Laplace Approximated Gaussian Process State-Space Models (Supplementary Material)

Jakob Lindinger[1,2,3] **Barbara Rakitsch**[1] **Christoph Lippert**[2,3]

[1]Bosch Center for Artificial Intelligence, Renningen, Germany
[2]Hasso Plattner Institute, Potsdam, Germany
[3]University of Potsdam, Germany

A GPSSMS AND THE FITC APPROXIMATION

In this section we formalize and prove a remark on the properties of variational approximations for GPSSMs that have been used in the literature so far. More precisely, we show that the ELBO of Ialongo et al. [2019] remains unchanged when the FITC approximation [Snelson and Ghahramani, 2005], $p(F_T \mid F_M) \approx \prod_{t=0}^{T-1} p(f_t \mid F_M)$ is used in both the prior and the approximate posterior. This remark is a direct extension of the equivalent remark of Frigola et al. [2014] about the same property of their model, which can be found in their appendix.

Before we begin, we first reiterate some previous results: The ELBO of the VCDT method from Ialongo et al. [2019] is given as

$$\begin{aligned}\mathcal{L}_{\text{VCDT}} = &\sum_{t=1}^{T} \mathbb{E}_{q(x_t)}\left[\log p(y_t \mid x_t)\right] - KL\left[q(x_0) \parallel p(x_0)\right] - KL\left[q(F_M) \parallel p(F_M)\right] \\ &- \sum_{t=1}^{T} \mathbb{E}_{q(x_{t-1}, f_{t-1}, F_M)}\left[KL\left[q(x_t \mid F_M, x_{t-1}) \parallel p(x_t \mid f_{t-1}, x_{t-1})\right]\right], \end{aligned} \tag{21}$$

where the posterior marginals are given by

$$q(x_t) = \int \left[\prod_{t'=1}^{t} q(x_{t'} \mid F_M, x_{t'-1})\right] q(x_0) q(F_M) dF_M dx_{0:t-1}, \tag{22}$$

$$q(x_{t-1}, f_{t-1}, F_M) = \int \left[\prod_{t'=1}^{t-1} q(x_{t'} \mid F_M, x_{t'-1})\right] q(x_0) q(F_M) p(f_{t-1}|F_M) dx_{0:t-2}, \tag{23}$$

where we use the shorthand $dx_{i:j} = \prod_{t'=i}^{j} dx_{t'}$. This can be obtained by using the prior and approximate posterior,

$$p(Y_T, X_{T_0}, F_T, F_M) = p(x_0) p(F_T \mid F_M) p(F_M) \prod_{t=1}^{T} p(y_t \mid x_t) p(x_t \mid f_{t-1}, x_{t-1}), \tag{24}$$

$$q(X_{T_0}, F_T, F_M) = q(x_0) p(F_T \mid F_M) q(F_M) \prod_{t=1}^{T} q(x_t \mid F_M, x_{t-1}), \tag{25}$$

respectively. To clarify some of the above points, we first of all note that the expectation value is defined as $\mathbb{E}_{p(a,b)}\left[g(a,b)\right] = \int p(a,b) g(a,b) da db$, and the KL-divergence as $KL\left[q(a) \parallel p(a)\right] = \int q(a) \log \frac{q(a)}{p(a)} da$. The ELBO in Eq. (21) is the same as in Ialongo et al. [2019], where we simply collected all the terms that are somewhat spread over their paper and use our notation (introduced in Sec. 2). The marginals in Eqs. (22) and (23) are not written out in as much detail in Ialongo et al. [2019], but can be simply obtained from plugging in Eqs. (24) and (25) in the general formula for the ELBO in Eq. (9) and collecting the remaining terms. This will be almost equivalent to the very similar derivation that we perform as part of the proof below. Finally, the exact definitions of the distributions q and p appearing in Eqs. (21) - (25) [given in Ialongo et al., 2019] are not very important for our current undertaking, only their conditional (in)dependencies.

After these preliminaries we are ready to formally state our observation about the FITC approximation:

Remark 1. *The ELBO in Eq.* (21) *remains unchanged when the FITC approximation, $p(F_T \mid F_M) \approx \prod_{t=0}^{T-1} p(f_t \mid F_M)$, is used in Eqs.* (24) *and* (25)*.*

Proof of Remark 1. The proof is conceptually simple: It relies on altering Eqs. (24) and (25) with the FITC approximation and then calculating the ELBO by using Eq. (9) and showing that this coincides with Eqs. (21) - (23). The concepts needed in the proof are also very simple: We only require the definition of expectation value and KL-divergences from above, rules for the logarithm and the property for arbitrary probability densities $q(a, b)$ that

$$\int q(a,b)g(b)dadb = \int q(b)g(b)\left(\int q(a \mid b)da\right)db = \int q(b)g(b)db, \tag{26}$$

due to the normalization property of probability distributions.

We start by defining the alternative VCDT-FITC model by plugging in the FITC approximation $p(F_T \mid F_M) \approx \prod_{t=0}^{T-1} p(f_t \mid F_M)$ in Eqs. (24) and (25):

$$p_{\text{FITC}}(Y_T, X_{T_0}, F_T, F_M) = p(x_0)p(F_M)\prod_{t=1}^{T} p(f_{t-1} \mid F_M)p(y_t \mid x_t)p(x_t \mid f_{t-1}, x_{t-1}), \tag{27}$$

$$q_{\text{FITC}}(X_{T_0}, F_T, F_M) = q(x_0)q(F_M)\prod_{t=1}^{T} q(x_t \mid F_M, x_{t-1})p(f_{t-1} \mid F_M). \tag{28}$$

Plugging these equations in Eq. (9) yields the formula for the ELBO of this model:

$$\mathcal{L}_{\text{VCDT-FITC}} = \mathbb{E}_{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}\left[\log\frac{p_{\text{FITC}}(Y_T, X_{T_0}, F_T, F_M)}{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}\right] \tag{29}$$

Evaluating the terms inside the logarithm according to Eqs. (27) and (28) yields

$$\mathcal{L}_{\text{VCDT-FITC}} = \mathbb{E}_{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}\left[\sum_{t=1}^{T}\log p(y_t \mid x_t) + \log\frac{p(x_0)}{q(x_0)} + \log\frac{p(F_M)}{q(F_M)} + \sum_{t=1}^{T}\log\frac{p(x_t \mid f_{t-1}, x_{t-1})}{q(x_t \mid F_M, x_{t-1})}\right]. \tag{30}$$

Note that here the identical FITC terms, $\prod_{t=0}^{T-1} p(f_t \mid F_M)$ in the prior and approximate posterior have canceled each other out inside the logarithm, but this also happens for the terms $p(F_T \mid F_M)$ in the original VCDT formulation [Eqs. (24) and (25)].

