The Journal of Open Source Software

StateSpaceDynamics.jl: A Julia package for probabilistic state space models (SSMs)

Ryan Senne [1,2], **Zachary Loschinskey** [1], **Carson Loughridge** [1], **James Fourie** [1], **and Brian D. DePasquale** [1,2]

1 Department of Biomedical Engineering, Boston University **2** Graduate Program for Neuroscience, Boston University

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

State-space models (SSMs) are powerful tools for modeling time series data that naturally arise in a variety of domains, including neuroscience, finance, and engineering. The unifying principle of these models is they assume an observation sequence, $Y_1, Y_2, ..., Y_T$, is generated through an underlying latent sequence, $X_1, X_2, ..., X_T$. This framework encompasses two popular models for time series analysis: the hidden Markov model (HMM) and the (Gaussian) linear dynamical system (LDS, i.e., the Kalman filter). Thus, SSMs provide a probabilistic framework for describing the temporal evolution of many phenomena, and their generality naturally leads to a variety of use cases. We introduce `StateSpaceDynamics.jl` (Senne et al., 2025), an open-source, modular package designed to be fast, readable, and self-contained for the purpose of easily fitting a plurality of SSMs in the Julia language.

Statement of need

Advances in neuroscience have enabled the collection of massive, multivariate, and complex time-series datasets, where simultaneous observations from hundreds to thousands of neurons are increasingly common. Interpreting these high-dimensional datasets presents significant challenges. Recent modeling approaches suggest that neural activity can be characterized by a set of latent factors evolving within a low-dimensional manifold. Consequently, there is a growing need for models that combine dimensionality reduction with temporal dynamics, for which state-space models provide a natural framework.

While state-space model implementations exist in Python, such as the SSM package (S. Linderman, 2022) and Dynamax (Chang et al., 2024), the Julia programming language lacks an equivalent that meets the needs of modern neuroscience. Existing Julia offerings, like `StateSpaceModels.jl` (Bodin, Guilherme, 2024), can accommodate continuous-state SSMs (e.g., LDS) but are limited to Gaussian observation models and rely on analytical calculation of the marginal log-likelihood. This latter limitation precludes model inference and parameter learning for non-conjugate observations which are common in neuroscience, where neural activity follow Poisson or other discrete distributions. Packages for performing inference and learning using sampling-based methods exist in Julia (such as `Turing.jl` (Ge, H., 2018)) but are computationally inefficient compared to tailored approaches based on Expectation-Maximization (EM). For discrete SSMs, an existing Julia offering, `HiddenMarkovModels.jl` (Dalle, 2024), is efficient and scalable but not intentionally designed with the functionality for mixing models that contain both discrete and continuous latent variables, such as the switching linear dynamical system model (SLDS) (Ghahramani & Hinton, 2000; S. W. Linderman et al., 2016) increasingly used in neuroscience.

The Journal of Open Source Software

Package design

To address these limitations, we developed `StateSpaceDynamics.jl`, which provides a flexible framework for fitting a variety of SSMs–including non-Gaussian observation models and models that mix discrete and continuous latents–while maintaining computational efficiency.

For continuous latent-variable models, (e.g., LDS) `StateSpaceDynamics.jl` employs a previously advocated approach of directly maximizing the complete-data log-likelihood with respect to the hidden state path (Paninski et al., 2010). By leveraging the block-tridiagonal structure of the Hessian matrix, this method allows for the exact computation of the Kalman smoother in O(T) time (Paninski et al., 2010). Furthermore, it facilitates the generalization of the Rauch–Tung–Striebel (RTS) smoother to accommodate other observation models (e.g., Poisson and Bernoulli), requiring only the computation of the gradient and Hessian of the new model to obtain an *exact* maximum a posteriori (MAP) path (Macke et al., 2011).

Using analytically computable Hessians, `StateSpaceDynamics.jl` performs approximate EM for non-Gaussian models via Laplace approximation of the latent posterior. Speed is maintained by using fast inversion algorithms of the negative Hessian (i.e., Fisher Information Matrix), which are block-tridiagonal (Rybicki & Hummer, 1990). From here `StateSpaceDynamics.jl` computes the approximate second moments of the posterior i.e., Cov(X_t, X_t) and Cov(X_t, X_{t-1}), and uses the analytical updates of the canonical LDS (Bishop, 2006; Paninski et al., 2010). It is important to note that when the observations and state-evolution process are assumed to have Gaussian errors, this approach is exactly the same as using the standard Kalman Filter and RTS-Smoother, i.e., they will give the same results.

