Accelerated Inference for Partially Observed Markov Processes using Automatic Differentiation

JSM25: Innovative Methodologies for Spatiotemporal Modeling and Inference

Edward Ionides
University of Michigan

Coauthors: Kevin Tan and Giles Hooker

Other pypomp developers: Jesse Wheeler, Jun Chen, Bo Yang, Aaron Abkemeier, Aaron King, Kunyang He

August 4, 2025

Spatiotemporal partially observed Markov process (SpatPOMP) models

Likelihood-based inference for nonlinear non-Gaussian SpatPOMP models

- Asfaw, Park, King & Ionides (2024). spatPomp: An R package for spatiotemporal partially observed Markov process models. *Journal of Open Source Software.*
- Ning & Ionides (2023). Iterated block particle filter for high-dimensional parameter learning: Beating the curse of dimensionality. *Journal of Machine Learning Research.*
- Li, Ionides, King, Pascual & Ning (2024). Inference on spatiotemporal dynamics for coupled biological populations. *Journal of the Royal Society Interface*.
- Wheeler, Rosengart, Jiang, Tan, Treutle & Ionides (2024). Informing policy via dynamic models: Cholera in Haiti. *PLOS Computational Biology.*

Outline

- What is automatic differentiation (AD)?
- Why is AD important for statistical inference? Existing evidence.
- What has delayed the impact of AD for inference on nonlinear stochastic dynamic systems?
- AD for simulation-based inference on mechanistic models using particle filters and GPUs.

Introduction to automatic differentiation (AD)

- AD is numerical differentation where compiled code automatically computes a massive chain rule, taking advantage of exact expressions for the numerical derivative of basic operations ($+$, -, $\times$, $\log$, $\exp$, $\sin$, $\cos$, etc).
- AD does not involve symbolic algebra.
- Modern compilers make AD available for arbitrary programs.
- JAX, a Python extension, does AD with automatic parallelization for CPU or GPU.

Modern statistical methods powered by AD

- Deep learning (also uses GPU)
- Stan: Hamiltonian Markov chain Monte Carlo
- Prophet: automatic forecasting

Conclusion: AD has allowed growth in model compexity and data size.

Mechanistic POMP models and the particle filter (PF, a.k.a. sequential Monte Carlo, SMC)

- Partially observed Markov process (POMP) models provide a framework for mechanistic modeling of dynamic systems with noisy measurements.
- PF uses a POMP model to generate an ensemble of forecasts for each successive observation.
- When the observation is recorded, it is "assimilated" into the forecast by resampling the ensemble members with weight proportional to the probability of the observation given the forecast for that ensemble member.
- Remarkably, these iterated forecast and correction steps provide an unbiased estimate of the likelihood for the model.
- PF just needs a simulator from the dynamic model, not transition probabilities. It is plug-and-play.

Why is AD not standard for the particle filter (PF) likelihood?

- So, why doesn't everyone use AD for PF?
- Technical difficulties
  1. Resampling is discontinuous: discrete and so piecewise constant.
  2. The derivative estimate can be numerically unstable even when the log-likelihood estimate is stable.

Previous attempts at AD for PF

- Approximating the particle filter using deep learning.
- Ignoring the resampling when calculating the derivative.
- Obtaining a smooth particle filter at the cost of inferior scaling and loss of the plug-and-play property.
- AD for smooth functions estimated by Monte Carlo expectation of non-smooth functions has been widely studied. Applied to PF, previous methods have struggled with stability for a long time series.

We looked for a plug-and-play particle filter equipped with automatic differention that maintains the guarantee of unbiased likelihood evaluation and provides numerically stable derivate estimates.

Differentiated Measurement Off Parameter (DMOP) filter

- Numerically stable automatic derivatives, with controlled bias and variance, from a plug-and-play particle filter with unbiased likelihood evaluation (Tan et al., 2024).

Measurement Off-Parameter (MOP-α)

- A MOP particle filter at parameter θ is a smooth continuation of the filter at ϕ, fixing resampling decisions for ϕ and accounting for this via "off-parameter'' measurement weights.
- *Off-parameter* resampling is analogous to *off-policy* reinforcement learning.
- Log-measurement weights are discounted by a factor $\alpha = 1 - \epsilon$ to add stability. At $\theta = \phi$, log-measurement weights are all 0, so discounting has no effect.