Using the definition of the expectation value, then the property in Eq. (26) and then the definition of the KL-divergence, we identify two of the KL-terms in Eq. (21):

$$\begin{aligned}\mathcal{L}_{\text{VCDT-FITC}} = \mathbb{E}_{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}&\left[\sum_{t=1}^{T}\log p(y_t \mid x_t) + \sum_{t=1}^{T}\log\frac{p(x_t \mid f_{t-1}, x_{t-1})}{q(x_t \mid F_M, x_{t-1})}\right]\\ &- KL\left[q(x_0) \parallel p(x_0)\right] - KL\left[q(F_M) \parallel p(F_M)\right].\end{aligned} \tag{31}$$

In the following we carefully treat the remaining two terms inside the expectation: First,

$$\begin{aligned}&\mathbb{E}_{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}\left[\sum_{t=1}^{T}\log p(y_t \mid x_t)\right]\\ &= \sum_{t=1}^{T}\int q(x_0)q(F_M)\left[\prod_{t'=1}^{T} q(x_{t'} \mid F_M, x_{t'-1})\cancel{p(f_{t'-1} \mid F_M)}\right]\log p(y_t \mid x_t)dF_M dx_{0:T}\cancel{df_{0:T-1}}\\ &= \sum_{t=1}^{T}\int\left(\int q(x_0)q(F_M)\left[\prod_{t'=1}^{t} q(x_{t'} \mid F_M, x_{t'-1})\right]dF_M dx_{0:t-1}\right)\log p(y_t \mid x_t)dx_t\\ &= \sum_{t=1}^{T}\mathbb{E}_{q(x_t)}\left[\log p(y_t \mid x_t)\right],\end{aligned} \tag{32}$$

which we recognize as the first term in Eq. (21). Here, we have used Eq. (26) to get rid of the integrals over all $f_{t'}$ and over all $x_{t'}$ for $t' > t$ in the second step and we have identified $q(x_t)$ from Eq. (22) in the last step.

We note that for the term in Eq. (32) as well as for the KL-terms in Eq. (31), the difference between the FITC and the non-FITC approximate posterior does not play a role since the terms within the expectation value do not depend on any f_t. This is different for the remaining term in Eq. (31), which has a dependence on f_t:

$$\begin{aligned}
&\mathbb{E}_{q_{\text{FITC}}(X_{T_0}, F_T, F_M)}\left[\sum_{t=1}^{T} \log \frac{p(x_t \mid f_{t-1}, x_{t-1})}{q(x_t \mid F_M, x_{t-1})}\right] \\
=&\sum_{t=1}^{T} \int \left[q(x_0)q(F_M)\left(\prod_{t'=1}^{T} p(f_{t'-1} \mid F_M) q(x_{t'} \mid F_M, x_{t'-1})\right) \log \frac{p(x_t \mid f_{t-1}, x_{t-1})}{q(x_t \mid F_M, x_{t-1})} dF_M dx_{0:T} df_{0:T-1}\right] \\
=&\sum_{t=1}^{T} \int \left[q(x_0)q(F_M)p(f_{t-1} \mid F_M) \prod_{t'=1}^{t-1} q(x_{t'} \mid F_M, x_{t'-1}) dx_{0:t-2}\right. \\
&\qquad \left.\times \left(\int q(x_t \mid F_M, x_{t-1}) \log \frac{p(x_t \mid f_{t-1}, x_{t-1})}{q(x_t \mid F_M, x_{t-1})} dx_t\right) dF_M df_{t-1} dx_{t-1}\right] \\
=&-\sum_{t=1}^{T} \mathbb{E}_{q(x_{t-1}, f_{t-1}, F_M)}\left[KL\left[q(x_t \mid F_M, x_{t-1}) \parallel p(x_t \mid f_{t-1}, x_{t-1})\right]\right].
\end{aligned} \tag{33}$$

We recognize Eq. (33) as the last term in Eq. (21). Here, we have used Eq. (26) to get rid of the integrals over all $f_{t'}$ for $t' \neq t-1$ and over all $x_{t'}$ for $t' > t$ in the second step and additionally did some reordering of the terms. In the last step, we have identified $q(x_{t-1}, f_{t-1}, F_M)$ from Eq. (23) and a (negative) KL-divergence. It is interesting to note that at this point the non-FITC variational approximation still would not make a difference: Since every summand within the expectation value in the first line of Eq. (33) only depends on a single f_t, the switching of summation and expectation makes both versions, $p(F_T \mid F_M)$ and $\prod_{t=0}^{T-1} p(f_t \mid F_M)$, reduce to $p(f_{t-1} \mid F_M)$ [using Eq. (26)] in the end.

Together, Eqs. (29) - (33) show that the ELBO of the VCDT-FITC model matches that of the original VCDT model in Eq. (21), therefore completing the proof. □

We furthermore note that since Eleftheriadis et al. [2017] use the same conditional (in)dependencies in their variational family as Frigola et al. [2014], the remark in the appendix of the latter work holds also true for the former. In summary, all practically viable variational treatments of GPSSMs either explicitly use the FITC approximation in both prior and approximate posterior [Doerr et al., 2018] or implicitly do so [Frigola et al., 2014, Eleftheriadis et al., 2017, Ialongo et al., 2019], i.e., the latter methods could have used the approximation and would have arrived at the same optimization objective.[1] This somewhat weakens the criticism of Ialongo et al. [2019] on the choice of Doerr et al. [2018] to use a FITC prior and motivates us to do the same in our work.

B DERIVATION OF THE LAPLACE APPROXIMATION

In this section we provide a more extensive derivation for the Laplace approximation in our notation, see e.g. MacKay [2003], Skaug and Fournier [2006], Kristensen et al. [2016] for derivations in other notations or contexts. As explained in Sec. 2.1, we use the Laplace approximation to approximate the marginal likelihood [Eq. (2)],

$$p_\theta(Y_T) = \int p_\theta(Y_T, X_{T_0}) dX_{T_0}. \tag{34}$$

In some more detail, we do this with the following steps: We first introduce an exponential and a logarithm that cancel each other out,

$$p_\theta(Y_T) = \int \exp\left[\log p_\theta(Y_T, X_{T_0})\right] dX_{T_0}. \tag{35}$$

Then we find a mode $\hat{X}_{T_0}$ of $\log p_\theta(Y_T, X_{T_0})$,

$$\hat{X}_{T_0} = \text{argmax}_{X_{T_0}} \log p_\theta(Y_T, X_{T_0}), \tag{36}$$

[1]This does not hold for the method described as "non-factorised non-linear" in Ialongo et al. [2019], but the $\mathcal{O}(T^3)$ scaling makes this and similar methods unattractive to use in practice.

and perform a second order Taylor expansion of $\log p_\theta(Y_T, X_{T_0})$ around this mode leading to

$$p_\theta(Y_T) \approx \int \exp\left[\log p_\theta(Y_T, \hat{X}_{T_0}) - \frac{1}{2}\left(\hat{X}_{T_0} - X_{T_0}\right)^\top H \left(\hat{X}_{T_0} - X_{T_0}\right)\right] dX_{T_0}. \tag{37}$$

Here H is the negative Hessian of $\log p_\theta(Y_T, X_{T_0})$ as in Eq. (4) and the first order term vanishes since we are expanding around a mode, where, by definition, the gradient vanishes. The first term in the exponential in Eq. (37) is constant in X_{T_0} and can be pulled out of the integral:

$$p_\theta(Y_T) \approx p_\theta(Y_T, \hat{X}_{T_0}) \int \exp\left[-\frac{1}{2}\left(\hat{X}_{T_0} - X_{T_0}\right)^\top H \left(\hat{X}_{T_0} - X_{T_0}\right)\right] dX_{T_0}. \tag{38}$$

Remaining in the integral, we recognize the exponential of a negative quadratic form, i.e., an unnormalized multivariate Gaussian, which generally has the density [see e.g. MacKay, 2003],

$$\mathcal{N}(x|\mu,\Sigma) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(\mu - x)^\top \Sigma^{-1} (\mu - x)\right], \tag{39}$$

with mean μ and covariance matrix Σ. We therefore recognize $\hat{X}_{T_0}$ as the mean and H as the inverse of the covariance matrix, i.e., the precision matrix, in Eq. (38), leading to

$$p_\theta(Y_T) \approx p_\theta(Y_T, \hat{X}_{T_0}) \int \det(2\pi H^{-1})^{\frac{1}{2}} \mathcal{N}\left(X_{T_0} | \hat{X}_{T_0}, H^{-1}\right) dX_{T_0}. \tag{40}$$

This integral is very easy to solve since the factor with the determinant does not depend on X_{T_0}, leaving us with an integral over a normalized probability density which, by definition, evaluates to 1. Therefore

$$\begin{aligned} p_\theta(Y_T) &\approx p_\theta(Y_T, \hat{X}_{T_0}) \det(2\pi H^{-1})^{\frac{1}{2}} \\ &= \sqrt{2\pi}^{d_x(T+1)} p_\theta(Y_T, \hat{X}_{T_0}) \det(H)^{-\frac{1}{2}}, \end{aligned} \tag{41}$$

where we used standard rules for determinants, using that H is of dimension $d_x(T+1) \times d_x(T+1)$. This is the formula that can be found in Eq. (3).

