Computation-Aware Kalman Filtering with Model Selection for Neural Dynamics

JR Huml — *Department of Statistics, Columbia University; Zyphra Technologies*
Jonathan Wenger — *Department of Statistics, Columbia University; Zuckerman Institute*
John P. Cunningham — *Department of Statistics, Columbia University; Zuckerman Institute*

Abstract

Due to their explicit priors and ability to model uncertainty, Bayesian methods have played a major role in dynamical latent variable modeling of single-cell neural recordings. However, modern-sized datasets have made overparameterized deep networks the preferred methods of choice due to their predictive power and favorable computational scaling. While many posterior approximations exist, all incur approximation errors. Recent work accounts for this error in the form of computational uncertainty but comes at the cost of quadratic complexity and assumes fixed model hyperparameters. Here we extend this development to model selection, including a novel training loss and optimization scheme, which yields tractable inference in large state-spaces. We introduce a framework, the Computation-Aware State-Space Model (CASSM), specifically designed for the *scale-imbalanced regime*, where the number of trials is significantly lower than the number of recorded neurons. In this regime, for both synthetic and real data, we show that our method is competitive with data-hungry deep networks, with significantly improved uncertainty calibration over previous attempts to scale Bayesian methods. Our experiments provide a roadmap to neuroscience researchers in choosing from a host of potential dynamical latent variable models given key dataset properties and constraints.

Proceedings of the 2nd International Conference on Probabilistic Numerics (ProbNum) 2026,
Lappeenranta, Finland. PMLR Volume 341.

1 Introduction

While behaviors are governed by large populations of neurons, many such processes may be well-described by low-dimensional manifold structure [1]. Neural interactions can be summarized as a set of latent variables evolving in a low-dimensional space, allowing neuroscientists to decode internal representations and neural dynamics with computational models. However, with the largest datasets now far beyond the typical tens or hundreds neurons at varying temporal resolutions [2, 3], dynamical latent variable models are becoming computationally strained.

As our understanding of neural dynamics shifts with this increase in data, the manifold hypothesis may reveal itself as more of a computational convenience than scientific reality. "Low-dimensional systems" might be so only relative to the scale of systems with billions of components. [4], for example, posits that dimensionality reduction may be the wrong approach to modeling neural representations, showing that across the human visual cortex, dimensionality is unbounded and scales with dataset size.

To predict neural dynamics from spiking data is a precarious balance of model capacity and interpretability. As we seek to understand the structure of neural representations, an ideal model would both predict new data well *and* directly map its underlying components to experimental objects of interest, like a proposed differential equation. A Bayesian method typical of dynamical latent variable models like Gaussian Process Factor Analysis (GPFA) [5] can flexibly incorporate prior knowledge and quantify predictive uncertainty, but the model's cubic complexity with respect to temporal resolution is a limiting factor for modern datasets.

In contrast, nonlinear overparameterized deep algorithms like Latent Factor Analysis via Dynamical Systems (LFADS) [6] scale more favorably and have seen success across many applications, from calcium imaging modalities [7] to intracortical brain-computer interfaces [8], yet such models may have limited ability to quantify uncertainty and, most importantly, require large numbers of trials to offset their relative lack of inductive bias.

We term this data-limited setting the *scale-imbalanced regime*, where the number of trials is far less than the number of neurons. We consider this regime to be the future of neuroscience. In the context of the human brain, for example, hundreds of billions of neurons cannot reasonably yield hundreds of billions of trials without heavy data augmentation. In other words, compute-optimal scaling laws for neuroscience models will necessarily lag far behind other modalities like language [9].

1.1 Previous Work and Limitations

Previous work has addressed the scaling issues of Bayesian methods with approximation methods. Typically, this approximation exploits the low-dimensional structure of neural dynamics. Sparse variational inference and inducing point methods [10, 11] are two such examples. In particular, bGPFA [11] develops a scalable Bayesian version of GPFA. However, while one of the main advantages of a Bayesian framework is uncertainty quantification, this approximation then leaks into uncertainty (and mean) estimates without scrutiny. For example,

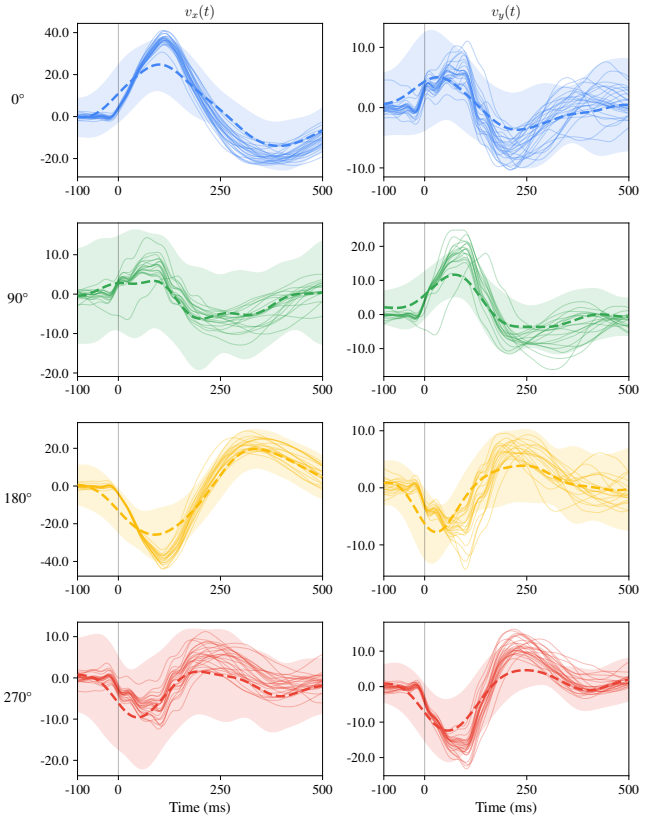


Figure 1: Hand velocities in the $x - y$ plane for four different directions (0°, 90°, 180°, 270°) for the Area 2 Bump dataset, where a primate is bumped from different angles during a reaching task. The mean predictions and standard deviations from CASSM are shown in the dashed line and filled areas, respectively. Kinematics are obtained by sampling from the CASSM filtering distribution after training and averaging over the linear decoder fits.

one typical manifestation of this leakage in Bayesian optimization is overconfidence or "variance starvation" [12, 13].

By abstracting this approximation away from inference, two neuroscientists could have the same model, prior knowledge, data, even hardware and computing resources, but reach different conclusions without accounting for the biases introduced by a seemingly-heuristic approximation. The Computation-Aware Kalman Filter (CAKF) [14] has addressed the quantification of approximation error with the notion of *computational uncertainty*, but not model selection. Prespecification of the dynamics and observation models is a major limitation in neuroscientific applications where these are *a priori* unknown and must be learned from data.

1.2 Our Approach

In this work, we introduce a probabilistic numerical method, the Computation Aware State-Space Model (CASSM), that learns a low-dimensional projection of Kalman-filtered dynamics by matching the projected and computationally infeasible true posterior [15]. In particular, we optimize a decomposition of the ELBO into a data-fit term and a divergence term that penalizes deviation from the exact filtering distribution. Extending core ideas of the CAKF [14] to model selection, this projection yields precise estimates of the uncertainty under incomplete computation, and increases uncertainty relative to the amount of approximation. We show this phenomenon in Figure 2, a decomposition that is not possible for previous approximation methods like [10, 11]. We show that our method can be mathematically interpreted as a principal components analogue wrapped in a state-space model, which benefits from but does not explicitly require low-dimensional manifold structure as the number of simultaneously recorded neurons increases. Instead, we assume that an idealized posterior exists, but limited computational capacity prohibits direct access to this posterior. Our method is *not* intended for generalized use cases: it is highly specialized for the scale-imbalanced regime, but in this regime, it is highly competitive.

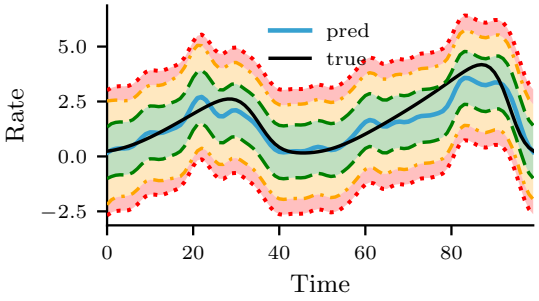


Figure 2: As the amount of computation decreases and posterior approximation increases, the uncertainty bands around the neural trajectory estimates of CASSM widen: (a) in green, we see that the Kalman smoother uncertainty estimates in the case of no approximation of a 30-dimensional state-space; (b) in orange, we see the added uncertainty from projecting this 30-dimensional system to a 10-dimensional space; (c) in red, we see the further added uncertainty from projecting to a 2-dimensional space.

By translating the Gaussian process representation to a state-space representation, we also introduce a novel, natural technique to explicitly encode a *spatial prior* over neurons, allowing the user to specify spatial coordinates or otherwise differentiate neurons by cell type and location, a feature which is only implicit in GPFA, LFADS, or their variants.

Contribution We introduce the Computation-Aware State-Space Model (CASSM), a probabilistic numerical framework that:

(1) *enables model selection* with a novel loss function that enables efficient hyperparameter optimization and preserves the mathematical interpretation of computational uncertainty for inference in the CAKF [14]

2 Background

2.1 State-Space Models

A state-space model (SSM) is one particular choice of representation for dynamical systems that has gained significant popularity in recent years for sequence modeling in natural language processing [16]. In this framework, we wish to infer latent state variables $\mathbf{u}_k$ from a given set of noisy observations $\{\boldsymbol{y}_k\}_{k=1}^{n_t}$ through the following

two-stage transformation:

$$\mathbf{u}_k = \boldsymbol{A}_{k-1}\mathbf{u}_{k-1} + \boldsymbol{b}_{k-1} + \mathbf{q}_{k-1} \in \mathbb{R}^d, \tag{2.1a}$$
$$\mathbf{y}_k = \boldsymbol{H}_k\mathbf{u}_k + \boldsymbol{\epsilon}_k \in \mathbb{R}^n. \tag{2.1b}$$

which we refer to as the *dynamics equation* and *observation equation*, respectively. Unlike the setting of the CAKF [14], we consider $\Theta_k = \{\boldsymbol{A}_k, \boldsymbol{b}_k, \mathbf{q}_k, \boldsymbol{H}_k, \boldsymbol{\epsilon}_k\}_{k=1}^{n_t}$ unknown in addition to $\{\mathbf{u}_k\}_{k=1}^{n_t}$, though we do also consider the setting of linear Gaussian state-space models. Under the conditions of a Gauss-Markov stochastic process, exact Bayesian inference of the conditional distributions $\mathbf{u}_k \mid \mathbf{y}_{1:k}$ can be done using the Kalman filter.

We include a more in-depth discussion of filtering in Appendix B, but at a high level, the exact Kalman filter proceeds as follows. First, we assume initial prior (unconditional) state $\mathbf{u}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. Next, we assume the *process noise model* $\mathbf{q}_{k-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{Q}_{k-1})$ and *measurement noise model* $\boldsymbol{\epsilon}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}_k)$. The algorithm recursively computes the filtering distribution $\mathbf{u}_k^f \coloneqq (\mathbf{u}_k \mid \mathbf{y}_{1:k} = \boldsymbol{y}_{1:k}) \sim \mathcal{N}(\boldsymbol{m}_k, \boldsymbol{P}_k)$ by first computing the *pre-update predictive distribution* $\mathbf{u}_k^{f-}$:

$$\begin{aligned}\boldsymbol{m}_k^- &\coloneqq \boldsymbol{A}_{k-1}\boldsymbol{m}_{k-1}\\ \boldsymbol{P}_k^- &\coloneqq \boldsymbol{A}_{k-1}\boldsymbol{P}_{k-1}\boldsymbol{A}_{k-1}^\top + \boldsymbol{Q}_{k-1}\end{aligned}$$

for $\mathbf{u}_k^{f-} \coloneqq (\mathbf{u}_k \mid \mathbf{y}_{1:k-1}) \sim \mathcal{N}(\boldsymbol{m}_k^-, \boldsymbol{P}_k^-)$ in the *predict step* and the moments

$$\boldsymbol{m}_k \coloneqq \boldsymbol{m}_k^- + \boldsymbol{P}_k^-\boldsymbol{H}_k\boldsymbol{G}_k^{-1}(\mathbf{y}_k - \boldsymbol{H}_k\boldsymbol{m}_k^-) \tag{2.2}$$
$$\boldsymbol{P}_k \coloneqq \boldsymbol{P}_k^- - \boldsymbol{P}_k^-\boldsymbol{H}_k\boldsymbol{G}_k^{-1}\boldsymbol{H}_k^\top\boldsymbol{P}_k^- \tag{2.3}$$

of $\mathbf{u}_k^f \coloneqq (\mathbf{u}_k \mid \mathbf{y}_{1:k}) \sim \mathcal{N}(\boldsymbol{m}_k, \boldsymbol{P}_k)$ in the *update step*, where the innovation matrix $\boldsymbol{G}_k \coloneqq \boldsymbol{H}_k\boldsymbol{P}_k^-\boldsymbol{H}_k^\top + \boldsymbol{\Lambda}_k$.

If the initial state, process noise, and measurement noise are pairwise independent, then $\mathbf{u}_k \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, where:

$$\boldsymbol{\mu}_k \coloneqq \boldsymbol{A}_{k-1}\boldsymbol{\mu}_{k-1} + \boldsymbol{b}_{k-1} \tag{2.4a}$$
$$\boldsymbol{\Sigma}_k \coloneqq \boldsymbol{A}_{k-1}\boldsymbol{\Sigma}_{k-1}\boldsymbol{A}_{k-1}^\top + \boldsymbol{Q}_{k-1} \tag{2.4b}$$

in the *prior update step*.

