Differentiable State-Space Models and Hamiltonian Monte Carlo Estimation*

David Childers
CMU

Jesús Fernández-Villaverde
University of Pennsylvania

Jesse Perla
UBC

Christopher Rackauckas
MIT and Maryland

Peifan Wu
Amazon

October 6, 2022

Abstract

We propose 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) sampler. First, we introduce a method for implicit automatic differentiation of perturbation solutions of DSGE models with respect to the model's parameters. We can use the resulting output for various tasks requiring gradients, such as building an HMC sampler, to estimate first- and second-order approximations of DSGE models. The availability of derivatives also enables a general filter-free method to estimate nonlinear, non-Gaussian DSGE models by sampling the joint likelihood of parameters and latent states. We show that the gradient-based joint likelihood sampling approach is superior in efficiency and robustness to standard Metropolis-Hastings samplers by estimating a canonical real business cycle model, a real small open economy model, and a medium-scale New Keynesian DSGE model.

*All errors are our own. We thank Cameron Pfiffer for extensive participation and feedback at earlier stages of this project. We would also like to thank seminar participants at UBC, Georgetown, the Society for Economic Dynamics 2021 meeting, and the China International Conference in Macroeconomics 2022 for helpful comments and suggestions. Arnav Sood, Jan Rosa, Andrew Owens, Seb Gomez, and James Yu provided research assistance. Peifan Wu's contribution to the paper was completed prior to his employment by Amazon, and this paper does not represent Amazon's views. We gratefully acknowledge the computational resources provided by UBC Advanced Research Computing, UBC ARC Sockeye, and UBC ARC RONIN. All code is available at https://github.com/HighDimensionalEconLab/HMCExamples.jl.

1 Introduction

In this paper, we propose a methodology to take dynamic stochastic general equilibrium (DSGE) models (and related dynamic equilibrium models in other fields) to the data based on the combination of differentiable state-space models and the Hamiltonian Monte Carlo (HMC) sampler. Differentiable state-space models allow us to implement HMC by providing an easy method to differentiate the perturbation solutions of DSGE models. HMC has two great advantages with respect to other Markov Chain Monte Carlo (MCMC) methods. First, it draws much more efficiently from the posterior of a DSGE model than existing alternatives. Second, HMC scales very well with the dimension of the model. Hence, we can draw from the joint distribution of parameters and latent states of the model simultaneously without having to resort to a filter to marginalize the latent state variables of the model.

Let us unpack the many ideas in the previous paragraph. DSGE models are one of the major workhorses of modern macroeconomics. Thus, it is not a surprise that an extensive strand of the literature has focused on how to take these models to the data, both from a classical and from a Bayesian perspective (see the reviews in Fernández-Villaverde et al., 2016, and Fernández-Villaverde and Guerrón-Quintana, 2021).

While the Bayesian approach is particularly popular, following it is not without peril. Since DSGE models rarely have a closed-form solution, we need to resort to numerical approximations to evaluate their moments or likelihood functions and sample from them—for instance, to find posterior distributions of parameters of interest.

Despite many years of research, some questions remain open regarding these tasks. First, implementing the Bayesian approach usually requires an MCMC sampler, the most popular of which is the Random Walk Metropolis-Hastings (RWMH) algorithm. It is well known that RWMH (and many of the improvements built on top of it) suffers from high autocorrelation across draws. Due to this high correlation, the effective sample size is small compared to the total draws.[1] Even if we run the sampler 100,000 times, we might have only the equivalent of around 1,000 samples coming from a hypothetical independent Monte Carlo sampler (which, unfortunately, we cannot design). Furthermore, the sampling results might be sensitive to the choice of starting points, even after long runs of the sampler. These drawbacks are particularly binding when we deal with DSGE models that are richly parameterized, as the proportion of effective samples can drop rapidly as the number of dimensions increases.

Second, evaluating the likelihood function implied by the solution of a DSGE model is usually done by some filter, such as the Kalman filter—when we have a first-order

[1] Effective sample size (ESS) is measured as the number of draws divided by the estimated long-run autocorrelation of the chain (Gelman et al., 2013, Ch 11.5).

perturbation and Gaussian shocks—or the particle filter—when we deal with nonlinear solutions and/or non-Gaussian shocks (Fernández-Villaverde and Rubio-Ramírez, 2007). The filters deliver the marginal likelihood of the model with respect to its parameters by integrating over the distribution of latent states. Unfortunately, these filters are either restrictive in their requirements (e.g., the Kalman filter) or computationally costly, non-differentiable, and difficult to tune (e.g., the particle filter).

We tackle these challenges by implementing two complementary methods. Our first method is to apply the HMC sampler to the estimation of DSGE models. HMC is a gradient-based sampling method that traverses the posterior efficiently and works particularly well in high-dimensional cases. While HMC is an attractive alternative to the RWMH algorithm, it requires computing the gradients of the solution to a DSGE model with respect to the parameters, a cumbersome task.

We get around this problem by showing how to differentiate both first- and second-order perturbation solutions of DSGE models with respect to the parameters (although our argument, built around the implicit function theorem, is applicable to any higher-order perturbation solutions). Essentially, we provide a local sensitivity analysis similar to Iskrev (2010), but we extend the results to second-order perturbation solutions. For example, consider a canonical medium-scale Keynesian DSGE such as Fernández-Villaverde and Guerrón-Quintana (2021) with 14 state variables, 24 controls, and 28 parameters. The first-order perturbation solution to this is a matrix of 14×14 values for the evolution of the state, and one with 24×14 for controls. Our procedure provides the gradient of these two matrices with respect to the 28 parameters of the underlying model. Those gradients could be used for all sorts of purposes, such as examining how impulse response functions change with parameters, better calculating the loss function with simulated methods of moments, or—in our main application—computing gradients of the likelihood for Bayesian samplers.[2]

Our second method is to sample both parameter and latent variables simultaneously, instead of sampling the posterior of the model parameter by marginalizing out the latent variables. Hence, we avoid filtering and can both examine estimated latent variables and easily marginalize on samples ex-post by ignoring the estimated latent variables. The idea of sampling parameters and latent states simultaneously has been around for decades (see Kim et al. 1998, for an early incarnation), but its implementation was difficult because, as soon as we have more than a few observations, we are dealing with a high-dimensional inference problem that the RWMH algorithm cannot handle even with

[2]We specify the state-space model in a discrete-time setting, and the linear equation system will be Sylvester equations. In a continuous-time setting that is widely used in HANK models, the equations have a similar form but will be Lyapunov equations. Both Sylvester equations and Lyapunov equations are linear and can be solved with standard libraries like SLICOT and MatrixEquations.jl.

extremely long simulations. In comparison, the scalability of HMC means that sampling the joint likelihood becomes feasible. More generally, we can bring much larger models to the data as long as we can find the required derivatives. Gradient-based approaches such as HMC are limited by difficult geometry and the cost of gradient calculations, but not by the dimensionality of the problem.

We illustrate the two ideas above by estimating three models: a canonical real business cycle (RBC) model, a real small open economy model based on Schmitt-Grohé and Uribe (2003), and the medium-scale New Keynesian model in Fernández-Villaverde and Guerrón-Quintana (2021). In our first experiment, we simulate data from the RBC model solved with a first-order perturbation and use this simulated sample to evaluate the associated likelihood using the Kalman filter. Then, we implement the RWMH and HMC samplers. Even if we only need to sample 3 parameters, the fraction of effective draws per sample for RWMH ranges between 0.4% and 6.2% depending on the parameter. In comparison, the HMC sampler gets $49-56\%$ effective draws, a rate of $10-100$ times more. This experiment shows how much more efficient the HMC sampler is.

In our second experiment, we generate data from the same RBC model but now solved using a second-order perturbation. Then, we estimate the model with the HMC sampler and the joint likelihood function of parameters and latent states. Since there are three parameters and 200 latent states, we sample along 203 dimensions overall. HMC delivers a proportion of effective samples of around 4%. In comparison, the RWMH sampler on the marginal likelihood of just the three parameters evaluated using the particle filter, a dramatically lower-dimensionality problem, delivers an effective sample proportion of only 0.3%. Furthermore, HMC is more robust to the starting point of the Markov chain. After six minutes of execution, the posterior mean of the parameters settles within a narrow range of the final value for all initial conditions when using HMC with the joint likelihood approach. In comparison, and also after six minutes, the RWMH means remain away from the mode for many initial values (in fact, they remain away for even much longer runs). When the number of particles used is insufficient, the chains stop moving, requiring many-particle runs to ensure stability.

In our third and fourth experiments, we repeat the first and second experiments but with simulated data from a version of the model in Schmitt-Grohé and Uribe (2003) augmented with additional shocks, with three latent states, three observables, and seven estimated parameters. In this larger application, HMC again outperforms RWMH in speed and stability, particularly at second order, where it achieves a sampling speed approximately 10-40 times faster per effectively independent sample drawn than the particle filter approach and with better measures of posterior quality. The relative performance advantage of gradient-based samplers compared to RWMH seems to grow as the problem size increases.

In our final experiments, we estimate a New Keynesian model with 22 parameters and 240 latent states. As in our previous experiments, the HMC sampler is much superior to RWMH. For instance, while HMC yields good MCMC mixing, RWMH with the particle filter fails to escape from the neighborhood of the starting point. Taken together, these experiments demonstrate that HMC offers a compelling alternative to other MCMC methods, particularly for larger models.

We implement all our methods through open-source modular building blocks that we hope will allow researchers to develop their own applications. First, the package `DifferentiableStateSpaceModels.jl` provides a symbolic domain-specific language for defining nonlinear DSGE models embedded in `Julia`, and making them easy to embed in the `Julia` auto-differentiation ecosystem through `ChainRules.jl`. Second, the SciML ecosystem (through the package `DifferenceEquations.jl`) provides tools to enable differentiable simulations, Kalman filters, and joint likelihoods for state-space models generated manually or using `DifferentiableStateSpaceModels.jl`. These packages can expand the researchers' imagination for what is a feasible scale in the computation and estimation of DSGE models thanks to our use of differentiable programming. While `Julia` is a natural language given its support for symbolic computing and auto-differentiation, these packages provide a benchmark implementation for porting the methods to other programming languages and probabilistic programming environments.

Differentiable Programming. At the core of our paper, we have the computation of gradients of state-space models and their likelihoods. "Differentiable programming"—a term encompassing classic automatic differentiation (AD)—is a programming paradigm going back to the late 1950s. However, the field has exploded with the recent popularity of machine learning (ML), where it plays a key role. Classic examples of differentiable programming include calculating Jacobians (Griewank and Walther, 2008) and the training of neural networks (Baydin et al., 2017). Differentiable programming/AD ecosystems exist in `Julia`, `Python`, `Stan`, `Matlab`, `Fortran`, and many other programming languages.

The basic idea of differentiable programming is simple and has nothing to do with numerical derivatives. Imagine a procedure that looks at code and substitutes analytic derivatives where appropriate—i.e., if it encounters `sin`, it uses `cos` as the derivative—and applies the chain rule when required. Moreover, if we think about a computer program as a sequence of mathematical functions calling each other, then a method that can analyze a program can also differentiate arbitrary functions by recursively applying the chain rule until it hits primitive gradients.

The previous steps are easy to visualize with standard mathematical functions (e.g., the `log` function). But the surprising result is that code one would not expect to have gradients (e.g., loops, accessing a subset of a vector, solving a linear system, constructors

of parametric structs, Kronecker products) can be formalized with primitive gradients with respect to continuous arguments. This latter feature is what radically changes the scale of problems one can solve.

To see this, we must understand the difference between forward- and reverse-mode AD. If one wants to calculate some function $f : \mathbb{R}^N \to \mathbb{R}$ composed of nested calls to other functions and find its gradients $\nabla f(\cdot) \in \mathbb{R}^N$ at a particular point $x^0 \in \mathbb{R}^N$, we can evaluate the chain rule forward or backward. Intuitively, reverse-mode AD takes the x^0 and applies the functions nested inside of f until it eventually has calculated $f(x^0)$.[3] Then, it perturbs $f(x^0)$ and applies the chain rule backward until it has calculated $\nabla f(x^0)$.[4]

The critical insight here is that if a function maps to a univariate output (e.g., a likelihood function, a moment function, a loss function), the computation of the gradient may require no more computation than (a constant factor times) the evaluation of the original function, regardless of whether it depends on 10 or 10,000 variables. This result is known as the *cheap gradient principle* (Griewank and Walther, 2008). Thus, when $f : \mathbb{R}^N \to \mathbb{R}$, the calculation of $\nabla f(x^0)$ may be only 2 to 10 times slower than the calculation of $f(x^0)$, a factor independent of N. The combination of univariate loss functions with programming language support for reverse-mode AD (and better hardware in GPUs, which are ideal for parallelizing many gradient calculations) has made the deep learning revolution possible.

There are many ways to implement reverse-mode AD, but the most common approach is a package for a programming language that either analyzes a function statically or traces the function during execution. In the case of `Python`, this is the core feature of `Pytorch`, `Tensorflow`, and `JAX`, and in `Julia`, there are many alternatives such as `Zygote.jl` (see Innes et al., 2019) and Enzyme. These packages manage the application of the chain rule, compiling both the function calculation and its gradients, and accessing libraries of primitive gradients, which are themselves black boxes to the AD package.[5]

With differentiable programming, finding the right level of abstraction for providing gradients is crucial. While a small library of gradient primitives might suffice to build up a differentiable program, when the intermediate function calls involve complex algo-

[3] In the ML community, they often refer to reverse-mode AD as "backpropagation," whereas, in differential equations and control theory, it is referred to as "adjoint sensitivity."

[4] Forward-mode AD applies the chain rule forward starting from x^0 and calculates $f(x^0)$ and $\nabla f(x^0)$ at the end. Forward-mode AD is equivalent to using "numerical derivatives" in its computational order, even if it is always more accurate. This is attractive for calculating: $f : \mathbb{R}^N \to \mathbb{R}$ with a small N such that the overhead of reverse-mode AD dominates its advantages; $f : \mathbb{R} \to \mathbb{R}^N$ (where reverse-mode AD is at its worst); sparse Jacobians of $f : \mathbb{R}^N \to \mathbb{R}^M$; and higher-order derivatives nested with reverse-mode AD.

[5] See White et al. (2021) for a description of `ChainRules.jl`, which provides a package-agnostic way to register primitive gradients in `Julia`. Though less flexible, custom gradients can be implemented in Pytorch/Autograd. `JAX` custom derivative rules and primitives are more restrictive (e.g., loops are not easily possible within the implementation), but integrate well into the ecosystem and are easily composable for higher-order AD. See ssm-jax for a library for state-space models (but not perturbation solutions) in JAX.

rithms (such as when we solve DSGE models given a set of parameters), custom gradient rules ensure efficient and correct gradient computations (see Appendix E for details).

In our implementation, we determine the appropriate level by providing gradients and implementing two orthogonal and composable components for gradient-based algorithms like HMC—the perturbation solution of the DSGE and the state-space likelihood.

For the first component, standard DSGE solvers (e.g., Klein 2000; Schmitt-Grohé and Uribe 2004) involve steps like a generalized Schur decomposition, which are, in principle, differentiable but require custom gradients in practice. Other steps in those algorithms, such as the reordering of the eigenvalues, may not even be differentiable in all cases. Instead, we derive an implicit gradient formula for the whole calculation of the DSGE solution, bypassing these issues since we no longer need to differentiate the internal algorithm. This provides a component that both solves the DSGE models and provides gradients with respect to the parameters of interest.[6]

For the second component, state-space models provide a stochastic difference equation that we use to simulate solutions and accumulate likelihoods—which must be differentiated with respect to the state-space model primitives and the underlying "shocks" in cases where we form the joint likelihood. We build on an active area of research known as "scientific machine learning" (Rackauckas, 2022) that expands AD systems to incorporate efficient and scalable primitives for scientific computing applications such as nonlinear system solvers and differential equations. Within that framework, we provide differentiable simulations and filters that can operate on discrete-time state-space models, supplied by our differentiable DSGE solver or third-party code.

Literature Review. The paper closet to ours is Farkas and Tatar (2020), which proposes estimating linear-Gaussian DSGE models with HMC methods on the marginal likelihood of parameters, using Smets and Wouters (2007) as an example. Our work nests such an application as a special case. More importantly, Farkas and Tatar (2020) use a non-customizable AD and so the differentiation has to go step-by-step, which precludes them from using Schur decomposition methods, the dominant approach to solving linear DSGE models due to stability and speed. Instead, we offer a general framework to implement HMC in perturbation solutions of DSGE models by passing the required derivatives to the downstream sampler with customized AD.

The use of the joint likelihood for sampling from state-space models has been known for decades but not widely adopted due to the poor scaling with dimension of more traditional MCMC samplers (Särkkä, 2013). When it is combined with the use of HMC, performance again becomes competitive and the method has been applied to ARCH and

[6]Since the implementation of gradients for the DSGE solution requires higher-order derivatives of the model itself, we use symbolic-numeric methods via `Symbolics.jl` (see Gowda et al. 2022).

ARMA models (Stan Development Team, 2022), stochastic volatility models (Hoffman and Gelman, 2014), and diffusion processes (Graham et al., 2022).

The key bottleneck to application to DSGE models is the fact that HMC requires gradients of the likelihood and model solution. Derivative formulas have been derived for some of the elements necessary for our applications, including first-order perturbations of DSGE models (Iskrev, 2010) and the marginal likelihood of linear-Gaussian state-space models (Watson, 1989), but our extension to the higher-order case is novel. Moreover, the rules are implemented in a form applicable for use within a (reverse-mode) AD system, which enables improvements in speed and modularity. Previous applications of other forms of AD to differentiate the likelihood of macroeconomic models include Bastani and Guerrieri (2008); applications of AD to optimization routines and solution algorithms include Maliar et al. (2021) and Cao et al. (2020). By implementing our methods as components in differentiable and probabilistic programming languages, our intention is that they may be combined with other components to enable further applications.

Outline. The rest of this paper is organized as follows. Section 2 presents the state-space representation of DSGE models and introduces the joint likelihood approach associated with this mathematical representation. Section 3 describes HMC. Section 4 shows the gradients of solutions to DSGE models. Section 5 briefly discusses our implementation. Section 6 illustrates the estimation results. Section 7 concludes. An extensive set of appendixes collect additional details and numerical experiments.

2 Likelihood Function and Evaluation

This section fixes the notation for the state-space representation of a dynamic equilibrium model and defines the likelihood terms we will use throughout the paper. While DSGE models fit easily into this framework, we will first present an AR(1) example that illustrates the difference between the marginal and joint likelihoods and their sampling processes. This exercise clarifies ideas in a transparent case.

2.1 The State-Space Model

Given a dynamic equilibrium model, let x_t be the vector of state variables that determine its behavior. These states are buffeted by shocks ϵ_t. Without loss of generality, we assume these shocks are i.i.d. Let θ be the vector of parameters of the model. In the case of a DSGE model, θ determines the preferences, technology, information sets, and government policy rules. We assume that θ is time-invariant, but, with extra notation, we can allow

correlated or time-varying parameters and shocks. See Fernández-Villaverde and Rubio-Ramírez (2007). The initial distribution of the state variables is given by $p(x_0|\theta)$.

The optimization behavior of the agents is summarized by a policy function $g(\cdot)$ that links state variables with the control variables y_t:

$$y_t = g\left(x_t; \theta\right). \tag{1}$$

Given this policy function, other equilibrium conditions (such as resource constraints and exogenous laws of motion), and $p(x_0|\theta)$, we can build a state-space representation of the model. The first component of this representation is a transition equation $h(\cdot)$ that characterizes the law of motion of the states:

$$x_{t+1} = h\left(x_t; \theta\right) + \eta\epsilon_{t+1}. \tag{2}$$

This transition equation gives us the conditional density $p\left(x_{t+1}|x_t, \theta\right)$ induced by the distribution of ϵ_{t+1} and given x_t. The assumption that ϵ_{t+1} enters linearly is without loss of generality. See Andreasen et al. (2018) for an explanation.

The second component of the state-space representation is a measurement equation $q(\cdot)$ that links states and controls with the observables z_t:

$$z_t = q\left(x_t, v_t; \theta\right), \tag{3}$$

where v_t is either a measurement error or a shock that hits observables but not the states of the model. Thus, the measurement equation defines a conditional density $p\left(z_t|x_t, \theta\right)$ given by the distribution of v_t. One or more states may be part of the observables z_t. In that case, $q(\cdot)$ is the identity function along the relevant dimensions and the conditional density is a Dirac.

Lastly, let $z^T = \{z_t\}_{t=1,\ldots,T}$ be the observation sequence for T periods, $x^T = \{x_t\}_{t=0,\ldots,T}$ the state sequence, and $p\left(z^T|\theta\right)$ the likelihood function of the model.

2.2 Marginal vs. Joint Likelihood

We now review the concepts of the marginal and joint likelihood function associated with the previous state-space representation.

The Marginal Likelihood. If we specify a prior distribution $p(\theta)$ for the parameters of the model, Bayes' theorem gives us:

$$\underbrace{\ln p\left(\theta|z^T\right)}_{\text{log-posterior}} = \ln p\left(\theta, z^T\right) + C \tag{4}$$

$$= \underbrace{\ln p(\theta)}_{\text{log-prior}} + \underbrace{\ln p\left(z^T|\theta\right)}_{\text{log-likelihood}} + C \tag{5}$$

$$= \ln p(\theta) + \sum_{t=1}^{T} \ln p\left(z_t|z^{t-1}, \theta\right) + C. \tag{6}$$

Equation (5) tells us that the log-posterior density is (up to a constant C) the sum of the log prior and the conditional density of the data. As shown in equation (6), the second term of equation (5) is the sum of the conditional densities for z_t across all T periods.

We can recursively construct the sequence of $\left\{p\left(z_t|z^{t-1}, \theta\right)\right\}_{t=1,\ldots,T}$ by filtering:

1. We start with the initial density at $t = 1$:

$$p\left(x_{t-1}|z^{t-1}, \theta\right) = p\left(x_0|\theta\right). \tag{7}$$

2. We use the Chapman-Kolmogorov equation to combine $p\left(x_{t-1}|z^{t-1}, \theta\right)$ with the conditional distribution $p\left(x_t|x_{t-1}, \theta\right)$ and integrate over the states to get:

$$p\left(x_t|z^{t-1}, \theta\right) = \int p\left(x_t|x_{t-1}, \theta\right) p\left(x_{t-1}|z^{t-1}, \theta\right) \mathrm{d}x_{t-1}. \tag{8}$$

3. We take the resulting $p\left(x_t|z^{t-1}, \theta\right)$, the measurement z_t, and $p\left(z_t|x_t, \theta\right)$ from the measurement equation, and we plug them into Bayes' theorem to get:

$$p\left(x_t|z^t, \theta\right) = \frac{p\left(z_t|x_t, \theta\right) p\left(x_t|z^{t-1}, \theta\right)}{p\left(z_t|z^{t-1}, \theta\right)}. \tag{9}$$

4. We find the marginal likelihood:

$$p\left(z_t|z^{t-1}, \theta\right) = \int p\left(z_t|x_t, \theta\right) p\left(x_t|z^{t-1}, \theta\right) \mathrm{d}x_t. \tag{10}$$

5. We loop over steps 2.-4. for all T periods of observations.

We call $p\left(z_t|z^{t-1}, \theta\right)$ the "marginal likelihood" because we have integrated out the

states x_t.[7]

While equations (8) and (9) are conceptually simple, they can be daunting to implement in practice. In the case where $h(\cdot)$ and $q(\cdot)$ are linear in x and both ϵ and v follow a Gaussian distribution, equations (8) and (9) can be evaluated exactly with the Kalman filter. In nonlinear or non-Gaussian cases, a closed form for the marginal likelihood is rarely available, and we must use numerical approximations such as particle filters.

Joint Likelihood. We now focus on the joint likelihood approach, which directly applies Bayes' theorem to the likelihood of the data z^T jointly in the parameters θ and the states $x^T = \{x_t\}_{t=0,\dots,T}$, i.e., $p\left(z^T|x^T,\theta\right)$, in order to compute the joint posterior $p\left(\theta,x^T|z^T\right)$ without first marginalizing out the states:

$$p\left(\theta,x^T|z^T\right) = \frac{p\left(z^T|\theta,x^T\right)p\left(x^T|\theta\right)p\left(\theta\right)}{\int\int p\left(x^T|\theta\right)p\left(\theta\right)\mathrm{d}x^T\mathrm{d}\theta}.$$

Taking logs and ignoring constant terms, we can decompose the posterior as:

$$\underbrace{\ln p\left(\theta,x^T|z^T\right)}_{\text{log-posterior}} = \underbrace{\ln p\left(\theta,x^T\right)}_{\text{log-prior}} + \underbrace{\ln p\left(z^T|x^T,\theta\right)}_{\text{log-likelihood}} + C \tag{11}$$

$$= \ln p\left(\theta\right) + \ln\left(x^T|\theta\right) + \sum_{t=1}^{T}\ln p\left(z_t|x^t,\theta\right) + C \tag{12}$$

$$= \ln p\left(\theta\right) + \sum_{t=1}^{T}\ln p\left(z_t|x_t,\theta\right) + \sum_{t=1}^{T}\ln p\left(x_t|x_{t-1},\theta\right) + \ln p\left(x_0|\theta\right) + C \tag{13}$$

$$= \ln p\left(\theta\right) + \sum_{t=1}^{T}\ln p\left(z_t|\epsilon^t,x_0,\theta\right) + \sum_{t=1}^{T}\ln p\left(\epsilon_t|\theta\right) + \ln p\left(x_0|\theta\right) + C. \tag{14}$$

In this case, we can evaluate the log-likelihood sequentially by following equation (12). Because of the Markov structure of the state-space model, equation (13) shows that x_t only depends on x_{t-1} given θ.

Equations (11)–(13) compute the log-posterior taking not only the data but also the history of states as given, while the law of motion of the states is pinned down by the initial state x_0, the sequence of shocks $\epsilon^T = \{\epsilon_t\}_{t=1,\dots,T}$, and the model parameters θ. Hence, the second term in (14) comes from $q(\cdot)$, and the third term in (13) comes from

[7] Sometimes, the literature refers to "marginal likelihood" when the parameters have also been integrated out over the whole sample, i.e., $p(z^T) = \int p(z^T|\theta)p(\theta)\mathrm{d}\theta$.

$h(\cdot)$. To bridge equations (13) and (14), we can rewrite these two terms in (13) as:

$$\sum_{t=1}^{T} \ln p\left(z_t|x_t,\theta\right) = \sum_{t=1}^{T} \ln p\left(z_t|\epsilon^t, x_0, \theta\right) \tag{15}$$

$$\sum_{t=1}^{T} \ln p\left(x_t|x_{t-1},\theta\right) = \sum_{t=1}^{T} \ln p\left(\epsilon_t|\theta\right). \tag{16}$$

Therefore, the joint likelihood equation can be explicitly written down given an exogenous series of shocks ϵ_t. This joint likelihood computation is filter-free, as we do not have to infer the latent state series x^T or their posterior distributions, but rather retrieve a series of x^T from the shock series.

Consequently, instead of sampling just θ to compute a marginal likelihood $\ln p\left(\theta|z^T\right)$, we will sample θ as well as ϵ^T and x_0 to compute the joint likelihood $\ln p\left(\theta, x^T|z^T\right)$, which is equivalent to $\ln p\left(\theta, \epsilon^T, x_0|z^T\right)$. The marginal likelihood can be obtained from the joint likelihood by integrating over ϵ^T and x_0:

$$p\left(\theta|z^T\right) = \int p\left(\theta, x^T|z^T\right) \mathrm{d}x^T = \int\int p\left(\theta, \epsilon^T, x_0|z^T\right) \mathrm{d}\epsilon^T \mathrm{d}x_0. \tag{17}$$

Given samples from the joint posterior over θ, ϵ^T, and x_0, samples from the marginal posterior over θ can be obtained by dropping the ϵ^T and x_0 components of each sample.

2.3 An AR(1) Example with Measurement Noise

We employ a simple example to show the difference between the marginal and joint likelihood approaches. Assume that our dynamic equilibrium model takes the form of an AR(1) process with a measurement noise:

$$x_t = \rho x_{t-1} + \epsilon_t$$
$$z_t = x_t + v_t.$$

The shock process $\{\epsilon_t\}$ is i.i.d and follows a standard Gaussian distribution, and the measurement noise $\{v_t\}$ follows an i.i.d. Gaussian distribution with zero mean and variance Ω. We assume that the initial x_0 is drawn from the invariant distribution, which, when $|\rho| < 1$, exists and is given by $\mathcal{N}\left(0, \frac{1}{1-\rho^2}\right)$. We denote the probability density function of a standard Gaussian distribution as $\varphi(\cdot)$. We are interested in estimating ρ, the persistence parameter of the states, given data z^T.

The Marginal Likelihood. The steps to evaluate the marginal likelihood given z^T and ρ are:

1. Evaluate the log-prior at ρ and set $L = \ln p(\rho)$.

2. Initialize the distribution of x_0 as $\mathcal{N}\left(0, \frac{1}{1-\rho^2}\right)$. Using standard notation for conditioning, we write the mean of the distribution as $x_{0|0} = 0$ and its variance as $P_{0|0} = \frac{1}{1-\rho^2}$.

3. Loop over $t = 1, \ldots, T$. For each t, we apply the Kalman filter to infer the posterior distribution of x_t, and accrue the likelihood L:

$$\begin{aligned} x_{t|t-1} &= \rho x_{t-1|t-1} \\ P_{t|t-1} &= \rho^2 P_{t-1|t-1} + 1 \\ x_{t|t} &= x_{t|t-1} + \frac{P_{t|t-1}}{P_{t|t-1} + \Omega}\left(z_t - x_{t|t-1}\right) \\ P_{t|t} &= P_{t|t-1} - \frac{P_{t|t-1}^2}{P_{t|t-1} + \Omega} \\ L &= L + \ln \varphi\left(\frac{z_t - x_{t|t-1}}{\sqrt{P_{t|t-1} + \Omega}}\right). \end{aligned}$$

The Joint Likelihood Approach. The steps to evaluate the joint likelihood given z^T, ρ, initial condition x_0, and vector of T elements $\{\epsilon_t\}$ are:

1. Set L equal to the sum of the log-prior $\ln p(\rho)$ at ρ and the log-prior of the latent states $\sum_{t=1}^{T} \ln \varphi\left(\epsilon_t\right)$ at $\{\epsilon_t\}$.

