Temporal Gaussian Process Regression in Logarithmic Time

Adrien Corenflos*, Zheng Zhao†, and Simo Särkkä*
*Department of Electrical Engineering and Automation
Aalto University, Finland
†Department of Information Technology
Uppsala University, Sweden

***Abstract*—The aim of this article is to present a novel parallelization method for temporal Gaussian process (GP) regression problems. The method allows for solving GP regression problems in logarithmic $O(\log N)$ time, where N stands for the number of observations and test points. Our approach uses the state-space representation of GPs which, in its original form, allows for linear $O(N)$ time GP regression by leveraging Kalman filtering and smoothing methods. By using a recently proposed parallelization method for Bayesian filters and smoothers, we are able to reduce the linear computational complexity of the temporal GP regression problems into logarithmic span complexity. This ensures logarithmic time complexity when parallel hardware such as a graphics processing unit (GPU) are employed. We experimentally show the computational benefits of our approach on simulated and real datasets via our open-source implementation leveraging the GPflow framework.**

***Index Terms*—Gaussian process, state space, parallelization, logarithmic time, Kalman filter and smoother**

I. Introduction

Gaussian processes (GPs) are a family of function-space priors used to solve regression and classification problems arising in machine learning [1]. In their naive form their complexity scales as $O(N^3)$, where N is the number of training data points, which is problematic for large datasets. For a large class of covariance functions, the associated GP regression problem can be reformulated as a smoothing problem for a linear state-space model [2], [3]. This reduces the GP regression problem into a Kalman smoothing algorithm with linear time complexity $O(N)$. This improvement comes at the cost of a loss of precision for all covariance functions that do not have an exact state-space representation [4]. The linear complexity is optimal on single-threaded computational architectures, as processing data needs to be done sequentially. However, it is suboptimal on hardware where parallelization is possible, such as multi-core central processing units (CPUs) or, more importantly, on massively threaded architectures such as graphics processing units (GPUs). The aim of this letter is therefore to develop parallel state-space GP (PSSGP) methods which reduce the computational complexity (in the sense of parallel span complexity) of state-space GPs to logarithmic $O(\log N)$ (see Fig. 1 in experiments). To do so, we leverage the parallel Bayesian filtering and smoothing methodology presented in [5].

Over the recent years, several other approaches to parallelization of GPs have been proposed. For instance, in [6], [7] the authors consider mini-batching the dataset to form mixtures of local GP experts. This incurs a cubic cost only in the size of the batches, and achieves additional problem decomposition that could potentially be combined with our approach. More closely related to this letter are the works in [8], [9] which proposed to leverage the sparse Markovian structure of Markovian and state-space GPs (SSGPs). Specifically, they use parallel matrix computations, thereby reaching $O(\log N)$ span complexity in the dataset size in some special cases. However, the methods outlined in [8], [9] effectively require computations with large (albeit sparse) matrices, and their logarithmic span complexity is hard to guarantee for all the subproblems [8]. Orthogonally to these parallelization efforts, different approximation methods have been introduced in order to reduce the computational complexity of GPs. These include, for example, inducing points, spectral sampling, and basis function methods (see, e.g., [1], [10]–[13]).

The contribution of our paper is three-fold:

1) We combine the state-space formulation of GPs with parallel Kalman filters and smoothers [5].
2) We extend the parallel formulation to missing measurements to allow for predicting with state-space GPs.
3) We experimentally show the computational gains of our proposed methods on simulated and real datasets[1].

II. Gaussian processes in state-space form

In this section, we quickly recall results about the state-space formulation of Gaussian processes before we present their temporal parallelization formulation. Given a covariance function $C(t, t')$ and a set of observations $\{y_k \colon k = 1, \dots, N\}$, a temporal GP regression problem of the form

$$\begin{aligned} f(t) &\sim \mathrm{GP}(0, C(t, t')), \\ y_k &= f(t_k) + e_k, \qquad e_k \sim \mathcal{N}(0, \sigma_k^2), \end{aligned} \tag{1}$$

This work was supported Academy of Finland (projects 321900 and 321891) and Aalto ELEC doctoral school. Email: adrien.corenflos@aalto.fi.

