Differentiable State Space Models and Hamiltonian Monte Carlo Estimation

David Childers,[1] Jesús Fernández-Villaverde,[2] Jesse Perla[3], Chris Rackauckas,[4] Peifan Wu[5]

February 15, 2023

[1]CMU [2]University of Pennsylvania [3]UBC [4]MIT and Maryland [5]Amazon

Inference for state space models

- Many economic models can be represented in state space form:
  - Observables z_t driven by the dynamics of some states x_t.
  - The law of motion of x_t is governed by some parameters θ.
- The bayesian approach estimates θ and $x^T = \{x_t\}_{t=1}^T$ with data $z^T = \{z_t\}_{t=1}^T$.
- Algorithms to deal with the nonlinear/non-Gaussian case are slow, numerically unreliable, and difficult to code, particularly in high dimensions.
- *Goal*: Efficient, reliable, and scalable methods for nonlinear models.
- Today: Can we have better samplers?

Hamilton Monte Carlo

- Yes, gradient information enables improved samplers $\Rightarrow$ Hamiltonian Monte Carlo (HMC):
  - Key idea: we add a momentum vector that induces a kinetic energy term (i.e., Hamiltonian dynamics).
  - Thus, we direct sampling towards high-probability regions and explore high-dimensional space efficiently.
  - HMC can perform joint estimation of θ and x^T, thus bypassing the need to filter.
- But, how do we (efficiently) find the required gradients of the likelihood of the model?
- Do not even think about numerical or symbolic derivatives!
- Automatic differentiation (AD) gets you part of the way there.
- But default implementations of AD (e.g., `Stan`) can be inefficient or unusable.

What do we do?

- Integrate DSGE models into a differentiable probabilistic programming environment.
- In particular, we design custom implicit gradient rules in an open-source library and provide a domain-specific language (DSL) that extends `Julia` with a `Dynare`-like syntax.
- You write your DSGE model as a composable building block.
- Language solves the model (up to a perturbation of order 2), computes all the required gradients with a custom differentiable backend, and samples from posterior using Hamiltonian Monte Carlo.
- Tools are applicable beyond this model class or inference algorithm. For instance, Hamiltonian Monte Carlo for maximum likelihood on time-series models.

Environment

- Work in standard setting for discrete-time dynamic stochastic expectational difference models:
  - Fernández-Villaverde, Rubio-Ramírez, and Schorfheide (2016).
- Many macro state-space models can be represented as in Schmitt-Grohé and Uribe (2001):

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

  - x: state variables.
  - y: control variables.
  - θ: parameters.
  - x', y': next period states.

Perturbation approximations

- The solution to the model is of the form:

$$y = g\left(x; \theta\right)$$
$$x' = h\left(x; \theta\right) + \eta\varepsilon'$$

- We approximate the solutions by perturbing the deterministic steady state:

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

  - Denote $\hat{x} = x - \bar{x}$, deviations from the deterministic steady state.
  - First-order: $\hat{y} = g_x\left(\bar{x}\right)\hat{x}, \hat{x}' = h_x\left(\bar{x}\right)\hat{x} + \eta\varepsilon'$.
  - Second-order: $\hat{y} = g_x\hat{x} + \frac{1}{2}\hat{x}' g_{xx}\hat{x} + \frac{1}{2}g_{\sigma\sigma}, \hat{x}' = h_x\hat{x} + \frac{1}{2}\hat{x}' h_{xx}\hat{x} + \frac{1}{2}h_{\sigma\sigma} + \eta\varepsilon'$.

Estimation problem

- General setup:
    - Prior distribution $p(\theta)$.
    - Dynamics (possibly pruned) generate likelihood of states $x^T : \Pi_{t=1}^T p(x_t|x_{t-1}, \theta)$.
    - Observed data z^T generated from (possibly noisy) observation equation: $\{z_t = q(x_t, y_t, \varepsilon_t; \theta)\}_{t=1}^T$.
- We apply Bayes rule to infer the posterior distribution of θ and states x^T:

$$p(\theta, x^T|z^T) \propto p(\theta)\Pi_{t=1}^T \left(p(z_t|x_t, \theta)p(x_t|x_{t-1}, \theta)\right)$$

- Often, we only care about marginal posterior: $p(\theta|z^T) = \int p(\theta, x^T|z^T)dx^T$.

