

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

Scott W. Linderman^{1¶}, 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 DeepMind, USA ³ CSAIL, Massachusetts Institute of Technology, USA ⁴ Cambridge University, UK ⁵ Computer Science Department, Technical University of Munich Garching, Germany ⁶ Statistics Department, 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Published: unpublished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 (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, \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, 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?
- What dynamics govern how latent states evolve over time? E.g., are they linear or nonlinear?
- What is the conditional distribution of the observations given the latent states? E.g., are they Gaussian, Poisson, etc.?

Some design choices are so common that they have their own names. For example, 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 these canonical SSMs and allows you to construct more complex models if needed.

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

49 Statement of need

50 Dynamax is an open-source Python package for state space modeling. Since it is built with JAX
51 (Bradbury et al., 2018), it supports just-in-time (JIT) compilation for hardware acceleration on
52 CPU, GPU, and TPU machines. It also supports automatic differentiation for gradient-based
53 model learning. While other libraries exist for state space modeling in Python (Corenflos &
54 Särkkä, 2021; Johnson, 2020; Linderman et al., 2020; Seabold & Perktold, 2010; Weiss et al.,
55 2024), some using JAX (Duran-Martin et al., 2022), Dynamax provides a unique combination of
56 low-level inference algorithms and high-level modeling objects that can support a wide range
57 of research applications.

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

- 62 ■ Forward-Backward algorithms for discrete-state hidden Markov models (HMMs),
- 63 ■ Kalman filtering and smoothing algorithms for linear Gaussian SSMs,
- 64 ■ Extended and unscented generalized Kalman filtering and smoothing for nonlinear and/or
- 65 non-Gaussian SSMs, and
- 66 ■ Parallel message passing routines that leverage GPU or TPU acceleration to perform
- 67 message passing in sublinear time (Hassan et al., 2021; Särkkä & García-Fernández,
- 68 2020; Stone, 1975).

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

72 Dynamax has supported several publications. The low-level API has been used in machine
73 learning research (Chang et al., 2023; Lee et al., 2023; Zhao & Linderman, 2023). More
74 sophisticated, special purpose models on top of Dynamax, like the Keypoint-MoSeq library for
75 modeling postural dynamics of animals (Weinreb et al., 2024). Finally, the Dynamax tutorials
76 are used as reference examples in a major machine learning textbook (Murphy, 2023).

77 Acknowledgements

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

80 References

- 81 Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G.,
82 Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable*
83 *transformations of Python+NumPy programs* (Version 0.3.13). <http://github.com/google/jax>
84
- 85 Chang, P. G., Durán-Martín, G., Shestopaloff, A., Jones, M., & Murphy, K. P. (2023). Low-rank
86 extended Kalman filtering for online learning of neural networks from streaming data. In S.

- 87 Chandar, R. Pascanu, H. Sedghi, & D. Precup (Eds.), *Proceedings of the 2nd conference*
88 *on lifelong learning agents* (Vol. 232, pp. 1025–1071). PMLR.
- 89 Corenflos, A., & Särkkä, S. (2021). *Code companion for Bayesian Filtering and Smoothing*
90 (Version 1.0). <https://github.com/EEA-sensors/Bayesian-Filtering-and-Smoothing>
- 91 Duran-Martin, G., Murphy, K., & Kara, A. (2022). *JSL: JAX State-Space models (SSM)*
92 *Library*. <https://github.com/probml/JSL>
- 93 Durbin, R., Eddy, S. R., Krogh, A., & Mitchison, G. (1998). *Biological sequence analysis:*
94 *Probabilistic models of proteins and nucleic acids*.
- 95 Hassan, S. S., Särkkä, S., & García-Fernández, Á. F. (2021). Temporal parallelization of
96 inference in hidden Markov models. *IEEE Transactions on Signal Processing*, 69, 4875–4887.
- 97 Jacquier, E., Polson, N. G., & Rossi, P. E. (2002). Bayesian analysis of stochastic volatility
98 models. *Journal of Business & Economic Statistics*, 20(1), 69–87.
- 99 Johnson, M. J. (2020). *PyHSMM: Bayesian inference in HSMMs and HMMs* (Version 0.0.0).
100 <https://github.com/mattjj/pyhsmm>
- 101 Lee, H. D., Warrington, A., Glaser, J., & Linderman, S. (2023). Switching autoregressive low-
102 rank tensor models. *Advances in Neural Information Processing Systems*, 36, 57976–58010.
- 103 Linderman, S., Antin, B., Zoltowski, D., & Glaser, J. (2020). *SSM: Bayesian Learning and*
104 *Inference for State Space Models* (Version 0.0.1). <https://github.com/lindermanlab/ssm>
- 105 Murphy, K. P. (2023). *Probabilistic machine learning: Advanced topics*. MIT Press. [http://](http://probml.github.io/book2)
106 probml.github.io/book2
- 107 Ott, E., Hunt, B. R., Szunyogh, I., Zimin, A. V., Kostelich, E. J., Corazza, M., Kalnay, E.,
108 Patil, D., & Yorke, J. A. (2004). A local ensemble Kalman filter for atmospheric data
109 assimilation. *Tellus A: Dynamic Meteorology and Oceanography*, 56(5), 415–428.
- 110 Patterson, T. A., Thomas, L., Wilcox, C., Ovaskainen, O., & Matthiopoulos, J. (2008).
111 State-space models of individual animal movement. *Trends in Ecology & Evolution*, 23(2),
112 87–94.
- 113 Särkkä, S., & García-Fernández, Á. F. (2020). Temporal parallelization of Bayesian smoothers.
114 *IEEE Transactions on Automatic Control*, 66(1), 299–306.
- 115 Särkkä, S., & Svensson, L. (2023). *Bayesian filtering and smoothing* (Vol. 17). Cambridge
116 University Press.
- 117 Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with
118 python. *9th Python in Science Conference*.
- 119 Stone, H. S. (1975). Parallel tridiagonal equation solvers. *ACM Transactions on Mathematical*
120 *Software (TOMS)*, 1(4), 289–307.
- 121 Vyas, S., Golub, M. D., Sussillo, D., & Shenoy, K. V. (2020). Computation through neural
122 population dynamics. *Annual Review of Neuroscience*, 43(1), 249–275.
- 123 Weinreb, C., Pearl, J. E., Lin, S., Osman, M. A. M., Zhang, L., Annapragada, S., Conlin,
124 E., Hoffmann, R., Makowska, S., Gillis, W. F., Jay, M., Ye, S., Mathis, A., Mathis, M.
125 W., Pereira, T., Linderman, S. W., & Datta, S. R. (2024). Keypoint-MoSeq: Parsing
126 behavior by linking point tracking to pose dynamics. *Nature Methods*, 21(7), 1329–1339.
127 ISBN: 1548-7105
- 128 Weiss, R., Du, S., Grobler, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Mueller, A.,
129 Thirion, B., Nouri, D., Louppe, G., Vanderplas, J., Benediktsson, J., Buitinck, L., Korobov,
130 M., McGibbon, R., Lattarini, S., Niculae, V., Gramfort, A., Lebedev, S., ... Rockhill, A.
131 (2024). *Hmmlearn* (Version 0.3.2). <https://github.com/hmmlearn/hmmlearn>

¹³² Zhao, Y., & Linderman, S. (2023). Revisiting structured variational autoencoders. *International*
¹³³ *Conference on Machine Learning*, 42046–42057.

DRAFT