Lastly, `StateSpaceDynamics.jl` provides implementations of discrete state-space models i.e., hidden Markov models, and the ability to fit these models using EM. While this is not the primary development target of the package, these models are necessary for the development of hierarchical models that mix discrete and continuous latents, e.g., the switching LDS (SLDS) and the recurrent switching LDS (rSLDS) (Ghahramani & Hinton, 2000; S. W. Linderman et al., 2016; Murphy, 1998) which have become immensely popular in neuroscience and require similarly tailored computational routines for efficient inference and learning. To illustrate the functionality of `StateSpaceDynamics.jl` for this model class, we include an implementation of the SLDS fit via structured variational EM (vEM) (Ghahramani & Hinton, 2000). The development of `HiddenMarkovModels.jl`, may make our approach to discrete model learning redundant, and future work may entail directly interfacing with this package (Dalle, 2024). Nonetheless, we provide a suite of HMM models popular in neuroscience including the classic Gaussian HMM and a variety of input-output HMMs (Bengio & Frasconi, 1994), commonly referred to as generalized linear model-HMMs (GLM-HMMs) (Ashwood et al., 2022) in neuroscience.

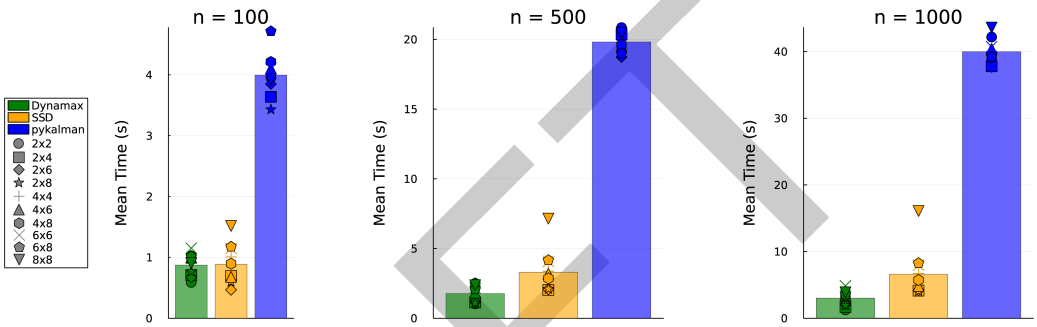
By providing these features, `StateSpaceDynamics.jl` fills a critical gap in the Julia ecosystem, offering modern computational neuroscientists the tools to model complex neural data with state-space models that incorporate both dimensionality reduction and temporal dynamics.

Benchmarks

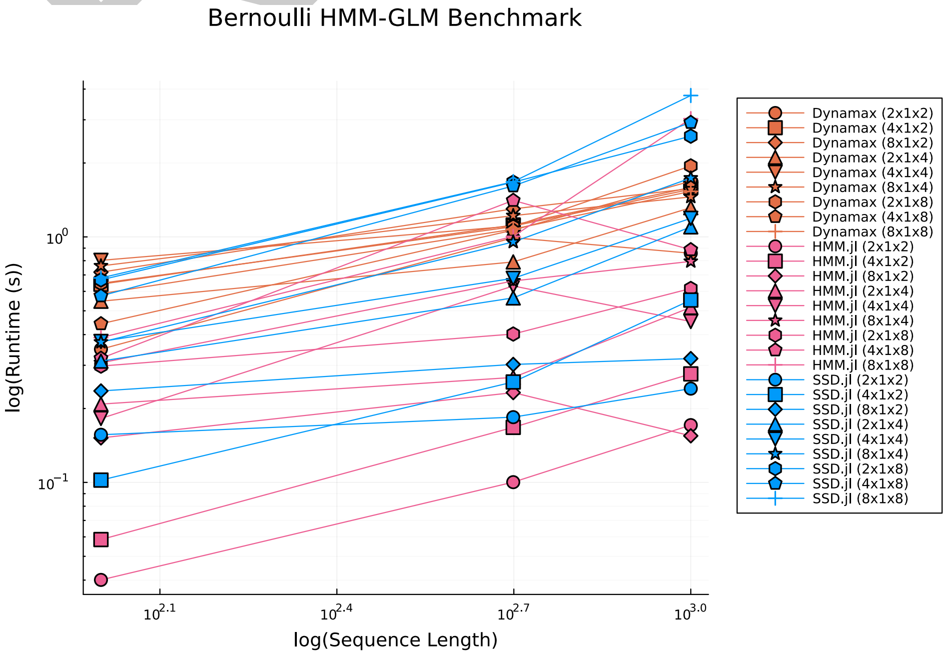
To evaluate the performance of `StateSpaceDynamics.jl`, we conducted two benchmarking studies focusing on fitting a Gaussian LDS and a Bernoulli GLM-HMM. For the Gaussian LDS benchmark, we compared our package against two alternatives: the NumPy-based Kalman filter-smoother package `pykalman` and the more recent JAX-based `Dynamax`. We intentionally excluded `StateSpaceModels.jl` from our comparison as its scope is geared towards structured time-series models. `Dynamax` was properly JIT-compiled by running a warm-up iteration before benchmarking to ensure fair comparison.