2. Add the log-density of x_0 in the initial distribution $L = L + \ln \varphi(x_0 \sqrt{1-\rho^2})$.

3. Loop over $t = 1, \ldots, T$. For each t, compute x_t with the law of motion, and accrue the likelihood L:

$$\begin{aligned} x_t &= \rho x_{t-1} + \epsilon_t \\ L &= L + \ln \varphi\left(\frac{z_t - x_t}{\sqrt{\Omega}}\right). \end{aligned}$$

While in the end, the posterior of ρ should be the same, there are significant differences between both approaches. The marginal likelihood depends on a filter to infer the distribution of the latent states over time while it only samples the parameters that govern the model. The joint likelihood approach is filter-free because the law of motion of the state variables is deterministic given some shocks, but at a cost that it has to draw both the parameters and the shocks. Hence, by using the joint likelihood, we are sampling from a very high-dimensional space consisting of both the parameters and the shocks. Thus,

if filtering is costly, as when we deal with nonlinear or non-Gaussian DSGE models, the joint likelihood might be an attractive alternative.

Given the high dimensionality of the sampling problem when using the joint likelihood, a RWMH sampler will not be efficient enough to deliver accurate answers in reasonable amounts of time. Instead, we apply HMC—a scalable and efficient sampler that copes with high-dimensional spaces—that we now describe.

3 Hamiltonian Monte Carlo

HMC is a sampler designed to avoid some of the problems encountered by standard MCMC samplers such as RWMH.[8] While RWMH is straightforward to code, it also suffers from a basic problem: it spends much of the time outside of the typical set, i.e., the part of the parameter space containing the relevant information to compute the expectations we care about in Bayesian analysis.

To understand this statement, we must define the typical set (see Betancourt, 2018, and Fernández-Villaverde and Guerrón-Quintana, 2021). For $\epsilon > 0$ and and I, the typical set A_ϵ^I with respect to the target posterior $p\left(\theta|z^T\right)$ is:

$$A_\epsilon^I \equiv \left\{(\theta_0, \theta_1, \ldots, \theta_I) : \left| -\frac{1}{I+1}\sum_{i=0}^{I} \ln p\left(\theta_i|z^T\right) - b\left(\theta\right)\right| \leqslant \epsilon \right\},$$

where $b\left(\theta\right) = -\int p\left(\theta_i|z^T\right) \ln p\left(\theta_i|z^T\right) \mathrm{d}\theta$ is the differential entropy of the parameters with respect to their posterior density. By a weak law of large numbers, we have that $\Pr(A_\epsilon^I) > 1 - \epsilon$ if I is sufficiently large. That is to say, A_ϵ^I includes "most" sequences of θ_i's that are distributed according to the posterior $p\left(\theta_i|z^T\right)$.

Two properties of the typical set are crucial. First, the typical set is not the region where the posterior density is the highest. Second, the typical set narrows as the dimensionality of θ grows. These two properties explain why RWMH cannot efficiently sample in high-dimensional spaces. The RWMH method wastes many iterations because the proposal density is blind regarding the typical set of the posterior. Therefore, most draws will be either far away from the typical set, or highly auto-correlated, and the typical set will not be traversed efficiently. Furthermore, initiating the sampling process from the mode will barely help, as the mode is usually not in the typical set.

HMC improves sampling efficiency with respect to RWMH by exploiting information from the posterior's gradient. To avoid pushing the samples to the mode of the posterior, HMC augments the gradient information with an extra momentum force, in an analogy

[8] See Betancourt (2018) for a conceptual discussion of HMC methods and Appendix G for a discussion of how and why HMC improves sampling performance in our setting.

to Hamiltonian mechanics in physics. In that way, the resulting Markov chain spends much more time in the typical set.

More concretely, HMC augments the vector of parameters θ with an auxiliary momentum vector $\mathbf{p}$ of the same dimensions and that follows a Gaussian distribution $\mathcal{N}(0, M)$. Thus, the Hamiltonian associated with the posterior of θ is:

$$H(\theta, \mathbf{p}) = -\ln p(\theta) + \frac{1}{2}\mathbf{p}^\top M^{-1}\mathbf{p}, \tag{18}$$

where $-\ln p(\theta)$, the (minus) log-posterior, is the analog of a "potential energy" function and $\frac{1}{2}\mathbf{p}^\top M^{-1}\mathbf{p}$ acts as a "kinetic energy" term.

Recall that the total energy of the Hamiltonian is preserved when $\{\theta, \mathbf{p}\}$ evolve over time following Hamilton's equations of motion:

$$\frac{d\theta}{dt} = \frac{\partial H}{\partial \mathbf{p}} \tag{19}$$

$$\frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \theta}. \tag{20}$$

This result suggests a simple sampling scheme: we generate a random momentum $\mathbf{p} \sim \mathcal{N}(0, M)$, and then we move $\{\theta, \mathbf{p}\}$ using Hamilton's equations for some length of time to obtain the next draw (and, hence, requiring gradient information on $H(\cdot)$). This draw has a stationary marginal density $p(\theta)$, which provides a theoretical justification for HMC.

The physics analogy here is straightforward. Think about a particle that moves on the manifold characterized by the log posterior of θ. Every time we get a new sample from the current position, we randomly kick the particle in an arbitrary direction. The particle moves following the Hamiltonian dynamics, and we stop the particle after a fixed time and record the new θ.

We solve the ODE system given by equations (19) and (20) as follows. First, we draw $\mathbf{p}$ from the Gaussian distribution specified above. Second, we run the Verlet integrator, a numerical ODE solver that maintains the energy-preserving property of the original system, with step ϵ for each of the L iterations, resulting in an integration time of $L\epsilon$. Both ϵ and L are tuning parameters, with ϵ controlling the step size of Hamiltonian approximation and L determining the number of steps in each iteration. To account for the error induced by using a numerical integrator, at the end of each iteration, the move is accepted or rejected according to a Metropolis step. Since the exact integration already preserves the density, one can usually accept a much higher proportion of proposals than when using RWMH.

Since the introduction of HMC, many variants and improvements have been proposed, particularly in terms of automated methods for selecting and adapting hyper-

parameters. The most popular of these is the No-U-Turn sampler (NUTS) proposed by Hoffman and Gelman (2014), which endogenously picks ϵ and L with sample adaptations. NUTS and other advanced samplers are provided by probabilistic programming languages, enabling the user to define (auto-differentiable) components of the likelihood function and priors. In our experiments, we use the generalized version of NUTS in Betancourt (2018) and implemented in `Turing.jl`. This package is designed to be equivalent to the default implementation in `Stan` and qualitatively similar to implementations in other probabilistic programming languages including `NumPyro` and `PyMC`.

4 Gradients to DSGE Solutions

This section provides the derivations of the derivatives of DSGE first- and second-order solutions with respect to the model parameters. As the downstream HMC sampler requires gradient information, we compute these values by solving equations derived from the implicit function theorem. While the first-order gradient result is documented in the literature (Iskrev, 2010), the second-order result we provide here is a novel extension and further generalizes to even higher-order perturbations.

4.1 Notation

We use the "canonical form" following Schmitt-Grohé and Uribe (2004) to characterize the equilibrium conditions of a variety of DSGE models:

$$\mathbb{E}_t \mathcal{H}\left(y', y, x', x; \theta\right) = 0, \tag{21}$$

following the notation of Section 2. Since the model is Markov, we omit the time subscript and denote x' and y' as states and control variables in the next period.

The solution to the model is:

$$y = g\left(x; \theta\right) \tag{22}$$

$$x' = h\left(x; \theta\right) + \eta\epsilon', \tag{23}$$

where $h(\cdot)$ is the law of motion of the states, and $g(\cdot)$ is the policy function.

The deterministic steady state (DSS) of the system is given by vectors $\bar{x}$ and $\bar{y}$ such that:

$$\mathcal{H}\left(\bar{y}, \bar{y}, \bar{x}, \bar{x}; \theta\right) = 0 \tag{24}$$

and we approximate the solutions to g and h by perturbing around $\bar{x}$.

Let $\hat{x} = x - \bar{x}$ and $\hat{y} = y - \bar{y}$ be the deviations from the DSS. Thus, we can write the

first-order solution to the DSGE model as:

$$\hat{y} = g_x \hat{x} \tag{25}$$

$$\hat{x}' = h_x \hat{x} + \eta \epsilon', \tag{26}$$

where subindices denote partial derivatives. Similarly, we can write the second-order solution as:

$$[\hat{y}]^i = [g_x \hat{x}]^i + \frac{1}{2} \hat{x}^\top [g_{xx}]^i \hat{x} + \frac{1}{2} [g_{\sigma\sigma}]^i \tag{27}$$

$$[\hat{x}']^j = [h_x \hat{x}]^j + \frac{1}{2} \hat{x}^\top [h_{xx}]^j \hat{x} + \frac{1}{2} [h_{\sigma\sigma}]^j + [\eta \epsilon']^j. \tag{28}$$

We use conventional tensor notation here since g_{xx} and h_{xx} are three dimensional.

We find these solutions with the methods proposed by Klein (2000) and Schmitt-Grohé and Uribe (2004). See Appendix A for details.

4.2 Solution Gradients

We find the derivatives of the first- and second-order solutions with respect to the parameters θ mostly via the implicit function theorem. Appendix B provides the details.

Steady State. For any parameter θ, we totally differentiate (24), which yields:

$$\mathcal{H}_x \frac{\partial \bar{x}}{\partial \theta} + \mathcal{H}_y \frac{\partial \bar{y}}{\partial \theta} + \mathcal{H}_{x'} \frac{\partial \bar{x}}{\partial \theta} + \mathcal{H}_{y'} \frac{\partial \bar{y}}{\partial \theta} + \mathcal{H}_\theta = 0,$$

where all the derivatives of $\mathcal{H}$ are evaluated at the DSS. When we treat $\frac{\partial \bar{x}}{\partial \theta}$ and $\frac{\partial \bar{y}}{\partial \theta}$ as unknowns, this is a linear equation system.

First-Order Solutions. We are interested in the derivatives of the first-order solutions, g_x and h_x, with respect to the parameters θ. We denote these derivatives as $\frac{\partial g_x}{\partial \theta}$ and $\frac{\partial h_x}{\partial \theta}$. When evaluated at the DSS, g_x and h_x satisfy:

$$\mathcal{H}_x + \mathcal{H}_y g_x + \mathcal{H}_{x'} h_x + \mathcal{H}_{y'} g_x h_x = 0. \tag{29}$$

If we differentiate (29) with respect to θ:

$$\begin{bmatrix} \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta} \\ \frac{\mathrm{d}\mathcal{H}_{y}}{\mathrm{d}\theta} \\ \frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta} \\ \frac{\mathrm{d}\mathcal{H}_{x}}{\mathrm{d}\theta} \end{bmatrix}^{\top} \begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix} + \begin{bmatrix} \mathcal{H}_y & \mathcal{H}_{x'} + \mathcal{H}_{y'} g_x \end{bmatrix} \begin{bmatrix} \frac{\partial g_x}{\partial \theta} \\ \frac{\partial h_x}{\partial \theta} \end{bmatrix} + \begin{bmatrix} \mathcal{H}_{y'} & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial g_x}{\partial \theta} \\ \frac{\partial h_x}{\partial \theta} \end{bmatrix} h_x = 0,$$

which is a Sylvester equation system in $\frac{\partial g_x}{\partial \theta}$ and $\frac{\partial h_x}{\partial \theta}$.[9]

Second-Order Solutions. The second-order solutions to the DSGE include four additional objects besides the first-order solutions: g_{xx}, h_{xx}, $g_{\sigma\sigma}$, and $h_{\sigma\sigma}$. Again, we denote these derivatives as $\frac{\partial g_{xx}}{\partial \theta}$,$\frac{\partial h_{xx}}{\partial \theta}$,$\frac{\partial g_{\sigma\sigma}}{\partial \theta}$,$\frac{\partial h_{\sigma\sigma}}{\partial \theta}$, respectively.

First, we derive the solutions to $\frac{\partial g_{xx}}{\partial \theta}$ and $\frac{\partial h_{xx}}{\partial \theta}$. By differentiating (29) with respect to x, and flattening the second and third dimensions of g_{xx} and h_{xx}, we get:

$$\begin{bmatrix} \mathcal{H}_{y'} & 0 \end{bmatrix} \begin{bmatrix} g_{xx} \\ h_{xx} \end{bmatrix} h_x \otimes h_x + \begin{bmatrix} \mathcal{H}_y & \mathcal{H}_{y'} g_x + \mathcal{H}_{x'} \end{bmatrix} \begin{bmatrix} g_{xx} \\ h_{xx} \end{bmatrix} + C = 0. \tag{30}$$

Equation (30) is a Sylvester equation on $[g_{xx}, h_{xx}]'$. If we further differentiate it with respect to θ, we get another Sylvester equation in $\frac{\partial g_{xx}}{\partial \theta}$ and $\frac{\partial h_{xx}}{\partial \theta}$.

The solutions to $\frac{\partial g_{\sigma\sigma}}{\partial \theta}$ and $\frac{\partial h_{\sigma\sigma}}{\partial \theta}$ are simpler. Solving (31) below yields $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$:

$$\begin{pmatrix} \mathcal{H}_{y'} + \mathcal{H}_y & \mathcal{H}_{y'} g_x + \mathcal{H}_{x'} \end{pmatrix} \begin{pmatrix} g_{\sigma\sigma} \\ h_{\sigma\sigma} \end{pmatrix} + B = 0, \tag{31}$$

and differentiating (31) with respect to the parameters θ yields a linear equation system for $\frac{\partial g_{\sigma\sigma}}{\partial \theta}$ and $\frac{\partial h_{\sigma\sigma}}{\partial \theta}$. Matrices B and C are constants given the second-order solution to the DSGE model. Appendix B provides a detailed construction of these two matrices.

5 Implementation

We briefly introduce our implementation here. Our software consists of two generic parts, built to work with various economic models, and a third component that glues these together with problem-specific details required for Bayesian estimation. The first is a library that handles first- and second-order solutions to state-space models, gradients, and sensitivity analyses. The library, `DifferentiableStateSpaceModels.jl`, is an open-

[9] A matrix Sylvester equation has the form $AXB + CXD + E = 0$, which is a (bi-)linear equation of the real matrix X. Since Sylvester equations are widely used in control theory, fast solvers are available.

source `Julia` package that provides a domain-specific language to represent macro state-space models, providing the underlying derivatives for the perturbation and its derivatives symbolically. The second component, `DifferenceEquations.jl`, provides differentiable simulations and likelihoods given state-space models. These can be used for many simulation and estimation algorithms, and the state-space inputs do not necessarily need to come from perturbation solutions to DSGEs—although it is especially convenient to use with `DifferentiableStateSpaceModels.jl`. Finally, we provide examples using `Turing.jl` that glues these composable functions together with priors on parameters to use with HMC samplers, which involves a small amount of problem-specific code.

The package `DifferentiableStateSpaceModels.jl` is narrowly focused on calculating perturbation solutions and their derivatives. First, we develop a domain-specific language to specify state-space model variables and equations using `Symbolics.jl` (see Gowda et al., 2022). Users can also specify optional equations for steady-state evaluation or initial conditions. Second, we generate the symbolic derivatives for the model to use in the perturbation solution algorithms.[10] The `Symbolics.jl` ecosystem allows us to define a `Dynare`-like domain-specific language, and then the necessary functions are differentiated symbolically and exported as regular `Julia` functions. At that point, the connection to the previous symbolic language is severed, and the functions we generate behave as—and perform identically to— handcrafted functions and gradients in `Julia`. Second, we implement the first- and second-order solutions to the approximated state-space model using standard algorithms. Finally, we find the gradient of those solutions with respect to the parameters. Tying this back to our earlier discussion of differentiable programming, if we think of a perturbation solution as a function that takes a model and a set of parameter values and generates a state-space representation, then we are providing a custom gradient for that function so that any downstream usage of the state-space (e.g., its matrices, vectors, and tensors) is differentiable with respect to the model parameters.

The generated state-space model for a given set of parameters is typically fed into `DifferenceEquations.jl`, which provides a variety of standard features such as simulations and likelihood calculations—both using a marginalized approach with a Kalman filter and a joint approach fixing the noise. Crucially, these functions themselves are given high-performance custom derivatives. Again, if we think of a log-likelihood as taking a given state-space representation, observables, and possibly noise, then we are providing a function that calculates the log-likelihood and finds its gradients with respect to the state-space matrices/tensors and possibly the high-dimensional latent variables.

[10] To get those derivatives, we can either use AD or symbolic derivatives. We chose the latter. While AD might be more efficient in some cases (e.g., heterogeneous agent models) where we do not want to flatten the computational graph, it is harder to implement as it requires a mixed reverse and forward approach given our need to have both derivatives of the model and cross-derivatives with respect to the parameters of interest.

Finally, the most easily replaceable component is the problem-specific implementation of the likelihood and priors for the HMC sampler. This is used within the `Turing.jl` probabilistic programming language embedded within Julia. Users only need to define the prior distributions of the parameters to draw, manipulate the observables as required, call `DifferentiableStateSpaceModels.jl` to calculate the perturbation given sampled parameters, and then use the solution to the DSGE downstream in `DifferenceEquations.jl` to calculate the log-likelihood. Gradients are provided to the sampler by `Turing` using the underlying AD package (`Zygote.jl` in our implementation), given the custom rules defined in the core packages. The full examples using the above packages to replicate the experiments in the paper are provided in HMCExamples.jl.

6 Main Results

Next, we present our main results. First, we estimate a canonical real business cycle model and compare the efficiency and robustness of HMC against RWMH. Second, we estimate a real small open economy model based on Schmitt-Grohé and Uribe (2003), where we show even larger efficiency gains. Third, we estimate a medium-scale New Keynesian DSGE model, namely Fernández-Villaverde and Guerrón-Quintana (2021), close to the models used for real policy analysis.

6.1 Estimating the Real Business Cycle Model

A canonical real business cycle model can be characterized by equation system (32), where c, k, y, z represent consumption, capital, output, and the TFP level respectively. The system includes four equations: an intertemporal Euler equation, a resource constraint, a production function, and the law of motion for TFP:

$$\begin{aligned} \frac{1}{c_t} - \beta \frac{\alpha e^{z_{t+1}} k_{t+1}^{\alpha-1} + (1-\delta)}{c_{t+1}} &= 0 \\ c_t + k_{t+1} - (1-\delta) k_t - y_t &= 0 \\ y_t - e^{z_t} k_t^{\alpha} &= 0 \\ z_{t+1} - \rho z_t - \sigma \epsilon_{t+1} &= 0, \end{aligned} \tag{32}$$

where $\{\epsilon_t\}$ follows an i.i.d standard Gaussian distribution.

We set as pseudotrue parameter values $\alpha = 0.3$, $\beta = 0.998$, $\rho = 0.9$, $\delta = 0.025$, and $\sigma = 0.1$, all standard values in quarterly calibrated models. Then, we simulate two sets of 200 artificial observations for consumption c_t and investment $i_t = k_{t+1} - (1-\delta) k_t$, one set coming from the first-order solution of the model and the other from the second-

order solution. To avoid stochastic singularity, we introduce a small measurement error (Gaussian with variance of $1e-5$ for each variable).

We estimate the model using the simulated data and assuming the researcher knows $\delta = 0.025$, $\sigma = 0.1$, and the variance of the measurement error, but must learn about α, β, and ρ. As β is close to 1, we sample $\beta_{draw} = 100(1/\beta - 1)$ instead. As priors, we use a truncated normal for α with mean 0.3, standard deviation 0.025, 0.2 lower truncation bound, and 0.5 upper truncation bound; for β_{draw} we use a Gamma with mean 0.25 and standard deviation 0.1; and for ρ we use a Beta with mean 0.5 and standard deviation 0.2.

Table 1: RWMH with Marginal Likelihood, RBC Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.3	0.2996	0.0078	821.2	1.0009	0.8295	2.6029	315
β_{draw}	0.2	0.204	0.0529	418.99	1.0009	0.4232	1.328	315
ρ	0.9	0.8981	0.0074	6188.4	1.0	6.2509	19.615	315

Notes: We draw 110,000 samples in total and discard the first 11000 samples. The sampling time is measured in seconds and excludes model file generation and compilation.

Table 1 reports the results of estimating the first-order RBC model evaluating the marginal likelihood with a Kalman filter and sampling from the posterior with a RWMH. We run this exercise with `Dynare`. We picked `Dynare` as a benchmark because it is a) the most popular software for the solution and estimation of DSGE models and b) a high-quality, state-of-the-art alternative based on highly optimized `C++` code, incorporating several algorithmic advances into the solution and estimation algorithms, and so represents a stronger benchmark relative to existing Julia implementations.

Table 2: NUTS with Marginal Likelihood, RBC Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.3	0.2994	0.0076	3214.6	1.0007	49.456	10.152	317
β_{draw}	0.2	0.2003	0.0512	3282.7	1.0002	50.503	10.367	317
ρ	0.9	0.8985	0.0073	3638.4	1.0	55.976	11.491	317

Notes: We draw 7,150 samples in total and discard the first 650 samples. The sampling time is measured in seconds and excludes `Julia` compilation time.

Table 2 reports the results when, instead, we sample from the posterior using NUTS using our `Julia` library. The length of the samples is chosen such that the total sampling times are roughly the same in both experiments.[11] NUTS traverses the posterior more efficiently than RWMH: the ESS as a percentage of the total draws (ESS%) is several orders of magnitude higher for NUTS. As an example, for β_{draw}, the ESS% is less than 0.5% with RWMH and 50% with NUTS. The efficiency advantage, however, comes with the overhead cost of gradient evaluation. The ESS per second of the HMC approach is

[11] All timed numerical experiments in this section are conducted on an AWS t3.xlarge instance with four vCPUs and 16 GiB memory. Untimed experiments and the empirical application are run on an m5.8xlarge instance with 32 vCPUs and 128 GiB memory.

only between four and seven times higher than for RWMH for α and β_{draw}, which are harder to explore, but drops 60% for ρ, whose posterior distribution is easier to transverse for RWMH. As an end result, both of the posteriors drawn with RWMH and HMC are centered around the pseudotrue parameter values and their standard deviations are similar.[12]

Table 3: NUTS with Joint Likelihood, RBC Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.3	0.2982	0.0071	41.168	1.0191	1.0292	0.0501	822
β_{draw}	0.2	0.1932	0.0504	84.815	1.0048	2.1204	0.1032	822
ρ	0.9	0.8982	0.0075	248.1	1.0064	6.2024	0.3019	822

Notes: We draw 4,400 samples in total and discard the first 400 samples. The sampling time is measured in seconds and excludes `Julia` compilation time.

Table 3 reports the results from the joint likelihood method when we draw jointly the parameters and the latent states using NUTS. While we still get roughly the same posteriors, the ESS% drops because now we draw from a much higher dimensional parameter space: $3 + 200 = 203$ overall. Also, we need more time (822 seconds) to get convergence.

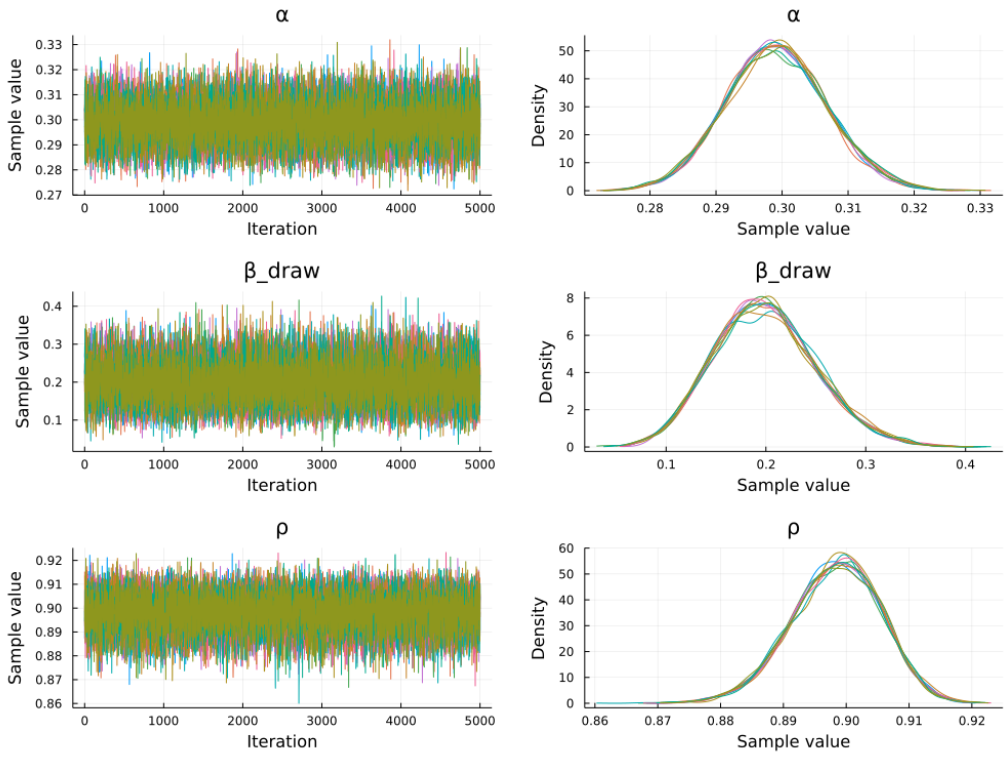


Figure 1: NUTS with Marginal Likelihood, RBC Model, First-order

[12]To monitor convergence, we employ the R-hat value, which measures the comparability of samples over draws and across chains and, by doing so, provides a diagnostic for failures of the chain to provide representative samples from the posterior space (Gelman and Rubin, 1992). Values substantially greater than 1.0 indicate systematic differences across chains that may result from poor mixing.

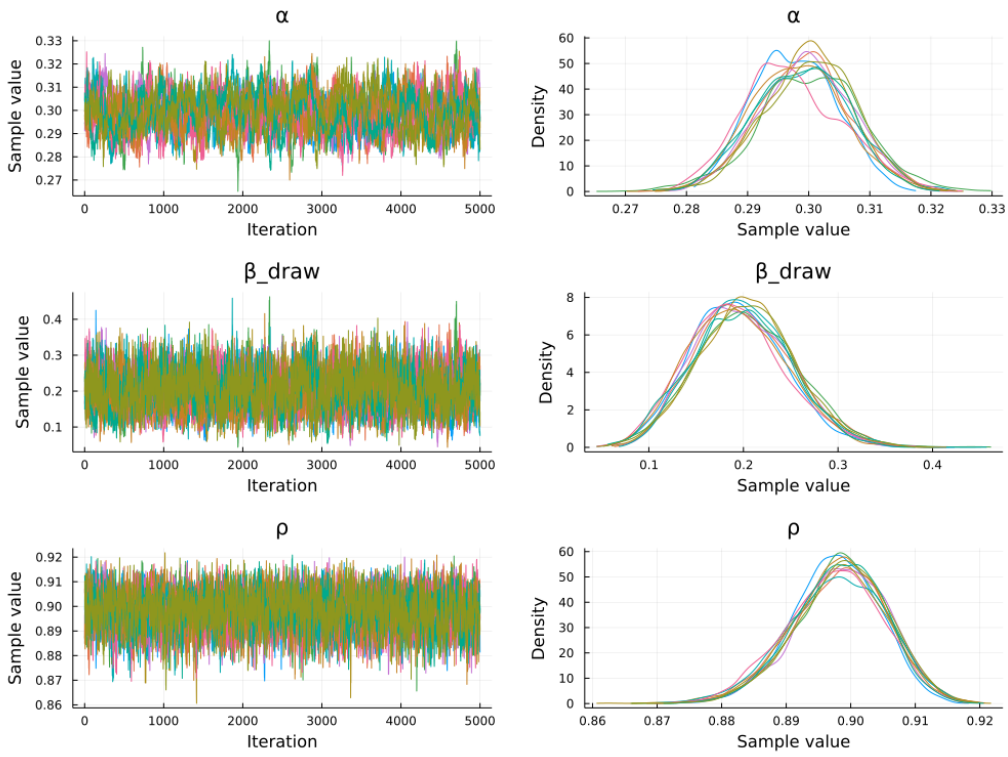


Figure 2: NUTS with Joint Likelihood, RBC Model, First-order

Nonetheless, this experiment shows how NUTS can work in the joint likelihood case. To see this, we can look at the trace plots and the posterior densities of 10 chains sampled from the posterior, each with 5000 draws. Figure 1 shows the case for the marginal likelihood and Figure 2 the scenario for the joint likelihood (here and in all the relevant figures, the posterior is reported after applying a kernel density smoother). Even with just 5000 draws, the NUTS sampler traverses the posterior nearly as efficiently in the joint likelihood case as in the marginal likelihood scenario.

Table 4: RWMH with Marginal Likelihood on Particle Filter, RBC Model, Second-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.3	0.3057	0.0074	43.986	1.0287	0.4887	0.0034	13127
β_{draw}	0.2	0.2248	0.0447	33.342	1.0434	0.3705	0.0025	13127
ρ	0.9	0.9023	0.0064	414.87	1.0068	4.6097	0.0316	13127

Notes: We draw 10,000 samples in total and discard the first 1000 samples. The sampling time is measured in seconds and excludes model file generation and compilation.

Table 4 reports the results of estimating the second-order RBC model. In this case, the data-generating process is no longer linear-Gaussian. Hence, we apply the particle filter to filter through the latent variables in the marginal likelihood approach. For our comparisons, we apply the bootstrap particle filter (arguably the simplest variant of particle filter) in `Dynare`, with 20,000 particles.[13]

[13]The particle size chosen is within the range of those commonly used in DSGE applications and around

Table 5: NUTS with Joint Likelihood, RBC Model, Second-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.3	0.3053	0.0077	89.406	1.0131	2.2351	0.0355	2519
β_{draw}	0.2	0.2243	0.046	115.37	1.009	2.8842	0.0458	2519
ρ	0.9	0.9021	0.0047	481.7	1.0046	12.042	0.1912	2519

Notes: We draw 11,000 samples in total and discard the first 1000 samples. The sampling time is measured in seconds and excludes `Julia` compilation time.