C USING THE IMPLICIT FUNCTION THEOREM

In order to obtain a formula for $\frac{\partial \hat{X}_{T_0}(\theta)}{\partial \theta}$ we start by defining a function h, that is the Jacobian of the function g_{GP} in Eq. (15):

$$h(x, \theta, F_M) = -\frac{\partial \log p_\theta(Y_T, X_{T_0} \mid F_M)}{\partial X_{T_0}}\bigg|_{X_{T_0}=x}. \tag{42}$$

By definition, plugging in the mode $\hat{X}_{T_0}$ for the fixed value $\hat{\theta}$ of the parameters θ that has been used when obtaining $\hat{X}_{T_0}$, yields a vanishing Jacobian, $h(\hat{X}_{T_0}(\hat{\theta}), \hat{\theta}, F_M) = 0$. Here, we made the dependence of the mode on the parameter setting $\hat{\theta}$ explicit. Under mild differentiability assumptions, the IFT, roughly speaking, guarantees that $\hat{X}_{T_0}(\theta)$ is in fact a function of θ that is *implicitly defined* through a vanishing Jacobian in the vicinity of $\hat{\theta}$, i.e., through the equation

$$h(\hat{X}_{T_0}(\theta), \theta, F_M) = 0, \tag{43}$$

and that is also differentiable at $\hat{\theta}$ and in its vicinity. Therefore, we can calculate the total derivative wrt. θ on both sides of Eq. (43), yielding

$$\frac{\partial h(\hat{X}_{T_0}, \theta, F_M)}{\partial \theta} + \frac{\partial h(x, \theta, F_M)}{\partial x}\bigg|_{x=\hat{X}_{T_0}} \frac{\partial \hat{X}_{T_0}(\theta)}{\partial \theta} = 0, \tag{44}$$

where we used the chain rule. Recognizing

$$\frac{\partial h(x, \theta, F_M)}{\partial x}\bigg|_{x=\hat{X}_{T_0}} = H(\theta, F_M),$$

i.e., the Hessian H [cf. Eq. (4)] of the function g_{GP} in Eq. (15), we can solve Eq. (44) for the required derivative, obtaining

$$\frac{\partial \hat{X}_{T_0}(\theta)}{\partial \theta} = H^{-1}(\theta, F_M) \frac{\partial h(\hat{X}_{T_0}, \theta, F_M)}{\partial \theta}. \tag{45}$$

This is Eq. (17) appearing in Sec. 3.2.

D EXPLOITING STRUCTURE AND SPARSITY OF THE HESSIAN

In the following we provide some technical details on how we obtain the blocks of the Hessian in Eq. (18) in Sec. D.1 before we show in Sec. D.2 how these blocks can be used to efficiently calculate the determinant of the Hessian and performing a Hessian solve needed for Eqs. (14) and (17), respectively.

D.1 EFFICIENTLY OBTAINING THE NON-ZERO BLOCKS OF THE HESSIAN

Reverse mode automatic differentiation frameworks such as TensorFlow [Abadi et al., 2016] implement efficient vector-Jacobian products. As we can think of the Hessian as the Jacobian of the Jacobian of the function g_{GP} in Eq. (15), we can naively obtain the $d_x(T+1)$ columns of the Hessian in Eq. (18) by considering the vector-Hessian products $e_j^\top H$ with all $d_x(T+1)$ unit vectors $e_j = \{\delta_{jj'}\}_{j'=0}^{d_xT} \in \mathbb{R}^{d_x(T+1)}$, where δ_{ij} is the Kronecker delta. This would require $\mathcal{O}(T^2 d_x^2)$ storage and computation time. As in Sec. 3.3, this is wasteful since many unnecessary zeros are being calculated. Therefore, in order to tackle the problem of obtaining only the non-zero blocks of the Hessian, we propose instead to use only vector-Hessian products with the three block-vectors $\widetilde{e}_0$, $\widetilde{e}_1$, and $\widetilde{e}_2$ defined by $\widetilde{e}_k = \{\delta_{k,k'\%3}\mathbb{I}_{d_x}\}_{k'=0}^{T} \in \mathbb{R}^{d_x(T+1)\times d_x}$. Here $\%$ denotes the modulo operation and $\mathbb{I}_{d_x}$ is the identity matrix of size $d_x \times d_x$. As an example, $\widetilde{e}_0 = (\mathbb{I}_{d_x}, 0, 0, \mathbb{I}_{d_x}, \cdots)^\top$ such that

$$\widetilde{e}_0^\top H = (A_0, B_1, B_3^\top, A_3, B_4, B_6^\top, \cdots)^\top,$$

which can be implemented as d_x vector-Hessian products. Similarly $\widetilde{e}_1^\top H = (B_1^\top, A_1, B_2, \cdots)^\top$ and $\widetilde{e}_2^\top H = (0, B_2^\top, A_2, B_3 \cdots)^\top$ such that we can obtain all non-zero elements of the Hessian with only $3d_x$ vector-Hessian products, reducing the memory and time requirements to $\mathcal{O}(Td_x^2)$. After some reshaping and transposing this provides us with the quantities $\{A_t\}_{t=0}^T$ and $\{B_t\}_{t=1}^T$ which are needed for the further steps in our sparse algorithm in the following section.

D.2 EFFICIENTLY CALCULATING THE HESSIAN DETERMINANT AND PERFORMING HESSIAN SOLVES

We follow e.g. Koulaei and Toutounian [2007] in noting that the Hessian H in Eq. (18) allows the factorization

$$H = (\Lambda + B^\top)\Lambda^{-1}(\Lambda + B), \tag{46}$$

where B is the strictly upper-triangular part of H (consisting only of the different B_t blocks) and Λ is the block diagonal matrix of recursively defined blocks

$$\Lambda_0 = A_0, \qquad \Lambda_t = A_t - B_t^\top \Lambda_{t-1}^{-1} B_t, \quad t = 1, \ldots, T. \tag{47}$$

The factorization in Eq. (46) allows us to calculate the determinant of the Hessian as

$$\det H = \prod_{t=0}^{T} \det \Lambda_t, \tag{48}$$

using $\det(\Lambda + B^\top) = \det(\Lambda + B) = \det \Lambda$ (since Λ is block diagonal and B and $B^\top$ are strictly upper and lower triangular, respectively), and $\det(CD) = \det C \det D$ [see also Salkuyeh, 2006]. The Hessian solve in Eq. (17) can be done by exploiting Eq. (46) as well, yielding

$$H^{-1} = (\Lambda + B)^{-1}\Lambda(\Lambda + B^\top)^{-1}. \tag{49}$$

Therefore a solve with a block banded lower triangular matrix $(\Lambda + B^\top)$, then a matrix multiplication with a block diagonal matrix (Λ) followed by a solve with a block banded upper triangular matrix $(\Lambda + B)$ are equivalent to a solve with H. Since all of these operations can be performed in $\mathcal{O}(Td_x^3)$ steps, which is true as well for the calculation of the blocks of Λ in Eq. (47) and the determinant in Eq. (48), we have achieved the desired theoretical speed-ups.

