Project Proposal: Machine Learning for Economics

Zhenqi (Luke) Hu

December 25, 2023

1 Introduction

This project is aimed at replicating the results of Childers et al. (2022). This paper proposes a methodology to take dynamic stochastic general equilibrium (DSGE) models to the data based on the combination of differentiable state-space models and the Hamiltonian Monte Carlo (HMC) algorithm.

Differentiable State-Space DSGE Models

The class of DSGE models is a standard tool for macroeconomic analysis and includes various models based on different assumptions (e.g. real business cycle models (RBC), New Keynesian models (NK), etc.)[1]. Following Schmitt-Grohé and Uribe (2004), the general form of a DSGE model can be written as a system of equations:

$$E_t H(y_{t+1}, y_t, x_{t+1}, x_t) = 0$$

where x denotes the vector of state variables that determine the economy's evolution, and y denotes the vector of endogenous variables that are determined by the state variables (control variables). H is the system of equations that characterize the model. The expectation operator E_t is the conditional expectation given the information set at time t.

Since the model is Markov, we can omit the time subscripts and write the model as:

$$EH(y', y, x', x) = 0$$

[1]Fernández-Villaverde and Guerrón-Quintana (2021) is a comprehensive review paper for the general framework and estimation of DSGE model.

where prime denotes the next period value. The solutions of this class of models can be written as:

$$\begin{aligned} y &= g(x;\theta) && \text{(Policy Equation)} \\ x' &= h(x;\theta) + \eta\epsilon' && \text{(Transition Equation)} \end{aligned}$$

where θ is the vector of parameters, ϵ is a vector of i.i.d. shocks, η is a matrix of shock loadings. The policy equation $g(\cdot)$ describes the optimal behavior of the agents, and the transition equation $h(\cdot)$ characterizes the law of motion of the state variables.

The transition equation is formed as the first leg of the state-space representation[2], while the second leg is the measurement equation:

$$\begin{aligned} z &= q(x, v; \theta) \\ &= Q \cdot [y \quad x]^T + v \quad \text{(Measurement Equation)} \end{aligned}$$

where z may contain one or more states, as well as one or more control variables. v is a vector of i.i.d. measurement errors or shocks to observables other than the states.

Hence, the objective of estimating DSGE models is to estimate the parameters θ, as well as the unobserved states x, given the observed data z, that is, the joint posterior distribution $p(\theta, x|z)$.

Hamiltonian Monte Carlo Estimation

Monte Carlo Markov Chain (MCMC) methods are widely used in Bayesian estimation of state-space models. Gelman et al. (2013) is a comprehensive reference for Bayesian estimation and MCMC methods, and for researchers who are from an economics or finance background (like me), Johannes and Polson (2010) is a good starting point.

Hamiltonian Monte Carlo (HMC) sampler is a variant of MCMC methods, which borrows the idea from Hamiltonian dynamics in physics and improves the efficiency of the sampling process compared to the traditional Metropolis-Hastings (MH) algorithm. For conciseness, I will not go into the details of HMC, and refer the readers to Gelman et al. (2013) and Neal et al. (2011) for more information [3]. However, I will emphasize some important features of HMC:

1. Based on Baye's rule, the HMC sampler requires the prior distribution $p(\theta)$ and the likelihood function $p(z|\theta)$ as inputs, and then generates samples from

[2]Hamilton (1994) provides a detailed introduction to state-space representation and Kalman filter in Chapter 13.

[3]Stan Reference Manual also provides a short introduction of HMC that is easy to understand.

the posterior distribution $p(\theta|z)$. The prior distribution is usually easy to specify, while the likelihood function is the key to the success of HMC and needs to be carefully constructed from the model. (See the parts one and two of the replication design.)

2. One feature of HMC is that it requires the gradient of the log-posterior density, so automatic differentiation (AD) plays an important role in the implementation, which is also an embedded feature of the TensorFlow ecosystem.

3. Besides the prior and likelihood, there are two hyperparameters in HMC: the step size ϵ and the number of steps L. The optimal values of these two hyperparameters need to be carefully tuned. (See the part three of the replication design.)

Since we need the likelihood function to implement the HMC sampler, the authors propose two alternative approaches to construct the likelihood in a typical state-space model: (1) The Marginal Likelihood (with Kalman Filter to infer the unobserved states x); (2) The Joint Likelihood (with the unobserved states x as additional parameters). The former is used in the benchmark random walk Metropolis-Hastings (RWMH) algorithm, while the latter is used in the HMC sampler. See Section 2.2-2.3 of the paper for more details.

2 Replication Design

The authors provide a Github repo, HMCExamples.jl, which is written in Julia. My replicating project will be based on their structure and methodology, but rewritten using Python. I briefly outline the steps below.

Part One: Build the Model Symbolically

The author developed a Julia package DifferentiableStateSpaceModels.jl to handle first- and second-order solutions to state-space models, gradients, and sensitivity analysis. They build the model symbolically using Symbolics.jl, and then generate the symbolic derivatives for the model to use in the perturbation solution algorithms. All the symbolic results are then converted to Julia functions and saved in modules.