2.2 Tradeoffs in LGSSMs

Standard regression approaches like Gaussian Process Factor Analysis have cubic cost $\mathcal{O}(n_t^3)$ for temporal resolution n_t, which requires computational improvisations on modern datasets. Instead of artificially chunking time series into pseudo-trials and treating these as independent samples, one can leverage the inherent temporal structure of neural dynamics by performing Bayesian filtering and smoothing (BFS), which has linear time complexity $\mathcal{O}(n_t)$ [17].

However, there are multiple problems with BFS, as we: (P1) maintain the $\mathcal{O}(n_x^3)$ time and $\mathcal{O}(n_x^2)$ memory of the classic regressions for spatial resolution n_x due to the inversion and storage of the innovation matrix $\boldsymbol{G}_k$; (P2) require $\mathcal{O}(d^2)$ memory for covariances $\boldsymbol{P}_k$ and $\boldsymbol{P}_k^-$; (P3) Θ_k must be pre-specified, and learning these quantities is infeasible when n and d are large without significant structual modifications. The CAKF addresses (P1) and (P2), while CASSM addresses (P3).

2.3 Computation-Aware Kalman Filtering

Core ideas of efficient filtering have been introduced in [14] and [18]. Central to both works are the assumed low-rank structure on the posterior covariance matrix. [18] approximates the posterior covariance using the Woodbury identity, while the CAKF [14] introduces a fixed projection of the data. In either case, the posterior covariance can be approximated quite accurately by a low-rank perturbation of the prior equilibrium (stationary) covariance of the state $\mathbf{u}_k^f$. To reduce time and memory complexity of the update step in CASSM, we project the observation as in the CAKF: $\check{\mathbf{y}}_k \coloneqq \boldsymbol{S}_k^\top\mathbf{y}_k$ but with a *learned* policy $\boldsymbol{S}_k \in \mathbb{R}^{n\times\check{n}}$ and $\check{n} \ll n$. The updated observation equation is:

$$\check{\mathbf{y}}_k = \underbrace{\boldsymbol{S}_k^\top\boldsymbol{H}_k}_{=:\check{\boldsymbol{H}}_k}\mathbf{u}_k + \underbrace{\boldsymbol{S}_k^\top\boldsymbol{\epsilon}_k}_{=:\check{\boldsymbol{\epsilon}}_k} \in \mathbb{R}^{\check{n}} \tag{2.5}$$

Storage of $\hat{\boldsymbol{G}}_k$ is reduced to $\mathcal{O}(\check{n}^2)$ and its inversion is reduced to $\mathcal{O}(\check{n}^3)$. In other words, the algorithm only scales prohibitively with respect to the intrinsic dimensionality of the data, solving (P1). The resulting predictive and filtering moments $\{\hat{\boldsymbol{m}}_k^-, \hat{\boldsymbol{P}}_k^-\}_{k=1}^{n_t}$ and $\{\hat{\boldsymbol{m}}_k, \hat{\boldsymbol{P}}_k\}_{k=1}^{n_t}$ are approximations of the filtering distributions. The updated filtering covariance yields:

$$\hat{\boldsymbol{P}}_k = \hat{\boldsymbol{P}}_k^- - \hat{\boldsymbol{P}}_k^-\check{\boldsymbol{H}}_k^\top\check{\boldsymbol{V}}_k(\hat{\boldsymbol{P}}_k^-\check{\boldsymbol{H}}_k^\top\check{\boldsymbol{V}}_k)^\top$$

for $\check{\boldsymbol{V}}_k\check{\boldsymbol{V}}_k^\top = \check{\boldsymbol{G}}_k^{-1}$. We can see that the projection alone does not solve (P2), as $\hat{\boldsymbol{P}}_k$ remains an $\mathcal{O}(d^2)$ object. Instead of considering the update to $\hat{\boldsymbol{P}}_k$, we can consider low-rank perturbations around the prior process noise. The update-predict recursions have an equivalent low-rank downdate structure:

$$\begin{aligned}\hat{\boldsymbol{P}}_k^- &= \boldsymbol{\Sigma}_k - \hat{\boldsymbol{M}}_k^-(\hat{\boldsymbol{M}}_k^-)^\top,\\ \hat{\boldsymbol{P}}_k &= \boldsymbol{\Sigma}_k - \hat{\boldsymbol{M}}_k\hat{\boldsymbol{M}}_k^\top.\end{aligned} \tag{2.6}$$

where $\hat{\boldsymbol{M}}_k^- = \boldsymbol{A}_{k-1}\hat{\boldsymbol{M}}_{k-1}$ and $\hat{\boldsymbol{M}}_k = (\hat{\boldsymbol{M}}_k^- \quad \hat{\boldsymbol{P}}_k^-\check{\boldsymbol{H}}_k^\top\check{\boldsymbol{V}}_k)$. Thus, in our update step, we only store the "skinny" low-rank square roots $\hat{\boldsymbol{M}}_k^-, \hat{\boldsymbol{M}}_k \in \mathbb{R}^{d\times r}$. The latent covariances can be lazily accessed with matrix-vector products $\boldsymbol{A}_{k-1}\boldsymbol{w}$, $\boldsymbol{\Sigma}_k\boldsymbol{w}$, and $\boldsymbol{H}_k\boldsymbol{w}$. By imposing kernel structure on the prior covariance $\boldsymbol{\Sigma}[(t_1, \mathbf{x}_1), (t_2, \mathbf{x}_2)]$, we decrease the number of trainable parameters and perform these multiplications without explicitly forming $\hat{\boldsymbol{P}}_k^-, \hat{\boldsymbol{P}}_k$ in memory. This solves (P2), reducing memory requirements from $\mathcal{O}(d^2)$ to $\mathcal{O}(dr)$. We include more filter-specific details for the CAKF in Appendix B.

However, this filtering structure does not address selection of hyperparameters Θ_k (P3). Furthermore, choice of action can influence runtime and memory considerations. Previous work either suggests but does not implement strategies for the model selection task or assumes that hyperparameters are already known. To address principled and fast hyperparameter selection, we first introduce a new interpretation of the policy.

3 CASSM

3.1 Actions as Principal Components Analysis

Model selection enables a data-driven interpretation of the learned policy $\boldsymbol{S}_k$. In particular, we establish a connection to Principal Components Analysis (PCA) via Theorem 1 from an information-theoretic perspective. We include the proof in Appendix 3.1.

Theorem 1 (Greedy Posterior Entropy Minimization)**.** *Given a sequence of actions $\boldsymbol{S}_1, \dots \boldsymbol{S}_{k-1}$, the action $\boldsymbol{S}_k$ that maximally reduces the latent uncertainty given by the entropy of the posterior update is given by the top $\check{n}$ eigenvectors of the innovation covariance $\boldsymbol{G}_k$*

Geometrically, the optimal policies are the directions of greatest expected uncertainty in the measurement residual $\boldsymbol{r}_k := \mathbf{y}_k - \boldsymbol{H}_k \boldsymbol{m}_k^-$ during the update step (Eq. 2.2). If GPFA [5] is a Factor Analysis model wrapped in a Gaussian Process (GP) regression, then CASSM is essentially a PCA analogue wrapped in a state-space model.

3.1.1 Subspace matching

We verify this theorem empirically in Figure 3. To validate our theorem, we fix the model hyperparameters Θ_k obtained from a classical Kalman Filter that was trained with a log-marginal likelihood loss [17], and then train $\boldsymbol{S}_k$ for dense and sparse actions. While both flavors incur linear memory requirements, dense actions naively incur quadratic time complexity in the state-space dimensionality n. To decrease computational cost and fully leverage hardware acceleration, we impose a sparse block structure on $\boldsymbol{S}_k$. For each $j \in [\check{n}]$, a block is a column vector $\boldsymbol{s}_j \in \mathbb{R}^{k \times 1}$ with $k = n/\check{n}$ entries such that the total number of trainable action parameters equals the number of neurons. This sparsity constraint reduces the time complexity of posterior inference and model selection to linear in the state-space dimensionality.

In Figure 3, we compute the top five eigenvectors for a small $n = 50$ system of neurons where PCA of $\boldsymbol{G}_k$ is trivial. We then compute the Grassmann distance between these eigenvectors and either of the sparse or dense actions. Compared to a choice of randomly fixed actions, optimizing these actions with our proposed hyperparameter optimization in Section 3.2 produces an alignment with optimal action choice minimizing posterior entropy, where the dense actions align more closely than the sparse actions, which are more heavily constrained in their class of directions.

3.1.2 Utility of the Theorem

While Theorem 1 gives some intuition for the function of $\boldsymbol{S}_k$ and our overall framework, its practicality is limited as it shares the $\mathcal{O}(n_x^3)$ time and $\mathcal{O}(n_x^2)$ memory for computing the original GP posterior. In the regime where this is possible, approximation is not necessary. Even if there existed structural exploits on $\boldsymbol{G}_k$ in the infeasible

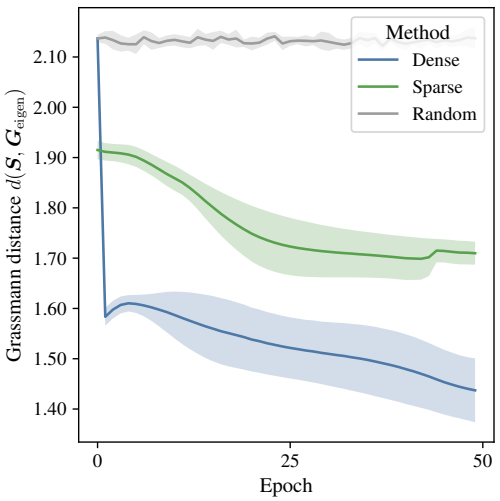


Figure 3: Grassmann distance between the learned S_k (dense and sparse variations) and the top eigenvectors of G_k. The distance decreases during training, indicating that the learned policy converges to the optimal theoretical policy at small neuron scales where PCA is possible. The random policy is a random projection that is kept fixed throughout training.

regime, the assumption of fixed model hyperparameters Θ_k is unwieldy, regardless. Because $\boldsymbol{A}_k$ is assumed to be stable in CASSM, which is a common assumption [19] [20] in general, long-ago observations are forgotten for $k \to \infty$ as the double multiplication of $\boldsymbol{A}_k$ in Eq. (2.6) causes exponential decay in the downdate.

Thus, an EM-style strategy as hinted at in [18] wastes gradient computation of $\boldsymbol{S}_k$ with respect to noisier estimates of Θ_k in earlier timesteps. Ideally, we would optimize the actions alongside the hyperparameters end-to-end, such that the training loss for model selection defines what data projections are informative.

3.2 Model Selection in CASSM

A core contribution of our framework is an efficient loss function for model selection. [17] proposed the negative log-likelihood (NLL) for this task in the exact filter. Given the projected models, a reasonable solution is to simply evaluate the NLL in the approximate space. Empirically, the performance of this solution degrades rapidly with respect to the projection compression, irrespective of the data's intrinsic dimensionality. Of course, this solution only considers the update $\hat{\boldsymbol{r}}_k := (\check{\boldsymbol{y}}_k - \check{\boldsymbol{H}}_k \boldsymbol{m}_k^-)$. If the prediction $\boldsymbol{H}_k \boldsymbol{m}_k^-$ were computationally infeasible, however, then summarizing the activity of large populations of neurons would be a futile task in general.

Another proposed solution is the application of the low-rank determinant lemma to compute the marginal log-likelihood in approximate space that is mentioned, but not implemented, in [18]. However, this is quite similar to the aforementioned NLL in the approximate space, as we are maximizing likelihood under the wrong posterior. Instead, we wish to maximize likelihood under true model and penalize deviation from the true posterior as in [15]. Specifically, we utilize a regularization factor to minimize the divergence between $\mathbf{u}_k^f$ and the distribution

of the approximation space $\hat{\mathbf{u}}_k^f$. While the loss ultimately balances multiple objectives, including the minimization of the projected residual, most crucially the CASSM loss predicts neural activity in the original space using the Mahalanobis distance $d_M(\mathbf{y}_k, \boldsymbol{H}_k \boldsymbol{m}_k^-, \boldsymbol{\Lambda}_k)$. We formalize this idea in Definition 2.

Definition 2 (CASSM Objective)**.**

$$\mathcal{L}(\Theta) = \sum_{k\in[K]} \underbrace{\mathbb{E}_q \left[\log p(\mathbf{y}_k \mid \mathbf{u}_k)\right]}_{\text{data-fit term}} + \sum_{k\in[K]} \underbrace{D_{KL}\left(q(\mathbf{u}_k \mid \mathbf{y}_{1:k}) \parallel p(\mathbf{u}_k \mid \mathbf{y}_{k-1})\right)}_{\text{divergence from exact filtering distribution}}.$$

The objective is an evidence lower bound (ELBO) on the marginal likelihood $\log p(\mathbf{y}_{1:K} \mid \Theta)$, where the approximate distribution $q(\mathbf{u}_k \mid \mathbf{y}_{1:k})$ plays the role of a variational posterior. When $n = \check{n}$, the divergence term vanishes and CASSM reduces to a low-rank Kalman Filter with model selection, similar to [19] [18]. Although both the expectation and divergence terms of the Evidence Lower Bound (ELBO) involve computationally intractable terms, these problematic terms cancel out when combined. We include a derivation of the specific numerical form and a discussion of other practical considerations in Lemma D.2 in Appendix D.