Standard approach

- Notice that:

$$\underbrace{\ln p\left(\theta|z^T\right)}_{\text{log-posterior}} = \underbrace{\ln p\left(\theta\right)}_{\text{log-prior}} + \underbrace{\ln p\left(z^T|\theta\right)}_{\text{log-likelihood}} + C = \ln p\left(\theta\right) + \sum_{t=1}^{T} \ln p\left(z_t|z^{t-1},\theta\right) + C$$

where:

$$\begin{aligned} p\left(z_t|z^{t-1},\theta\right) &= \int p\left(z_t, x_t|z^{t-1},\theta\right) \mathrm{d}x_t \\ &= \int p\left(z_t|x_t,\theta\right) p\left(x_t|z^{t-1}\right) \mathrm{d}x_t \end{aligned}$$

  - For linear-Gaussian models, $p\left(x_t|z^{t-1}\right)$ is updated by Kalman filter.
  - For others, $p\left(x_t|z^{t-1}\right)$ usually goes through particle filter.

- We draw from $p\left(\theta|z^T\right)$ using a Random Walk Metropolis-Hastings (RWMH) or a related refinement.

Joint likelihood

- But recall that we also have the joint likelihood of θ conditional on data z^T and state states x^T:

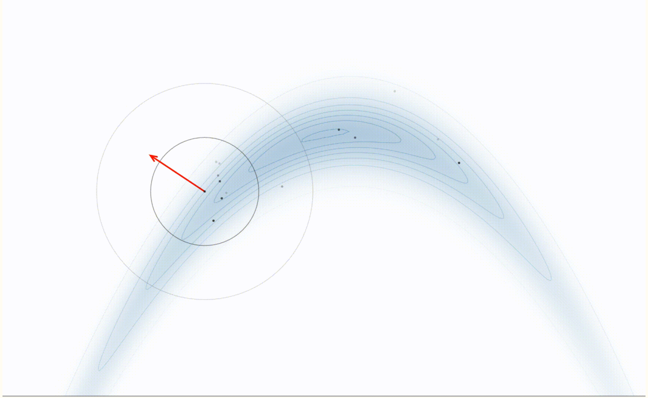
$$\begin{aligned}
\underbrace{\ln p\left(\theta, x^T | z^T\right)}_{\text{log-posterior}} &= \underbrace{\ln p\left(\theta, x^T\right)}_{\text{log-prior}} + \underbrace{\ln p\left(z^T | x^T, \theta\right)}_{\text{log-likelihood}} + C \\
&= \ln p(\theta) + \ln p\left(x^T | \theta\right) + \sum_{t=1}^{T} \ln p\left(z_t | x^t, \theta\right) + C \\
&= \ln p(\theta) + \sum_{t=1}^{T} \ln p\left(z_t | x_t, \theta\right) + \sum_{t=1}^{T} \ln p\left(x_t | x_{t-1}, \theta\right) + \ln p\left(x_0 | \theta\right) + C \\
&= \ln p(\theta) + \sum_{t=1}^{T} \ln p\left(z_t | \epsilon^t, x_0, \theta\right) + \sum_{t=1}^{T} \ln p\left(\epsilon_t | \theta\right) + \ln p\left(x_0 | \theta\right) + C
\end{aligned}$$

- Unfeasible to use RWMH to draw from this joint likelihood.

Our approach

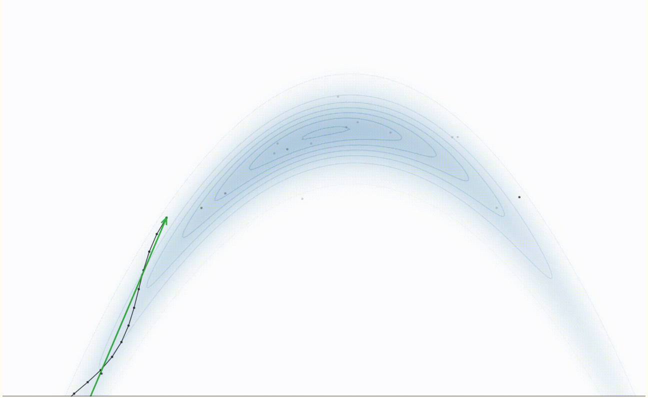
- Standard approach scales poorly:
    - Exact filters applicable only to a limited model class.
    - Approximate filters are costly and prone to failure.
    - RWMH exhibits dimension-dependent time to draw "effective" samples.
- In this paper:
    - Replace RWMH with HMC: more efficient and reliable sampling.
    - Draw from joint likelihood, allowed by HMC, bypassing filtering.
    - A filter-free, universal way to estimate non-linear, non-Gaussian models.