We note that further speed-ups are possible, such as considering the inverse subset algorithm to calculate the derivative of the logarithm of the determinant of the Hessian [see e.g. Kristensen et al., 2016, Durrande et al., 2019]. Furthermore, several inherently sequential parts of the code could be written in C++ as is done in Durrande et al. [2019].

Algorithm 1 ELBO for GPSSM

```
Given time series $Y_T$
Choose number of iteration and samples $I$ and $N$, latent state dimension $d_x$
Initialize model parameters $\theta$, variational parameters $\psi = \{m, S\}$
for $i = 1 \dots I$ do                                                    ▷ Optimization steps
    for $n = 1 \dots N$ do                                                ▷ Reparameterized samples
        Sample $F_M^{(n)} \sim q_\psi(F_M)$
        Find $\hat{X}_{T_0}^{(n)}$ by maximizing $g_{\text{GP}}(X_{T_0}, \theta, F_M^{(n)})$    ▷ Eq. (15)
        Obtain non-zero elements of $H$ $(A_t, B_t)$                      ▷ Appx. D.1
        Evaluate $\det H$                                                 ▷ Eqs. (47) and (48)
        Evaluate $\widetilde{p}_\theta(Y_T \mid F_M^{(n)})$               ▷ Eq. (14)
    end for
    Evaluate $\mathcal{L}(\theta, \psi)$                                  ▷ Eqs. (16) and (20)
    Obtain gradients $\frac{\partial \mathcal{L}}{\partial \theta}$, $\frac{\partial \mathcal{L}}{\partial \psi}$, using custom $\frac{\partial \hat{X}_{T_0}^{(n)}}{\partial \theta}$, $\frac{\partial \hat{X}_{T_0}^{(n)}}{\partial \psi}$    ▷ Eqs. (17) and (49)
    Update $\theta$ and $\psi$
end for
```

E ALGORITHM AND IMPLEMENTATION DETAILS

In Alg. 1, we provide the basic algorithm to evaluate and optimize the optimization objective $\mathcal{L}(\theta, \psi)$ in Eq. (20). Then, in the following we provide some details on our implementation of Alg. 1. First general details in Sec. E.1 and then some practical details that are required to make the algorithm work on practical problems in Sec. E.2.

E.1 GENERAL IMPLEMENTATION DETAILS

We build our implementation on TensorFlow [Abadi et al., 2016] and use the multioutput GP implementation provided by GPflow [van der Wilk et al., 2020]. For finding the mode in Eq. (36) required for the Laplace approximation, we use the BFGS algorithm [see e.g. Fletcher, 1987] provided by TensorFlow-Probability. For the implementation of the custom derivative coming from the Implicit Function Theorem in Eq. (17), we use the *custom_gradient* function provided by TensorFlow.

E.2 PRACTICAL IMPLEMENTATION DETAILS

In the following we first tackle the practical details of the three extensions of Alg. 1 mentioned in Sec. 3.4: Minibatches, multi-dimensional latent states and control inputs. At the end of this section, we also discuss some practical details related to the optimization required to get the mode $\hat{X}_{T_0}$ for the Laplace approximation.

Minibatches Time series are typically too long to be handled in one batch such that using minibatches helps obtaining a computationally tractable algorithm. The first term in our optimization objective [Eq. (20)] can be written as a sum over the observations y_t [cf. Eqs. (14) and (10)]. We can approximate this using minibatches (or rather subsequences) of length T_b, $Y_{T_b} = \{y_t\}_{t=t_0}^{t_0+T_b}$ starting at some (random) time index t_0. Note that this is an approximation that ignores the temporal structure of the data and ignores effects coming from observation and transitions before and after the minibatch which is discussed in more detail in Aicher et al. [2019]. Nevertheless, minibatching nicely integrates with the sampling step in Eq. (20), where we can draw a new minibatch for every sample from $q_\psi(F_M)$, resulting in a new approximation,

$$\int q_\psi(F_M) \log \widetilde{p}_\theta(Y_T \mid F_M) dF_M \approx \frac{T}{T_b} \sum_{n=1}^{N} \log \widetilde{p}_\theta(Y_{T_b}^{(n)} \mid F_M^{(n)}), \quad F_M^{(n)} \sim q_\psi(F_M). \tag{50}$$

On a more technical note, we find that when using minibatches, we do not need a buffer to reduce the effect of the initial distribution as suggested in Aicher et al. [2019] and e.g. used in Longi et al. [2021]. This might be the case since we usually normalize the data and use an uninformative initial distribution such as $p(x_0) = \mathcal{N}(x_0|0, 1)$, such that the Laplace approximation can flexibly find a value for $\hat{x}_0$ (or $\hat{x}_{t_0-1}$ for the minibatches) that has high mass under the initial distribution.

Multi-dimensional latent states For challenging problems a one-dimensional latent state is typically not expressive enough [Frigola, 2015] and we require multi-dimensional latent states x_t. This is problematic in the transition model [Eq. (11)], where the mean and the covariance of the GP appear, both of which are one-dimensional. We follow the literature [e.g. Doerr et al., 2018, Ialongo et al., 2019] and choose independent (sparse) GPs for each dimension of the latent state resulting in

$$p_\theta(x_t \mid x_{t-1}, F_M^{d_x}) = \prod_{d=1}^{d_x} \mathcal{N}\left(x_t^{(d)} | x_{t-1}^{(d)} + \mu^{(d)}(x_{t-1}, F_M^{(d)}), Q_d + \Sigma^{(d)}(x_{t-1})\right),$$

where $F_M^{d_x} = \{F_M^{(d)}\}_{d=1}^{d_x}$ is the collection of all inducing outputs, $x_t^{(d)}$ is the d-th dimension of the latent state and $\mu^{(d)}$ and $\Sigma^{(d)}$ are the mean and covariance of the sparse GP responsible for the d-th dimension, respectively [cf. Eqs. (5) and (6)]. In order to reduce non-identifiabilities and to simplify training, we choose a diagonal transition noise variance, $Q = \text{diag}(\{Q_d\}_{d=1}^{d_x})$, the same inducing points X_M and kernel hyperparameters for all GPs, thus only leaving the variational distributions $q_{\psi_d}^{(d)}(F_M^{(d)})$ as the distinguishing property of the different dimensions of the transition function.

Control inputs Lastly, time series prediction problems often come with an additional time series $U_T = \{u_t\}_{t=1}^T$, $u_t \in \mathbb{R}^{d_u}$ of control inputs that are applied at time index t and change the behavior of the system. In GPSSMs these are modeled as additional (constant) inputs to the GP which are simply concatenated with the latent states x_t at the same time index. This leads only to two small changes in the algorithm: The kernel of the GPs have to accept input pairs coming from $\mathbb{R}^{d_x+d_u}$ and the inducing points X_M also have to live in that higher dimensional space.