3.3 Training Procedure and Numerical Considerations

3.3.1 Benefits of Kernel Structure

The prior process update (Eqs. 2.4a, 2.4b), in the case of a linear time-invariant (LTI) system, induces a recursion that involves computing a series of matrix powers, which is computationally wasteful and sometimes numerically unstable. In deterministic settings, alternative SSM frameworks like H3 [21] overcome this difficulty by implementing a global convolutional form. Similarly, in the case of an LTI stochastic differential equation (SDE), the CASSM prior update can also be computed in closed form, avoiding this brute force recursion. Other low-rank LTI filter implementations that impose various structures on $\boldsymbol{A}_k$ like block-tridiagonal or symmetric sparse matrix structure cannot leverage this form and avoid the recursion, which is one of the benefits of the kernel structure.

We show how this efficient reformulation has been done [22] without approximation for certain kernel classes in Appendix G. In particular, the reformulation makes our connection with Gaussian Processes explicit, and exemplifies why GPFA is such a foundational work in models of latent neural trajectories.

3.4 Spatial Priors

While Bayesian filters and smoothers are efficient solvers of spatiotemporal regression problems, they also allow for more explicit representations of prior scientific knowledge. In GPFA and its modern variants [11], spatial interaction is only modeled through a single loading matrix $\boldsymbol{C}_k$. In LFADS, spatial structure is learned via the generator and readout functions. Spatial interaction in CASSM is not only modeled through the learned measurement function $\boldsymbol{H}_k$, but in the prior process covariance decomposition of $\boldsymbol{\Sigma}\left[(t_1, \mathbf{x}_1), (t_2, \mathbf{x}_2)\right] = \boldsymbol{\Sigma}^{(\mathbf{x})}(\mathbf{x}_1, \mathbf{x}_2) \cdot \boldsymbol{\Sigma}^{(t)}(t_1, t_2)$ through $\boldsymbol{\Sigma}^{(\mathbf{x})}(\mathbf{x}_1, \mathbf{x}_2)$. While the chosen kernel class determines $\boldsymbol{\Sigma}^{(t)}(t_1, t_2)$, which is fixed during training and inference, $\boldsymbol{\Sigma}^{(\mathbf{x})}(\mathbf{x}_1, \mathbf{x}_2)$ is user-determined.

This *spatial prior* allows the user to explicitly specify neuron locations or cell types before training, if desired. If this input is unavailable, as is the case for certain datasets in our experiments like the synthetic Lorenz dataset, one can simply learn a position vector from data. We simply learn 3D coordinates for each neuron in all experiments, but if even this is too much of a refinement, one could simply pass neuron labels or brain regions into this vector instead.

3.4.1 Preprocessing

The Kalman Filter and its low-rank variants specify Gaussian likelihoods, which we found is best served by two specific preprocessing steps. First, the discrete, binary-valued input spike trains (see Figure A.1) are convolved with a Gaussian kernel to obtain real-valued smoothed firing rates. While one of the original benefits of GPFA is to unify the spike-smoothing and dimensionality-reduction operations in a common probabilistic framework, we can simply add the kernel standard deviation to the computational graph and learn this parameter from data. After convolution, we then apply the Anscombe variance stabilizing transform:

$$A(x) := 2\sqrt{x + \frac{3}{8}}$$

to convert (pseudo) Poissonian data to approximately Gaussian data and ensure firing rate comparability across trials.

3.4.2 Model Selection and Numerical Stability

Many low-rank filters propose an optimal truncation step for the low-rank downdate $\hat{\boldsymbol{M}}_k$ to avoid accumulation of $d \times r$ matrices over time, which can easily exceed the state-space dimensionality if T is large. This is typically accomplished with a singular value decomposition on $\hat{\boldsymbol{M}}_k$ and dropping the subspace corresponding to the smallest singular values s_i. However, when introducing hyperparameter optimization and backpropogating through time, the truncation step becomes numerically unstable due to the gradient computation in the backward pass, which depends the matrix [23]:

$$\boldsymbol{F}_{ij} = \begin{cases} \dfrac{1}{s_j^2 - s_i^2}, & i \neq j, \\ 0, & i = j. \end{cases}$$

For degenerate singular values, the Jacobian becomes ill-conditioned and training fails. In our training procedure,

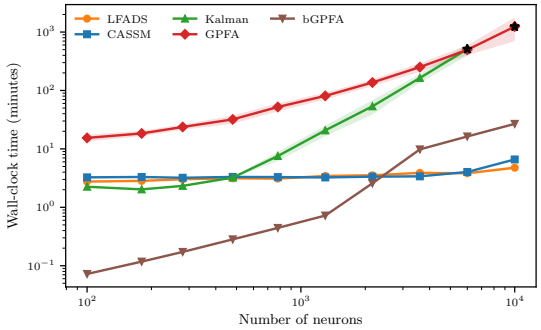


Figure 4: CASSM does not begin to enter its linear scaling regime until around 10^4 neurons, whereas bGPFA begins its quasilinear regime much earlier. At large neural scales with fixed computational budgets, CASSM is almost a day faster than GPFA or the Kalman Filter, and several hours faster than other scalable Bayesian methods.

we utilized a differentiable singular value decomposition based on the Moore-Penrose pseudoinverse [24] to ensure numerical stability in the truncation. We then keep the top $\check{n}$ singular vectors in the update step.

4 Experiments

4.1 Synthetic Tasks

A standard benchmark for previous dynamical latent variable models [6, 25] is the synthetic Lorenz spiking dataset. In this dataset, the neural dynamics are generated from the Lorenz system of nonlinear ordinary differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x),\\ \dot{y} &= x(\rho - z) - y,\\ \dot{z} &= xy - \beta z,\end{aligned}$$

where we choose the standard $(\sigma, \rho, \beta) = (10, 28, \frac{8}{3})$. The dynamics are projected to a space of artificial neurons in $\mathbb{R}^n$ with a random linear projection which produces a rate function λ_t for a Poisson random vector $y_t \sim \text{Pois}(\lambda_t)$ at each timestep $t \in [T]$ to generate artificial spikes. Evaluation of performance is between the model predicted firing rates and the true latent firing rates. We initialize C latent initial conditions with random normal initial states and R trials per condition for $C \cdot R$ total time series, randomly mixing the conditions to force successful models to learn the effects of input pertubations. In addition to denoising, a central difficulty of this task is that slight perturbations in the initial conditions lead to large changes in dynamics.

4.1.1 Computational Scaling

In Figure 4, we evaluate mean training times across 5 seed runs for GPFA, bGPFA (an inducing point method that scales GPFA quasilinearly in time instead of cubically), the Kalman Filter trained with a log-likelihood loss, LFADS, and CASSM. The time, in general, represents the completion of 50 epochs, which was chosen due to its uniformity in reporting and sufficiency for convergence of the majority of methods for most neural scales. We include more justification for our fixed-budget reporting in Appendix H.2.1. In Figures 6(a) and 6(b), we report 100 epochs, and as we can see, this cutoff choice is most favorable to GPFA and LFADS, as CASSM typically converges much earlier than 50 epochs.

Both the Kalman Filter and CASSM are trained with Matérn$\left(\frac{1}{2}\right)$ temporal dynamics models, while GPFA uses a Laplacian kernel, which is a special case of a Matérn$\left(\frac{1}{2}\right)$ kernel. We vary the number of neurons spaced evenly apart in logarithmic space from 10^2 to 10^4 neurons, fixing $T = 100$ timesteps, $C = 10$ conditions, and $R = 10$ trials per condition, or 100 trials. Thus, as the state-space dimensionality increases, the ratio of data to dimensionality is decreasing.

Our synthetic experiments are designed to vary the balance of trials and neurons. In the scale-imbalanced regime, where the ratio of trials to neurons becomes extremely low, we demonstrate that model bias and the structure of optimization becomes much more important. In log-log space, we find that GPFA scales less favorably in a computational sense than CASSM, LFADS, or bGPFA. At $n = 10^4$, the optimization is not able to complete an epoch within a day of computation, which we denote with a black star on the scaling figure (and thus do not report performance for in Figure 6(b)).

The Kalman Filter has roughly the same slope as GPFA at larger neural scales, but runs out of memory near 10^4, and we similarly do not report performance for it in Figure 6(b). CASSM and LFADS only begin to reach their linear scaling regimes around 10^4 neurons, and CASSM is hours faster than bGPFA at this scale.

We also note memory issues for bGPFA unless the number of Monte Carlo samples is small, even when all methods are only equipped with a latent dimensionality of 3. CASSM is slightly slower than LFADS, most likely due to the memory overhead of the linear operators used for performing matrix-free computations. This theory is supported by the performance of the Kalman filter, which is actually faster than LFADS for most of the common neural scales ($10^2 - 10^3$ neurons) of simultaneous recordings due to the efficiency of dense matrix multiplications on GPUs.

4.1.2 Performance Scaling

In Figures 6(a) and 6(b), we report MSE for the difference between the true firing rates and the inferred firing rates at various scales from the experiments of Section 4.1.1. We notice a trend that holds in real data as well: LFADS dominates when the trials far outnumber the neurons, but degrades rapidly in performance when the trials are limited, a direct result of its flexibility in optimization. Here, both scalable Bayesian methods have significantly more competitive predictive performance, and the separating factor is the meaningfulness of uncertainty and computational or memory complexity.

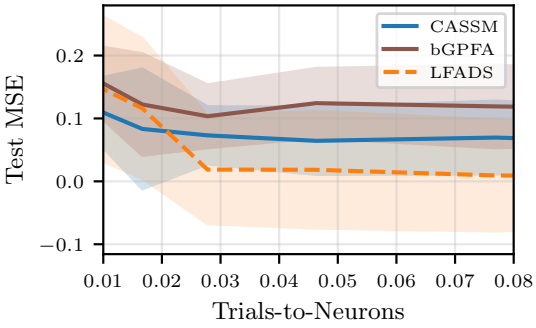


Figure 5: We vary the ratio of trials to neurons, holding trials constant. Here, Bayesian priors can give powerful inductive biases to offset the lack of data. This is evident in the slopes of CASSM (ours) and bGPFA [11], which are approximate Bayesian methods of the Kalman Filter and Gaussian Process Factor Analysis [5], respectively. When holding the number of trials constant relative to the number of neurons, the Bayesian methods are much less sensitive than LFADS, a nonlinear variational autoencoder [6].

	Coverage (95%) $n \in \{100, 1000, 10000\}$		
Method	100	1000	10000
bGPFA [11]	0.701±0.03	0.743±0.03	0.749±0.09
GPFA [5]	0.893±0.05	0.878±0.06	N/A
Kalman Filter	0.872±0.03	0.868±0.03	N/A
CASSM	0.917±0.03	0.919±0.05	0.886±0.07

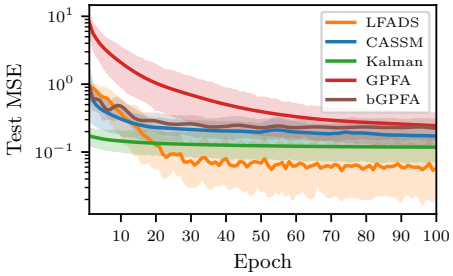
Table 1: Coverage performance for varying orders of magnitude of neurons. Approximation leads to methods that are overconfident in their uncertainty estimates, a problem that CASSM is able to improve upon. We report standard errors across 5 random seeds.

4.1.3 Coverage

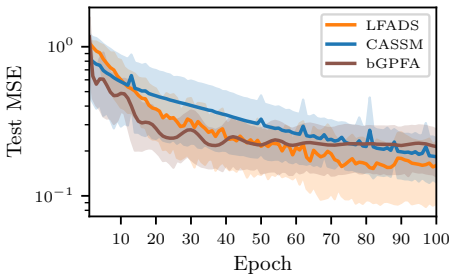
For the Bayesian methods, we checked our uncertainty estimates by evaluating the coverage of the 95% confidence intervals for the true firing rates across the same scales as in the previous experiments. The outcomes confirm analysis on SVGPs and GPs from previous papers [13] [12] in the context of neural data. The results show the need for probabilistic numerical methods in neural data analysis, as the uncertainty quantification of approximation methods is less meaningful when not accounting for computational uncertainty.

4.2 Real Datasets

The Neural Latents Benchmark (NLB) [26] provides four datasets that span motor, sensory, and cognitive brain regions to evaluate unsupervised latent variable models. The primary scoring tool is a co-smoothing metric ("co-bps") [27] that evaluates the log-likelihood of held-out neurons relative to a control model that predicts the trial-averaged firing rate. There are train/validation/test sets for each of held-in and held-out neurons included by NLB: the goal is not to generalize to unseen neurons, but to infer relationships between neurons that can generalize across stimuli. Held-in and held-out neurons are thus both available at training time. At test time, a held-out neuron subset is predicted given held-in test neurons. Secondary metrics include behavior decoding accuracy and peristimulus time histogram matching.



(a) At 10^2 neurons, where the number of trials equals the number of neurons, LFADS quickly dominates the Bayesian methods in performance. At this scale, all methods train in under 20 minutes, and unless uncertainty is explicitly desired, CASSM's value proposition is minimal.



(b) At 10^4 neurons, the inductive bias of Bayesian methods becomes important when the number of trials is several orders of magnitude smaller than the number of neurons. Given CASSM's favorable computational scaling and performance, its value proposition strengthens even when uncertainty quantification is not explicitly desired.

Figure 6: Test MSE across model classes under different neuron-count regimes.

See Appendix 4.2 for more details on the datasets and evaluation metrics.

4.2.1 Evaluating Held-Out Neurons

For GPFA as well as a Bayesian filter and smoother with a linear dynamics model, conditioning the held-out test neurons on the activity of the held-in test neurons is not directly possible due to latent indeterminacy, as the inferred latent trajectory exists in an arbitrarily rotated and scaled subspace of the true latent manifold via the readout function. These models must be trained on the held-in neurons only, and the map between the held-in dynamics and held-out neurons is learned via a linear decoder after training.