Random Walk Metropolis-Hastings



- $\theta_{i+1} \sim \mathcal{N}(\theta_i, M)$.
- Accept with probability $\min\left(1, \frac{p(\theta_{i+1}|y^T)}{p(\theta_i|y^T)}\right)$.

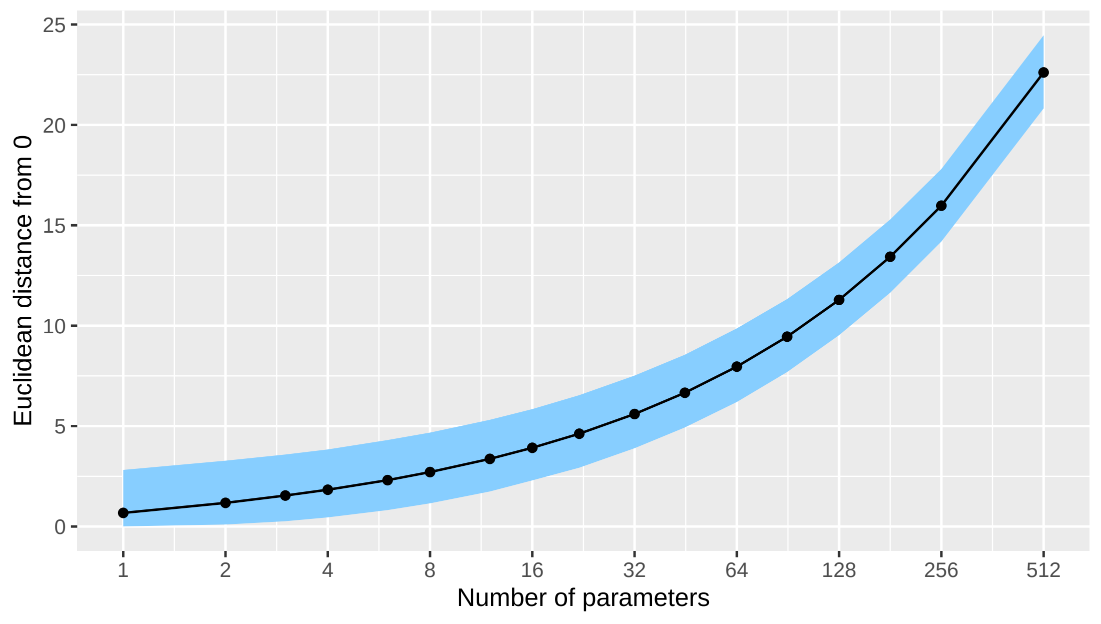
Hamiltonian Monte Carlo



- $q_{i+1} \sim \mathcal{N}(0, M)$.
- Set $q(0) = q_{i+1}$ and $\theta(0) = \theta_i$.
- $\theta_{i+1} | q_{i+1}$ through $\tau = 0, \ldots, L$:
  - $q(\tau + \epsilon/2) =$
    $q(\tau) + \frac{\epsilon}{2} \nabla_\theta \log p(\theta(\tau)|y^T)$.
  - $\theta(\tau + \epsilon) = \theta(\tau) + \epsilon M^{-1} q(\tau + \epsilon/2)$.
  - $q(\tau + \epsilon) =$
    $q(\tau + \epsilon/2) + \frac{\epsilon}{2} \nabla_\theta \log p(\theta(\tau + \epsilon)|y^T)$.
- Accept with probability:
  $\min\left(1, \frac{\exp(\log p(\theta_{i+1}|y^T) - \frac{1}{2} q_{i+1}^T M^{-1} q_{i+1})}{\exp(\log p(\theta_i|y^T) - \frac{1}{2} q_i^T M^{-1} q_i)}\right)$.

High-dimensional geometry

- Expectation values are given by accumulating the integrand over a volume.
- In regular models, posterior density decays exponentially with distance from mode: there is not much volume at the mode!
- Simple example: think about tossing a coin 1000 times, with $p(H) = 0.500000001$.
  1. $\{H, H, ..., H\}$ is the most likely event.
  2. And yet, most events will have around 500 heads!
- In high D, volumes concentrates in thin shell $O(\sqrt{D})$ away from mode: typical set (this a manifestation of concentration of measure).
- Without preferred direction, RWMH must take small steps to stay on the typical set.
- HMC can use gradients to stay on the typical set and explore posterior better.
- So, everything is about getting derivatives right!