[1]We implemented the method as an open-source extensible library. The code can be found at https://github.com/EEA-sensors/parallel-gps.

can be converted into a smoothing problem for an n_x-dimensional continuous-discrete state-space model

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{G}\,\mathbf{x}(t) + \mathbf{L}\,\mathbf{w}(t), \quad y_k = \mathbf{H}\,\mathbf{x}(t_k) + e_k, \tag{2}$$

where $\mathbf{x}$ is the state, y_k is the measurement, $\mathbf{w}$ is a white noise process with a constant spectral density matrix $\mathbf{Q}$, and e_k is the Gaussian measurement noise [2], [3]. The dimension n_x of the state, as well as the matrices $\mathbf{G}$, $\mathbf{L}$, $\mathbf{H}$, and $\mathbf{Q}$ in the model, depend on (and define) the covariance function at hand.

In the state-space formulation (2), the Gaussian process in Equation (1) has the representation $f(t) = \mathbf{H}\,\mathbf{x}(t)$. In the case of Matérn covariance functions, this representation is exact and available in closed form [2]. Other stationary covariance functions, such as the squared exponential, can be approximated up to an arbitrary precision by using Taylor series or Páde approximants [2], [4], [14]–[17] in the spectral domain.

The continuous-time state-space model (2) can be discretized into an equivalent discrete-time linear Gaussian state-space model (LGSSM, e.g., [18]) of the form

$$\mathbf{x}_k = \mathbf{F}_{k-1}\,\mathbf{x}_{k-1} + \mathbf{q}_{k-1} \quad y_k = \mathbf{H}\,\mathbf{x}_k + e_k, \tag{3}$$

where $\mathbf{q}_{k-1} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_{k-1})$. Then, the GP regression problem can be solved by applying Kalman filtering and smoothing algorithms on model (3) in $O(N)$ time [2].

In the rest of the article, we show how the sequential Kalman filters and smoothers used in SSGP can be replaced by their parallel versions [5]. This reduces the computational complexity of SSGP regression to $O(\log N)$. Additionally, by combining these with automatic differentiation softwares (e.g., TensorFlow [19]), we also show how this parallelization benefits GP parameter learning.

III. Handling Missing Observations in Parallel Kalman Filter

In [5], the authors introduce an equivalent formulation of Kalman filters and smoothers in terms of an associative operator. This enables them to leverage distributed implementations of scan (prefix-sum) algorithms, such as [20] and [21], in order to reduce the time complexity of Kalman filtering and smoothing down to $O(\log N)$. However, this formulation does not take into account missing observations. This prevents its application for inference in state-space GP models, where test points are treated as missing data [3].

The method introduced in [5] consists in writing the filtering step in terms of an associative operator of a sequence of five elements $(\mathbf{A}_k, \mathbf{b}_k, \mathbf{C}_k, \boldsymbol{\eta}_k, \mathbf{J}_k)$, which are first initialized in parallel and then combined using parallel associative scan [20], [21]. At the initialization step of the original algorithm, these elements need to be computed so as to correspond to the following quantities:

$$p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, y_k) = \mathcal{N}(\mathbf{x}_t \mid \mathbf{A}_k\mathbf{x}_{k-1} + \mathbf{b}_k, \mathbf{C}_k), \tag{4}$$

$$p(y_k \mid \mathbf{x}_{k-1}) = \mathcal{N}_I(\mathbf{x}_{k-1} \mid \boldsymbol{\eta}_k, \mathbf{J}_k), \tag{5}$$

where $\mathcal{N}_I$ denotes the information form of the Gaussian distribution. However, when no observation is available at step k, these equations do not hold directly and need to be modified.

By redoing the original derivation, it turns out that, in the case of missing measurements, the posterior density $p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, y_k)$ should be replaced by the transition density $p(\mathbf{x}_k \mid \mathbf{x}_{k-1}) = \mathcal{N}(\mathbf{x}_k \mid \mathbf{F}_{k-1}\mathbf{x}_{k-1}, \mathbf{Q}_{k-1})$ for $k > 1$ and $p(\mathbf{x}_1)$ for $k = 1$. Specifically, in the case of missing measurements, the initialization equations for $\mathbf{A}_k$, $\mathbf{b}_k$, $\mathbf{C}_k$, $\boldsymbol{\eta}_k$, and $\mathbf{J}_k$ can be written as $\boldsymbol{\eta}_k = \mathbf{0}$, $\mathbf{J}_k = \mathbf{0}$, for all k, and

$$\mathbf{A}_k = \mathbf{F}_{k-1}, \quad \mathbf{b}_k = \mathbf{0}, \quad \mathbf{C}_k = \mathbf{Q}_{k-1}, \tag{6}$$

for $k > 1$, while, for $k = 1$, they become

$$\mathbf{A}_1 = \mathbf{0}, \quad \mathbf{b}_1 = \mathbf{0}, \quad \mathbf{C}_1 = \mathbf{P}_\infty. \tag{7}$$

