The Journal of Open Source Software

Dynamax: A Python package for probabilistic state space modeling with JAX

Scott W. Linderman[1¶], **Peter Chang**[2], **Giles Harper-Donnelly**[3], **Aleyna Kara**[4], **Xinglong Li**[5], **Gerardo Duran-Martin**[6], and **Kevin Murphy**[7¶]

1 Department of Statistics and Wu Tsai Neurosciences Institute, Stanford University, USA **2** CSAIL, Massachusetts Institute of Technology, USA **3** Cambridge University, England, UK **4** Computer Science Department, Technical University of Munich Garching, Germany **5** Statistics Department, University of British Columbia, Canada **6** Queen Mary University of London, England, UK **7** Google DeepMind, USA
¶ Corresponding author

DOI: 10.21105/joss.07069

Software

- Review
- Repository
- Archive

Editor: Øystein Sørensen

Reviewers:

- @thomaspinder
- @gdalle

Submitted: 24 July 2024
Published: 26 March 2025

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

State space models (SSMs) are fundamental tools for modeling sequential data. They are broadly used across engineering disciplines like signal processing and control theory, as well as scientific domains like neuroscience (Vyas et al., 2020), genetics (Durbin et al., 1998), ecology (Patterson et al., 2008), computational ethology (Weinreb et al., 2024), economics (Jacquier et al., 2002), and climate science (Ott et al., 2004). Fast and robust tools for state space modeling are crucial to researchers in all of these application areas.

State space models specify a probability distribution over a sequence of observations, $y_1, \ldots y_T$, where y_t denotes the observation at time t. The key assumption of an SSM is that the observations arise from a sequence of *latent states*, $z_1, \ldots, z_T$, which evolve according to a *dynamics model* (aka transition model). An SSM may also use inputs (aka controls or covariates), $u_1, \ldots, u_T$, to steer the latent state dynamics and influence the observations. For example, in a neuroscience application from Vyas et al. (2020), y_t represents a vector of spike counts from ~ 1000 measured neurons, and z_t is a lower dimensional latent state that changes slowly over time and captures correlations among the measured neurons. If sensory inputs to the neural circuit are known, they can be encoded in u_t. In the computational ethology application of Weinreb et al. (2024), y_t represents a vector of 3D locations for several key points on an animal's body, and z_t is a discrete behavioral state that specifies how the animal's posture changes over time. In both examples, there are two main objectives: First, we aim to infer the latent states z_t that best explain the observed data; formally, this is called *state inference*. Second, we need to estimate the dynamics that govern how latent states evolve; formally, this is part of the *parameter estimation* process. `Dynamax` provides algorithms for state inference and parameter estimation in a variety of SSMs.

There are a few key design choices to make when constructing an SSM:

- What is the type of latent state? E.g., is z_t a continuous or discrete random variable?
- How do the latent states evolve over time? E.g., are the dynamics linear or nonlinear?
- How are the observations distributed? E.g., are they Gaussian, Poisson, etc.?

Some design choices are so common they have their own names. Hidden Markov models (HMM) are SSMs with discrete latent states, and linear dynamical systems (LDS) are SSMs with continuous latent states, linear dynamics, and additive Gaussian noise. `Dynamax` supports canonical SSMs and allows the user to construct bespoke models as needed, simply by inheriting from a base class and specifying a few model-specific functions. For example, see the *Creating Custom HMMs* tutorial in the Dynamax documentation.

Finally, even for canonical models, there are several algorithms for state inference and parameter estimation. Dynamax provides robust implementations of several low-level inference algorithms to suit a variety of applications, allowing users to choose among a host of models and algorithms for their application. More information about state space models and algorithms for state inference and parameter estimation can be found in the textbooks by Murphy (2023) and Särkkä & Svensson (2023).

Statement of need

Dynamax is an open-source Python package for state space modeling. Since it is built with JAX (Bradbury et al., 2018), it supports just-in-time (JIT) compilation for hardware acceleration on CPU, GPU, and TPU machines. It also supports automatic differentiation for gradient-based model learning. While other libraries exist for state space modeling in Python (Corenflos & Särkkä, 2021; Johnson, 2020; Linderman et al., 2020; Seabold & Perktold, 2010; Weiss et al., 2024) and Julia (Dalle, 2024), Dynamax provides a diverse combination of low-level inference algorithms and high-level modeling objects that can support a wide range of research applications in JAX. Additionally, Dynamax implements parallel message passing algorithms that leverage the associative scan (a.k.a., parallel scan) primitive in JAX to take full advantage of modern hardware accelerators. Currently, these primitives are not natively supported in other frameworks like PyTorch. While various subsets of these models and algorithms may be found in other libraries, Dynamax is a "one stop shop" for state space modeling in JAX.

The API for Dynamax is divided into two parts: a set of core, functionally pure, low-level inference algorithms, and a high-level, object oriented module for constructing and fitting probabilistic SSMs. The low-level inference API provides message passing algorithms for several common types of SSMs. For example, Dynamax provides JAX implementations for:

- Forward-Backward algorithms for discrete-state hidden Markov models (HMMs),
- Kalman filtering and smoothing algorithms for linear Gaussian SSMs,
- Extended and unscented generalized Kalman filtering and smoothing for nonlinear and/or non-Gaussian SSMs, and
- Parallel message passing routines that leverage GPU or TPU acceleration to perform message passing in $O(\log T)$ time on a parallel machine (Hassan et al., 2021; Särkkä & García-Fernández, 2020; Stone, 1975). Note that these routines are not simply parallelizing over batches of time series, but rather using a parallel algorithm with sublinear depth or span.

The high-level model API makes it easy to construct, fit, and inspect HMMs and linear Gaussian SSMs. Finally, the online Dynamax documentation and tutorials provide a wealth of resources for state space modeling experts and newcomers alike.