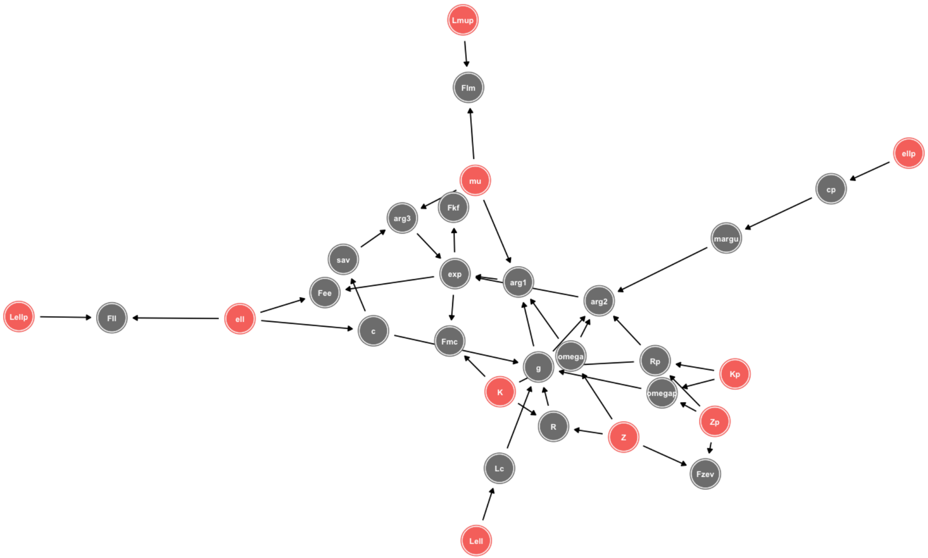

25
20
15
10
5
0
Euclidean distance from 0
1
2
4
8
16
32
64
128
256
512
Number of parameters

Differentiable programming

- Differentiable programming is one of the top research areas in computer science right now.
- This is the programming approach used by `ChatGPT`.
- Idea: write code that can be easily differentiated. How?
- Think about any program as a compositional function that maps inputs to outputs by composing functions along directed acyclical graph (DAG).
- Derivative computed by accumulating derivatives of node functions along a DAG using AD.

Computational Graph for Krusell Smith Model (Childers & Dogra 2018)

Highlighted nodes are input variables



Automatic differentiation and the cheap gradient principle

- We apply AD within and between blocks by relying on a large library of primitives.
- By grouping terms, we can reduce cost *exponentially* relative to naive symbolic derivatives.
- Order of accumulation in chain rule affect performance:
  - *Forward topological mode*: accumulate from inputs to outputs.
  - *Reverse topological mode*: pass along sensitivities ("adjoints") from outputs to inputs.
- *Cheap gradient principle*: Reverse mode computes gradients in $O(1)$ time:
  - Upshot: gradients same order of cost as function evaluation.
  - Gradient-based algorithms (e.g., HMC) as cheap per iterate as 0th order (e.g., RWMH).

How to differentiate a program block?

- One could directly *unroll* a block: differentiate through the steps. In practice impossible for DSGE models. Why?
  - Think about the QZ algorithm complex-valued, eigenvalue sort only almost surely pointwise differentiable.
  - Sorry, `Stan` fans. Though, see Farkas and Tatar (2020).
- Instead, we register *custom adjoint rules* for DSGE models to bypass AD system for efficient derivative program.
  - Improvement with respect to existing methods, such as Iskrev (2010).

Three software components

- Companion library: `DifferentiableStateSpaceModels.jl` provides differentiable rational expectations solver implementations at first and second order with custom adjoint rules.
- Methods for difference equations added to `DifferenceEquations.jl`: Conditional and unconditional sequence and likelihood evaluation for simulations and IRFs.
- Likelihoods handled within `Turing.jl` probabilistic programming library.

Three models

- Real business cycle model at first (linearization) and second order (with simulated data).
- The real small open economy model of Schmitt-Grohé and Uribe (2003) (with simulated data).
- The mid-size New Keynesian model of Fernández-Villaverde and Guerrón-Quintana (2021) (with real data).
- Examples use NUTS (Hoffman and Gelman, 2014) No U-Turn Sampler: Variant of HMC with adaptively-chosen integration length.