The model learned by bGPFA can subsequently be used for predictions on held-out data by conditioning on partial observations as used for cross-validation in §3.1 of [11]. In this case, the model is trained on both held-in and held-out neurons. In the case of a Bayesian filter and smoother with a Matérn dynamics model, as is the case in CASSM, we can directly infer the held-out neurons by training on the entire population and masking the measurement functions for the held-out test neurons during inference.

The results for LFADS and GPFA were tuned by the Neural Latents team [26] to achieve the best performance on the datasets. We tuned the other methods by a grid search over each method's hyperparameters, and report more training details in the Appendix.

	LFADS	bGPFA	GPFA	KF	CASSM
Area 2 Bump Trials: 270 Neurons: 20					
co-bps	0.2569	0.1611	0.1680	0.1699	0.1693
vel R^2	0.8492	0.5573	0.5975	0.5805	0.5818
psth R^2	0.6318	0.5021	0.5289	0.5260	0.5653
DMFC RSG Trials: 740 Neurons: 50					
co-bps	0.1829	0.1105	0.1176	0.1119	0.1097
tp corr	–	–	–	–	–
psth R^2	0.6359	0.4657	0.2142	0.4573	0.4333

Table 2: Method performance on NLB real-world datasets.

	LFADS	bGPFA	GPFA	KF	CASSM
MC Maze Trials: 1720 Neurons: 140					
co-bps	0.3324	0.1840	0.1872	0.1871	0.1859
vel R^2	0.9097	0.5977	0.6399	0.6027	0.6022
psth R^2	0.6360	0.3144	0.5150	0.0548	0.1191
MC RTT Trials: 810 Neurons: 100					
co-bps	0.1868	0.1438	0.1548	0.1434	0.1401
vel R^2	0.6167	0.4720	0.5339	0.4552	0.4444
psth R^2	–	–	–	–	–

Table 3: Method performance on NLB real-world datasets: MC Maze and MC RTT. The real world datasets confirm our previous synthetic experiments, where LFADS dominates when the number of trials is at least the number of neurons. In this regime, the data benefit the flexibility of the LFADS model, and the linear Bayesian methods are unable to capture the complex dynamics.

4.2.2 Hand Kinematics Visualization

Figure 1 demonstrates the time evolution of behaviorally relevant variables predicted by CASSM: for the Area2 Bump recordings of the somatosensory cortex, these are the hand velocities in the $x-y$ plane for a primate in a center-out reaching task. In a portion of the trials, the monkey's arm was bumped from four different angles with a mechanical pertubation before the reach. Linear decoders fitted on predicted activity yield realistic kinematic outputs with calibrated uncertainty quantification, which is facilitated by sampling from the Gaussian filtering distribution after training and averaging over the linear decoder fits. These results validate the accuracy of the neural activity reconstruction, demonstrating the practical utility of our approach in decoding behaviorally relevant signals.

4.2.3 Zebrafish Dataset

Our final experiment considers a large-scale, low-trial dataset of approximately 2.5 million datapoints of fluorescence data from larval zebrafish subjects [28], analyzing roughly 2 orders of magnitude more neurons than the largest experiment in the bGPFA [11] analysis. To enable trial-based learning from recordings, we segmented the continuous hour-long recording by taking advantage of the periodic nature of the experimental stimulus presentations of varying light sources, a procedure which is also used in the trial-averaging cluster analysis of that original paper [28] and is common practice in general [29]. We then hold out neurons for evaluation as performed in our Neural Latents Benchmark experiments.

Zebrafish Trials: 96 Neurons: 88K			
Method	MSE	NLL	Time (hrs)
LFADS [6]	0.038 ± 0.012	–	3.3
bGPFA [11]	OOM	OOM	OOM
CASSM-SP	0.035 ± 0.007	-0.203 ± 0.078	4.0
CASSM	0.044 ± 0.010	-0.132 ± 0.034	4.3

Table 4: Unlike in the Lorenz experiments, where spatial information is absent, the Zebrafish dataset gives approximate $3D$ coordinates for each neuron. CASSM's spatial prior (CASSM-SP) allows it to perform more competitively with LFADS, while also providing calibrated uncertainty quantification at similar computational cost. The performance of bGPFA is not reported due to out-of-memory errors.

5 Conclusion

In this work, we presented CASSM, a probabilistic numerical framework that extends the Computation-Aware Kalman Filter, scales to modern-sized neural datasets, and performs model selection using a novel loss function. In particular, our algorithm is designed for the *scaled-imbalanced regime*, where trials may not scale with the number of recorded neurons and stronger inductive biases are useful for structured learning. We provide a decision tree based on our findings for key dataset properties in Appendix C.1. We found that at smaller neural dataset sizes with many trials like [26], nonlinear methods are clear winners for prediction, while Bayesian methods are essentially "take-your-pick" as approximation is not as essential. However, when scaling to larger datasets with fewer relative trials, CASSM suddenly becomes more competitive with nonlinear, deep learning methods that do not model uncertainty. In particular, when exploiting spatial priors in the zebrafish dataset, we found that CASSM and LFADS perform statistically similar.

5.1 Limitations

Tuning CASSM requires choosing the appropriate number of actions, and the rank of the approximate posterior update is not always clear from the problem structure. Furthermore, as a linear method, choosing too few basis functions can severely degrade performance. Several updated versions of GPFA have incorporated more flexible likelihoods and priors into their analyses, while CASSM is limited to GP regression with a conjugate observational noise model. Lastly, we do not address model misspecification, which is not obvious to incorporate into the LGSSM framework.

5.2 Future Work

CASSM can be applied to many different types of spatiotemporal data beyond neuroscience. In addition, our GPU-accelerated algorithm probably has low-hanging fruit to reduce memory overhead of the linear operators and speed up inference. Additional latent dynamical structures are also possible.

References

[1] Juan A Gallego, Matthew G Perich, Lee E Miller, and Sara A Solla. Neural manifolds for the control of movement. *Neuron*, 94(5):978–984, 2017.

[2] J.H. Siegle, Xiaoxuan Jia, and Severine Durand. Survey of spiking in the mouse visual system reveals functional hierarchy. *Nature*, 2021.

[3] Nicholas Steinmetz, Peter Zatka-Haas, Matteo Carandini, and Kenneth Harris. Distributed coding of choice, action, and engagement across the mouse brain. *Nature*, 2019.

[4] Raj Magesh Gauthaman, Brice Ménard, and Michael F. Bonner. Universal scale-free representations in human visual cortex. *PLOS Computational Biology*, 21(11):e1013714, 2025. doi: 10.1371/journal.pcbi.1013714. URL https://doi.org/10.1371/journal.pcbi.1013714.

[5] Byron M Yu, John P Cunningham, Gopal Santhanam, Stephen Ryu, Krishna V Shenoy, and Maneesh Sahani. Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. 21, 2008. URL https://proceedings.neurips.cc/paper_files/paper/2008/file/ad972f10e0800b49d76fed33a21f6698-Paper.pdf.

[6] Chethan Pandarinath, Daniel J. O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D. Stavisky, Jonathan C. Kao, Eric M. Trautmann, Matthew T. Kaufman, Stephen I. Ryu, Leigh R. Hochberg, Jaimie M. Henderson, Krishna V. Shenoy, L. F. Abbott, and David Sussillo. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature Methods*, 15(10):805–815, oct 2018. doi: 10.1038/s41592-018-0109-9. URL https://www.nature.com/articles/s41592-018-0109-9.

[7] Feng Zhu, Andrew R. Sedler, Harrison A. Grier, Nauman Ahad, Mark A. Davenport, Matthew T. Kaufman, Andrea Giovannucci, and Chethan Pandarinath. Deep inference of latent dynamics with spatio-temporal super-resolution using selective backpropagation through time. 2021. URL https://arxiv.org/abs/2111.00070.

[8] Brianna M. Karpowicz, Yahia H. Ali, Lahiru N. Wimalasena, Andrew R. Sedler, Mohammad Reza Keshtkaran, Kevin Bodkin, Xuan Ma, Lee E. Miller, and Chethan Pandarinath. Stabilizing brain-computer interfaces through alignment of latent dynamics. *bioRxiv*, 2022. doi: 10.1101/2022.04.06.487388.

[9] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv preprint*, 2203.15556, 2022. URL https://arxiv.org/abs/2203.15556.

[10] Lea Duncker and Maneesh Sahani. Temporal alignment and latent gaussian process factor inference in population spike trains. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

[11] Kristopher T. Jensen, Ta-Chu Kao, Jasmine T. Stone, and Guillaume Hennequin. Scalable Bayesian GPFA with automatic relevance determination and discrete noise models. *bioRxiv*, 2021. doi: 10.1101/2021.06.03.446788. URL https://www.biorxiv.org/content/early/2021/06/03/2021.06.03.446788.

[12] Zi Wang, Clement Gehring, Pushmeet Kohli, and Stefanie Jegelka. Batched large-scale bayesian optimization in high-dimensional spaces. 2018. URL https://arxiv.org/abs/1706.01445.

[13] Jonathan Wenger, Geoff Pleiss, Marvin Pförtner, Philipp Hennig, and John P. Cunningham. Posterior and computational uncertainty in Gaussian processes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[14] Marvin Pförtner, Jonathan Wenger, Jon Cockayne, and Philipp Hennig. Computation-aware Kalman filtering and smoothing. 2024. doi: 10.48550/arxiv.2405.08971. URL https://arxiv.org/abs/2405.08971.

[15] Jonathan Wenger, Kaiwen Wu, Philipp Hennig, Jacob R. Gardner, Geoff Pleiss, and John P. Cunningham. Computation-aware gaussian processes: Model selection and linear-time inference. 2024.

[16] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, 2023. URL http://arxiv.org/abs/2312.00752.

[17] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*, volume 17. Cambridge University Press, 2nd edition, 2023. ISBN 978-1-108-91230-3.

[18] Eftychios A. Pnevmatikakis, Kamiar Rahnama Rad, Justin Huggins, and Liam Paninski. Fast kalman filtering and forward–backward smoothing via a low-rank perturbative approach. *Journal of Computational and Graphical Statistics*, 22(3):671–690, 2014. doi: 10.1080/10618600.2012.760461.

[19] Liam Paninski. Fast kalman filtering on quasi-linear dendritic trees. *Journal of Computational Neuroscience*, 28(2):211–228, 2010. doi: 10.1007/s10827-009-0200-4.

[20] Silvère Bonnabel and Rodolphe Sepulchre. The geometry of low-rank kalman filters. *arXiv preprint*, 2012. URL https://arxiv.org/abs/1203.4049. Revised version: 2013.

[21] Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022. doi: 10.48550/arXiv.2212.14052. URL https://arxiv.org/abs/2212.14052. ICLR 2023.

[22] Simo Sarkka. Kalman filtering and smoothing solutions to temporal gaussian process regression models. 2010. URL https://users.aalto.fi/~ssarkka/pub/gp-ts-kfrts.pdf.

[23] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Robust differentiable svd. *arXiv preprint arXiv:2104.03821*, 2021. doi: 10.1109/TPAMI.2021.3072422.

[24] Yinghao Zhang and Yue Hu. Differentiable svd based on moore-penrose pseudoinverse for inverse imaging problems. 2024.

[25] Yuan Zhao and Il Memming Park. Variational latent gaussian process for recovering single-trial dynamics from population spike trains. *Neural Computation*, 29(5):1293–1316, may 2017. doi: 10.1162/neco_a_00953.

[26] Felix Pei, Joel Ye, David M. Zoltowski, Anqi Wu, Raeed H. Chowdhury, Hansem Sohn, Joseph E. O'Doherty, Krishna V. Shenoy, Matthew T. Kaufman, Mark Churchland, Mehrdad Jazayeri, Lee E. Miller, Jonathan Pillow, Il Memming Park, Eva L. Dyer, and Chethan Pandarinath. Neural latents benchmark '21: Evaluating latent variable models of neural population activity. 2021. URL https://arxiv.org/abs/2109.04463.

[27] Jakob H. Macke, Lars Büsing, John P. Cunningham, Byron M. Yu, Krishna V. Shenoy, and Maneesh Sahani. Empirical models of spiking in neural populations. In *Advances in Neural Information Processing Systems 24 (NIPS 2011)*, pages 1350–1358, 2011. URL https://papers.nips.cc/paper/2011/file/7143d7fbadfa4693b9eec507d9d37443-Paper.pdf.

[28] X. Chen, Y. Mu, Y. Hu, A. T. Kuan, M. Nikitchenko, O. Randlett, A. B. Chen, J. P. Gavornik, H. Sompolinsky, F. Engert, and M. B. Ahrens. Brain-wide organization of neuronal activity and convergent sensorimotor transformations in larval zebrafish. *Neuron*, 100(4):876–890.e5, 2018. doi: 10.1016/j.neuron.2018.09.064.

[29] Mohammad Reza Keshtkaran, Andrew R. Sedler, Raeed H. Chowdhury, Raghav Tandon, Diya Basrai, Sarah L. Nguyen, Hansem Sohn, Mehrdad Jazayeri, Lee E. Miller, and Chethan Pandarinath. A large-scale neural network training framework for generalized estimation of single-trial population dynamics. *Nature Methods*, 19(12):1572–1577, 2022. doi: 10.1038/s41592-022-01675-0.

[30] David Sussillo, Rafal Jozefowicz, L. F. Abbott, and Chethan Pandarinath. Lfads - latent factor analysis via dynamical systems. 2016. URL https://arxiv.org/abs/1608.06315.

