




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

Scott W. Linderman^{1,2¶}, Peter Chang³, Giles Harper-Donnelly⁴, Aleyna Kara⁵, Xinglong Li⁶, and Kevin Murphy^{2¶}

¹ Department of Statistics and Wu Tsai Neurosciences Institute, Stanford University, USA ² Google Research, USA ³ CSAIL, Massachusetts Institute of Technology, USA ⁴ Cambridge University, UK ⁵ Boğaziçi University, Turkey ⁶ University of British Columbia, Canada ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

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, genetics, ecology, and climate science. 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, \dots, 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, \dots, z_T , which evolve according to a *dynamics model* (aka transition model). An SSM may also use inputs (aka controls or covariates), u_1, \dots, u_T , to steer the latent state dynamics and influence the observations.

For example, SSMs are often used in neuroscience to model the dynamics of neural spike train recordings (Vyas et al., 2020). Here, y_t is a vector of spike counts from each of, say, 100 measured neurons. The activity of nearby neurons is often correlated, and SSMs can capture that correlation through a lower dimensional latent state, z_t , which may change slowly over time. If we know that certain sensory inputs may drive the neural activity, we can encode them in u_t . A common goal in neuroscience is to infer the latent states z_t that best explain the observed neural spike train; formally, this is called *state inference*. Another goal is 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.

The key design choices when constructing an SSM include the type of latent state (is z_t a continuous or discrete random variable?), the dynamics that govern how latent states evolve over time (are they linear or nonlinear?), and the conditional distribution of the observations (are they Gaussian, Poisson, etc?). Canonical examples of SSMs include hidden Markov models (HMM), which have discrete latent states, and linear dynamical systems (LDS), which have continuous latent states with linear dynamics and additive Gaussian noise. Dynamax supports these canonical examples as well as more complex models.

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; Duran-Martin et al., n.d.; Johnson, 2020; Linderman et al., 2020; Seabold & Perktold, 2010; Weiss et al., 2024; ?), some of which use JAX, this library provides a unique combination of low-level inference algorithms and high-level modeling objects that can support a wide range of research applications.

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 Kalman filtering and smoothing for nonlinear Gaussian SSMs,
- Conditional moment filtering and smoothing algorithms for models with non-Gaussian emissions, and
- Parallel message passing routines that leverage GPU or TPU acceleration to perform message passing in sublinear time.

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). More sophisticated, special purpose models on top of Dynamax, like the Keypoint-MoSeq library for modeling postural dynamics of animals (Weinreb et al., 2024). 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.
- Corenflos, A., & Särkkä, S. (2021). *Code companion for Bayesian Filtering and Smoothing* (Version 1.0). <https://github.com/EEA-sensors/Bayesian-Filtering-and-Smoothing>
- Duran-Martin, G., Murphy, K., & Kara, A. (n.d.). *JSL: JAX State-Space models (SSM) Library*. <https://github.com/probml/JSL>
- 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.

- 85 Linderman, S., Antin, B., Zoltowski, D., & Glaser, J. (2020). *SSM: Bayesian Learning and*
86 *Inference for State Space Models* (Version 0.0.1). <https://github.com/lindermanlab/ssm>
- 87 Murphy, K. P. (2023). *Probabilistic machine learning: Advanced topics*. MIT Press. [http://](http://probml.github.io/book2)
88 probml.github.io/book2
- 89 Särkkä, S., & Svensson, L. (2023). *Bayesian filtering and smoothing* (Vol. 17). Cambridge
90 University Press.
- 91 Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with
92 python. *9th Python in Science Conference*.
- 93 Vyas, S., Golub, M. D., Sussillo, D., & Shenoy, K. V. (2020). Computation through neural
94 population dynamics. *Annual Review of Neuroscience*, 43(1), 249–275.
- 95 Weinreb, C., Pearl, J. E., Lin, S., Osman, M. A. M., Zhang, L., Annapragada, S., Conlin,
96 E., Hoffmann, R., Makowska, S., Gillis, W. F., Jay, M., Ye, S., Mathis, A., Mathis, M.
97 W., Pereira, T., Linderman, S. W., & Datta, S. R. (2024). Keypoint-MoSeq: Parsing
98 behavior by linking point tracking to pose dynamics. *Nature Methods*, 21(7), 1329–1339.
99 ISBN: 1548-7105
- 100 Weiss, R., Du, S., Grobler, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Mueller, A.,
101 Thirion, B., Nouri, D., Louppe, G., Vanderplas, J., Benediktsson, J., Buitinck, L., Korobov,
102 M., McGibbon, R., Lattarini, S., Niculae, V., Gramfort, A., Lebedev, S., ... Rockhill, A.
103 (2024). *Hmmlearn* (Version 0.3.2). <https://github.com/hmmlearn/hmmlearn>
- 104 Zhao, Y., & Linderman, S. (2023). Revisiting structured variational autoencoders. *International*
105 *Conference on Machine Learning*, 42046–42057.

DRAFT