NUTS

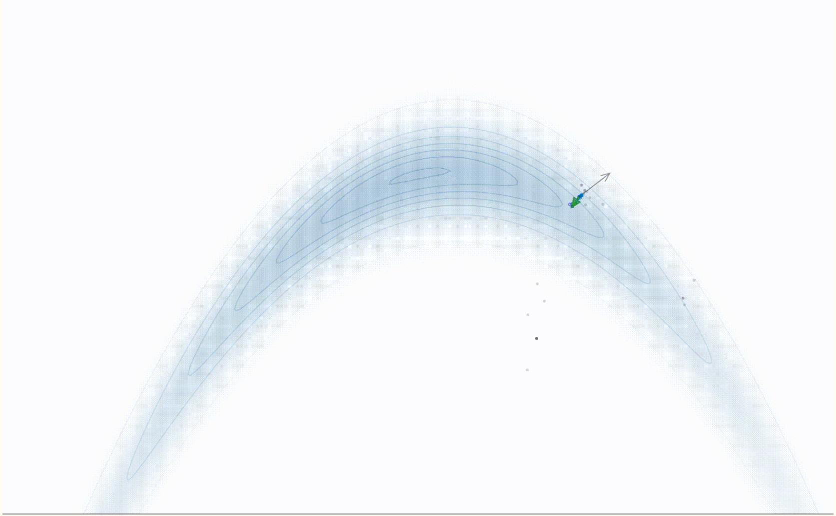


Table 1: RWMH with Marginal Likelihood, RBC Model, First-order

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

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

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

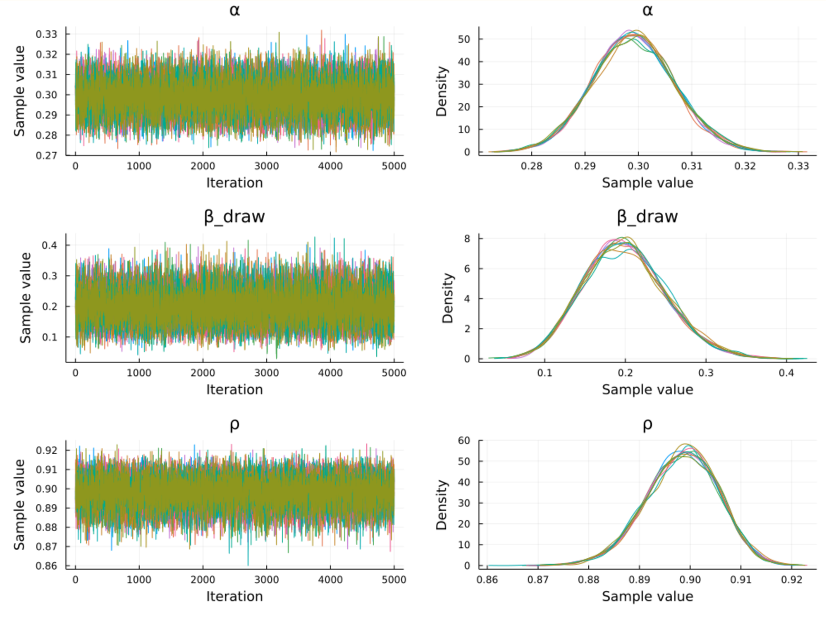


Figure 1: NUTS with Marginal Likelihood, RBC Model, First-order