Supplementary Material

The supplementary materials contain derivations for our theoretical framework and proofs for the mathematical statements in the main text.

We also provide implementation specifics and describe our experimental setup in more detail.

A Description of the Computational Problem

A.1 Experimental Setup and Data Collection

We study the problem of inferring latent neural dynamics from multi-electrode array recordings collected during controlled behavioural experiments.

Figure A.1 illustrates the generative structure of such data: a low-dimensional latent trajectory (panel a) drives per-neuron firing rates (panel b), from which discrete spike trains are observed (panel c).

In practice, the latent trajectory is unobserved and must be inferred from spike trains alone.

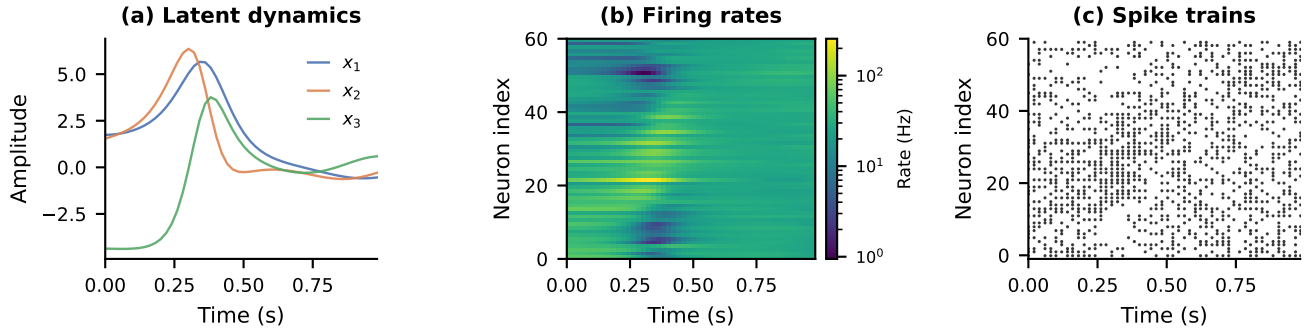


Figure A.1: Generative model for neural population data illustrated on the Lorenz system. **(a)** The three-dimensional latent state $\mathbf{u}_k$ evolves according to chaotic Lorenz dynamics; each trace corresponds to one latent coordinate. **(b)** A linear readout of the latent state passed through a softplus nonlinearity gives per-neuron firing rates (Hz, log scale). Neurons are sorted by their peak-rate time for visual clarity. **(c)** Spike trains are sampled as Bernoulli draws from the instantaneous firing rates.

A.1.1 Recording technology

The main datasets used in our real experiments are drawn from the Neural Latents Benchmark (NLB) [26], a curated collection of primate electrophysiology recordings.

Neural activity is recorded via surgically implanted microelectrode arrays (high-density silicon probes) targeting specific cortical areas.

Each electrode captures the spiking activity of one or more nearby neurons at sub-millisecond (ms) temporal resolution.

Individual neuron action potentials ("spikes") are identified through spike-sorting or thresholding and binned into 5 ms or 20 ms windows, yielding an integer-valued spike count for each neuron at each time step.

Beyond primate electrophysiology, recent large-scale neural recording efforts have also explored whole-brain imaging in small vertebrates such as the zebrafish.

In particular, the experiments of [28] use light-sheet calcium imaging to record activity from tens of thousands of neurons simultaneously across the entire zebrafish brain.

Larval zebrafish are head-fixed while exposed to different frequencies and patterns of light.

The resulting data consist of continuous fluorescence traces with high spatial coverage but lower temporal resolution than electrophysiology.

These datasets provide a complementary regime to spiking recordings: rather than sparse, high-temporal-resolution point processes, they yield dense, smooth signals capturing brain-wide population dynamics, making them particularly well-suited for studying large-scale latent structure and global dynamical patterns.

A.2 Modeling and State-Space Representation

In the neural context, the latent state $\mathbf{u}_k \in \mathbb{R}^d$ represents a compressed description of the instantaneous *population activity vector* at time step k.

The dynamics matrix $\boldsymbol{A}_k$ encodes how the population state propagates forward in time, capturing recurrent structure such as oscillations, fixed points, or slow drift.

In CASSM, this is parameterised as a discretised linear time-invariant SDE whose continuous-time limit corresponds to a Matérn Gaussian process prior (see Appendix G), providing a principled way to encode smoothness assumptions

over the temporal trajectory.

The process noise $\mathbf{q}_k$ accounts for unexplained variability in the latent trajectory (e.g. trial-to-trial fluctuations), while $\boldsymbol{\epsilon}_k$ captures spiking noise and residual variability not explained by the latent representation.

All parameters $\Theta = \{\boldsymbol{A}_k, \boldsymbol{b}_k, \boldsymbol{Q}_k, \boldsymbol{H}_k, \boldsymbol{\Lambda}_k\}_{k=1}^T$ are unknown and must be learned from data.

B Filtering

B.1 Objects of Interest

B.1.1 State-Space Models

$\boldsymbol{A}$: Dynamics function

$\boldsymbol{Q}$: Process noise covariance

$\boldsymbol{\Sigma}$: Prior (unconditional) state covariance

$\boldsymbol{H}$: Measurement function

$\boldsymbol{\Lambda}$: Observation noise covariance

Θ: State-space model parameters

B.1.2 Kalman Filter

$\boldsymbol{m}^-$: Pre-update (predictive) distribution mean

$\boldsymbol{P}^-$: Pre-update (predictive) distribution variance

$\boldsymbol{m}$: Filtering distribution mean

$\boldsymbol{P}$: Filtering distribution variance

$\boldsymbol{K}$: Kalman gain

$\boldsymbol{G}$: Innovation matrix

B.1.3 CASSM

$\boldsymbol{S}$: Policy or action

$\check{\boldsymbol{H}}$: Projected measurement function

$\check{\boldsymbol{\Lambda}}$: Projected observation noise covariance

$\hat{\boldsymbol{m}}^-$: Pre-update (predictive) distribution approximate mean

$\hat{\boldsymbol{P}}^-$: Pre-update (predictive) distribution approximate covariance

$\hat{\boldsymbol{m}}$: Filtering distribution approximate mean

$\hat{\boldsymbol{P}}$: Filtering distribution approximate covariance

$\check{\boldsymbol{G}}$: Projected innovation matrix

$\hat{\boldsymbol{M}}^-$: Pre-update (predictive) belief covariance downdate

$\check{\boldsymbol{y}}$: Projected observation/input

$\hat{\boldsymbol{w}}$: Dynamics message mean

$\hat{\boldsymbol{W}}$: Dynamics message covariance

$\hat{\boldsymbol{M}}$: Filtering belief covariance downdate

B.2 Algorithms

Similar to how the square-root Kalman Filter implements the Kalman Filter, Algorithms 1 and 2 mathematically implement the Computation-Aware Kalman Filter (CAKF) [14], but require modifications to be computationally efficient and differentiable.

Algorithm 1 Computation-Aware State-Space Model (CASSM)

function FILTER($\hat{\boldsymbol{m}}_0, \{\boldsymbol{\Sigma}_k, \Theta_k, \boldsymbol{y}_k\}_{k=1}^{n_t}$)
  $\hat{\boldsymbol{M}}_0^+ \leftarrow 0 \in \mathbb{R}^{d \times r}$ ▷ Zero matrix
  for $k = 1, \ldots, n_t$ **do**
    $\hat{\boldsymbol{m}}_k^- \leftarrow \boldsymbol{A}_{k-1}[\hat{\boldsymbol{m}}_{k-1}] + \boldsymbol{b}_{k-1}$ ▷ Predict
    $\hat{\boldsymbol{M}}_k^- \leftarrow \boldsymbol{A}_{k-1}[\hat{\boldsymbol{M}}_{k-1}^+]$
    $\hat{\boldsymbol{m}}_k, \hat{\boldsymbol{M}}_k \leftarrow$ UPDATE$(\hat{\boldsymbol{m}}_k^-, \hat{\boldsymbol{M}}_k^-, \ldots)$
    $\hat{\boldsymbol{M}}_k^+ \leftarrow$ PSEUDOINVSVD$(\hat{\boldsymbol{M}}_k)$ ▷ [24]
    $\mathcal{L}^{(k-1)}(\Theta_k)$ += $\mathcal{L}^{(k)}(\Theta_k)$ ▷ Eq. D.1
  return $\{\hat{\boldsymbol{m}}_k, \hat{\boldsymbol{M}}_k\}_{k=1}^{n_t}$

Algorithm 2 CASSM Update Step

function UPDATE$(\hat{\boldsymbol{m}}^-, \hat{\boldsymbol{M}}^-, \boldsymbol{\Sigma}, \boldsymbol{H}, \boldsymbol{\Lambda}, \boldsymbol{y})$
  $\hat{\boldsymbol{P}}^- \leftarrow \boldsymbol{\Sigma} - \hat{\boldsymbol{M}}^-(\hat{\boldsymbol{M}}^-)^\top$
  $\boldsymbol{S} \leftarrow$ POLICY$(\hat{\boldsymbol{m}}^-, \hat{\boldsymbol{P}}^-, \ldots)$
  $\check{\boldsymbol{H}}^\top \leftarrow \boldsymbol{H}^\top \boldsymbol{S}$
  $\check{\boldsymbol{\Lambda}} \leftarrow \boldsymbol{S}^\top \boldsymbol{\Lambda} \boldsymbol{S}$
  $\check{\boldsymbol{y}} \leftarrow \boldsymbol{S}^\top \boldsymbol{y}$
  $\check{\boldsymbol{G}} \leftarrow \check{\boldsymbol{H}} \hat{\boldsymbol{P}}^- \check{\boldsymbol{H}}^\top + \check{\boldsymbol{\Lambda}}$
  $\hat{\boldsymbol{w}} \leftarrow \check{\boldsymbol{H}}^\top \check{\boldsymbol{G}}^\dagger (\check{\boldsymbol{y}} - \check{\boldsymbol{H}} \hat{\boldsymbol{m}}^-)$
  $\hat{\boldsymbol{W}} \leftarrow \check{\boldsymbol{H}}^\top (\check{\boldsymbol{G}}^\dagger)^{\frac{1}{2}}$
  $\hat{\boldsymbol{m}} \leftarrow \hat{\boldsymbol{m}}^- + \hat{\boldsymbol{P}}^- \hat{\boldsymbol{w}}$
  $\hat{\boldsymbol{M}} \leftarrow \begin{pmatrix} \hat{\boldsymbol{M}}^- & \hat{\boldsymbol{P}}^- \hat{\boldsymbol{W}} \end{pmatrix}$
  return $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{M}})$

Most notably, the truncation step in [14] cannot be stably differentiated through, so we replace this with a pseudo-inverse of the covariance downdate as discussed in 3.3.

We highlight matrix-free objects in Algorithms 1 and 2 in blue.

The pre-update latent covariance $\hat{\boldsymbol{P}}^-$ is already the difference between a Kronecker-product linear operator and a root linear operator, so it admits efficient lazy multiplication without explicitly materializing a dense covariance.

In CASSM, the policy $\boldsymbol{S}$ introduces an additional challenge: it is itself represented as a matrix-free object, implemented as a `BlockDiagonalSparseLinearOperator` in the `LinearOperator` Python package.

This creates several design constraints that do not arise in a dense implementation.

First, model selection requires evaluating loss terms that depend not only on filtered means, but also on covariance-derived quantities such as projected noise terms, traces, diagonals, and log-determinants.

When both $\hat{\boldsymbol{P}}^-$ and $\boldsymbol{S}$ are lazy operators, one must decide which of these quantities can be computed through matrix-vector products alone, and which require partial materialization.

For example, the projected observation covariance $\check{\boldsymbol{\Lambda}} = \boldsymbol{S}^\top \boldsymbol{\Lambda} \boldsymbol{S}$ and the projected Gram matrix $\check{\boldsymbol{G}} = \check{\boldsymbol{H}} \hat{\boldsymbol{P}}^- \check{\boldsymbol{H}}^\top + \check{\boldsymbol{\Lambda}}$ may remain implicit during filtering, but loss evaluation requires their diagonals, traces, Cholesky factors, or log-determinants.

These are not interchangeable computationally: a diagonal is trivial to extract, whereas a log-determinant generally forces some smaller dense representation after projection.

C Choosing Between Latent Variable Models

C.1 Description of Competing Models

We compare CASSM against four baseline models.

GPFA [5] is a linear-Gaussian latent variable model that jointly performs temporal smoothing and dimensionality reduction using Gaussian-process priors over latent trajectories. Inference is exact but requires cubic time in the number of time steps and quadratic memory, making it impractical for long recordings.

bGPFA [11] extends GPFA with a variational objective and Bayesian loading matrix, and uses structured approximations to reduce the cost of temporal inference. Training scales approximately linearly in the number of time steps (up to a logarithmic factor), but also grows linearly with the number of neurons and Monte Carlo samples, and quadratically with the latent dimensionality.

LFADS [6] is a sequential variational autoencoder with recurrent neural network dynamics. It is highly expressive and capable of modeling nonlinear trajectories, but training requires backpropagation through time and scales linearly in the number of time steps and neurons, and quadratically in the size of the recurrent hidden states.

Kalman Filter [17] provides exact recursive inference for linear-Gaussian state-space models. Each update requires dense matrix operations on both the latent state and observation dimensions, leading to cubic scaling in both the number of neurons and the latent dimensionality, with quadratic memory requirements.