For our Gaussian LDS experiments, we constructed a synthetic dataset as follows. The state transition matrix A was generated as a random n-dimensional rotation matrix, while the observation matrix C was created as a random $m \times n$ matrix. Both the state noise covariance Q and observation noise covariance R were set to identity matrices. To ensure a fair comparison, all packages were initialized using identical random parameters, after which we executed the EM algorithm for 100 iterations. We conducted these benchmarks using `PythonCall.jl` (Doris, 2021) and `BenchmarkTools.jl` (Revels, 2017), with the assumption that Julia-to-Python overhead is negligible for these computationally intensive operations.

To thoroughly assess performance across different scales, we tested three sequence lengths ($T = 100, 500, 1000$) and explored various dimensionality combinations, with state dimensions $n = 2, 4, 8$ and observation dimensions $m = 2, 4, 8$.



For the subsequent benchmarking experiments, we compared `StateSpaceDynamics.jl`, `HiddenMarkovModels.jl`, and `Dynamax` in their abilities to fit a Bernoulli GLM-HMM. We completed the benchmarking on the same combinations of sequence lengths, state dimensions, and observation dimensions as in the Gaussian LDS benchmarking. Synthetic datasets were generated for each of these combinations by sampling from a randomly generated Bernoulli GLM-HMM using `StateSpaceDynamics.jl`. All packages were initialized to the same random parameters and the EM algorithm was run for 200 iterations. We disabled the convergence criteria to ensure each package completed all 200 iterations of EM. These benchmarks were conducted using `PythonCall.jl` and `BenchmarkTools.jl` to ensure timing accuracy.



In our benchmarking, we find that for the LDS, both variants of `StateSpaceDynamics.jl` are faster than `pykalman` across all sequence lengths and dimension configurations. More generally, `StateSpaceDynamics.jl` outperforms `Dynamax` at shorter sequence lengths across various dimensional settings. However, `Dynamax` exhibits superior scaling. In our current implementation, the Hessian is represented as a sparse matrix with block-tridiagonal structure, resulting in O(Tn^2) memory scaling — which is optimal. However, we do not yet exploit this structure fully during inference. In particular, our solver does not leverage specialized routines for block-banded systems, which can result in unnecessary fill-in and degraded performance at large T. Future versions will use banded or block-tridiagonal solvers to achieve truly linear-time inference.

In our GLM-HMM benchmarks, our package generally outperforms `Dynamax` at small to moderate sequence lengths. However, this trend does not hold for larger state dimensions, again suggesting that there is room to improve our scaling performance. `HiddenMarkovModels.jl` consistently performs best across a range of sequence lengths and dimensional configurations. As previously noted, our implementation of these models is intended primarily to provide a seamless interface with our continuous state-space models. These benchmarks underscore the potential for a future release that leverages direct interoperability with that package.

Taken together, these benchmarks demonstrate the competitiveness of `StateSpaceDynamics.jl` for fitting state-space models.

Availability

`StateSpaceDynamics.jl` is publicly available under the GNU license at https://github.com/depasquale-lab/StateSpaceDynamics.jl.