Table 5 reports the results that NUTS with joint likelihood yields for the same second-order RBC model using our `Julia` library. The performance of the joint likelihood approach is clearly superior to the one with RWMH. The HMC sampler only takes around 19.2% of the time to deliver results comparable to or better than RWMH in terms of the R-hat value.[14]

Figure 3 plots the posterior densities of β_{draw} obtained with each sampler. The red line is the posterior density from RWMH with 9,000 draws and the black line is the posterior density from NUTS and 10,000 draws. The posterior from RWMH suggests a lack of convergence. Even a NUTS with 4,000 draws (blue line) is smoother than the posterior from RWMH, despite taking much less time to run.

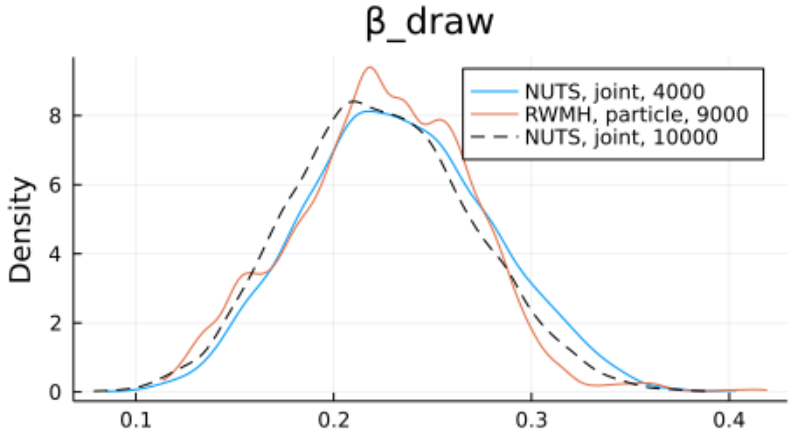


Figure 3: Posterior Density of β_{draw}

Figure 4 shows 10 independent samples from the posterior using the HMC joint likelihood case. The results are mixing well, even if we are drawing samples from a relatively high dimensional space. Figure 5 displays the scatter plots of the pairwise joint distributions across the three parameters in the RBC model. The three methods yield similar joint distributions of the samples, with α and β highly correlated and ρ near independent.

the smallest that yielded posterior estimates of acceptable quality. Experiments with fewer particles yielded slow mixing or completely stopped chains, a known issue for particle filters (Pitt et al., 2012).

[14]We do not compute in any of our estimates the case with HMC and the particle filter, because the latter is not differentiable. While there are some proposals for differentiable particle filters (e.g., Corenflos et al., 2021), they remain an area of active research.

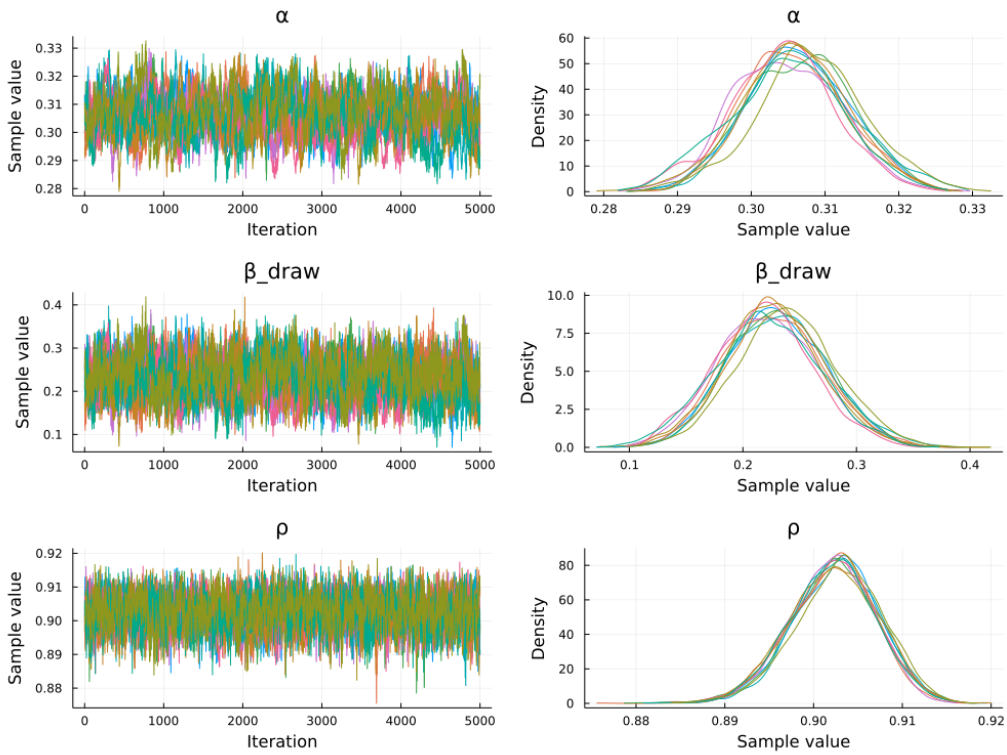
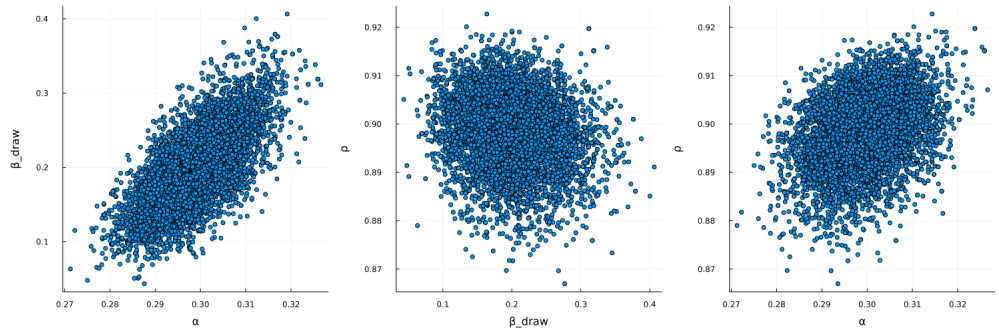
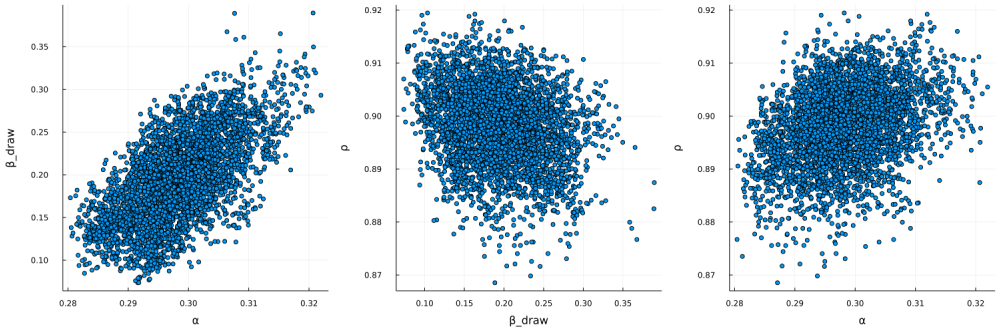


Figure 4: NUTS with Joint Likelihood, RBC Model, Second-order

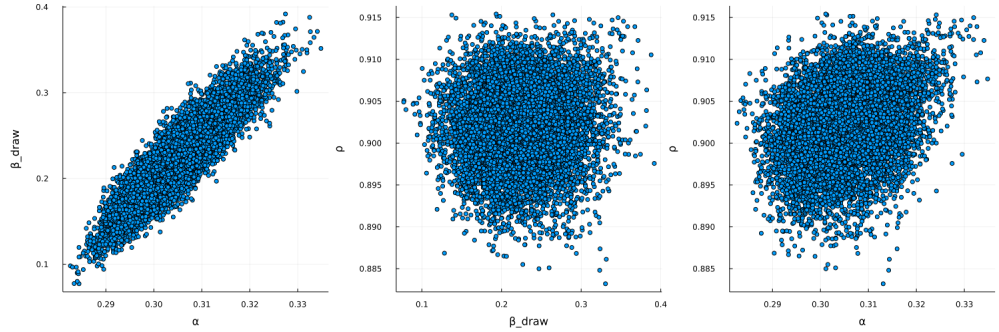
The joint likelihood approach provides an interesting by-product. Since we draw from the latent state variables, we can back them out without having to run a smoother. Figure 6 compares the estimated posterior mean of the TFP shocks (blue lines) and their true value (red lines), with the left panel for the first-order RBC and right panel for the second-order RBC. The light blue region shows two standard deviation credible regions. Our estimates back out the value of original shocks accurately and the regions show the heteroskedasticity in the shock posteriors.



(a) First-order, Marginal Likelihood



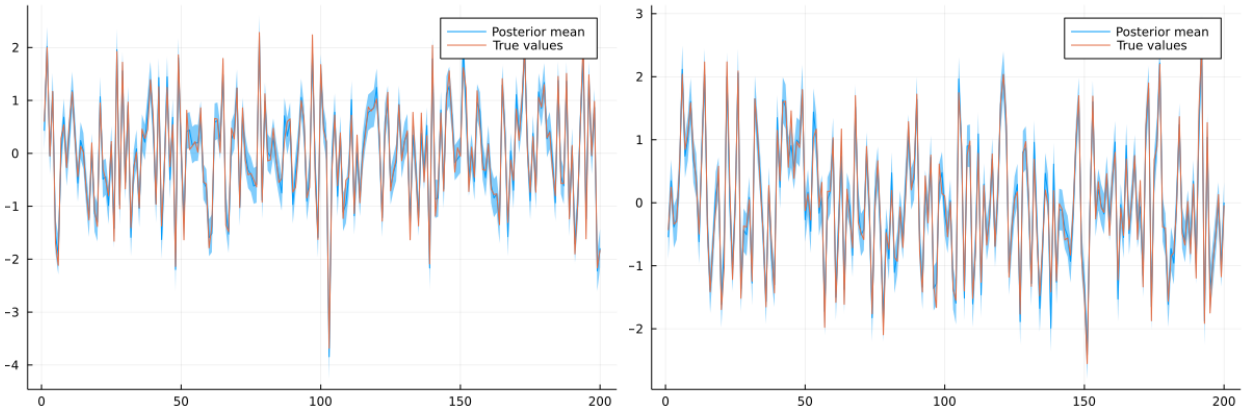
(b) First-order, Joint Likelihood



(c) Second-order, Joint Likelihood

Figure 5: Joint Distribution of Parameters

Notes: The left panel shows the correlation with α on the horizontal axis and β_{draw} on the vertical axis. The middle panel shows the correlation with β_{draw} on the horizontal axis and ρ on the vertical axis. The right panel shows the correlation with α on the horizontal axis and ρ on the vertical axis.



(a) First-order RBC (b) Second-order RBC

Figure 6: Inferred TFP Shocks of RBC Model

Frequentist Statistics. While we carry out Bayesian estimation, frequentist statistics can provide a complementary check. More concretely, we estimate the RBC model with all three methods mentioned above with multiple samples of simulated data from the model. The pseudotrue values of the underlying parameters are the same, but each generated sample is different because shock realizations vary. Moreover, we generate samples of different lengths to gauge small sample effects.

Table 6: Frequentist Statistics – Kalman

	Parameters	Mean Bias	MSE	Cov. Prob. 80%	Cov. Prob. 90%
$T = 50$	α	0.0013	7.18×10^{-5}	92%	98%
	β_{draw}	0.0347	0.0030	93%	96%
	ρ	-0.006	0.0002	82%	95%
$T = 100$	α	0.0009	5.26×10^{-5}	91%	95%
	β_{draw}	0.0171	0.0016	95%	96%
	ρ	-0.003	9.44×10^{-5}	79%	93%
$T = 200$	α	0.0009	4.45×10^{-5}	87%	94%
	β_{draw}	0.0139	0.0014	90%	98%
	ρ	-0.001	3.44×10^{-5}	82%	87%

Notes: All these statistics are generated from 100 estimation replications. We draw 5500 samples in total and discard the first 500 samples.

Table 6 shows the frequentist statistics for the estimation exercises for the first-order RBC with the Kalman filter and HMC. The first two columns show the mean and standard deviation of the estimated mean with respect to the pseudotrue value. The third and fourth columns show the coverage probability, i.e., whether the 80% and 90% credible interval of the estimated posterior contains the pseudotrue value. The three panels from top to bottom vary the sample length, T, from 50 to 200.[15] As lengthier data lead to more accurate identification, the mean bias and mean squared error both decrease, on average, as T increases. Also, coverage probabilities become more accurate.

Table 7 and 8 report the frequentist statistics for the joint likelihood approach in first- and second-order respectively. We see patterns similar to those in the previous case, which means the joint likelihood approach we propose is credible and robust to different shock realizations.

Robustness. Interestingly, we find that the joint likelihood approach is more robust than the marginal likelihood approach with respect to the starting values of the chain. We

[15]While Bayesian credible intervals need not have the coverage properties of confidence intervals and posterior summaries like the mean need not be unbiased point estimates in a frequentist sense, the Bernstein–von Mises theorem tells us that, for well-identified regular models, they have these properties asymptotically when maximum likelihood estimates do (as in our model).

Table 7: Frequentist Statistics – First-order Joint

	Parameters	Mean Bias	MSE	Cov. Prob. 80%	Cov. Prob. 90%
$T = 50$	α	0.0013	7.26×10^{-5}	92%	97%
	β_{draw}	0.0353	0.0031	93%	96%
	ρ	-0.006	0.0002	81%	93%
$T = 100$	α	0.0009	5.26×10^{-5}	92%	96%
	β_{draw}	0.0172	0.0016	95%	96%
	ρ	-0.003	9.50×10^{-5}	78%	94%
$T = 200$	α	0.0008	4.59×10^{-5}	85%	93%
	β_{draw}	0.0130	0.0013	89%	97%
	ρ	-0.001	3.48×10^{-5}	81%	87%

Notes: All these statistics are generated from 100 estimation replications. We draw 5500 samples in total and discard the first 500 samples.

Table 8: Frequentist Statistics – Second-order Joint

	Parameters	Mean Bias	MSE	Cov. Prob. 80%	Cov. Prob. 90%
$T = 50$	α	-0.001	7.14×10^{-5}	96%	98%
	β_{draw}	0.0313	0.0027	94%	98%
	ρ	-0.009	0.0004	74%	84%
$T = 100$	α	0.0010	4.46×10^{-5}	94%	96%
	β_{draw}	0.0181	0.0017	94%	100%
	ρ	-0.002	8.69×10^{-5}	72%	90%
$T = 200$	α	0.0013	3.36×10^{-5}	88%	98%
	β_{draw}	0.0086	0.0017	84%	92%
	ρ	-0.001	2.03×10^{-5}	88%	94%

Notes: All these statistics are generated from 50 estimation replications. We draw 5500 samples in total and discard the first 500 samples.

illustrate this robustness by starting the samplers from a Cartesian product of grids on α, β, and ρ respectively.[16] Figure 7 and Figure 8 show the cumulative mean values and Figure 9 and Figure 10 show the trace plots.

While different sampling methods share similar robustness in estimating the first-order RBC model, the results differ for the second-order case. For example, the upper panel of Figure 8 is from our joint likelihood approach with samples drawn from HMC, and the lower panel of Figure 8 shows the counterparts from particle-filtered marginal likelihood with RWMH run with `Dynare`. The parameters, from left to right, are α, β_{draw},

[16] We create a grid of 4 points for each of these three parameters, and the Cartesian product will be 64 starting points overall. The grid for α is $[0.25, 0.3, 0.35, 0.4]$. The grid for β_{draw} is $[0.1, 0.175, 0.25, 0.325]$. The grid for ρ is $[0.4625, 0.625, 0.7875, 0.95]$.

and ρ respectively, with the black dashed line showing the pseudotrue values that generated the simulated data.

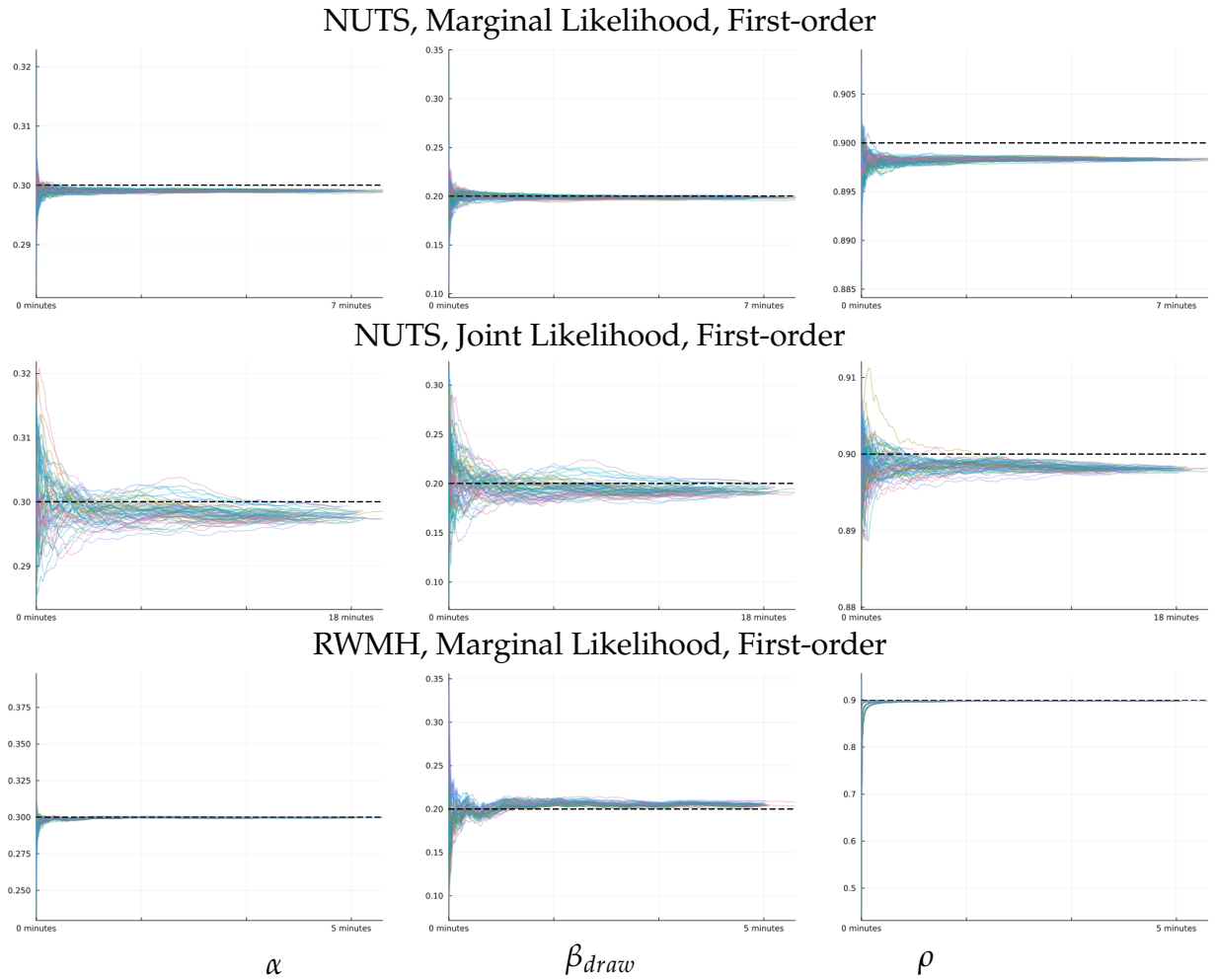


Figure 7: Robustness Comparison on First-order RBC: Cumulative Mean

Figure 8 tells us that even in a relatively simple problem like second-order RBC models, RWMH with a particle filter takes a long time to converge to the high-density region of the posterior, while the HMC sampler with the joint likelihood approaches a stable target value quickly across different starting points. This phenomenon is rooted in the mechanism of the sampling methods. Since HMC utilizes gradient information, the sampler arrives at the typical set quickly, even in a high-dimensional sample space. RWMH, on the other hand, takes only gradual steps toward the typical set when we do not start at it. Comparing the time it takes for the cumulative mean of the parameter draws for the second-order RBC model to approach a stable value, HMC is seen to be more time-efficient even at the cost of gradient computations. This is more apparent in the trace plots, Figure 10, which have an even dispersion and limited autocorrelation almost from the beginning for the HMC approach, but display limited dispersion and high persistence for the RWMH approach. Thus, other moments of the posterior distribution beyond the

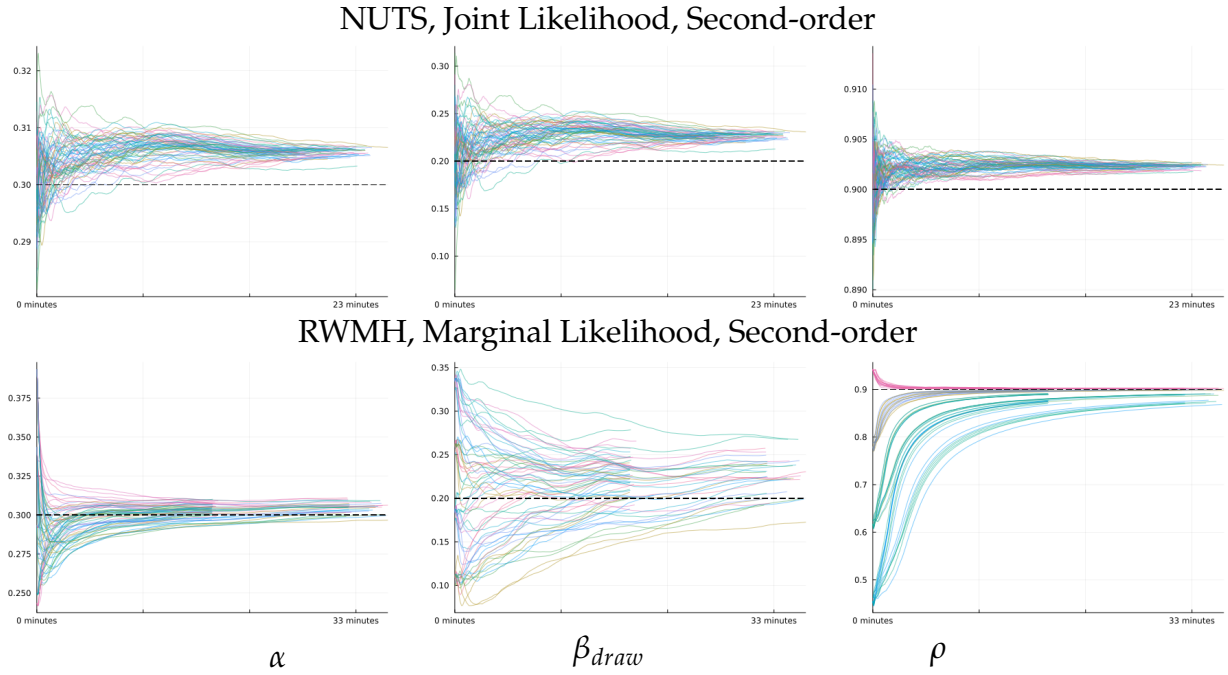


Figure 8: Robustness Comparison on Second-order RBC: Cumulative Mean

mean may display even less robustness when using RWMH.

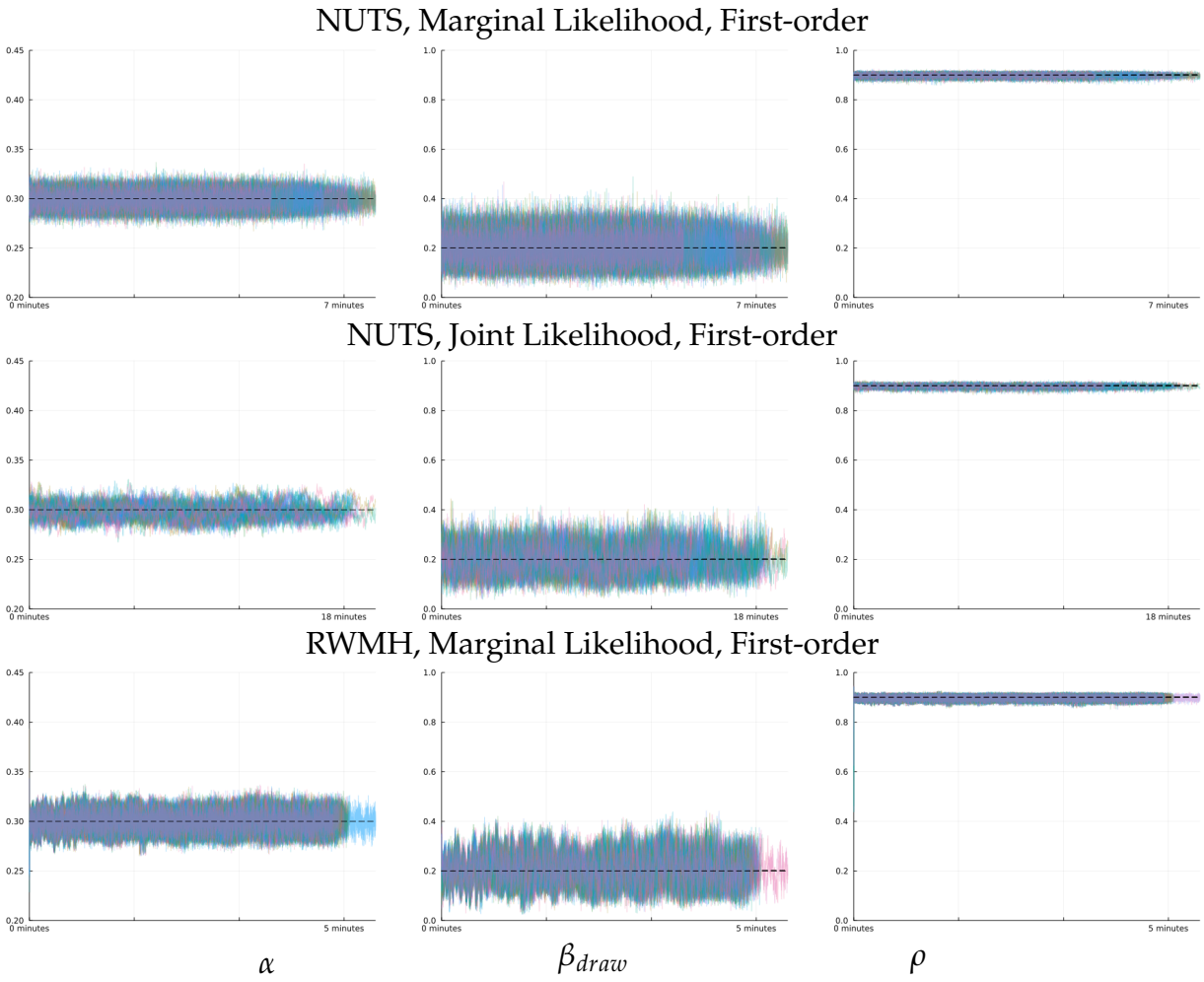


Figure 9: Robustness Comparison on First-order RBC: Trace Plot

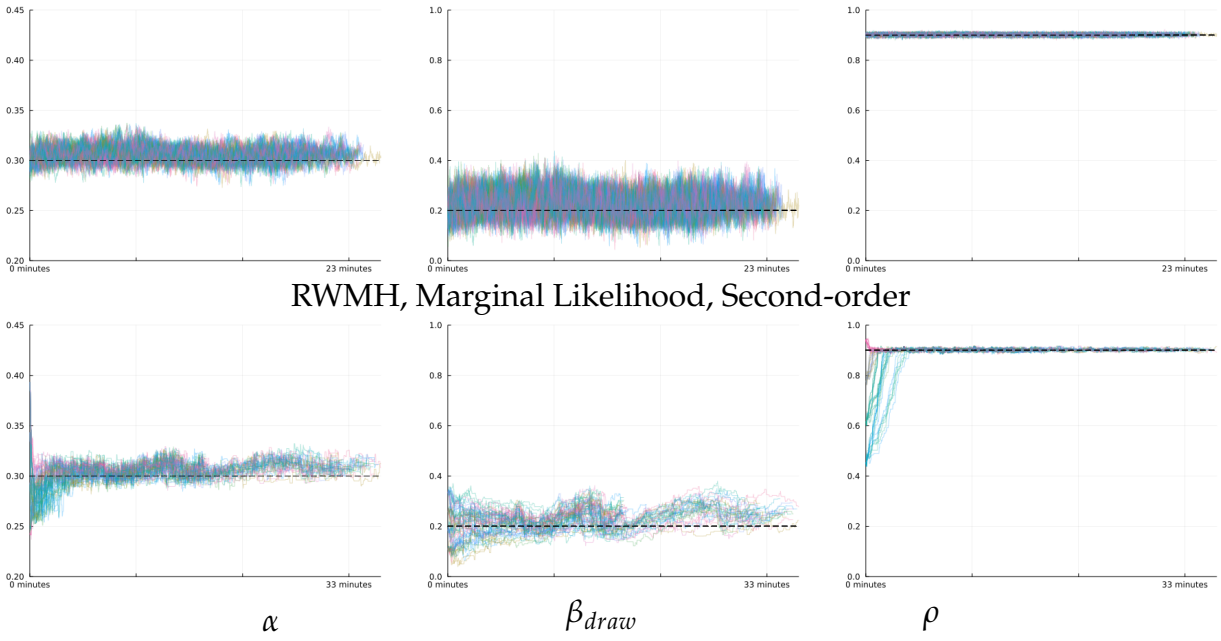


Figure 10: Robustness Comparison on Second-order RBC: Trace Plot

6.2 Estimating a Small Open Economy Model

To demonstrate how performance scales with model size and complexity, we apply our methods to estimate a slightly larger model, a version of the real small open economy model of Schmitt-Grohé and Uribe (2003). We use a model based on their model 2, with debt elastic interest rates, but in addition to the TFP shock, we add two AR(1) shocks: a shock to the intertemporal marginal utility of consumption and a shock to the interest rate premium representing global stochastic discount factor variation as in Arellano (2008). Thus, we can estimate the model with three observables: output, interest rates, and the current account balance, all measured with Gaussian observation noise with a variance of 0.001. Beyond these additions, the model equations are identical to those used in Schmitt-Grohé and Uribe (2003) and so are deferred to Appendix H.