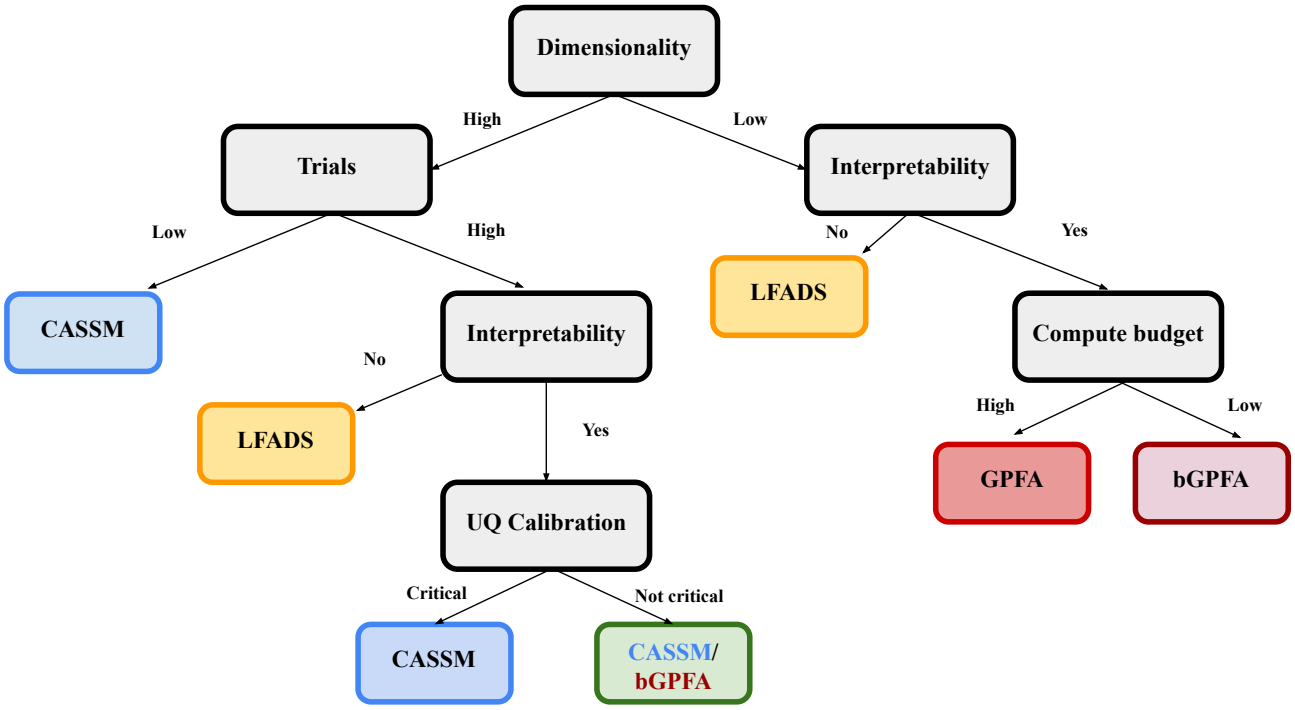


Figure C.1: Decision analysis for choosing between latent variable models based on our experimental findings.

C.2 Tradeoffs and Decision Analysis

Because of the variety of models and algorithms for this problem, an important contribution of our analysis is to provide a basic decision tree for choosing between latent variable models in neuroscience in Figure C.1.

We demonstrated that in the scaled-imbalanced regime, CASSM has superior mean predictive performance to LFADS in the real and synthetic experiments.

However, LFADS is a more flexible model class and can be expected to outperform CASSM in the low scale-imbalanced regime.

In this case, the performance gap is substantial and the justification to use CASSM is quite low.

We were surprised to see a basic Kalman filter with model selection to perform so well at these smaller scales, often more accurate and faster than GPFA.

We note that naive implementation of the Kalman Filter, though not included in our final experiments, can be very difficult to optimize.

This is the simple linear dynamics and linear observation model, where one learns these matrices directly with no structure.

We added manifold parameterizations (e.g. Stiefel manifold, etc.) to the dynamics and observation matrices, which substantially improved optimization, but still encountered difficulties with proper initialization and training stability.

As we explore in Appendix G, the kernel formulation of the Kalman Filter is mathematically equivalent to the Gaussian Process formulation, but is much easier to optimize in practice.

D Deriving the Objective Function

Definition D.1 (CASSM Objective)**.** The CASSM objective function is:

$$\mathcal{L}(\Theta) = -\log p\left(\mathbf{y}_{1:K} \mid \Theta\right) + \sum_{k \in [K]} D_{KL}\left(q(\mathbf{u}_k \mid \mathbf{y}_{1:k}) \,\|\, p(\mathbf{u}_k \mid \mathbf{y}_{1:k})\right)$$

Our primary objects of interest are the pre-update Kalman predictive distribution:

$$p(\mathbf{u}_k \mid \mathbf{y}_{1:k-1}) \sim \mathcal{N}\left(\mathbf{u}_k; \boldsymbol{m}_k^-, \boldsymbol{P}_k^-\right)$$

and the approximate filter distribution:

$$q(\mathbf{u}_k \mid \mathbf{y}_{1:k}) \sim \mathcal{N}\left(\mathbf{u}_k; \hat{\boldsymbol{m}}_k, \hat{\boldsymbol{P}}_k\right)$$

The motivation of $\mathcal{L}$ is to directly regularize each term of the log-marginal likelihood with the KL-divergence of the compressed filter and the original filter.

Because the log-likelihood is a linear function of Θ, this reduces to a recursion in the style of [17]:

$$\mathcal{L}^{(k)}(\Theta) = \mathcal{L}^{(k-1)}(\Theta) - \log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + D_{KL}\left(q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right) \parallel p\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right)$$

We will consider the update $\Delta(\Theta) = \mathcal{L}^{(k)}(\Theta) - \mathcal{L}^{(k-1)}(\Theta)$.

Our goal is to show that the update corresponds to an evidence lower bound from the variational approximation q to the true posterior p.

$$\begin{aligned}
\Delta(\Theta) &= -\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + D_{KL}\left(q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right) \parallel p\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right) \\
&= -\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + \mathbb{E}_{q(\mathbf{u}_k \mid \mathbf{y}_{1:k})}\left[\log \frac{q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)}{p\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)}\right] \\
&= -\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log p\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] \\
&= -\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log \frac{p\left(\mathbf{u}_k, \mathbf{y}_k \mid \mathbf{y}_{k-1}\right)}{p\left(\mathbf{y}_k \mid \mathbf{y}_{1:k-1}\right)}\right] \\
&= -\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right) + \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log p\left(\mathbf{u}_k, \mathbf{y}_k \mid \mathbf{y}_{k-1}\right)\right] + \mathbb{E}_q\left[\log p\left(\mathbf{y}_k \mid \mathbf{y}_{k-1}\right)\right] \\
&= \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log p\left(\mathbf{u}_k, \mathbf{y}_k \mid \mathbf{y}_{k-1}\right)\right] \\
&= \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log\left(p\left(\mathbf{y}_k \mid \mathbf{u}_k, \mathbf{y}_{k-1}\right) \cdot p\left(\mathbf{u}_k \mid \mathbf{y}_{k-1}\right)\right)\right] \\
&= \mathbb{E}_q\left[\log q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right)\right] - \mathbb{E}_q\left[\log p\left(\mathbf{y}_k \mid \mathbf{u}_k, \mathbf{y}_{k-1}\right)\right] - \mathbb{E}_q\left[\log p\left(\mathbf{u}_k \mid \mathbf{y}_{k-1}\right)\right] \\
&= \mathbb{E}_q\left[\log p(\mathbf{y}_k \mid \mathbf{u}_k)\right] + D_{KL}\left(q(\mathbf{u}_k \mid \mathbf{y}_{1:k}) \parallel p(\mathbf{u}_k \mid \mathbf{y}_{k-1})\right)
\end{aligned}$$

We now examine the expectation and KL-divergence terms, respectively.

D.1 Expectation Term

$$\begin{aligned}
\mathbb{E}_{q(\mathbf{u}_k \mid \mathbf{y}_{1:k})}\left[\log p(\mathbf{y}_k \mid \mathbf{u}_k)\right] &= \mathbb{E}_{q(\mathbf{u}_k \mid \mathbf{y}_{1:k})}\left[\log \mathcal{N}(\mathbf{y}_k; \boldsymbol{H}\mathbf{u}_k, \boldsymbol{\Lambda})\right] \\
&= \mathbb{E}_{q(\mathbf{u}_k \mid \mathbf{y}_{1:k})}\left[\frac{1}{2}\log\det(\boldsymbol{\Lambda}) + \frac{1}{2}(\mathbf{y}_k - \boldsymbol{H}\mathbf{u}_k)^T\boldsymbol{\Lambda}^{-1}(\mathbf{y}_k - \boldsymbol{H}\mathbf{u}_k) + \frac{1}{2}n\log(2\pi)\right] \\
&= \frac{1}{2}\left[\log\det(\boldsymbol{\Lambda}) + \mathbb{E}_q\left[(\mathbf{y}_k - \boldsymbol{H}\mathbf{u}_k)^T\boldsymbol{\Lambda}^{-1}(\mathbf{y}_k - \boldsymbol{H}\mathbf{u}_k)\right] + n\log(2\pi)\right] \\
&= \frac{1}{2}\left[\log\det(\boldsymbol{\Lambda}) + (\mathbf{y}_k - \boldsymbol{H}\hat{\boldsymbol{m}}_k)^T\boldsymbol{\Lambda}^{-1}(\mathbf{y}_k - \boldsymbol{H}\hat{\boldsymbol{m}}_k) + \mathrm{tr}(\boldsymbol{\Lambda}^{-1}\boldsymbol{H}\hat{\boldsymbol{P}}_k\boldsymbol{H}^\top) + n\log(2\pi)\right]
\end{aligned}$$

D.2 KL Term

$$
\begin{aligned}
D_{KL}\left(q\left(\mathbf{u}_k \mid \mathbf{y}_{1:k}\right) \| \, p\left(\mathbf{u}_k \mid \mathbf{y}_{1:k-1}\right)\right) &= D_{KL}\left(\mathcal{N}(\mathbf{u}_k; \hat{\boldsymbol{m}}_k, \hat{\boldsymbol{P}}_k) \, \| \, \mathcal{N}(\mathbf{u}_k; \boldsymbol{m}_k^-, \boldsymbol{P}_k^-)\right) \\
&= \frac{1}{2}\Big[\mathrm{tr}(\boldsymbol{P}_k^- \hat{\boldsymbol{P}}_k) - n + (\boldsymbol{m}_k^- - \hat{\boldsymbol{m}}_k)^T (\boldsymbol{P}_k^-)^{-1}(\boldsymbol{m}_k^- - \hat{\boldsymbol{m}}_k) + \log\det(\boldsymbol{P}_k^-) \\
&\quad - \log\det(\hat{\boldsymbol{P}}_k)\Big] \\
&= \frac{1}{2}\Bigg[\mathrm{tr}((\boldsymbol{P}_k^-)^{-1}(\underbrace{\boldsymbol{P}_k^- - \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^-}_{\hat{\boldsymbol{P}}_k})) - n \\
&\quad + (\boldsymbol{m}_k^- - (\underbrace{\boldsymbol{m}_k^- + \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k}_{\hat{\boldsymbol{m}}_k}))^T (\boldsymbol{P}_k^-)^{-1}(\boldsymbol{m}_k^- - \boldsymbol{m}_k^- - \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k) \\
&\quad - \log\det((\boldsymbol{P}_k^-)^{-1}\hat{\boldsymbol{P}}_k)\Big] \\
&= \frac{1}{2}\left[\mathrm{tr}(\boldsymbol{I}_{n\times n} - \hat{\boldsymbol{w}}_k \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^-) - n + \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \log\det(\boldsymbol{I}_{n\times n} - \hat{\boldsymbol{w}}_k \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^-)\right] \\
&= \frac{1}{2}\left[n - \mathrm{tr}(\hat{\boldsymbol{w}}_k \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^-) - n + \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \log\det(\boldsymbol{I}_{n\times n} - \underbrace{\hat{\boldsymbol{w}}_k \hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^-}_{\check{\boldsymbol{H}}^T \check{\boldsymbol{G}}_k^{-1} \check{\boldsymbol{H}} \boldsymbol{P}_k^-})\right]
\end{aligned}
$$

We are able to work in $\mathbb{R}^{\check{n}}$ instead of $\mathbb{R}^n$ in the log-determinant term due to the matrix determinant lemma:

$$\det(\boldsymbol{I} + \boldsymbol{U}\boldsymbol{V}^T) = \det(\boldsymbol{I} + \boldsymbol{V}^T \boldsymbol{A}^{-1}\boldsymbol{U}) \cdot \det(\boldsymbol{A})$$

Continuing with the expansion:

$$
\begin{aligned}
&= \frac{1}{2}\left[\hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \mathrm{tr}(\check{\boldsymbol{G}}_k^{-1} \check{\boldsymbol{H}} \boldsymbol{P}_k^- \check{\boldsymbol{H}}^T) - \log\det(\boldsymbol{I}_{\check{n}\times\check{n}} - \check{\boldsymbol{G}}_k^{-1} \check{\boldsymbol{H}}^T \boldsymbol{P}_k^- \check{\boldsymbol{H}})\right] \\
&= \frac{1}{2}\Bigg[\hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \mathrm{tr}(\check{\boldsymbol{G}}_k^{-1} \check{\boldsymbol{H}} \boldsymbol{P}_k^- \check{\boldsymbol{H}}^T) - \log\det(\check{\boldsymbol{G}}_k^{-1}(\underbrace{\check{\boldsymbol{G}}_k - \check{\boldsymbol{H}}^T \boldsymbol{P}_k^- \check{\boldsymbol{H}}}_{\boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S}}))\Bigg] \\
&= \frac{1}{2}\left[\hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \mathrm{tr}(\check{\boldsymbol{G}}_k^{-1} \check{\boldsymbol{H}} \boldsymbol{P}_k^- \check{\boldsymbol{H}}^T) - \log\det(\check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S})\right] \\
&= \frac{1}{2}\left[\hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \mathrm{tr}(\boldsymbol{I}_{\check{n}\times\check{n}}) + \mathrm{tr}(\check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S}) - \log\det(\check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S})\right]
\end{aligned}
$$

D.3 Final numerical form

For timestep k, combining the results of D.1 and D.2 we obtain the final numerical form of Definition D.1:

Lemma D.2.

$$
\begin{aligned}
\mathcal{L}^{(k)}(\Theta) = &\frac{1}{2}\left[\log\det(\boldsymbol{\Lambda}) + (\mathbf{y}_k - \boldsymbol{H}\hat{\boldsymbol{m}}_k)^T \boldsymbol{\Lambda}^{-1}(\mathbf{y}_k - \boldsymbol{H}\hat{\boldsymbol{m}}_k) + \mathrm{tr}(\boldsymbol{\Lambda}^{-1}\boldsymbol{H}\hat{\boldsymbol{P}}_k\boldsymbol{H}^\top) + n\log(2\pi)\right] \\
&+ \frac{1}{2}\left[\hat{\boldsymbol{w}}_k^T \boldsymbol{P}_k^- \hat{\boldsymbol{w}}_k - \check{n} + \mathrm{tr}(\check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S}) - \log\det(\check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}^T \boldsymbol{\Lambda} \boldsymbol{S})\right]
\end{aligned}
$$

All quantities of the CASSM loss are inexpensive to compute.