Optimizing the latent states In order to obtain the mode $\hat{X}_{T_0}$ required for the Laplace approximation, we have to perform an optimization in the inner loop of our algorithm (Alg. 1). This requires some fine-tuning to get a numerically stable algorithm: First, since this is a non-stochastic problem, we use an approximate second-order optimizer for fast convergence and choose the BFGS optimizer [see e.g. Fletcher, 1987]. Second, while it is generally no longer considered as state of the art to use the Adam optimizer on all parameters including the variational ones for optimizing models involving sparse GPs [see e.g. Salimbeni et al., 2018, Adam et al., 2021], we also empirically found that this approach leads to the BFGS optimizer struggling to find good optima (or even not converging at all). We therefore opted for the following two strategies: we use the BFGS optimizer to find the mode in the inner loop and then i) the Natural gradient optimizer [Salimbeni et al., 2018] on the variational parameter ψ and the Adam optimizer on all other model parameters or ii) using a diagonal covariance for the variational distribution $q_\psi(F_M)$ and then optimizing all parameters with the Adam optimizer.

We consider the optimization for finding the mode $\hat{X}_{T_0}^{(n)}$ (see Alg. 1) as failed if (i) the maximum number of iterations has been reached without converging within the tolerance, (ii) the determinant of the Hessian [required for Eq. (14)] is negative, or iii) some of the optimal latent states $\hat{x}_t$ converge to positions far outside of the region where there are inducing points X_M (which is justified when normalizing the data as we do). In all cases we simply ignore the contributions of these terms to the optimization objective by removing the n-th term from the sum in Eq. (20). Typically this leads to the removal of well below 1% of the optimization runs.

F EXPERIMENTAL DETAILS

F.1 KINK

Training data For each of the ten different seeds, we generate a (different, noise dependent) sequence of latent states X_T according to the description in Sec. 5.1. Then we add (seed dependent) zero-mean iid. Gaussian noise with emission noise variances $\sigma_y^2 \in \{0.008, 0.08, 0.8\}$ to each of the sequences of latent states, thus generating Y_T, our training data, resulting in 30 time series in total. See also Fig. 5 for a visualization of the data.

VCDT model For the experiment in Sec. 5.1, the VCDT model from Ialongo et al. [2019] uses the same emission model, $p(y_t \mid x_t) = \mathcal{N}\left(y_t | x_t, \sigma_y^2\right)$ that is fixed to the groundtruth. The initial distribution is chosen slightly more flexible, $p(x_0) = \mathcal{N}(x_0 | m_0, \sigma_0)$, with learnable parameters m_0 and σ_0. This is important for this model as the sequential sampling in the inference requires a well tuned starting point. Another difference lies in the GP contribution to the transition function. Whereas we use a sparse GP with the FITC approximation, resulting in Eq. (11), Ialongo et al. [2019] use a proper prior on the GP and inducing outputs, resulting in the standard GP transition model $p(x_t \mid x_{t-1}) = \mathcal{N}(x_t | f_{t-1}, Q)$, where $f_{t-1} = f(x_{t-1})$. In this experiment a zero-mean GP prior is placed on the function f, as was done in the original work. See also Appx. A for a discussion on the role of the FITC prior.

VCDT training For the training of the VCDT model we took the code provided in Ialongo et al. [2019] and contacted the authors for the full experimental details. Taking their suggestions into account and after additional fine-tuning, we fix the inducing points to 50 points in the range of $[-4, 2]$ and take an RBF kernel for the GP, where we initialize the trainable variance and lengthscale to 2. We leave all other model parameters at their standard initialization values except for the Cholesky factor of the covariance matrix of the variational distribution and the transition noise which we scale by a factor of 0.1 and 0.01, respectively. For training we set the number of samples to be drawn from the posterior to 100 and take the Adam optimizer [Kingma and Ba, 2015] using a learning rate of 10^{-3} for 10^4 iterations.

Laplace training We use the same kernel and inducing point settings as for the VCDT model. We initialize the means of the variational distribution to zero-mean Gaussian noise with standard deviation 10^{-3} and the Cholesky factor of its covariance matrix to 10^{-5} times that of the prior Cholesky factor. The transition noise variance is initialized to 0.0025. For the training we use $N = 10$ samples from the variational posterior and a minibatch size of $T_b = 30$. We optimize the parameters using a combination of the Adam optimizer [Kingma and Ba, 2015] and the Natural gradient optimizer [Salimbeni et al., 2018] provided by GPflow [Matthews et al., 2017]: The Natural gradient optimizer is used on the variational parameters with an initial learning rate of 0.004 and the Adam optimizer on all other parameters with an initial learning rate of 0.0008. We train for 7000 iterations where we scale both learning rates with a factor of 0.99 every 500 steps. For the first 1000 iterations we scale the KL-divergence in our optimization objective [Eq. (16)] by a factor which we initialize at 0 and linearly increase to 1 over these iterations. We found this to help learning the sparse GP initially, which in turn improved the optimization behavior for finding the mode required for the Laplace approximation. For finding this mode, we initialize X_{T_0} to the observations Y_T adding zero-mean Gaussian noise with standard deviation 10^{-3}. We set the parameters of the *bfgs_minimize* function provided by TensorFlow-Probability to 500 for *max_iterations* and to 10^{-8} for *tolerance*, counting runs as not converged if this (gradient norm) tolerance is not achieved within the 500 iterations.

Evaluation for Tab. 1 For the evaluation of the (train) log-likelihoods we proceed as follows: We take $J = 70$ linearly spaced points $\{x_j\}_{j=1}^J$ between the minimum and maximum latent state of the ground truth time series X_{T_0} (different for every seed but not for different σ_y^2), since this is the region in which the GP should be able to reasonably learn the ground truth. For each of the trained sparse GPs we obtain (marginal) predictions for each of these test points x_j, i.e., mean and variance predictions $\mu(x_j)$, and $\Sigma(x_j)$, respectively using Eqs. (5) and (6). These are then compared with the groundtruth, i.e., the *kink* function evaluated at these locations, $f_k(x_j)$, via the average log-likelihood LL,

$$\mathrm{LL} = \frac{1}{J}\sum_{j=1}^{J} \log \mathcal{N}\left(f_k(x_j)|\mu(x_j), \Sigma(x_j)\right).$$

Evaluation for Tab. 2 For tracking the convergence of the parameters of $q(x_t|x_{t-1}, F_M)$ of the VCDT method, which is given as

$$q(x_{t+1}|x_t, F_M) = \mathcal{N}\left(x_{t+1}|A_t\mu(x_t, F_M) + b_t, S_t + A_t\Sigma(x_t)A_t^\top\right), \tag{51}$$

where $\mu(x_t, F_M)$ and $\Sigma(x_t)$ are as in Eqs. (5) and (6), respectively, we proceed as follows: For all of the ten randomly generated *kink* data sets with $T = 120$ and $\sigma_y^2 = 0.08$ we train the VCDT method and record the values A_t^i, b_t^i, and S_t^i for $t \in \{0, 40, 80, 120\}$ and for every tenth iteration i until $I = 10000$. As expected, the variance parameter S_t decreases towards the end of the optimization, indicating that the GP model can explain larger parts of the data and less transition noise is needed. The parameters A_t and b_t converge to some one-dimensional value (due to the one-dimensional latent state chosen for the *kink* data set) that is in general different for every time point t and depends on the realization of the toy data set. We define the parameters to be converged when they change only very little until the end of the optimization. More precisely, we define the iteration at which the parameters are converged as

$$I_t^{(\mathrm{con})} = (\max_i |A_t^i - A_t^I| > 0.03) + 10,$$

and equivalently for b_t. The values reported in Tab. 2 are then the parameters $A_t^{I_t^{(\mathrm{con})}}$ and $b_t^{I_t^{(\mathrm{con})}}$ averaged over the ten different realizations.

