Pypomp: Inference for partially observed Markov process models in Python with JAX

With applications in epidemiology and elsewhere

Edward Ionides
University of Michigan

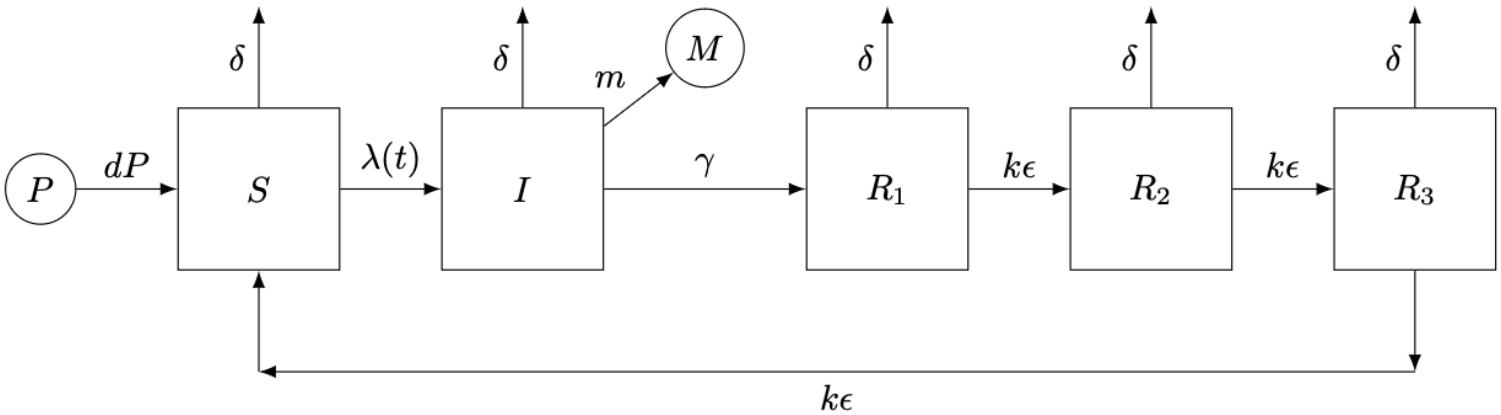
Coauthors: Aaron Abkemeier and the rest of the Pypomp develor team