We simulate two samples of 200 periods with pseudotrue values taken from Schmitt-Grohé and Uribe (2003) and reproduced in Appendix H.[17] We estimate seven structural parameters, including α, β_{draw}, and ρ (the parameters analogous to the ones we estimated in the RBC model), as well as the persistence of the additional two shocks ρ_u and ρ_v, the Frisch elasticity of labor supply γ, and the response of the interest premium to debt ψ. The priors are described in Table 16 in Appendix H.

We estimate the first-order model using the marginal likelihood computed by Kalman filter both via HMC and RWMH. We also estimate at first order using the joint likelihood approach with HMC. At second order, we compare HMC with the joint likelihood approach and RWMH with the approximate marginal likelihood computed with the particle filter. First-order results are in Tables 9-11, and second-order in Tables 12 and 13.

Table 9: NUTS with Marginal Likelihood, SGU Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.32	0.3096	0.0176	5833.2	1.0000	116.66	5.3217	1096
γ	2.0	2.1082	0.4123	4881.7	1.0004	97.634	4.4537	1096
ψ	$7.42 \cdot 10^{-4}$	$8.28 \cdot 10^{-4}$	$2.82 \cdot 10^{-4}$	7068.4	0.9998	141.37	6.4486	1096
β_{draw}	4	3.7819	0.1571	5875.8	1.0001	117.52	5.3606	1096
ρ	0.42	0.4742	0.0494	6044.9	1.0000	120.90	5.5148	1096
ρ_u	0.2	0.1264	0.0552	5742.3	0.9998	114.85	5.2388	1096
ρ_v	0.4	0.4144	0.0933	4021.2	0.9998	80.424	3.6686	1096

Notes: We draw 6,500 samples in total and discard the first 1,500 samples. The sampling time is measured in seconds and the acceptance rate is automatically tuned to 65%.

By estimating a model similar to the RBC but with 3 latent shocks and 3 observable variables instead of 2 shocks and 2 observables, and with 7 estimated parameters instead of 3, we show the improved relative performance of HMC in larger models. In terms of speed as measured by ESS/second, at first order with the marginal likelihood, HMC

[17] The structure of the experiment is the same as that for the RBC model. One sample comes from the first-order solution and the other from the second-order solution. The estimation with RWMHs is undertaken in `Dynare` and the estimation with HMC in our `Julia` library.

Table 10: RWMH with Marginal Likelihood, SGU Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.32	0.3252	0.0148	4310.0	1.0000	0.0035	2.1842	1973
γ	2.0	2.5377	0.4158	2709.8	1.0030	0.0022	1.3732	1973
ψ	$7.42 \cdot 10^{-4}$	$7.78 \cdot 10^{-4}$	$2.7 \cdot 10^{-4}$	10949.0	1.0000	0.0090	5.5484	1973
β_{draw}	4	4.2672	0.1215	2734.6	1.0006	0.0023	1.3858	1973
ρ	0.42	0.3618	0.0523	2857.7	1.0003	0.0024	1.4482	1973
ρ_u	0.2	0.2976	0.0653	2802.0	1.0024	0.0023	1.4200	1973
ρ_v	0.4	0.1842	0.0834	2765.1	1.0076	0.0023	1.4013	1973

Notes: We draw 1,350,000 samples in total and discard the first 135,000 samples. The sampling time is measured in seconds and excludes model file generation and compilation.

Table 11: NUTS with Joint Likelihood, SGU Model, First-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.32	0.3083	0.0180	3653.5	1.0000	73.070	1.2544	2912
γ	2.0	2.1508	0.4231	4581.1	0.9998	91.622	1.5729	2912
ψ	$7.42 \cdot 10^{-4}$	$7.45 \cdot 10^{-4}$	$2.83 \cdot 10^{-4}$	5629.5	1.0006	112.59	1.9329	2912
β_{draw}	4	3.9452	0.1728	1643.0	1.0000	32.860	0.5641	2912
ρ	0.42	0.4064	0.0540	1843.8	0.9998	36.876	0.6331	2912
ρ_u	0.2	0.2215	0.0684	1572.9	1.0012	31.458	0.5401	2912
ρ_v	0.4	0.3237	0.1081	2523.9	0.9999	50.479	0.8666	2912

Notes: We draw 6,500 samples in total and discard the first 1,500 samples. The sampling time is measured in seconds and the acceptance rate is automatically tuned to 90%.

dominates for all parameters, achieving a sampling rate over 3 times as fast for most parameters. The joint likelihood approach, which must sample 600 latent shock values in addition to the 7 structural parameters, remains somewhat slower than RWMH for most parameters, but never by more than a factor of 3.

But while RWMH with the Kalman filter is faster in terms of speed than the NUTS with the joint likelihood approach, it does worse in terms of posterior quality. While all procedures display acceptable R-hat statistics, RWMH densities, shown in Figure 11, display non-smoothness characteristic of uneven local exploration while HMC densities do not.[18]

Table 12: NUTS with Joint Likelihood, SGU Model, Second-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.32	0.3117	0.0160	1707.5	0.9999	34.150	0.2154	7926
γ	2.0	1.9895	0.3637	1943.6	0.9998	38.873	0.2452	7926
ψ	$7.42 \cdot 10^{-4}$	$6.66 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$	1354.1	1.0011	27.083	0.1708	7926
β_{draw}	4	4.0892	0.0563	1076.5	1.0003	21.530	0.1358	7926
ρ	0.42	0.4557	0.0523	991.79	1.0004	19.836	0.1251	7926
ρ_u	0.2	0.1616	0.0642	781.89	1.0040	15.638	0.0986	7926
ρ_v	0.4	0.3772	0.0561	1772.3	0.9998	35.447	0.2236	7926

Notes: We draw 6,500 samples in total and discard the first 1,500 samples. The sampling time is measured in seconds and the acceptance rate is automatically tuned to 65%.

At second order, the differences are much more dramatic: as measured by ESS/sec-

[18] Because data are simulated separately for the RWMH and `Julia` runs, and the joint and Kalman approaches differ in period 0 initialization, caution is warranted in interpreting visible differences in posterior centering, which may reflect sampling differences.

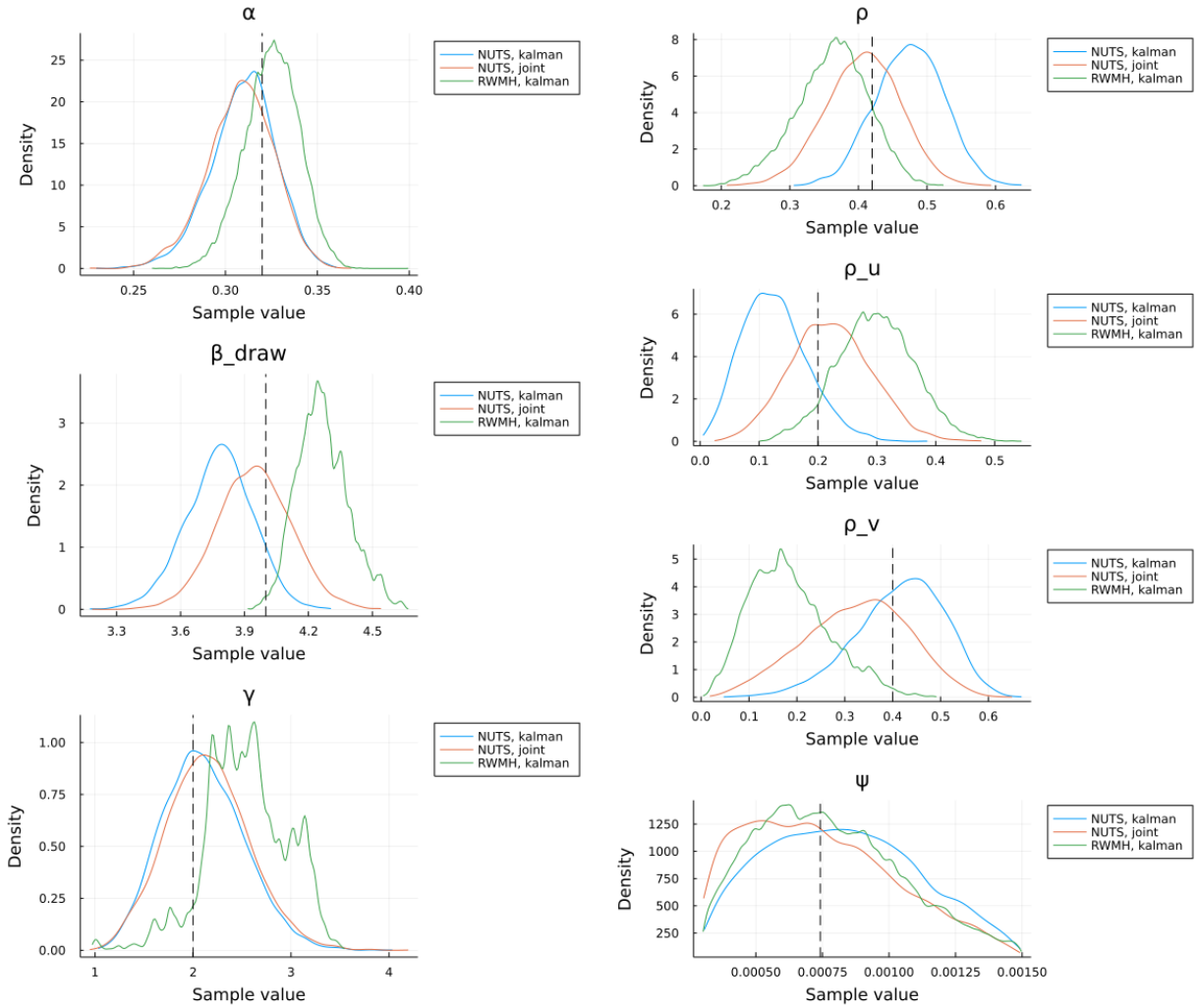


Figure 11: First Order RWMH+Kalman vs. HMC+Kalman vs. HMC+Joint

ond, HMC outperforms RWMH with the particle filter by a factor of 10-40x. Further, this raw speed difference understates the performance difference, because the particle filter approach exhibits substantial quality issues. This can be seen in the R-hat statistics, which remain substantially above 1 for some parameters even with 60,000 particles used and 100,000 draws. This issue is also visible in the estimated posterior densities, shown in Figure 12, which deviate from the HMC estimates in location and shape. These quality problems are a consequence of the slow and uneven mixing of RWMH methods even when the raw number of draws is large.

Compared to the RBC experiments, the performance and quality advantages for HMC for this larger model are even more pronounced, particularly at second order. Given theoretical results suggesting that the computational cost of particle filtering can scale exponentially in the dimension of the latent parameter space (Rebeschini and van Handel, 2015) while HMC can scale polynomially in total number of parameters for well-behaved

Table 13: RWMH with Marginal Likelihood on Particle Filter, SGU Model, Second-order

Parameters	Pseudotrue	Post. Mean	Post. Std.	ESS	R-hat	ESS%	ESS/second	Time
α	0.32	0.2975	0.0201	253.97	1.0008	0.0028	0.0068	37556
γ	2.0	1.4900	0.1673	201.79	1.3876	0.0022	0.0054	37556
ψ	$7.42 \cdot 10^{-4}$	$7.92 \cdot 10^{-4}$	$2.78 \cdot 10^{-4}$	472.07	1.0007	0.0052	0.0126	37556
β_{draw}	4	4.5813	0.1409	203.95	1.0442	0.0023	0.0054	37556
ρ	0.42	0.3772	0.0443	212.68	1.0010	0.0024	0.0057	37556
ρ_u	0.2	0.3226	0.0822	204.25	1.0239	0.0023	0.0054	37556
ρ_v	0.4	0.2933	0.0624	206.64	1.0378	0.0023	0.0055	37556

Notes: We draw 100,000 samples in total and discard the first 10,000 samples. We use 60,000 particles. The sampling time is measured in seconds and excludes model file generation and compilation.

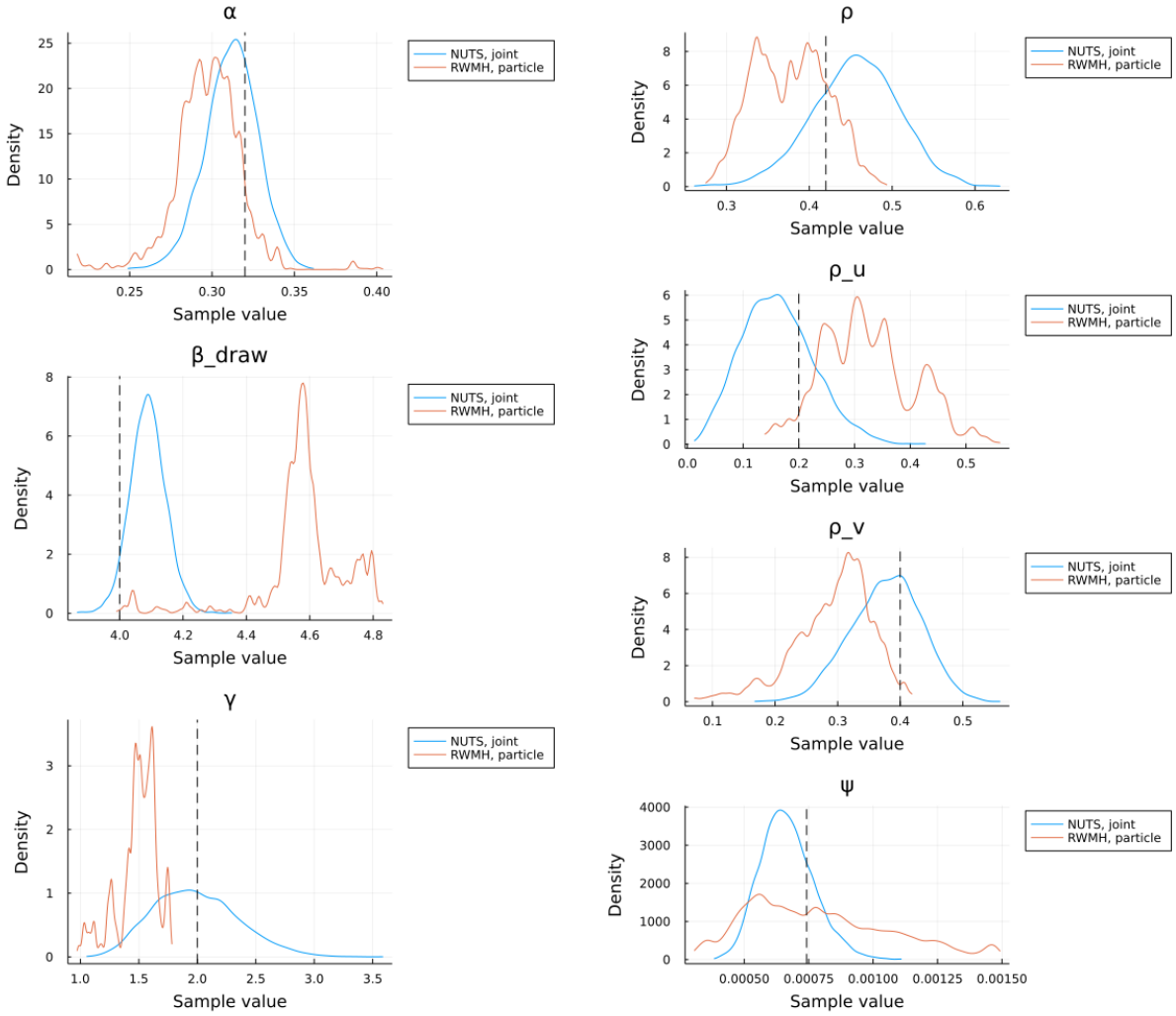


Figure 12: Second Order RWMH+Particle vs. HMC+Joint

posterior distributions (see Appendix G), this suggests that HMC may be particularly useful for estimating nonlinear models at medium scales.

6.3 Estimating a New Keynesian Model

To demonstrate our methods in a larger scale application using real data, we estimate a version of Fernández-Villaverde and Guerrón-Quintana (2021) (FVGQ), a canonical medium-scale New Keynesian model. We have a representative household that consumes, saves, supplies labor, and holds monet. A final good firm produces output with a continuum of intermediate goods that are produced by monopolistic competitors facing Calvo-type nominal rigidities. The representative household is the owner of all these firms. The government sets monetary and fiscal policy. There are two unit root processes: one governs the level of Hicks-neutral TFP and the other investment-specific technology.[19]

Table 14: Prior Distribution for Structural Parameters

Parameter	Distribution	Mean	Std
β_{draw}	Gamma	0.25	0.1
h	Beta	0.7	0.1
κ	Normal	4	1.5
α	Normal	0.3	0.05
θ_p	Beta	0.5	0.1
χ	Beta	0.5	0.15
γ_R	Beta	0.75	0.1
γ_y	Normal	0.12	0.05
γ_Π	Normal	1.5	0.25
$100\,(\bar{\Pi}-1)$	Gamma	0.95	0.1
$\bar{g}$	Beta	0.3	0.05
ρ_d	Beta	0.5	0.2
ρ_φ	Beta	0.5	0.2
ρ_g	Beta	0.5	0.2
σ_A	Inverse Gamma	0.1	1
σ_d	Inverse Gamma	0.1	1
σ_ϕ	Inverse Gamma	0.1	1
σ_μ	Inverse Gamma	0.1	1
σ_m	Inverse Gamma	0.1	1
σ_g	Inverse Gamma	0.1	1
Λ_μ	Gamma	0.0034	0.001
Λ_A	Gamma	0.00178	0.00075

There are six exogenous shocks in the model: household consumption preference, labor supply, TFP, investment efficiency, fiscal policy, and monetary policy. The first two shocks are to preferences, and the third and fourth are to the supply-side. We select six time series in real data for our estimation: inflation measured by CPI, the federal

[19] See Fernández-Villaverde and Guerrón-Quintana (2021) for a full description of model and data sources. Our implementation is identical except for the choice of priors, summarized below.

funds rate, the growth rate of real wages, the growth rate of real GDP per capita, per capita working hours, and the inverse relative price of investment with respect to the price of consumption growth. This model contains 28 parameters to estimate and Table 14 summarizes the priors.

We estimate the model using first- and second-order perturbations, and, within first-order, produce estimates using both the marginal likelihood approach with the Kalman filter and the joint likelihood approach. In all versions, the observation equations are augmented with isotropic Gaussian measurement noise with variance $4e-4$ for each variable.

Table 15: Estimation, FVGQ model

Parameters	**Mean**			**Std.**			**ESS**			**R-hat**		
	Kalman	Joint 1st	Joint 2nd	Kalman	Joint 1st	Joint 2nd	Kalman	Joint 1st	Joint 2nd	Kalman	Joint 1st	Joint 2nd
β_{draw}	0.2107	0.2091	0.2034	0.0778	0.0795	0.0777	364.18	356.43	781.69	1.0030	1.0024	1.0037
h	0.7534	0.7372	0.7538	0.1137	0.1188	0.1148	189.01	181.83	502.38	1.0035	1.0144	1.0004
κ	4.2667	4.1932	4.1359	1.3054	1.2162	1.2679	499.63	468.10	1475.4	1.0091	1.0006	1.0002
χ	0.4893	0.5086	0.5060	0.1501	0.1517	0.1450	218.89	250.94	1199.4	1.0316	1.0002	1.0000
γ_R	0.4636	0.4806	0.4650	0.0745	0.0791	0.0780	262.76	308.30	796.87	1.0004	0.9999	1.0010
γ_Π	1.9077	1.9004	1.8969	0.0761	0.0824	0.0849	315.30	293.19	1797.0	1.0014	1.0022	1.0019
$100\,(\bar{\Pi}-1)$	0.8991	0.8867	0.8961	0.0826	0.0842	0.0800	351.67	225.21	923.10	1.0014	1.0083	1.0003
ρ_d	0.5781	0.5894	0.5924	0.2064	0.2081	0.2098	178.16	133.55	431.25	1.0037	1.0008	0.9999
ρ_φ	0.9619	0.9574	0.9569	0.0235	0.0309	0.0310	250.04	63.954	278.07	1.0000	1.0002	1.0050
ρ_g	0.7921	0.7767	0.7910	0.1570	0.1618	0.1586	128.31	137.32	530.22	1.0025	1.0110	1.0001
$\bar{g}$	0.3656	0.3708	0.3712	0.0543	0.0564	0.0574	316.70	177.06	828.77	1.0003	1.0230	1.0009
σ_A	0.0073	0.0075	0.0075	0.0012	0.0013	0.0013	279.13	229.15	1093.1	0.9999	1.0003	1.0014
σ_d	0.0269	0.0285	0.0296	0.0126	0.0150	0.0171	223.30	181.06	413.28	1.0049	1.0100	1.0000
σ_ϕ	0.0146	0.0142	0.0140	0.0024	0.0022	0.0023	290.50	206.74	677.21	1.0008	0.9999	1.0009
σ_μ	0.0072	0.0072	0.0073	0.0012	0.0011	0.0012	270.72	318.07	816.76	1.0005	1.0067	1.0008
σ_m	0.0078	0.0075	0.0077	0.0015	0.0014	0.0015	214.51	312.64	776.08	1.0073	1.0001	1.0006
σ_g	0.0095	0.0093	0.0095	0.0020	0.0021	0.0020	172.03	106.32	503.85	1.0038	1.0136	1.0002
Λ_μ	0.0037	0.0038	0.0038	0.0009	0.0010	0.0009	238.88	280.80	916.97	1.0035	1.0010	1.0002
Λ_A	0.0015	0.0015	0.0015	0.0005	0.0005	0.0005	313.22	268.18	1344.7	1.0146	1.0032	0.9999

Notes: We draw 7,700 samples in total and discard the first 700 samples for the marginal likelihood and 1st-order joint likelihood estimations. We draw 11,000 samples in total and discard the first 1,000 samples for the 2nd-order joint likelihood estimation. The marginal likelihood takes 18,454 seconds to run, the 1st-order joint takes 16,102 seconds, and the 2nd-order joint takes 236,303 seconds.

Summary statistics from estimates are displayed in Table 15. Inferred shocks for second-order with joint likelihood are plotted in Figure 13, while density and trace plots for second order are in Figure 14. Figures for other approaches are collected in Appendix I. As is apparent from the ESS/second results, both the marginal and joint likelihood methods exhibit comparable speed, in spite of the much larger number of variables for which the joint method must sample. Parameter estimates from both methods, which should produce identical results up to sampling error, are similar, and R-hat statistics and trace plots for both indicate acceptable mixing, both signs indicating that the joint likelihood approach also produces estimates of high quality. The second-order perturbation approximation takes a longer time per effective sample, reflecting the additional cost of model computations, but also produces samples that appear to be well mixed and precisely estimated.

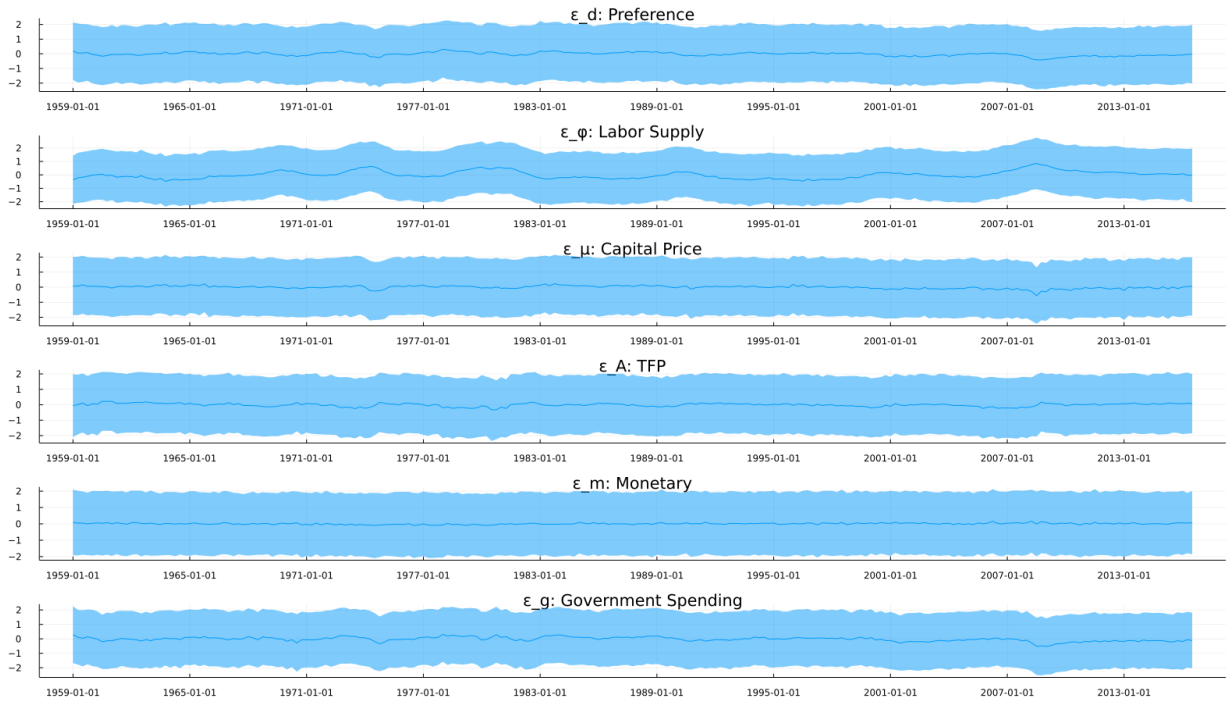


Figure 13: Inferred Shocks of Fernández-Villaverde and Guerrón-Quintana (2021): Second-order with joint likelihood

The level of measurement noise has a substantial impact on the speed and quality of HMC sampling due to the smoothing out of the likelihood function. At low noise levels, due to the weak identification of the parameters, the posterior contains ridges that are difficult to sample from, and so require small steps, regardless of whether gradient information is used. Thus, the estimates suffer from poor mixing and unreliable estimates. With larger noise, the posterior becomes more diffuse and so the sampler can traverse it more rapidly, resulting in fast mixing and an accurate approximation of the true posterior. However, this also results in the data becoming less informative for the parameter values, so the posterior remains close to the prior. However, HMC is not at the core of this issue, but the well-known weak identification of New Keynesian models. Appendix G discusses how these modeling choices relate to posterior geometry and how the performance of HMC varies with this geometry.

Our results indicate that HMC is capable of sampling models in high dimensions, but that other posterior features, including multimodality or near-singularity, may still create problems for the sampler (or for any other MCMC method). In fact, one final advantage of gradient-based methods like HMC is that the ill-conditioning of the gradients allowed diagnosis of the problem, as high curvature can result in gradients that differ strongly from their finite difference approximations, which can be caught in software unit testing. Also, there is active research into more specialized methods that can handle such likelihood features more robustly. Many of these methods, such as that of Graham et al.

(2022), also rely on gradient information, and so we believe that differentiable programming tools will continue to be a key component of a possible new generation of fast and reliable sampling algorithms.