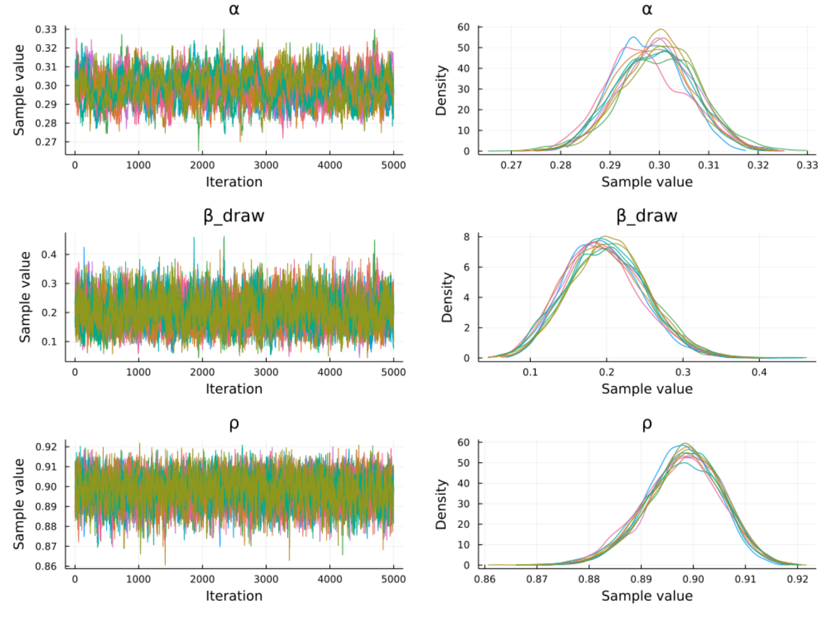


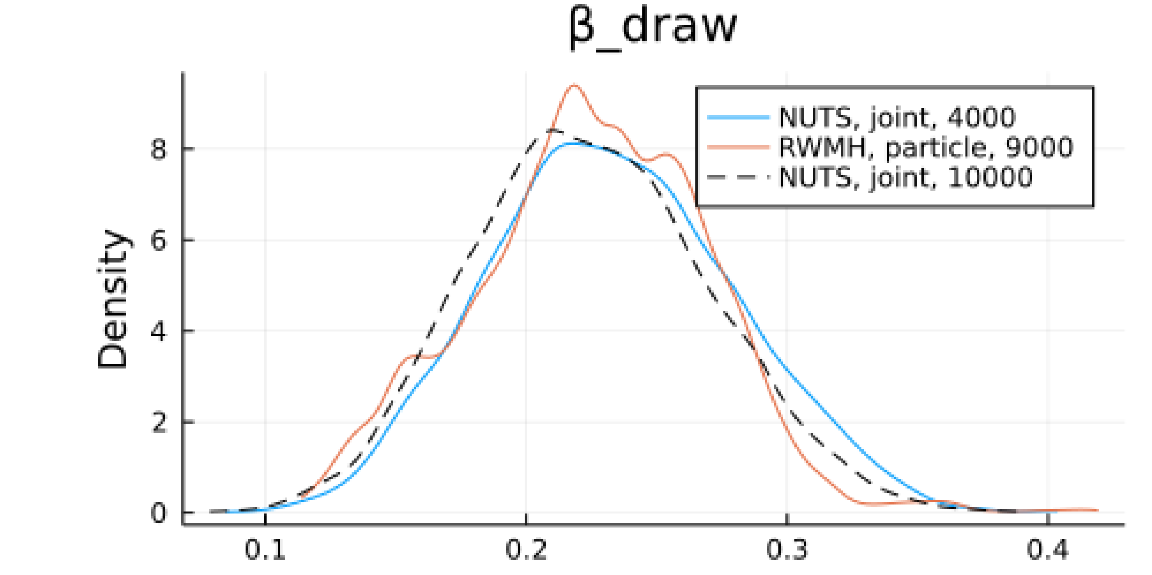
Figure 2: NUTS with Joint Likelihood, RBC Model, First-order

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

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


β_draw
Density
NUTS, joint, 4000
RWMH, particle, 9000
NUTS, joint, 10000
0
2
4
6
8
0.1
0.2
0.3
0.4

