon_{t+1}^I I_{t+1}}{I_t^2}\right)\right]
\end{aligned} \tag{2.1.25}$$

In other words the capital utilization rate is set so that the revenue r_t^k of the marginal utilization equals the marginal costs $\Psi'(z_t)$, the today's real value of the stock of capital $\lambda_t Q_t$ is balanced with the expected value of sum of not depreciated stock of capital of the next period $\lambda_{t+1} Q_{t+1}(1-\tau)$ and the expected revenue of the future utilization $z_{t+1} r_{t+1}^k$ minus the related costs $\Psi(z_{t+1})$, while the costs of the marginal investment (included the adjustment costs) have to be identical to the expected marginal revenue of investment.

Additionally, $\lambda_t Q_t$ can be regarded as the marginal value of capital. Thus, basing on the fact that λ_t break evens the marginal utility of consumption, Q_t can be viewed as marginal Tobin-Q (the ratio of the marginal value of capital by virtue of the increment of the capital stock over the marginal opportunity cost):

$$Q_t = \frac{\lambda_t Q_t}{\lambda_t} \tag{2.1.26}$$

At the steady-state $Q = 1$, thus marginal revenues and costs mutually reimburse. If $Q > 1$ there are enticements to upturn the capital accumulation through investment, while if $Q < 1$ the reverse holds true.

2.2. Firms

The economy produces only one final product Y_t and a continuum of intermediates goods defined over $j \in [0,1]$, such that firm j produces the intermediate product y_t^j . Firms of the final product sector operate in a (perfect) competitive market, while firms in the intermediate sector function in a monopolistic competition and set their prices forward-looking.

2.2.1. Final product sector

For the production of the final product Y_t firm use the intermediate products y_t^j as input, with aggregation performed through the following technology:

$$Y_t = \left[\int_0^1 \left(y_t^j \right)^{\frac{1}{1+\lambda_{p,t}}} dj \right]^{1+\lambda_{p,t}} \tag{2.2.1}$$

Define p_t^j as the price of the intermediate product y_t^j , then the representative firm minimizes the cost function $\int_0^1 p_t^j y_t^j dj$ of a particular combination of inputs observing the constraint 2.2.1. Solving the minimization problem through Lagrangian, with P_t as the Lagrange multiplier, yields:

$$Y_t = \left[\int_0^1 \left(\frac{p_t^j}{P_t} \right)^{\frac{-1}{\lambda_{p,t}}} Y_t^{\frac{1}{1+\lambda_{p,t}}} dj \right]^{1+\lambda_{p,t}}$$

$$\iff P_t = \left[\int_0^1 \left(p_t^j \right)^{\frac{-1}{\lambda_{p,t}}} dj \right]^{-\lambda_{p,t}} \tag{2.2.2}$$

where P_t can be interpreted as the price index of the final good sector, while $\lambda_{p,t}$ is a stochastic parameter representing the mark-up of the good market (or a cost-push shock), such that:

$$\lambda_{p,t} = \lambda_p + \eta_t^p, \quad \eta_t^p \sim \mathcal{N}\left(0, \sigma_{\eta_t^p}^2\right) \tag{2.2.3}$$

2.2.2. Intermediate product sector

Each firm j of the intermediate good sector minimizes its labour and capital costs $W_t L_{j,t} + r_t^k \widetilde{K}_{j,t}$, observing its Cobb-Douglas production function:

$$y_t^j = \varepsilon_t^a \widetilde{K}_{j,t}^{\alpha} L_{j,t}^{1-\alpha} - \Phi \tag{2.2.4}$$

where $\widetilde{K}_{j,t}$ is the effective utilized stock of capital given by $\widetilde{K}_{j,t} = z_t K_{j,t-1}$, $L_{j,t}$ is the index of the differently utilized labour force, while Φ are the fixed costs. Moreover, it is assumed that the aggregate productivity shock ε_t^a follows a $AR(1)$ process such that:

$$\varepsilon_t^a = \rho_a \varepsilon_{t-1}^a + \eta_t^a, \quad \eta_t^a \sim \mathcal{N}(0,1) \tag{2.2.5}$$

Each firm j has a market power in the market of its good and hence it maximizes its expected profit π_t^j with respect to p_t^j:

$$\pi_t^j = \underbrace{p_t^j y_t^j}_{\text{Revenues}} - \underbrace{MC_t \left(y_t^j + \Phi \right)}_{\text{Costs}} = \left(p_t^j - MC_t \right) \underbrace{\left(\frac{p_t^j}{P_t} \right)^{\frac{-(1+\lambda_{p,t})}{\lambda_{p,t}}} Y_t}_{y_t^j} - MC_t \Phi \tag{2.2.6}$$

where marginal costs takes the form:

$$MC_t = \frac{1}{\varepsilon_t^a} W_t^{1-\alpha} \left(r_t^k \right)^{\alpha} \left(\alpha^{-\alpha} (1-\alpha)^{-(1-\alpha)} \right) \tag{2.2.7}$$

The profits of intermediate goods firms are assumed to be returned as dividends D_t.

In order to perform the maximization task, it uses a discount factor ($\beta^k \rho_{t+k}$) that is based on the fact that firms belong to households. From the Euler-equation of consumption, it holds:

$$\beta^k \rho_{t+k} = \beta^k \frac{\lambda_{t+k}}{\lambda_t P_{t+k}} \tag{2.2.8}$$

Finally, the determination of prices also follows a Calvo-rule in order to model the nominal price rigidity. Each firm j is allowed in period t to peg its nominal price with probability ς_p. Since firms are defined over a continuum between 0 and 1, prices that are updated in each period are $1 - \varsigma_p$ and prices that are not updated amount to ς_p. Prices of firms that was unable to re-set their prices are partially indexed by the past inflation rate Π_{t-1}:

$$p_t^j = (\Pi_{t-1})^{\gamma_p} p_{t-1}^j = \left(\frac{P_{t-1}}{P_{t-2}}\right)^{\gamma_p} p_{t-1}^j \tag{2.2.9}$$

where γ_p is the degree of price indexation, such that $\gamma_p = 0$ entails no indexation ($p_t^j = p_{t-1}^j$), while $\gamma_p = 1$ suggests a complete indexation ($p_t^j = \Pi_{t-1} p_{t-1}^j$).

After denoting by $\widetilde{p}_t$ nominal price of the firm that is able in period t to reoptimize, it follows:

$$p_t^j = \begin{cases} \widetilde{p}_t & \text{with probability } 1 - \varsigma_p \\ (\Pi_{t-1})^{\gamma_p} p_{t-1}^j & \text{with probability } \varsigma_p \end{cases} \tag{2.2.10}$$

The equation of motion for the aggregated price index P_t can be obtained through 2.2.2:

$$\begin{aligned} (P_t)^{\frac{1}{-\lambda_{p,t}}} &= \varsigma_p \int_0^1 \left(P_{t-1} \left(\frac{P_{t-1}}{P_{t-2}}\right)^{\gamma_p} \right)^{\frac{-1}{\lambda_{p,t}}} dj + (1 - \varsigma_p) \int_0^1 (\widetilde{p}_t)^{\frac{-1}{\lambda_{p,t}}} dj = \\ &= \varsigma_p \left[P_{t-1} \left(\frac{P_{t-1}}{P_{t-2}}\right)^{\gamma_p} \right]^{\frac{-1}{\lambda_{p,t}}} + (1 - \varsigma_p) (\widetilde{p}_t)^{\frac{-1}{\lambda_{p,t}}} \end{aligned} \tag{2.2.11}$$

The optimal price $\widetilde{p}_t$ set in t has a probability of ς_p^i not to be modified until period $t + i$. Furthermore, through the indexation procedure it holds that:

$$
\begin{aligned}
p_t^j &= \widetilde{p}_t \\
p_{t+1}^j &= \left(\frac{P_t}{P_{t-1}}\right)^{\gamma_p} p_t^j = \left(\frac{P_t}{P_{t-1}}\right)^{\gamma_p} \widetilde{p}_t \\
p_{t+2}^j &= \left(\frac{P_{t+1}}{P_t}\right)^{\gamma_p} p_{t+1}^j = \left(\frac{P_{t+1}}{P_t}\right)^{\gamma_p} \left(\frac{P_t}{P_{t-1}}\right)^{\gamma_p} \widetilde{p}_t = \left(\frac{P_{t+1}}{P_{t-1}}\right)^{\gamma_p} \widetilde{p}_t \\
&\vdots \\
p_{t+i}^j &= \left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_p} \widetilde{p}_t
\end{aligned}
\tag{2.2.12}
$$

Firms that are not allowed to change their prices in period t, maximize their profit function 2.2.6 under the constraint 2.2.12, which yields:

$$
\mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \frac{\lambda_{t+i}}{\lambda_t} y_{t+i}^j \left\{ \frac{\widetilde{p}_t}{P_t} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_p}}{\frac{P_{t+i}}{P_t}} \right] - (1 + \lambda_{p,t+i}) \frac{MC_{t+i}}{P_{t+i}} \right\} = 0 \tag{2.2.13}
$$

Assuming perfect flexibility of prices ($\varsigma_p = 0$), the above-mentioned equation becomes:

$$
\widetilde{p}_t = (1 + \lambda_{p,t}) \cdot MC_t \tag{2.2.14}
$$

The optimal price is set so that firms impose a premium over the average marginal costs.

2.3. The log-linearised DSGE model

In the following section, the log-linearised version of the DSGE model adopted in this paper is described. The hat symbol above the variable will denote the log deviations from its steady state. The dynamics of aggregate consumption is seized by:

$$
R_t + \mathbb{E}_t \hat{\varepsilon}_{t+1}^B - \frac{\sigma_c}{1-h} \left(\mathbb{E}_t \hat{C}_{t+1} - h \hat{C}_t \right) = \hat{\varepsilon}_{t+1}^B - \frac{\sigma_c}{1-h} \left(\hat{C}_t - h \hat{C}_{t-1} \right) + \mathbb{E}_t \hat{\Pi}_{t+1} \tag{2.3.1}
$$

For h = 0 one obtains the usual Euler equation, nevertheless in the presence of the external habit, present consumption depends on both the expected consumption of the next period and on consumption of the last period. The elasticity of consumption with respect to interest rates depends both on the intertemporal elasticity of substitution σ_c and on the degree of the habit h. A higher h reduces the effect of a variation of interest rate on consumption holding σ_c constant.

In addition, through rearranging terms, one obtains an explicit expression for $\hat{C}_t$:

$$\hat{C}_t = \frac{h}{1+h}\hat{C}_{t-1} + \frac{1}{1+h}\mathbb{E}_t\hat{C}_{t+1} - \frac{1-h}{(1+h)\sigma_c}\left(\hat{R}_t - \mathbb{E}_t\hat{\Pi}_{t+1}\right) + \frac{1-h}{(1+h)\sigma_c}\left(\hat{\varepsilon}_t^B - \mathbb{E}_t\hat{\varepsilon}_{t+1}^B\right) \tag{2.3.2}$$

Furthermore, the linearization of the investment equation yields:

$$\hat{I}_t = \frac{1}{1+\beta}\hat{I}_{t-1} + \frac{\beta}{1+\beta}\mathbb{E}_t\hat{I}_{i+1} + \frac{1}{S''(1)(1+\beta)}\hat{Q}_t + \frac{\beta\mathbb{E}_t\hat{\varepsilon}_{t+1}^I - \hat{\varepsilon}_t^I}{1+\beta} \tag{2.3.3}$$

The dependency of the optimal investment of past values introduce the dynamic in this equation. In addition, a positive shock of the adjustment costs function reduces investment and hence it can be considered as a negative investment shock.

Due to the steady state condition $\beta = \frac{1}{1-\tau+r^k}$, the linearization of the Q-equation yields:

$$\hat{Q}_t = -\left(\hat{R}_t - \mathbb{E}\hat{\pi}_{t+1}\right) + \frac{1-\tau}{1-\tau+r^k}\mathbb{E}_t\hat{Q}_{t+1} + \frac{r^k}{1-\tau+r^k}\mathbb{E}_t\hat{r}_{t+1}^k + \eta_t^Q \tag{2.3.4}$$

An equity premium shock η_t^Q, normally and independently distributed, is added in order to take into account the effect of changes of the external refinancing premium over the the capital stock value.

The linearization of the equation of the capital accumulation yields:

$$\hat{K}_t = (1-\tau)\hat{K}_{t-1} + \tau\hat{I}_{t-1} \tag{2.3.5}$$

Through the partial indexation, it follows that the linearized prices equation depends not only on the present marginal costs and on the expectations of the future inflation but also on the past inflation rate.

$$\hat{\pi}_t = \frac{\beta}{1+\beta\gamma_p}\mathbb{E}_t\hat{\pi}_{t+1} + \frac{\gamma_p}{1+\beta\gamma_p}\hat{\pi}_{t-1} + \frac{(1-\beta\varsigma_p)(1-\varsigma_p)}{(1+\beta\gamma_p)\varsigma_p}\left(\underbrace{\alpha\hat{r}_t^k + (1-\alpha)\hat{w}_t - \hat{\varepsilon}_t^a + \eta_t^p}_{\hat{mc}_t}\right) \tag{2.3.6}$$

For $\gamma_p = 0$ one obtains the usual forward-looking Phillips curve.

Similarly one obtains the linearized equation of motion for the real wage:

$$\begin{aligned}\hat{w}_t = {} & \frac{\beta}{1+\beta}\mathbb{E}_t\hat{w}_{t+1} + \frac{1}{1+\beta}\hat{w}_{t-1} + \frac{\beta}{1+\beta}\mathbb{E}_t\hat{\pi}_{t+1} - \frac{1+\beta\gamma_w}{1+\beta}\hat{\pi}_t + \\ & + \frac{\gamma_w}{1+\beta}\hat{\pi}_{t-1} - \frac{(1-\beta\varsigma_w)(1-\varsigma_w)}{(1+\beta)\left(1+\frac{(1+\lambda_w)\sigma_l}{\lambda_w}\right)\varsigma_p}\left[\hat{w}_t - \sigma_l\hat{L}_t - \frac{\sigma_c}{1-h}\left(\hat{C}_t - h\hat{C}_{t-1}\right) - \hat{\varepsilon}_t^L - \hat{\eta}_t^w\right]\end{aligned} \tag{2.3.7}$$

The real wage is a function of expected and past real wages and the expected, current and past inflation rate if $\gamma_w > 0$. There is a negative effect of the deviation of the actual real wage from one that would abound in a flexible labour market.

The linearized demand of labor is:

$$\hat{L}_t = -\hat{w}_t + \left(1 + \frac{\Psi'(1)}{\Psi''(1)}\right) \hat{r}_t^k + \hat{K}_{t-1} \tag{2.3.8}$$

or equivalently:

$$\hat{L}_t = -\hat{w}_t + (1 + \Psi)\, \hat{r}_t^k + \hat{K}_{t-1} \tag{2.3.9}$$

where $\Psi = \frac{\Psi'(1)}{\Psi''(1)}$ is the inverse of the elasticity of the capital utilization cost function. In other words, given a capital stock, the demand of labor negatively depends on real wages.

Furthermore, the good market is in equilibrium if the production is equal to the public expense G_t , the household demand of consumption and investment plus related costs:

$$Y_t = C_t + G_t + I_t + \Psi(z_t) K_{t-1} \tag{2.3.10}$$

where government consumption G_t is financed by the lump sum taxation:

$$G_t = T_t \tag{2.3.11}$$

In the steady state deviations:

$$\hat{Y}_t = (1 - \tau k_y - g_y)\, \hat{C}_t + \tau k_y \hat{I}_t + \varepsilon_t^G \tag{2.3.12}$$

where k_y and g_y denote the steady state proportions of capital and public spending on output. On the other hand, from the production function:

$$\hat{Y}_t = \vartheta \hat{\varepsilon}_t^a + \vartheta \alpha \hat{K}_{t-1} + \vartheta \alpha \frac{\Psi'(1)}{\Psi''(1)} \hat{r}_t^k + \vartheta (1 - \alpha) \hat{L}_t \tag{2.3.13}$$

where $1 + \vartheta$ is the proportion of fix costs of the production. Moreover, it is assumed that the shock of the public spending ε_t^G follows a $AR(1)$ process:

$$\varepsilon_t^G = \rho_G \varepsilon_{t-1}^G + \eta_t^G, \quad \eta_t^G \sim \mathcal{N}(0, 1) \tag{2.3.14}$$

Analogously, the capital market is in equilibrium if the demand (of firms of the intermediate sector) of capital goods equals the supply of households:

$$\int_0^1 K_{j,t-1}\, dj = K_{t-1} \tag{2.3.15}$$

while the labour market is in equilibrium if the demand of labour of firms equals the supply of households at wages set by households:

$$L_t = \left[\int_0^1 (l_t^\tau)^{\frac{1}{1+\lambda_{w,t}}}\, d\tau\right]^{1+\lambda_{w,t}} \tag{2.3.16}$$

Finally, the monetary policy follows a generalized Taylor Rule:

$$\begin{aligned}\hat{R}_t =& \rho\hat{R}_{t-1} + (1-\rho)\left[\bar{\pi}_t + r_\pi\left(\hat{\pi}_{t-1} - \bar{\pi}_t\right) + r_y\hat{Y}_t^{\text{gap}}\right] + \\ &+ r_{\Delta_\pi}\left(\hat{\pi}_t - \hat{\pi}_{t-1}\right) + r_{\Delta_Y}\left(\hat{Y}_t^{\text{gap}} - \hat{Y}_{t-1}^{\text{gap}}\right) + \eta_t^R\end{aligned} \tag{2.3.17}$$

The Taylor rule is obtained by minimizing the central bank loss function subject to the Phillips curve and the IS curve. The central bank loss function is generally set to be equal to a weighted sum of the squared difference between the actual and the target inflation and the squared output gap. The output gap $\hat{Y}_t^{\text{gap}}$ is the difference between the actual and the potential output. The potential output is the production that theoretically the economy performs under perfect flexibility of prices and wages.

The central bank reacts to deviations of both the lagged inflation with respect to the target inflation (normalized to 0) and of the lagged output gap. The monetary policy is subject to two different kinds of shock: a shock of the target inflation process $\bar{\pi}_t = \rho_\pi\bar{\pi}_{t-1} + \eta_t^\pi$ and shock of the interest rate η_t^R. The central smooths the interest rates by setting the present rate as a weighted average of the past interest rate and the optimal rate as a function of the above quantities.

To sum up, in order to calculate the results, two systems have to be created: one is the flexible scheme without price stickiness, wage stickiness or three cost-push shocks, on contrary the other one should be viewed as the sticky system where prices and wages follow a Calvo mechanisms. Then the potential output obtained from the flexible system is implemented to compute the output gap in the Taylor rule. In each system, there are eight endogenous variables (capital, consumption, investment, inflation, wages, output, interest rate and real capital stock) and two state variables (labor and return on capital).

Chapter 3

Solution

Since Blanchard and Kahn (1980) a significant amount of substitutable approaches for solving linear rational expectations models have materialized, not to mention the methods of Uhlig et al. (1998), Klein (2000) or Sims (2002). Economists adopt then their output for simulating and estimating models, calculating impulse response functions and asymptotic covariance, solving infinite horizon linear quadratic control problems or finally constructing terminal constraints for nonlinear models. Despite the fact that for models satisfying the Blanchard and Kahn (1980) conditions, the algorithms provide equivalent solutions, three algorithms will be presented in depth: the approach of Klein (2000) for the illustrational purposes, the Uhlig et al. (1998) toolkit for the actual implementation in Python and the Linear Time Iteration algorithm, whose differentiability will be substantial for the Automatic Differentiation application.