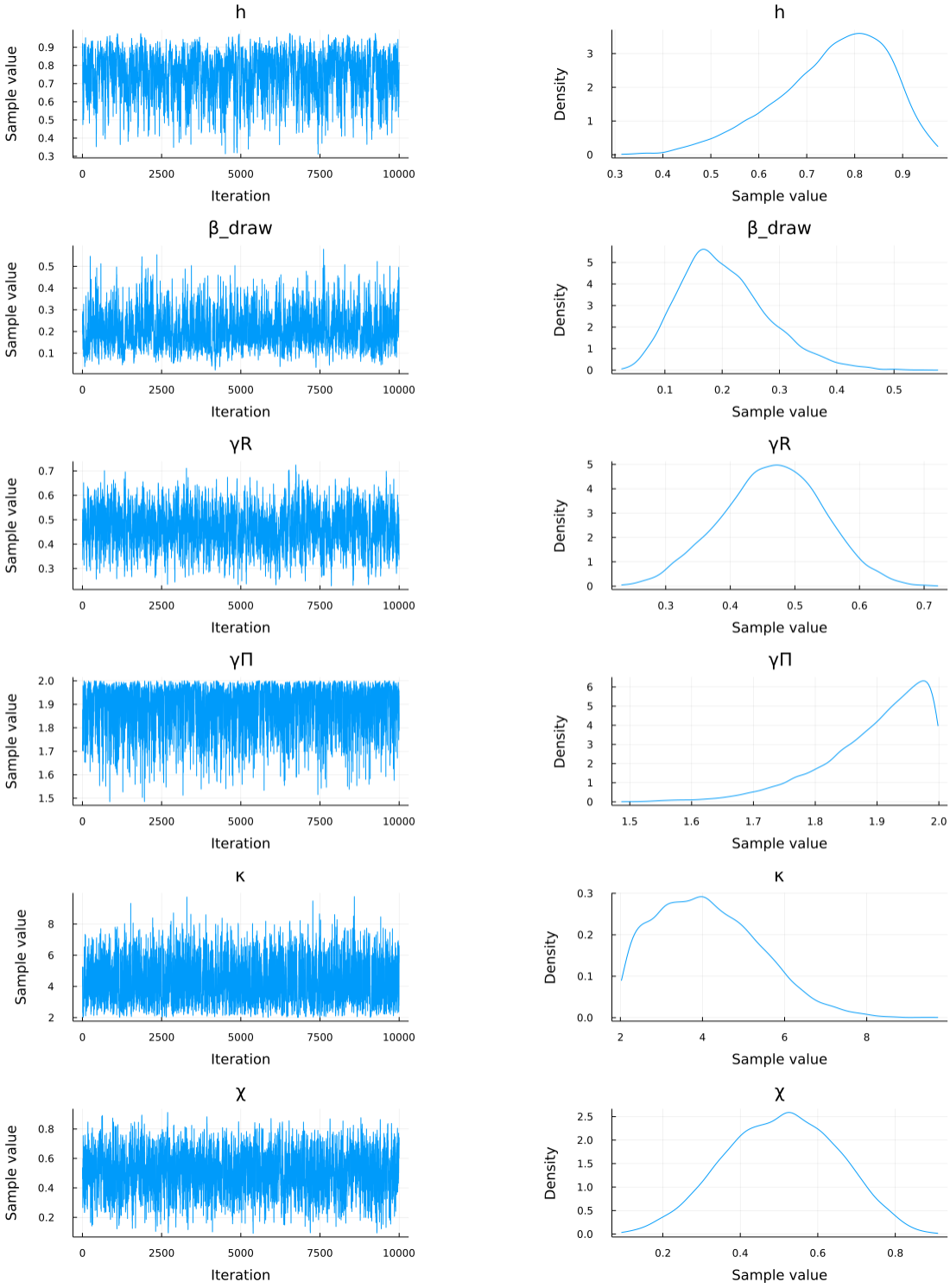


Figure 14: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), Second-order, 1

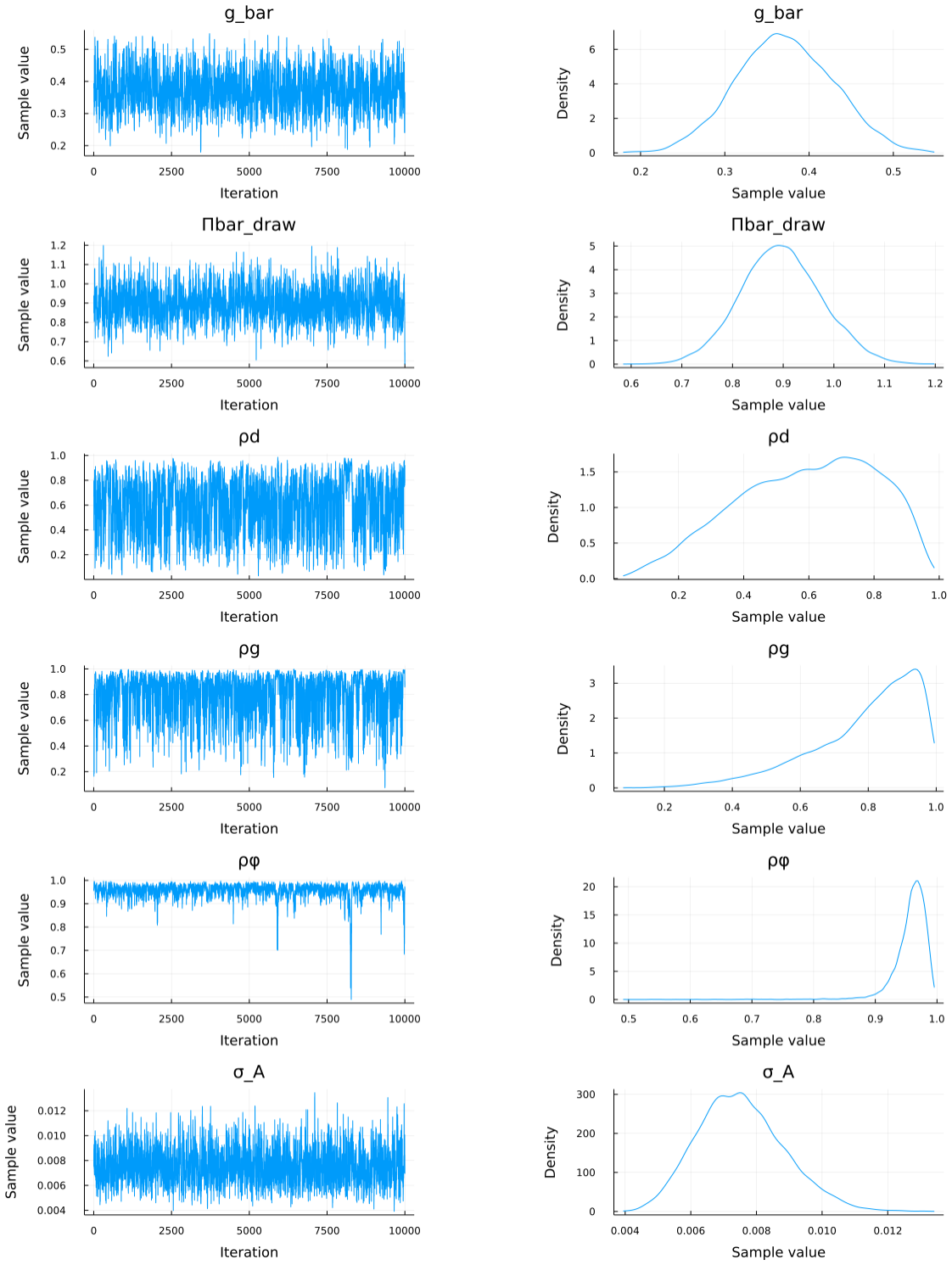


Figure 15: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), Second-order, 2

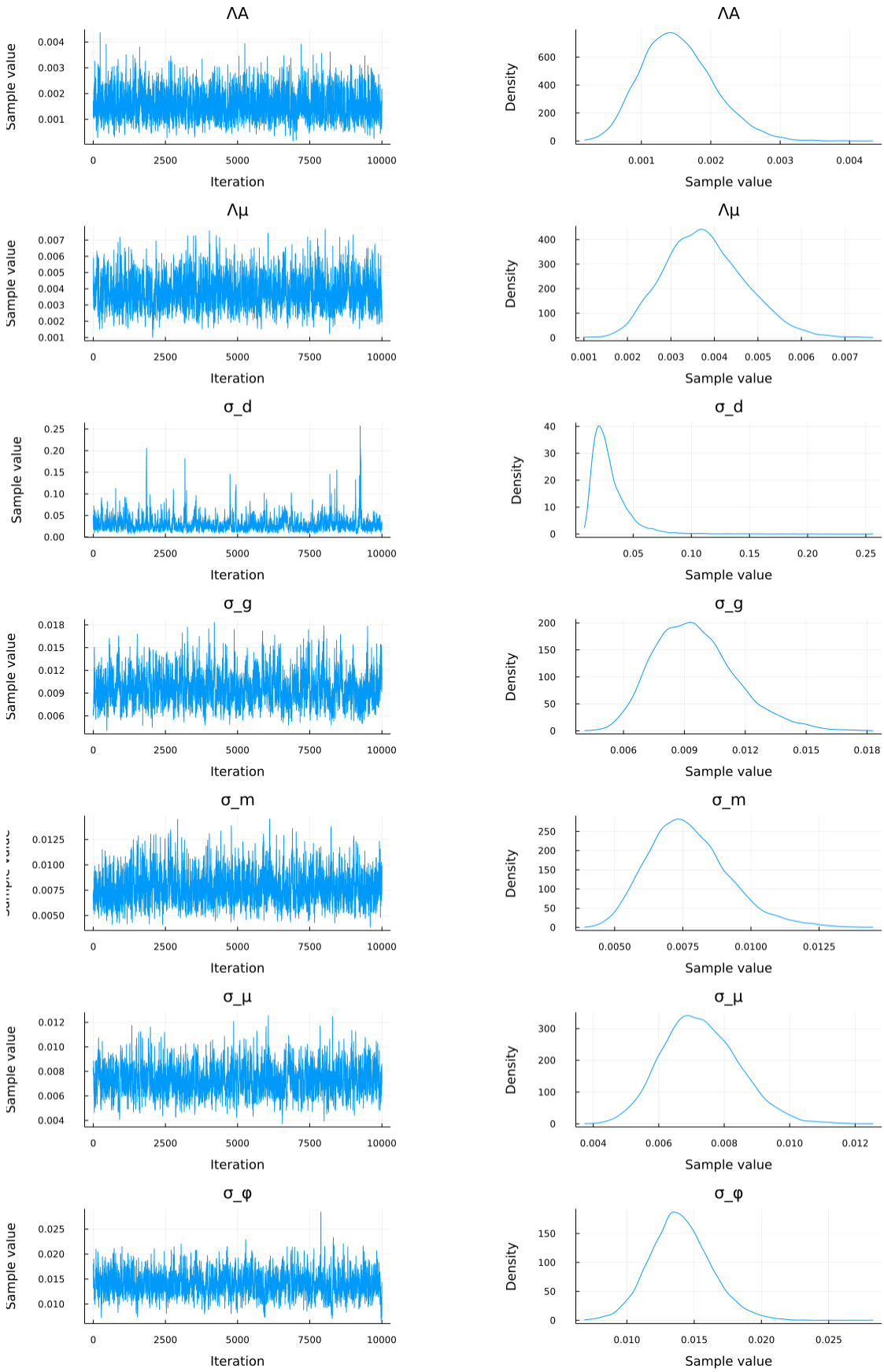


Figure 16: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), Second-order, 3

7 Conclusion

State-space models are widely utilized in economics. The estimation of this set of models is not a trivial task. In particular, estimating nonlinear and non-Gaussian state-space models has always been time-consuming and challenging. In this paper, we have proposed a new set of methods applying HMC with the differentiable programming paradigm to tackle this estimation problem from a novel perspective. We first show that the AD can support relatively complicated derivative computations. Then, we sample the underlying latent state variables along with the model parameters and evaluate the joint likelihood rather than a marginal likelihood only on the parameters.

Future research along this strand can be vibrant. First of all, with the toolkit we provide, a variety of gradient-based estimation methods can be applied to state-space models. For example, variational inference can use gradient-based optimization to produce posterior approximations at greater scale and speed than sampling-based methods. We have already applied Automatic Differentiation Variational Inference (Kucukelbir et al., 2017) to the model in Fernández-Villaverde and Guerrón-Quintana (2021). We anticipate that advanced variational methods will enable further improvements beyond those offered by HMC. Second, the methods we propose are scalable with the dimension of estimation, which will simplify the process of estimating heterogeneous agent models (for instance, in the spirit of Kaplan et al., 2018). Third, there are many possible applications to general state-space models, including micro applications to panel and repeated cross-section data. We hope to see widespread usage of differentiable programming in economics in the next few years.

References

AMOS, B., I. D. J. RODRIGUEZ, J. SACKS, B. BOOTS, AND J. Z. KOLTER (2018): "Differentiable MPC for End-to-end Planning and Control," Tech. rep.

ANDREASEN, M. M., J. FERNÁNDEZ-VILLAVERDE, AND J. F. RUBIO-RAMÍREZ (2018): "The pruned state-space system for non-linear DSGE models: Theory and empirical applications," *Review of Economic Studies*, 85, 1–49.

ARELLANO, C. (2008): "Default risk and income fluctuations in emerging economies," *American Economic Review*, 98, 690–712.

BASTANI, H. AND L. GUERRIERI (2008): "On the Application of Automatic Differentiation to the Likelihood Function for Dynamic General Equilibrium Models," in *Advances in Automatic Differentiation*, ed. by T. J. Barth, M. Griebel, D. E. Keyes, R. M. Nieminen, D. Roose, and T. Schlick, Springer, vol. 64, 303–314.

BAYDIN, A. G., B. A. PEARLMUTTER, A. A. RADUL, AND J. M. SISKIND (2017): "Automatic Differentiation in Machine Learning: A Survey," *Journal of Machine Learning Research*, 18, 5595–5637.

BESKOS, A., N. PILLAI, G. ROBERTS, J.-M. SANZ-SERNA, AND A. STUART (2013): "Optimal tuning of the hybrid Monte Carlo algorithm," *Bernoulli*, 19, 1501 – 1534.

BETANCOURT, M. (2018): "A Conceptual Introduction to Hamiltonian Monte Carlo," Tech. rep.

BLONDEL, M., Q. BERTHET, M. CUTURI, R. FROSTIG, S. HOYER, F. LLINARES-LÓPEZ, F. PEDREGOSA, AND J.-P. VERT (2021): "Efficient and Modular Implicit Differentiation," Tech. rep.

CAO, D., W. LUO, AND G. NIE (2020): "Global DSGE models," Tech. rep.

CHEN, R. T., Y. RUBANOVA, J. BETTENCOURT, AND D. DUVENAUD (2018): "Neural ordinary differential equations," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 6572–6583.

CHEN, Y., R. DWIVEDI, M. J. WAINWRIGHT, AND B. YU (2020): "Fast mixing of Metropolized Hamiltonian Monte Carlo: Benefits of multi-step gradients," *Journal of Machine Learning Research*.

CORENFLOS, A., J. THORNTON, A. DOUCET, AND G. DELIGIANNIDIS (2021): "Differentiable Particle Filtering via Entropy-Regularized Optimal Transport," Tech. rep.

FARKAS, M. AND B. TATAR (2020): "Bayesian estimation of DSGE models with Hamiltonian Monte Carlo," Tech. rep., IMFS Working Paper Series.

FERNÁNDEZ-VILLAVERDE, J. AND P. A. GUERRÓN-QUINTANA (2021): "Estimating DSGE models: Recent advances and future challenges," *Annual Review of Economics*, 13.

FERNÁNDEZ-VILLAVERDE, J. AND J. F. RUBIO-RAMÍREZ (2007): "Estimating macroeconomic models: A likelihood approach," *Review of Economic Studies*, 74, 1059–1087.

FERNÁNDEZ-VILLAVERDE, J., J. F. RUBIO-RAMÍREZ, AND F. SCHORFHEIDE (2016): "Solution and estimation methods for DSGE models," in *Handbook of Macroeconomics*, Elsevier, vol. 2, 527–724.

GE, R., H. LEE, AND A. RISTESKI (2018): "Simulated Tempering Langevin Monte Carlo II: An Improved Proof using Soft Markov Chain Decomposition," Tech. rep.

GELMAN, A., J. B. CARLIN, H. S. STERN, D. B. DUNSON, A. VEHTARI, AND D. B. RUBIN (2013): *Bayesian Data Analysis*, Chapman and Hall/CRC.

GELMAN, A. AND D. B. RUBIN (1992): "Inference from Iterative Simulation Using Multiple Sequences," *Statistical Science*, 7, 457–472.

GOWDA, S., Y. MA, A. CHELI, M. GWÓŹZDŹ, V. B. SHAH, A. EDELMAN, AND C. RACKAUCKAS (2022): "High-Performance Symbolic-Numerics via Multiple Dispatch," *ACM Commun. Comput. Algebra*, 55, 92–96.

GRAHAM, M. M., A. H. THIERY, AND A. BESKOS (2022): "Manifold Markov chain Monte Carlo methods for Bayesian inference in diffusion models," Tech. rep.

GRIEWANK, A. AND A. WALTHER (2008): *Evaluating derivatives: principles and techniques of algorithmic differentiation*, SIAM.

HOFFMAN, M. D. AND A. GELMAN (2014): "The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research*, 15, 1593–1623.

INNES, M., A. EDELMAN, K. FISCHER, C. RACKAUCKAS, E. SABA, V. B. SHAH, AND W. TEBBUTT (2019): "A differentiable programming system to bridge machine learning and scientific computing," Tech. rep.

ISKREV, N. (2010): "Local identification in DSGE models," *Journal of Monetary Economics*, 57, 189–202.

KAPLAN, G., B. MOLL, AND G. L. VIOLANTE (2018): "Monetary policy according to HANK," *American Economic Review*, 108, 697–743.

KIM, S., N. SHEPHARD, AND S. CHIB (1998): "Stochastic Volatility: Likelihood Inference and Comparison with ARCH Models," *Review of Economic Studies*, 65, 361–393.

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

KUCUKELBIR, A., D. TRAN, R. RANGANATH, A. GELMAN, AND D. M. BLEI (2017): "Automatic differentiation variational inference," *Journal of Machine Learning Research*, 18, 1–45.

LEE, H., J. LU, AND Y. TAN (2022): "Convergence for score-based generative modeling with polynomial complexity," Tech. rep.

MALIAR, L., S. MALIAR, AND P. WINANT (2021): "Deep learning for solving dynamic economic models." *Journal of Monetary Economics*, 122, 76–101.

PITT, M. K., R. DOS SANTOS SILVA, P. GIORDANI, AND R. KOHN (2012): "On some properties of Markov chain Monte Carlo simulation methods based on the particle filter," *Journal of Econometrics*, 171, 134–151.

RACKAUCKAS, C. (2022): *Parallel Computing and Scientific Machine Learning (SciML): Methods and Applications*.

REBESCHINI, P. AND R. VAN HANDEL (2015): "Can local particle filters beat the curse of dimensionality?" *Annals of Applied Probability*, 25, 2809 – 2866.

SÄRKKÄ, S. (2013): *Bayesian Filtering and Smoothing*, 3, Cambridge University Press.

SCHMITT-GROHÉ, S. AND M. URIBE (2003): "Closing small open economy models," *Journal of International Economics*, 61, 163–185.

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

SMETS, F. AND R. WOUTERS (2007): "Shocks and frictions in US business cycles: A Bayesian DSGE approach," *American Economic Review*, 97, 586–606.

STAN DEVELOPMENT TEAM (2022): "Stan User's Guide, Version 2.30," Tech. rep.

WATSON, M. W. (1989): "Recursive solution methods for dynamic linear rational expectations models," *Journal of Econometrics*, 41, 65–89.

WHITE, L., M. ZGUBIC, M. ABBOTT, J. REVELS, A. ARSLAN, S. AXEN, S. SCHAUB, N. ROBINSON, Y. MA, G. DHINGRA, WILLTEBBUTT, N. HEIM, A. D. W. ROSEMBERG, D. WIDMANN, N. SCHMITZ, C. RACKAUCKAS, R. HEINTZMANN, FRANKSCHAE, K. FISCHER, A. ROBSON, MATTBRZEZINSKI, A. ZHABINSKI, M. BESANÇON, P. VERTECHI, S. GOWDA, A. FITZGIBBON, C. LUCIBELLO, C. VOGT, D. GANDHI, AND F. CHORNEY (2021): "JuliaDiff/ChainRules.jl: v1.14.0," Tech. rep.

Appendix A Perturbation Solution and Notation

In this section, we define the perturbation solution to the DSGE model, in both first- and second-order. The algebra is implemented in the `DifferentiableStateSpaceModels.jl` repository and is independent of downstream usage such as simulation or calculating likelihoods.

A.1 Tensor Notation

We use tensors for convenience when referring to high-dimensional objects. For Jacobian matrices, $\left[f_y\right]_\alpha^i = \frac{\partial f^i}{\partial y^\alpha}$ is the (i, α) (i-th row, j-th column) element of the derivative of f with respect to y. The dimension of the Jacobian matrix will be $m \times n$ where m is the dimension of f and n is the dimension of y. $i = 1...m$ and $\alpha = 1...n$. An example of tensor contraction notation is $\left[f_y\right]_\alpha^i \left[g_x\right]_j^\alpha = \sum_{\alpha=1}^n \frac{\partial f^i}{\partial y^\alpha} \frac{\partial g^\alpha}{\partial x^j}$.

For Hessian matrices, $\left[\mathcal{H}_{xy}\right]_{\alpha\gamma}^i$ is row i, column α, page γ of a 3-dimensional object. Denote m, n, k as the dimensions for $\mathcal{H}, x, y$ respectively, then $i = 1, \ldots, m$, $\alpha = 1, \ldots, n$, $\gamma = 1, \ldots, k$.

A.2 Definitions

Dimensions of related vectors and matrices:

- x: $n_x \times 1$
- y: $n_y \times 1$
- ϵ: $n_\epsilon \times 1$
- η: $n_x \times n_\epsilon$
- Σ: $n_\epsilon \times n_\epsilon$.

Denote $\bar{x}$, $\bar{y}$ as the DSS that satisfies: $\mathcal{H}\left(\bar{y}, \bar{y}, \bar{x}, \bar{x}\right) = 0$ when muting the shock processes.

Also:

- All vectors are in columns.
- n_x: the number of state variables; n_y: the number of control variables; n_θ: the number of model parameters; n_ϵ: the number of exogenous shocks.
- There are $n = n_x + n_y$ equations in the system. $\mathcal{H} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_y} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_x} \to \mathbb{R}^n$.

- $x' = h(x) + \eta\epsilon$: function h is the law of motion of the states; ϵ represents the i.i.d. exogenous shocks with a variance-covariance (VCV) matrix Σ; η represents the loadings of the shocks.

- $y = g(x)$: g is the policy function.

A.3 First-Order Solution

We need to solve for g_x and h_x. The dimensions of g_x and h_x are $n_y \times n_x$ and $n_x \times n_x$, respectively. A first-order Taylor expansion of (21) yields:

$$\mathcal{H}_{y'}y' + \mathcal{H}_y y + \mathcal{H}_{x'}x' + \mathcal{H}_x x = 0,$$

where $\mathcal{H}_{x;ij} = \frac{\partial \mathcal{H}_i}{\partial x_j}$, $i = 1, \ldots, n$, $j = 1, \ldots n_x$; $\mathcal{H}_{y;ij} = \frac{\partial \mathcal{H}_i}{\partial y_j}$, $i = 1, \ldots, n$, $j = 1, \ldots n_y$. All of these derivatives are evaluated at the DSS. Then:

$$\begin{bmatrix} \mathcal{H}_{x'} & \mathcal{H}_{y'} \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} \mathcal{H}_x & \mathcal{H}_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0. \tag{A.1}$$

Following Klein (2000), we apply the generalized Schur decomposition to matrices $\begin{bmatrix} \mathcal{H}_{x'} & \mathcal{H}_{y'} \end{bmatrix}$ and $\begin{bmatrix} \mathcal{H}_x & \mathcal{H}_y \end{bmatrix}$ from equation (A.1), and follow the Blanchard-Kahn condition to reorder so that $\forall i, \left|\frac{S_{22,ii}}{T_{22,ii}}\right| < 1$. Therefore:

$$\begin{pmatrix} S_{11} & S_{12} \\ 0 & S_{22} \end{pmatrix} \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix} \begin{pmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0. \tag{A.2}$$

The dimension of T_{22} or S_{22} should be $n_y \times n_y$ as in the Blanchard-Kahn condition to ensure that the converging solution exists and is unique. Hence:

$$g_x = -Z_{22}^{-1} Z_{21} \tag{A.3}$$

$$h_x = -\left(Z_{11} + Z_{12}g_x\right)^{-1} \left(S_{11}\right)^{-1} T_{11} \left(Z_{11} + Z_{12}g_x\right). \tag{A.4}$$

A.4 Second-Order Solution

We need to solve for g_{xx}, h_{xx}, $g_{\sigma\sigma}$, $h_{\sigma\sigma}$. The dimensions of g_{xx} and h_{xx} are $n_y \times n_x \times n_x$ and $n_x \times n_x \times n_x$. The dimensions of $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$ are $n_y \times 1$ and $n_x \times 1$.

Second-order perturbation of (21) yields:

$$\begin{aligned}
&\left(\left[\mathcal{H}_{y'y'}\right]^i_{\alpha\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k+\left[\mathcal{H}_{y'y}\right]^i_{\alpha\gamma}[g_x]^\gamma_k+\left[\mathcal{H}_{y'x'}\right]^i_{\alpha\delta}[h_x]^\delta_k+\left[\mathcal{H}_{y'x}\right]^i_{\alpha k}\right)[g_x]^\alpha_\beta[h_x]^\beta_j\\
&\qquad+\left[\mathcal{H}_{y'}\right]^i_\alpha[g_{xx}]^\alpha_{\beta\delta}[h_x]^\delta_k[h_x]^\beta_j+\left[\mathcal{H}_{y'}\right]^i_\alpha[g_x]^\alpha_\beta[h_{xx}]^\beta_{jk}\\
&+\left(\left[\mathcal{H}_{yy'}\right]^i_{\alpha\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k+\left[\mathcal{H}_{yy}\right]^i_{\alpha\gamma}[g_x]^\gamma_k+\left[\mathcal{H}_{yx'}\right]^i_{\alpha\delta}[h_x]^\delta_k+\left[\mathcal{H}_{yx}\right]^i_{\alpha k}\right)[g_x]^\alpha_j\\
&\qquad+\left[\mathcal{H}_{y}\right]^i_\alpha[g_{xx}]^\alpha_{jk}\\
&+\left(\left[\mathcal{H}_{x'y'}\right]^i_{\beta\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k+\left[\mathcal{H}_{x'y}\right]^i_{\beta\gamma}[g_x]^\gamma_k+[\mathcal{H}_{x'x'}]^i_{\beta\delta}[h_x]^\delta_k+[\mathcal{H}_{x'x}]^i_{\beta k}\right)[h_x]^\beta_j\\
&\qquad+[\mathcal{H}_{x'}]^i_\beta[h_{xx}]^\beta_{jk}\\
&\qquad+\left[\mathcal{H}_{xy'}\right]^i_{j\gamma}[g_x]^\gamma_\delta[h_x]^\delta_k+\left[\mathcal{H}_{xy}\right]^i_{j\gamma}[g_x]^\gamma_k+[\mathcal{H}_{xx'}]^i_{j\delta}[h_x]^\delta_k+[\mathcal{H}_{xx}]^i_{jk}=0,
\end{aligned}$$

for all combinations of i,j,k, where $i=1,\ldots,n$, $\alpha,\gamma=1,\ldots,n_y$, $\beta,\delta,j,k=1,\ldots,n_x$. If we vectorize the 2nd and 3rd dimension of g_{xx} and h_{xx} objects, and turn them into matrices (hence the dimensions are $n_y\times n_x^2$ and $n_x\times n_x^2$ respectively), then the above equation can be written as:

$$\begin{bmatrix}\mathcal{H}_{y'} & 0\end{bmatrix}\begin{bmatrix}g_{xx}\\ h_{xx}\end{bmatrix}h_x\otimes h_x+\begin{bmatrix}\mathcal{H}_y & \mathcal{H}_{y'}g_x+\mathcal{H}_{x'}\end{bmatrix}\begin{bmatrix}g_{xx}\\ h_{xx}\end{bmatrix}+C=0, \tag{A.5}$$

where C is a $(n_x+n_y)*(n_x^2)$ matrix. We compute each row i of C below,

$$C_{i,\cdot}=\text{vec}\left(\begin{bmatrix}g_xh_x\\ g_x\\ h_x\\ I\end{bmatrix}^\top\begin{bmatrix}\mathcal{H}_{y'y'} & \mathcal{H}_{y'y} & \mathcal{H}_{y'x'} & \mathcal{H}_{y'x}\\ \mathcal{H}_{yy'} & \mathcal{H}_{yy} & \mathcal{H}_{yx'} & \mathcal{H}_{yx}\\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'y} & \mathcal{H}_{x'x'} & \mathcal{H}_{x'x}\\ \mathcal{H}_{xy'} & \mathcal{H}_{xy} & \mathcal{H}_{xx'} & \mathcal{H}_{xx}\end{bmatrix}_i\begin{bmatrix}g_xh_x\\ g_x\\ h_x\\ I\end{bmatrix}\right). \tag{A.6}$$

Notice that the matrix in the middle of (A.6) is the Hessian of $\mathcal{H}_i$, the i-th equation of $\mathcal{H}$.

Expression A.5 is a Sylvester equation for $\begin{bmatrix}g_{xx}\\ h_{xx}\end{bmatrix}$. We can exploit the radius of h_x by using a doubling method.[20] Alternately, one can use off-the-shelf solvers for the generalized Sylvester equation such as `SLICOT`, the method we implement in practice.

[20] Rewrite the above equation as $AXF\otimes F+BX+C=0$ where

$$\begin{aligned}A&=\begin{bmatrix}\mathcal{H}_{y'} & 0\end{bmatrix}\\ F&=h_x\\ B&=\begin{bmatrix}\mathcal{H}_y & \mathcal{H}_{y'}g_x+\mathcal{H}_{x'}\end{bmatrix}\end{aligned}$$

After we solve g_{xx} and h_{xx}, we can write down the equations for $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$:

$$\begin{aligned}\left[\mathcal{H}_{y'}\right]^i_\alpha [g_x]^\alpha_\beta [h_{\sigma\sigma}]^\beta + \left[\mathcal{H}_{y'}\right]^i_\alpha [g_{\sigma\sigma}]^\alpha + \left[\mathcal{H}_y\right]^i_\alpha [g_{\sigma\sigma}]^\alpha + [\mathcal{H}_{x'}]^i_\beta [h_{\sigma\sigma}]^\beta \\ + \left[\mathcal{H}_{y'y'}\right]^i_{\alpha\gamma} [g_x]^\gamma_\delta [\eta]^\delta_\xi [g_x]^\alpha_\beta [\eta]^\beta_\phi [\Sigma]^\phi_\xi \\ + \left[\mathcal{H}_{y'x'}\right]^i_{\alpha\delta} [\eta]^\delta_\xi [g_x]^\alpha_\beta [\eta]^\beta_\phi [\Sigma]^\phi_\xi \\ + \left[\mathcal{H}_{y'}\right]^i_\alpha [g_{xx}]^\alpha_{\beta\delta} [\eta]^\delta_\xi [\eta]^\beta_\phi [\Sigma]^\phi_\xi \\ + \left[\mathcal{H}_{x'y'}\right]^i_{\beta\gamma} [g_x]^\gamma_\delta [\eta]^\delta_\xi [\eta]^\beta_\phi [\Sigma]^\phi_\xi \\ + [\mathcal{H}_{x'x'}]^i_{\beta\delta} [\eta]^\delta_\xi [\eta]^\beta_\phi [\Sigma]^\phi_\xi = 0,\end{aligned}$$

for all i, where $i = 1, \ldots, n$, $\alpha, \gamma = 1, \ldots, n_y$, $\beta, \delta = 1, \ldots, n_x$, $\phi, \xi = 1, \ldots, n_\epsilon$. This yields a linear equation for $\begin{bmatrix} g_{\sigma\sigma} \\ h_{\sigma\sigma} \end{bmatrix}$. We reorganize the above equation to get:

$$\begin{pmatrix} \mathcal{H}_{y'} + \mathcal{H}_y & \mathcal{H}_{y'} g_x + \mathcal{H}_{x'} \end{pmatrix} \begin{pmatrix} g_{\sigma\sigma} \\ h_{\sigma\sigma} \end{pmatrix} + B = 0. \tag{A.7}$$

Define function sum $(A) = \sum_{i,j} A_{ij}$, and the notation for element-wise product (Hadamard product) as $\odot$. Then, the vector B follows:

$$B^i = \text{sum}\left(\left[\begin{bmatrix} g_x \\ I \end{bmatrix}^\top \begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'x'} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'x'} \end{bmatrix}^i \begin{bmatrix} g_x \\ I \end{bmatrix} + \left[\mathcal{H}_{y'}\right]^i g_{xx} \right] \odot \left[\eta \Sigma \eta'\right] \right)$$

for all $i = 1, \ldots, n$. Here $\left[\mathcal{H}_{y'}\right]^i g_{xx}$ is a tensor shrink, that is to say, $\left\{ \left[\mathcal{H}_{y'}\right]^i g_{xx} \right\}_{\beta\delta} = \sum_{\alpha=1}^{n_y} \left[\mathcal{H}_{y'}\right]^i_\alpha [g_{xx}]^\alpha_{\beta\delta}$. To implement this we can first squeeze g_{xx} into an $n_y \times \left(n_x^2\right)$ matrix, and then reshape the matrix product $\left[\mathcal{H}_{y'}\right]^i g_{xx}$ into an $n_x \times n_x$ matrix.