The most concerning term is the inverse of the measurement noise function, $\boldsymbol{\Lambda}$. As a dense matrix, this operation requires $O(n^3)$ time.

Experimentally, however, such a measurement device is unlikely to exist in practice: channels can only capture so much long-range dependency (hence the need for hundreds of channels for one probe), and any noise structure can be reasonably treated as at most block-diagonal with a uniformly bounded block size.

Therefore, its inverse can be computed in $O(n)$ time.

The other inverse, $\check{\boldsymbol{G}}_k^{-1}$, exists in $\mathbb{R}^{\check{n}\times\check{n}}$. Here, the projection solves the unfavorable scaling from (P1), as $\check{n} << n$.

E Connection to PCA

Entropy in Time and the Greedy Policy

The entropy of a Gaussian random vector $\mathbf{x} \in \mathbb{R}^n \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given by:

$$\mathrm{H}(\mathbf{x}) = \frac{n}{2}(1 + \log(2\pi)) + \frac{1}{2}\log\det\boldsymbol{\Sigma}$$

Now, let the total latent reduction in entropy be defined as:

$$\tilde{\mathrm{H}}(\mathbf{x}) := \mathrm{H}(\mathbf{x}_{1:T}) - \mathrm{H}(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T})$$

where $\mathbf{x}_{1:T}$ is the set of all latent states and $\mathbf{y}_{1:T}$ is the set of all measurements. By the chain rule for entropy, we have:

$$\mathrm{H}(\mathbf{x}_{1:T}) = \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{x}_{1:k-1})$$

Because we assume a linear Gaussian state-space prior, we also have:

$$\mathrm{H}(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}) = \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k-1})$$

which implies

$$\mathrm{H}(\mathbf{x}_{1:T}) = \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k-1})$$

However, the Gauss-Markov structure also implies:

$$\mathrm{H}(\mathbf{x}_k \mid \mathbf{x}_{1:k-1}, \mathbf{y}_{1:T}) = \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k})$$

because the current latent state only depends on the previous latent state. Taking these two ideas together, we conclude:

$$\mathrm{H}(\mathbf{x}_{1:T} \mid \mathbf{y}_{1:T}) = \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k})$$

So the total reduction in entropy is:

$$\begin{aligned}\tilde{\mathrm{H}}(\mathbf{x}) &= \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}) - \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k}) \\ &= \sum_{k=1}^{T} \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k-1}) - \mathrm{H}(\mathbf{x}_k \mid \mathbf{y}_{1:k})\end{aligned}$$

In other words, the Gaussian entropy for the difference in prior and posterior Kalman distributions decomposes as a sum over time indexes $k \in [T]$. In general:

$$\begin{aligned}\max_{\boldsymbol{S}_{k\in[T]}} \tilde{\mathrm{H}}(\mathbf{x}) &= \max_{\boldsymbol{S}_{k\in[T]}} \sum_{k=1}^{T} \frac{1}{2}\left(\log\det\hat{\boldsymbol{P}}_k^- - \log\det\hat{\boldsymbol{P}}_k\right) \\ &\leq \sum_{k=1}^{T} \max_{\boldsymbol{S}_{k\in[T]}} \frac{1}{2}\left(\log\det\hat{\boldsymbol{P}}_k^- - \log\det\hat{\boldsymbol{P}}_k\right)\end{aligned}$$

Therefore, to achieve the upper bound of differential entropy, it suffices to choose a policy $\boldsymbol{S}_k$ given a sequence of policies $\boldsymbol{S}_1, \ldots, \boldsymbol{S}_{k-1}$ for each $k \in [T]$.

Given that the Kalman Filter is an online algorithm, the greedy policy at time k is a natural strategy for this problem.

Update Equations and Eigenvalue Maximization

Note that in the exact Kalman Filter, our posterior covariance update reads:

$$\boldsymbol{P}_k = \boldsymbol{P}_k^- - \boldsymbol{P}_k^- \boldsymbol{H}_k^T (\boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^T + \boldsymbol{\Lambda}_k)^{-1} \boldsymbol{H}_k \boldsymbol{P}_k^-$$

And in CASSM, an approximate Kalman Filter, we restrict $\mathbf{y}$ to the subspace spanned by $\boldsymbol{S}$, or:

$$\hat{\boldsymbol{P}}_k = \hat{\boldsymbol{P}}_k^- - \hat{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T \boldsymbol{S}_k (\boldsymbol{S}_k^T \boldsymbol{H}_k \hat{\boldsymbol{P}}_k^- \boldsymbol{H}_k^T \boldsymbol{S}_k + \boldsymbol{S}_k^T \boldsymbol{\Lambda}_k \boldsymbol{S}_k)^{-1} \boldsymbol{S}_k^T \boldsymbol{H}_k \hat{\boldsymbol{P}}_k^-$$

We consider the problem of choosing $\boldsymbol{S}_k$, or:

$$\begin{aligned}\arg\max_{\boldsymbol{S}_k} \mathrm{H}^{(k)} &= \arg\max_{\boldsymbol{S}_k} \mathrm{H}\left(\hat{\mathbf{u}}_k^{f-}\right) - \mathrm{H}\left(\hat{\mathbf{u}}_k^{f}\right) \\ &= \arg\max_{\boldsymbol{S}_k} \frac{1}{2}\log\frac{\det \hat{\boldsymbol{P}}_k^-}{\det \hat{\boldsymbol{P}}_k}\end{aligned}$$

Define $\check{\boldsymbol{G}}_k := \boldsymbol{S}_k^\top \left(\boldsymbol{H}_k \hat{\boldsymbol{P}}_k^- \boldsymbol{H}_k^\top + \boldsymbol{\Lambda}_k\right) \boldsymbol{S}_k = \boldsymbol{S}_k^\top \boldsymbol{G}_k \boldsymbol{S}_k$. This is the projected innovation matrix.

Then:

$$\begin{aligned}\det\left(\hat{\boldsymbol{P}}_k\right) &= \det\left(\hat{\boldsymbol{P}}_k^- - \hat{\boldsymbol{P}}_k^- \boldsymbol{H}_k^\top \boldsymbol{S}_k \check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}_k^\top \boldsymbol{H}_k \hat{\boldsymbol{P}}_k^-\right) \\ &= \det\left(\sqrt{\hat{\boldsymbol{P}}_k^-}\left[\boldsymbol{I} - \sqrt{\hat{\boldsymbol{P}}_k^-}\boldsymbol{H}_k^\top \boldsymbol{S}_k \check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}_k^\top \boldsymbol{H} \sqrt{\hat{\boldsymbol{P}}_k^-}\right]\sqrt{\hat{\boldsymbol{P}}_k^-}\right) \\ &= \det\left(\sqrt{\hat{\boldsymbol{P}}_k^-}\right)^2 \det\left(\boldsymbol{I} - \sqrt{\hat{\boldsymbol{P}}_k^-}\boldsymbol{H}^\top \boldsymbol{S}_k \check{\boldsymbol{G}}_k^{-1} \boldsymbol{S}_k^\top \boldsymbol{H}_k \sqrt{\hat{\boldsymbol{P}}_k^-}\right)\end{aligned}$$

Define $\boldsymbol{L}_k := \sqrt{\hat{\boldsymbol{P}}_k^-}\boldsymbol{H}^\top \boldsymbol{S}_k$, such that $\boldsymbol{L}_k^\top \boldsymbol{L}_k = \boldsymbol{S}_k^\top \boldsymbol{H} \hat{\boldsymbol{P}}_k^- \boldsymbol{H}^\top \boldsymbol{S}_k$

$$\begin{aligned}&= \det\left(\hat{\boldsymbol{P}}_k^-\right)\det\left(\boldsymbol{I} - \boldsymbol{L}_k \check{\boldsymbol{G}}_k^{-1} \boldsymbol{L}_k^\top\right) \\ &= \det\left(\hat{\boldsymbol{P}}_k^-\right)\det\left(\boldsymbol{I} - \check{\boldsymbol{G}}_k^{-1} \boldsymbol{L}_k^\top \boldsymbol{L}_k\right) \\ &= \det\left(\hat{\boldsymbol{P}}_k^-\right)\det\left(\check{\boldsymbol{G}}_k^{-1}\left(\check{\boldsymbol{G}}_k - \boldsymbol{L}_k^\top \boldsymbol{L}_k\right)\right) \\ &= \det\left(\hat{\boldsymbol{P}}_k^-\right)\det\left(\check{\boldsymbol{G}}_k^{-1}\boldsymbol{\Lambda}_k\right) \\ &= \det\left(\hat{\boldsymbol{P}}_k^-\right)\cdot\det\left(\boldsymbol{\Lambda}_k\right)\cdot\det\left(\check{\boldsymbol{G}}_k\right)^{-1}\end{aligned}$$

Therefore, our original optimization problem becomes:

$$\begin{aligned}\arg\max_{\boldsymbol{S}_k} \mathrm{H}^{(k)} &= \arg\max_{\boldsymbol{S}_k} \frac{1}{2}\log\left(\frac{\det\left(\hat{\boldsymbol{P}}_k^-\right)}{\det\left(\hat{\boldsymbol{P}}_k^-\right)\cdot\det\left(\boldsymbol{\Lambda}_k\right)\cdot\det\left(\check{\boldsymbol{G}}_k\right)^{-1}}\right) \\ &= \arg\max_{\boldsymbol{S}_k} \frac{1}{2}\left[\log\det\left(\check{\boldsymbol{G}}_k\right) - \log(\det\left(\boldsymbol{\Lambda}_k\right))\right] \\ &= \arg\max_{\boldsymbol{S}_k} \frac{1}{2}\left[\log\det\left(\check{\boldsymbol{G}}_k\right)\right] \\ &= \arg\max_{\boldsymbol{S}_k} \frac{1}{2}\left[\sum_{i=1}^{m}\log(d_i)\right]\end{aligned}$$

where $d_i > 0$ are the eigenvalues of the positive definite matrix $\check{\boldsymbol{G}}_k$.

The diagonalization of $\boldsymbol{H}_k \hat{\boldsymbol{P}}_k^- \boldsymbol{H}_k^\top + \boldsymbol{\Lambda}_k = \boldsymbol{G}_k = \boldsymbol{U}_m \boldsymbol{D} \boldsymbol{U}_m^T$ restricted to the top m eigenvectors $\boldsymbol{U}_m$ then yields the result by the Eckart-Young Theorem.

$\boldsymbol{S}_k = \boldsymbol{U}_m$ is a solution to this optimization problem.

F Smoothing Error Bound and Uncertainty Increase

A core contribution of the CAKF [14] is the introduction of a smoothing error bound, which quantifies the increase in uncertainty from approximation during inference in the Kalman Smoother.

CASSM preserves this interpretation during inference, though model misspecification is of course a higher-level source of error.

In Figure F.1, we decompose the three cases of Figure 2 to demonstrate that the increase in uncertainty is not spurious or automatic, but *relative* to the mean prediction.

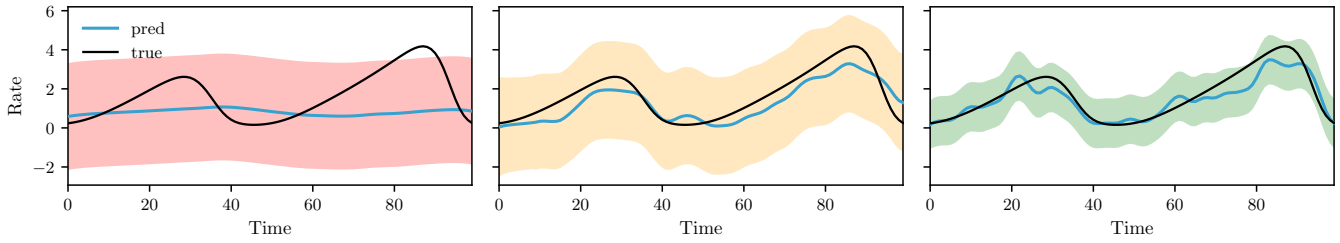


Figure F.1: Moving from left to right, we see the effect of posterior approximation: (A) we project the Lorenz observations to a 2-dimensional approximate space, (B) we project to a 10-dimensional approximate space, (C) in the Kalman filter and smoother, we compute the full posterior of a 30-dimensional state space. Thus, the increase in uncertainty is *relative* to the mean prediction, which is expected to be worse in a subspace that is insufficiently large to capure latent dynamics.