F.2 SYSTEM IDENTIFICATION

Training data We prepare the data as follows: First, we normalize the data (observations and control inputs) to have zero mean and unit variance. Then we prepare the ten splits of the data: These are deterministic and depend on the total length of the time series T_{tot}, the length of the training sequence T_{train} (which we always choose to be $T_{\text{tot}}/2$) and the length of the test sequence T_{test} on which we want to predict, which we choose to be 120. For the first data split, the training sequence starts with the first data point in the time series, while for the last data split the test sequence ends with the last data point (meaning, the training sequence starts with data point $T_{\text{tot}} - T_{\text{tot}}/2 - T_{\text{test}} = T_{\text{tot}}/2 - 120$). All other 8 data splits are chosen such that the inital points of the training sequence are linearly spaced between the extreme cases described above, thus maximizing the diversity between the training tasks.

Laplace training We use an RBF kernel with automatic relevance determination and initialize its lengthscales to $\sqrt{3}$ and its variance to 0.5^2. We choose $M = 100$ trainable inducing points and initialize them with a Sobol sequence in the hypercube of dimension $d_x + d_c = 3$ centered around the origin with side lengths equal to the largest value $|y_t|$ in the (normalized) training sequence. We initialize the means of the variational distributions to zero-mean Gaussian noise with standard deviation 5×10^{-3} and choose diagonal covariance matrices which we initialize to 10^{-6} times the identity matrix. The transition noise variance Q is initialized to 0.01 and the trainable parameters of the emission model to $b = 0$ and $\Omega = 0.005$. For the training we use $N = 10$ samples from the variational posterior and a minibatch size of $T_b = 30$. As a pretraining step that we found helpful to get the sparse GP and the inner optimization to "interact nicely with each other", we train all parameters for 50 iterations with the Adam optimizer and a learning rate of 0.003. During this pretraining we scale the KL-divergence in our optimization objective [Eq. (16)] by a factor which we initialize at 0 and linearly increase to 1, while exponentially scaling the parameter Q down to 0.001. Then, we optimize all parameters using the Adam optimizer with an initial (dataset dependent) learning rate of $0.2/T_{\text{tot}}$. We train for 20000 iterations where we scale the learning rate with a factor of 0.95 every 500 steps. For finding the mode in the inner optimization, we initialize the first dimension of X_{T_0} to the observations Y_T adding zero-mean Gaussian noise with standard deviation 10^{-3}, and we initialize the second dimension to zero-mean Gaussian noise with standard deviation 10^{-1}. We set the parameters of the *bfgs_minimize* function provided by TensorFlow-Probability to 500 for *max_iterations* and to 2×10^{-7} for *f_relative_tolerance*, counting runs as not converged if this tolerance (on the relative change of the objective value) is not achieved within the 500 iterations.

VCDT model For the VCDT model, we use the one employed in Ialongo et al. [2019], i.e., a four-dimensional latent state x_t with initial distribution $p(x_0) = \mathcal{N}(x_0|m_0, \sigma_0)$, with learnable parameters m_0 and σ_0, and the GP transition model from Eq. (7), using a fixed linear mean function (residual). The emission model is the same that we also use.

VCDT training For the training of the VCDT model we took the code provided in Ialongo et al. [2019] and, after contact with the authors, employed similar settings as in Doerr et al. [2018] for the following hyperparameters: We use an RBF kernel with automatic relevance determination and initialize its lengthscales to 2 and its variance to 0.5^2. We choose $M = 100$ trainable inducing points of dimensionality $d_x + d_c = 5$ and initialize them uniformly in the range $[-2, 2]$. We initialize the means of the variational distributions to zero-mean Gaussian noise with standard deviation 0.05 and initialize the covariance matrices to 10^{-4} times the identity matrix. The transition noise variance Q is initialized to 0.002^2, Ω to 10^{-4}, and σ_0 to 10^{-2}. For the training we use 100 latent state samples and the Adam optimizer with an initial learning rate of 0.01 running for 20000 iterations, where the learning rate is scaled by a factor of 0.98 every 500 iterations. For all other settings, we employed the default values provided in the code of Ialongo et al. [2019].

PRSSM model and training For the PRSSM model we take the exact model used in Doerr et al. [2018] and use the provided code for training, leaving the recommended settings unchanged. The only difference is the new data split in training and test sequence as explained above.

Evaluation For the evaluation all models are only provided with the test control inputs for $T_{\text{pred}} = 120$ time points after the end of the training series and with those have to predict the future observations $Y_{T_{\text{pred}}}$. This is done by getting the latent state at the end of the training sequence that then becomes x_0 for the test sequence. Starting from this, each trained model can then recursively sample future observations from the prior model [Eq. (8), with trained parameters θ]. We store the means μ_t and standard deviations σ_t over multiple recursive sampling runs at each time point $t = t_0, \ldots, t_0 + T_{\text{pred}}$, where t_0 is the initial time index of the test sequence. The way how x_0 is obtained and the recursive sampling is performed is slightly different for each model due to the different inference schemes. For PRSSM we simply follow the procedure outlined in the code. For VCDT we take the x_0 as the last state obtained in the last iteration of training, together with the sample from $q(F_M)$ that has been used to obtain those. See Ialongo et al. [2019] for more information about the inference scheme. There are multiple, 100, of such x_0 and $q(F_M)$ pairs as the inference scheme requires multiple latent state samples per iteration. The prediction coming from recursively sampling from the model with these values are then averaged over to obtain μ_t and σ_t. For our method we obtain the x_0 from the converged model as follows: We take the subsequence of Y_T of length T_b that ends at the last observation of the training sequence and obtain N samples from $q(F_M)$. For each of these samples we run

Table 3: Comparison of average mean predictive log-likelihoods (higher is better) and their standard errors for ten repetitions on five different benchmark datasets. We evaluate our model (Laplace), VCDT [Ialongo et al., 2019] and PRSSM [Doerr et al., 2018] on predictions for $T \in \{30, 60, 90, 120\}$ steps in the future. See Appx. F.2 for more details and Fig. 4 for a visualization.

DATASET	MODEL	T=30	T=60	T=90	T=120
ACTUATOR	LAPLACE	-0.6(0.2)	-0.8(0.2)	-1.7(0.7)	-1.7(0.5)
	VCDT	-0.6(0.3)	-0.9(0.3)	-1.0(0.3)	-1.0(0.2)
	PRSSM	-1.0(0.3)	-0.8(0.2)	-0.8(0.1)	-0.8(0.2)
BALLBEAM	LAPLACE	1.6(0.4)	1.2(0.3)	0.9(0.3)	1.0(0.2)
	VCDT	2.5(0.7)	0.0(1.6)	-2.2(2.2)	-2.2(2.0)
	PRSSM	-1.8(1.4)	-1.6(0.9)	-2.7(1.4)	-2.4(1.5)
DRIVE	LAPLACE	-1.7(0.5)	-1.5(0.3)	-1.4(0.2)	-1.4(0.1)
	VCDT	-1.2(0.1)	-1.2(0.0)	-1.2(0.0)	-1.2(0.0)
	PRSSM	-4.8(1.6)	-3.9(1.0)	-3.4(0.6)	-3.3(0.5)
DRYER	LAPLACE	-0.3(0.1)	-0.2(0.1)	-0.3(0.1)	-0.3(0.1)
	VCDT	-3.2(0.6)	-3.2(0.5)	-2.8(0.4)	-2.9(0.4)
	PRSSM	-0.9(0.5)	-0.3(0.2)	-0.1(0.2)	-0.0(0.2)
FURNACE	LAPLACE	-2.2(0.1)	-2.4(0.1)	-2.5(0.1)	-2.6(0.1)
	VCDT	-14.0(1.5)	-14.2(0.9)	-15.6(0.8)	-21.2(1.0)
	PRSSM	-10.1(2.9)	-8.5(2.5)	-8.4(2.6)	-11.8(3.6)

