

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

Ryan Senne^{1,2}, Zachary Loschinsky¹, James Fourie¹, Carson Loughridge¹, and Brian D. DePasquale^{1,2}

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

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

Software

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

Editor: [Evan Spotte-Smith](#)

Reviewers:

- [@gdalle](#)
- [@slinderman](#)
- [@THargreaves](#)

Submitted: 10 April 2025

Published: unpublished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 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, \dots, Y_T , is generated through an underlying Markovian latent sequence, X_1, X_2, \dots, 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` (Scott W. Linderman et al., 2025), the Julia programming language lacks an equivalent that meets the needs of modern neuroscience. Existing Julia offerings, like `StateSpaceModels.jl` (Saavedra et al., 2019), 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` (Fjelde et al., 2025; Ge et al., 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; Scott W. Linderman et al., 2016) increasingly used in neuroscience. Although our primary motivation arises from challenges in modeling high-dimensional neural population

activity, the package is not specific to neuroscience. The algorithms and abstractions apply equally well to time-series problems in engineering, econometrics, and other fields where latent variable models and structured inference are required.

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 $\mathcal{O}(T)$ 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., $\text{Cov}(X_t, X_t)$ and $\text{Cov}(X_t, X_{t-1})$, 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; Scott 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.

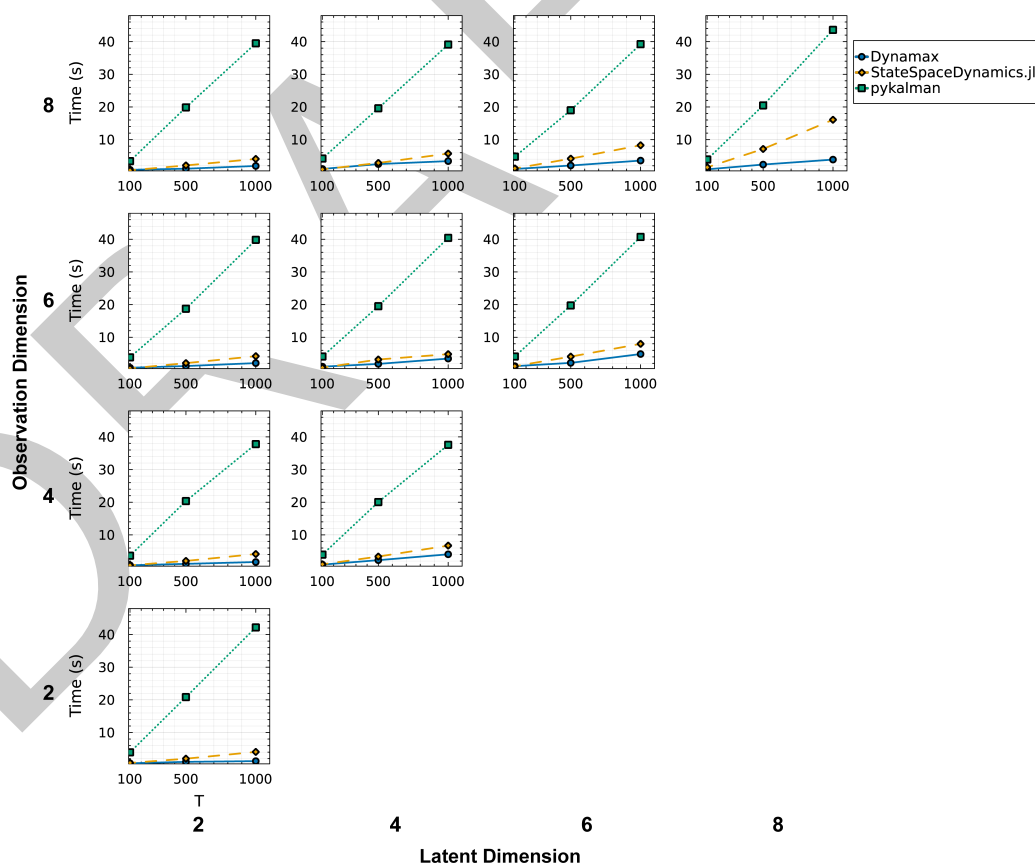
Benchmarks

To evaluate the performance of `StateSpaceDynamics.jl`, we conducted two benchmarking studies focusing on fitting a Gaussian LDS and a Gaussian 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

89 excluded `StateSpaceModels.jl` from our comparison as its scope is geared towards structured
90 time-series models. `Dynamax` was properly JIT-compiled using the `jax.jit` function prior to
91 benchmarking to ensure fair comparison.