Dynamax has supported several publications. The low-level API has been used in machine learning research (Chang et al., 2023; Lee et al., 2023; Zhao & Linderman, 2023). Special purpose libraries have been built on top of Dynamax, like the Keypoint-MoSeq library for modeling animal behavior (Weinreb et al., 2024) and the Structural Time Series in JAX library, sts-jax (Li & Murphy, 2022). Finally, the Dynamax tutorials are used as reference examples in a major machine learning textbook (Murphy, 2023).

Acknowledgements

A significant portion of this library was developed while S.W.L. was a Visiting Faculty Researcher at Google and P.C., G.H.D., A.K., and X.L. were Google Summer of Code participants.

References

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). http://github.com/google/jax

Chang, P. G., Durán-Martín, G., Shestopaloff, A., Jones, M., & Murphy, K. P. (2023). Low-rank extended Kalman filtering for online learning of neural networks from streaming data. In S. Chandar, R. Pascanu, H. Sedghi, & D. Precup (Eds.), *Proceedings of the 2nd conference on lifelong learning agents* (Vol. 232, pp. 1025–1071). PMLR. https://doi.org/10.48550/arXiv.2305.19535

Corenflos, A., & Särkkä, S. (2021). *Code companion for Bayesian Filtering and Smoothing* (Version 1.0). https://github.com/EEA-sensors/Bayesian-Filtering-and-Smoothing

Dalle, G. (2024). HiddenMarkovModels.jl: Generic, fast and reliable state space modeling. *Journal of Open Source Software*, *9*(96), 6436. https://doi.org/10.21105/joss.06436

Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. https://doi.org/10.1017/cbo9780511790492

Hassan, S. S., Särkkä, S., & García-Fernández, Á. F. (2021). Temporal parallelization of inference in hidden Markov models. *IEEE Transactions on Signal Processing*, *69*, 4875–4887. https://doi.org/10.1109/TSP.2021.3103338

Jacquier, E., Polson, N. G., & Rossi, P. E. (2002). Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics*, *20*(1), 69–87. https://doi.org/10.1198/073500102753410408

Johnson, M. J. (2020). *PyHSMM: Bayesian inference in HSMMs and HMMs* (Version 0.0.0). https://github.com/mattjj/pyhsmm

Lee, H. D., Warrington, A., Glaser, J., & Linderman, S. (2023). Switching autoregressive low-rank tensor models. *Advances in Neural Information Processing Systems*, *36*, 57976–58010. https://doi.org/10.48550/arXiv.2306.03291

Li, X., & Murphy, K. (2022). *Structural time series (STS) in JAX*. https://github.com/probml/sts-jax

Linderman, S., Antin, B., Zoltowski, D., & Glaser, J. (2020). *SSM: Bayesian Learning and Inference for State Space Models* (Version 0.0.1). https://github.com/lindermanlab/ssm

Murphy, K. P. (2023). *Probabilistic machine learning: Advanced topics*. MIT Press. http://probml.github.io/book2

Ott, E., Hunt, B. R., Szunyogh, I., Zimin, A. V., Kostelich, E. J., Corazza, M., Kalnay, E., Patil, D., & Yorke, J. A. (2004). A local ensemble Kalman filter for atmospheric data assimilation. *Tellus A: Dynamic Meteorology and Oceanography*, *56*(5), 415–428. https://doi.org/10.3402/tellusa.v56i5.14462

Patterson, T. A., Thomas, L., Wilcox, C., Ovaskainen, O., & Matthiopoulos, J. (2008). State-space models of individual animal movement. *Trends in Ecology & Evolution*, *23*(2), 87–94. https://doi.org/10.1016/j.tree.2007.10.009

Särkkä, S., & García-Fernández, Á. F. (2020). Temporal parallelization of Bayesian smoothers. *IEEE Transactions on Automatic Control*, *66*(1), 299–306. https://doi.org/10.1109/TAC.2020.2976316

Särkkä, S., & Svensson, L. (2023). *Bayesian filtering and smoothing* (Vol. 17). Cambridge University Press. https://doi.org/10.1017/CBO9781139344203

Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*. https://doi.org/10.25080/majora-92bf1922-011

Stone, H. S. (1975). Parallel tridiagonal equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, *1*(4), 289–307. https://doi.org/10.1145/355656.355657

Vyas, S., Golub, M. D., Sussillo, D., & Shenoy, K. V. (2020). Computation through neural population dynamics. *Annual Review of Neuroscience*, *43*(1), 249–275. https://doi.org/10.1146/annurev-neuro-092619-094115

Weinreb, C., Pearl, J. E., Lin, S., Osman, M. A. M., Zhang, L., Annapragada, S., Conlin, E., Hoffmann, R., Makowska, S., Gillis, W. F., Jay, M., Ye, S., Mathis, A., Mathis, M. W., Pereira, T., Linderman, S. W., & Datta, S. R. (2024). Keypoint-MoSeq: Parsing behavior by linking point tracking to pose dynamics. *Nature Methods*, *21*(7), 1329–1339. https://doi.org/10.1038/s41592-024-02318-2

Weiss, R., Du, S., Grobler, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Mueller, A., Thirion, B., Nouri, D., Louppe, G., Vanderplas, J., Benediktsson, J., Buitinck, L., Korobov, M., McGibbon, R., Lattarini, S., Niculae, V., Gramfort, A., Lebedev, S., … Rockhill, A. (2024). *Hmmlearn* (Version 0.3.2). https://github.com/hmmlearn/hmmlearn

Zhao, Y., & Linderman, S. (2023). Revisiting structured variational autoencoders. *International Conference on Machine Learning*, 42046–42057. https://doi.org/10.48550/arXiv.2305.16543