State Space Model Programming in Turing.jl

LAFI 2025

Tim Hargreaves [1] Charles Knipp [2]

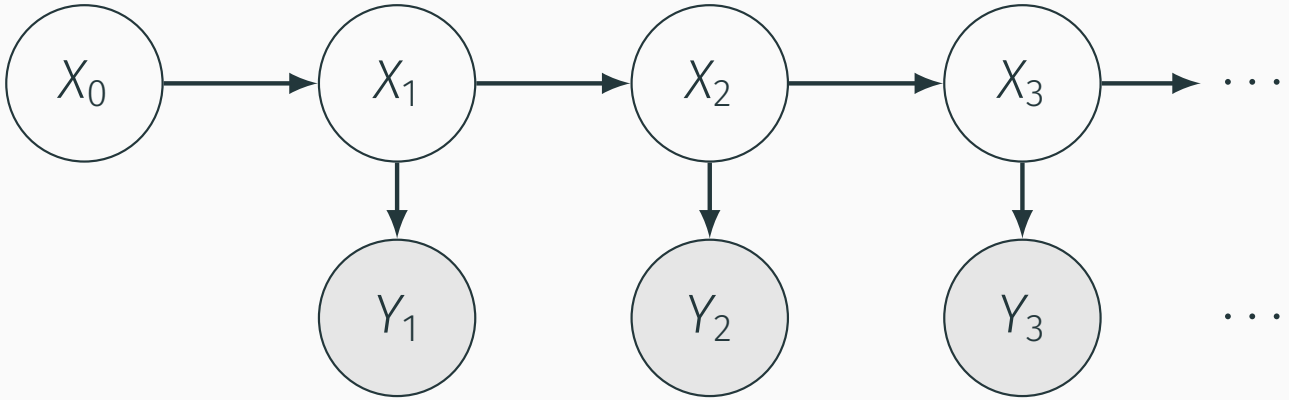
[1] Department of Engineering, University of Cambridge, UK

[2] Federal Reserve Board of Governors, USA

DISCLOSURE

The beliefs presented by the authors are not a reflection of the Federal Reserve Board or the greater Federal Reserve System

State Space Models



Governed by three distributions:

Initialisation: $p_\theta(x_0)$
Transition: $p_\theta(x_t|x_{t-1})$
Observation: $p_\theta(y_t|x_t)$

Joint distribution of all x_t and y_t is

$$p_\theta(x_{0:T}, y_{1:T}) = p_\theta(x_0) \prod_{t=1}^{T} p_\theta(x_t|x_{t-1}) p_\theta(y_t|x_t)$$

Filtering

Targeting the posterior distribution $p_\theta(x_t|y_{1:t})$ is called the *filtering* problem, which recursively breaks down into two steps:

Predict: $p_\theta(x_t|y_{1:t-1}) = \int p_\theta(x_t|x_{t-1})p_\theta(x_{t-1}|y_{1:t-1})dx_{t-1}$

Update: $p_\theta(x_t|y_{1:t}) \propto p_\theta(y_t|x_t)p_\theta(x_t|y_{1:t-1})$

SMC vs MCMC

Unlike MCMC, the filtered posterior is calculated *online*. Neither Turing nor Stan supports this treatment out of the box.

A Note on Structure

In a fully modular setting, these posteriors should be able to produce **(1)** a sample and **(2)** a marginal likelihood.