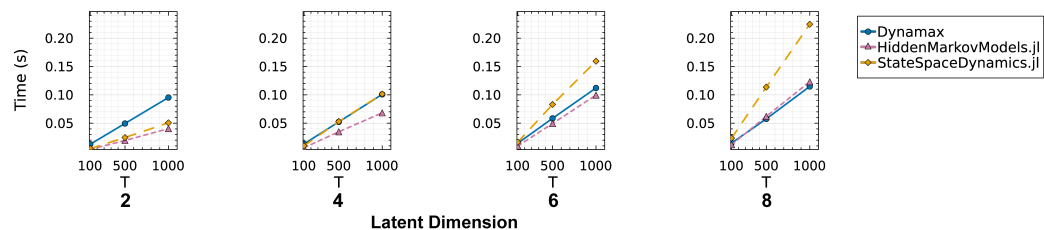
92 For our Gaussian LDS experiments, we constructed a synthetic dataset as follows. The state
93 transition matrix A was generated as a random n -dimensional rotation matrix, while the
94 observation matrix C was created as a random $m \times n$ matrix. Both the state noise covariance
95 Q and observation noise covariance R were set to identity matrices. To ensure a fair comparison,
96 all packages were initialized using identical random parameters, after which we executed the
97 EM algorithm for 100 iterations. We conducted these benchmarks using `PythonCall.jl`
98 (Doris, 2021) and `BenchmarkTools.jl` (Chen & Revels, 2016), with the assumption that
99 Julia-to-Python overhead is negligible for these computationally intensive operations.

100 To thoroughly assess performance across different scales, we tested three sequence lengths
101 ($T = 100, 500, 1000$) and explored multiple dimensionality settings, with state dimensions
102 $n = 2, 4, 6, 8$ and observation dimensions $m = 2, 4, 6, 8$. In all cases, we restricted evaluations
103 to settings where the latent dimension was less than or equal to the observation dimension.
104 Finally, it is worth noting that `Dynamax` includes a temporally parallel smoother with $\mathcal{O}(\log T)$
105 complexity. We did not include this method in our comparisons because it is GPU-specific
106 and incompatible with our direct optimization approach, which is designed for inference in
107 non-conjugate models.



108 For the second benchmarking study, we compared `StateSpaceDynamics.jl`, `HiddenMarkovModels.jl`,
109 and `Dynamax` in their ability to fit a Gaussian HMM. Once again, we ensured that `Dynamax`
110 was JIT-compiled for a fair comparison. To construct synthetic datasets, we sampled from a
111 Gaussian HMM with randomly selected emission models, transition matrices, and initial state
112 distributions. Each package was initialized using identical random parameters to maintain

113 consistency. EM was run for 100 iterations.



114 In our benchmarking, we find that for the LDS, both `StateSpaceDynamics.jl` and `Dynamax`
 115 are faster than `pykalman` across all sequence lengths and dimension configurations. More
 116 generally, `StateSpaceDynamics.jl` and `Dynamax` exhibit similar performance at lower sequence
 117 lengths (with `Dynamax` slightly outperforming `StateSpaceDynamics.jl`). However, `Dynamax`
 118 exhibits superior scaling in both the dimensions of the state and observation matrices as well
 119 as the temporal sequence length. In our current implementation, the Hessian is represented as a
 120 sparse matrix with block tridiagonal structure, resulting in $\mathcal{O}(Tn^2)$ memory scaling — which is
 121 optimal. However, we do not yet exploit this structure fully during inference. In particular, our
 122 solver does not leverage specialized routines for block-banded systems (e.g., the block Thomas
 123 algorithm), which can result in unnecessary fill-in and degraded performance at large T . Future
 124 versions will use banded or block tridiagonal solvers to achieve truly linear-time inference.