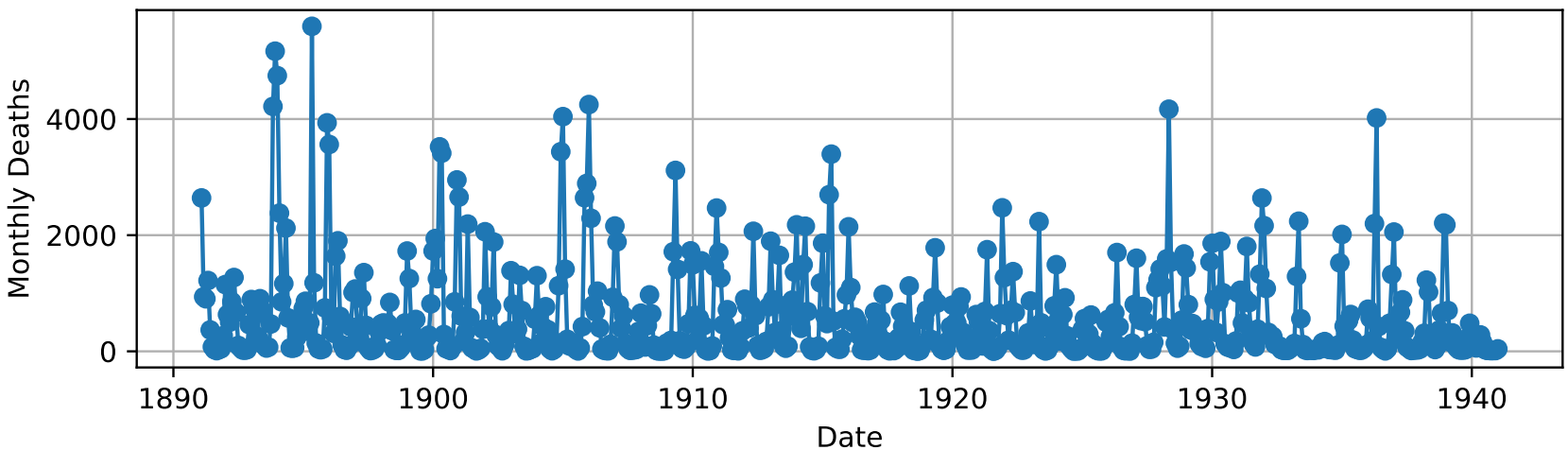
September 2, 2026

What is a partially observed Markov process (POMP) model?

- A formalization of a **compartment model** for dynamic systems.
- The latent state dynamics are stochastic and indirectly observed.
- We suppose the dynamics are independent of the past given the current state,i.e., **Markovian**. If necessary, we can include memory in the current state.
- A POMP can also be callled a hidden Markov models (HMM) or state space model (SSM).

POMP example: Diagram (top) and data (bottom) for historic cholera mortality in Dhaka, Bangladesh, via a Susceptible-Infected-Recovered (SIR) model





Why is epidemiological and ecological inference hard?

- **Complex, open, nonlinear, and non-stationary systems:**
  Precise "laws of nature" are unavailable.
  It is useful to model them as stochastic processes.
  Multiple competing explanations are possible.
- **Central scientific goals:**
  Which explanations are most favored by the data?
  Which kinds of data are most informative?
- **Central applied goals:**
  How to design ecological or epidemiological intervention?
  How to make accurate forecasts?
- **Time series** are particularly useful sources of data.

Obstacles to inference

*Obstacles for **ecological** modeling and inference via nonlinear mechanistic models enumerated by* Bjørnstad and Grenfell (2001):

1. Combining measurement noise and process noise.
2. Including covariates in mechanistically plausible ways.
3. Using continuous-time models.
4. Modeling and estimating interactions in coupled systems.
5. Dealing with unobserved variables.
6. Modeling spatial-temporal dynamics.

The same issues arise for **epidemiological** modeling and inference via nonlinear mechanistic models.

The *partially observed Markov process* (POMP) modeling framework addresses most of these problems effectively.

POMP case studies: Questions and answers I

1. How does one combine various data types to quantify asymptomatic COVID-19 infections? (Subramanian et al., 2021)
2. How effective have various non-pharmaceutical interventions been at controlling SARS-CoV-2 spread in hospitals? (Shirreff et al., 2022)
3. How does one use incidence and mobility data to infer key epidemiological parameters? (Andrade and Duggan, 2022)
4. How does one make forecasts for an outbreak of an emerging infectious disease? (King et al., 2015)
5. How does one build a system for real-time surveillance of COVID-19 using epidemiological and mobility data? (Fox et al., 2022)
6. What strategies are effective at containing mumps spread on college campuses? (Shah et al., 2022)

POMP case studies: Questions and answers II

7. What explains the resurgence of pertussis in countries with sustained high vaccine coverage? (Domenech de Cellès et al., 2018)
8. Do subclinical infections of pertussis play an important epidemiological role? (Lavine et al., 2013)
9. Can serotype-specific immunity explain the strain dynamics of human enteroviruses? (Pons-Salort and Grassly, 2018)
10. How does dynamic variation in individual sexual behavior contribute to the HIV epidemic? How does this compare to the role of heterogeneity between individuals? (Romero-Severson et al., 2015)
11. What is the contribution of adults to polio transmission? (Blake et al., 2014)
12. What explains the interannual variability of malaria? (Laneri et al., 2010)

POMP case studies: Questions and answers III

13. Can hydrology explain the seasonality of cholera? (Baracchini et al., 2017)
14. What roles are played by asymptomatic infection and waning immunity in cholera epidemics? (King et al., 2008)

Why Pypomp? Why new software for POMP models?

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 modern advances in computing:
  - Automatic differentation (AD)
  - Graphical processing unit (GPU) hardware

  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 available at https://github.com/pypomp. You are welcome to join the Pypomp community.