Then, one can apply an iterative doubling method to solve out X for this Sylvester equation:

$$\begin{aligned} X_1 &= -B^{-1}C \\ X_{n+1} &= X_n + A_n X_n F_n \otimes F_n \\ A_{n+1} &= A_n A_n \\ F_{n+1} &= F_n F_n. \end{aligned}$$

Appendix B Perturbation Solution Derivatives

This section derives the derivatives of the DSS and solutions in first- and second-order with respect to the model parameters. Again, the algebra in this section is implemented in the `DifferentiableStateSpaceModels.jl` repository, and is independent of downstream usage such as simulation or calculating likelihoods.

B.1 Derivative of the DSS

The parameters θ in general will change the DSS values $\bar{x}, \bar{y}$. We are interested in $\frac{\partial \bar{x}}{\partial \theta}$ and $\frac{\partial \bar{y}}{\partial \theta}$. Let $F(x;\theta) \equiv \mathbb{E}_t \mathcal{H}\left(y', y, x', x; \theta\right)$. We take the derivatives of (21) with respect to θ:

$$F_\theta\left(x;\theta\right) = \mathcal{H}_x \frac{\partial x}{\partial \theta} + \mathcal{H}_y \frac{\partial y}{\partial \theta} + \mathcal{H}_{x'} \frac{\partial x'}{\partial \theta} + \mathcal{H}_{y'} \frac{\partial y'}{\partial \theta} + \mathcal{H}_\theta = 0,$$

where $\mathcal{H}_{\theta;ij} = \frac{\partial \mathcal{H}_i}{\partial \theta_j}$, $i = 1, \ldots, n$, $j = 1, \ldots n_\theta$. We evaluate this function at the DSS $x' = x = \bar{x}, y' = y = \bar{y}$,

$$\left[\begin{array}{cc} \mathcal{H}_y + \mathcal{H}_{y'} & \mathcal{H}_x + \mathcal{H}_{x'} \end{array}\right] \left[\begin{array}{c} \frac{\partial \bar{y}}{\partial \theta} \\ \frac{\partial \bar{x}}{\partial \theta} \end{array}\right] + \mathcal{H}_\theta = 0. \tag{B.1}$$

Expression B.1 is a linear equation system when we treat $\frac{\partial \bar{x}}{\partial \theta}$ and $\frac{\partial \bar{y}}{\partial \theta}$ as unknown variables.

B.2 First-Order Results

We are interested in $\frac{\partial g_x}{\partial \theta}$ and $\frac{\partial h_x}{\partial \theta}$, both evaluated at the DSS. From (21), the first-order model solution satisfies:

$$F_x\left(x;\theta\right) = \mathcal{H}_x + \mathcal{H}_y g_x + \mathcal{H}_{x'} h_x + \mathcal{H}_{y'} g_x h_x = 0. \tag{B.2}$$

For each element θ_i of θ, we take the derivative of equation (B.2):

$$\begin{aligned} F_{x,\theta_i}\left(x;\theta\right) = &\frac{\mathrm{d}\mathcal{H}_x}{\mathrm{d}\theta_i} + \frac{\mathrm{d}\mathcal{H}_y}{\mathrm{d}\theta_i} g_x + \mathcal{H}_y \frac{\partial g_x}{\partial \theta_i} + \frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta_i} h_x + \mathcal{H}_{x'} \frac{\partial h_x}{\partial \theta_i} \\ &+ \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} g_x h_x + \mathcal{H}_{y'} \frac{\partial g_x}{\partial \theta_i} h_x + \mathcal{H}_{y'} g_x \frac{\partial h_x}{\partial \theta_i} = 0, \end{aligned}$$

and we evaluate the system at the DSS $x' = x = \bar{x}$, $y' = y = \bar{y}$. Notice that $\mathcal{H}$ is

$\mathcal{H}\left(y', y, x', x; \theta\right)$, and the chain rule applies here as we compute the total derivative of $\mathcal{H}$:

$$
\begin{aligned}
\left[\frac{\mathrm{d}\mathcal{H}_x}{\mathrm{d}\theta_i}\right]_\beta^\alpha &= \left[\mathcal{H}_{xy'} + \mathcal{H}_{xy}\right]_{\beta\xi}^\alpha \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\mathcal{H}_{xx'} + \mathcal{H}_{xx}\right]_{\beta\delta}^\alpha \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta + \left[\frac{\partial \mathcal{H}_x}{\partial \theta_i}\right]_\beta^\alpha \\
\left[\frac{\mathrm{d}\mathcal{H}_y}{\mathrm{d}\theta_i}\right]_\gamma^\alpha &= \left[\mathcal{H}_{yy'} + \mathcal{H}_{yy}\right]_{\gamma\xi}^\alpha \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\mathcal{H}_{yx'} + \mathcal{H}_{yx}\right]_{\gamma\delta}^\alpha \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta + \left[\frac{\partial \mathcal{H}_y}{\partial \theta_i}\right]_\gamma^\alpha \\
\left[\frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta_i}\right]_\beta^\alpha &= \left[\mathcal{H}_{x'y'} + \mathcal{H}_{x'y}\right]_{\beta\xi}^\alpha \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\mathcal{H}_{x'x'} + \mathcal{H}_{x'x}\right]_{\beta\delta}^\alpha \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta + \left[\frac{\partial \mathcal{H}_{x'}}{\partial \theta_i}\right]_\beta^\alpha \\
\left[\frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i}\right]_\gamma^\alpha &= \left[\mathcal{H}_{y'y'} + \mathcal{H}_{y'y}\right]_{\gamma\xi}^\alpha \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\mathcal{H}_{y'x'} + \mathcal{H}_{y'x}\right]_{\gamma\delta}^\alpha \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta + \left[\frac{\partial \mathcal{H}_{y'}}{\partial \theta_i}\right]_\gamma^\alpha,
\end{aligned} \tag{B.3}
$$

where $\alpha = 1, \ldots, n$, $\gamma, \xi = 1, \ldots, n_y$, $\beta, \delta = 1, \ldots, n_x$. We can apply $\frac{\partial \bar{x}}{\partial \theta}$ and $\frac{\partial \bar{y}}{\partial \theta}$, the results from the last subsection here.

Thus, for each element θ_i of θ, we stack the unknowns as: $\begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ \frac{\partial h_x}{\partial \theta_i} \end{bmatrix}$ and solve a Sylvester equation:

$$
\begin{bmatrix} \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} \\ \frac{\mathrm{d}\mathcal{H}_y}{\mathrm{d}\theta_i} \\ \frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta_i} \\ \frac{\mathrm{d}\mathcal{H}_x}{\mathrm{d}\theta_i} \end{bmatrix}^\top \begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix} + \begin{bmatrix} \mathcal{H}_y & \mathcal{H}_{x'} + \mathcal{H}_{y'} g_x \end{bmatrix} \begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ \frac{\partial h_x}{\partial \theta_i} \end{bmatrix} + \begin{bmatrix} \mathcal{H}_{y'} & 0 \end{bmatrix} \begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ \frac{\partial h_x}{\partial \theta_i} \end{bmatrix} h_x = 0,
$$

B.3 Second-Order Results

We are interested in $\frac{\partial g_{xx}}{\partial \theta}, \frac{\partial h_{xx}}{\partial \theta}, \frac{\partial g_{\sigma\sigma}}{\partial \theta}, \frac{\partial h_{\sigma\sigma}}{\partial \theta}$, all evaluated at the DSS.

We differentiate the Sylvester equation (A.5) here, which yields a Sylvester equation itself. For each element θ_i of θ:

$$
\begin{aligned}
& AXB + DX + C = 0 \\
& \qquad \Downarrow \\
& A\frac{\partial X}{\partial \theta_i}B + D\frac{\partial X}{\partial \theta_i} + \left[\frac{\partial A}{\partial \theta_i}XB + AX\frac{\partial B}{\partial \theta_i} + \frac{\partial D}{\partial \theta_i}X + \frac{\partial C}{\partial \theta_i}\right] = 0.
\end{aligned} \tag{B.4}
$$

Equation (B.4) has a form similar to equation (A.5), and we can solve it with a similar algorithm. Since the coefficients A, B, D do not change across different θ_i, we solve them in parallel.

For simplicity of notation, we define Ψ as the Hessian of the original equation system

with respect to variable stacking $[y', y, x', x]$. Ψ_i is, for the i-th equation:

$$\Psi_i \equiv \begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'y} & \mathcal{H}_{y'x'} & \mathcal{H}_{y'x} \\ \mathcal{H}_{yy'} & \mathcal{H}_{yy} & \mathcal{H}_{yx'} & \mathcal{H}_{yx} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'y} & \mathcal{H}_{x'x'} & \mathcal{H}_{x'x} \\ \mathcal{H}_{xy'} & \mathcal{H}_{xy} & \mathcal{H}_{xx'} & \mathcal{H}_{xx} \end{bmatrix}_i . \tag{B.5}$$

We derive the variables above in equation (B.4) by applying chain rules:

$$\frac{\partial A}{\partial \theta_i} = \begin{bmatrix} \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} & 0 \end{bmatrix} \tag{B.6}$$

$$\frac{\partial B}{\partial \theta_i} = \frac{\partial h_x}{\partial \theta_i} \otimes h_x + h_x \otimes \frac{\partial h_x}{\partial \theta_i} \tag{B.7}$$

$$\frac{\partial D}{\partial \theta_i} = \begin{bmatrix} \frac{\mathrm{d}\mathcal{H}_y}{\mathrm{d}\theta_i} & \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} g_x + \mathcal{H}_{y'} \frac{\partial g_x}{\partial \theta_i} + \frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta_i} \end{bmatrix} \tag{B.8}$$

$$\begin{aligned} \frac{\partial C_{j,\cdot}}{\partial \theta_i} = \operatorname{vec} & \left(\begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} h_x + g_x \frac{\partial h_x}{\partial \theta_i} \\ \frac{\partial g_x}{\partial \theta_i} \\ \frac{\partial h_x}{\partial \theta_i} \\ 0 \end{bmatrix}^\top \Psi_j \begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix} \right) \\ + \operatorname{vec} & \left(\begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix}^\top \Psi_j \begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} h_x + g_x \frac{\partial h_x}{\partial \theta_i} \\ \frac{\partial g_x}{\partial \theta_i} \\ \frac{\partial h_x}{\partial \theta_i} \\ 0 \end{bmatrix} \right) \\ + \operatorname{vec} & \left(\begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix}^\top \frac{\mathrm{d}\Psi_j}{\mathrm{d}\theta_i} \begin{bmatrix} g_x h_x \\ g_x \\ h_x \\ I \end{bmatrix} \right), \end{aligned} \tag{B.9}$$

where $j = 1, \ldots, n$ iterating on each equation of $\mathcal{H}$.

We compute the derivatives with the form $\frac{\mathrm{d}[\mathcal{H}_{ab}]_j}{\mathrm{d}\theta_i}$ in the last line of equation (B.9) through a total derivative that requires third-order derivatives of the original equation system:

$$\frac{\mathrm{d}\left[\mathcal{H}_{ab}\right]_j}{\mathrm{d}\theta_i} = \left\{ \left[\mathcal{H}_{aby'} + \mathcal{H}_{aby}\right]_\xi \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\mathcal{H}_{abx'} + \mathcal{H}_{abx}\right]_\delta \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta \right\}_j + \frac{\partial \left[\mathcal{H}_{ab}\right]_j}{\partial \theta_i}, \tag{B.10}$$

where $a, b \in [y', y, x', x]$, $\xi = 1, \ldots, n_y$, $\delta = 1, \ldots, n_x$. The combination, therefore, can be expressed as:

$$\frac{\mathrm{d}\Psi_j}{\mathrm{d}\theta_i} = \left[\frac{\partial \Psi_j}{\partial y'} + \frac{\partial \Psi_j}{\partial y}\right]_\xi \left[\frac{\partial \bar{y}}{\partial \theta_i}\right]^\xi + \left[\frac{\partial \Psi_j}{\partial x'} + \frac{\partial \Psi_j}{\partial x}\right]_\delta \left[\frac{\partial \bar{x}}{\partial \theta_i}\right]^\delta + \frac{\partial \Psi_j}{\partial \theta_i}. \tag{B.11}$$

For the equations with $g_{\sigma\sigma}$ and $h_{\sigma\sigma}$, we know they satisfy equation (A.7):

$$A \begin{bmatrix} g_{\sigma\sigma} \\ h_{\sigma\sigma} \end{bmatrix} + B = 0. \tag{B.12}$$

Therefore, for each θ_i in θ, we take the derivative on both sides of the equation:

$$\frac{\partial A}{\partial \theta_i} \begin{bmatrix} g_{\sigma\sigma} \\ h_{\sigma\sigma} \end{bmatrix} + A \begin{bmatrix} \frac{\partial g_{\sigma\sigma}}{\partial \theta_i} \\ \frac{\partial h_{\sigma\sigma}}{\partial \theta_i} \end{bmatrix} + \frac{\partial B}{\partial \theta_i} = 0. \tag{B.13}$$

Equation (B.13) is a linear equation in $\begin{bmatrix} \frac{\partial g_{\sigma\sigma}}{\partial \theta_i} \\ \frac{\partial h_{\sigma\sigma}}{\partial \theta_i} \end{bmatrix}$. We enumerate $j = 1, \ldots, n$ for the index of the equations:

$$\frac{\partial A}{\partial \theta_i} = \begin{bmatrix} \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} + \frac{\mathrm{d}\mathcal{H}_y}{\mathrm{d}\theta_i} & \frac{\mathrm{d}\mathcal{H}_{y'}}{\mathrm{d}\theta_i} g_x + \mathcal{H}_{y'} \frac{\partial g_x}{\partial \theta_i} + \frac{\mathrm{d}\mathcal{H}_{x'}}{\mathrm{d}\theta_i} \end{bmatrix} \tag{B.14}$$

$$\begin{aligned}
\frac{\partial B^j}{\partial \theta_i} = & \operatorname{sum}\left(\left[\begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ 0 \end{bmatrix}^\top \begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'x'} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'x'} \end{bmatrix}^j \begin{bmatrix} g_x \\ I \end{bmatrix}\right] \odot \left[\eta \Sigma \eta'\right]\right) \\
& + \operatorname{sum}\left(\left[\begin{bmatrix} g_x \\ I \end{bmatrix}^\top \begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'x'} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'x'} \end{bmatrix}^j \begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ 0 \end{bmatrix}\right] \odot \left[\eta \Sigma \eta'\right]\right) \\
& + \operatorname{sum}\left(\left[\begin{bmatrix} g_x \\ I \end{bmatrix}^\top \frac{\mathrm{d}\begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'x'} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'x'} \end{bmatrix}^j}{\mathrm{d}\theta_i} \begin{bmatrix} g_x \\ I \end{bmatrix}\right] \odot \left[\eta \Sigma \eta'\right]\right) \\
& + \operatorname{sum}\left(\left[\frac{\mathrm{d}\left[\mathcal{H}_{y'}\right]^j}{\mathrm{d}\theta_i} g_{xx} + \left[\mathcal{H}_{y'}\right]^j \frac{\partial g_{xx}}{\partial \theta_i}\right] \odot \left[\eta \Sigma \eta'\right]\right) \\
& + \operatorname{sum}\left(\left[\begin{bmatrix} g_x \\ I \end{bmatrix}^\top \begin{bmatrix} \mathcal{H}_{y'y'} & \mathcal{H}_{y'x'} \\ \mathcal{H}_{x'y'} & \mathcal{H}_{x'x'} \end{bmatrix}^j \begin{bmatrix} g_x \\ I \end{bmatrix} + \left[\mathcal{H}_{y'}\right]^j g_{xx}\right] \odot \left[\eta \frac{\partial \Sigma}{\partial \theta_i} \eta'\right]\right),
\end{aligned} \tag{B.15}$$

or, for $\frac{\partial B^j}{\partial \theta_i}$, if we use the same matrix for $\mathcal{H}_{..}$ as in equation (B.9):

$$\frac{\partial B^j}{\partial \theta_i} = \text{sum}\left(\left(\left(\begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ 0 \\ 0 \\ 0 \end{bmatrix}^\top \Psi_j \begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix} + \begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix}^\top \Psi_j \begin{bmatrix} \frac{\partial g_x}{\partial \theta_i} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix}^\top \frac{\mathrm{d}\Psi_j}{\mathrm{d}\theta_i} \begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix} \right.\right.$$
$$\left.\left. + \frac{\mathrm{d}\left[\mathcal{H}_{y'}\right]^j}{\mathrm{d}\theta_i} g_{xx} + \left[\mathcal{H}_{y'}\right]^j \frac{\partial g_{xx}}{\partial \theta_i} \right) \odot \left[\eta \Sigma \eta'\right] \right)$$
$$+ \text{sum}\left(\left(\begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix}^\top \Psi_j \begin{bmatrix} g_x \\ 0 \\ I \\ 0 \end{bmatrix} + \left[\mathcal{H}_{y'}\right]^j g_{xx} \right) \odot \left[\eta \frac{\partial \Sigma}{\partial \theta_i} \eta'\right] \right). \tag{B.16}$$

Since Σ is a positive definite matrix, we can impose a Cholesky decomposition $\Sigma = \Gamma\Gamma'$, and in this case, $\frac{\partial \Sigma}{\partial \theta_i} = \frac{\partial \Gamma}{\partial \theta_i}\Gamma' + \Gamma\left(\frac{\partial \Gamma}{\partial \theta_i}\right)'$.

Here, for $\frac{\mathrm{d}\left[\mathcal{H}_{y'}\right]^j}{\mathrm{d}\theta_i} g_{xx}$ and $\left[\mathcal{H}_{y'}\right]^j \frac{\partial g_{xx}}{\partial \theta_i}$, we first squeeze the second and the third dimensions of the g_{xx} matrix, and then reshape the result coming from matrix multiplication.

Appendix C Sequential Solution and Derivatives

Given Appendices A and B, we now derive the $\{\hat{x}_t, \hat{y}_t\}_{t=1}^T$ and $\frac{d\hat{x}_t}{d\theta}$, $\frac{d\hat{y}_t}{d\theta}$, $\frac{d\hat{x}_t}{dx_0}$, $\frac{d\hat{y}_t}{dx_0}$, and $\frac{d\hat{x}_t}{d\epsilon}$, $\frac{d\hat{y}_t}{d\epsilon}$ for all $t = 1, \ldots T$. These are conditional on a particular ϵ, θ, and $\hat{x}_0$. As before, the algebra in this section is implemented in the `DifferenceEquations.jl` repository, and is not specific to perturbation solutions.

C.1 Solution

The system could be either linear or nonlinear. For nonlinear systems, we apply pruning (Andreasen et al., 2018), shutting down certain higher-order terms in the state transition equation to preserve the monotonicity of the likelihood in the shocks.

First-order (linear) terms come from:

$$\hat{y}_t = g_x \hat{x}_t \tag{C.1}$$
$$\hat{x}_{t+1} = h_x \hat{x}_t + \eta \epsilon_{t+1}. \tag{C.2}$$

Second-order (nonlinear) terms ($i = 1, \ldots, n_y$ and $j = 1, \ldots, n_x$) come from:

$$\hat{x}_{t+1}^f = h_x \hat{x}_t^f + \eta \epsilon_{t+1} \tag{C.3}$$

$$[\hat{y}_t]^i = [g_x \hat{x}_t]^i + \frac{1}{2} \left(\hat{x}_t^f\right)^\top [g_{xx}]^i \hat{x}_t^f + \frac{1}{2} [g_{\sigma\sigma}]^i \tag{C.4}$$

$$[\hat{x}_{t+1}]^j = [h_x \hat{x}_t]^j + \frac{1}{2} \left(\hat{x}_t^f\right)^\top [h_{xx}]^j \hat{x}_t^f + \frac{1}{2} [h_{\sigma\sigma}]^j + [\eta \epsilon_{t+1}]^j \tag{C.5}$$

$$\hat{x}_0^f = \hat{x}_0. \tag{C.6}$$

C.2 Derivatives

With Respect to Parameters. We derive $\frac{\partial \hat{x}_t}{\partial \theta_i}, \frac{\partial \hat{y}_t}{\partial \theta_i}$:

$$\begin{aligned}
\frac{\partial \hat{x}_t}{\partial \theta_i} &= \frac{\partial \left[h\left(\hat{x}_{t-1}; \theta\right) + \eta \epsilon_t\right]}{\partial \theta_i} \\
&= h_1\left(\hat{x}_{t-1}; \theta\right) \frac{\partial \hat{x}_{t-1}}{\partial \theta_i} + h_{2,i}\left(\hat{x}_{t-1}; \theta\right) \\
&= h_1\left(\hat{x}_{t-1}; \theta\right) h_1\left(\hat{x}_{t-2}; \theta\right) \frac{\partial \hat{x}_{t-2}}{\partial \theta_i} + h_1\left(\hat{x}_{t-1}; \theta\right) h_{2,i}\left(\hat{x}_{t-2}; \theta\right) + h_{2,i}\left(\hat{x}_{t-1}; \theta\right) \\
&\vdots \\
&= \sum_{j=0}^{t-1} \left[\prod_{k=j+1}^{t-1} h_1\left(\hat{x}_k; \theta\right)\right] h_{2,i}\left(\hat{x}_j; \theta\right)
\end{aligned} \tag{C.7}$$

$$\begin{aligned}
\frac{\partial \hat{y}_t}{\partial \theta_i} &= \frac{\partial \left[g\left(\hat{x}_t; \theta\right)\right]}{\partial \theta_i} \\
&= g_1\left(\hat{x}_t; \theta\right) \frac{\partial \hat{x}_t}{\partial \theta_i} + g_{2,i}\left(\hat{x}_t; \theta\right).
\end{aligned} \tag{C.8}$$

In the first-order case, we use the recursion for $\frac{\partial \hat{x}_t}{\partial \theta_i}, \frac{\partial \hat{y}_t}{\partial \theta_i}$:

$$\frac{\partial \hat{x}_{t+1}}{\partial \theta_i} = h_x \frac{\partial \hat{x}_t}{\partial \theta_i} + \frac{\partial h_x}{\partial \theta_i} \hat{x}_t \tag{C.9}$$

$$\frac{\partial \hat{y}_t}{\partial \theta_i} = g_x \frac{\partial \hat{x}_t}{\partial \theta_i} + \frac{\partial g_x}{\partial \theta_i} \hat{x}_t \tag{C.10}$$

$$\frac{\partial \hat{x}_0}{\partial \theta_i} = 0. \tag{C.11}$$

In the second-order case with pruning, we use the recursion:

$$\frac{\partial \hat{x}_{t+1}^f}{\partial \theta_i} = h_x \frac{\partial \hat{x}_t^f}{\partial \theta_i} + \frac{\partial h_x}{\partial \theta_i} \hat{x}_t^f \tag{C.12}$$

$$\frac{\partial \hat{x}_0^f}{\partial \theta_i} = 0 \tag{C.13}$$

$$\begin{aligned}\left[\frac{\partial \hat{x}_{t+1}}{\partial \theta_i}\right]^k = & \left[h_x \frac{\partial \hat{x}_t}{\partial \theta_i}\right]^k + \left[\frac{\partial h_x}{\partial \theta_i} \hat{x}_t\right]^k \\ & + \frac{1}{2}\left(\frac{\partial \hat{x}_t^f}{\partial \theta_i}\right)^\top [h_{xx}]^k \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top \left[\frac{\partial h_{xx}}{\partial \theta_i}\right]^k \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top [h_{xx}]^k \frac{\partial \hat{x}_t^f}{\partial \theta_i} + \frac{1}{2}\left[\frac{\partial h_{\sigma\sigma}}{\partial \theta_i}\right]^k\end{aligned} \tag{C.14}$$

$$\begin{aligned}\left[\frac{\partial \hat{y}_t}{\partial \theta_i}\right]^j = & \left[g_x \frac{\partial \hat{x}_t}{\partial \theta_i}\right]^j + \left[\frac{\partial g_x}{\partial \theta_i} \hat{x}_t\right]^j \\ & + \frac{1}{2}\left(\frac{\partial \hat{x}_t^f}{\partial \theta_i}\right)^\top [g_{xx}]^j \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top \left[\frac{\partial g_{xx}}{\partial \theta_i}\right]^j \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top [g_{xx}]^j \frac{\partial \hat{x}_t^f}{\partial \theta_i} + \frac{1}{2}\left[\frac{\partial g_{\sigma\sigma}}{\partial \theta_i}\right]^j\end{aligned} \tag{C.15}$$

$$\left[\frac{\partial \hat{x}_0}{\partial \theta_i}\right]^k = 0, \tag{C.16}$$

where $k = 1, \ldots, n_x$, $j = 1, \ldots, n_y$.

With Respect to Shocks. From the chain rule, we have:

$$\begin{aligned}\frac{\partial \hat{x}_t}{\partial \epsilon_i} &= \frac{\partial h\left(\hat{x}_{t-1}\right)}{\partial \hat{x}_{t-1}} \frac{\partial \hat{x}_{t-1}}{\partial \epsilon_i} \\ &= \left(\prod_{j=i}^{t-1} \frac{\partial h\left(\hat{x}_j\right)}{\partial \hat{x}_j}\right) \eta\end{aligned} \tag{C.17}$$

$$\begin{aligned}\frac{\partial \hat{y}_t}{\partial \epsilon_i} &= \frac{\partial g\left(\hat{x}_t\right)}{\partial \hat{x}_t} \frac{\partial \hat{x}_t}{\partial \epsilon_i} \\ &= \frac{\partial g\left(\hat{x}_t\right)}{\partial \hat{x}_t} \left(\prod_{j=i}^{t-1} \frac{\partial h\left(\hat{x}_j\right)}{\partial \hat{x}_j}\right) \eta.\end{aligned} \tag{C.18}$$

In the first-order case, we use the recursions for $\frac{\partial \hat{x}_t}{\partial \epsilon_i}$ and $\frac{\partial \hat{y}_t}{\partial \epsilon_i}$ where $t \geqslant i$,

$$\frac{\partial \hat{x}_{t+1}}{\partial \epsilon_i} = h_x \frac{\partial \hat{x}_t}{\partial \epsilon_i} \tag{C.19}$$

$$\frac{\partial \hat{y}_t}{\partial \epsilon_i} = g_x \frac{\partial \hat{x}_t}{\partial \epsilon_i} \tag{C.20}$$

$$\frac{\partial \hat{x}_i}{\partial \epsilon_i} = \eta. \tag{C.21}$$

For the second-order pruning case, we use the recursions for $\frac{\partial \hat{x}_t}{\partial \epsilon_i}$ and $\frac{\partial \hat{y}_t}{\partial \epsilon_i}$ where $t \geqslant i$,

$$\frac{\partial \hat{x}^f_{t+1}}{\partial \epsilon_i} = h_x \frac{\partial \hat{x}^f_t}{\partial \epsilon_i} \tag{C.22}$$

$$\left[\frac{\partial \hat{y}_t}{\partial \epsilon_i}\right]^j = \left[g_x \frac{\partial \hat{x}_t}{\partial \epsilon_i}\right]^j + \frac{1}{2}\left(\frac{\partial \hat{x}^f_t}{\partial \epsilon_i}\right)^\top [g_{xx}]^j \hat{x}^f_t + \frac{1}{2}\left(\hat{x}^f_t\right)^\top [g_{xx}]^j \frac{\partial \hat{x}^f_t}{\partial \epsilon_i} \tag{C.23}$$

$$\left[\frac{\partial \hat{x}_{t+1}}{\partial \epsilon_i}\right]^k = \left[h_x \frac{\partial \hat{x}_t}{\partial \epsilon_i}\right]^k + \frac{1}{2}\left(\frac{\partial \hat{x}^f_t}{\partial \epsilon_i}\right)^\top [h_{xx}]^k \hat{x}^f_t + \frac{1}{2}\left(\hat{x}^f_t\right)^\top [h_{xx}]^k \frac{\partial \hat{x}^f_t}{\partial \epsilon_i} \tag{C.24}$$

$$\frac{\partial \hat{x}^f_i}{\partial \epsilon_i} = \eta \tag{C.25}$$

$$\frac{\partial \hat{x}_i}{\partial \epsilon_i} = \eta. \tag{C.26}$$

With respect to initial conditions. By applying the chain rule, we have:

$$\frac{\partial \hat{x}_t}{\partial \hat{x}_0} = \frac{\partial \left[h\left(\hat{x}_{t-1}; \theta\right) + \eta \epsilon_t\right]}{\partial \hat{x}_0} = \frac{\partial h\left(\hat{x}_{t-1}; \theta\right)}{\partial \hat{x}_{t-1}} \frac{\partial \hat{x}_{t-1}}{\partial \hat{x}_0} \tag{C.27}$$

$$\frac{\partial \hat{y}_t}{\partial \hat{x}_0} = \frac{\partial g\left(\hat{x}_t; \theta\right)}{\partial \hat{x}_0} = \frac{\partial g\left(\hat{x}_t; \theta\right)}{\partial \hat{x}_t} \frac{\partial \hat{x}_t}{\partial \hat{x}_0}, \tag{C.28}$$

where for the first-order case:

$$\frac{\partial \hat{y}_t}{\partial \hat{x}_0} = g_x \frac{\partial \hat{x}_t}{\partial \hat{x}_0} \tag{C.29}$$

$$\frac{\partial \hat{x}_{t+1}}{\partial \hat{x}_0} = h_x \frac{\partial \hat{x}_t}{\partial \hat{x}_0} \tag{C.30}$$

$$\frac{\partial \hat{x}_0}{\partial \hat{x}_0} = I, \tag{C.31}$$

and for the second-order case:

$$\frac{\partial \hat{x}_{t+1}^f}{\partial \hat{x}_0} = h_x \frac{\partial \hat{x}_t^f}{\partial \hat{x}_0} \tag{C.32}$$

$$\frac{\partial \hat{x}_0^f}{\partial \hat{x}_0} = I \tag{C.33}$$

$$\left[\frac{\partial \hat{x}_{t+1}}{\partial \hat{x}_0}\right]^k = \left[h_x \frac{\partial \hat{x}_t}{\partial \hat{x}_0}\right]^k + \frac{1}{2}\left(\frac{\partial \hat{x}_t^f}{\partial \hat{x}_0}\right)^\top [h_{xx}]^k \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top [h_{xx}]^k \frac{\partial \hat{x}_t^f}{\partial \hat{x}_0} \tag{C.34}$$

$$\left[\frac{\partial \hat{y}_t}{\partial \hat{x}_0}\right]^j = \left[g_x \frac{\partial \hat{x}_t}{\partial \hat{x}_0}\right]^j + \frac{1}{2}\left(\frac{\partial \hat{x}_t^f}{\partial \hat{x}_0}\right)^\top [g_{xx}]^j \hat{x}_t^f + \frac{1}{2}\left(\hat{x}_t^f\right)^\top [g_{xx}]^j \frac{\partial \hat{x}_t^f}{\partial \hat{x}_0} \tag{C.35}$$

$$\frac{\partial \hat{x}_0}{\partial \hat{x}_0} = I. \tag{C.36}$$

Appendix D The Kalman Filter and Its Derivatives

For linear-Gaussian state-space models, we can evaluate the series of posterior distributions of the latent state and the marginal likelihood with the Kalman filter. We compute the derivatives associated with this whole process as well. As before, the algebra in this section is implemented in the `DifferenceEquations.jl` repository, and is not specific to perturbation solutions.