When the quantities $\mathbf{A}_k$, $\mathbf{b}_k$, $\mathbf{C}_k$, $\boldsymbol{\eta}_k$, and $\mathbf{J}_k$ have been initialized for time steps with and without observations, they can be combined using parallel scan, with the associative operator $\otimes$ defined in the same way as in [5]:

$$\begin{aligned}
\mathbf{A}_{ij} &= \mathbf{A}_j \left(\mathbf{I}_{n_x} + \mathbf{C}_i\mathbf{J}_j\right)^{-1} \mathbf{A}_i, \\
\mathbf{b}_{ij} &= \mathbf{A}_j \left(\mathbf{I}_{n_x} + \mathbf{C}_i\mathbf{J}_j\right)^{-1} \left(\mathbf{b}_i + \mathbf{C}_i\boldsymbol{\eta}_j\right) + \mathbf{b}_j, \\
\mathbf{C}_{ij} &= \mathbf{A}_j \left(\mathbf{I}_{n_x} + \mathbf{C}_i\mathbf{J}_j\right)^{-1} \mathbf{C}_i\mathbf{A}_j^\top + \mathbf{C}_j, \\
\boldsymbol{\eta}_{ij} &= \mathbf{A}_i^\top \left(\mathbf{I}_{n_x} + \mathbf{J}_j\mathbf{C}_i\right)^{-1} \left(\boldsymbol{\eta}_j - \mathbf{J}_j\mathbf{b}_i\right) + \boldsymbol{\eta}_i, \\
\mathbf{J}_{ij} &= \mathbf{A}_i^\top \left(\mathbf{I}_{n_x} + \mathbf{J}_j\mathbf{C}_i\right)^{-1} \mathbf{J}_j\mathbf{A}_i + \mathbf{J}_i.
\end{aligned}$$

Then, running a parallel scan algorithm on the elements above with the operator $\otimes$ produces a sequence of "prefix-sum" elements $\{(\mathbf{A}_k^*, \mathbf{b}_k^*, \mathbf{C}_k^*, \boldsymbol{\eta}_k^*, \mathbf{J}_k^*) \colon k = 1, \ldots, N\}$. Finally, the terms $\overline{\mathbf{x}}_k \triangleq \mathbf{b}_k^*$ and $\mathbf{P}_k \triangleq \mathbf{C}_k^*$ will correspond to the filtering mean and covariance at time step k, respectively.

Proposition 1 (Equivalence of sequential and parallel Kalman filters)**.** *For any $k = 1, 2, \ldots, N + M$, the Kalman filter means and covariances are given by $\overline{\mathbf{x}}_k = \mathbf{b}_k^*$ and $\mathbf{P}_k = \mathbf{C}_k^*$, respectively.*

Proof. The detailed proof is omitted due to space limitations, but the result follows by explicitly writing down the forward recursion for the elements $(\mathbf{A}_k^*, \mathbf{b}_k^*, \mathbf{C}_k^*, \boldsymbol{\eta}_k^*, \mathbf{J}_k^*)$, and checking that equations for $\mathbf{b}_k^*$ and $\mathbf{P}_k = \mathbf{C}_k^*$ coincide with the Kalman filter equations. □

On the other hand, the parallel smoothing algorithm needs no modifications with respect to [5] in order to handle missing observations, as it only relies on the result of the filtering algorithm and not directly on the observations.

IV. Temporal parallelization of Gaussian Processes

An immediate consequence of the equivalence of the parallel and sequential Kalman filters and smoothers [5] is the fact that the parallel and sequential versions of SSGP are equivalent too. In this section, we provide the details of the steps needed to create the linear Gaussian state-space model (LGSSM) representation of SSGPs. The resulting end-to-end algorithm is automatically differentiable and has a total span complexity of $O(\log N)$, from training to inference.