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
- AD Model Builder (ADMB) and Template Model Builder (TMB): random effect models in ecology and fisheries

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 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.

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

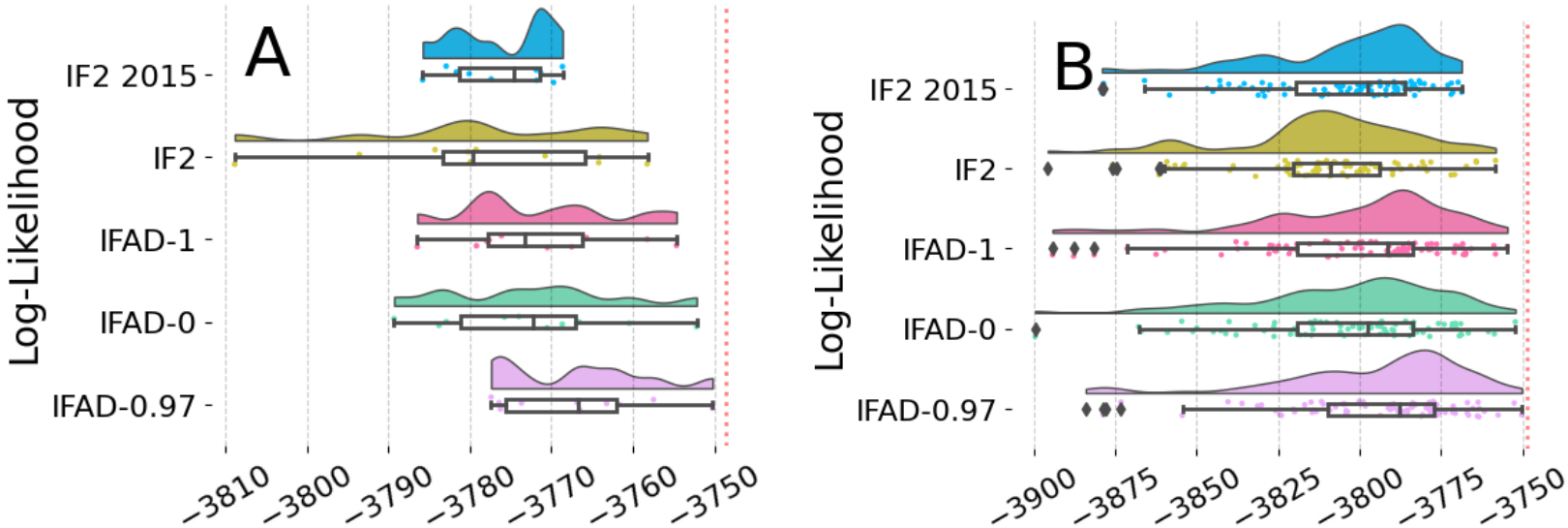


Figure 1: Performance of IFAD and IF2. **A:** the best out of every 10 runs. **B:** All searches. The dotted red line shows the true maximized log-likelihood.

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 differentiation that maintains the guarantee of unbiased likelihood evaluation and provides numerically stable derivative 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. $w^{P,\theta}_{n,j} = \left(w^{F,\theta}_{n-1,j}\right)^{\alpha}$
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:** $L^{\theta,\alpha}_n = \sum_{j=1}^{J} g^{\theta}_{n,j} w^{P,\theta}_{n,j} \,/ \sum_{j=1}^{J} w^{P,\theta}_{n,j}$.
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. $w^{F,\theta}_{n,j} = w^{P,\theta}_{n,k_j} \, g^{\theta}_{n,k_j} \,/\, g^{\phi}_{n,k_j}$.