DMOP-α is differentiated MOP-α

- The derivative of MOP-α at $\theta = \phi$ can be computed using AD applied to an algorithm similar to a basic particle filter.

Measurement off-parameter filter, MOP-α

First pass: Set $\theta = \phi$ and compute $g^{\phi}_{n,j}$.

Second pass: Set $\theta \neq \phi$ and use the same randon number seed

For $n = 1, ..., N$:

1. $w^{P,\theta}_{n,j} = \big(w^{F,\theta}_{n-1,j}\big)^{\alpha}$
2. $X^{P,\theta}_{n,j} \sim \texttt{process}_n\big(\,\cdot\,|X^{F,\theta}_{n-1,j};\theta\big)$.
3. $g^{\theta}_{n,j} = f_{Y_n|X_n}(y^*_n|X^{P,\theta}_{n,j};\theta)$.
4. **Likelihood:** $L^{\theta,\alpha}_n = \sum_{j=1}^{J} g^{\theta}_{n,j} w^{P,\theta}_{n,j} \Big/ \sum_{j=1}^{J} w^{P,\theta}_{n,j}$.
5. Draw $k_{1:J}$ with $P(k_j = m) \propto g^{\phi}_{n,m}$.
6. $X^{F,\theta}_{n,j} = X^{P,\theta}_{n,k_j}$.
7. $w^{F,\theta}_{n,j} = w^{P,\theta}_{n,k_j}\, g^{\theta}_{n,k_j} \Big/ g^{\phi}_{n,k_j}$.

Differentiated filter, DMOP-α

First pass: Evaluate and build computation graph

Second pass: Differentiate via the chain rule

For $n = 1, ..., N$:

1. $w^{P,\theta}_{n,j} = \left(w^{F,\theta}_{n-1,j}\right)^{\alpha}$
2. $X^{P,\theta}_{n,j} \sim \texttt{process}_n(\,\cdot\,|X^{F,\theta}_{n-1,j};\theta)$.
3. $g^{\theta}_{n,j} = f_{Y_n|X_n}(y^*_n|X^{P,\theta}_{n,j};\theta)$.
4. **Likelihood:** $L^{\theta,\alpha}_n = \sum_{j=1}^{J} g^{\theta}_{n,j} w^{P,\theta}_{n,j} \Big/ \sum_{j=1}^{J} w^{P,\theta}_{n,j}$.
5. Draw $k_{1:J}$ with $P(k_j = m) \propto g^{\theta}_{n,m}$.
6. $X^{F,\theta}_{n,j} = X^{P,\theta}_{n,k_j}$.
7. $w^{F,\theta}_{n,j} = w^{P,\theta}_{n,k_j}\, g^{\theta}_{n,k_j} \Big/ \texttt{stop\_gradient}(g^{\theta}_{n,k_j})$.

Comparing MOP-α and DMOP-α

- The two algorithms are similar
- DMOP-α only has θ, not ϕ.
- The role of ϕ in MOP-α is taken by the `stop_gradient` operator in DMOP-α
  - `stop_gradient(x)` evaluates to `x` on the forward pass but is not differentiated on the backward pass.

Outline of theory for MOP-α and DMOP-α

1. MOP-1 converges almost surely to the likelihood for all θ, as does MOP-α for $\theta = \phi$, as the number of particles, J, increases.
2. The derivative of MOP-1 provides a strongly consistent estimate of the log-likelihood derivative.
3. DMOP-α evaluates the derivative of MOP-α at $\theta = \phi$.
4. DMOP-α has a bias bounded linearly by the length, N, of the time series with a coefficient that decreases to zero as $\alpha \to 1$.
5. The variance of DMOP-α is $O(N^4/J)$ at $\alpha = 1$ but $O(N/J)$ for $\alpha < 1$.

Regularity conditions: 1 and 2 require bounded derivatives. 4 and 5 require a mixing property for the latent Markov process.

Software for DMOP: pypomp

First, an introduction to the R-pomp family of packages

- The pomp R package has provided a solid, principled, extendable framework for combining time series data with nonlinear stochastic partially observed mechanistic dynamic models.
- Extensions: panel time series data (panelPomp), spatiotemporal data (spatPomp), phylodynamic data (phyloPomp).
- Limitations: R is not convenient for AD or GPU computing. To incorporate these, it may be better to start afresh.

pypomp is a Python extension of pomp enabling AD and GPU.

- Current status: pypomp is in alpha development at https://github.com/pypomp. The brave are welcome to join.