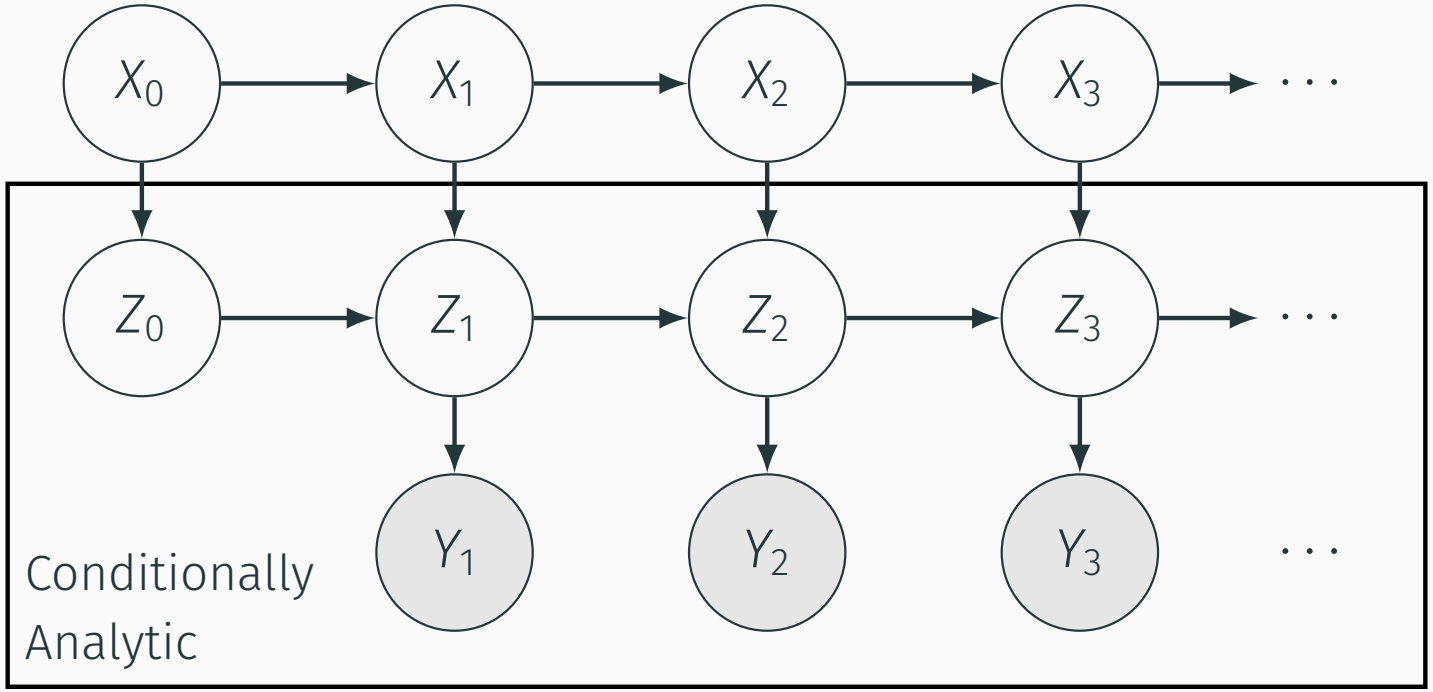
`ParticleDistribution`

1. Weighted sample of particles
2. Output from the particle filter

`GaussianDistribution`

1. Analytical sampling
2. Output from the Kalman filter

Rao-Blackwellisation



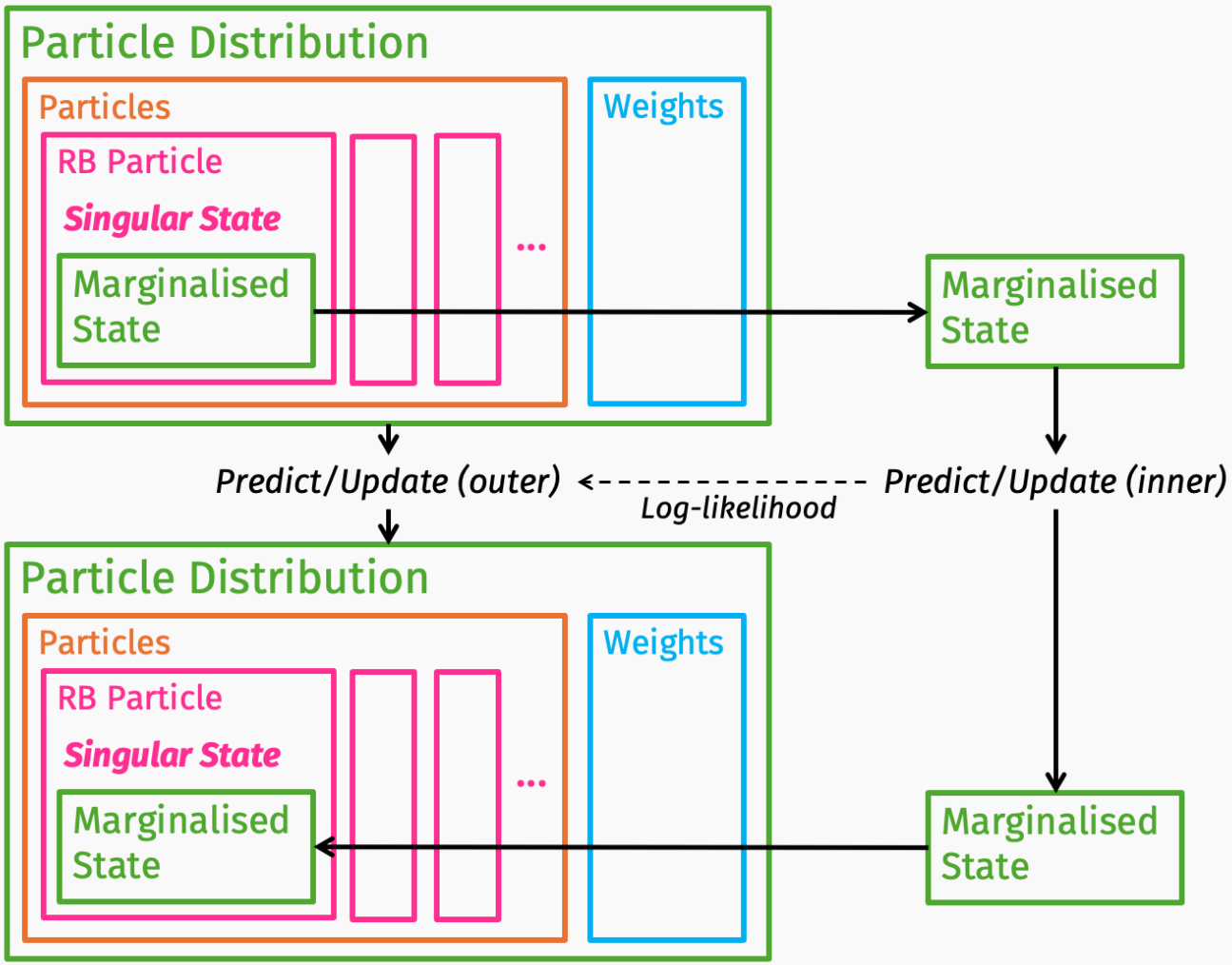
Partially-analytic inference using Rao-Blackwellised particle filter drastically reduces variance.

Essentially, replaces part of particle state with an entire distribution and runs the analytic inference algorithm over the particles.

No general purpose implementation exists, until now.

Benefits of Interface Design

Arbitrary analytic conditional sub-models can be handled by dispatching to the inner algorithm.



Benchmarks

Benchmarks[1] for Rao-Blackwellised Particle Filter on a hierarchical linear Gaussian model with $D_{\text{outer}} = D_{\text{inner}} = D_y = 2$, $N_{\text{particles}} = 10^5$:

Implementation	Mean Time per Filtering Step (ms)
Vanilla CPU	800
StaticArrays.jl	140
torch-kf[2]	3.1
GPU (CUBLAS)	1.9
GPU (Custom Kernels)	0.4

Developing custom CUDA kernels for fast linear algebra on large batches of tiny matrices, discussed in future publication.

[1] CPU: AMD Ryzen Threadripper 7960X 24-Core; GPU: NVIDIA GeForce RTX 4090

[2] Performing just batch Kalman updates, not full RBPF

Design Features

CALLBACKS

At each iteration of the filter, we allow the user to perform any number of user-defined callbacks

- Storage of filtered states [2]
- Forecasting calculations (smoothed and filtered)
- Calculation of summary statistics

MODULAR RESAMPLING

We have designed a modular resampling framework to encourage custom resamplers such as the following:

- Differentiable resampling with optimal transport [1]
- Parallel resampling on the GPU [3]

Automatic Differentiation

Very difficult because of the recursive nature of SMC.