125 In our HMM benchmarks, `HiddenMarkovModels.jl` outperforms both `StateSpaceDynamics.jl`
 126 and `Dynamax` across most sequence lengths and state dimensions, with `Dynamax` only becoming
 127 slightly faster for high state dimensions and long sequence lengths. `StateSpaceDynamics.jl`
 128 outperforms `Dynamax` at low state dimensions for all sequence lengths but exhibits worse scaling
 129 with the number of states, allowing `Dynamax` to overtake it as the number of states increases.
 130 These results, combined with our primary development goals in hierarchical SSMs, highlight the
 131 benefits of interfacing with `HiddenMarkovModels.jl` for HMM-specific functionality. Efforts
 132 are currently underway to make this interface seamless.

133 Taken together, these benchmarks demonstrate the competitiveness of `StateSpaceDynamics.jl`
 134 for fitting state-space models. Our benchmarks are available in the benchmarking folder of
 135 our repository, and instructions for running these are available in a `README.md` file.

136 Availability

137 `StateSpaceDynamics.jl` is publicly available under the [GNU license](https://github.com/depasquale-lab/StateSpaceDynamics.jl) at <https://github.com/depasquale-lab/StateSpaceDynamics.jl>.
 138

139 Future Directions

140 The current release of `StateSpaceDynamics.jl` emphasizes efficient CPU-based implementa-
 141 tions and analytically derived gradients and Hessians for commonly used observation mod-
 142 els. Several avenues of future development will broaden the scope and accessibility of the
 143 package. First, we plan to add optional support for automatic differentiation (AD) using
 144 Julia's AD ecosystem (e.g., `ForwardDiff.jl` (Revels et al., 2016), `Zygote.jl` (Innes, 2018),
 145 `DifferentiationInterface.jl` (Dalle & Hill, 2025)). This will allow users to prototype new
 146 observation models without requiring hand-coded derivatives, while maintaining the existing
 147 optimized implementations for speed-critical cases. Second, we aim to extend hardware support
 148 to GPU backends by exploiting Julia's GPU array abstractions and block-tridiagonal solvers,
 149 enabling large-scale inference with temporally parallel methods. Finally, we plan to expand
 150 parameter inference options beyond maximum likelihood and Laplace-EM, including Bayesian

151 approaches via variational inference and interoperability with probabilistic programming frame-
 152 works such as Turing.jl. Together, these developments will further enhance the package's
 153 flexibility, performance, and utility across scientific disciplines.

154 Conclusion

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

162 Author contributions

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

172 Acknowledgements

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

174 References

- 175 Ashwood, Z. C., Roy, N. A., Stone, I. R., Urai, A. E., Churchland, A. K., Pouget, A., & Pillow,
 176 J. W. (2022). Mice alternate between discrete strategies during perceptual decision-making.
 177 *Nature Neuroscience*, 25(2), 201–212. <https://doi.org/10.1038/s41593-021-01007-z>
- 178 Bengio, Y., & Frasconi, P. (1994). An input output HMM architecture. In G. Tesauro,
 179 D. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems*
 180 (Vol. 7). MIT Press. [https://proceedings.neurips.cc/paper_files/paper/1994/file/](https://proceedings.neurips.cc/paper_files/paper/1994/file/8065d07da4a77621450aa84fee5656d9-Paper.pdf)
 181 [8065d07da4a77621450aa84fee5656d9-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/1994/file/8065d07da4a77621450aa84fee5656d9-Paper.pdf)
- 182 Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- 183 Chen, J., & Revels, J. (2016). Robust benchmarking in noisy environments. *arXiv e-Prints*.
 184 <https://arxiv.org/abs/1608.04295>
- 185 Dalle, G. (2024). HiddenMarkovModels.jl: Generic, fast and reliable state space modeling.
 186 *Journal of Open Source Software*, 9(96), 6436. <https://doi.org/10.21105/joss.06436>
- 187 Dalle, G., & Hill, A. (2025). *A common interface for automatic differentiation*. [https://](https://arxiv.org/abs/2505.05542)
 188 arxiv.org/abs/2505.05542
- 189 Doris, C. (2021). *PythonCall.jl*. <https://github.com/JuliaPy/PythonCall.jl>.
- 190 Fjelde, T. E., Xu, K., Widmann, D., Tarek, M., Pfiffer, C., Trapp, M., Axen, S. D., Sun,
 191 X., Hauru, M., Yong, P., Tebbutt, W., Ghahramani, Z., & Ge, H. (2025). Turing.jl: A