The Kalman Filter. To run the Kalman filter, in addition to the initial condition $\hat{x}_0$, we need its prior covariance matrix P_0. A natural choice is the solution to the Lyapunov equation:

$$h_x P_0 h_x' - P_0 + \eta \Sigma \eta' = 0. \tag{D.1}$$

For simplicity of notation, we define: $G = Q \begin{bmatrix} g_x \\ I \end{bmatrix}$. Let the constant loadings of steady state values be H. Let $\bar{u} = \begin{bmatrix} \bar{y} \\ \bar{x} \end{bmatrix}$. Then the formula to update with period t data

follows the recursion:

$$x_{t|t-1} = h_x x_{t-1} \tag{D.2}$$
$$P_{t|t-1} = h_x P_{t-1} h_x' + \eta \Sigma \eta' \tag{D.3}$$
$$z_t = G x_{t|t-1} + H\bar{u} \tag{D.4}$$
$$V_t = G P_{t|t-1} G' + \Omega \tag{D.5}$$
$$z_t \sim \mathcal{N}(z_t, V_t) \tag{D.6}$$
$$x_t = x_{t|t-1} + P_{t|t-1} G' V_t^{-1} (z_t - z_t) \tag{D.7}$$
$$P_t = P_{t|t-1} - P_{t|t-1}' G' V_t^{-1} G P_{t|t-1}. \tag{D.8}$$

Derivatives. By differentiating the Lyapunov equation (D.1) above, we get another Lyapunov equation and $\frac{\partial P_0}{\partial \theta_i}$ is the root of the equation:

$$h_x \frac{\partial P_0}{\partial \theta_i} h_x' - \frac{\partial P_0}{\partial \theta_i} + \left(\frac{\partial h_x}{\partial \theta_i} P_0 h_x' + h_x P_0 \frac{\partial h_x'}{\partial \theta_i} + \eta \frac{\partial \Sigma}{\partial \theta_i} \eta' \right) = 0. \tag{D.9}$$

We also track the derivatives of each of the quantities above with respect to the parameters that we are interested in:

$$\frac{\partial x_{t|t-1}}{\partial \theta_i} = \frac{\partial h_x}{\partial \theta_i} x_{t-1} + h_x \frac{\partial x_{t-1}}{\partial \theta_i} \tag{D.10}$$
$$\frac{\partial P_{t|t-1}}{\partial \theta_i} = \frac{\partial h_x}{\partial \theta_i} P_{t-1} h_x' + h_x \frac{\partial P_{t-1}}{\partial \theta_i} h_x + h_x P_{t-1} \left(\frac{\partial h_x}{\partial \theta_i} \right)^\top + \eta \frac{\partial \Sigma}{\partial \theta_i} \eta' \tag{D.11}$$
$$\frac{\partial z_t}{\partial \theta_i} = \frac{\partial G}{\partial \theta_i} x_{t|t-1} + G \frac{\partial x_{t|t-1}}{\partial \theta_i} + H \frac{\partial \bar{u}}{\partial \theta_i} \tag{D.12}$$
$$\frac{\partial V_t}{\partial \theta_i} = \frac{\partial G}{\partial \theta_i} P_{t|t-1} G' + G \frac{\partial P_{t|t-1}}{\partial \theta_i} G' + G P_{t|t-1} \left(\frac{\partial G}{\partial \theta_i} \right)^\top \tag{D.13}$$
$$\begin{aligned} \tilde{z}_t &\sim \mathcal{N}(z_t, V_t) \\ \frac{\partial x_t}{\partial \theta_i} &= \frac{\partial x_{t|t-1}}{\partial \theta_i} + \frac{\partial P_{t|t-1}}{\partial \theta_i} G' V_t^{-1} (\tilde{z}_t - z_t) + P_{t|t-1} \left(\frac{\partial G}{\partial \theta_i} \right)^\top V_t^{-1} (\tilde{z}_t - z_t) \\ &\quad - P_{t|t-1} G' V_t^{-1} \frac{\partial V_t}{\partial \theta_i} V_t^{-1} (\tilde{z}_t - z_t) - P_{t|t-1} G' V_t^{-1} \frac{\partial z_t}{\partial \theta_i} \end{aligned} \tag{D.14}$$
$$\begin{aligned} \frac{\partial P_t}{\partial \theta_i} &= \frac{\partial P_{t|t-1}}{\partial \theta_i} - \frac{\partial P_{t|t-1}}{\partial \theta_i}' G' V_t^{-1} G P_{t|t-1} - P_{t|t-1}' \left(\frac{\partial G}{\partial \theta_i} \right)^\top V_t^{-1} G P_{t|t-1} \\ &\quad + P_{t|t-1}' G' V_t^{-1} \frac{\partial V_t}{\partial \theta_i} V_t^{-1} G P_{t|t-1} - P_{t|t-1}' G' V_t^{-1} \frac{\partial G}{\partial \theta_i} P_{t|t-1} \\ &\quad - P_{t|t-1}' G' V_t^{-1} G \frac{\partial P_{t|t-1}}{\partial \theta_i}. \end{aligned} \tag{D.15}$$

Appendix E Reverse-Mode AD

AD enables the fast computation of gradients via the chain rule for functions that are composed of primitive functions for which gradients are already available. While existing AD systems generally maintain a large library of such primitives, for more advanced numerical tasks one may need to implement a new primitive, as we do for perturbative DSGE solutions.

To explain the ingredients necessary to perform such a task in a reverse-mode system, we give a brief overview of reverse-mode AD with examples of custom implementations.[21] For more in-depth coverage, see Griewank and Walther (2008), as well as the relevant sections of documentation for `ChainRules.jl`, `PyTorch`, and `JAX`.

Reverse vs. Forward Mode. Consider a composite function $f \circ g : \mathbb{R}^n \to \mathbb{R}$ defined by composing $g : \mathbb{R}^n \to \mathbb{R}^m$ and $f : \mathbb{R}^m \to \mathbb{R}$ by steps $y = g(x)$, $a = f(y)$. By the chain rule, $\frac{da}{dx} = \frac{da}{dy} \cdot \frac{dy}{dx}$. Reverse-mode AD computes this derivative by first evaluating the component functions in the order from input to output, then evaluating their derivatives in order from output to input. The key ingredient needed for each component function is an *adjoint rule*, also called a vector-Jacobian product (vJp) or pullback. It takes as input a sensitivity (or cotangent) representing the gradient of an output function with respect to an input. Letting $\bar{y} = \frac{da}{dy} \in \mathbb{R}^{1\times m}$ be the sensitivity returned by the adjoint rule of f, the chain rule gives the result that the sensitivity returned by the adjoint rule for g must be:

$$\bar{x} := \frac{da}{dx} = \bar{y} \cdot \frac{dy}{dx} \in \mathbb{R}^{1\times n}. \tag{E.1}$$

The map that produces this result is the product of the sensitivity vector and the Jacobian of g, hence the name vector-Jacobian product. Treating the space of sensitivities $\bar{x}$ (the "cotangent space") as R^m, the pullback is computed by applying the transpose (or *adjoint*, hence the other name) of the $m \times n$ Jacobian matrix to produce a sensitivity in $\mathbb{R}^n$.

To illustrate, let $f(y) = \frac{1}{2}y'Cy = \sum_{i=1}^m \frac{c_i}{2} y_i^2$ for $C \in \mathbb{R}^{m\times m}$ be a diagonal matrix with entries c_i along the diagonal, and $g(x) = Mx$ for $M \in \mathbb{R}^{m\times n}$ and consider evaluating $f \circ g(x) := \frac{1}{2}x'M'CMx$ at $\tilde{x}$. The Jacobians of these functions are equal to $\frac{dg}{dx} = M$ and $\frac{df}{dy} = y'C$, so by the chain rule,

$$\frac{da}{dx} = \frac{df}{dy} \circ \frac{dg}{dx} = (Mx)'CM = \sum_{i=1}^m c_i (Mx)_i M_{i\cdot}. \tag{E.2}$$

Forward mode would implement this expression by first evaluating g and its derivative,

[21] Implementation of new function $f(x)$ in a forward-mode system is generally less complex and can coincide with writing code to evaluate $\nabla f(x)$ as a function of its inputs x.

then passing to f and its derivative. To implement this computation in reverse mode, starting with the forward pass, execute the steps in order and store the intermediates:

$$\tilde{y} = M\tilde{x}$$
$$\tilde{a} = \sum_{i=1}^{m} \frac{c_i}{2} \tilde{y}_i^2.$$

Next, apply the adjoint rules. The rule for $f(\tilde{y})$ takes in a sensitivity $\bar{a} \in \mathbb{R} = 1$ (since it is the final step) and outputs a sensitivity $\bar{y} \in \mathbb{R}^{1\times m}$ with i^{th} entry $c_i\tilde{y}_i$. The adjoint rule for $g(\tilde{x})$ takes in a sensitivity $\bar{y}$ and outputs a sensitivity $\bar{x} \in \mathbb{R}^{1\times n}$ equal to $\bar{y}M$, which is the gradient, and recognizing that $\tilde{y} = M\tilde{x}$, identical to the result returned by equation (E.2).

Advantages of Reverse-Mode AD. To see why reverse mode may yield performance improvements, consider composing the above operations with another function $x = h(z) := Az$ for $A \in \mathbb{R}^{n\times n}$, with the goal of evaluating the gradient of $f \circ g \circ h(z) = f(M \cdot Az)$ with respect to z. Function h has adjoint rule $\bar{z} = \bar{x}A$, which is a vector-matrix multiply, with computational cost $O(n^2)$, which adds to the $O(mn)$ cost of the adjoint rule for g and the $O(m)$ cost of the adjoint rule for f. Had differentiation instead proceeded from input to output, as in forward mode, it would have been necessary to compute the Jacobian of $g \circ h$, which requires the matrix-matrix multiply $M \cdot A$, an $O(mn^2)$ operation. As a result, for this example, reverse mode achieves a speedup roughly by a factor of n over the forward mode. This result illustrates the general principle that reverse mode dominates for single-output functions with high-dimensional inputs (e.g., a likelihood over many observables).

The choice of primitives here was purely illustrative, and each component function could have been decomposed further. For example, the matrix-vector multiply Mx could have been represented as a sum of products of scalars $\sum_j M_{i,j}\, x_j$ for $i = 1 \ldots m$. The choice to represent at the matrix level ensures that in the reverse pass, the computation can also be represented as a matrix operation (a vector-matrix multiply), which may be faster due to linear algebra subroutines. In this and other cases, rather than a single chain of function compositions, the final result may take the form of a directed computational graph, with functions as nodes passing the output to other functions along edges. As some outputs may be connected to multiple inputs or vice versa, the efficiency of primal and derivative evaluation may depend on the order of operations along the graph. Without any reuse, the gradient may contain a number of operations exponential in the number of nodes, while the output-to-input traversal provided by reverse mode incurs a cost that is linear, an exponential speedup (Griewank and Walther, 2008).[22]

[22]While reverse mode is generically applicable and possesses strong guarantees, it need not be strictly

Implicit Functions. Further improvements to reverse mode may be obtained by expanding the set of primitive functions to group computationally linked operations for which one can write a custom gradient rule. Implicit functions provide an archetypal example. Let $g(x)$ be implicitly defined as the solution to $f(x, y) = 0$, supposing that the hypotheses of the implicit function theorem are satisfied so that it exists and is differentiable. To compute the forward pass, one might apply a nonlinear function solver, such as a (quasi-)Newton method, which, if $y \in \mathbb{R}^n$, may take k iterations at cost $O(n^3)$ each to compute y. Applying reverse mode to each step sequentially results in an adjoint computation that will likewise have cost $O(kn^3)$. However, the implicit function theorem states that when $\frac{\mathrm{d}f}{\mathrm{d}y}$ is full rank at the solution (x, y^*), the derivative is given by $\frac{\mathrm{d}y}{\mathrm{d}x} = -(\frac{\mathrm{d}f(x,y^*)}{\mathrm{d}y})^{-1}\frac{\mathrm{d}f(x,y^*)}{\mathrm{d}x}$. The corresponding adjoint rule is a function that takes in a sensitivity $\bar{y} \in \mathbb{R}^{1\times n}$ and outputs a sensitivity $\bar{x} \in \mathbb{R}^{1\times k}$:

$$\bar{x} := -\bar{y}(\frac{\mathrm{d}f(x, y^*)}{\mathrm{d}y})^{-1}\frac{\mathrm{d}f(x, y^*)}{\mathrm{d}x}.$$

As the components of this derivative are available from the forward pass (often themselves computed by forward-mode AD), this can be computed in one linear system solve, at cost $O(n^3)$, saving a factor of k relative to unrolling.

Similar principles have been applied to generate custom adjoint rules that enable use within AD systems of many other high-level operations including optimization (Blondel et al., 2021), differential equations (Chen et al., 2018), optimal control (Amos et al., 2018), and a variety of other applications. We directly apply the above implicit function rule when solving for the derivative of a steady state, and we derive new rules for several other steps in the rational expectations solution algorithms. This illustrates the motivation of scientific ML (Rackauckas, 2022) that while reverse-mode systems are highly effective and scalable in large data applications, there is substantial flexibility in terms of choices of algorithms to implement rules for gradients that can be critical to implementation in particular scientific computing settings.

Appendix F Reverse-Mode AD Derivation

In this section, we derive the reverse-mode AD rules for both the sequential solution and the Kalman filter likelihood. While the Julia package `Zygote.jl` can handle the calculation automatically, we implement the actual algebra for performance improvement. The algebra in this section is implemented in the `DifferenceEquations.jl` repository and is

optimal. Optimization of computational graph traversal based on function structure can sometimes reduce the operation count even further. However, computing such optimization is computationally hard, so reverse mode, possibly augmented with custom gradient rules, remains dominant in ML.

not specific to perturbation solutions.

For notation simplicity, we denote the log-posterior evaluated at each time period as lp_t. Eventually, we take the adjoint of the log-posterior in the last period $\bar{\text{lp}}_T$ as the input.

First-order System. In the first-order system, the evolution of x_t and y_t and the calculation of log-posterior follow:

$$\begin{aligned} x_{t+1} &= Ax_t + B\epsilon_t \\ y_t &= Cx_t \\ \text{lp}_t &= \text{lp}_{t-1} + L\left(y_t\right) \end{aligned}$$

Thus, we reverse the iteration in t to back out the adjoints. For $t = T, T-1, \ldots, 2, 1$,

$$\begin{aligned} \bar{y}_t &= \bar{\text{lp}}_T \cdot L'\left(y_t\right) \\ \bar{x}_t &= \bar{x}_t + C^\top \bar{y}_t \\ \bar{x}_{t-1} &= A^\top \bar{x}_t \\ \bar{\epsilon}_t &= B^\top \bar{x}_t \\ \bar{C} &= \bar{C} + \bar{y}_t x_t^\top \\ \bar{A} &= \bar{A} + \bar{x}_t x_{t-1}^\top \\ \bar{B} &= \bar{B} + \bar{x}_t \epsilon_t^\top \end{aligned}$$

and we will return $\bar{A}, \bar{B}, \bar{C}, \bar{x}_0, \{\bar{\epsilon}_t\}$.

Second-order System. In the pruned second-order system, the evolution of x_t and y_t and the calculation of log-posterior follow:

$$\begin{aligned} x_{t+1}^f &= A_1 x_t^f + B\epsilon_{t+1} \\ \left[y_t\right]^i &= \left[C_0\right]^i + \left[C_1 x_t\right]^i + \frac{1}{2}\left(x_t^f\right)^\top \left[C_2\right]^i x_t^f \\ \left[x_{t+1}\right]^j &= \left[A_0\right]^j + \left[A_1 x_t\right]^j + \frac{1}{2}\left(x_t^f\right)^\top \left[A_2\right]^j x_t^f + \left[B\epsilon_{t+1}\right]^j \\ \text{lp}_t &= \text{lp}_{t-1} + L\left(y_t\right). \end{aligned}$$

Therefore, we reverse the iteration in t to back out the adjoints. For $t = T, T-$

$1, \ldots, 2, 1,$

$$
\begin{aligned}
\bar{y}_t &= \bar{\mathrm{lp}}_T \cdot L'(y_t) \\
\bar{x}_t &= \bar{x}_t + C_1^\top \bar{y}_t \\
\bar{x}_t^f &= \bar{x}_t^f + \sum_i \left(C_{2,i} + C_{2,i}^\top \right) x_t^f \bar{y}_{t,i} \\
\bar{x}_{t-1} &= A_1^\top \bar{x}_t \\
\bar{x}_{t-1}^f &= A_1^\top \bar{x}_t^f + \sum_i \left(A_{2,i} + A_{2,i}^\top \right) x_t^f \bar{x}_{t,i} \\
\bar{\epsilon}_t &= B^\top \left(\bar{x}_t + \bar{x}_t^f \right) \\
\bar{C_0} &= \bar{C_0} + \bar{y}_t \\
\bar{C_1} &= \bar{C_1} + \bar{y}_t x_t^\top \\
\bar{C_{2,i}} &= \bar{C_{2,i}} + x_t^f \left(x_t^f \right)^\top \bar{y}_{t,i} \\
\bar{A_0} &= \bar{A_0} + \bar{x}_t \\
\bar{A_1} &= \bar{A_1} + \bar{x}_t x_{t-1}^\top + \bar{x}_t^f \left(x_{t-1}^f \right)^\top \\
\bar{A_{2,i}} &= \bar{A_{2,i}} + x_t^f \left(x_t^f \right)^\top \bar{x}_{t,i} \\
\bar{B} &= \bar{B} + \left(\bar{x}_t + \bar{x}_t^f \right) \epsilon_t^\top .
\end{aligned}
$$

Kalman Filter. With initial x_0 and P_0, for $t = 1, \ldots, T$, the Kalman filter evolves as:

$$
\begin{aligned}
x_{t|t-1} &= A x_{t-1} \\
P_{t|t-1} &= A P_{t-1} A' + BB' \\
z_t &= C x_{t|t-1} \\
V_t &= C P_{t|t-1} C' + \Omega \\
\tilde{z}_t &\sim \mathcal{N}(z_t, V_t) \\
b &= C P_{t|t-1} \\
K &= b^\top V_t^{-1} \\
x_t &= x_{t|t-1} + K(\tilde{z}_t - z_t) \\
P_t &= P_{t|t-1} - Kb.
\end{aligned}
$$

Therefore for $t = T, \ldots, 1$. We initialize $\bar{P}_t = \mathbf{0}$ and $\bar{x}_t = \mathbf{0}$.

$$
\begin{aligned}
\bar{P}_{t|t-1} &= \bar{P}_t \\
\bar{K} &= -\bar{P}_t b^\top \\
\bar{b} &= -K^\top \bar{P}_t \\
\bar{x}_{t|t-1} &= \bar{x}_t \\
\bar{K} &= \bar{K} + \bar{x}_t \left(\tilde{z}_t - z_t\right)^\top \\
\bar{z}_t &= -K^\top \bar{x}_t \\
\bar{b} &= \bar{b} + V_t^{-1} \bar{K}^\top \\
\bar{V}_t &= -V_t^{-\top} b \bar{K} V_t^{-\top} \\
\bar{C} &= \bar{C} + \bar{b} P_{t|t-1}^\top \\
\bar{P}_{t|t-1} &= \bar{P}_{t|t-1} + C^\top \bar{b} \\
\bar{z}_t &= \bar{z}_t + \bar{\mathrm{lp}} \cdot L_z \left(\tilde{z}_t - z_t, V_t\right) \\
\bar{V}_t &= \bar{V}_t + \bar{\mathrm{lp}} \cdot L_V \left(\tilde{z}_t - z_t, V_t\right) \\
\bar{C} &= \bar{C} + \bar{V}_t C P_{t|t-1}^\top + \bar{V}_t^\top C P_{t|t-1} \\
\bar{P}_{t|t-1} &= \bar{P}_{t|t-1} + C^\top \bar{V}_t C \\
\bar{C} &= \bar{C} + \bar{z}_t x_{t|t-1}^\top \\
\bar{x}_{t|t-1} &= \bar{x}_{t|t-1} + C^\top \bar{z}_t \\
\bar{A} &= \bar{A} + \bar{P}_{t|t-1} A P_{t-1}^\top + \bar{P}_{t|t-1}^\top A P_t \\
\bar{P}_{t-1} &= A^\top \bar{P}_{t|t-1} A \\
\bar{B} &= \bar{B} + \left(\bar{P}_{t|t-1} + \bar{P}_{t|t-1}^\top\right) B \\
\bar{A} &= \bar{A} + \bar{x}_{t|t-1} x_{t-1}^\top \\
\bar{x}_{t-1} &= A^\top \bar{x}_{t|t-1},
\end{aligned}
$$

and we will return $\bar{A}, \bar{B}, \bar{C}, \bar{x}_0, \bar{P}_0$.

Appendix G HMC Performance

While our results in the main text demonstrate the strength of HMC when applied to DSGE models, its speed and reliability might depend on the likelihood functions and priors and their numerical representation; see Stan Development Team (2022, Ch. 23-25) for advice on building and troubleshooting models. Here, we offer a brief summary of these issues in the context of state-space models.

Theory of HMC Performance. An active theoretical literature describes conditions under which HMC, or idealized versions more suitable for theoretical analysis, provably yields improved speed or accuracy. While these conditions are rarely verifiable for or directly applicable to economic models used in practice, they do describe features that qualitatively agree with situations where HMC performs well or poorly in our experiments. In the idealized setting of sampling from a d-dimensional $\mathcal{N}(0, I_d)$ vector, asymptotic analysis suggests that optimally tuned RWMH requires $O(d)$ function evaluations per (effective) independent sample drawn, while HMC requires $O(d^{1/4})$ likelihood and gradient evaluations per sample (Beskos et al., 2013). As reverse-mode AD provides gradients at an approximately equal cost to function evaluations, these results suggest a major speed advantage for HMC for large models.

Beyond this simple setting, results are known for broader classes of distributions; see Chen et al. (2020) and references therein. A typical setting for these results is distributions in which the posterior is log-concave and, thus, unimodal. In such cases, the gradient directs the sampler toward the vicinity of the mode and yields rapid exploration of the typical set. Without log-concavity, and particularly in cases with multiple, widely separated modes, as might be expected in cases of non-identification with non-interval identified sets, methods that make local proposals, including HMC and RWMH, may become "trapped" in a single mode for an exponentially long amount of time and may fail to accurately sample the posterior. Both practical experience and worst-case lower bound results show that this situation is indeed challenging.[23]

Within the class of log-concave distributions, curvature as measured by the condition number of the Hessian at the mode enters into bounds on the number of iterations needed to draw a sample. Curvature can be large when some variables are on very different scales than others, resulting in a long and narrow posterior that requires the sampler to move exactly along the preferred direction or else fall into a very low probability region. While gradient information helps find this direction more easily relative to a random walk, as gradients are only local, this shape requires the sampler to take small steps, slowing down exploration. The precise extent of this problem depends on the smoothness of the gradient and Hessian, which also enter into bounds, as high-frequency fluctuations in the density may be missed by a sampler. Finally, a good initial condition, formalized as a warm start in which the initial sample is drawn from a density with a bounded ratio to the true posterior, can prevent the sampler from requiring a long burn-in time of exploration before it reaches the typical set and begins exploring more rapidly. While initializing

[23]There are many approaches that often achieve reasonable performance in practice in structured or non-worst-case examples in the non-log-concave setting, such as methods based on tempering or annealing; see Ge et al. (2018) for discussion. As many such methods build on gradient-based methods like HMC, they are a potentially fruitful avenue for future applications of our procedures.

at the mode is a common approach that works reasonably well in low dimensions, in high dimensions the typical set may be far from the mode, and better performance may be found by also using curvature information to obtain a distribution, obtainable by a Laplace or variational approximation (Kucukelbir et al., 2017), or by annealing from a sample from a previous run, possibly with a slightly modified distribution.

Sampling Issues. For state-space models, the conditions we discussed above provide recommendations for practical model building and tuning. The choice to sample from the ϵ_t's, which are a priori independent, instead of the z_t's, which are a priori dependent, tends to result in a posterior that is much less correlated across variables, better conditioned, and easier to sample from. This is one example of reparameterization, a strategy for improving the performance of HMC that is recommended as part of "best practices" for model building (Stan Development Team, 2022, Ch. 24). Additional model features that lead to less correlated posteriors and faster sampling include avoiding highly persistent series by working with detrended data, modeling growth rates rather than levels, or using flow variables rather than stocks. For example, in simulations of the RBC model, numerical performance improved substantially when using investment rather than capital stock as an observable.

Another feature in state-space models that influences performance is the presence and degree of noise in the observable variables, which facilitates sampling for two related but conceptually distinct reasons. The first reason is to avoid near-stochastic singularity, where the model is not stochastically singular, but close to it. In our experience, near-singularity is a common source of failure both for HMC and particle filters. While this manifests in numerical problems, it is in essence a problem with the model, not with the sampler, and thus it needs to be solved at the model level.

The second reason can arise when using the joint likelihood approach. As latent states are among the random variables to be sampled, sampling them requires that they have a well-behaved density. In the case where some variables are observed without noise, one can back out the latent states (or a subset of them) exactly, meaning that conditional on the data, their distribution is a point mass without a density with respect to Lebesgue measure. Note the contrast with stochastic singularity, in which only a submanifold of the data space is in the support of the model distribution and so $p(\text{data}|\epsilon)$ has an infinite density with respect to Lebesgue measure, whereas in this situation a submanifold of the noise terms is known exactly and so $p(\epsilon|\text{data})$ has infinite density with respect to Lebesgue measure. In this case, one should analytically marginalize if possible, or use a constrained sampling method like Graham et al. (2022) if not. In comparison, when observations are convolved with additional noise, the conditional distribution has a density with respect to Lebesgue measure, but if the noise is very small, it will be close to

the point mass case, with a narrow peak that is highly curved. Thus, it will continue to be difficult to sample from this distribution. However, if the noise is sufficiently wide, this density will be smooth and with wide, flat optima, and sampling will be relatively easy. To see why this is helpful, consider the extreme case, where noise is so wide as to be equivalent to having no data at all, in which case the posterior will be equal to the prior, which is the distribution of the latent variables without conditioning.

Thus, in practice, the size of the observation noise term is the most important determinant of the speed of sampling from the joint likelihood. Reducing the variance of the term too far can result in slow mixing, numerical errors, and extreme sensitivity to floating point noise that render the sampler unusable. With heavy noise, even extremely large and highly nonlinear models, like our second-order DSGE, sample very rapidly with no numerical problems. This leads to a somewhat delicate paradox: in the very noisy case, there is less information in the data on the precise value of the latent variables, but this information can be obtained easily. As the noise is reduced, the data are more informative, but it is harder to learn about them using a sampler. This suggests that the joint approach is most useful with moderately noisy measurements. We found that starting the sampler at a larger noise and then reducing it, similar to annealing methods (Lee et al., 2022), leads to a more stable final output.