Conclusion

`StateSpaceDynamics.jl` fills an existing gap in the Julia ecosystem for general state-space modeling that exists in Python. Importantly, our package's approach is simple enough that other candidate state-space models can be easily implemented. Further, this work provides a foundation for future development of more advanced state-space models, such as the rSLDS, which are essential for modeling complex neural data. We expect that this package will be of interest to computational neuroscientists and other researchers working with high-dimensional time series data and we are currently using its functionality in three separate projects.

Author contributions

RS (Ryan Senne) was the primary developer of `StateSpaceDynamics.jl`, implementing the core algorithms, designing the package architecture, and writing the manuscript. ZL (Zachary Loschinskey) was the secondary developer, whose contributions include optimizing and extending HMM/GLM-HMM functionality, implementing core multi-trial EM algorithms, and assisting with sLDS development. CL (Carson Loughridge) and JF (James Fourie) contributed to package development, including implementation of key features, testing, and documentation. BDD (Brian D. DePasquale) conceived the project, provided theoretical guidance and technical oversight throughout development, secured funding, and supervised the work. All authors reviewed and approved the final manuscript.

Acknowledgements

This work was supported by the Biomedical Engineering Department at Boston University.

References

Ashwood, Z. C., Roy, N. A., Stone, I. R., Urai, A. E., Churchland, A. K., Pouget, A., & Pillow, J. W. (2022). Mice alternate between discrete strategies during perceptual decision-making. *Nature Neuroscience*, *25*(2), 201–212. https://doi.org/10.1038/s41593-021-01007-z

Bengio, Y., & Frasconi, P. (1994). An input output HMM architecture. In G. Tesauro, D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems* (Vol. 7). MIT Press. https://proceedings.neurips.cc/paper_files/paper/1994/file/8065d07da4a77621450aa84fee5656d9-Paper.pdf

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Bodin, Guilherme, S., Raphael. (2024). *StateSpaceModels.jl*. https://github.com/LAMPSPUC/StateSpaceModels.jl

Chang, P., Harper-Donnelly, G., Kara, A., Li, X., Linderman, S., & Murphy, K. (2024). *Dynamax: State space models library in JAX* [Computer software]. Github.

Dalle, G. (2024). *HiddenMarkovModels.jl*. https://github.com/gdalle/HiddenMarkovModels.jl

Doris, C. (2021). *PythonCall.jl*. https://github.com/JuliaPy/PythonCall.jl.

Ge, H., G. Z., Xu, K. (2018). *Turing.jl: Bayesian inference with probabilistic programming*. https://github.com/TuringLang/Turing.jl

Ghahramani, Z., & Hinton, G. E. (2000). Variational learning for switching state-space models. *Neural Computation*, *12*(4), 831–864. https://doi.org/10.1162/089976600300015619

Linderman, S. (2022). *SSM: Bayesian learning and inference for state space models*. https://github.com/lindermanlab/ssm

Linderman, S. W., Miller, A. C., Adams, R. P., Blei, D. M., Paninski, L., & Johnson, M. J. (2016). Recurrent switching linear dynamical systems. *arXiv [Stat.ML]*. https://doi.org/10.48550/arXiv.1610.08466

Macke, J. H., Buesing, L., Cunningham, J. P., Yu, B. M., Shenoy, K. V., & Sahani, M. (2011). *Empirical models of spiking in neural populations*. *24*. https://proceedings.neurips.cc/paper_files/paper/2011/file/7143d7fbadfa4693b9eec507d9d37443-Paper.pdf

Murphy, K. P. (1998). *Switching kalman filters* [Technical report].

Paninski, L., Ahmadian, Y., Ferreira, D. G., Koyama, S., Rahnama Rad, K., Vidne, M., Vogelstein, J., & Wu, W. (2010). A new look at state-space models for neural data. *J. Comput. Neurosci.*, *29*, 107–126. https://doi.org/10.1007/s10827-009-0179-x

Revels, J. (2017). *BenchmarkTools.jl*. https://github.com/JuliaCI/BenchmarkTools.jl.

Rybicki, G. B., & Hummer, D. G. (1990). *Fast solution for the diagonal elements of the inverse of a tridiagonal matrix*.

Senne, R., Loschinskey, Z., Loughridge, C., & DePasquale, B. (2025). *StateSpaceDynamics.jl: A julia package for probabilistic state space models*. https://github.com/depasquale-lab/StateSpaceDynamics.jl. https://github.com/depasquale-lab/StateSpaceDynamics.jl