- 192 general-purpose probabilistic programming language. *ACM Trans. Probab. Mach. Learn.*
 193 <https://doi.org/10.1145/3711897>
- 194 Ge, H., Xu, K., & Ghahramani, Z. (2018). Turing: A language for flexible probabilistic
 195 inference. In A. Storkey & F. Perez-Cruz (Eds.), *Proceedings of the twenty-first international*
 196 *conference on artificial intelligence and statistics* (Vol. 84, pp. 1682–1690). PMLR.
 197 <https://proceedings.mlr.press/v84/ge18b.html>
- 198 Ghahramani, Z., & Hinton, G. E. (2000). Variational learning for switching state-space models.
 199 *Neural Computation*, 12(4), 831–864. <https://doi.org/10.1162/089976600300015619>
- 200 Innes, M. (2018). Don't unroll adjoint: Differentiating SSA-form programs. *CoRR*,
 201 [abs/1810.07951](https://arxiv.org/abs/1810.07951). <http://arxiv.org/abs/1810.07951>
- 202 Linderman, S. (2022). *SSM: Bayesian learning and inference for state space models*. <https://github.com/lindermanlab/ssm>
- 203
 204 Linderman, Scott W., Chang, P., Harper-Donnelly, G., Kara, A., Li, X., Duran-Martin, G., &
 205 Murphy, K. (2025). Dynamax: A Python package for probabilistic state space modeling
 206 with JAX. *Journal of Open Source Software*, 10(108), 7069. <https://doi.org/10.21105/joss.07069>
- 207
 208 Linderman, Scott W., Miller, A. C., Adams, R. P., Blei, D. M., Paninski, L., & Johnson,
 209 M. J. (2016). Recurrent switching linear dynamical systems. *arXiv [Stat.ML]*. <https://doi.org/10.48550/arXiv.1610.08466>
- 210
 211 Macke, J. H., Buesing, L., Cunningham, J. P., Yu, B. M., Shenoy, K. V., & Sahani, M. (2011).
 212 *Empirical models of spiking in neural populations*. 24. https://proceedings.neurips.cc/paper_files/paper/2011/file/7143d7fbadfa4693b9eec507d9d37443-Paper.pdf
- 213
 214 Murphy, K. P. (1998). *Switching kalman filters* [Technical report].
- 215 Paninski, L., Ahmadian, Y., Ferreira, D. G., Koyama, S., Rahnama Rad, K., Vidne, M.,
 216 Vogelstein, J., & Wu, W. (2010). A new look at state-space models for neural data. *J.*
 217 *Comput. Neurosci.*, 29, 107–126. <https://doi.org/10.1007/s10827-009-0179-x>
- 218 Revels, J., Lubin, M., & Papamarkou, T. (2016). Forward-mode automatic differentiation in
 219 Julia. *arXiv:1607.07892 [Cs.MS]*. <https://arxiv.org/abs/1607.07892>
- 220 Rybicki, G. B., & Hummer, D. G. (1990). *Fast solution for the diagonal elements of the inverse*
 221 *of a tridiagonal matrix*.
- 222 Saavedra, R., Bodin, G., & Souto, M. (2019). StateSpaceModels.jl: A julia package for
 223 time-series analysis in a state-space framework. *arXiv Preprint arXiv:1908.01757*.
- 224 Senne, R., Loschinsky, Z., Loughridge, C., & DePasquale, B. (2025). *StateSpaceDynamics.jl:*
 225 *A julia package for probabilistic state space models*. <https://github.com/depasquale-lab/StateSpaceDynamics.jl>
 226 <https://github.com/depasquale-lab/StateSpaceDynamics.jl>