A. Computation of the steady-state covariance

To represent a stationary GP, one must start the state-space model SDE from the stationary initial state $\mathbf{x}(t_0) \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_\infty)$ given by the Lyapunov equation [2]. The complexity of this step is independent of the number of time steps and does not need time-parallelization. There exist a number of iterative methods for solving this kind of algebraic equations [22]. However, in order to make automatic differentiation efficient, in this work, we use the closed-form vectorization solution given in [23] (p. 229). This relies on matrix algebra, and does not need any explicit looping. This solution is feasible because we only need to numerically solve the Lyapunov equation for small state dimensions. Furthermore, as this solution only involves matrix inversions and multiplications, it is easily parallelizable on GPU architectures.

B. Balancing of the state space model

In practice, the state-space model (3) obtained via discretization is often numerically unstable due to the transition matrix having a poor conditioning number. This in turn results in inaccuracies in computing both the GP predictions and the marginal log-likelihood of the observations. To alleviate this issue, we need to resort to balancing algorithms [24] in order to obtain a transition matrix $\mathbf{F}$ which has rows and columns that have approximately equal norms, thereby obtaining a more stable state-space model. Formally, for any diagonal matrix $\mathbf{D} \in \mathbb{R}^{n_x \times n_x}$, the continuous-discrete model

$$\begin{aligned} \frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} &= \mathbf{D}^{-1}\,\mathbf{G}\,\mathbf{D}\,\mathbf{z}(t) + \mathbf{D}^{-1}\,\mathbf{L}\,\mathbf{w}(t), \\ y_k &= \mathbf{H}\,\mathbf{D}\,\mathbf{z}_k + e_k, \end{aligned} \tag{8}$$

with its initial condition given by $\mathbf{z}(t_0) \sim \mathcal{N}(\mathbf{0}, \mathbf{D}^{-1}\,\mathbf{P}_\infty\mathbf{D})$, is equivalent to the state-space model (2) started at $\mathbf{x}(t_0) \sim \mathcal{N}(\mathbf{0}, \mathbf{P}_\infty)$, in the sense that for all $t \geq t_0$ we have $f(t) = \mathbf{H}\,\mathbf{D}\,\mathbf{z}(t)$.

In particular, this means that the gradient of the log-likelihood $\log p(y_1, \ldots, y_N \mid \boldsymbol{\theta})$ with respect to the parameter $\boldsymbol{\theta}$ is left unchanged by the choice of the scaling matrix $\mathbf{D}$. This property allows us to condition our state-space representation of the original GP using a scaling matrix $\mathbf{D}$ computed with the iterative methods described in [24]. It also allows us to compute the gradient of the log-likelihood of the observations with respect to the GP parameters as if $\mathbf{D}$ did not depend on $\mathbf{F}$ and, therefore, on the parameter $\boldsymbol{\theta}$. This is crucial to obtain a stable gradient by avoiding to unroll the gradient through the iteration necessary to compute $\mathbf{D}$.

C. Converting GPs into discrete-time state space

In order to use the parallel formulation of Kalman filters and smoothers in Section III, we need to first form the continuous state-space model representation from the initial Gaussian process definition. This operation is independent of the number of measurements and, therefore, has a complexity of $O(1)$. When it has been formed, we then need to transform it into a discrete-time LGSSM as given by Equation (3). In practice, the discretization can be implemented using, for example, matrix fractions or various other methods [18], [25]. These operations are fully parallelizable across the time dimension and, therefore, have a span complexity of $O(1)$ when run on a parallel architecture.

It is worth noting that, in the parameters learning phase, the discretization needs only to happen for the training data points. However, when predicting, it is necessary to insert the M requested prediction times at the right location in the training data so as to be able to run the Kalman filter and smoother routines. When done naively, this operation has complexity $O(M + N)$. However, it can be done in parallel with span complexity $O(\log(\min(M, N)))$ by using merging operation [26]. In addition, for some GP models, such as Matérn GPs the discretization can be done offline, as it admits closed-form solutions [18].