3.1. Klein (2000) approach

DSGE model consists of state variables (x_t), control variables (y_t), transition equations for structural shocks and measurement errors ($\mathbf{u}_t$) and a vector of deep parameters θ, which can be injected into a nonlinear first-order system of expectational difference equations:

$$\mathbb{E}_t f\left(x_{t+1}, y_{t+1}, x_t, y_t \middle| \theta\right) = 0 \tag{3.1.1}$$

Solution is then characterized by transition equations for state and control variables, so-called *policy-functions*, that solve (at least approximately) the aforementioned system of equations:

$$\begin{cases} x_{t+1} &= \widetilde{h}\left(x_t \middle| \theta\right) + \eta_x\left(\theta\right)\mathbf{u}_{t+1} \\ y_t &= \widetilde{g}\left(x_t \middle| \theta\right) \end{cases} \tag{3.1.2}$$

Unfortunately, in the majority of cases the analytical derivation of g and h is extremely difficult or even impossible. One therefore distinguishes between linear (such as in Klein

(2000) or Sims (2002)) and non-linear methods (for instance Schmitt-Grohé and Uribe (2004) or Gomme and Klein (2011)) of local approximation of the policy functions in a neighborhood of a precise point (mostly steady-state). The fundamental idea is to introduce the perturbation parameter σ that captures the stochastic nature of the model. The general perturbation framework (in the spirit of Schmitt-Grohé and Uribe (2004)) is as follows:

$$\left\{ \begin{array}{rl} 0 & = \mathbb{E}_t f\left(x_{t+1}, y_{t+1}, x_t, y_t \middle| \theta\right) \\ x_{t+1} & = h\left(x_t, \sigma \middle| \theta\right) + \sigma \eta_x \mathbf{u}_{t+1} \\ y_t & = g\left(x_t, \sigma \middle| \theta\right) \end{array} \right. \tag{3.1.3}$$

with $\mathbb{E}(\mathbf{u}_t) = 0$ and $\mathbb{E}(\mathbf{u}_t \mathbf{u}_t^{'}) = \Sigma_{\mathbf{u}}$. The non-stochastic steady state is then defined as:

$$\left\{ \begin{array}{rl} 0 & = f\left(\bar{x}, \bar{y}, \bar{x}, \bar{y} \middle| \theta\right) \\ \bar{y} & = h\left(\bar{x}, 0 \middle| \theta\right) \\ \bar{y} & = g\left(\bar{x}, 0 \middle| \theta\right) \end{array} \right. \tag{3.1.4}$$

where exogenous shocks and measurement errors are stacked into $\mathbf{u}_t$ for convenience.

As far as the first-order approximations of g and h around the point $(x_t, \sigma) = (\bar{x}, 0)$ are concerned:

$$\left\{ \begin{array}{l} g(x_t, \sigma) = g(\bar{x}, 0) + g_x(\bar{x}, 0)(x_t - \bar{x}) + g_\sigma(\bar{x}, 0)(\sigma - 0) \\ h(x_t, \sigma) = h(\bar{x}, 0) + h_x(\bar{x}, 0)(x_t - \bar{x}) + h_\sigma(\bar{x}, 0)(\sigma - 0) \end{array} \right. \tag{3.1.5}$$

Substituting into the model:

$$f^e(x_t, \sigma) = \mathbb{E}_t f\left(\underbrace{h(x_t, \sigma) + \sigma \mathbf{u}_{t+1}}_{x_{t+1}}, \underbrace{g\left(\overbrace{h\left(x_t, \sigma\right) + \sigma \mathbf{u}_{t+1}}^{x_{t+1}}, \sigma \right)}_{y_{t+1}}, x_t, \underbrace{g(x_t, \sigma)}_{y_t} \right) \tag{3.1.6}$$

After noticing that f and its derivatives are 0 when assessed at the non-stochastic steady state $(\bar{x}, 0)$, taking derivative of f^e with respect to σ and evaluating at the non-stochastic steady state yields:

$$\begin{aligned} f^e_\sigma(\bar{x}, 0) &= E_t f_1 [h_\sigma + \mathbf{u}_{t+1}] + E_t f_2 [g_x (h_\sigma + \mathbf{u}_{t+1}) + g_\sigma] + f_3 \cdot 0 + f_4 g_\sigma = 0 \\ &\iff \underset{(n \times n_x)}{\begin{bmatrix} f_1 + f_2 g_x & f_2 + f_4 \end{bmatrix}} \underset{(n_y \times 1)}{\begin{bmatrix} h_\sigma \\ g_\sigma \end{bmatrix}} = \underset{(n \times 1)}{0} \end{aligned} \tag{3.1.7}$$

with $f_1 = \frac{\partial f(\bar{x},0)}{\partial x_{t+1}}, f_2 = \frac{\partial f(\bar{x},0)}{\partial y_{t+1}}, f_3 = \frac{\partial f(\bar{x},0)}{\partial x_t,} f_4 = \frac{\partial f(\bar{x},0)}{\partial y_t}$.

There are n equations in n unknowns in a linear and homogenous system, hence in the case of unique solution, it must be:

$$\begin{cases} h_\sigma &= \underset{(n_x \times 1)}{0} \\ g_\sigma &= \underset{(n_y \times 1)}{0} \end{cases} \tag{3.1.8}$$

Furthermore, taking derivative of f^e with respect to x_t and evaluating at the non-stochastic steady state yields:

$$\begin{aligned} & f_x^e(\overline{x}, 0) = f_1 h_x + f_2 g_x h_x + f_3 + f_4 g_x = 0 \\ \iff & \underbrace{-\begin{bmatrix} \underset{(n \times n_x)}{f_1} & \underset{(n \times n_y)}{f_2} \end{bmatrix}}_{:=A} \begin{bmatrix} \underset{(n_x \times n_x)}{h_x} \\ \underset{(n_y \times n_x)(n_x \times n_x)}{g_x \quad h_x} \end{bmatrix} = \underbrace{\begin{bmatrix} \underset{(n \times n_x)}{f_3} & \underset{(n \times n_y)}{f_4} \end{bmatrix}}_{:=B} \begin{bmatrix} \underset{(n_x \times n_x)}{I} \\ \underset{(n_y \times n_x)}{g_x} \end{bmatrix} \end{aligned} \tag{3.1.9}$$

Therefore there are $n \times n_x$ equations for $n \times n_x$ unknown elements of both h_x and g_x. One can additionally multiply both sided by the deviation from the steady state $\widehat{x}_t := (x_t - \overline{x})$, what yields:

$$A \begin{bmatrix} h_x \widehat{x}_t \\ g_x h_x \widehat{x}_t \end{bmatrix} = B \begin{bmatrix} \widehat{x}_t \\ g_x \widehat{x}_t \end{bmatrix} \tag{3.1.10}$$

It is useful to discern that the coefficient matrices are tantamount to the first order approximation:

$$A \mathbb{E}_t \begin{bmatrix} \widehat{x}_{t+1} \\ \widehat{y}_{t+1} \end{bmatrix} = B \begin{bmatrix} \widehat{x}_t \\ \widehat{y}_t \end{bmatrix} + \begin{bmatrix} \sigma \eta_x \mathbf{u}_{t+1} \\ 0 \end{bmatrix} \tag{3.1.11}$$

Alternatively, appropriate stacking leads to the more concise formulation (up to scaling factor):

$$A \mathbb{E}_t X_{t+1} = B X_t + \mathbf{u}_t \tag{3.1.12}$$

If A and B are known matrices, then undoubtedly inverting A yields solution for h_x and g_x. Moreover, A is generally singular and non-invertible. Klein (2000) approach to approximate the first order solution employing the generalized Schur decomposition is therefore implemented, in order to uncouple system into a (block) triangular system of equations and solve system recursively.

3.1.1. Matrix theory

Definition 1. *Let A and B be two square $(n \times n)$ matrices. The set of all matrices of the form $A - \lambda B$ with $\lambda \in \mathbb{C}$ is forenamed to be a* ***pencil****. The eigenvalues of the pencil are elements of the set $\lambda(A, B)$ characterized by:*

$$\lambda(A, B) = \{z \in \mathbb{C} : det(A - zB) = 0\}$$

Definition 2. *Let A and B be two square $(n \times n)$ matrices. Then $\lambda \in \lambda(A, B)$ is forenamed a* ***generalized eigenvalue*** *if there exists a nonzero vector $q \in \mathbb{C}^n$ such that*

$$Aq = \lambda Bq$$

It is advantageous to detect that if $B = I$, then the above-mentioned intricacy simplifies to the ordinary eigenvalue problem: $Aq = \lambda q$. Moreover, the n eigenvalues of A can be ordered (generally in more than one way) to form an $n \times n$ diagonal matrix Λ and a corresponding matrix of nonzero columns Q that satisfies the eigenvalue equation:

$$AQ = Q\Lambda \tag{3.1.13}$$

Theorem 1 (Spectral decomposition)**.** *Assume that A is a square $(n \times n)$ matrix with n linearly independent columns. Then A can be factorized as:*

$$AQ = Q\Lambda \Leftrightarrow A = Q\Lambda Q^{-1}$$

where Λ is a diagonal matrix such that $\lambda_i = \Lambda_{ii}$ is the eigenvalue of A related to the eigenvector q_i gathered in column i of the squared $(n \times n)$ matrix Q.

The proofs for the following section can be found for instance in Roman (2008) or Brown (1988).

Theorem 2 (Generalized (complex) Schur decomposition or QZ decomposition)**.** *Let A and B be square $(n \times n)$ matrices. Then there exist matrices Q, Z, T and S such that*

$$Q^*AZ = S \Leftrightarrow A = QSZ^*$$
$$Q^*BZ = T \Leftrightarrow B = QTZ^*$$

1. *Q and Z are unitary, i.e. $Q^*Q = QQ^* = I_n$ and $Z^*Z = ZZ^* = I_n$.*
2. *S and T are upper triangular.*
3. *Pairs (s_{ii}, t_{ii}) can be arranged in any desired order.*
4. *If for some i, t_{ii} and s_{ii} are both zero, then $\lambda(A, B) = \mathbb{C}$. Otherwise: $\lambda(A, B) = \left\{\frac{s_{ii}}{t_{ii}} : t_{ii} \neq 0\right\}$*

The interest in the following paper will be limited to the case $\lambda(A, B) \neq \mathbb{C}$ and unit roots will not be allowed, that is t_{ii} and s_{ii} are not both zero, as well as $|t_{ii}| \neq |s_{ii}|$.

3.1.2. Schur decomposition application

Returning to the problem of finding g_x and h_x:

$$AE_t \begin{bmatrix} \widehat{x}_{t+1} \\ \widehat{y}_{t+1} \end{bmatrix} = B \begin{bmatrix} \widehat{x}_t \\ \widehat{y}_t \end{bmatrix} \tag{3.1.14}$$

The Schur decompositions of A and B are given by:

$$\begin{cases} Q^*A = SZ^* \\ Q^*B = TZ^* \end{cases} \tag{3.1.15}$$

where the following order is chosen: the stable generalized eigenvalues ($|s_{ii}| > |t_{ii}|$) come first.

Multiplying both sides by Q^* and using:

$$\begin{bmatrix} s_t \\ {\scriptstyle n_x \times 1} \\ u_t \\ {\scriptstyle n_y \times 1} \end{bmatrix} := Z^* \begin{bmatrix} \widehat{x}_t \\ {\scriptstyle n_x \times 1} \\ \widehat{y}_t \\ {\scriptstyle n_y \times 1} \end{bmatrix} \tag{3.1.16}$$

one gets:

$$S \begin{bmatrix} E_t s_{t+1} \\ E_t u_{t+1} \end{bmatrix} = T \begin{bmatrix} s_t \\ u_t \end{bmatrix} \tag{3.1.17}$$

where S and T are upper triangular, such that:

$$\begin{bmatrix} S_{11} & S_{12} \\ {\scriptstyle n_x \times n_x} & {\scriptstyle n_x \times n_y} \\ 0 & S_{22} \\ {\scriptstyle n_y \times n_x} & {\scriptstyle n_y \times n_y} \end{bmatrix} \begin{bmatrix} E_t s_{t+1} \\ E_t u_{t+1} \end{bmatrix} = \begin{bmatrix} T_{11} & T_{12} \\ {\scriptstyle n_x \times n_x} & {\scriptstyle n_x \times n_y} \\ 0 & T_{22} \\ {\scriptstyle n_y \times n_x} & {\scriptstyle n_y \times n_y} \end{bmatrix} \begin{bmatrix} s_t \\ u_t \end{bmatrix} \tag{3.1.18}$$

Solving the lower block first:

$$S_{22} E_t[u_{t+1}] = T_{22} u_t \tag{3.1.19}$$

Note that due to the ordering $|s_{ii}| < |t_{ii}|$ (or $|s_{ii}| \leq |t_{ii}|$). Using backward-substitution, it can be proved, that each solution with bounded mean and variance must satisfy $u_t = 0$ for all t (unless $\Sigma = 0$), otherwise the solution explodes. In other words, as many state variables as there are generalized eigenvalues with $|s_{ii}| > |t_{ii}|$ are needed. This property has a name:

Definition 3 (Blanchard and Kahn (1980) conditions)**.** *The number of generalized eigenvalues, that are in absolute terms greater than 1, must be equal to the number of state variables, in order to get a stable solution (saddle-path).*

Furthermore, given the fact that $u_t = 0$, both x_t and y_t from the definition of s_t and u_t become:

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \begin{bmatrix} s_t \\ u_t \end{bmatrix} \Rightarrow \begin{cases} x_t = Z_{11} s_t \\ y_t = Z_{21} s_t \end{cases} \tag{3.1.20}$$

If Z_{11} is non-degenerate, then:

$$y_t = \underbrace{Z_{21} Z_{11}^{-1}}_{=g_x} x_t \tag{3.1.21}$$

Thus, in order to compute g_x a nonsingular Z_{11} is needed. Assuming that S_{11} is invertible, the first block yields:

$$E_t[s_{t+1}] = S_{11}^{-1} T_{11} s_t \tag{3.1.22}$$

Plugging in $s_t = Z_{11}^{-1} x_t$ one gets:

$$E_t[x_{t+1}] = \underbrace{Z_{11} S_{11}^{-1} T_{11} Z_{11}^{-1}}_{=h_x} x_t \tag{3.1.23}$$

Thus, in order to compute h_x, nonsingular S_{11} and Z_{11} are required. However, S_{11} has full rank by construction. The Klein (2000) algorithm can be abridged in the following manner:

Data: Nonlinear first-order system of expectational difference equations;
Result: DSGE solution;
Calculate the generalized Schur decomposition of:

$$\begin{cases} A = -\begin{bmatrix} f_1 & f_2 \end{bmatrix} \\ B = \begin{bmatrix} f_3 & f_4 \end{bmatrix} \end{cases}$$

Reorder generalized eigenvalues such that $|s_{ii}| > |t_{ii}|$ in the upper left;
if *Blanchard and Kahn (1980) condition fulffiled* **then**
  if Z_{11} *invertible* **then**
    Compute $h_x = Z_{11} S_{11}^{-1} T_{11} Z_{11}^{-1}$ and $g_x = Z_{21} Z_{11}^{-1}$;
  else
    Infeasible solution;
  end
else
  Infeasible solution;
end

Algorithm 1: Klein (2000) algorithm

Note that the squareness of Z_{11} is referred to in the literature as the Blanchard and Kahn (1980) order condition, while the non-singularity of Z_{11} as the Blanchard and Kahn (1980) rank condition.

3.2. Uhlig et al. (1998) toolkit

The Uhlig et al. (1998) algorithm likewise employs generalized eigenvalue calculations to glean a solution for the matrix polynomial equation, based on the method of undetermined coefficients. The method is applied to systems written as:

$$\begin{aligned} 0 &= U_1 x_t + U_2 x_{t-1} + U_3 y_{t-1} + U_4 z_{t-1} \\ 0 &= \mathbb{E}_t \left[U_5 x_{t+1} + U_6 x_t + U_7 x_{t-1} + U_8 y_t + U_9 y_{t-1} + U_{10} z_t + U_{11} z_{t-1} \right] \\ z_t &= U_{12} z_{t-1} + \mathbf{u}_t \end{aligned} \tag{3.2.1}$$

where x_t are the endogenous state variables, y_t the endogenous control (also called *jump*) variables, while z_t denotes the exogenous variables. Moreover, the apostrophe stands for the leading operator. Alternatively, in the stacked formulation:

$$A \mathbb{E}_t X_{t+1} + B X_t + C X_{t-1} + \mathbf{u}_t = 0 \tag{3.2.2}$$

where:

$$X_t = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \tag{3.2.3}$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ U_5 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.2.4}$$

$$B = \begin{bmatrix} U_1 & 0 & 0 \\ U_6 & U_8 & U_{10} \\ 0 & 0 & I \end{bmatrix} \tag{3.2.5}$$

$$C = \begin{bmatrix} U_2 & U_3 & U_4 \\ U_7 & U_9 & U_{11} \\ 0 & 0 & -U_{12} \end{bmatrix} \tag{3.2.6}$$

$$\tag{3.2.7}$$

Assuming that the decision rules can be expressed as:

$$\begin{aligned} x_t &= \tilde{U}_1 x_{t-1} + \tilde{U}_2 z_{t-1} \\ y_t &= \tilde{U}_3 x_t + \tilde{U}_4 z_t \end{aligned} \tag{3.2.8}$$

one can substitute the decision rules into the system:

$$\begin{aligned} 0 &= (U_1\tilde{U}_1 + U_2 + U_3\tilde{U}_3)x_{t-1} + (U_1\tilde{U}_2 + U_3\tilde{U}_4 + U_4)z_{t-1} \\ 0 &= (U_6\tilde{U}_1 + U_7 + U_8\tilde{U}_3\tilde{U}_1 + U_9\tilde{U}_3)x_{t-1} + (U_6\tilde{U}_2 + U_8\tilde{U}_3\tilde{U}_2 + U_8\tilde{U}_4U_{12} + U_9\tilde{U}_4 + U_{11})z_{t-1} \end{aligned} \tag{3.2.9}$$

Since these equations must hold for any x_{t-1} and z_{t-1}:

$$0 = U_1\tilde{U}_1 + U_2 + U_3\tilde{U}_3 \tag{3.2.10}$$

$$0 = U_1\tilde{U}_2 + U_3\tilde{U}_4 + U_4 \tag{3.2.11}$$

$$0 = U_6\tilde{U}_1 + U_7 + U_8\tilde{U}_3\tilde{U}_1 + U_9\tilde{U}_3 \tag{3.2.12}$$

$$0 = U_6\tilde{U}_2 + U_8\tilde{U}_3\tilde{U}_2 + U_8\tilde{U}_4U_{12} + U_9\tilde{U}_4 + U_{11} \tag{3.2.13}$$

Solving 3.2.10 for $\tilde{U}_3$:

$$\tilde{U}_3 = -U_3^{-1}\left(U_1\tilde{U}_1 + U_2\right) \tag{3.2.14}$$

Substituting it for $\tilde{U}_3$ into equation 3.2.12:

$$-U_8U_3^{-1}U_1\tilde{U}_1^2 - \left(U_8U_3^{-1}U_2 + U_9U_3^{-1}U_1 - U_6\right)\tilde{U}_1 - \left(U_9U_3^{-1}U_2 - U_7\right) = 0 \quad (3.2.15)$$

The above-mentioned quadratic matrix equation in $\tilde{U}_1$ can be then solved by generalized Schur decomposition. Next, solving for $\tilde{U}_4$ as the function of $\tilde{U}_2$ with 3.2.11:

$$\tilde{U}_4 = -U_3^{-1}\left(U_1\tilde{U}_2 + U_4\right) \quad (3.2.16)$$

Substituting further in 3.2.13

$$\begin{aligned} &U_6\tilde{U}_2 + U_8\tilde{U}_3\tilde{U}_2 - U_8U_3^{-1}(U_1\tilde{U}_2 + U_4)U_{12} - U_9U_3^{-1}(U_1\tilde{U}_2 + U_4) + U_{11} = 0 \\ &\quad \Rightarrow \left(U_6 + U_8\tilde{U}_3 - U_9U_3^{-1}U_1\right)\tilde{U}_2 - U_8U_3^{-1}U_4U_{12} - U_9U_3^{-1}U_4 + U_{11} = 0 \quad (3.2.17) \end{aligned}$$

Matrix $\tilde{U}_2$ is the only unknown, but it is not trivial, as $\tilde{U}_2$ is sandwiched. Nevertheless, one can use the Kronecker product $\otimes$ such that $vec\,(XYZ) = \left(Z^T \otimes X\right) vec(Y)$, applied to 3.2.13:

$$\begin{aligned} 0 &= I \otimes \left(U_6 + U_8\tilde{U}_3 - U_9U_3^{-1}U_1\right) vec(\tilde{U}_2) - U_{12}^T \otimes \left(U_8U_3^{-1}U_1\right) vec(\tilde{U}_2) + \quad (3.2.18) \\ &\quad - vec\left(U_8U_3^{-1}U_4U_{12} - U_9U_3^{-1}U_4 + U_{11}\right) \\ \Rightarrow vec(\tilde{U}_2) &= \left[I \otimes \left(U_6 + U_8\tilde{U}_3 - U_9U_3^{-1}U_1\right) - U_{12}^T \otimes \left(U_8U_3^{-1}U_1\right)\right]^{-1} \cdot \quad (3.2.19) \\ &\quad \cdot vec\left(U_8U_3^{-1}U_4U_{12} - U_9U_3^{-1}U_4 + U_{11}\right) \quad (3.2.20) \end{aligned}$$

It has to be underlined that for the Uhlig et al. (1998) procedure, one must choose *jump* variables to maintain the full rank of U_3 matrix, namely augment the system with a *dummy variable* and respective equations.