In the Python ecosystem, Sympy package provides similar tools for symbolic computation. The replication procedure is straightforward, after constructing the symbolic model using the 'Symbol', 'Functions', and 'Matrix' classes in Sympy, we can use 'diff' and 'hessian' functions to compute the symbolic derivatives. In the end, we can use 'lambdify' function to convert SymPy expressions to equivalent numeric functions, which will be used in the later steps.

It is worth noting that Julia has some features that are not available in Python, such as multiple dispatch and meta-programming. Hence, the replication may not be exactly the same as the original package.

Part Two: Simulation and Likelihood Evaluation

In the next step, the generated state-space model, given a set of parameters, will be used to:

1. Simulate the model and generate the simulated data.

2. Evaluate the likelihood of the model given the data.

The author utilized the DifferentialEquations.jl package to achieve their goals. In the Python ecosystem, the Scipy plays a similar role. First, we need the sovler from scipy.integrate and scipy.linalg to solve the differential equations (i.e. the Sylvester equation). Then, we can use the scipy.stats, as well as the TensorFlow Probability to evaluate the likelihood, for both a marginalized approach with a Kalman filter and a joint approach, and for both Gaussian and No-Gaussian shocks. The calculated log-likelihood will be a key input to construct the HMC sampler in the next step.

Part Three: Bayesian Estimation using HMC Sampler

Finally, we need to implement the Bayesian estimation using the HMC sampler. The author used the Turing.jl probabilistic programming language, and Zygote.jl for automatic differentiation. In the Python ecosystem, there are many alternatives, while we are required to use TensorFlow Probability in this project.

The tfp.mcmc package provides a set of Markov chain Monte Carlo (MCMC) algorithms for sampling from probability distributions, including a basic HMC sampler tfp.mcmc.HamiltonianMonteCarlo, as well as a more advanced type No-U-Turn Sampler (NUTS), tfp.mcmc.NoUTurnSampler. The later variant, proposed by Hoffman et al. (2014), endogenously pick ϵ and L with sample adaptations, and is the main choice for this paper. An alternative is to wrap the basic HMC sampler in a tfp.mcmc.SimpleStepSizeAdaptation "meta-kernel", which adaptively tunes the step size of the inner kernel.

Benchmark

For comparing the performance of our HMC implementation in TensorFlow Probability, we need several benchmarks.

The first straightforward one is to compare the efficiency and robustness of our implementation with the original Julia implementation. The authors present their results for three DSGE models: a simple real business cycle (RBC) model, a real small open economy model based on Schmitt-Grohé and Uribe (2003) (SGU), and a medium-scale New Keynesian DSGE model, namely Fernández-Villaverde and Guerrón-Quintana (2021) (FVGQ).

Besides, we can also compare the performance with Dynare, the most popular software for estimating DSGE models and can act as a strong benchmark. Both the Julia and Dynare implementations are provided in the authors' Github repo.

Testing

To ensure the correctness of our implementation, we will follow a systematic testing plan:

1. **Unit Testing:** We will test individual components of the code to ensure they function as expected in isolation.

2. **Integration Testing:** After unit testing, we will test the interaction between different components of the code.

3. **Functional Testing:** We will test the functionality of the software as a whole to ensure it meets the requirements.

4. **Performance Testing:** We will test the performance of the software under various conditions to ensure it meets the performance criteria.

5. **Comparison with Benchmark:** We will compare our results with those obtained from Dynare to validate our implementation.

Each test will be documented with its purpose, expected results, and actual results. Any discrepancies will be logged and addressed before proceeding to the next phase of testing.

3 Future Plan

Up to now, I have finished the first part of the replication design, and the symbolic model is built using Sympy. The next step is to simulate the model and evaluate the likelihood. I will keep updating the project's progress on Github, and the final report will be submitted on time.

Here is the snapshot for the "__init__.py" file of the generated RBC model:

References

Childers, David, Jesús Fernández-Villaverde, Jesse Perla, Christopher Rackauckas, and Peifan Wu, 2022, Differentiable State-Space Models and Hamiltonian Monte Carlo Estimation.

Fernández-Villaverde, Jesús, and Pablo A. Guerrón-Quintana, 2021, Estimating DSGE Models: Recent Advances and Future Challenges, *Annual Review of Economics* 13, 229–252.

Gelman, A., J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin, 2013, *Bayesian Data Analysis, Third Edition*, Chapman & Hall/CRC Texts in Statistical Science (Taylor & Francis).

Hamilton, James Douglas, 1994, *Time Series Analysis* (Princeton university press).

Hoffman, Matthew D, Andrew Gelman, et al., 2014, The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo., *J. Mach. Learn. Res.* 15, 1593–1623.

Johannes, Michael, and Nicholas Polson, 2010, CHAPTER 13 - MCMC Methods for Continuous-Time Financial Econometrics, in YACINE Aït-sahalia, and LARS PETER Hansen, eds., *Handbook of Financial Econometrics: Applications*, volume 2 of *Handbooks in Finance*, 1–72 (Elsevier, San Diego).

Neal, Radford M, et al., 2011, Mcmc using hamiltonian dynamics, *Handbook of markov chain monte carlo* 2, 2.

Schmitt-Grohé, Stephanie, and Martın Uribe, 2003, Closing small open economy models, *Journal of international Economics* 61, 163–185.

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