D. End-to-end complexity of parallelized state-space GPs

The complexity analysis of the six stages for running the parallellized state-space GPs are the following:

1) Forming the continuous state-space model has both $O(1)$ work and span complexities.
2) Discretizing the state-space model has $O(N)$ work complexity and $O(1)$ span complexity.
3) At training time, the parallel Kalman filtering operations have $O(N)$ total work complexity and $O(\log N)$ total span complexity.
4) At training time, automatic differentiation shares the same computational graph structure as the parallel Kalman filter. Therefore, it has the same work and span complexities: $O(N)$ and $O(\log N)$, respectively.
5) At prediction time, merging the training and test data has work complexity $O(N + M)$ and span complexity $O(\log(\min(M, N)))$.
6) At prediction time, the parallel Kalman filtering and smoothing operations have $O(N + M)$ total work complexity and $O(\log(N + M))$ total span complexity.

Putting together the above we can conclude that the total work and span complexities of doing end-to-end inference to prediction in parallelized state-space GPs are $O(N + M)$ and $O(\log(N + M))$, respectively. The memory complexity is similar, although higher, than that of the sequential algorithm, whereby we need to store all the parameters $(\mathbf{A}_k, \mathbf{b}_k, \mathbf{C}_k, \boldsymbol{\eta}_k, \mathbf{J}_k)$ at each time step, resulting in $O(Nn_x^2)$ memory complexity.

V. Experiments

In this section, we show the benefits of our approach for inference and prediction on simulated and real datasets. Because GPUs are inherently massively parallelized architectures, they are not optimized for sequential data processing, and have a lower clock speed than cost-comparable CPUs. This makes running the standard state-space GPs on a GPU less attractive than running them on a CPU, contrarily to standard GPs which can leverage GPU-enabled linear algebra routines. In order to offer a fair comparison between our proposed methodology, the standard GPs, and the standard state-space GPs, we have

therefore chosen to run the sequential implementation of state-space GP on a CPU while we run the standard GP and our proposed parallel state-space GP on a GPU. We verified empirically that running the standard state-space GP on the same GPU architecture resulted in a tremendous performance loss for it ($\sim 100\times$ slower), justifying running it on a CPU for benchmarking. All the results were obtained using an AMD® Ryzen Threadripper 3960X CPU with 128 GB DDR4 RAM, and an Nvidia® GeForce RTX 3090 GPU with 24GB memory.

A. Simulation model

We first study the behavior of our proposed methodology on a simple noisy sinusoidal model given by

$$\begin{aligned} f(t) &= \sin(\pi\, t) + \sin(2\,\pi\, t) + \sin(3\,\pi\, t), \\ y_k &= f(t_k) + r_k, \end{aligned} \tag{9}$$

with observations and prediction times being equally spaced on $(0, 4)$. By increasing the number of training points we can measure the empirical time complexity (in terms of wall clock) of our proposed parallel state-space GP (PSSGP) method compared to the standard GP and state-space GP (SSGP).

We have taken the covariance function to be the squared exponential (approximated to the 6th order for the state-space GPs), corresponding to $n_x = 6$. The training dataset size ranges from 2^{12} to 2^{15} points, while the test dataset contains $10\,000$ points. As it can be inferred from Figure 1, our proposed method consistently outperforms standard GP and SSGP across the chosen range of dataset sizes with standard GP eventually running out of memory for larger datasets.

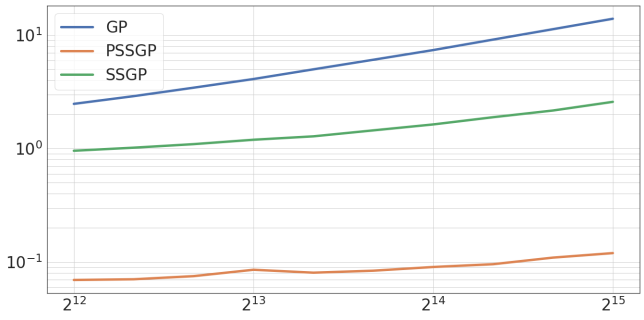

Figure 1. Average run time in seconds required to predict $M = 10\,000$ test points for noisy sinusoidal data with RBF covariance function for standard GP, SSGP, and PSSGP (ours). The number of training data points N is given on the x-axis.