Solution of Uhlig et al. (1998) toolkit is a state space system: a transition equation supplemented with a measurement equation, equivalent to Klein (2000) solution (Bonaldi et al. (2010)).

3.3. Linear Time Iteration procedure

Uhlig et al. (1998) procedure provided the foundations for the techniques to solve a Smets and Wouters (2003) model. Nevertheless, due to the implementation of non-differentiable QZ-algorithm, its usefulness for the machine learning frameworks exploiting the benefits of automatic differentiation is questionable. Therefore, the Linear Time Iteration Procedure of Rendahl (2017) is proposed as an alternative, yielding identical results to the Uhlig et al. (1998) approach.

The model of interest is once again given by:

$$AX_{t+1} + BX_t + CX_{t-1} + \mathbf{u}_t = 0 \tag{3.3.1}$$

where X_t is an $n \times 1$ vector containing endogenous and exogenous variables, $\mathbf{u}_t$ is an $n \times 1$ vector of mean-zero disturbances, and A, B and C are conformable matrices.

Similarly, one assumes the recursive solutions in form:

$$X_t = FX_{t-1} + G\mathbf{u}_t \tag{3.3.2}$$

Inserting into the preceding equation:

$$CX_{t-1} + BX_t + AFX_t + \mathbf{u}_t = 0 \tag{3.3.3}$$

Thus, the matrix Q is trivially defined by:

$$G = -\left(B + AF\right)^{-1} \tag{3.3.4}$$

Moreover, F must be the solution to the quadratic matrix equation:

$$C + BF + AF^2 = 0 \tag{3.3.5}$$

Undoubtedly, if F is known, finding Q is a trivial operation, therefore one should focus on finding F. Recall that it is sufficient to solve the deterministic part of the problem in order to retrieve it:

$$CX_{t-1} + BX_t + AX_{t+1} = 0 \tag{3.3.6}$$

Suppose one has a candidate solvent: F_n. Then the solution of the succeeding problem for X_t:

$$CX_{t-1} + BX_t + AF_nX_t = 0 \tag{3.3.7}$$

is trivially given by:

$$X_t = -\left(B + AF_n\right)^{-1} CX_{t-1} \tag{3.3.8}$$

Thus, the initial guess should be updated according to:

$$F_{n+1} = -\left(B + AF_n\right)^{-1} C \tag{3.3.9}$$

Furthermore, the author proves that the above-mentioned procedure converges to the DSGE solution under the Blanchard and Kahn (1980) conditions and provides an enhanced technique to deal with common situations in which solvents contain eigenvalues that are zero.

In addition, ADD citation provides the simple criterion, equivalent to the Blanchard and Kahn (1980) conditions, to determine whether the obtained solution is stable and unique:

$$\begin{cases} \lambda(F) & < 1 \\ \lambda\left((CF+B)^{-1}C\right) & < 1 \end{cases}$$

where $\lambda(\cdot)$ denotes the spectral radius of a linear operator (the norm of its biggest eigenvalue).

It is worth noticing that the stability implies not only the long-term *determinancy* (the model does not explode), but also the *predictability* (also called *non-chaoticity*), meaning that the small deviations disappear over time.

Chapter 4

Estimation

In the previous section the linear rational expectations model was solved by several different techniques, with the emphasis imposed on the Linear Time Iteration algorithm due to its differentiability, resulting in a state equation in the predetermined state variables. As the fundamental interest of the paper is to invoke the machine learning techniques to the DSGE model estimation, the formerly derived model should be written in the state space form, by adding a measurement equation linking the three observable variables (labor, consumption, investment) to the vector of state variables. Moreover, the likelihood function should be calculated using the Kalman filter, whose maximization by the Stochastic Gradient Descent leads to the parameters estimation. All three aforementioned components are described in the following sections.

4.1. Kalman filter

With unobservable state variables, one can exploit the recursive nature of the model and its fully-specified laws of motion to apply filters like the one proposed by Kalman (1960). Namely the solved model, augmented by the measurement equation, in the state space representation is as follows:

$$X_t = F_t X_{t-1} + G_t \mathbf{u}_t, \quad t \geqslant 0 \tag{4.1.1}$$

$$Y_t = H_t X_t + \mathbf{v}_t \tag{4.1.2}$$

where $\mathbf{u}_t$ and $\mathbf{v}_t$ are independent, zero-mean Gaussian white processes with covariances:

$$\mathbb{E}\left(\mathbf{u}_t \mathbf{u}_d^T\right) = \Sigma_t^{\mathbf{u}} \iota_{td} \tag{4.1.3}$$

$$\mathbb{E}\left(\mathbf{v}_t \mathbf{v}_d^T\right) = \Sigma_t^{\mathbf{v}} \iota_{td} \tag{4.1.4}$$

Moreover, the initial state X_0 is assumed to be Gaussian random variable, independent of the noise processes, with:

$$X_0 \sim \mathcal{N}\left(\bar{X}_0, P_0\right) \tag{4.1.5}$$

In addition, let $\tilde{Y}_t = (Y_0, \ldots, Y_t)$. The goal is to compute recursively the following optimal estimator ox X_t:

$$\hat{X}_t^+ \equiv \hat{X}_{t|t} = \mathbb{E}(X_t|\tilde{Y}_t) \tag{4.1.6}$$

Define as well the *one-step predictor* of X_t:

$$\hat{X}_t^- \equiv \hat{X}_{t|t-1} = \mathbb{E}(X_t|\tilde{Y}_{t-1}) \tag{4.1.7}$$

and the respective covariance matrices:

$$P_t^+ \equiv P_{t|t} = \mathbb{E}\left[\left(X_t - \hat{X}_t^+\right)\left(X_t - \hat{X}_t^+\right)^T |\tilde{Y}_t\right] \tag{4.1.8}$$

$$P_t^- \equiv P_{t|t-1} = \mathbb{E}\left[\left(X_t - \hat{X}_t^+\right)\left(X_t - \hat{X}_t^+\right)^T |\tilde{Y}_{t-1}\right] \tag{4.1.9}$$

Finally denote:

$$P_0^- = P_0 \tag{4.1.10}$$

$$\hat{X}_t^- = \bar{X}_0 \tag{4.1.11}$$

From the normality, it follows that the conditional random variable $(X_t|\tilde{Y}_d)$ for any t, d. In particular:

$$(X_t|\tilde{Y}_t) \sim \mathcal{N}\left(\hat{X}_t^+, P_t^+\right) \tag{4.1.12}$$

$$(X_t|\tilde{Y}_{t-1}) \sim \mathcal{N}\left(\hat{X}_t^-, P_t^-\right) \tag{4.1.13}$$

In order to derive the Kalman filter, suppose, at time t, that $\left(\hat{X}_t^-, P_t^-\right)$ is given. Then $\left(\hat{X}_t^+, P_t^+\right)$ and $\left(\hat{X}_{t+1}^-, P_{t+1}^-\right)$ shall be computes in two steps.

Measurement update step: Since $Y_t = H_t X_t + \mathbf{v}_t$, then the conditional vector $\left(\binom{X_t}{Y_t}|\tilde{Y}_{t-1}\right)$ is Gaussian, with mean:

$$\begin{bmatrix} \hat{X}_t^- \\ H_t\hat{X}_t^- \end{bmatrix} \tag{4.1.14}$$

and covariance:

$$\begin{bmatrix} P_t^- & P_t^- H_t^T \\ H_t P_t^- & M_t \end{bmatrix} \tag{4.1.15}$$

where:

$$M_t = H_t P_t^- H_t^T + \Sigma_t^{\mathbf{u}} \tag{4.1.16}$$

To compute $\left(X_t|\tilde{Y}_t\right)$, it is sufficient to apply the formula for conditional expectation of Gaussian random variable, with everything pre-conditioned on $\tilde{Y}_{t-1}$. In other words, it follows that $\left(X_t|\tilde{Y}_t\right)$ is Gaussian with mean:

$$\hat{X}_t^+ = \mathbb{E}(X_t|\tilde{Y}_t) = \hat{X}_t^- + P_t^- H_t^T \left(M_t\right)^{-1} \left(Y_t - H_t \hat{X}_t^-\right) \tag{4.1.17}$$

and covariance:

$$P_t^+ = \mathbb{C}(X_t|\tilde{Y}_t) = P_t^- - P_t^- H_t^T \left(M_t\right)^{-1} H_t P_t^- \tag{4.1.18}$$

Time update step: Recall that $X_t = F_t X_{t-1} + G_t \mathbf{u}_t$ and that X_t and $\mathbf{u}_t$ are independent given $\tilde{Y}_t$. Therefore:

$$\hat{X}_{t+1}^- = \mathbb{E}(X_{t+1}|\tilde{Y}_t) = F_t \hat{X}_t^+ \tag{4.1.19}$$

$$P_{t+1}^- = \mathbb{C}(X_{t+1}|\tilde{Y}_t) = F_t P_t^+ F_t^T + G_t \Sigma_t^{\mathbf{v}} G_t^T \tag{4.1.20}$$

4.2. Maximum likelihood

Since, by construction, the forecast error from the Kalman filter is serially uncorrelated and normally distributed, it is relatively straightforward to deliver the log-likelihood (Ireland (2004)). What is more, the obtained likelihood is differentiable, which enables the implementation of Automatic Differentiation in the maximization process, which in turn yields consistent and asymptotically normal estimates.

The Kalman filter allows the construction of inferences about the unobserved state vector and permits the evaluation of the joint likelihood function of observable endogenous variables. Moreover, the so-obtained likelihood becomes a differentiable functions of its parameters, allowing to implement the machine learning techniques described in Chapter 1.

Nevertheless, as pointed out by Fernández-Villaverde (2010), maximizing the likelihood of the DSGE models, which in fact is a complicated and highly-dimensional function, is a

tedious task. Moreover, due to the sparsity of the macroeconomics, usually quarterly data, as well as he flexibility of DSGE models in generating similar behavior with relatively different combination of parameter values, the likelihoods of DSGE models are full of local extrema and of virtually flat surfaces, which further overburden the optimization. Finally, the standard errors of the estimates are notably problematic to calculate and their asymptotic distribution inadequately approximates the small sample one. To deal some of the above-mentioned obstacles, the Bayesian interference can be introduced, as in the estimation of the Smets and Wouters (2003).

4.3. Stochastic Gradient Descent

The computational ramification of learning algorithm becomes the decisive restricting factor when one envisions large scale machine learning problems. One of the most esteemed optimization algorithms is called Stochastic Gradient Descent (SGD, Robbins and Monro (1951)).

Assume that for each example $z = (x, y)$, consisting of an arbitrary input x and scalar output y, one can construct the *loss function* $l(\hat{y}, y)$ that evaluates the cost of predicting $\hat{y}$ when the actual value is y. The aim is to identify a function $f \in \mathcal{F}$ that minimizes the loss $L(z, w) = l(f_w, y)$ averaged on the examples, where f_w is the permissible function parametrized by a weight vector w. In fact, one desires to average over the unknown distribution $d\mathbb{P}(z)$:

$$\mathbb{E}(f) = \int l(f(x), y) d\mathbb{P}(z) \tag{4.3.1}$$

when only the average on the sample $z_1, \ldots, z_n$ is available:

$$\mathbb{E}_n = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i), y_i) \tag{4.3.2}$$

The *empirical risk* $\mathbb{E}_n$ measures the training set performance. The *expected risk* $\mathbb{E}(f)$ regulates the generalization performance, defined as the expected performance on future examples. The statistical learning theory (Vapnik and Chervonenkis (2015)) countenances minimizing the empirical risk instead of the expected one when the selected family $\mathcal{F}$ is sufficiently restrictive.

It has repeatedly been suggested (Rumelhart et al. (1985)) to minimize the empirical risk $\mathbb{E}(f_w)$ using Gradient Descent (GD). Each iteration updates the weights w on the basis of the gradient of $\mathbb{E}_n(f_w)$:

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w L(z_i, w_t) \tag{4.3.3}$$

where γ is a competently chosen learning rate. Under sufficient regularity assumptions, when the initial estimate w_0 is close enough to the optimum and when the learning rate γ is amply modest, this algorithm accomplishes linear convergence (Dennis Jr and Schnabel (1996)).

The Stochastic Gradient Descent (SGD) algorithm is a drastic simplification. Instead of computing the gradient of $\mathbb{E}_n(f_w)$ exactly, each iteration estimates it on the basis of a single randomly picked example z_i:

$$w_{t+1} = w_t - \gamma \nabla_w L(z_i, w_t) \tag{4.3.4}$$

As a result, SGD is far less computationally demanding than GD. On the other hand, although SGD minimizes loss faster, it is noisier and it oscillates around the the minimum giving some variation in accuracy and loss run to run (Bottou (2012)).

Chapter 5

Results

The small-sample properties of Maximum Likelihood estimator augmented with machine learning are studied here using Monte Carlo analysis .All experiments are based on 100 replications using a sample size of 200 observations, corresponding to the set of quarterly observations of the series for a period of 50 years. In order to restrict the impact of the initial values used to generate the series, 100 extra observations were produced in every single replication. Then, for the estimation part, the starting 100 observations were repealed.

In order to limit the computational burden in the Monte Carlo experiments, the author concentrates on up to five (of the thirty two) model parameters, namely discount rate, Calvo prices stickiness, standard deviation of government spending shock, capital income share and depreciation rate, keeping the rest fixed. The parameters values used for the simulation are presented in Table 5.1.

Table 5.1: True parameters values

Behavioral parameters

ψ	Investment adjustment cost	1/0.169
σ_L	Labour utility	2.4
σ_c	Consumption utility	1.353
h	Consumption habit	0.573
ϕ	Fixed cost	1.408
adj	Capital utilization adjustment cost	1/6.771
β	Discount factor	0.99

Wage and prices parameters

γ_p	Indexation prices	0.469
γ_w	Indexation wages	0.763
ς_p	Calvo prices	0.908
ς_w	Calvo wages	0.737

Monetary policy function parameters

r_π	Inflation	1.684
$r_{d\pi}$	Difference of inflation	0.14
r_Y	Output	0.099
r_{dY}	Difference of output	0.159
λ_w	Calvo employment	0.5
ρ	Lagged interest rate	0.961

Persistence of shocks

ρ_L	Labour supply shock	0.889
ρ_a	Productivity shock	0.823
ρ_b	Consumption preference shock	0.855
ρ_G	Government spending shock	0.949
ρ_π	Inflation objective shock	0.924
ρ_I	Investment shock	0.927

Innovations of the shock

sd_R	Interest rate shock	0.081
sd_p	Price mark-up shock	0.16
sd_w	Wage mark-up shock	0.289
sd_Q	Equity premium shock	0.604

Other parameters

τ	Depreciation rate	0.025
α	Capital income share	0.3
r_k	Rental rate of capital	0.078
k	Steady state capital-output ratio	8.8
g	Steady state government spending-output ratio	0.18

Moreover, as the maximum likelihood algorithm requires the initial guess for the estimated parameters, the draws from following intervals, widely accepted in the DSGE literature, are considered:

Table 5.2: Initial guesses of parameters values

Parameter	Description	Lower bound	Upper bound
β	Discount factor	0.90	0.99
ς_p	Calvo prices	0.90	0.99
ρ_G	Government spending shock	0.90	0.99
α	Capital income share	0.2	0.4
τ	Depreciation rate	0.025	0.05

Moreover, due to the suspicion of the existence of local maxima, the multiple starting points are selected during each iteration and the final value is chosen as one maximizing the obtained likelihoods.

5.1. Preeliminary results

Before conducting the main Monte Carlo analysis, the author decided to test the algorithm on the estimation of one specific parameter: the discount factor. Being more specific, it is useful to test the dependance on the initial guess matrix in the Linear Time Iteration algorithm, with particular emphasis imposed on the duration of the execution and its closeness to the real parameter value. Moreover, it is beneficial to verify the reliance on the performed number of optimization steps, as well as the amount of starting values used in the calculations.

Table 5.3: Beta estimation stress testing

F matrix	n_{start}	n_{opt}	β	Std. Error	Time
Zeros	100	100	0.9893	3.9111e-09	11 100s ($\sim$ 3h)
Zeros	1000	100	0.9893	3.8888e-09	113 100s ($\sim$ 31h)
Zeros	100	1000	0.9894	2.9755e-09	75 600s ($\sim$ 24h)
Correct F	100	100	0.9894	6.2042e-09	7 800s ($\sim$ 2h)
Correct F	1000	100	0.9894	4.6190e-09	77 100s ($\sim$ 22h)
Correct F	100	1000	0.9896	3.8645e-09	79 400s ($\sim$ 22h)

Where the columns denote respectively guessed matrix F, number of starting points, number of optimization steps, β estimate, standard error and execution time.

The maximization of likelihood function during the scenarios listed in Table ?? points out that even the elementary scheme with 100 optimization steps and exactly the same number of starting points can lead to the acceptable estimate. Moreover, the results

clearly promotes the implementation of F matrix with true parameters as the initial guess in the Linear Time Iteration algorithm, thus the same technique is adopted in the Monte Carlo analysis.

On the other hand, the standard errors, also reported in Table ??, correspond to the square roots of the diagonal elements of negative of the inverted matrix of second derivatives of the maximized log-likelihood function. Generally, calculating these standard errors requires two steps, numerically evaluating the matrix of second derivatives of the log-likelihood function and then inverting that very large matrix having elements of varying magnitudes, both of which may introduce approximation error into the statistics (Iskrev (2008)). In spite of the fact that the automatic differentiation procedure provides the Hessian matrix relatively costlessly, significantly reducing the computational tine, its inversion still may be problematic. Hence, these standard errors, though useful, do need to be interpreted with a bit of caution.

Finally, increasing both the amount of optimization steps and the number of starting points lead directly to the better estimate, simultaneously dramatically increasing the computational time. Therefore, the elementary scheme (100, 100) should be firstly considered to be adopted in the Monte Carlo analysis. Nonetheless, in furtherance of performing at least 100 Monte Carlo trials, the amount of starting points is diminished to 10.

5.2. Monte Carlo results

In the following section the results of 100 Monte Carlo trials of estimation of the DSGE model in different scenarios are illustrated. Similar scheme to the previous chapter is the first to consider, namely 100 Monte Carlo trials of discount factor estimation performed with 10 starting points and 100 optimization steps.

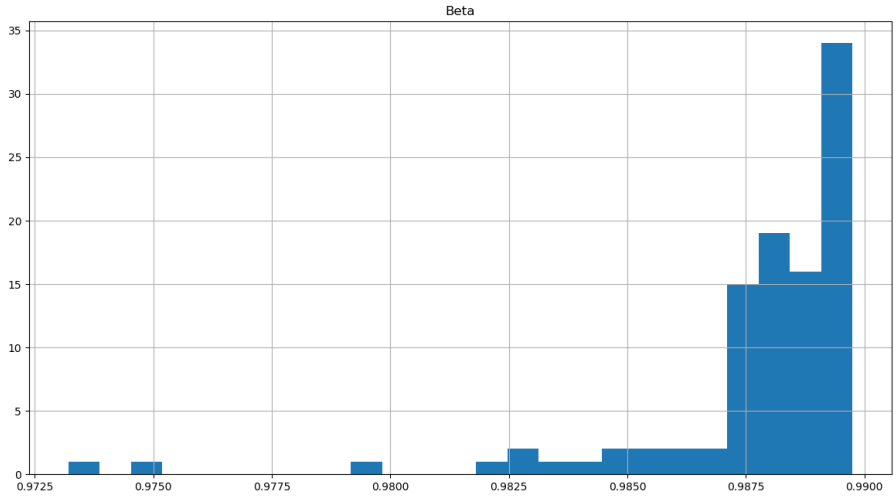

Figure 5.1: Histogram of one parameter estimation

The histogram 5.1 confirms that the implemented estimation method lead to the relevant results, nevertheless as can be observed in the upper part of Table 5.1 it demands a significant portion of time and computational power.

As the next step, the Monte Carlo trials with identical framework are performed, trying to estimate three and five parameters at once respectively. It can be observed that the β estimate is further from the true value, pointing out that the convergence in higher dimensional space is much more difficult to obtain in the same amount of steps. Moreover, the obtained values are wider spread, as can be deducted from the inspection of Figure 5.2. Similarly, the average estimates of two other parameters, namely ς_p and ρ_G are far from optimal, while the computational time increases significantly.