Differentiated filter, DMOP-α

First pass: Evaluate and build computation graph

Second pass: Differentiate via the chain rule

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

1. $w_{n,j}^{P,\theta} = (w_{n-1,j}^{F,\theta})^{\alpha}$
2. $X_{n,j}^{P,\theta} \sim \texttt{process}_n(\,\cdot\,|X_{n-1,j}^{F,\theta}; \theta)$.
3. $g_{n,j}^{\theta} = f_{Y_n|X_n}(y_n^*|X_{n,j}^{P,\theta}; \theta)$.
4. **Likelihood:** $L_n^{\theta,\alpha} = \sum_{j=1}^{J} g_{n,j}^{\theta} w_{n,j}^{P,\theta} \,/ \sum_{j=1}^{J} w_{n,j}^{P,\theta}$.
5. Draw $k_{1:J}$ with $P(k_j = m) \propto g_{n,m}^{\theta}$.
6. $X_{n,j}^{F,\theta} = X_{n,k_j}^{P,\theta}$.
7. $w_{n,j}^{F,\theta} = w_{n,k_j}^{P,\theta} \, g_{n,k_j}^{\theta} \,/ \texttt{stop\_gradient}(g_{n,k_j}^{\theta})$.

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.

The Pypomp library

- Pypomp is a Python framework for building, simulating and fitting partially observed Markov process (POMP) models Abkemeier et al. (2025).
- Pypomp builds methodology for POMP models in terms of arbitrary user-specified POMP models.
- Pypomp provides tools, documentation, and examples to help users specify POMP models.
- Pypomp provides a platform for modification and sharing of models, data-analysis workflows, and methodological development.
- Pypomp is built on JAX for just-in-time compilation, automatic differentiation (AD) and CPU/GPU/TPU execution.

Structure of the pypomp package

We conventionally denote the `pypomp` package by `pp`:

```
import pypomp as pp
```

Suppose `mod` is a pypomp representation of a POMP model - `mod` has class `Pomp`. - `mod` is built using the `pp.Pomp` constructor.

It is useful to divide the `pypomp` package functionality into different levels:

- Basic model components: building `mod` to specify the desired POMP model
- Elementary POMP algorithms: investigating `mod` at a fixed set of parameters
- Inference algorithms: parameter estimation, model selection and diagnostics

Basic model components

- Basic model components are user-specified methods belonging to `mod` that perform the elementary computations that specify a POMP model. There are five of these:

`rinit`: simulator for the initial-state distribution, i.e., the distribution of the latent state at time t_0.

`rproc`: simulator for the process model.

`rmeas` and **`dmeas`**: simulator and density evaluation procedure, respectively, for the measurement model.

`par_trans`: parameter transformations. Optimization is helped by transforming parameters to remove constraints.

- The scientist must specify whichever basic model components are required for the algorithms that the scientist uses.

Classes for basic model components

`mod.rinit` has class `RInit`

`mod.dmeas` has class `DMeas`

`mod.rmeas` has class `RMeas`

`mod.rproc` has class `RProc`

`mod.par_trans` has class `ParTrans`

These methods are not vectorized, i.e., they evaluate the model properties for a single realization of the model.

They are designed to be vectorized via a call to JAX `vmap`, and this happens internally in pypomp methods.

Elementary POMP algorithms

These are methods for `mod` that use the basic components or the data to interrogate the confrontation without attempting to estimate parameters. There are currently two of these:

`simulate` performs simulations of the POMP model, i.e., it samples from the joint distribution of latent states and observables.

`pfilter` runs a sequential Monte Carlo (particle filter) algorithm to compute the likelihood and (optionally) estimate the prediction and filtering distributions of the latent state process.

POMP inference algorithms

These are methods for `Pomp` models that call the elementary algorithms and are used for estimation of parameters and other inferential tasks. There are currently three of these:

`mif`: Likelihood maximization via the IF2 iterated filtering algorithm.

`dmop`: An AD gradient estiamte using a "measurement off-parameter" particle filter.

`train`: Likelihood maximization based on `dmop` or related strategies.

Summary

- Remarkably, likelihood-based inference is possible from noisy measurements of complex nonlinear stochastic partially observed dynamic systems.
- AD and massively parallel computing will push capabilities to a new level.
- Pypomp provides a software implementation for modern POMP data analysis.
- Some useful links
  - A short course on epidemic modeling and inference using Pypomp: https://pypomp.github.io/tutorials/sbied
  - A full semester course on time series analysis, leading up to POMP modeling: https://ionides.github.io/531w26/
  - Various Pypomp repositories: https://github.com/pypomp

Thank you!