B. Sunspots dataset

In this section, we compare regression and parameter learning via likelihood maximization using L-BFGS [27], [28] on the monthly sunspot activity dataset provided by World Data Center SILSO, Royal Observatory of Belgium, Brussels[2]. We learn the GP parameters on the whole dataset which contains $N = 3\,200$ points. Then, we interpolate the data on every single day from 1749-01-31 to 2018-07-31. This results in $96\,000$ prediction points.

[2]The data is available at http://www.sidc.be/silso/datafiles.

Table I
RUNNING TIME OF LEARNING THE GP PARAMETERS ON THE SUNSPOT DATASET RELATIVE TO PSSGP. PSSGP TOOK 39, 46, AND 48 MILLISECONDS PER FUNCTION/GRADIENT EVALUATION WHEN $N = 1\,200, 2\,200$, AND $3\,200$, RESPECTIVELY.

N	GP	SSGP	PSSGP
1200	1.03	12.08	**1**
2200	3.82	25.7	**1**
3200	10.1	43.86	**1**

The running times of the different algorithms are shown in Table I. PSSGP is respectively 10 and 43 times faster than GP and SSGP, when $N = 3\,200$. Interpolating daily took 0.14s for PSSGP, while SSGP on our CPU and standard GP on our GPU were respectively 23 and 33 times slower.

C. CO2 concentration dataset

In order to understand the impact of the dimensionality of the SDE, we finally consider the Mauna Loa carbon dioxide concentration dataset[3]. Specifically, to model the periodic pattern of the data, we use the composite covariance function $C_{\mathrm{co2}}(t - t') = C_{\mathrm{Per.}}(t - t')\, C_{\mathrm{Mat.}}(t - t') + C_{\mathrm{Mat.}}(t - t')$ and convert its periodic component to its state-space form using its Taylor expansion [14][4] up to order 1, 2, and 3. This results in SDEs of dimensions $n_x = 10$, 14, and 18, respectively. Then we perform HMC sampling (see, e.g., [29]) on the GP regression model parameters. We selected monthly and weekly data from year 1974 to 2021, which contains 3192 training points.

Table II
RELATIVE TIME OF SAMPLING FROM THE PARAMETERS POSTERIOR DISTRIBUTION USING HMC WITH 10 LEAPFROG STEPS ON THE CO2 DATASET. THE GP TOOK 3 SECONDS PER SAMPLE.

Order	GP	SSGP	PSSGP
1	1	4.5	**0.55**
2	**1**	5.73	1.36
3	**1**	6.9	2.55

As it can be inferred from Table II, while PSSGP is still competitive compared to SSGP for high dimensional SDE representations, its complexity increases with the dimension to the point where it eventually does not outperform the standard GP anymore.

VI. DISCUSSION

We have presented a sublinear algorithm for learning and inference in state space Gaussian processes, leveraging and extending the parallel Kalman filter and smoother introduced in [5]. This has allowed us to dramatically reduce the training time for regression problems on large datasets as evidenced

[3]The data is available at https://www.esrl.noaa.gov/gmd/ccgg/trends/.

[4]Our choice of covariance function is slightly different from the one suggested in [14] where the authors also add an RBF covariance function term to C_{co2}. However, we did not see any improvement from adding this supplementary degree of freedom and therefore left it out.

by our experiments on synthetic and real data. However, our final experiment has also revealed that PSSGP scales worse than SSGP as the dimension of the state in the state-space representation of the GP regression problem increases. This is due to the necessity of solving a system of n_x equations in the current parallel form of Kalman filtering, whereas the sequential form only requires to solve $n_y < n_x$ ones. Finally, recent works [30] show that similar parallelization techniques can also be used for non-linear state-space models, which could then make it possible to exploit the present methodology for Gaussian process classification and deep state-space Gaussian processes [31].

REFERENCES

[1] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[2] J. Hartikainen and S. Särkkä, "Kalman filtering and smoothing solutions to temporal Gaussian process regression models," in *Proceedings of the 2010 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2010, pp. 379–384.

[3] S. Särkkä, A. Solin, and J. Hartikainen, "Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing," *IEEE Signal Processing Magazine*, vol. 30, no. 4, pp. 51–61, 2013.