Table 5.4: Results of estimation

Single parameter estimation

Parameter	True value	Estimation	Std. Error	Time
β	0.99	0.9878	1.5356e-09	750s ($\sim$ 0.20h)

Three parameters estimation

Parameter	True value	Estimation	Std. Error	Time
β	0.99	0.9872	2.4556e-09	900s ($\sim$ 0.25h)
ς_p	0.908	0.9650	7.3073e-09	
ρ_G	0.949	0.9759	4.6946e-09	

Five parameters estimation

Parameter	True value	Estimation	Std. Error	Time
β	0.99	0.9833	2.6776e-09	1500s ($\sim$ 0.30 h)
ς_p	0.908	0.9674	8.0667e-09	
ρ_G	0.949	0.9678	5.8213e-09	
α	0.3	0.3983	1.8797e-09	
τ	0.025	0.0309	1.3171e-09	

Where the columns denote respectively the parameter name, mean estimate, average standard error and average execution time (per one Monte Carlo iteration).

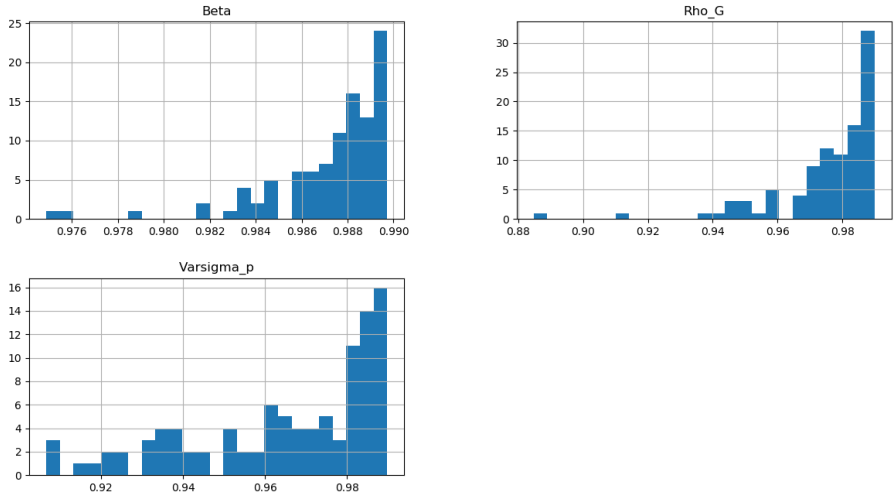

Figure 5.2: Histograms of three parameters estimation

Finally, the simultaneous estimation of five parameters confirms all of the above-mentioned findings. Particularly important is the time dimension, which significant in absolute value, does not increases linearly. In other words, one may assume that the estimation procedure outlined in the paper should be especially considered in the high-dimensional estimation problems.

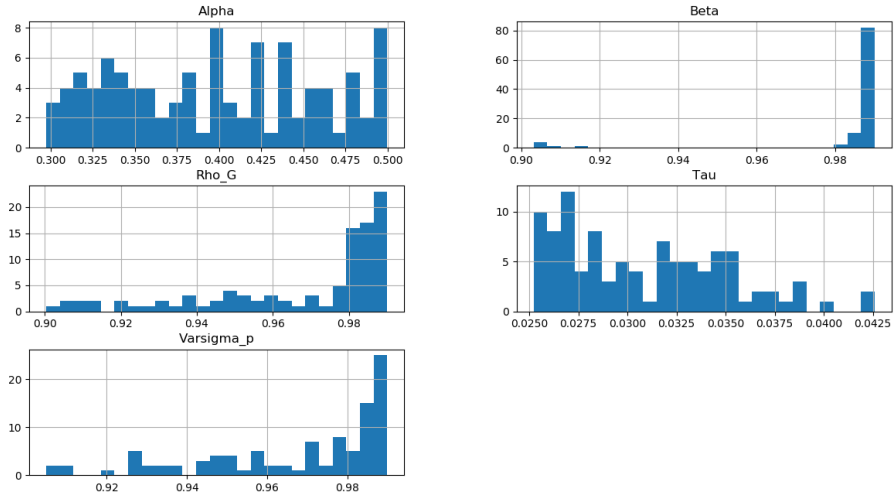

Figure 5.3: Histograms of five parameters estimation

It has to be accentuated that the choice of observable variables employed in Maximum Likelihood estimation matters (Ruge-Murcia (2007)). Therefore replication of the performed exercise with different set of observables may be seen as a potential extension. Nevertheless, the obtained results clearly indicates that the Machine-Learning-enhanced Maximum Likelihood estimation of the DSGE model should be considered as a helpful alternative to the Bayesian frameworks only in the case of highly-dimensional problems, with a plethora of parameters to estimate.

Chapter 6

Concluding remarks

With the quick advance on computers verified on the past few years, Dynamic Stochastic General Equilibrium framework became an important tool in macro-analysis. Moreover, recently one can observe an expeditious developments in the Machine Learning domain. Therefore the paper aimed to implement the progressive achievements of ML to the medium-size DSGE model estimation.

Performed Monte Carlo analysis evidently demonstrated that the proposed framework can be cogitated as an effective substitute to the Bayesian interference in case of highly-dimensional space parameter estimation. Undoubtedly, the combination of Machine Learning and DSGE models has a great perspectives of growth in the near future.However, there are important gaps and uncertainties that remain to be solved.

Thus, the code implementation on the Graphics Processing Unit can be regarded as a valuable extension, potentially disentangling the speed and memory allocation issues. In other words, rewriting code to be run on the GPU may significantly boost the algorithms performance even 10 times (Michalakes and Vachharajani (2008)). Moreover, the Maximum Likelihood estimation may surpass the classical Bayesian framework in the case of higher approximations of the solution function, incorporating non-linearities.

Bibliography

Stéphane Adjemian, Houtan Bastani, Michel Juillard, Ferhat Mihoubi, George Perendia, Marco Ratto, and Sébastien Villemot. Dynare: Reference manual, version 4. 2011.

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

Susan Athey. The impact of machine learning on economics. In *The Economics of Artificial Intelligence: An Agenda*. University of Chicago Press, 2018.

Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

Olivier Jean Blanchard and Charles M Kahn. The solution of linear difference models under rational expectations. *Econometrica: Journal of the Econometric Society*, pages 1305–1311, 1980.

Julien Boelaert and Étienne Ollion. The Great Regression. *Revue française de sociologie*, 59(3):475–506, 2018.

Pietro Bonaldi et al. *Identification problems in the solution of linearized DSGE models*. Citeseer, 2010.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

William Clough Brown. *A second course in linear algebra*. Wiley-Interscience, 1988.

H Martin Bücker, George Corliss, Paul Hovland, Uwe Naumann, and Boyana Norris. *Automatic differentiation: applications, theory, and implementations*, volume 50. Springer Science & Business Media, 2006.

Lawrence J Christiano, Martin Eichenbaum, and Charles L Evans. Nominal rigidities and the dynamic effects of a shock to monetary policy. *Journal of Political Economy*, 2001.

John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam, 1996.

Jesús Fernández-Villaverde. The econometrics of DSGE models. *SERIEs*, 1(1-2):3–49, 2010.

Paul Gomme and Paul Klein. Second-order approximation of dynamic models without the use of tensors. *Journal of Economic Dynamics and Control*, 35(4):604–615, 2011.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

Peter N Ireland. A method for taking models to the data. *Journal of Economic dynamics and control*, 28(6):1205–1226, 2004.

Nikolay Iskrev. Evaluating the information matrix in linearized DSGE models. *Economics Letters*, 99(3):607–610, 2008.

Max E Jerrell. Automatic differentiation and interval arithmetic for estimation of disequilibrium models. *Computational Economics*, 10(3):295–316, 1997.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

Paul Klein. Using the generalized Schur form to solve a multivariate linear rational expectations model. *Journal of Economic Dynamics and Control*, 24(10):1405–1423, 2000.

John Michalakes and Manish Vachharajani. Gpu acceleration of numerical weather prediction. *Parallel Processing Letters*, 18(04):531–548, 2008.

Sendhil Mullainathan and Jann Spiess. Machine learning: an applied econometric approach. *Journal of Economic Perspectives*, 31(2):87–106, 2017.

Pontus Rendahl. Linear Time Iteration. 2017.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

Steven Roman. Graduate Texts in Mathematics, Advanced Linear Algebra, 2008.

Francisco J Ruge-Murcia. Methods to estimate dynamic stochastic general equilibrium models. *Journal of Economic Dynamics and Control*, 31(8):2599–2636, 2007.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Stephanie Schmitt-Grohé and Martın Uribe. Solving dynamic general equilibrium models using a second-order approximation to the policy function. *Journal of Economic Dynamics and Control*, 28(4):755–775, 2004.

Christopher A Sims. Solving linear rational expectations models. *Computational economics*, 20(1):1–20, 2002.

Frank Smets and Rafael Wouters. An estimated Dynamic Stochastic General Equilibrium model of the euro area. *Journal of the European Economic Association*, 1(5):1123–1175, 2003.

Frank Smets and Rafael Wouters. Shocks and frictions in US business cycles: A Bayesian DSGE approach. *American Economic Review*, 97(3):586–606, 2007.

Suvrit Sra, Sebastien Nowozin, and Stephen J Wright. Introduction: optimization and machine learning. In *Optimization for Machine Learning*, pages 1–17. MIT Press, 2011.

Harald Uhlig et al. A toolkit for analysing nonlinear dynamic stochastic models easily. *QM&RBC Codes*, 1998.

Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

Sébastien Villemot et al. Solving rational expectations models at first order: what Dynare does. Technical report, Citeseer, 2011.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

Appendix A

Consumption and savings behavior

The maximization of the objective function of the household subject to the budget constraint with respect to consumption and holdings of bonds, may be performed through the Lagrangian-based technique:

$$\begin{aligned} L = \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \varepsilon_t^B \left[\frac{1}{1-\sigma_c} \left(C_t^\tau - H_t^\tau\right)^{1-\sigma_c} - \frac{\varepsilon_t^L}{1+\sigma_l} \left(l_t^\tau\right)^{1+\sigma_l} \right] + \\ -\beta^t \lambda_t \left[C_t^\tau + I_t^\tau + b_t \frac{B_t^\tau}{P_t} - Y_t^\tau - \frac{B_{t-1}^\tau}{P_t} \right] \end{aligned} \tag{6.0.1}$$

The derivatives with respect to C_t^τ and B_t^τ are respectively:

$$\begin{aligned} \frac{\partial L}{\partial C_t^\tau} &= \mathbb{E}_t \left[\beta^t \left(\varepsilon_t^B (C_t^\tau - H_t)^{-\sigma_c} - \lambda_t \right) \right] = 0 \\ &\iff \lambda_t = \varepsilon_t^B (C_t^\tau - H_t)^{-\sigma_c} = \frac{\partial U_t^\tau}{\partial C_t^\tau} := U_t^c \end{aligned} \tag{6.0.2}$$

$$\begin{aligned} \frac{\partial L}{\partial B_t^\tau} &= -\mathbb{E}_t \left[\beta^t \lambda_t \frac{b_t}{P_t} \right] - \mathbb{E}_t \left[\beta^{t+1} \lambda_{t+1} \frac{-1}{P_{t+1}} \right] = 0 \\ &\iff \mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} \frac{1}{b_t} \frac{P_t}{P_{t+1}} \right] = 1 \end{aligned} \tag{6.0.3}$$

Exploiting the fact that households are homogeneous in their consumption-saving decisions, meaning that the marginal utility of consumption is identical for all τ, yields the following Euler equation:

$$\begin{aligned} \mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} \frac{1}{b_t} \frac{P_t}{P_{t+1}} \right] &= \mathbb{E}_t \left[\beta \frac{U_{t+1}^c}{U_t^c} (1+i_t) \frac{P_t}{P_{t+1}} \right] = \\ &= \mathbb{E}_t \left[\beta \frac{\varepsilon_{t+1}^B (C_{t+1}^\tau - H_{t+1})^{-\sigma_c}}{\varepsilon_t^B (C_t^\tau - H_t)^{-\sigma_c}} R_t \frac{P_t}{P_{t+1}} \right] = \\ &= \mathbb{E}_t \left[\beta \frac{\varepsilon_{t+1}^B (C_{t+1}^\tau - h \cdot C_t)^{-\sigma_c}}{\varepsilon_t^B (C_t^\tau - h \cdot C_{t-1})^{-\sigma_c}} R_t \frac{P_t}{P_{t+1}} \right] = 1 \end{aligned} \tag{6.0.4}$$

Alternatively one can rearrange the aforementioned equation as:

$$\mathbb{E}_t\left[\beta\frac{\lambda_{t+1}}{\lambda_t}\frac{1}{b_t}\frac{P_t}{P_{t+1}}\right] = \mathbb{E}_t\left[\beta\frac{U^c_{t+1}}{U^c_t}(1+i_t)\frac{P_t}{P_{t+1}}\right] = 1$$
$$\iff \beta(1+i_t)\mathbb{E}_t\left[\frac{U^c_{t+1}}{P_{t+1}}\right] = \frac{U^c_t}{P_t} \tag{6.0.5}$$

Labor supply decisions

The problem of the firm is as following:

$$\min_{l^\tau_t} \int_0^1 W^\tau_t l^\tau_t d\tau \quad \text{s.t.} \quad L_t = \left[\int_0^1 (l^\tau_t)^{\frac{1}{1+\lambda_{w,t}}}\, d\tau\right]^{1+\lambda_{w,t}} \tag{6.0.6}$$

Define W_t as the Lagrange multiplier, then the cost minimization becomes:

$$L = \int_0^1 W^\tau_t l^\tau_t d\tau + W_t\left(L_t - \left[\int_0^1 (l^\tau_t)^{\frac{1}{1+\lambda_{w,t}}}\, d\tau\right]^{1+\lambda_{w,t}}\right) \tag{6.0.7}$$

Taking the first order conditions:

$$\frac{\partial L}{\partial l^\tau_t} = W^\tau_t - W_t\left((1+\lambda_{w,t})\left[\int_0^1 (l^\tau_t)^{\frac{1}{1+\lambda_{w,t}}}\, d\tau\right]^{\lambda_{w,t}} \frac{1}{1+\lambda_{w,t}}(l^\tau_t)^{\frac{-\lambda_{w,t}}{1+\lambda_{w,t}}}\right) = 0$$
$$\iff W^\tau_t = W_t L_t^{\frac{\lambda_{w,t}}{1+\lambda_{w,t}}}(l^\tau_t)^{\frac{-\lambda_{w,t}}{1+\lambda_{w,t}}}$$
$$\iff l^\tau_t = L_t\left(\frac{W^\tau_t}{W_t}\right)^{\frac{-(1+\lambda_{w,t})}{\lambda_{w,t}}} \tag{6.0.8}$$

The derived optimal demand function of labor can be then substituted in the aggregate labor demand equation yielding:

$$L_t = \left[\int_0^1 \left(\frac{W^\tau_t}{W_t}\right)^{\frac{-1}{\lambda_{w,t}}} L_t^{\frac{1}{1+\lambda_{w,t}}} d\tau\right]^{1+\lambda_{w,t}}$$
$$\iff W_t = \left[\int_0^1 (W^\tau_t)^{\frac{-1}{\lambda_{w,t}}}\, d\tau\right]^{-\lambda_{w,t}} \tag{6.0.9}$$

The Lagrange multiplier W_t, generally considered as the representation of the marginal value of relaxing the constrain, can be interpreted as well as the price of a working hour and thus a wage index.

Wage setting equation

Households that can re-set their wages maximize their objective function subject to their budget constraint and the demand of labor, taking into consideration the fact that wages remain fixed until period i with a probability $(\varsigma_w)^i$. The Lagrangian function in t becomes:

$$L = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left\{ \underbrace{U\left(C^{\tau}_{t+i}, l^{\tau}_{t+i}\right)}_{\text{Utility function}} - \lambda_{t+i} \left(\underbrace{\ldots + \frac{W^{\tau}_{t+i}}{P_{t+i}} l^{\tau}_{t+i} + \ldots}_{\text{Budget constraint}} \right) + \right.$$
$$\left. - \mu_{t+i} \left(\underbrace{l^{\tau}_{t+i} - L^{\tau}_{t+i} \left(\frac{W^{\tau}_{t+i}}{W_{t+i}} \right)^{\frac{-\left(1-\lambda_{w,t+i}\right)}{\lambda_{w,t+i}}}}_{\text{Labor demand}} \right) \right\} \tag{6.0.10}$$

Substituting the demand of labour (6.0.8) in the objective function and in the budget constraint and considering the formulas for not reoptimized wages in period $t+i$, one obtains:

$$L = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left\{ \underbrace{U\left(C^{\tau}_{t+i}, L_{t+i} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \widetilde{W}_t}{W_{t+i}} \right)^{\frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}}} \right)}_{\text{Utility function}} + \right.$$
$$\left. - \lambda_{t+i} \left(\underbrace{\ldots + \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{\widetilde{W}_t}{P_{t+i}} L_{t+i} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \widetilde{W}_t}{W_{t+i}} \right)^{\frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}}} + \ldots}_{\text{Budget constraint}} \right) \right\} \tag{6.0.11}$$

Taking the first order conditions:

$$\frac{\partial L}{\partial \widetilde{W}_t} = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left\{ U_{t+i}^L \frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}} L_{t+i} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_w} \widetilde{W}_t}{W_{t+i}} \right)^{\frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}}-1} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_w}}{W_{t+i}} \right) + \right.$$

$$-\lambda_{t+i} \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{1}{P_{t+i}} L_{t+i} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_w} \widetilde{W}_t}{W_{t+i}} \right)^{\frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}}} + \right.$$

$$\left. \left. - \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{\widetilde{W}_t}{P_{t+i}} L_{t+i} \frac{1+\lambda_{w,t+i}}{\lambda_{w,t+i}} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_w} \widetilde{W}_t}{W_{t+i}} \right)^{\frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}}-1} \left(\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_w}}{W_{t+i}} \right) \right] \right\} = 0 \tag{6.0.12}$$

Therefore:

$$\begin{aligned}
\frac{\partial L}{\partial \widetilde{W}_t} =& \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left\{ U_{t+i}^L \frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}} \frac{l_{t+1}^\tau}{\widetilde{W}_t} + \right. \\
& \left. -\lambda_{t+i} \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{1}{P_{t+i}} l_{t+i}^\tau - \frac{1+\lambda_{w,t+i}}{\lambda_{w,t+i}} \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{1}{P_{t+i}} l_{t+i}^\tau \right] \right\} = \\
=& \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left\{ U_{t+i}^L \frac{-(1+\lambda_{w,t+i})}{\lambda_{w,t+i}} \frac{l_{t+1}^\tau}{\widetilde{W}_t} + \right. \\
& \left. -\lambda_{t+i} \left[\frac{-1}{\lambda_{w,t+i}} \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{1}{P_{t+i}} l_{t+i}^\tau \right] \right\} = 0
\end{aligned} \tag{6.0.13}$$

Multiplying by the factor $\frac{-\lambda_{w,t+i}}{1+\lambda_{w,t+i}}$ and considering that $\lambda_t = U_t^c$ one obtains:

$$\begin{aligned}
& \frac{\widetilde{W}_t}{P_t} \mathbb{E}_t \left\{ \sum_{i=0}^{\infty} \beta^i \varsigma_w^i \left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_w} \frac{P_t}{P_{t+i}} \left(\frac{l_{t+1}^\tau U_{t+i}^C}{1+\lambda_{w,t+i}} \right) \right\} = \\
& \qquad = \mathbb{E}_t \left\{ \sum_{i=0}^{\infty} \beta^i \varsigma_w^i l_{t+1}^\tau U_{t+i}^l \right\}
\end{aligned} \tag{6.0.14}$$

Assuming perfect flexibility of wages ($\varsigma_w = 0$), the above-mentioned equation becomes:

$$\frac{\widetilde{W}_t}{P_t} = (1+\lambda_{w,t}) \frac{U_t^L}{U_t^C} \tag{6.0.15}$$

Capital accumulation and investment decision

Define the Lagrange multiplier $\lambda_t Q_t$ for the maximization problem with respect to K_t, I_t, z_t:

$$L = \mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t \Bigg[\underbrace{U\left(C_t^\tau, l_t^\tau\right)}_{\text{Objective function}} - \lambda_t \Bigg(\underbrace{I_t + \left(\Psi(z_t) - r_t^k z_t\right) K_{t-1} + \dots}_{\text{Budget constraint}} \Bigg) + \\ - \lambda_t Q_t \Bigg(\underbrace{K_t - K_{t-1}(1-\tau) - I_t + I_t \cdot S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right)}_{\text{Capital accumulation}} \Bigg) \Bigg] \tag{6.0.16}$$

The first-order conditions:

$$\begin{aligned} \frac{\partial L}{\partial z_t} &= \mathbb{E}_t \left[-\beta^t \lambda_t \left(\Psi'(z_t) - r_t^k \right) K_{t-1} \right] = 0 \\ &\iff r_t^k = \Psi'(z_t) \end{aligned} \tag{6.0.17}$$

The capital utilization rate is set so that the revenue r_t^k of the marginal utilization equals the marginal costs $\Psi'(z_t)$.

$$\begin{aligned} \frac{\partial L}{\partial K_t} &= \mathbb{E}_t \left[\beta^{t+1} \lambda_{t+1} \left(z_{t+1} r_{t+1}^k - \Psi(z_{t+1}) \right) - \beta^t \lambda_t Q_t + \beta^{t+1} \lambda_{t+1} Q_{t+1} (1-\tau) \right] = 0 \\ &\iff Q_t = \mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} \left(Q_{t+1}(1-\tau) + z_{t+1} r_{t+1}^k - \Psi(z_{t+1}) \right) \right] \end{aligned} \tag{6.0.18}$$

The real value of the stock of capital today $\lambda_t Q_t$ is equal to the expected value of sum of not depreciated stock of capital of the next period $\lambda_{t+1} Q_{t+1}(1-\tau)$ and the expected revenue of the future utilization $z_{t+1} r_{t+1}^k$ minus the related costs $\Psi(z_{t+1})$.

$$\begin{aligned} \frac{\partial L}{\partial I_t} = \mathbb{E}_t \left[-\beta^t \lambda_t - \beta^t \lambda_t Q_t \left(-1 + S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right) + I_t \cdot S'\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right) \cdot \frac{\varepsilon_t^I}{I_{t-1}} \right) \right] + \\ - \mathbb{E}_t \left[\beta^{t+1} \lambda_{t+1} Q_{t+1} \left(I_{t+1} \cdot S'\left(\frac{\varepsilon_{t+1}^I I_{t+1}}{I_t}\right) \cdot \frac{-\varepsilon_{t+1}^I I_{t+1}}{I_t^2} \right) \right] = 0 \end{aligned} \tag{6.0.19}$$

The costs of the marginal investment (included the adjustment costs) must be equal to the expected marginal revenue of investment.