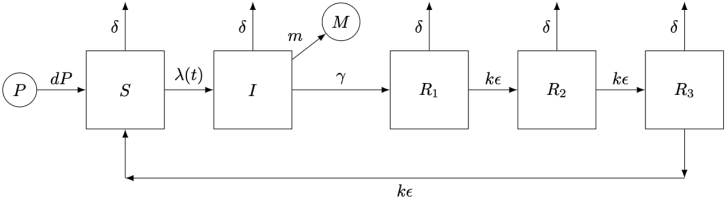
Why JAX?

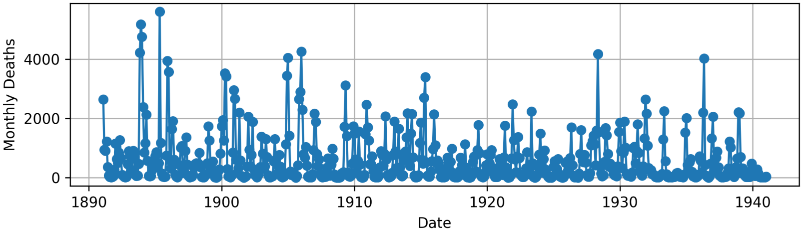
- Actively developed by Google: “Google researchers have built and trained models like Gemini and Gemma on JAX, and it’s also used by researchers for a wide range of advanced applications.” (https://io.google/2025/explore/technical-session-1)
- JAX code is Python, replacing numpy and scipy with jax.numpy and jax.scipy
- JAX composable transformations:
  - vmap: vectorization
  - grad: autodiff
  - jit: just-in-time compilation
- JAX uses XLA: accelerated linear algebra
  - distributes work across CPU/GPU/TPU cores

A previously difficult POMP likelihood maximization is easy with AD and quick/cheap with GPU

- We benchmark on an $\mathrm{SIR}^3\mathrm{S}$ model fitted to 50yr of monthly cholera mortality in historical Dacca, Bangladesh.
- Flexible modeling of seasonality and long-term trends lead to a highly parameterized model.
- Tractable, but difficult, using an early iterated filtering (IF1) algorithm in King et al. (2008). Possible with a large computing cluster and considerable dedication.
- Considerably easier, yet still hard, using the IF2 algorithm of Ionides et al. (2015).
- Here, we make a preliminary search with IF2, followed by stochastic gradient ascent with DMOP-α.

Model diagram (top) and data (bottom)





Results 1. Speed

- PF on 1 cpu core with 10^4 particles using R-pomp with the model compiled into C
  - 9.75 sec
- PF on a 10^4 core GPU with 10^4 particles using pypomp
  - 0.17 sec
- Here, a 10^4 core GPU is worth 57 CPU cores.
- This is deliberately indirect: we are also comparing compiled C with jit-compiled Python.
- Additional vectorization helps for replicated GPU evaluations
  - For 360 replications of iterated filtering, a GPU is worth 450 CPU cores.

Results 2. The value of AD

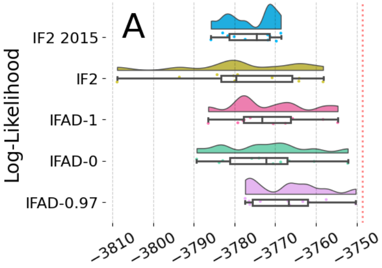


Figure 1: Performance of IFAD and IF2. **A:** the best out of every 10 runs. **B:** All searches. The dotted red line shows the true maximized log-likelihood.

Summary

- Remarkably, likelihood-based inference is possible from noisy measurements of complex, high-dimensional, nonlinear dynamic systems.
- AD and massively parallel computing will push capabilities to a new level.
- Next step: AD PF advances can be combined with SpatPOMP filtering advances for high-dimensional systems

Thank you!

References I

Ionides, E. L., Nguyen, D., Atchadé, Y., Stoev, S., and King, A. A. (2015). Inference for dynamic and latent variable models via iterated, perturbed Bayes maps. *Proceedings of the National Academy of Sciences of the USA*, 112:719–724.

King, A. A., Ionides, E. L., Pascual, M., and Bouma, M. J. (2008). Inapparent infections and cholera dynamics. *Nature*, 454:877–880.

Tan, K., Hooker, G., and Ionides, E. L. (2024). Accelerated inference for partially observed Markov processes using automatic differentiation. *arXiv:2407.03085*.