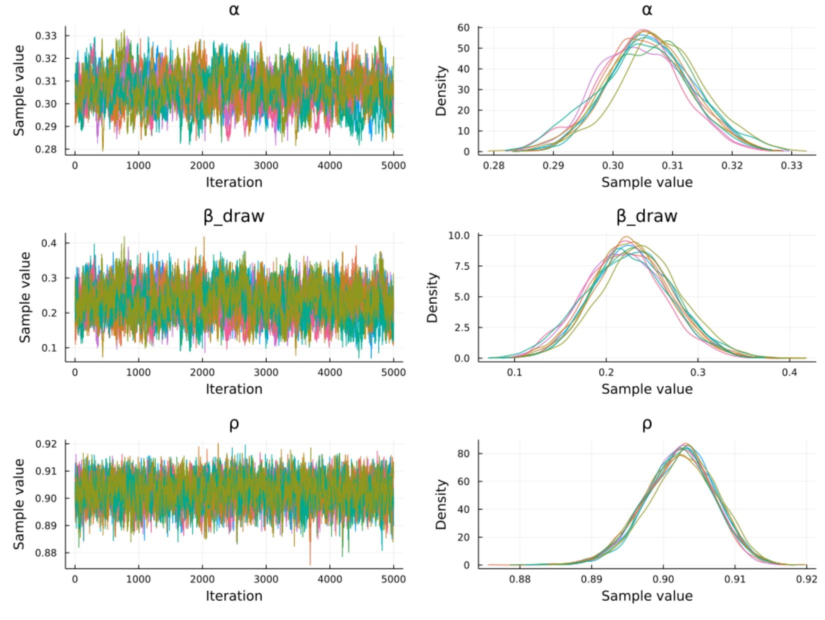


Figure 3: NUTS with Joint Likelihood, RBC Model, Second-order

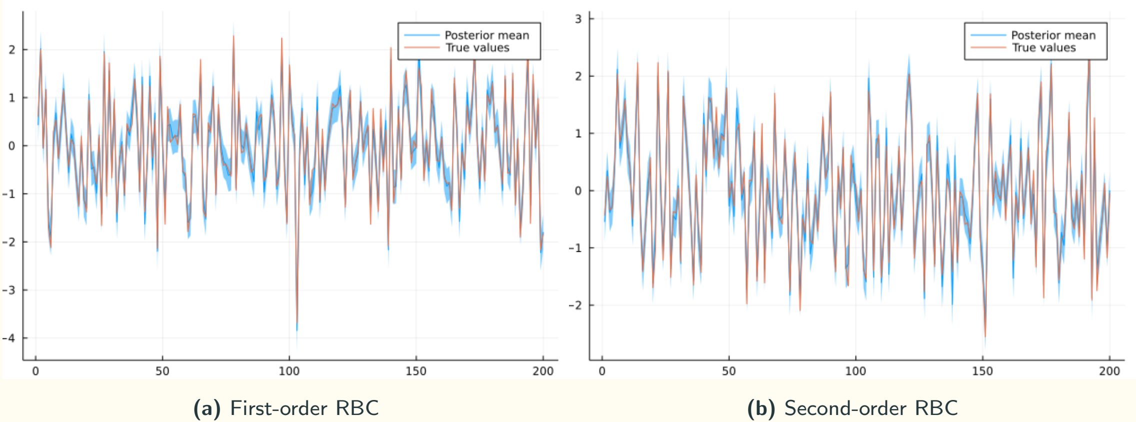


Figure 4: Inferred TFP Shocks of RBC Model

Table 6: 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%

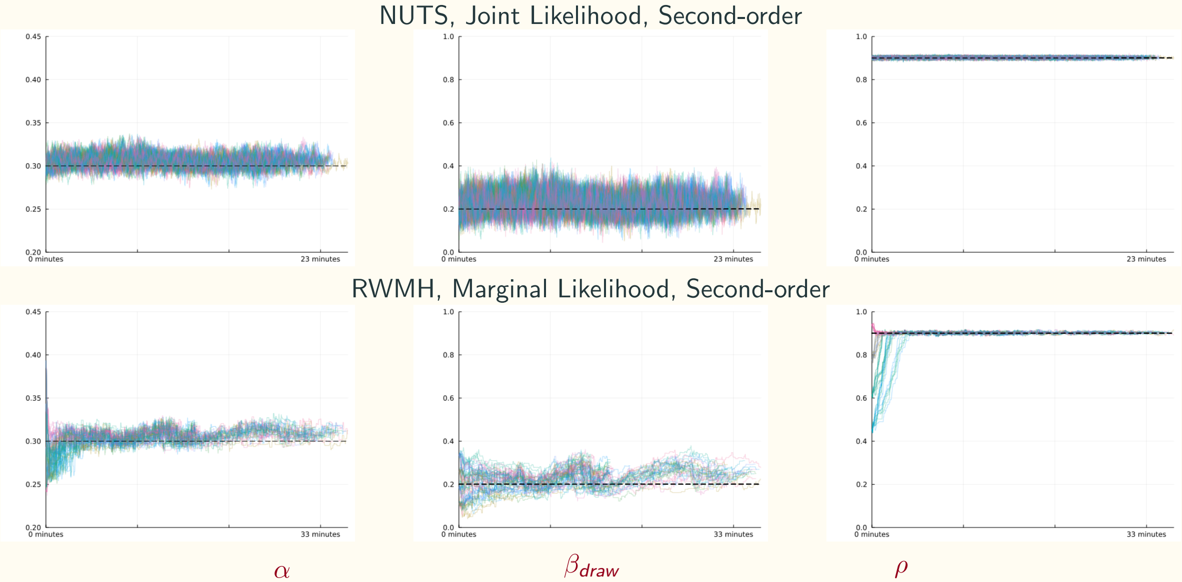


Figure 5: Robustness Comparison on Second-order RBC: Trace Plot

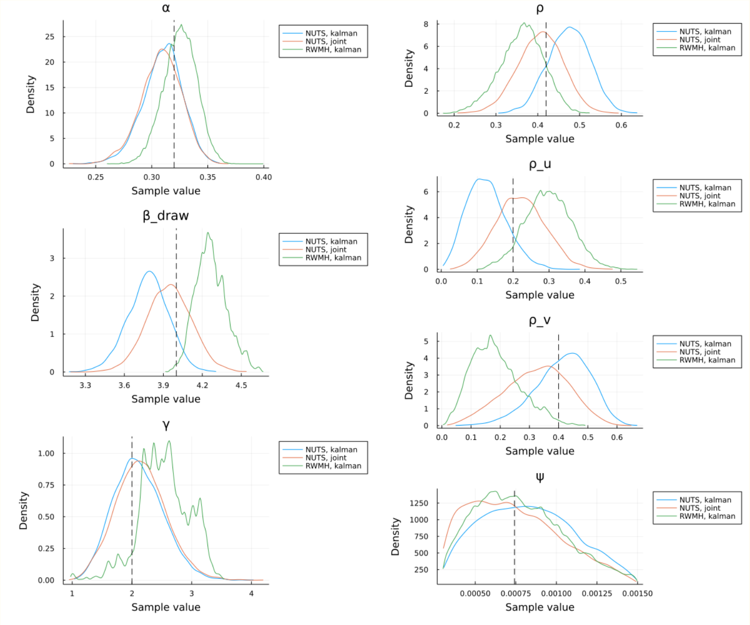


Figure 6: First Order RWMH+Kalman vs. HMC+Kalman vs. HMC+Joint

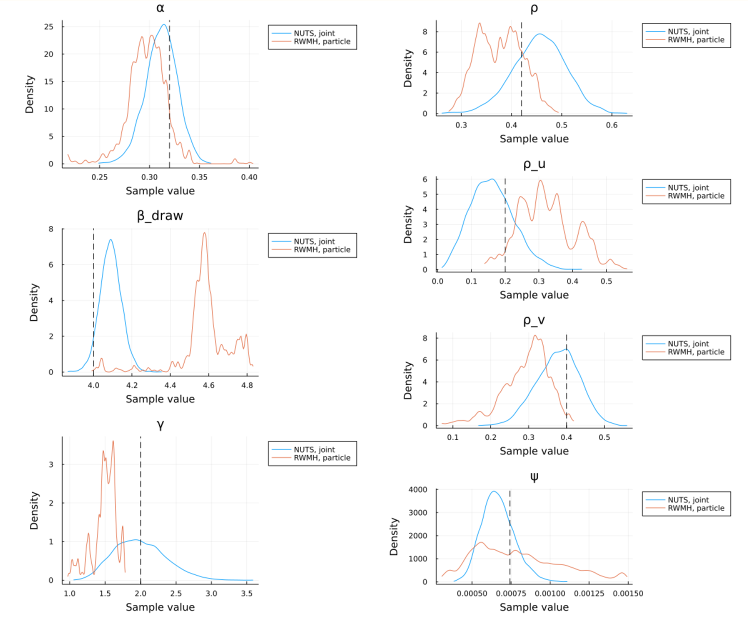


Figure 7: Second Order RWMH+Particle vs. HMC+Joint

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
$100\left(\bar{\Pi}-1\right)$	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

Conclusion

- Differentiable state space models enable easy and scalable nonlinear: DSGE inference by HMC.
- Many more applications are possible:
  - VI, SVGD, SGMCMC, parallel tempering, HMC within SMC.
  - Projection, higher order perturbation, differentiable filtering.
  - Neural networks, optimizers.
- Porting into FPGAs (Field Programmable Gate Arrays): Fernández-Villaverde et al. (2022).