Firms - Final product sector

The Lagrangian for the minimization problem:

$$\int_0^1 p_t^j y_t^j dj + P_t \left(Y_t - \left[\int_0^1 \left(y_t^j \right)^{\frac{1}{1+\lambda_{p,t}}} dj \right]^{1+\lambda_{p,t}} \right) \tag{6.0.20}$$

First order conditions:

$$\begin{aligned} \frac{\partial L}{\partial y_t^j} &= p_t^j - P_t \left((1+\lambda_{p,t}) \left[\int_0^1 \left(y_t^j \right)^{\frac{1}{1+\lambda_{p,t}}} dj \right]^{\lambda_{p,t}} \frac{1}{1+\lambda_{p,t}} \left(y_t^j \right)^{\frac{-\lambda_{p,t}}{1+\lambda_{p,t}}} \right) = 0 \\ &\iff p_t^j = P_t Y_t^{\frac{\lambda_{p,t}}{1+\lambda_{p,t}}} \left(y_t^j \right)^{\frac{-\lambda_{p,t}}{1+\lambda_{p,t}}} \\ &\iff y_t^j = Y_t \left(\frac{p_t^j}{P_t} \right)^{\frac{-(1+\lambda_{p,t})}{\lambda_{p,t}}} \end{aligned} \tag{6.0.21}$$

Substituting into the technology constraint:

$$\begin{aligned} Y_t &= \left[\int_0^1 \left(\frac{p_t^j}{P_t} \right)^{\frac{-1}{\lambda_{p,t}}} Y_t^{\frac{1}{1+\lambda_{p,t}}} dj \right]^{1+\lambda_{p,t}} \\ &\iff P_t = \left[\int_0^1 \left(p_t^j \right)^{\frac{-1}{\lambda_{p,t}}} dj \right]^{-\lambda_{p,t}} \end{aligned} \tag{6.0.22}$$

Firms - Intermediate product sector

In order to find the optimal production quantity, it is convenient to show that marginal costs are constant. One proceeds first by finding a functional relationship between $\widetilde{K}_{j,t}$ and $L_{j,t}$ derived by the minimization problem of $W_t L_{j,t} + r_t^k \widetilde{K}_{j,t}$ with respect to $L_{j,t}$ and $\widetilde{K}_{j,t}$ subject to $y_t^j = \varepsilon_t^a \widetilde{K}_{j,t}^{\alpha} L_{j,t}^{1-\alpha} - \Phi$.

$$\begin{aligned} \frac{W_t}{r_t^k} &= \frac{(1-\alpha)\varepsilon_t^a \widetilde{K}_{j,t}^{\alpha} L_{j,t}^{-\alpha}}{\alpha \varepsilon_t^a \widetilde{K}_{j,t}^{\alpha-1} L_{j,t}^{1-\alpha}} \\ &\iff \frac{W_t L_{t,j}}{r_t^k \widetilde{K}_{j,t}} = \frac{1-\alpha}{\alpha} \end{aligned} \tag{6.0.23}$$

The utilization rate between capital and labour is identical for all firms and constant. This is the same of the economy as a whole. The marginal costs of product j are derived by substituting the above-mentioned relation in the production function:

$$\widetilde{K}_{j,t} = \left(y_t^j + \Phi\right) \frac{1}{\varepsilon_t^a} \left(\frac{\alpha}{1-\alpha} \frac{W_t}{r_t^k}\right)^{1-\alpha} \tag{6.0.24}$$

$$L_{j,t} = \left(y_t^j + \Phi\right) \frac{1}{\varepsilon_t^a} \left(\frac{\alpha}{1-\alpha} \frac{W_t}{r_t^k}\right)^{-\alpha} \tag{6.0.25}$$

Therefore the total cost equals:

$$TC_t = W_t L_{j,t} + r_t^k \widetilde{K}_{j,t} = \left(y_t^j + \Phi\right) \frac{1}{\varepsilon_t^a} W_t^{1-\alpha} \left(r_t^k\right)^{\alpha} \left(\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)}\right) \tag{6.0.26}$$

Hence the marginal cost takes the form:

$$MC_t = \frac{\partial TC_t}{\partial y_t^j} = \frac{1}{\varepsilon_t^a} W_t^{1-\alpha} \left(r_t^k\right)^{\alpha} \left(\alpha^{-\alpha}(1-\alpha)^{-(1-\alpha)}\right) \tag{6.0.27}$$

In order to derive the optimal mark-up formula for the reoptimize price, the Lagrangian can be formed:

$$L = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \rho_{t+i} \left\{ \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t - MC_{t+i} \right] Y_{t+i} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t}{P_{t+i}} \right]^{\frac{-(1+\lambda_{p,t+i})}{\lambda_{p,t+i}}} - \Phi MC_{t+i} \right\} \tag{6.0.28}$$

First order condition:

$$\begin{aligned}
\frac{\partial L}{\partial \widetilde{p}_t} = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \rho_{t+i} \Bigg\{ & \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \right] Y_{t+i} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t}{P_{t+i}} \right]^{\frac{-(1+\lambda_{p,t+i})}{\lambda_{p,t+i}}} + \\
& + \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t - MC_{t+i} \right] Y_{t+i} \frac{-(1+\lambda_{p,t+i})}{\lambda_{p,t+i}} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t}{P_{t+i}} \right]^{\frac{-(1+\lambda_{p,t+i})}{\lambda_{p,t+i}} - 1} \frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p}}{P_{t+i}} \Bigg\} = \\
= \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \rho_{t+i} & \left\{ \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \right] y_{t+i}^j + \left[\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p} \widetilde{p}_t - MC_{t+i} \right] \frac{-(1+\lambda_{p,t+i})}{\lambda_{p,t+i}} \frac{y_{t+i}^j}{\widetilde{p}_t} \right\} = 0
\end{aligned} \tag{6.0.29}$$

Multiplying through the factor $\widetilde{p}_t \lambda_{p,t+i}$ and considering equation 2.2.8, the following mark-up formula for the reoptimized price obtains:

$$\mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \frac{\lambda_{t+i}}{\lambda_t} y_{t+i}^j \left\{ \frac{\widetilde{p}_t}{P_t} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}} \right)^{\gamma_p}}{\frac{P_{t+i}}{P_t}} \right] - (1 + \lambda_{p,t+i}) \frac{MC_{t+i}}{P_{t+i}} \right\} = 0 \tag{6.0.30}$$

Log-linearised DSGE model - Consumption

Recall that:

$$\lambda_t = \varepsilon_t^B \left(C_t^\tau - H_t \right)^{-\sigma_c} \tag{6.0.31}$$

Then from the household maximization problem it holds that:

$$\mathbb{E}_t \left[\beta \lambda_{t+1} \frac{R_t}{P_{t+1}} \right] = \frac{\lambda_t}{P_t} \tag{6.0.32}$$

Then the left-hand side:

$$\begin{aligned}
&\mathbb{E}_t \left[\beta \lambda_{t+1} \frac{R_t}{P_{t+1}} \right] \\
&= \mathbb{E}_t \left[\beta \varepsilon_{t+1}^B \left(C_{t+1}^\tau - H_{t+1} \right)^{-\sigma_c} \frac{R_t}{P_{t+1}} \right] = \\
&= \mathbb{E}_t \left[\beta \varepsilon_{t+1}^B \left(C_{t+1}^\tau - hC_t \right)^{-\sigma_c} \frac{R_t}{P_{t+1}} \right] \simeq \\
&\simeq \mathbb{E}_t \Bigg[\beta \varepsilon^B \left(C - hC \right)^{-\sigma_c} \frac{R}{P} + \beta \varepsilon^B \left(C - hC \right)^{-\sigma_c} \frac{R}{P} \hat{\varepsilon}_{t+1}^B - \sigma_c \beta \varepsilon^B \left(C - hC \right)^{-\sigma_c - 1} \left(C\hat{C}_{t+1} - hC\hat{C}_t \right) \frac{R}{P} + \\
&\quad + \beta \varepsilon^B \left(C - hC \right)^{-\sigma_c} \frac{R}{P} \hat{R}_t - \beta \varepsilon^B \left(C - hC \right)^{-\sigma_c} \frac{R}{P} \hat{P}_{t+1} \Bigg] = \\
&= \mathbb{E}_t \left[\beta \varepsilon^B \left(C - hC \right)^{-\sigma_c} \frac{R}{P} \cdot \left(1 + \hat{\varepsilon}_{t+1}^B - \sigma_c \left(C - hC \right)^{-1} \left(C\hat{C}_{t+1} - hC\hat{C}_t \right) + \hat{R}_t - \hat{P}_{t+1} \right) \right]
\end{aligned} \tag{6.0.33}$$

While the right-hand side:

$$\begin{aligned}
\frac{\lambda_t}{P_t} &= \varepsilon_t^B \left(C_t^\tau - H_t \right)^{-\sigma_c} P_t^{-1} = \varepsilon_t^B \left(C_t^\tau - hC_{t-1} \right)^{-\sigma_c} P_t^{-1} \simeq \\
&\simeq \varepsilon^B \left(C - hC \right)^{-\sigma_c} P^{-1} + \varepsilon^B \left(C - hC \right)^{-\sigma_c} P^{-1} \hat{\varepsilon}_t^B + \\
&\quad - \sigma_C \varepsilon^B \left(C - hC \right)^{-\sigma_c - 1} P^{-1} \left(C\hat{C}_t - hC\hat{C}_{t-1} \right) + \\
&\quad - \varepsilon^B \left(C - hC \right)^{-\sigma_c} P^{-1} \hat{P}_t = \\
&= \varepsilon^B \left(C - hC \right)^{-\sigma_c} P^{-1} \left(1 + \hat{\varepsilon}_t^B - \sigma_C \left(C - hC \right)^{-1} \left(C\hat{C}_t - hC\hat{C}_{t-1} \right) - \hat{P}_t \right)
\end{aligned} \tag{6.0.34}$$

Finally considering that in the steady-state:

$$\begin{aligned}\beta R\left[\varepsilon^{B}\left(C-hC\right)^{-\sigma_c}\right] &= \varepsilon^{B}\left(C-hC\right)^{-\sigma_c}\frac{P}{P}\\ \iff \beta &= \frac{1}{R}\end{aligned} \tag{6.0.35}$$

Putting both sides together yields:

$$\hat{R}_t + \mathbb{E}_t\hat{\varepsilon}^{B}_{t+1} - \frac{\sigma_c}{1-h}\left(\mathbb{E}_t\hat{C}_{t+1} - h\hat{C}_t\right) = \hat{\varepsilon}^{B}_t - \frac{\sigma_c}{1-h}\left(\hat{C}_t - h\hat{C}_{t-1}\right) + \mathbb{E}_t\hat{\Pi}_{t+1} \tag{6.0.36}$$

from which one obtains the explicit expression for $\hat{C}_t$.

Log-linearised DSGE model - Investment

Recall the optimal condition for investment:

$$\begin{aligned}\frac{\partial L}{\partial I_t} &= \mathbb{E}_t\left[-\beta^t\lambda_t - \beta^t\lambda_t Q_t\left(-1 + S\left(\frac{\varepsilon^I_t I_t}{I_{t-1}}\right) + I_t\cdot S'\left(\frac{\varepsilon^I_t I_t}{I_{t-1}}\right)\cdot\frac{\varepsilon^I_t}{I_{t-1}}\right)\right] +\\ &- \mathbb{E}_t\left[\beta^{t+1}\lambda_{t+1}Q_{t+1}\left(I_{t+1}\cdot S'\left(\frac{\varepsilon^I_{t+1}I_{t+1}}{I_t}\right)\cdot\frac{-\varepsilon^I_{t+1}I_{t+1}}{I_t^2}\right)\right] = 0\\ &\iff Q_t S'\left(\frac{\varepsilon^I_t I_t}{I_{t-1}}\right)\left(\frac{\varepsilon^I_t I_t}{I_{t-1}}\right) - \beta\mathbb{E}_t\left[Q_{t+1}\frac{\lambda_{t+1}}{\lambda_t}S'\left(\frac{\varepsilon^I_{t+1}I_{t+1}}{I_t}\right)\frac{\varepsilon^I_{t+1}I_{t+1}}{I_t}\frac{I_{t+1}}{I_t}\right] + 1 =\\ &= Q_t\left(1 - S\left(\frac{\varepsilon^I_t I_t}{I_{t-1}}\right)\right)\end{aligned} \tag{6.0.37}$$

Since in the steady state $\varepsilon^I = 1$ and $S(1) = S'(1) = 0$, it holds:

$$QS'\left(\frac{\varepsilon^I I}{I}\right)\left(\frac{\varepsilon^I I}{I}\right) - \beta\left[Q\frac{\lambda}{\lambda}S'\left(\frac{\varepsilon^I I}{I}\right)\frac{\varepsilon^I I}{I}\frac{I}{I}\right] + 1 = Q\left(1 - S\left(\frac{\varepsilon^I I}{I}\right)\right) \iff Q = 1 \tag{6.0.38}$$

The approximation of the single terms yields:

$$S'\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right)\frac{\varepsilon_t^I I_t}{I_{t-1}} \simeq S'(1)\frac{\varepsilon^I I}{I} + \left[S''(1)\left(\frac{I}{I}\right)\frac{\varepsilon^I I}{I} + S'(1)\frac{I}{I}\right]\left(\varepsilon_t^I - \varepsilon^I\right) +$$
$$+ \left[S''(1)\left(\frac{\varepsilon^I}{I}\right)\frac{\varepsilon^I I}{I} + S'(1)\frac{\varepsilon^I}{I}\right](I_t - I) +$$
$$+ \left[S''(1)\left(\frac{-\varepsilon^I I}{I^2}\right)\frac{\varepsilon^I I}{I} + S'(1)\frac{-\varepsilon^I I}{I^2}\right](I_{t-1} - I) =$$
$$= S''(1)\left[\hat{I}_t - \hat{I}_{t-1} + \hat{\varepsilon}_t^I\right] \tag{6.0.39}$$

Note that since $S'(1) = 0$ one does not need the approximation of Q_t, Q_{t+1}, λ_t and λ.

$$\beta\mathbb{E}_t\left[\frac{Q_{t+1}}{Q_t}\frac{\lambda_{t+1}}{\lambda_t}S'\left(\frac{\varepsilon_{t+1}^I I_{t+1}}{I_t}\right)\frac{\varepsilon_{t+1}^I I_{t+1}}{I_t}\frac{I_{t+1}}{I_t}\right] \simeq$$
$$\simeq \beta\frac{Q}{Q}\frac{\lambda}{\lambda}S'\left(\frac{\varepsilon^I I}{I}\right)\frac{\varepsilon^I I^2}{I^2} + \beta\frac{Q}{Q}\frac{\lambda}{\lambda}\left[S''(1)\frac{\varepsilon^I}{I}\frac{\varepsilon^I I^2}{I^2} + 2S'(1)\frac{\varepsilon^I I}{I^2}\right](\mathbb{E}_t I_{t+1} - I) +$$
$$+ \beta\frac{Q}{Q}\frac{\lambda}{\lambda}\left[S''(1)\frac{-\varepsilon^I I}{I^2}\frac{\varepsilon^I I^2}{I^2} - 2S'(1)\frac{\varepsilon^I I^2}{I^3}\right](I_t - I) +$$
$$+ \beta\frac{Q}{Q}\frac{\lambda}{\lambda}\left[S''(1)\frac{I}{I}\frac{\varepsilon^I I^2}{I^2} + S'(1)\frac{I^2}{I^2}\right]\left(\mathbb{E}_t\varepsilon_{t+1}^I - \varepsilon^I\right) =$$
$$= \beta S''(1)\left[\mathbb{E}_t\hat{I}_{t+1} - \hat{I}_t + \mathbb{E}_t\varepsilon_{t+1}^I\right] \tag{6.0.40}$$

Since $Q = 1$:

$$\frac{1}{Q_t} \simeq \frac{1}{Q} - \frac{1}{Q^2}(Q_t - Q) \,. = 1 - \hat{Q}_t \tag{6.0.41}$$

Moreover:

$$S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right) \simeq S(1) + S'(1)\left[\frac{I}{I}\left(\varepsilon_t^I - \varepsilon\right) + \frac{\varepsilon^I}{I}(I_t - I) - \frac{\varepsilon^I I}{I^2}(I_{t-1} - I)\right] = 0 \tag{6.0.42}$$

Hence it follows:

$$S''(1)\left[\hat{I}_t - \hat{I}_{t-1} + \varepsilon_t^I\right] - \beta S''(1)\left[\mathbb{E}_t\hat{I}_{t+1} - \hat{I}_t + \mathbb{E}_t\varepsilon_{t+1}^I\right] \simeq 1 - (1 - \hat{Q}_t) \tag{6.0.43}$$

Thus:

$$\hat{I}_t = \frac{1}{1+\beta}\hat{I}_{t-1} + \frac{\beta}{1+\beta}\mathbb{E}_t\hat{I}_{i+1} + \frac{1}{S''(1)(1+\beta)}\hat{Q}_t + \frac{\beta\mathbb{E}_t\hat{\varepsilon}_{t+1}^I - \hat{\varepsilon}_t^I}{1+\beta} \tag{6.0.44}$$

Log-linearised DSGE model - Q-equation

Recall that

$$Q_t = \mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} \left(Q_{t+1}(1-\tau) + z_{t+1} r^k_{t+1} - \Psi(z_{t+1}) \right) \right] \tag{6.0.45}$$

In the steady state $Q = 1$, $z = 1$ and $\Psi(1) = 0$, thus:

$$\begin{aligned} Q &= \beta \frac{\lambda}{\lambda} \left(Q(1-\tau) + z r^k - \Psi(z) \right) \\ &\iff 1 = \beta \left(1 - \tau + r^k \right) \end{aligned} \tag{6.0.46}$$

The linearization of the left side yields:

$$Q_t \simeq 1 + \hat{Q}_t \tag{6.0.47}$$

Exploiting the fact that $r^k = \Psi'(z)$, the right side approximates by:

$$\begin{aligned} &\mathbb{E}_t \left[\beta \frac{\lambda_{t+1}}{\lambda_t} \left(Q_{t+1}(1-\tau) + z_{t+1} r^k_{t+1} - \Psi(z_{t+1}) \right) \right] \simeq \\ &\simeq \beta \left(1 - \tau + r^k \right) + \beta \frac{1}{\lambda} \underbrace{\left[Q(1-\tau) + z r^k - \Psi(z) \right]}_{\beta^{-1}} \left(\mathbb{E}_t \lambda_{t+1} - \lambda \right) + \\ &\quad - \beta \frac{\lambda}{\lambda^2} \underbrace{\left[Q(1-\tau) + z r^k - \Psi(z) \right]}_{\beta^{-1}} \left(\lambda_t - \lambda \right) + \\ &\quad + \beta \frac{\lambda}{\lambda} \left(1 - \tau \right) \left(\mathbb{E}_t Q_{t+1} - Q \right) + \beta \frac{\lambda}{\lambda} z \left(\mathbb{E}_t r^k_{t+1} - r \right) + \\ &\quad + \beta \frac{\lambda}{\lambda} \left(r^k - \Psi'(z) \right) \left(z_{t+1} - z \right) = \\ &= 1 + \mathbb{E}_t \hat{\lambda}_{t+1} - \hat{\lambda}_t + \beta(1-\tau) \mathbb{E}_t \hat{Q}_{t+1} + \beta \mathbb{E}_t r^k \hat{r}_{t+1} \end{aligned} \tag{6.0.48}$$