[4] S. Särkkä and R. Piché, "On convergence and accuracy of state-space approximations of squared exponential covariance functions," in *Proceedings of the 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2014, pp. 1–6.

[5] S. Särkkä and Á. F. García-Fernández, "Temporal parallelization of Bayesian smoothers," *IEEE Transactions on Automatic Control*, vol. 66, no. 1, pp. 299–306, 2021.

[6] H. Liu, J. Cai, Y. Wang, and Y. S. Ong, "Generalized robust Bayesian committee machine for large-scale Gaussian process regression," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 3131–3140.

[7] M. M. Zhang and S. A. Williamson, "Embarrassingly parallel inference for Gaussian processes," *Journal of Machine Learning Research*, vol. 20, no. 169, pp. 1–26, 2019.

[8] A. Grigorievskiy, N. Lawrence, and S. Särkkä, "Parallelizable sparse inverse formulation Gaussian processes (SpInGP)," in *Proceedings of the 2017 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017, pp. 1–6.

[9] F. Lindgren, H. Rue, and J. Lindström, "An explicit link between Gaussian fields and Gaussian Markov random fields: The stochastic partial differential equation approach," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 73, no. 4, pp. 423–498, 2011.

[10] J. Quiñonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[11] M. Lázaro-Gredilla, J. Quiñonero-Candela, C. E. Rasmussen, and A. R. Figueiras-Vidal, "Sparse spectrum Gaussian process regression," *Journal of Machine Learning Research*, vol. 11, pp. 1865–1881, 2010.

[12] J. Hensman, N. Durrande, and A. Solin, "Variational Fourier features for Gaussian processes," *Journal of Machine Learning Research*, vol. 8, no. 151, pp. 1–52, 2017.

[13] A. Solin and S. Särkkä, "Hilbert space methods for reduced-rank Gaussian process regression," *Statistics and Computing*, vol. 30, no. 2, pp. 419–446, 2020.

[14] ——, "Explicit link between periodic covariance functions and state space models," in *Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 33, 2014, pp. 904–912.

[15] ——, "Gaussian quadratures for state space approximation of scale mixtures of squared exponential covariance functions," in *Proceedings of the 2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2014, pp. 1–6.

[16] T. Karvonen and S. Särkkä, "Approximate state-space Gaussian processes via spectral transformation," in *Proceedings of the 2016 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2016, pp. 1–6.

[17] F. Tronarp, T. Karvonen, and S. Särkkä, "Mixture representation of the Matérn class with applications in state space approximations and Bayesian quadrature," in *Proceedings of the 2018 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, 2018, pp. 1–6.

[18] S. Särkkä and A. Solin, *Applied Stochastic Differential Equations*. Cambridge University Press, 2019.

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[20] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on Computers*, vol. 38, no. 11, pp. 1526–1538, 1989.

[21] ——, "Prefix sums and their applications," School of Computer Science, Carnegie Mellon University, Tech. Rep. CMU-CS-90-190, 1990.

[22] A. S. Hodel, B. Tenison, and K. R. Poolla, "Numerical solution of the Lyapunov equation by approximate power iteration," *Linear Algebra and its Applications*, vol. 236, pp. 205–230, 1996.

[23] W. L. Brogan, *Modern Control Theory*, 3rd ed. Pearson, 2011.

[24] E. E. Osborne, "On pre-conditioning of matrices," *Journal of the ACM*, vol. 7, no. 4, pp. 338–345, 1960.

[25] P. Axelsson and F. Gustafsson, "Discrete-time solutions to the continuous-time differential Lyapunov equation with applications to Kalman filtering," *IEEE Transactions on Automatic Control*, vol. 60, no. 3, pp. 632–643, 2014.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. MIT press, 2009.

[27] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software*, vol. 23, no. 4, p. 550–560, 1997.

[28] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer, 1999.

[29] R. M. Neal, "MCMC using Hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo*. Chapman & Hall/CRC, 2011, ch. 5.

[30] F. Yaghoobi, A. Corenflos, S. Hassan, and S. Särkkä, "Parallel iterated extended and sigma-point Kalman smoothers," in *Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 5350–5354.

[31] Z. Zhao, M. Emzir, and S. Särkkä, "Deep state-space Gaussian processes," *Statistics and Computing*, vol. 31, no. 6, p. 75, 2021.