Appendix H Schmitt-Grohé and Uribe (2003) Model

Here we provide additional details for the real small open economy model in Section 6. We take Model 2 of Schmitt-Grohé and Uribe (2003) and add an AR(1) shock ζ_t to the risk premium and an AR(1) shock μ_t that multiplies the utility of consumption in each period. The model is expressed in logs and has 8 jump variables (c, h, GDP, i, kfu, λ, tb, ca), 7 state variables (d, k, r, a, riskpremium, ζ, μ), and 3 shocks $(\epsilon,\ \epsilon_u,\ \epsilon_v)$ that are mean 0 Gaussian with standard deviations $(\sigma_e, \sigma_u, \sigma_v)$ respectively. The equilibrium conditions

are (see Schmitt-Grohé and Uribe, 2003, for their interpretation):

$$\begin{bmatrix}
-e^{c(t)} - e^{i(t)} - \frac{1}{2}\left(-e^{k(t)} + e^{k(t+1)}\right)^2 \phi - \left(1 + e^{r(t)}\right) d\left(t\right) + d\left(t+1\right) + e^{\text{GDP}(t)} \\
-\left(e^{k(t)}\right)^{\alpha} \left(e^{h(t)}\right)^{1-\alpha} e^{a(t)} + e^{\text{GDP}(t)} \\
-e^{i(t)} - \left(1-\delta\right) e^{k(t)} + e^{k(t+1)} \\
-\beta\left(1 + e^{r(t+1)}\right) e^{\mu(t)} e^{\lambda(t+1)} + e^{\lambda(t)} \\
\left(\frac{-\left(e^{h(t)}\right)^{\omega}}{\omega} + e^{c(t)}\right)^{-\gamma} - e^{\lambda(t)} \\
\left(e^{h(t)}\right)^{-1+\omega} \left(\frac{-\left(e^{h(t)}\right)^{\omega}}{\omega} + e^{c(t)}\right)^{-\gamma} + \frac{-(1-\alpha) e^{\text{GDP}(t)} e^{\lambda(t)}}{e^{h(t)}} \\
\left(1 + \phi\left(-e^{k(t)} + e^{k(t+1)}\right)\right) e^{\lambda(t)} - \beta\left(1 - \delta + \phi\left(-e^{k(t+1)} + e^{\text{kfu}(t+1)}\right) + \frac{\alpha e^{\text{GDP}(t+1)}}{e^{k(t+1)}}\right) e^{\mu(t)} e^{\lambda(t+1)} \\
-r_w - \text{riskpremium}\left(t+1\right) + e^{r(t+1)} \\
-\zeta\left(t\right) - \psi\left(-1 + e^{-d_{bar} + d(t+1)}\right) + \text{riskpremium}\left(t+1\right) \\
-1 + \frac{\frac{1}{2}\left(-e^{k(t)} + e^{k(t+1)}\right)^2 \phi + e^{c(t)} + e^{i(t)}}{e^{\text{GDP}(t)}} + \text{tb}\left(t\right) \\
\frac{-d(t+1) + d(t)}{e^{\text{GDP}(t)}} + \text{ca}\left(t\right) \\
-k\left(t+1\right) + \text{kfu}\left(t\right) \\
-\rho a\left(t\right) - \epsilon\left(t+1\right) + a\left(t+1\right) \\
-\rho_u \zeta\left(t\right) - \epsilon_u\left(t+1\right) + \zeta\left(t+1\right) \\
-\rho_v \mu\left(t\right) - \epsilon_v\left(t+1\right) + \mu\left(t+1\right)
\end{bmatrix} \tag{H.1}$$

The variables (GDP, ca, r) are observed with Gaussian measurement error with variance 1e-3. Data are simulated from for 200 periods starting at the DSS, with pseudotrue parameter values $\gamma = 2.0, \omega = 1.455, \sigma_e = 0.0129, \delta = 0.1, \psi = 0.000742, \phi = 0.028, r_w = 0.04, d_{bar} = 0.7442, \rho_u = 0.2, \sigma_u = 0.003, \rho_v = 0.4, \sigma_v = 0.1$. For all parameters shared with Model 2 of Schmitt-Grohé and Uribe (2003), these are the calibrated values from that paper; other parameter choices are conventional values. Prior forms and parameterizations for estimated parameters are given in Table 16. All other parameters are fixed at their pseudotrue values in the simulation. In the `Julia` implementation, γ and ψ prior distributions are truncated to the intervals $[0.5, 7]$ and $[0.0003, 0.0015]$, respectively.

Appendix I Additional Figures

This appendix collects additional figures from the estimation of Fernández-Villaverde and Guerrón-Quintana (2021) using the first-order perturbation rather than the second-order perturbation reported in the main body of the paper.

Table 16: Prior Distribution for structural parameters

Parameter	Distribution	Mean	Std
α	Normal	0.3	0.025
γ	Normal	2.0	0.5
ψ	Normal	0.0007	0.0004
β_{draw}	Gamma	4	6
ρ	Beta	0.5	0.2
ρ_u	Beta	0.5	0.2
ρ_v	Beta	0.5	0.2

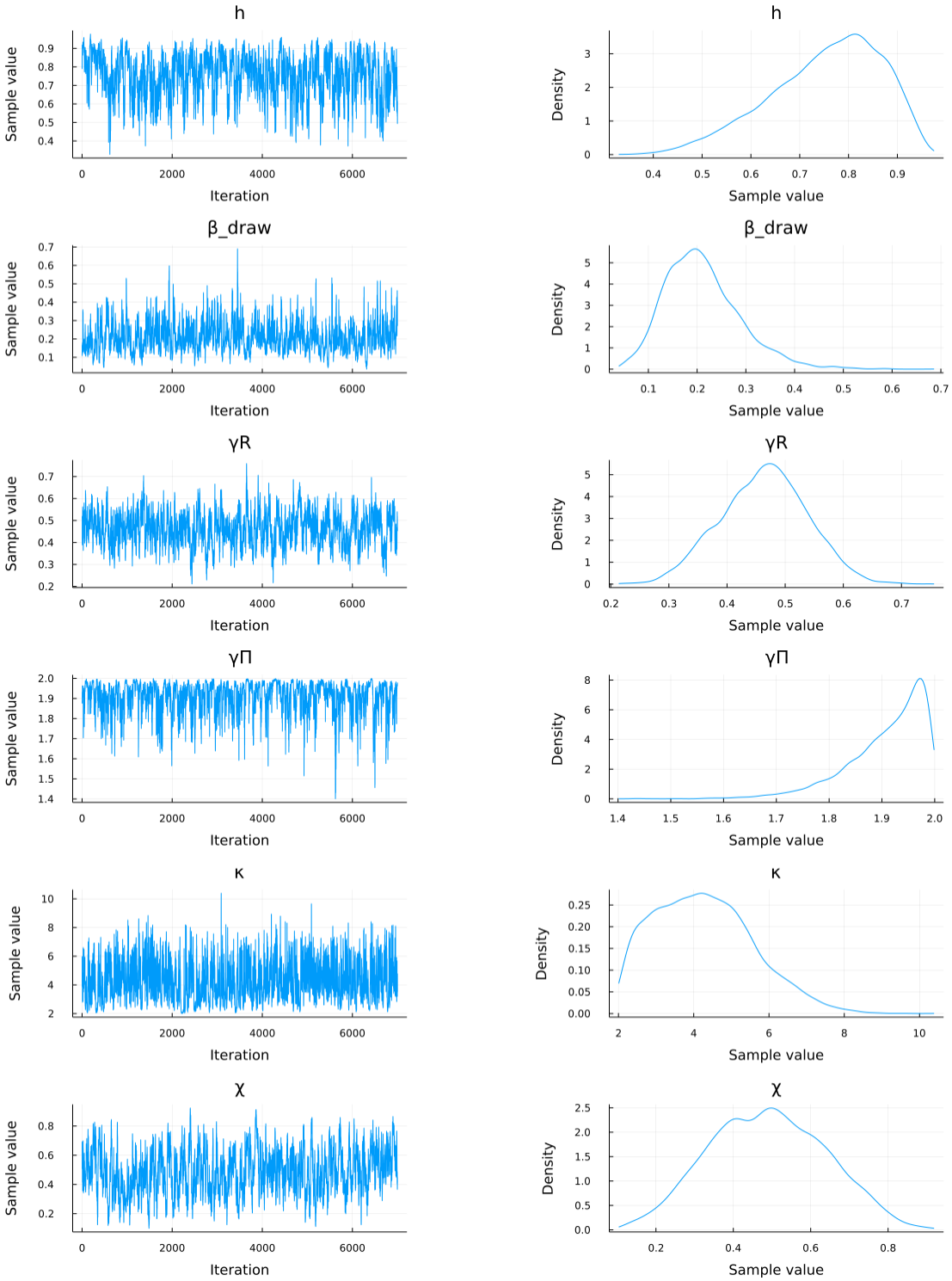


Figure 17: NUTS with Marginal likelihood via Kalman Filter, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 1

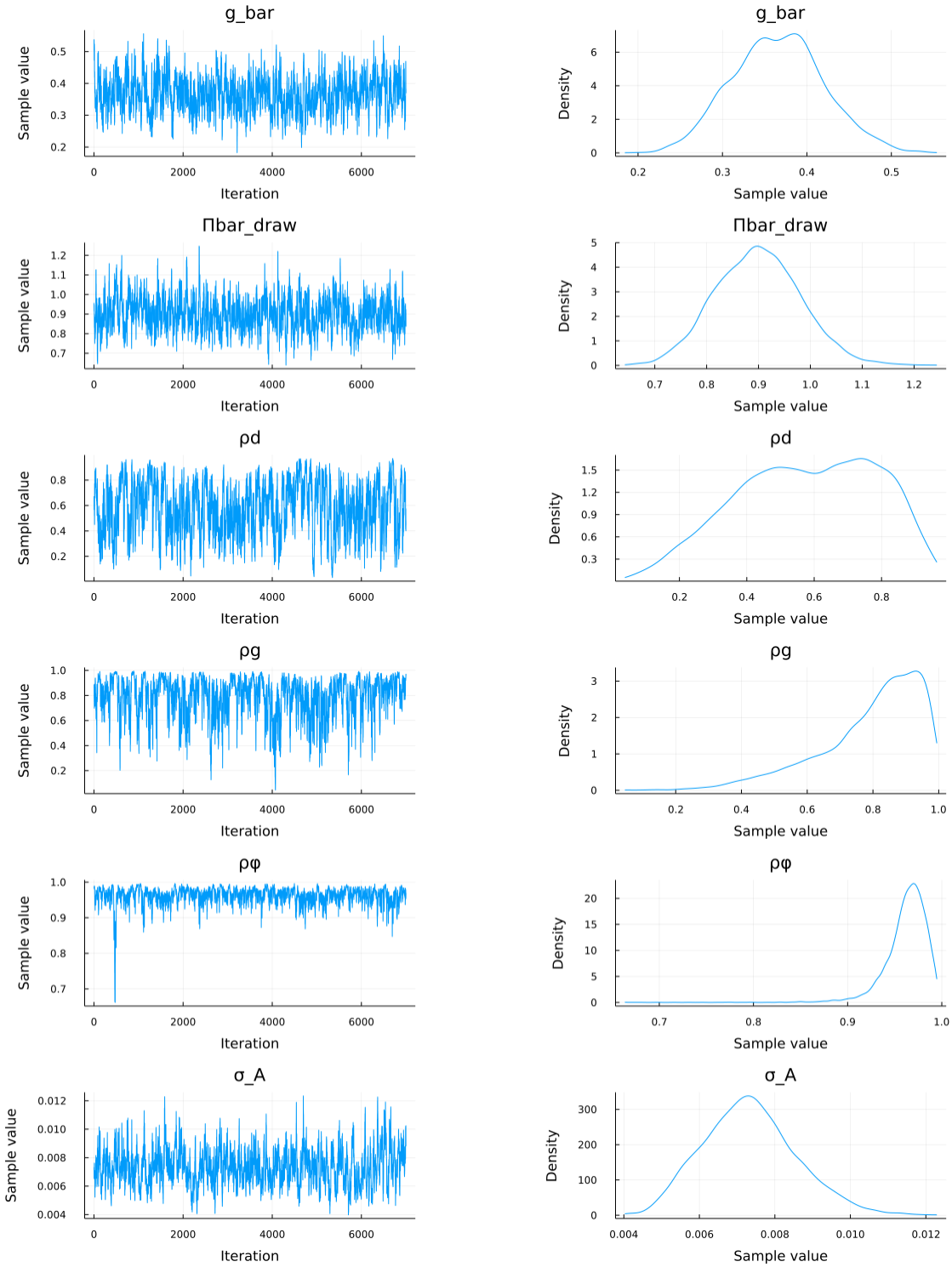


Figure 18: NUTS with Marginal likelihood via Kalman Filter, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 2

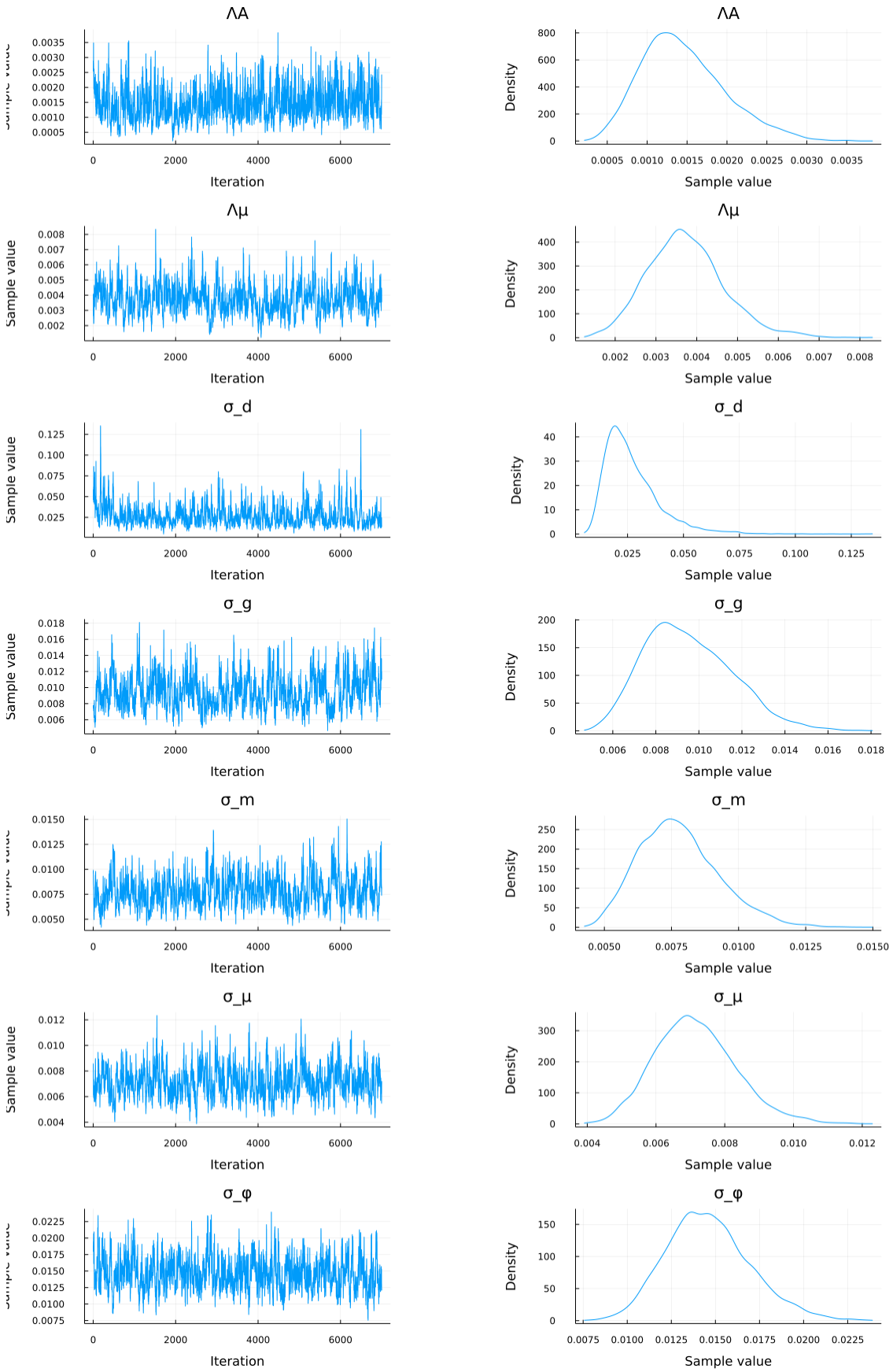


Figure 19: NUTS with Marginal likelihood via Kalman Filter, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 2

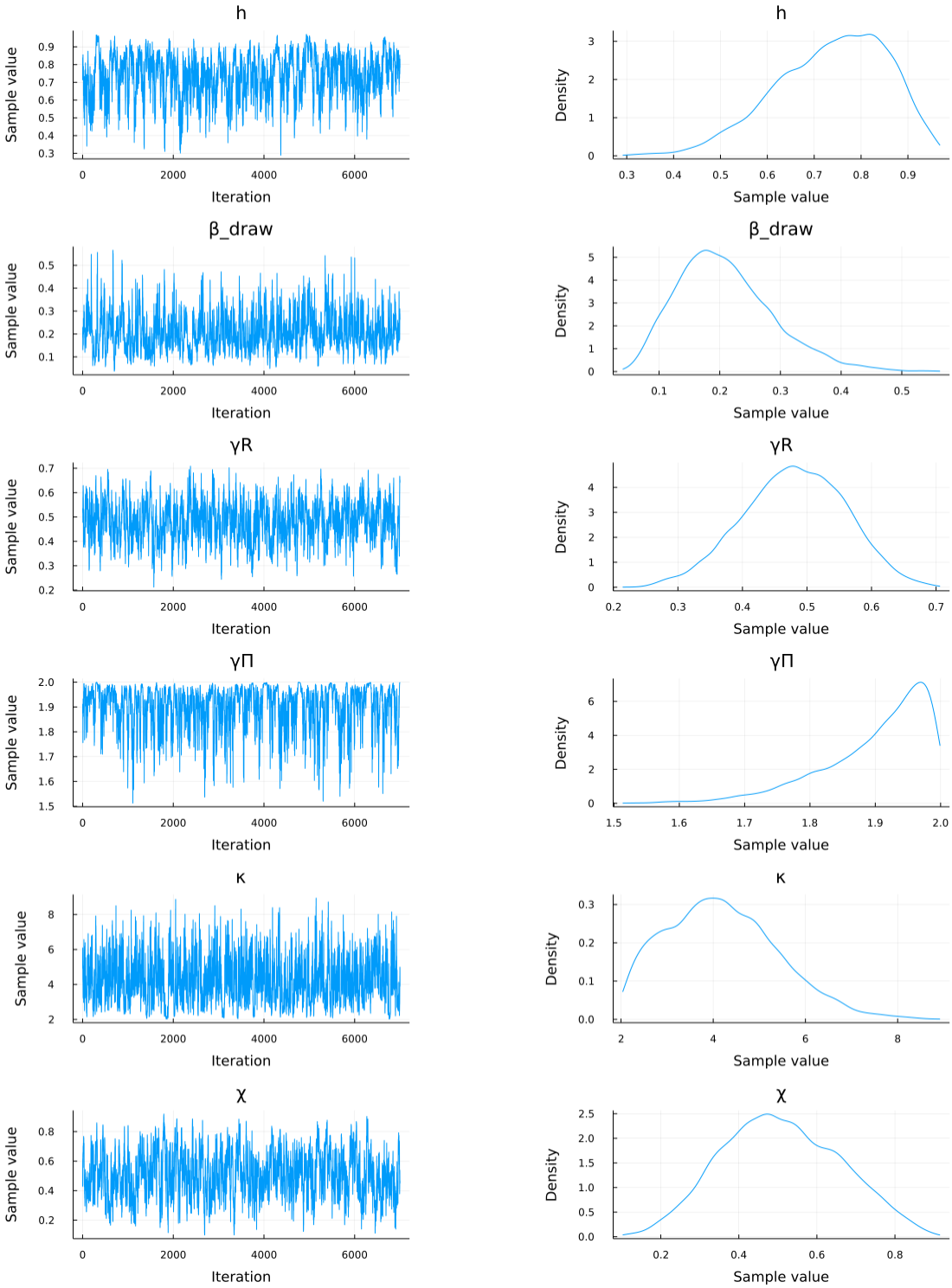


Figure 20: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 1

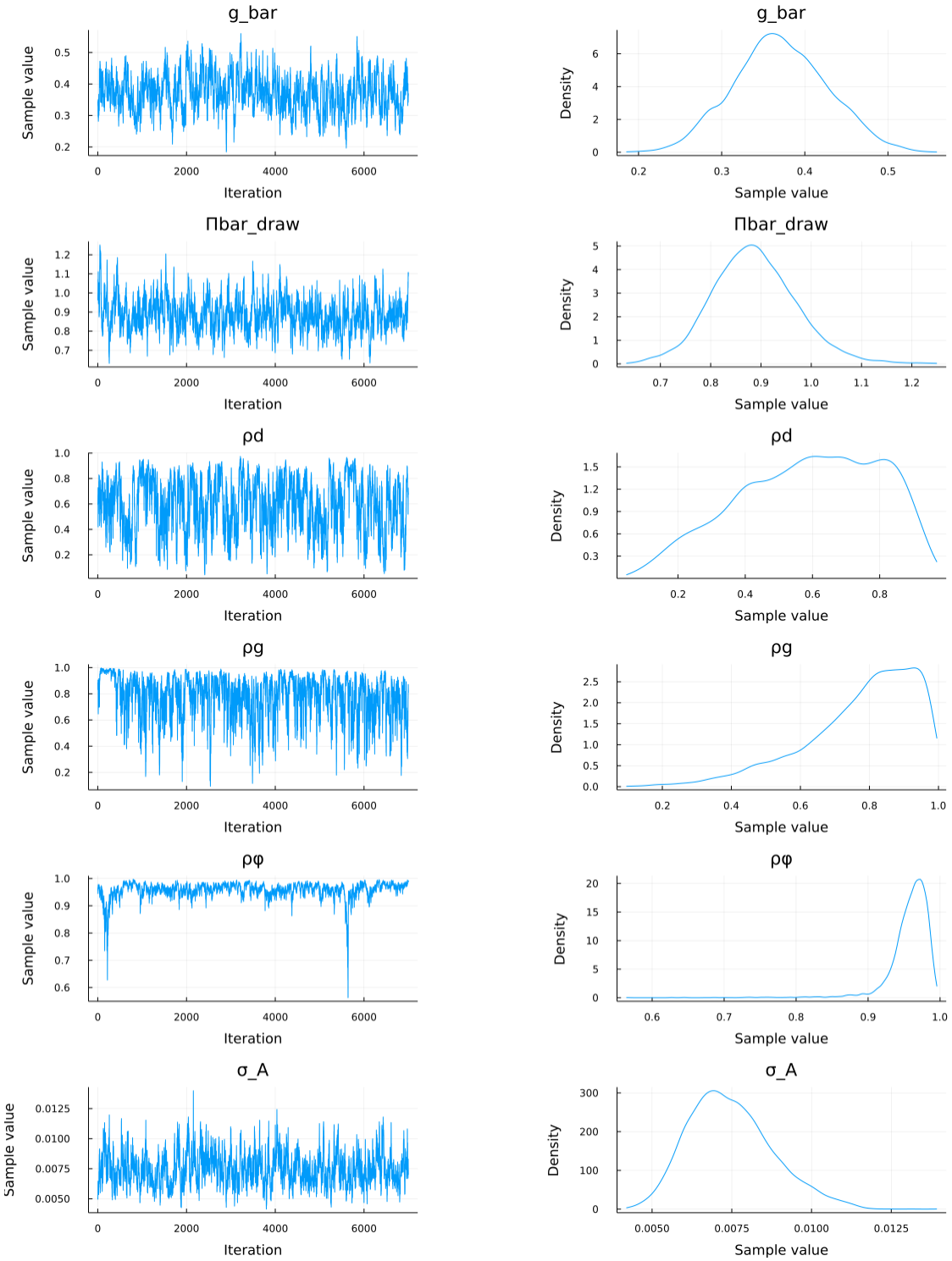


Figure 21: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 2

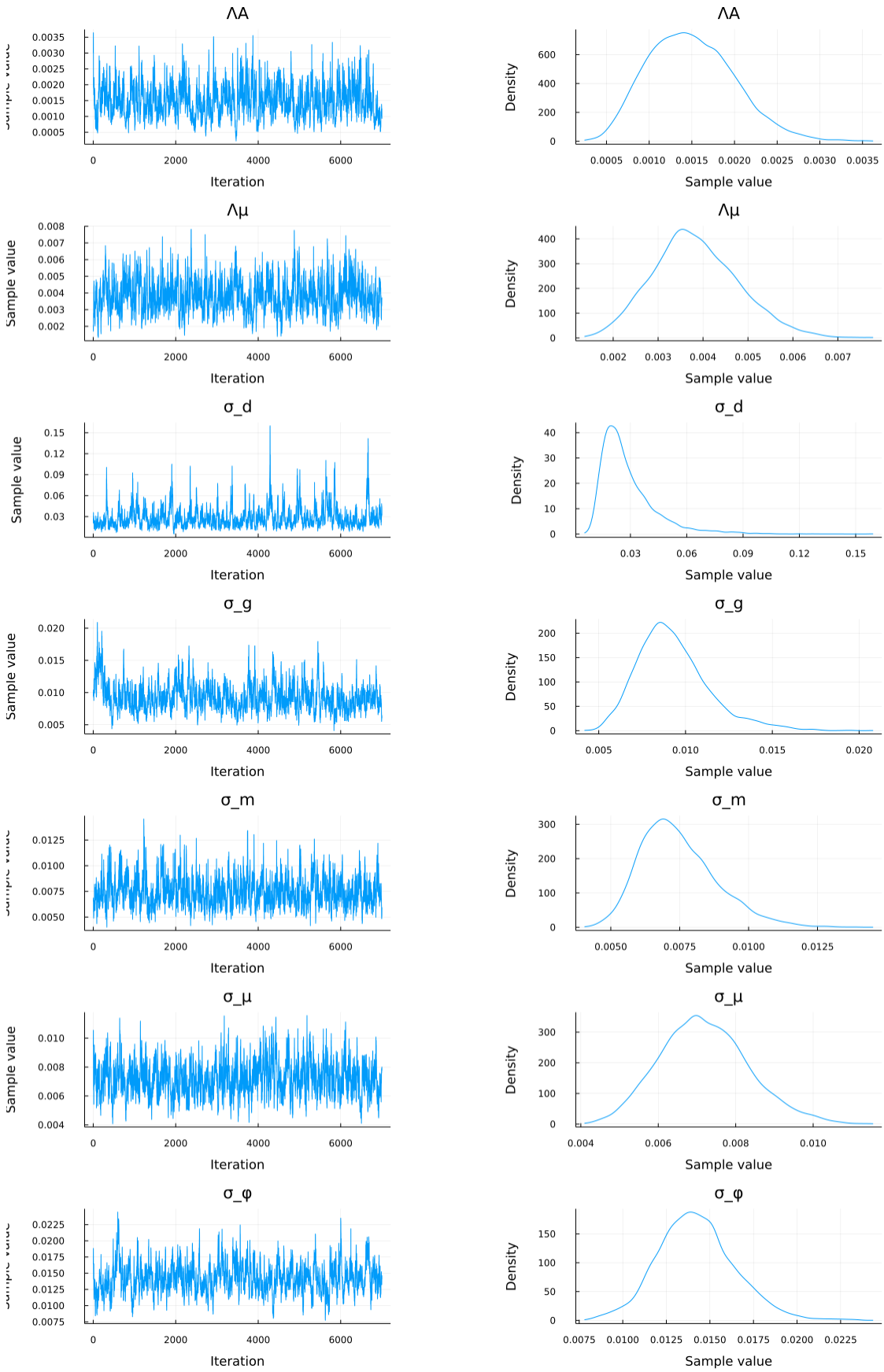


Figure 22: NUTS with Joint likelihood, Fernández-Villaverde and Guerrón-Quintana (2021), First-order, 3

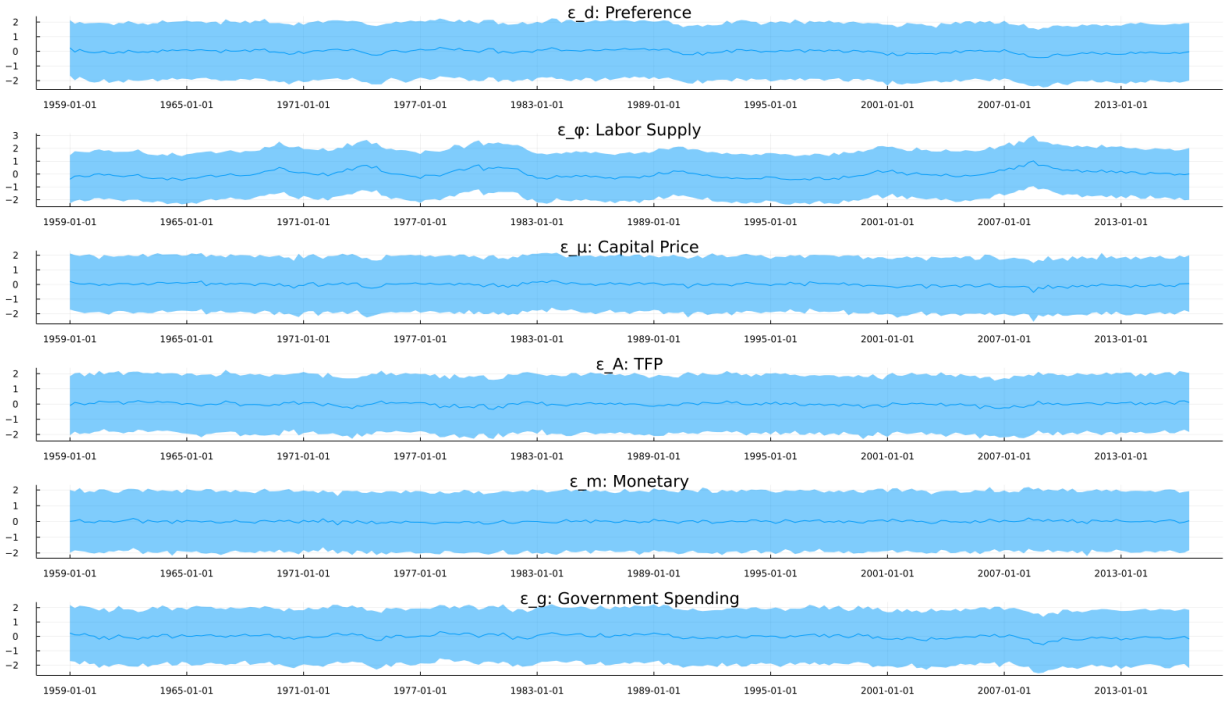


Figure 23: Inferred Shocks of Fernández-Villaverde and Guerrón-Quintana (2021): First-order with joint likelihood