From the Euler equation:

$$\mathbb{E}_t \hat{\lambda}_{t+1} - \hat{\lambda}_t = - \left(\hat{R}_t - \hat{\pi}_{t+1} \right) \tag{6.0.49}$$

Putting the above-mentioned conditions together:

$$\hat{Q}_t = - \left(\hat{R}_t - \hat{\pi}_{t+1} \right) + \beta(1-\tau) \mathbb{E}_t \hat{Q}_{t+1} + \beta \mathbb{E}_t r^k \hat{r}_{t+1} \tag{6.0.50}$$

Log-linearised DSGE model - Capital accumulation

Starting from:

$$K_t = K_{t-1}(1-\tau) + I_t \left[1 - S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right)\right] \tag{6.0.51}$$

In the steady state it holds:

$$K = (1-\tau)K + I\left[1 - \underbrace{S(1)}_{=0}\right] \iff I = \tau K \tag{6.0.52}$$

The approximation of the left side yields:

$$K_t = K + (K_t - K) \simeq K + \hat{K}_t K \tag{6.0.53}$$

Analogously for the right side:

$$K_{t-1}(1-\tau) + I_t \left[1 - S\left(\frac{\varepsilon_t^I I_t}{I_{t-1}}\right)\right] \simeq K(1-\tau) + I + (1-\tau)\left(K_{t-1} - K\right) + (I_t - I) \tag{6.0.54}$$

Considering the steady state relationships:

$$\begin{aligned} &\hat{K}_t K = (1-\tau)K\hat{K}_{t-1} + \hat{I}_t I = (1-\tau)K\hat{K}_{t-1} + \hat{I}_t \tau K \\ &\Rightarrow \hat{K}_t = (1-\tau)\hat{K}_{t-1} + \hat{I}_{t-1}\tau \end{aligned} \tag{6.0.55}$$

Log-linearised DSGE model - Prices

Starting from:

$$\mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \frac{\lambda_{t+i}}{\lambda_t} y_{t+i}^j \left\{ \frac{\widetilde{p}_t}{P_t} \left[\frac{\left(\frac{P_{t+i-1}}{P_{t-1}}\right)^{\gamma_p}}{\frac{P_{t+i}}{P_t}} \right] - (1 + \lambda_{p,t+i}) \frac{MC_{t+i}}{P_{t+i}} \right\} = 0 \tag{6.0.56}$$

Denote $\phi_t = \log(\widetilde{p}_t)$, $p_t = \log(P_t)$, $mc_t = \log\left(\frac{MC_t}{P_t}\right)$ and $\mu = \log(1+\lambda)$. Then:

$$\mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \frac{\lambda_{t+i}}{\lambda_t} y_{t+i}^j \left\{ e^{\phi_t - p_{t+i} + \gamma_p (p_{t+i-1} - p_{t-1})} - e^{\mu + mc_{t+i}} \right\} = 0 \tag{6.0.57}$$

In the steady state it holds $\phi_t = p_t = p$, thus:

$$\begin{aligned} & e^0 - e^{\mu+mc} = 0 \\ \iff & mc = -\mu \end{aligned} \tag{6.0.58}$$

Having noticed that the internal bracket of equation 6.0.57 is null in the steady state:

$$\begin{aligned} & \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \frac{\lambda}{\lambda} y \left\{ e^0 \left(\phi_t - p\right) + e^0(-1) \left(p_{t+i} - p\right) + e^0 \left(p_{t+i-1} - p\right) - e^0 \gamma_p \left(p_{t-1} - p\right) + \right. \\ & \left. - e^0 \left(mc_{t+i} - mc\right) \right\} = 0 \\ & \Rightarrow \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \phi_t = \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \left[\underbrace{-mc}_{=\mu} + mc_{t+i} + p_{t+i} - \gamma_p \left(p_{t+i-1} - p_{t-1}\right) \right] \\ & \Rightarrow \frac{1}{1-\beta\varsigma_p} \phi_t = \frac{1}{1-\beta\varsigma_p} \mu + \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \left[mc_{t+i} + p_{t+i} - \gamma_p \left(p_{t+i-1} - p_{t-1}\right) \right] \\ & \Rightarrow \phi_t = \mu + \left(1 - \beta\varsigma_p\right) \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \left[mc_{t+i} + p_{t+i} - \gamma_p \left(p_{t+i-1} - p_{t-1}\right) \right] \\ & \Rightarrow \hat{\phi}_t = \phi_t - p = \left(1 - \beta\varsigma_p\right) \mathbb{E}_t \sum_{i=0}^{\infty} \beta^i \varsigma_p^i \left[\hat{mc}_{t+i} + \hat{p}_{t+i} - \gamma_p \left(p_{t+i-1} - p_{t-1}\right) \right] \\ & \Rightarrow \hat{\phi}_t - \beta\varsigma_p \mathbb{E}_t \hat{\phi}_{t+1} = \left(1 - \beta\varsigma_p\right) \left(\hat{mc}_t + \hat{p}_t\right) \end{aligned} \tag{6.0.59}$$

Moreover, the linearization of the equation of motion of the price level yields:

$$\begin{aligned} & \left(P_t\right)^{\frac{1}{-\lambda_{p,t}}} = \varsigma_p \left[P_{t-1} \left(\frac{P_{t-1}}{P_{t-2}} \right)^{\gamma_p} \right]^{\frac{-1}{\lambda_{p,t}}} + \left(1 - \varsigma_p\right) \left(\widetilde{p}_t\right)^{\frac{-1}{\lambda_{p,t}}} \\ & \Rightarrow 1 = \varsigma_p \left[\Pi_{t-1}^{\gamma_p} \Pi_t^{-1} \right]^{\frac{-1}{\lambda_{p,t}}} + \left(1 - \varsigma_p\right) \left(\frac{\widetilde{p}_t}{P_t} \right)^{\frac{-1}{\lambda_{p,t}}} \\ & \Rightarrow 0 = \frac{-1}{\lambda_{p,t}} \left[\Pi^{\gamma_p} \Pi^{-1} \right]^{\frac{-1}{\lambda_{p,t}} - 1} \left[\gamma_p \pi^{\gamma_p - 1} \left(\Pi_{t-1} - \Pi\right) - \Pi^{-2} \left(\Pi_t - \Pi\right) \right] + \\ & \quad - \frac{-1}{\lambda_{p,t}} (1 - \varsigma_p) \frac{P}{P}^{\frac{-1}{\lambda_{p,t}} - 1} \frac{1}{P} \left[\left(\widetilde{p}_t - P\right) \left(P_t - P\right) \right] \\ & \Rightarrow \hat{\phi}_t - \hat{p}_t = \frac{\varsigma_p}{1 - \varsigma_p} \left(\hat{\pi}_t - \gamma_p \hat{\pi}_{t-1} \right) \end{aligned} \tag{6.0.60}$$

The substitution into 6.0.59 yields the new-keynesian Phillips curve:

$$\hat{\pi}_t = \frac{\beta}{1 + \beta\gamma_p} \mathbb{E}_t \hat{\pi}_{t+1} + \frac{\gamma_p}{1 + \beta\gamma_p} \hat{\pi}_{t-1} + \frac{\left(1 - \beta\varsigma_p\right)\left(1 - \varsigma_p\right)}{\left(1 + \beta\gamma_p\right)\varsigma_p} \left(\underbrace{\alpha \hat{r}_t^k + (1 - \alpha)\hat{w}_t - \hat{\varepsilon}_t^a + \eta_t^p}_{\hat{mc}_t} \right) \tag{6.0.61}$$

Log-linearised DSGE model - Labor demand

Starting with:

$$\frac{W_t L_t}{r_t^k z_t K_{t-1}} = \frac{1-\alpha}{\alpha} \tag{6.0.62}$$

In the steady state it holds:

$$\frac{WL}{r^k z K} = \frac{1-\alpha}{\alpha} \tag{6.0.63}$$

Therefore:

$$\begin{aligned}\log(L_t) - \log(L) = \log(r_t^k) - \log(r^k) - [(\log(W_t) + \log(P_t)) - (\log(W) - \log(P_t))] + \\ + \log(z_t) - \log(z) + \log(K_{t-1}) - \log(K)\end{aligned} \tag{6.0.64}$$

Hence:

$$\hat{L}_t = -\hat{w}_t + \hat{r}_t^k + \hat{z}_t + \hat{K}_{t-1} \tag{6.0.65}$$

On the other hand:

$$r_t^k = \Psi'(z_t) \Rightarrow r^k + r^k \hat{r}_t^k = \Psi'(z) + \Psi''(z) z_t \hat{z}_t \simeq \Psi'(z_t) \tag{6.0.66}$$

But simultaneously:

$$r^k = \Psi'(z) \tag{6.0.67}$$

$$z = 1 \tag{6.0.68}$$

Thus:

$$\begin{aligned}\hat{r}_t^k \Psi'(1) &= \Psi''(1)\hat{z}_t \\ \Rightarrow \hat{z}_t &= \frac{\Psi'(1)}{\Psi''(1)} \hat{r}_t^k\end{aligned} \tag{6.0.69}$$

It follows:

$$\hat{L}_t = -\hat{w}_t + \left(1 + \frac{\Psi'(1)}{\Psi''(1)}\right) \hat{r}_t^k + \hat{K}_{t-1} \tag{6.0.70}$$

Appendix B

```
% Smets and Wouters (2003)
%
% An estimated dynamic stochastic general
% equilibrium model of the euro area
%
% Journal of the European Economic Association
% September 2003, 1(5), 1123-1175
%
% Author:        Michal Miktus
% This version: 18.03.2019
%-----------------------------------------------------------------------------------------



%-----------------------------------------------------------------------------------------
% 1. Declare variables and parameters
%-----------------------------------------------------------------------------------------
  var  e_a e_I e_b e_L e_G e_pi  one pi w K Q I C R r L Y pi_f w_f K_f Q_f
      I_f C_f R_f r_f L_f Y_f;
  varexo ee_a ee_I ee_b ee_L ee_G ee_pi n_p n_Q n_R n_w;
  parameters h sigma_c beta adj tau gamma_p xi_p alpha gamma_w xi_w sigma_L
       lambda_w psi phi k g rho r_pi r_Y r_dpi r_dY r_k rho_a rho_I rho_b
      rho_L rho_G rho_pi sd_Q sd_p sd_w sd_R;
  varobs C I L;

%-----------------------------------------------------------------------------------------
% 2. Set parameters
%-----------------------------------------------------------------------------------------

    beta         = 0.99;
    tau          = 0.025;
    alpha        = 0.3;
    psi          = 1/0.169;
    gamma_p      = 0.469;
    gamma_w      = 0.763;
    lambda_w     = 0.5;
    xi_p         = 0.908;
    xi_w         = 0.737;
    sigma_L      = 2.4;
    sigma_c      = 1.353;
    h            = 0.573;
    phi          = 1.408;
```

```
    adj          = 1/6.771;
    r_k          = (1/beta)-1+tau;
    k            = 8.8;
    g            = 0.18;
    r_dpi        = 0.14;
    r_Y          = 0.099;
    r_dY         = 0.159;
    rho          = 0.961;
    r_pi         = 1.684;


    rho_L        = 0.889;
    rho_a        = 0.823;
    rho_b        = 0.855;
    rho_G        = 0.949;
    rho_pi       = 0.924;
    rho_I        = 0.927;

    sd_R         = 0.081;

    sd_p         = 0.16;
    sd_w         = 0.289;
    sd_Q         = 0.604;


%-----------------------------------------------------------------------
% 3. Model equations
%-----------------------------------------------------------------------
  model(linear);
    % System 1: Sticky prices & wages and cost-push shocks

    % Consumption, Eq. 1.3.2
    C = (h/(1 + h))*C(-1) + (1/(1 + h))*C(+1) - ((1 - h)/((1 + h)*sigma_c))
        *(R - pi(+1)) + ((1 - h)/((1 + h)*sigma_c))*(e_b-e_b(+1));

    % Investment, Eq. 1.3.3
    I = (1/(1 + beta))*I(-1) + (beta/(1 + beta))*I(+1) + (adj/(1 + beta))*Q
         + (beta*e_I(+1) - e_I)/(1 + beta);

    % Q-equation, Eq. 1.3.4
    Q = -(R - pi(+1)) + ((1 - tau)/(1 - tau + r_k))*Q(+1) + (r_k/(1 - tau +
         r_k))*r(+1) + n_Q;

    % Capital, Eq. 1.3.5
    K = (1 - tau)*K(-1) + tau*I(-1);

    % Phillips Curve, Eq. 1.3.6
    pi = (beta/(1 + beta*gamma_p))*pi(+1) + (gamma_p/(1 + beta*gamma_p))*pi
        (-1) + (((1 - beta*xi_p)*(1 - xi_p))/((1 + beta*gamma_p)*xi_p))*(
        alpha*r + (1 - alpha)*w - e_a + n_p);

    % Wages, Eq. 1.3.7
    w = (beta/(1 + beta))*w(+1) + (1/(1 + beta))*w(-1)
```

```
87          + (beta/(1 + beta))*pi(+1) - ((1 + beta*gamma_w)/(1 + beta))*pi + (
                gamma_w/( 1 + beta))*pi(-1) - ((1/(1 + beta))*((1 - beta*xi_w)
                *(1 - xi_w))/((1 + (1/lambda_w)*((1 + lambda_w)*sigma_L))*xi_w)
                )*(w - sigma_L*L - (sigma_c/(1 - h))*(C - h*C(-1)) - e_L - n_w)
                ;
88
89      % Labor demand, Eq. 1.3.9
90      L = -w + (1 + psi)*r + K(-1);
91
92      % Production,   Eq. 1.3.13
93      Y = phi*e_a + phi*alpha*K(-1) + phi*alpha*psi*r + phi*(1 - alpha)*L;
94
95      % Goods market, Eq. 1.3.12
96      Y = (1 - tau*k - g)*C + tau*k*I + e_G;
97
98      % Monetary policy, Eq. 1.3.17
99      R = rho*R(-1) + (1 - rho)*(e_pi + r_pi*(pi(-1) - e_pi) + r_Y*(Y - Y_f))
             + r_dpi*(pi - pi(-1)) + r_dY*((Y - Y_f) - (Y(-1) - Y_f(-1))) + n_R
            ;
100
101
102     % System 2: Flexible System
103
104     % Price Stickyness: xi_p = 0
105     % Wage Stickyness: xi_w = 0
106     % Cost shocks: n_p = 0, n_Q = 0, n_w = 0
107
108     C_f     = (h/(1 + h))*C_f(-1) + (1/(1 + h))*C_f(+1) - ((1 - h)/((1 + h)
            *sigma_c))*(R_f - pi_f(+1)) + ((1 - h)/((1 + h)*sigma_c))*(e_b-e_b
            (+1));
109     I_f     = (1/(1 + beta))*I_f(-1) + (beta/(1 + beta))*I_f(+1) + (adj/(1
            + beta))*Q_f + (beta*e_I(+1) - e_I)/(1 + beta);
110     Q_f     = -(R_f - pi_f(+1)) + ((1 - tau)/(1 - tau + r_k))*Q_f(+1) + (
            r_k/(1 - tau + r_k))*r_f(+1);
111     K_f     = (1 - tau)*K_f(-1) + tau*I_f(-1);
112     pi_f    = 0*one;
113     0       = alpha*r_f+(1-alpha)*w_f - e_a;
114     w_f     = sigma_L*L_f + (sigma_c/(1 - h))*(C_f - h*C_f(-1)) - e_L;
115     L_f     = -w_f + (1 + psi)*r_f + K_f(-1);
116     Y_f     = phi*e_a + phi*alpha*K_f(-1) + phi*alpha*psi*r_f + phi*(1 -
            alpha)*L_f;
117     Y_f     = (1 - tau*k - g)*C_f + tau*k*I_f + e_G;
118
119     % Shocks
120     e_I     = rho_I*e_I(-1) + ee_I;
121     e_b     = rho_b*e_b(-1) + ee_b;
122     e_L     = rho_L*e_L(-1) + ee_L;
123     e_G     = rho_G*e_G(-1) + ee_G;
124     e_a     = rho_a*e_a(-1) + ee_a;
125     e_pi    = rho_pi*e_pi(-1)+ ee_pi;
126     one     = 0*one(-1);
127     end;
128 %-----------------------------------------------------------------------
```

```
% 4. Specify shocks
%-----------------------------------------------------------------

    shocks;
  var ee_b; stderr 1;
    var ee_I; stderr 1;
    var ee_a; stderr 1;
    var ee_L; stderr 1;
    var ee_G; stderr 1;
    var ee_pi; stderr 1;
    var n_Q; stderr sd_Q;
    var n_p; stderr sd_p;
    var n_w; stderr sd_w;
    var n_R; stderr sd_R;

  end;

%-----------------------------------------------------------------
% 5. Initial values
%-----------------------------------------------------------------
    initval;
%   C       = 0;
%   I       = 0;
%   Q       = 0;
%   K       = 0;
%   pi      = 0;
%   w       = 0;
%   L       = 0;
%   Y       = 0;
%   R       = 0;
%   r       = 0;
    e_a     = 0;
    e_I     = 0;
    e_b     = 0;
    e_L     = 0;
    e_G     = 0;
    e_pi    = 0;
  end;

%-----------------------------------------------------------------
% 6. Solution & Simulation
%-----------------------------------------------------------------
steady; check;
stoch_simul(periods=1000,irf=0,order=1,simul_replic=100);

%get state indices
ipred = M_.nstatic+(1:M_.nspred)';
%get state transition matrices
[A,B] = kalman_transition_matrix(oo_.dr,ipred,1:M_.nspred,M_.exo_nbr);
%get observable position in decision rules
obs_var=oo_.dr.inv_order_var(options_.varobs_id);
%get observation equation matrices
[C,D] = kalman_transition_matrix(oo_.dr,obs_var,1:M_.nspred,M_.exo_nbr);
```

```
182 temp = oo_.dr
183
184 %----------------------------------------------------------------------------
185 % 7. Estimation
186 %----------------------------------------------------------------------------
```