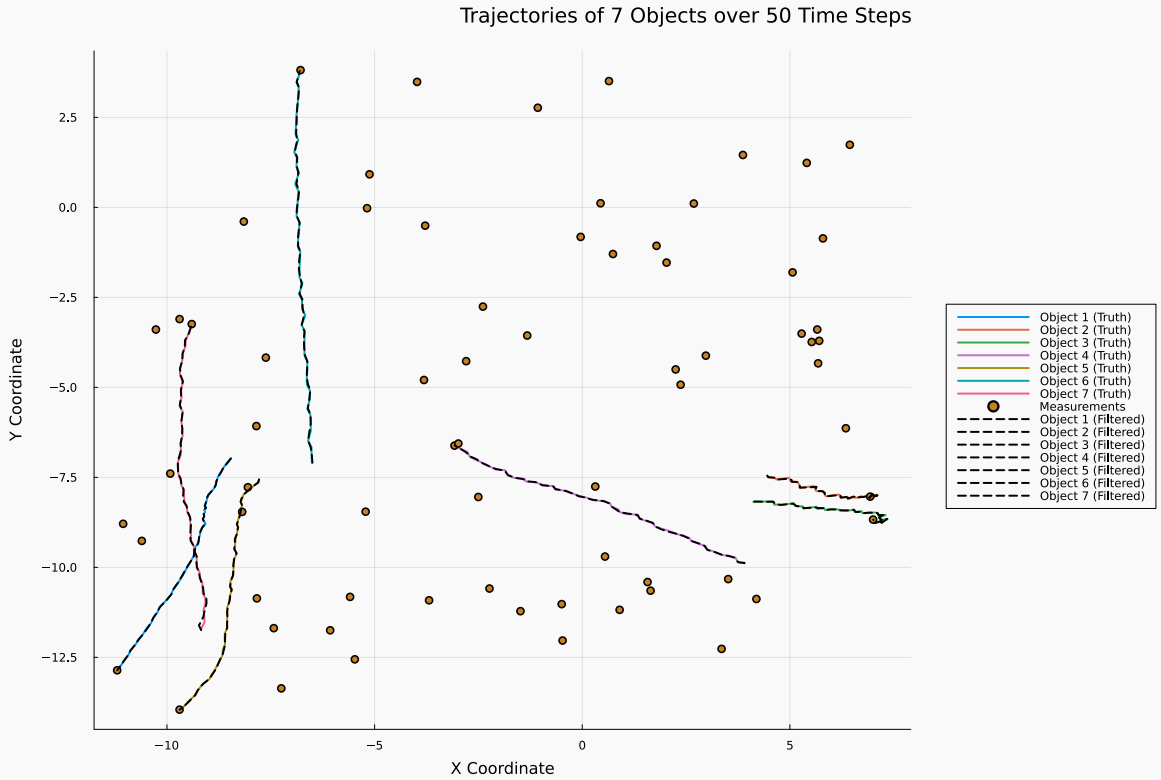
Currently, we support using `ForwardDiff.jl` for efficient autodiff.

MOTIVATIONS

- Support for variational filtering algorithms
- Maximum likelihood parameter estimation
- Potential use in Hamiltonian Monte Carlo

Target Tracking

Extension that supports multiple-target tracking with (joint) probabilistic data association.



Soon to add path initialization and deletion to have feature parity with the Stone Soup framework [4].

Particle Markov Chain Monte Carlo

Using `AdvancedMH.jl` in combination with our framework (`GeneralisedFilters.jl`), applying PMMH to a user-defined SSM is trivial:

```julia
function density(θ::Vector{T}) where {T<:Real}
    if insupport(prior_dist, θ)
        _, ll = GeneralisedFilters.filter(toy_model(θ), algo, data)
        return ll + logpdf(prior_dist, θ)
    else
        return -Inf
    end
end

# define the sampler
pmmh = RWMH(MvNormal(zeros(...), 0.01*I))
model = DensityModel(density)

# run 50,000 draws of PMMH
chains = sample(model, pmmh, 50_000)
```

Particle Gibbs also possible using conditional SMC implementation.

References i

- A. Corenflos, J. Thornton, G. Deligiannidis, and A. Doucet.
  Differentiable particle filtering via entropy-regularized optimal transport, 2021.
- P. E. Jacob, L. M. Murray, and S. Rubenthaler.
  Path storage in the particle filter, 2015.
- L. M. Murray, A. Lee, and P. E. Jacob.
  Parallel resampling in the particle filter, 2016.
- P. A. Thomas, J. Barr, B. Balaji, and K. White.
  An open source framework for tracking and state estimation ('stone soup'), 2017.