References I

Abkemeier, A. J., Chen, J., Tan, K., Wheeler, J., Yang, B., He, K., Terhorst, J., King, A. A., and Ionides, E. L. (2025). Pypomp: Inference for partially observed Markov process models in Python with JAX. https://github.com/pypomp.

Andrade, J. and Duggan, J. (2022). Inferring the effective reproductive number from deterministic and semi-deterministic compartmental models using incidence and mobility data. *PLoS Comput Biol*, 18(6):e1010206.

Baracchini, T., King, A. A., Bouma, M. J., Rodó, X., Bertuzzo, E., and Pascual, M. (2017). Seasonality in cholera dynamics: a rainfall-driven model explains the wide range of patterns in endemic areas. *Advances in Water Resources*, 108C:357–366.

Bjørnstad, O. N. and Grenfell, B. T. (2001). Noisy clockwork: Time series analysis of population fluctuations in animals. *Science*, 293:638–643.

References II

Blake, I. M., Martin, R., Goel, A., Khetsuriani, N., Everts, J., Wolff, C., Wassilak, S., Aylward, R. B., and Grassly, N. C. (2014). The role of older children and adults in wild poliovirus transmission. *Proceedings of the National Academy of Sciences of the USA*, 111(29):10604–10609.

Domenech de Cellès, M., Magpantay, F. M. G., King, A. A., and Rohani, P. (2018). The impact of past vaccination coverage and immunity on pertussis resurgence. *Sci Transl Med*, 10(434):eaaj1748.

Fox, S. J., Lachmann, M., Tec, M., Pasco, R., Woody, S., Du, Z., Wang, X., Ingle, T. A., Javan, E., Dahan, M., Gaither, K., Escott, M. E., Adler, S. I., Johnston, S. C., Scott, J. G., and Meyers, L. A. (2022). Real-time pandemic surveillance using hospital admissions and mobility data. *Proc Natl Acad Sci*, 119(7):e2111870119.

References III

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., Domenech de Cellès, M., Magpantay, F. M. G., and Rohani, P. (2015). Avoidable errors in the modelling of outbreaks of emerging pathogens, with special reference to Ebola. *Proc R Soc Lond B*, 282(1806):20150347.

King, A. A., Ionides, E. L., Pascual, M., and Bouma, M. J. (2008). Inapparent infections and cholera dynamics. *Nature*, 454(7206):877–880.

Laneri, K., Bhadra, A., Ionides, E. L., Bouma, M., Dhiman, R. C., Yadav, R. S., and Pascual, M. (2010). Forcing versus feedback: Epidemic malaria and monsoon rains in NW India. *PLoS Computational Biology*, 6(9):e1000898.

References IV

Lavine, J. S., King, A. A., Andreasen, V., and Bjørnstad, O. N. (2013). Immune boosting explains regime-shifts in prevaccine-era pertussis dynamics. *PLoS ONE*, 8(8):e72086.

Pons-Salort, M. and Grassly, N. C. (2018). Serotype-specific immunity explains the incidence of diseases caused by human enteroviruses. *Science*, 361(6404):800–803.

Romero-Severson, E. O., Volz, E., Koopman, J. S., Leitner, T., and Ionides, E. L. (2015). Dynamic variation in sexual contact rates in a cohort of HIV-negative gay men. *American Journal of Epidemiology*, 182(3):255–262.

Shah, M., Ferra, G., Fitzgerald, S., Barreira, P. J., Sabeti, P. C., and Colubri, A. (2022). Containing the spread of mumps on college campuses. *Royal Society Open Science*, 9(1):210948.

References V

Shirreff, G., Zahar, J.-R., Cauchemez, S., Temime, L., and Opatowski, L. (2022). Measuring basic reproduction number to assess effects of nonpharmaceutical interventions on nosocomial SARS-CoV-2 transmission. *Emerg Infect Dis*, 28(7):1345–1354.

Subramanian, R., He, Q., and Pascual, M. (2021). Quantifying asymptomatic infection and transmission of COVID-19 in New York City using observed cases, serology, and testing capacity. *Proc Natl Acad Sci*, 118(9):e2019716118.

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