SmetsWouters.mod

Appendix C

DSGE estimation

```
# Estimating DSGE by Maximum Likelihood in Python by PyTorch
# Author: Michal Miktus
# Date: 20.03.2019

# Useful to debug: with torch.autograd.set_detect_anomaly(True):

# Import libraries

from Solution_function_PyTorch import Solution
import torch
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from torch import tensor, zeros, mm, randn, normal, cat, squeeze, unsqueeze
    , argmax, max
from torch.distributions import uniform
from Loss_function_PyTorch import loss_function

# Set printing options

sns.set()
np.set_printoptions(precision=3, suppress=True, linewidth=120)
torch.set_printoptions(precision=3, linewidth=120)
pd.set_option('float_format', lambda x: '%.3g' % x, )

# Declare path

path = "/Users/miktus/Documents/PSE/Dissertation/DSGE_estimation"

#
    ------------------------------------------------------------------------

# -- Define the model
#
    ------------------------------------------------------------------------
```

```
34
35 #  VARIABLES   [M,3] cell array: Variable name (case sensitive) ~ Variable
       type ~ Description
36 #               Variable type: 'X' ... Endogenous state variable
37 #                              'Y' ... Endogenous other (jump) variable
38 #                              'Z' ... Exogenous state variable
39 #                              'U' ... Innovation to exogenous state
       variable
40 #                              ''  ... Skip variable
41
42 variable_symbols = [r'$e_a$', r'$e_I$', r'$e_b$', r'$e_L$',
43                      r'$e_G$', r'$e_pi$', r'$pi$', r'$w$', r'$K$',
44                      r'$Q$', r'$I$', r'$C$', r'$R$', r'$r$', r'$L$',
45                      r'$Y$', r'$pi_f$', r'$w_f$', r'$K_f$', r'$Q_f$',
46                      r'$I_f$', r'$C_f$', r'$R_f$', r'$r_f$', r'$L_f$',
47                      r'$Y_f$', r'$ee_a$', r'$ee_I$', r'$ee_b$', r'$ee_L$', r
                            '$ee_G$', r'$ee_pi$', r'$n_p$', r'$n_Q$', r'$n_R$',
                             r'$n_w$', r'$one$']
48
49 variable_types = ['Z',     'Z',     'Z',     'Z',     'Z',     'Z',
50                    'X',     'X',     'X',     'X',     'X',     'X',
51                    'X',     'Y',     'Y',     'X',     'X',     'X',
52                    'X',     'X',     'X',     'X',     'X',     'Y',
53                    'Y',     'X',     'U',     'U',     'U',     'U',
54                    'U',     'U',     'Z',     'Z',     'Z',     'Z',
55                    'X', ]
56
57 variable_names = ['Aggregate productivity shock', 'Adjustment cost shock',
58                   'Preference shock', 'Labor supply shock',
59                   'Public spending shock', 'Inflation target', 'Inflation
                        rate', 'Real wages', 'Capital', 'Q-Tobin ratio', '
                        Investment',
60                   'Consumption', 'Monetary policy interest rate', 'Capital
                        rental rate', 'Labor', 'Output',
61                   'Flexible inflation rate', 'Flexible wages', 'Flexible
                        capital', 'Flexible Q-Tobin ratio',
62                   'Flexible investment', 'Flexible consumption', 'Flexible
                        monetary policy interest rate',
63                   'Flexible capital rental rate', 'Flexible labor', '
                        Flexible output', 'Aggregate productivity shock error
                        ',
64                   'Adjustment cost shock error', 'Preference shock error',
                        'Labor supply shock error', 'Public spending shock
                        error',
65                   'Inflation target error', 'Prices error', 'Tobin Q-ratio
                        error', 'Monetary policy interest rate error', 'Wages
                         error',
66                   'Temporary variable']
67
68
69 variables = pd.DataFrame({
70         'names': variable_names,
71         'types': variable_types,
```

```
72          'symbols': variable_symbols
73      })
74
75  var_endo_states = variables.loc[variables['types'] == 'X']
76  var_endo_controls = variables.loc[variables['types'] == 'Y']
77  var_exo = variables.loc[variables['types'] == 'Z']
78
79  #    EQUATIONS    [N,3] cell array: Equation type ~ Equation name ~ equation
80  #                 Equation type: 'D' ... Deterministic equation
81  #                                'E' ... Expectational equation
82  #                                'S' ... Shock equation
83  #                                ''  ... Skip equation
84
85  equation_formulas = [
86      '0 = - C(t) + (h/(1 + h))*C(t-1) + (1/(1 + h))*C(t+1) - ((1 - h)/((1 +
            h)*sigma_c))*(R(t) - pi(t+1)) + ((1 - h)/((1 + h)*sigma_c))*(e_b(t)
            -e_b(t+1))',
87      '0 = - I(t) + (1/(1 + betta))*I(t-1) + (betta/(1 + betta))*I(t+1) + (
            adj/(1 + betta))*Q(t) + (betta*e_I(t+1) - e_I(t))/(1 + betta)',
88      '0 = - Q(t)  -R(t) + pi(t+1) + ((1 - tau)/(1 - tau + r_k))*Q(t+1) + (
            r_k/(1 - tau + r_k))*r(t+1) + n_Q(t)',
89      '0 = - K(t) + (1 - tau)*K(t-1) + tau*I(t-1)', '0 = - pi(t) + (betta/(1
            + betta*gamma_p))*pi(t+1) + (gamma_p/(1 + betta*gamma_p))*pi(t-1) +
             (((1 - betta*xi_p)*(1 - xi_p))/((1 + betta*gamma_p)*xi_p))*(alphaa
            *r(t) + (1 - alphaa)*w(t) - e_a(t) + n_p(t))',
90      '0 = (-1 - ((1/(1 + betta))*((1 - betta*xi_w)*(1 - xi_w))/((1 + (1/
            lambda_w)*((1 + lambda_w)*sigma_L))*xi_w)))*w(t) + (betta/(1 +
            betta))*w(t+1) + (1/(1 + betta))*w(t-1)+ (betta/(1 + betta))*pi(t
            +1) - ((1 + betta*gamma_w)/(1 + betta))*pi(t) + (gamma_w/( 1 +
            betta))*pi(t-1) - ((1/(1 + betta))*((1 - betta*xi_w)*(1 - xi_w))
            /((1 + (1/lambda_w)*((1 + lambda_w)*sigma_L))*xi_w))*(- sigma_L*L(t
            ) - (sigma_c/(1 - h))*(C(t) - h*C(t-1)) - e_L(t) - n_w(t))', '0 = -
             L(t) + -w(t) + (1 + psi)*r(t) + K(t-1)',
91      '0 = - Y(t) + phi*e_a(t) + phi*alphaa*K(t-1) + phi*alphaa*psi*r(t) +
            phi*(1 - alphaa)*L(t)',
92      '0 = - Y(t) + (1 - tau*k - g)*C(t) + tau*k*I(t) + e_G(t)',
93      '0 = - R(t) + rho*R(t-1) + ((1 - rho)*r_pi - r_dpi)*pi(t-1) + (1 - rho)
            *(1 - r_pi)*e_pi(t) + Y(t)*((1-rho)*r_Y+r_dY) - Y_f(t)*((1-rho)*r_Y
            +r_dY) + r_dpi*pi(t) - r_dY*Y(t-1) - r_dY*Y_f(t-1) + n_R(t)',
94      '0 = - C_f(t) + (h/(1 + h))*C_f(t-1) + (1/(1 + h))*C_f(t+1) - ((1 - h)
            /((1 + h)*sigma_c))*(R_f(t) - pi_f(t+1)) + ((1 - h)/((1 + h)*
            sigma_c))*(e_b(t)-e_b(t+1))',
95      '0 = - I_f(t) + (1/(1 + betta))*I_f(t-1) + (betta/(1 + betta))*I_f(t+1)
             + (adj/(1 + betta))*Q_f(t) + (betta*e_I(t+1) - e_I(t))/(1 + betta)
            ',
96      '0 = - Q_f(t) - R_f(t) + pi_f(t+1) + ((1 - tau)/(1 - tau + r_k))*Q_f(t
            +1) + (r_k/(1 - tau + r_k))*r_f(t+1)',
97      '0 = - K_f(t) + (1 - tau)*K_f(t-1) + tau*I_f(t-1)',
98      '0 = alphaa*r_f(t)+(1-alphaa)*w_f(t) - e_a(t)',
99      '0 = - w_f(t) + sigma_L*L_f(t) + (sigma_c/(1 - h))*(C_f(t) - h*C_f(t-1)
            ) - e_L(t)',
100     '0 = - L_f(t) + -w_f(t) + (1 + psi)*r_f(t) + K_f(t-1)',
```

```
    '0 = - Y_f(t) + phi*e_a(t) + phi*alphaa*K_f(t-1) + phi*alphaa*psi*r_f(t) + phi*(1 - alphaa)*L_f(t)',
    '0 = - Y_f(t) + (1 - tau*k - g)*C_f(t) + tau*k*I_f(t) + e_G(t)',
    'e_I(t+1) = rho_I*e_I(t) + ee_I(t+1)',
    'e_b(t+1) = rho_b*e_b(t) + ee_b(t+1)',
    'e_L(t+1) = rho_L*e_L(t) + ee_L(t+1)',
    'e_G(t+1) = rho_G*e_G(t) + ee_G(t+1)',
    'e_a(t+1) = rho_a*e_a(t) + ee_a(t+1)',
    'e_pi(t+1) = rho_pi*e_pi(t)+ ee_pi(t+1)',
    'pi_f(t) = 0*one(t)',
    'one(t) = 0*one(t-1)',
    'n_p(t) = 0',
    'n_Q(t) = 0',
    'n_R(t) = 0',
    'n_w(t) = 0'
    ]

equation_type = ['E',    'E',    'E',    'E',    'E',    'E',
                 'D',    'D',    'E',    'E',    'E',    'E',
                 'E',    'E',    'E',    'E',    'D',    'D',
                 'E',    'S',    'S',    'S',    'S',    'S',
                 'S',    'E',    'E',    'S',    'S',    'S',
                 'S']

equation_names = ['Consumption', 'Investment', 'Q-equation', 'Capital', 'Prices', 'Wages', 'Labor demand', 'Production',
                  'Goods market', 'Monetary policy', 'Flexible Consumption', 'Flexible Investment', 'Flexible Q-equation',
                  'Flexible Capital', 'Flexible Prices', 'Flexible Wages', 'Flexible Labor demand', 'Flexible Production',
                  'Flexible Goods market', 'Aggregate productivity shock', 'Adjustment cost shock',
                  'Preference shock', 'Labor supply shock', 'Public spending shock', 'Inflation target', 'Temporary first equation', 'Temporary second equation', 'Prices error expectation', 'Tobin Q-ratio error expectation',
                  'Monetary policy interest rate error expectation', 'Wages error expectation']

equations = pd.DataFrame({
        'names': equation_names,
        'types': equation_type,
        'formulas': equation_formulas
    })

#-----------------------------------------------------------------------

# --- Setting parameters
#-----------------------------------------------------------------------
```

```
140
141 betta = tensor(0.99, requires_grad=True)
142 tau = tensor(0.025, requires_grad=True)
143 alphaa = tensor(0.3, requires_grad=True)
144 psi = tensor(1/0.169, requires_grad=True)
145 gamma_p = tensor(0.469, requires_grad=True)
146 gamma_w = tensor(0.763, requires_grad=True)
147 lambda_w = tensor(0.5, requires_grad=True)
148 xi_p = tensor(0.908, requires_grad=True)
149 xi_w = tensor(0.737, requires_grad=True)
150 sigma_L = tensor(2.4, requires_grad=True)
151 sigma_c = tensor(1.353, requires_grad=True)
152 h = tensor(0.573, requires_grad=True)
153 phi = tensor(1.408, requires_grad=True)
154 adj = tensor(1/6.771, requires_grad=True)
155 r_k = (1/betta)-1+tau
156 k = tensor(8.8, requires_grad=True)
157 g = tensor(0.18, requires_grad=True)
158 r_dpi = tensor(0.14, requires_grad=True)
159 r_Y = tensor(0.099, requires_grad=True)
160 r_dY = tensor(0.159, requires_grad=True)
161 rho = tensor(0.961, requires_grad=True)
162 r_pi = tensor(1.684, requires_grad=True)
163
164 rho_L = tensor(0.889, requires_grad=True)
165 rho_a = tensor(0.823, requires_grad=True)
166 rho_b = tensor(0.855, requires_grad=True)
167 rho_G = tensor(0.949, requires_grad=True)
168 rho_pi = tensor(0.924, requires_grad=True)
169 rho_I = tensor(0.927, requires_grad=True)
170
171 sd_R = tensor(0.081, requires_grad=True)
172 sd_p = tensor(0.16, requires_grad=True)
173 sd_w = tensor(0.289, requires_grad=True)
174 sd_Q = tensor(0.604, requires_grad=True)
175
176 #
    ------------------------------------------------------------------------

177 # --- Solution
178 #
    ------------------------------------------------------------------------

179
180 F, Q, nx, ny, nz = Solution(F_initial=zeros((31, 31)), betta=betta, tau=tau
        , alphaa=alphaa, psi=psi, gamma_p=gamma_p,
181                             gamma_w=gamma_w, lambda_w=lambda_w, xi_p=xi_p,
                                    xi_w=xi_w, sigma_L=sigma_L,
182                             sigma_c=sigma_c, h=h, phi=phi, adj=adj, k=k, g=
                                    g, r_dpi=r_dpi, r_Y=r_Y,
183                             r_dY=r_dY, rho=rho, r_pi=r_pi, rho_L=rho_L,
                                    rho_a=rho_a, rho_b=rho_b, rho_G=rho_G,
```

```
                                rho_pi=rho_pi, rho_I=rho_I, sd_R=sd_R, sd_p=
                                    sd_p, sd_w=sd_w, sd_Q=sd_Q)

#
    ------------------------------------------------------------------------

# --- Monte Carlo design
#
    ------------------------------------------------------------------------


monte_carlo_iter = 1
par1_monte = zeros(monte_carlo_iter)
par2_monte = zeros(monte_carlo_iter)
par3_monte = zeros(monte_carlo_iter)
par4_monte = zeros(monte_carlo_iter)
par5_monte = zeros(monte_carlo_iter)
loss_monte = zeros(monte_carlo_iter)

for iter in range(monte_carlo_iter):

    start_time = time.time()

    print("Number of Monte Carlo experiment: " + str(iter))

    #
        ------------------------------------------------------------------------

    # --- Simulation
    #
        ------------------------------------------------------------------------


    T = 300  # Number of periods to simulate

    epsilon = tensor([normal(mean=0, std=sd_Q), normal(mean=0, std=sd_p),
        normal(mean=0, std=sd_w),
                      normal(mean=0, std=sd_R)])
    epsilon = cat((randn(6), epsilon))

    X_sim = zeros((nx+ny+nz, T))
    X_sim[:, 0] = squeeze(mm(Q, torch.t(unsqueeze(epsilon, 0))))

    for t in range(1, T):
        epsilon = tensor([normal(mean=0, std=sd_Q),
                          normal(mean=0, std=sd_p), normal(mean=0, std=sd_w
                              ),
                          normal(mean=0, std=sd_R)])
        epsilon = cat((randn(6), epsilon))
        X_sim[:, t] = squeeze(mm(F, torch.t(unsqueeze(X_sim[:, t-1].clone()
            , 0))) + mm(Q, torch.t(unsqueeze(epsilon, 0))))

    # Discard first 100 observations
```

```
225
226     X_sim_discarded = X_sim[:, 100:]
227
228     # Plot for consumption
229
230     # plt.plot(X_sim[11,:].detach().numpy())
231     # plt.show()
232
233     #
            ------------------------------------------------------------------------
234     # --- Estimation
235     #
            ------------------------------------------------------------------------
236
237     # Observables: investment, consumption, labor
238     # Indexes: 10, 11, 14
239
240     no = 3  # Number of observables
241
242     observation_matrix = tensor([
243         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0.,
244             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
245         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
246             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
247         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
248             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]],
                requires_grad=True)
249
250     observables = X_sim_discarded[(10, 11, 14), :]
251
252     # Plot likelihood function depending on one specific parameter value,
            eq. beta
253
254     plot = False
255
256     if plot:
257         beta_to_graph = torch.arange(start=0., end=1.01, step=0.005,
                requires_grad=True)
258         likelihood_to_graph = []
259         likelihood_to_graph = [loss_function([beta, tensor(0.908), tensor
                (0.949), tensor(0.3), tensor(0.025)], F, observation_matrix,
                observables, X_sim_discarded[:, 0]) for beta in beta_to_graph]
260
261         plt.plot(beta_to_graph.detach().numpy(), likelihood_to_graph, color
                ='blue')
262         plt.xlabel('Beta')
263         plt.ylabel('Likelihood')
264         plt.title('Likelihood depending on beta')
265         # plt.show()
266         plt.savefig(path + '/Graphs/Likelihood depending on beta.pdf')
267
```

```
268     # Optimization
269
270     optimization = True
271
272     if optimization:
273
274         number_of_tries = 1
275         par1_final = zeros(number_of_tries)
276         par2_final = zeros(number_of_tries)
277         par3_final = zeros(number_of_tries)
278         par4_final = zeros(number_of_tries)
279         par5_final = zeros(number_of_tries)
280         loss_final = zeros(number_of_tries)
281
282         for j in range(number_of_tries):
283
284             print("The try number: " + str(j))
285
286             distribution = uniform.Uniform(torch.Tensor([3.]), torch.Tensor
                    ([10.]))
287             par1 = distribution.sample(torch.Size([1, 1])).requires_grad_()
288             par2 = distribution.sample(torch.Size([1, 1])).requires_grad_()
289             par3 = distribution.sample(torch.Size([1, 1])).requires_grad_()
290             distribution = uniform.Uniform(torch.Tensor([0.65]), torch.
                    Tensor([1.]))
291             par4 = distribution.sample(torch.Size([1, 1])).requires_grad_()
292             distribution = uniform.Uniform(torch.Tensor([0.16]), torch.
                    Tensor([0.22]))
293             par5 = distribution.sample(torch.Size([1, 1])).requires_grad_()
294
295             learning_rate = 1e-12
296             n_iter = 100
297
298             optimizer = torch.optim.SGD(params=[par1, par2, par3, par4,
                    par5], lr=learning_rate)
299
300             def closure():
301
302                 # Before the backward pass, use the optimizer object to
                        zero all of the
303                 # gradients for the Tensors it will update (which are the
                        learnable weights
304                 # of the model)
305                 optimizer.zero_grad()
306
307                 loss_value = loss_function([par1, par2, par3, par4, par5],
                        F, observation_matrix, observables, X_sim_discarded[:,
                        0])
308
309                 # Backward pass: compute gradient of the loss with respect
                        to model parameters
310
311                 loss_value.backward(retain_graph=True)
```

```
312
313                 return par1, par2, par3, par4, par5, loss_value
314
315             # Calling the step function on an Optimizer makes an update to
                    its parameters
316
317             for i in range(n_iter):
318                 print("Optimization step number: " + str(i))
319                 par1_vector, par2_vector, par3_vector, par4_vector,
                        par5_vector, loss_vector = optimizer.step(closure)
320
321             par1_final[j] = par1_vector
322             par2_final[j] = par2_vector
323             par3_final[j] = par3_vector
324             par4_final[j] = par4_vector
325             par5_final[j] = par5_vector
326             loss_final[j] = loss_vector
327
328         loss_monte[iter] = max(loss_final)
329         index = argmax(loss_final)
330         par1_monte[iter] = par1_final[index]
331         par2_monte[iter] = par2_final[index]
332         par3_monte[iter] = par3_final[index]
333         par4_monte[iter] = par4_final[index]
334         par5_monte[iter] = par5_final[index]
335
336         final = pd.DataFrame({
337             'Beta': par1_monte.detach().numpy()**2/(1+par1_monte.detach().
                        numpy()**2),
338             'varsigma_p': par2_monte.detach().numpy()**2/(1+par2_monte.
                        detach().numpy()**2),
339             'rho_G': par3_monte.detach().numpy()**2/(1+par3_monte.detach().
                        numpy()**2),
340             'alphaa': par4_monte.detach().numpy()**2/(1+par4_monte.detach()
                        .numpy()**2),
341             'tau': par5_monte.detach().numpy()**2/(1+par5_monte.detach().
                        numpy()**2),
342             'Likelihood': loss_monte.detach().numpy()
343         })
344
345         # Export to csv
346
347         # final.to_csv(path + "/Results/MC_All_F_100_100.csv", index=False)
348
349         print("--- %s seconds ---" % (time.time() - start_time))
```

DSGE_PyTorch.py

Loss function

```python
"""
Calculates the loss function to be further optimized
"""

from Solution_function_PyTorch import Solution
from Kalman_PyTorch import Kalman


def loss_function(params, F_initial, observation_matrix, observables, initial_Kalman):
    """
    Calculates the loss function to be further optimized

    Parameters
    ----------
    params : array_like or scalar(float)
        Parameters to be optimized for
    observation_matrix : array_like
        Array of indices of observable variables
    observables : array_like
        Array of observable variables
    """

    beta = params[0]**2 / (1 + params[0]**2)
    xi_p = params[1]**2 / (1 + params[1]**2)
    rho_G = params[2]**2 / (1 + params[2]**2)
    alphaa = params[3]**2 / (1 + params[3]**2)
    tau = params[4]**2 / (1 + params[4]**2)

    F, Q, nx, ny, nz = Solution(betta=beta, xi_p=xi_p, tau=tau, alphaa=alphaa, rho_G=rho_G, F_initial=F_initial)
    kalman = Kalman(A=F, C=Q, G=observation_matrix, x_hat=initial_Kalman)
    log_like = kalman.compute_loglikelihood(observables)

    return(log_like)
```