the inner loop of Alg. 1 thus providing us with N optimal latent state sequences $\hat{X}_{T_0}^{(n)}$ from which we take the last state as an estimate for x_0 for the prediction task. We use the same N samples from $q(F_M)$, sample $S = 100$ times for each N from the prior model, thus providing an average over SN samples, yielding μ_t and σ_t. The log-likelihood is then evaluated as

$$\text{LL} = \frac{1}{T_{\text{test}}} \sum_{t=t_0}^{t_0+T_{\text{test}}} \log \mathcal{N}\left(y_t | \mu_t, \sigma_t\right)$$

and the RMSE as

$$\text{RMSE}^2 = \frac{1}{T_{\text{test}}} \sum_{t=t_0}^{t_0+T_{\text{test}}} (y_t - \mu_t)^2,$$

for $T_{\text{test}} \in \{30, 60, 90, 120\}$, respectively. Both these quantities are calculated on the unnormalized data, i.e., after reversing the normalization for the predictions.

G ADDITIONAL EXPERIMENTS

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, and others. Tensorflow: a system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

Vincent Adam, Paul E. Chang, Mohammad Emtiyaz Khan, and Arno Solin. Dual Parameterization of Sparse Variational Gaussian Processes. In *Advances in Neural Information Processing Systems*, 2021.

Christopher Aicher, Yi-An Ma, Nicholas J. Foti, and Emily B. Fox. Stochastic Gradient MCMC for State Space Models. *SIAM Journal on Mathematics of Data Science*, 1(3):555–587, 2019.

Andreas Doerr, Christian Daniel, Martin Schiegg, Duy Nguyen-Tuong, Stefan Schaal, Marc Toussaint, and Trimpe Sebastian. Probabilistic Recurrent State-Space Models. In *International Conference on Machine Learning*, 2018.

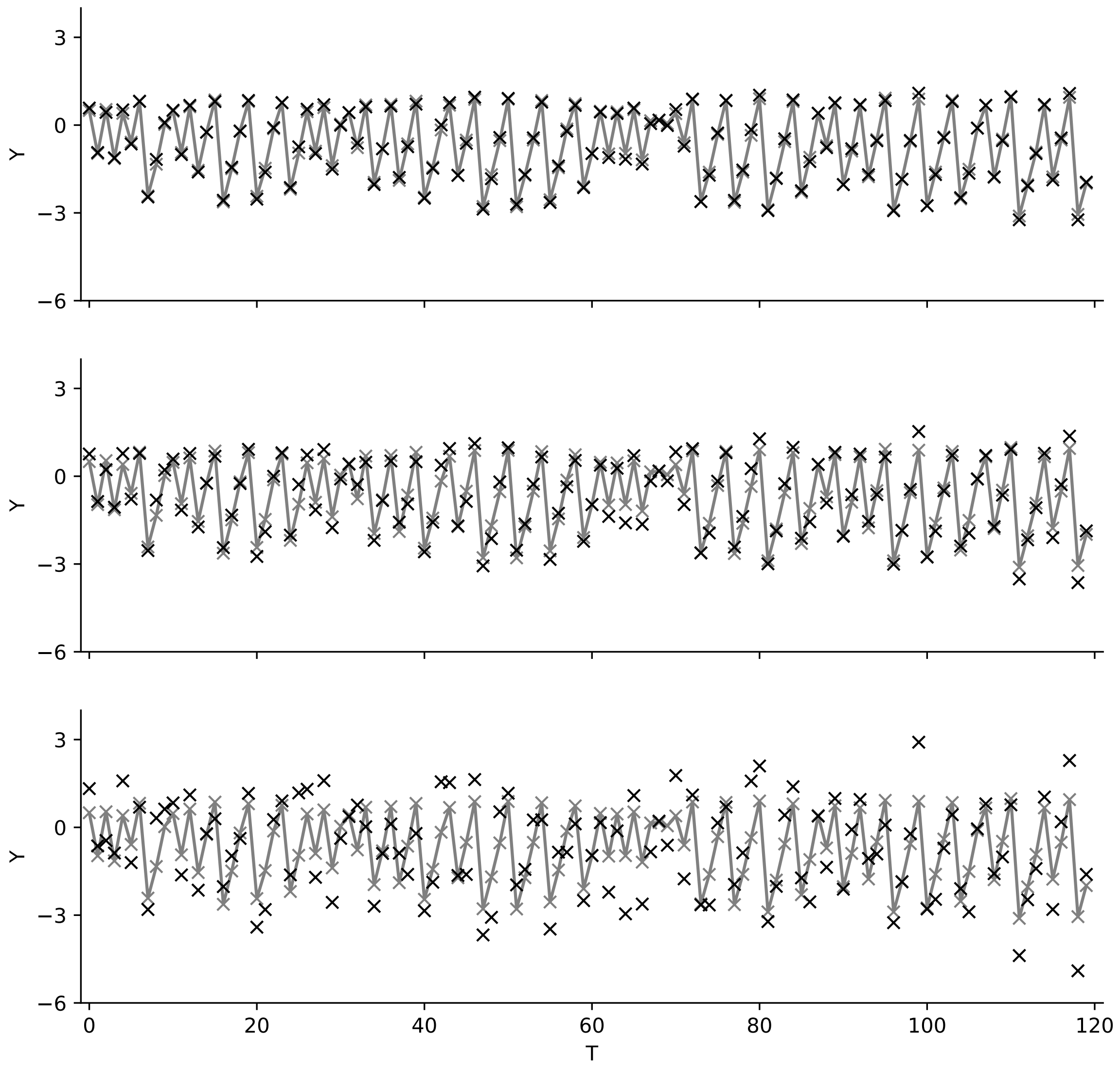

Figure 5: Visualization of the time series created by the *kink* transition function in Sec. 5.1. Shown are the latent states x_t in gray and the observations y_t in black for three different emission noise variance $\sigma_y^2 \in \{0.008, 0.08, 0.8\}$ (top to bottom). Note that the underlying latent states are the same in all plots.