G LTI-SDEs and the Matérn Kernel Formulation

The Matérn kernel is a widely used kernel in Gaussian Process regression.

Here we give on overview of the integration between the Matérn kernel and the LTI-SDE formulation of the latent dynamics as in [22].

In the case of a continuous-time LTI-SDE, we have:

$$d\mathbf{x} = \boldsymbol{F}\mathbf{x}\,dt + \boldsymbol{L}\,d\boldsymbol{\beta},$$

The drift matrix $\boldsymbol{F}$ and dispersion matrix $\boldsymbol{L}$ are constant, and $\boldsymbol{\beta}(t)$ has diffusion matrix $\boldsymbol{D}$. The mean, $\mathbf{m}$, and covariance, $\boldsymbol{P}$, are then given by:

$$\begin{aligned}\frac{d\mathbf{m}}{dt} &= \boldsymbol{F}\mathbf{m},\\ \frac{d\boldsymbol{P}}{dt} &= \boldsymbol{F}\boldsymbol{P} + \boldsymbol{P}\boldsymbol{F}^T + \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T.\end{aligned}$$

whose solution, in the case of our discretized dynamics model:

$$\mathbf{x}(t_{k+1}) = \boldsymbol{A}_k\mathbf{x}(t_k) + \mathbf{q}_k, \quad \mathbf{q}_k \sim \mathcal{N}(0, \boldsymbol{Q}_k)$$

where $\Delta t_k = t_{k+1} - t_k$, is given by:

$$\begin{aligned}\boldsymbol{A}_k &= \exp(\boldsymbol{F}\Delta t_k),\\ \boldsymbol{Q}_k &= \int_0^{\Delta t_k} \exp(\boldsymbol{F}(\Delta t_k - \tau))\boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T \exp(\boldsymbol{F}(\Delta t_k - \tau))^T d\tau.\end{aligned}$$

For stable drift matrices, solving the Lyapunov differential equation $\frac{d\boldsymbol{P}}{dt}$ yields a steady state covariance $\boldsymbol{\Sigma}_\infty$.

Now, in general, for a space-time separable Gauss-Markov Process, the covariance factorizes along time and space, or:

$$\boldsymbol{\Sigma}\left[(t_1, \mathbf{x}_1), (t_2, \mathbf{x}_2)\right] = \boldsymbol{\Sigma}^{(\mathbf{x})}(\mathbf{x}_1, \mathbf{x}_2) \cdot \boldsymbol{\Sigma}^{(t)}(t_1, t_2)$$

This observation is the key insight that allows Gaussian Process regressions to be solved in linear time under certain classes of kernel functions $\boldsymbol{\Sigma}^{(\cdot)}$. Under the Matern($\nu = p + \frac{1}{2}$) class of kernel, the state-space representation is indeed exact [22].

Under separability, the stationary covariance admits a Kronecker factorization:

$$\boldsymbol{\Sigma}_\infty = \boldsymbol{\Sigma}^{(\mathbf{x})}(\mathbf{x}_1, \mathbf{x}_2) \otimes \boldsymbol{\Sigma}_\infty^{(t)}(t_1, t_2)$$

Similarly, the drift matrix admits a Kronecker factorization $\boldsymbol{F} = \boldsymbol{I}_{d\times d} \otimes \boldsymbol{F}^{(t)}$ as the neurons only change in time and not space.

Thus, the matrix exponential of $\boldsymbol{F}$, which is otherwise a prohibitively expensive calculation, only scales unfavorably (cubically) with the respect to the smoothness of the time kernel p_t which reasonably is a small integer value below ten.

Indeed, in the case where the time dynamics can be described by a Matern kernel, $\boldsymbol{F}^{(t)}$ and $\boldsymbol{\Sigma}_\infty^{(t)}$ are given analytically for $p_t = 0, 1, 2$, and there is no need to solve the Lyapunov differential equation.

In any case, once $\boldsymbol{\Sigma}_\infty$ is obtained, this allows the process noise covariance, for any arbitrary timepoint, to be computed as:

$$\boldsymbol{Q}_k = \boldsymbol{\Sigma}_\infty - \boldsymbol{A}_k\boldsymbol{\Sigma}_\infty\boldsymbol{A}_k^T$$

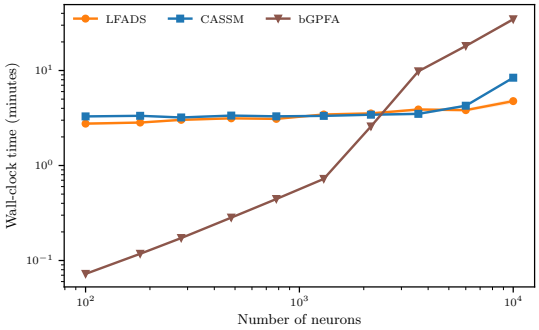


Figure H.1: Here we report time to convergence instead of a fixed computational budget. CASSM becomes slightly slower relative to LFADS but still does not begin to enter its linear scaling regime until around 10^4 neurons, whereas bGPFA still begins its quasilinear regime much earlier.

H Experiments

H.1 Hardware

All experiments except the Zebrafish experiment were run on a single NVIDIA GeForce RTX 4090 with 24GB of VRAM and an Intel Core i9-14900KF with 64GB of DRAM.

The Zebrafish experiment was run on a single NVIDIA A100 with 80GB of VRAM and an Intel Xeon Platinum CPU with 512 GB of DRAM.

H.2 Lorenz Experiments

H.2.1 Computational Budget vs. Time to Convergence

In Figure 4, we ran the algorithms for a fixed computational budget, there defined as a fixed number of passes through the data.

The computation-aware regime mainly considers the problem of limited compute, which is especially relevant for GPFA and the Kalman Filter, as running these to convergence at higher neural scales would require several days per run and thus several weeks total for a sufficient number of seeds.

Because of our problem statement, the practicality considerations of running these algorithms repeatedly, and our preliminary experiments of running algorithms to convergence, we chose to report all algorithms for a fixed number of passes through the data.

However, an equally important choice of reporting is the scaling with respect to time to convergence, which we report in Figure H.1.

Here, we show the three feasible methods at higher neural scales, and see that the ordering does not change.

CASSM takes slightly longer relative to LFADS to converge at these higher neural scales, but the trends in both perspectives mostly hold for the feasible methods.

We note that the time to convergence for GPFA and the Kalman Filter would yield sharper curves, but these methods are not the focus of the paper.

H.2.2 Hyperparameters

While LFADS has an array of hyperparameters that are chosen via the original paper [30] in that version of the Lorenz experiment, the major hyperparameters for the Lorenz experiments are as follows:

These were obtained by grid search over the following learning rates for each method:

$$\{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0\}$$

The non-LFADS methods are kernel-based and use a Matérn$\left(\frac{1}{2}\right)$ time kernel.

Note that the Laplacian kernel used in GPFA is equivalent to a Matérn$\left(\frac{1}{2}\right)$ kernel. Via space-time separability, CASSM also has a spatial Matérn$\left(\frac{3}{2}\right)$ kernel.

Training Hyperparameters		
Method	Optimizer	Learning Rate
LFADS [6]	Adam	0.100
GPFA [5]	LBFGS	0.050
bGPFA [11]	Adam	0.010
Kalman	Adam	0.001
CASSM	Adam	0.005

Table H.1: Optimizers and learning rates used for each method.

This spatial kernel has an outputscale σ^2 and one lengthscale ℓ_j per spatial input dimension, which is always 3 regardless of dataset due to the spatial location of each individual neuron.

All of the methods are given 3 latent factors, which is standard for the Lorenz experiment.

We note that bGPFA also has an additional hyperparameter, the number of Monte Carlo samples.

We pushed this up to the highest number that we could afford computationally, which was 10 samples at the larger scales (past 10^3 neurons) and 100 samples at the smaller scales.

We note that setting this too high causes out-of-memory errors, and setting it too low causes higher variance estimates.

H.3 Neural Latents Benchmark Experiments

The Neural Latents Benchmark [26] is a collection of datasets that are designed to evaluate the performance of latent variable models on real neural data.

The datasets consist of neural recordings from various brain regions and species, along with corresponding behavioral data.

H.3.1 Evaluation Metrics

The Neural Latents Benchmark (NLB) presents several evaluation metrics which assess different aspects of model performance on neural population data.

Co-smoothing (co-bps) The co-smoothing metric evaluates how well a model predicts the firing activity of held-out neurons given the activity of held-in neurons.

Concretely, models are trained on a subset of neurons and must predict firing rates for the remaining neurons.

Performance is measured as the log-likelihood improvement (in bits per spike) relative to a baseline model that predicts the trial-averaged firing rate.

This metric captures how well the inferred latent dynamics generalize across neurons and reflects the quality of the learned population-level representation [26].

Velocity R^2 (vel R^2) The velocity R^2 metric measures how well the inferred neural representations can decode behavior, like hand velocity in motor tasks.

A linear decoder is trained to map inferred firing rates (or latent features) to kinematic variables, and the coefficient of determination (R^2) is computed between predicted and true velocities.

This metric evaluates whether the model preserves behaviorally relevant information in the neural population activity and serves as a proxy for downstream decoding performance.

Peristimulus Time Histogram (psth R^2) The peristimulus R^2 metric evaluates how well the model captures trial-averaged neural responses.

For each neuron, the peristimulus time histogram (PSTH) is computed by averaging firing rates across trials aligned to a stimulus or behavioral event.

The coefficient of determination (R^2) is then computed between the predicted and true PSTHs.

Training Hyperparameters by Dataset		
Dataset 1: MC Maze		
Method	Optimizer	Learning Rate
bGPFA [11]	Adam	0.010
Kalman	Adam	0.001
CASSM	Adam	0.005
Dataset 2: MC RTT		
Method	Optimizer	Learning Rate
bGPFA [11]	Adam	0.050
Kalman	Adam	0.01
CASSM	Adam	0.001
Dataset 3: DMFC RSG		
Method	Optimizer	Learning Rate
bGPFA [11]	Adam	0.050
Kalman	Adam	0.001
CASSM	Adam	0.010
Dataset 4: Area 2 Bump		
Method	Optimizer	Learning Rate
bGPFA [11]	Adam	0.010
Kalman	Adam	0.005
CASSM	Adam	0.005

Table H.2: Optimizers and learning rates used for each method across the Neural Latents benchmark datasets.

This metric emphasizes the model's ability to capture consistent, stimulus-locked structure in neural activity while averaging out trial-to-trial variability.

H.3.2 Hyperparameters

We repeated our grid search for learning rates on the NLB datasets, except in this case the LFADS and GPFA models were tuned by the original authors of the NLB, and we used their recommended learning rates for those methods, a process described in that original paper.

The other methods were tuned by us via grid search over the same learning rates as in the Lorenz experiment. The results are summarized in Table H.2.

The models were saved at the epoch of convergence, where convergence was defined as the epoch at which the validation loss improvement was less than 10^{-5} for 5 epochs.

Unlike the previous Lorenz experiments, the number of latent factors were not fixed across methods where a ground truth existed.

Instead, we tuned the number of latent factors for each method by grid search over 5 to 50 latent factors with a step size of 5.

The best performing number of latent factors for each method and dataset was chosen by the lowest validation loss, which was consistently around 10 or 15 factors per dataset and method, likely reflecting some information about the intrinsic dimensionality of each dataset.

We note that bGPFA learns an appropriate latent dimensionality using automatic relevance determination, starting with a max dimension and letting the model turn off irrelevant latents.

H.4 Zebrafish Experiments

The Zebrafish dataset evaluation is similar to the Neural Latents Benchmark, except that the LFADS model was not tuned by the original authors of the dataset, and we had to perform our own grid search for learning rates for that method as well.

In addition, the evaluation metrics are purely fit-based as the data types differ: we are not predicting firing rates or

spikes, but fluorescence activity.

Due to that structure, this data is actually more suitable for the Gaussian likelihood, and we do not perform any normalization or preprocessing (smoothing, etc.) for CASSM in this case.

The raw data is fed directly to the model. The results are summarized in Table H.3.

We held out 25% randomly-selected test neurons for the zebrafish dataset, which was not used for training or validation, and we report test performance on that held-out data.

In addition, we initially subsampled a number of neurons, $30,000$ from approximately $95,000$ to keep our analysis consistent with our Lorenz experiments and to speed up inference given that each method was fit on a grid.

Even in this regime, we could not choose any number of latent factors or Monte Carlo samples for bGPFA without causing out-of-memory errors.

We note that the original bGPFA paper only goes up to a maximum of 200 neurons, so this is not particularly surprising.

The models were saved at the epoch of convergence, where convergence was again defined as the epoch at which the validation loss improvement was less than 10^{-5} for 5 epochs.

Training Hyperparameters		
Method	Optimizer	Learning Rate
LFADS [6]	Adam	0.100
bGPFA [11]	Adam	$--$
CASSM	Adam	0.005

Table H.3: Optimizers and learning rates used for each method.

For the number of latent factors, we performed grid search over 10 to 100 latent factors with a step size of 10, and the best performing number of latent factors for each method was chosen by the lowest validation loss, which was 20 latent factors for the CASSM variants and 30 latent factors for LFADS.