Loss_function_PyTorch.py

DSGE solution

```
"""
Function for solving DSGE models
Final version written by Michal Miktus, April 2019
"""

from torch import zeros, cat, eye
from Linear_Time_Iteration_PyTorch import Linear_Time_Iteration


def Solution(F_initial=zeros((31, 31)), betta=0.99, tau=0.025, alphaa=0.3, psi=1/0.169, gamma_p=0.469,
             gamma_w=0.763, lambda_w=0.5, xi_p=0.908, xi_w=0.737, sigma_L=2.4,
             sigma_c=1.353, h=0.573, phi=1.408, adj=1/6.771,
             k=8.8, g=0.18, r_dpi=0.14, r_Y=0.099, r_dY=0.159, rho=0.961, r_pi=1.684,
             rho_L=0.889, rho_a=0.823, rho_b=0.855, rho_G=0.949, rho_pi=0.924, rho_I=0.927,
             sd_R=0.081, sd_p=0.16, sd_w=0.289, sd_Q=0.604):

    r_k = (1 / betta) - 1 + tau

    #  ------------------------------------------------------------------------

    # --- Matrix form in spirit of Uhlig
    #  ------------------------------------------------------------------------


    # Structure of the model in Uhligs notation:
    #   0 = AA x(t) + BB x(t-1) + CC y(t) + DD z(t)
    #   0 = E_t [ FF x(t+1) + GG x(t) + HH x(t-1) + JJ y(t+1) + KK y(t) + LL z(t+1) + MM z(t)]
    #   z(t+1) = NN z(t) + epsilon(t+1),     with E_t [ epsilon(t+1) ] = 0
    #
    #   x(t)       : Endogenous state variables
    #   y(t)       : Endogenous other variables
    #   z(t)       : Exogenous state variables
    #   epsilon(t): Innovation to exogenous state variable

    AA = zeros((4, 17))
    BB = zeros((4, 17))
    CC = zeros((4, 4))
    DD = zeros((4, 10))
    FF = zeros((17, 17))
    GG = zeros((17, 17))
    HH = zeros((17, 17))
    JJ = zeros((17, 4))
    KK = zeros((17, 4))
    LL = zeros((17, 10))
```

```
    MM = zeros((17, 10))
    NN = zeros((10, 10))

    matrix_names = [AA, BB, CC, DD, FF, GG, HH, JJ, KK, LL, MM, NN]

    nx = AA.shape[1]
    ny = AA.shape[0]
    nz = DD.shape[1]

    # --- 1: Consumption

    GG[0, 5] = -1  # C(t)
    HH[0, 5] = (h/(1+h))  # C(t-1)
    FF[0, 5] = (1/(1+h))  # C(t+1)
    GG[0, 6] = -((1-h)/((1+h)*sigma_c))  # R(t)
    FF[0, 0] = ((1-h)/((1+h)*sigma_c))  # pi(t+1)
    MM[0, 2] = ((1-h)/((1+h)*sigma_c))  # e_b(t)
    LL[0, 2] = -((1-h)/((1+h)*sigma_c))  # e_b(t+1)

    # --- 2:  Investment

    GG[1, 4] = -1  # I(t)
    HH[1, 4] = (1/(1+betta))  # I(t-1)
    FF[1, 4] = (betta/(1+betta))  # I(t+1)
    GG[1, 3] = (adj/(1+betta))  # Q(t)
    LL[1, 1] = (1+betta)*betta  # e_I(t+1)
    MM[1, 1] = -(1+betta)  # e_I(t)

    # --- 3:  Q-equation

    GG[2, 3] = -1  # Q(t)
    GG[2, 6] = -1  # R(t)
    FF[2, 0] = 1  # pi(t+1)
    FF[2, 3] = ((1-tau)/(1-tau+r_k))  # Q(t+1)
    JJ[2, 0] = (r_k/(1-tau+r_k))  # r(t+1)
    MM[2, 7] = 1  # n_Q(t)

    # --- 4:  Capital

    GG[3, 2] = -1  # K(t)
    HH[3, 2] = (1-tau)  # K(t-1)
    HH[3, 4] = tau  # I(t-1)

    # --- 5:  Prices

    GG[4, 0] = -1  # pi(t)
    FF[4, 0] = (betta/(1+betta*gamma_p))  # pi(t+1)
    HH[4, 0] = (gamma_p/(1+betta*gamma_p))  # pi(t-1)
    KK[4, 0] = (((1-betta*xi_p)*(1-xi_p))/((1+betta*gamma_p)*xi_p))*alphaa
        # r(t)
    GG[4, 1] = (((1-betta*xi_p)*(1-xi_p))/((1+betta*gamma_p)*xi_p))*(1-
        alphaa)  # w(t)
```

```
93      MM[4, 0] = -(((1-betta*xi_p)*(1-xi_p))/((1+betta*gamma_p)*xi_p))  # e_a
            (t)
94      MM[4, 6] = (((1-betta*xi_p)*(1-xi_p))/((1+betta*gamma_p)*xi_p))  # n_p(
            t)
95
96      # --- 6:  Wages
97
98      GG[5, 1] = (-1-((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w
            )*((1+lambda_w)*sigma_L))*xi_w)))  # w(t)
99      FF[5, 1] = (betta/(1+betta))  # w(t+1)
100     HH[5, 1] = (1/(1+betta))  # w(t-1)
101     FF[5, 0] = (betta/(1+betta))  # pi(t+1)
102     GG[5, 0] = -((1+betta*gamma_w)/(1+betta))  # pi(t)
103     HH[5, 0] = (gamma_w/(1+betta))  # pi(t-1)
104     KK[
105         5, 1] = ((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w)
                *((1+lambda_w)*sigma_L))*xi_w))*sigma_L  # L(t)
106     HH[
107         5, 5] = -((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w)
                *((1+lambda_w)*sigma_L))*xi_w))*(sigma_c/(1-h))*h  # C(t-1)
108     GG[
109         5, 5] = ((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w)
                *((1+lambda_w)*sigma_L))*xi_w))*(sigma_c/(1-h))  # C(t)
110     MM[5, 3] = ((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w)
            *((1+lambda_w)*sigma_L))*xi_w))  # e_L(t)
111     MM[5, 9] = ((1/(1+betta))*((1-betta*xi_w)*(1-xi_w))/((1+(1/lambda_w)
            *((1+lambda_w)*sigma_L))*xi_w))  # n_w(t)
112
113     # --- 7:  Labor demand
114
115     CC[0, 1] = -1  # L(t)
116     AA[0, 1] = -1  # w(t)
117     CC[0, 0] = (1+psi)  # r(t)
118     BB[0, 2] = 1  # K(t-1)
119
120     # --- 8:  Production
121
122     AA[1, 7] = -1  # Y(t)
123     DD[1, 0] = phi  # e_a(t)
124     BB[1, 2] = phi*alphaa  # K(t-1)
125     CC[1, 0] = phi*alphaa*psi  # r(t)
126     CC[1, 1] = phi*(1-alphaa)  # L(t)
127
128     # --- 9:      Goods market
129
130     GG[6, 7] = -1  # Y(t)
131     GG[6, 5] = (1-tau*k-g)  # C(t)
132     GG[6, 4] = tau*k  # I(t)
133     MM[6, 4] = 1  # e_G(t)
134
135     # --- 10: Monetary policy
136
137     GG[7, 6] = -1  # R(t)
```

```
    HH[7, 6] = rho  # R(t-1)
    HH[7, 0] = ((1-rho)*r_pi-r_dpi)  # pi(t-1)
    MM[7, 5] = (1-rho)*(1-r_pi)  # e_pi(t)
    GG[7, 7] = ((1-rho)*r_Y+r_dY)  # Y(t)
    GG[7, 15] = -((1-rho)*r_Y+r_dY)  # Y_f(t)
    GG[7, 0] = r_dpi  # pi(t)
    HH[7, 7] = -r_dY  # Y(t-1)
    HH[7, 15] = -r_dY  # Y_f(t-1)
    MM[7, 8] = 1  # n_R(t)

    # --- 11:  Flexible Consumption

    GG[8, 13] = -1  # C_f(t)
    HH[8, 13] = (h/(1+h))  # C_f(t-1)
    FF[8, 13] = (1/(1+h))  # C_f(t+1)
    GG[8, 14] = -((1-h)/((1+h)*sigma_c))  # R_f(t)
    FF[8, 8] = ((1-h)/((1+h)*sigma_c))  # pi_f(t+1)
    MM[8, 2] = ((1-h)/((1+h)*sigma_c))  # e_b(t)
    LL[8, 2] = -((1-h)/((1+h)*sigma_c))  # e_b(t+1)

    # --- 12:  Flexible Investment

    GG[9, 12] = -1  # I_f(t)
    HH[9, 12] = (1/(1+betta))  # I_f(t-1)
    FF[9, 12] = (betta/(1+betta))  # I_f(t+1)
    GG[9, 11] = (adj/(1+betta))  # Q_f(t)
    LL[9, 1] = (1+betta)*betta  # e_I(t+1)
    MM[9, 1] = -(1+betta)  # e_I(t)

    # --- 13:  Flexible Q-equation

    GG[10, 11] = -1  # Q_f(t)
    GG[10, 14] = -1  # R_f(t)
    FF[10, 8] = 1  # pi_f(t+1)
    FF[10, 11] = ((1-tau)/(1-tau+r_k))  # Q_f(t+1)
    JJ[10, 2] = (r_k/(1-tau+r_k))  # r_f(t+1)

    # --- 14:  Flexible Capital

    GG[11, 10] = -1  # K_f(t)
    HH[11, 10] = (1-tau)  # K_f(t-1)
    HH[11, 12] = tau  # I_f(t-1)

    # --- 15:  Flexible Prices

    KK[12, 2] = alphaa  # r_f(t)
    GG[12, 9] = (1-alphaa)  # w_f(t)
    MM[12, 0] = -1  # e_a(t)

    # --- 16:  Flexible Wages

    GG[13, 9] = -1  # w_f(t)
    KK[13, 3] = sigma_L  # L_f(t)
```

```
    HH[13, 13] = -(sigma_c/(1-h))*h  # C_f(t-1)
    GG[13, 13] = (sigma_c/(1-h))  # C_f(t)
    MM[13, 3] = -1  # e_L(t)

    # --- 17:  Flexible Labor demand

    CC[2, 3] = -1  # L_f(t)
    AA[2, 9] = -1  # w_f(t)
    CC[2, 2] = (1+psi)  # r_f(t)
    BB[2, 10] = 1  # K_f(t-1)

    # --- 18:  Flexible Production

    AA[3, 15] = -1  # Y_f(t)
    DD[3, 0] = phi  # e_a(t)
    BB[3, 10] = phi*alphaa  # K_f(t-1)
    CC[3, 2] = phi*alphaa*psi  # r_f(t)
    CC[3, 3] = phi*(1-alphaa)  # L_f(t)

    # --- 19:  Flexible Goods market

    GG[14, 15] = -1  # Y_f(t)
    GG[14, 13] = (1-tau*k-g)  # C_f(t)
    GG[14, 12] = tau*k  # I_f(t)
    MM[14, 4] = 1  # e_G(t)

    # --- 20:   Aggregate productivity shock

    NN[0, 1] = rho_I  # e_I(t)

    # --- 21:   Adjustment cost shock

    NN[1, 2] = rho_b  # e_b(t)

    # --- 22:   Preference shock

    NN[2, 3] = rho_L  # e_L(t)

    # --- 23:   Labor supply shock

    NN[3, 4] = rho_G  # e_G(t)

    # --- 24:   Public spending shock

    NN[4, 0] = rho_a  # e_a(t)

    # --- 25:   Inflation target

    NN[5, 5] = rho_pi  # e_pi(t)

    # --- 26: Temporary first equation

    GG[15, 8] = -1  # pi_f(t)
```

```
    GG[15, 16] = 0  # one(t)

    # --- 27: Temporary second equation

    GG[16, 16] = -1  # one(t)
    HH[16, 16] = 0  # one(t-1)

    # --- 26: Prices error expectation

    NN[6, 6] = 0  # n_p(t)

    # --- 27: Tobin Q-ratio error expectation

    NN[7, 7] = 0  # n_Q(t)

    # --- 28: Monetary policy interest rate error expectation

    NN[8, 8] = 0  # n_R(t)

    # --- 29: Wages error expectation

    NN[9, 9] = 0  # n_w(t)

    #
        ------------------------------------------------------------------------

    # --- Matrix form in spirit of Linear Time Iteration
    #
        ------------------------------------------------------------------------


    A = cat((cat((matrix_names[1], zeros((ny, ny)), zeros((ny, nz))), 1),
        cat((matrix_names[6], zeros((nx, ny)), zeros((nx, nz))), 1),
            cat((zeros((nz, nx)), zeros((nz, ny)), matrix_names[11]), 1)),
                0)
    B = cat((cat((matrix_names[0], matrix_names[2], matrix_names[3]), 1),
        cat((matrix_names[5], matrix_names[8], matrix_names[10]), 1),
            cat((zeros((nz, nx)), zeros((nz, ny)), -eye((nz))), 1)), 0)
    C = cat((zeros((ny, nx + ny+nz)), cat((matrix_names[4], matrix_names
        [7], matrix_names[9]), 1), zeros((nz, nx+ny+nz))), 0)

    F_iter, Q_iter = Linear_Time_Iteration(A, B, C, F_initial, 1e-16, 1e-4)

    Q_iter = Q_iter[:, -10:]

    return F_iter, Q_iter, nx, ny, nz
```

Solution_function_PyTorch.py

Kalman filter

```
"""
Implements the Kalman filter for a linear Gaussian state space model.
"""

from torch import mm, inverse, t, zeros, eye, log, det, abs
from textwrap import dedent
import math
import torch


class Kalman:
    """
    Implements the Kalman filter for the Gaussian state space model
    .. math::
        x_{t+1} = A x_t + C w_{t+1} \\
        y_t = G x_t + H v_t
    Here :math:`x_t` is the hidden state and :math:`y_t` is the measurement
        .
    The shocks :math:`w_t` and :math:`v_t` are iid standard normals. Below
    we use the notation
    .. math::
        Q := CC'
        R := HH'
    Parameters
    ----------
    A : array_like or scalar(float)
        Part of the state transition equation.  It should be `n x n`
    C : array_like or scalar(float)
        Part of the state transition equation.  It should be `n x m`
    G : array_like or scalar(float)
        Part of the observation equation.  It should be `k x n`
    H : array_like or scalar(float), optional(default=None)
        Part of the observation equation.  It should be `k x l`
    x_hat : scalar(float) or array_like(float), optional(default=None)
        An n x 1 array representing the mean x_hat of the
        prior/predictive density.  Set to zero if not supplied.
    Sigma : scalar(float) or array_like(float), optional(default=None)
        An n x n array representing the covariance matrix Sigma of
        the prior/predictive density.  Must be positive definite.
        Set to the identity if not supplied.
    """

    def __init__(self, A, C, G, H=None, x_hat=None, Sigma=None):
        self.A = A
        self.C = C
        self.G = G
        self.Q = mm(self.C, t(self.C))
        self.m = self.C.shape[1]
        self.k, self.n = self.G.shape

        if H is None:
```

```
51              self.H = zeros((self.k, self.n))
52          else:
53              self.H = H
54          if Sigma is None:
55              self.Sigma = eye(self.n)
56          else:
57              self.Sigma = Sigma
58          if x_hat is None:
59              self.x_hat = zeros((self.n, 1))
60          else:
61              self.x_hat = torch.reshape(x_hat, (self.n, 1))
62
63          self.R = mm(self.H, t(self.H))
64
65      def __repr__(self):
66          return self.__str__()
67
68      def prior_to_filtered(self, y):
69          r"""
70          Updates the moments (x_hat, Sigma) of the time t prior to the
71          time t filtering distribution, using current measurement :math:`y_t
                `.
72          The updates are according to
73          .. math::
74              \hat{x}^F = \hat{x} + \Sigma G' (G \Sigma G' + R)^{-1}
75                  (y - G \hat{x})
76              \Sigma^F = \Sigma - \Sigma G' (G \Sigma G' + R)^{-1} G
77                  \Sigma
78          Parameters
79          ----------
80          y : scalar or array_like(float)
81              The current measurement
82          """
83
84          # Simplify notation
85          G, H = self.G, self.H
86          R = mm(H, t(H))
87
88          # Update
89          E = mm(self.Sigma, t(G))
90          F = mm(mm(G, self.Sigma), t(G)) + R
91          M = mm(E, inverse(F))
92          self.x_hat = self.x_hat + mm(M, (y - mm(G, self.x_hat)))
93          self.Sigma = self.Sigma - mm(M, mm(G, self.Sigma))
94
95      def filtered_to_forecast(self):
96          """
97          Updates the moments of the time t filtering distribution to the
98          moments of the predictive distribution, which becomes the time
99          t+1 prior
100         """
101
102         # Simplify notation
```

```
        A, C = self.A, self.C
        Q = mm(C, t(C))

        # Update
        self.x_hat = mm(A, self.x_hat)
        self.Sigma = mm(A, mm(self.Sigma, t(A))) + Q

    def update(self, y):
        """
        Updates x_hat and Sigma given k x 1 nparray y.  The full
        update, from one period to the next
        Parameters
        ----------
        y : np.ndarray
            A k x 1 ndarray y representing the current measurement
        """
        self.prior_to_filtered(y)
        self.filtered_to_forecast()

    def __str__(self):
        m = """\
        Kalman filter:
          - dimension of state space          : {n}
          - dimension of observation equation : {k}
        """
        return dedent(m.format(n=self.n, k=self.k))

    def log_likelihood(self, y):
        """
        Computes log-likelihood of period ``t``
        Parameters
        ----------
        y : np.ndarray
            A k x 1 ndarray y representing the current measurement
        """

        eta = y - mm(self.G, self.x_hat)  # forecast error
        P = mm(self.G, mm(self.Sigma, t(self.G))) + self.R  # covariance
            matrix of forecast error
        logL = -(y.shape[0] * (log(2*torch.tensor(math.pi)) + log(abs(det(
            P)))) + torch.sum(mm(t(eta), mm(inverse(P), eta))))/2
        return logL

    def compute_loglikelihood(self, y):
        """
        Computes log-likelihood of entire observations
        Parameters
        ----------
        y : np.ndarray
            n x T matrix of observed data.
            n is the number of observed variables in one period.
           Each column is a vector of observations at each period.
        """
```

```
154         T = y.shape[1]
155         logL = 0
156 
157         # Forecast and update
158 
159         for period in range(1, T):
160             logL = logL + self.log_likelihood(y[:, period-1])
161             self.update(y[:, period-1])
162 
163         return logL
```

Kalman_PyTorch.py

Linear Time Iteration algorithm

```
"""
Function implementing the Linear Time Iteration algorithm for solving DSGE models
in the spirit of P. Rendahl (2017)
Final version written by Michal Miktus, April 2019
"""

from torch import eye, abs, max, gesv, inverse, mm, matrix_power, zeros


def Linear_Time_Iteration(A, B, C, F_initial, mu, epsilon):
    """
    This function will find the linear time iteration solution to the system of equations in the form of
    AX(-1) + BX + CE[X(+1)] + epsilon = 0
    with a recursive solution in the form of X = FX(-1) + Q*epsilon
    Parameters
    ----------
    A : torch, array_like, dtype=float
        The matrix of coefficients next to endogenous variables entering with a lag
    B : torch, array_like, dtype=float
        The matrix of coefficients next to endogenous, contemporanous variables
    C : torch, array_like, dtype=float
        The matrix of coefficients next to endogenous variables entering with a lead
    F : torch, array_like, dtype=float
        The initial guess for F
    mu : number, dtype=float
        Small positive real number to be multiplied by a conformable identity matrix
    epsilon : number, dtype=float
        Threshold value, should be set to a small value like 1e-16
    Returns
    -------
    F : torch, array_like, dtype=float
        The matrix of coefficients next to the endogenous variable in the solution
    Q : torch, array_like, dtype=float
        The matrix of coefficients next to the disturbance term in the solution
    Notes
    -----
    """

    F = F_initial
    S = zeros(*A.shape)

    # F.requires_grad_()
    # S.requires_grad_()
```

```
44
45      Id = eye(*A.shape) * mu
46      Ch = C
47      Bh = (B + 2 * mm(C, Id))
48      Ah = (mm(C, matrix_power(Id, 2)) + mm(B, Id) + A)
49
50      metric = 1
51      iter = 1
52
53      while metric > epsilon:
54          if iter % 10000 == 0:
55              print(iter)
56          F = -gesv(Ah, (Bh + mm(Ch, F)))[0]
57          S = -gesv(Ch, (Bh + mm(Ah, S)))[0]
58          metric1 = max(abs(Ah + mm(Bh, F) + mm(Ch, (mm(F, F)))))
59          metric2 = max(abs(mm(Ah, mm(S, S)) + mm(Bh, S) + Ch))
60          metric = max(metric1, metric2)
61          iter += 1
62          if iter > 1000000:
63              break
64
65      # eig_F = max(abs(eig(F)[0]))
66      # eig_S = max(abs(eig(S)[0]))
67      # eig_stable = max(abs(eig(mm(inverse(mm(Ah, F) + Bh), Ah))[0]))
68
69      # if (eig_F > 1) or (eig_S > 1) or (mu > 1-eig_S):
70      #     print('Conditions of Proposition 3 violated')
71
72      # if (eig_F > 1) or (eig_stable > 1):
73      #     print('Conditions for stable and unique solution violated')
74
75      F = F + Id
76      Q = -inverse(B + mm(C, F))
77
78      return F, Q
```

Linear_Time_Iteration_PyTorch.py