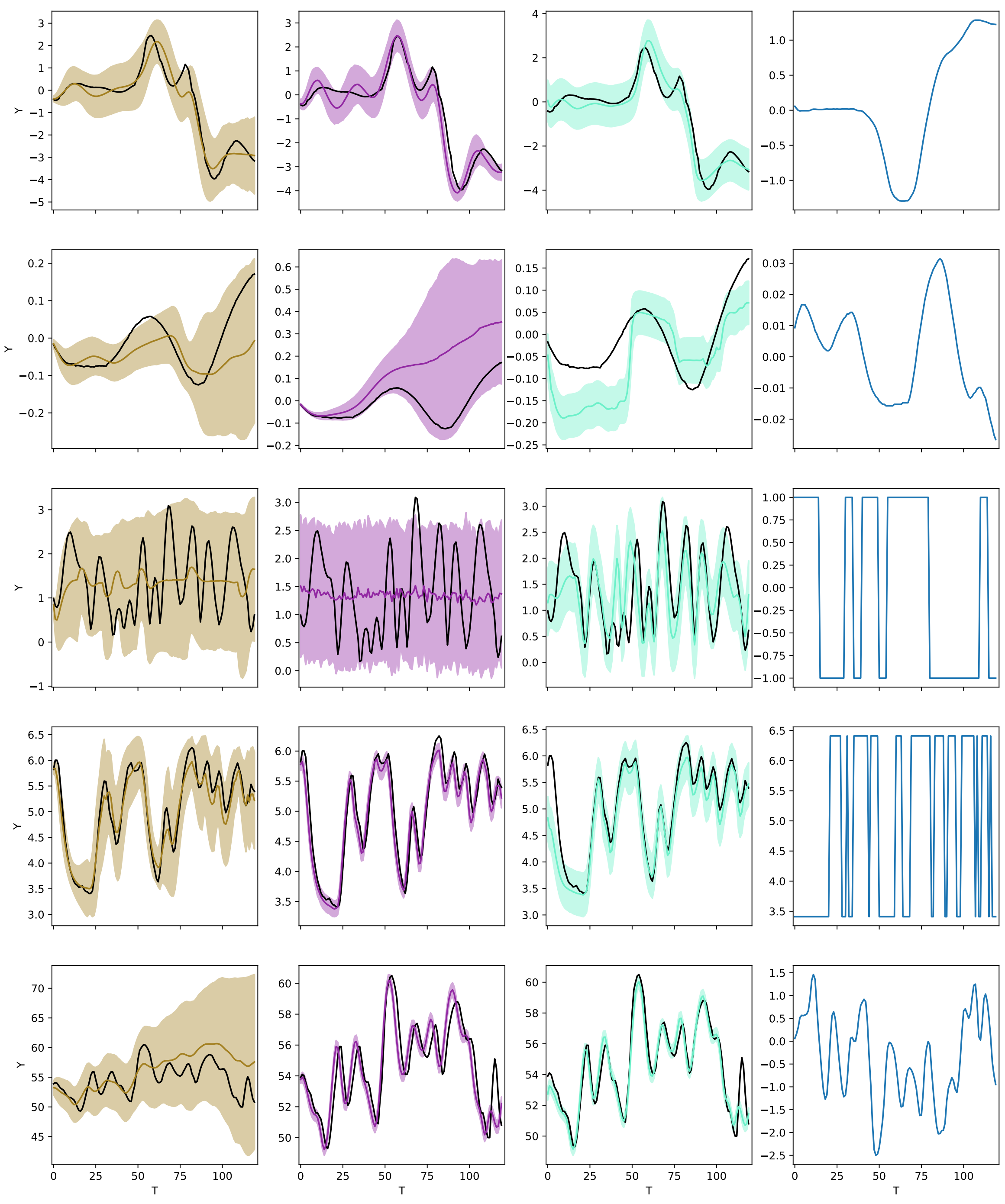

Figure 6: Exemplary figure showing the differences in predictions for the three different methods on the system identification data sets. Shown are predictions for the same seeds on the *Actuator*, *Ballbeam*, *Drive*, *Dryer*, and *Gas Furnace* datasets in the different rows (top to bottom). From left to right are the predictions for our method, VCDT [Ialongo et al., 2019], PRSSM [Doerr et al., 2018], and the control inputs (all unnormalized).

Table 4: Comparison of average RMSEs (lower is better) and their standard errors for ten repetitions on five different benchmark datasets. We evaluate our model (Laplace), VCDT [Ialongo et al., 2019] and PRSSM [Doerr et al., 2018] on predictions for $T \in \{30, 60, 90, 120\}$ steps in the future. See Appx. F.2 for more details.

DATASET	MODEL	T=30	T=60	T=90	T=120
ACTUATOR	LAPLACE	0.39(0.20)	0.42(0.18)	0.52(0.24)	0.55(0.23)
	VCDT	0.40(0.06)	0.45(0.07)	0.52(0.07)	0.54(0.07)
	PRSSM	0.53(0.09)	0.50(0.08)	0.53(0.05)	0.52(0.05)
BALLBEAM	LAPLACE	0.05(0.03)	0.07(0.05)	0.08(0.05)	0.08(0.05)
	VCDT	0.03(0.01)	0.07(0.03)	0.11(0.04)	0.14(0.04)
	PRSSM	0.06(0.01)	0.06(0.01)	0.07(0.01)	0.07(0.01)
DRIVE	LAPLACE	0.90(0.21)	0.88(0.11)	0.90(0.11)	0.94(0.20)
	VCDT	0.76(0.03)	0.77(0.01)	0.77(0.01)	0.77(0.01)
	PRSSM	0.58(0.07)	0.58(0.06)	0.56(0.04)	0.55(0.03)
DRYER	LAPLACE	0.31(0.07)	0.33(0.08)	0.39(0.24)	0.58(0.55)
	VCDT	0.22(0.01)	0.22(0.01)	0.21(0.01)	0.22(0.01)
	PRSSM	0.31(0.07)	0.26(0.06)	0.24(0.06)	0.23(0.06)
FURNACE	LAPLACE	2.15(1.05)	2.82(1.36)	3.48(1.64)	3.60(1.80)
	VCDT	1.07(0.05)	1.11(0.02)	1.15(0.01)	1.33(0.03)
	PRSSM	1.40(0.30)	1.75(0.55)	2.11(0.82)	2.34(0.86)

Nicolas Durrande, Vincent Adam, Lucas Bordeaux, Stefanos Eleftheriadis, and James Hensman. Banded Matrix Operators for Gaussian Markov Models in the Automatic Differentiation Era. In *International Conference on Artificial Intelligence and Statistics*, 2019.

Stefanos Eleftheriadis, Tom Nicholson, Marc Deisenroth, and James Hensman. Identification of Gaussian Process State Space Models. In *Advances in Neural Information Processing Systems*, 2017.

R. Fletcher. *Practical Methods of Optimization*. Wiley, 1987.

Roger Frigola. *Bayesian time series learning with Gaussian processes*. PhD thesis, University of Cambridge, 2015.

Roger Frigola, Yutian Chen, and Carl E. Rasmussen. Variational Gaussian Process State-Space Models. In *Advances in Neural Information Processing Systems*, 2014.

Alessandro Davide Ialongo, Mark van der Wilk, James Hensman, and Carl Edward Rasmussen. Overcoming Mean-Field Approximations in Recurrent Gaussian Process Models. In *International Conference on Machine Learning*, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

M. H. Koulaei and F. Toutounian. On computing of block ILU preconditioner for block tridiagonal systems. *Journal of Computational and Applied Mathematics*, 202(2):248–257, 2007.

Kasper Kristensen, Anders Nielsen, Casper W. Berg, Hans Skaug, and Bradley M. Bell. TMB : Automatic Differentiation and Laplace Approximation. *Journal of Statistical Software*, 70(5):1–21, 2016.

Krista Longi, Jakob Lindinger, Olaf Duennbier, Melih Kandemir, Arto Klami, and Barbara Rakitsch. Traversing Time with Multi-Resolution Gaussian Process State-Space Models. *arXiv preprint arXiv:2112.03230*, 2021.

David MacKay. *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40):1–6, 2017.

Hugh Salimbeni, Stefanos Eleftheriadis, and James Hensman. Natural Gradients in Practice: Non-Conjugate Variational Inference in Gaussian Process Models. In *International Conference on Artificial Intelligence and Statistics*, 2018.

Davod Khojasteh Salkuyeh. Comments on "A note on a three-term recurrence for a tridiagonal matrix". *Applied Mathematics and Computation*, 176(2):442–444, 2006.

Hans J. Skaug and David A. Fournier. Automatic approximation of the marginal likelihood in non-Gaussian hierarchical models. *Computational Statistics & Data Analysis*, 51(2):699–709, 2006.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems*, 2005.

Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A Framework for Interdomain and Multioutput Gaussian Processes. *arXiv preprint arXiv:2003.01115*, 2020.