Tensor network square root Kalman filter for online Gaussian process regression

Menzen, Clara; Kok, Manon; Batselier, Kim

DOI
10.1016/j.automatica.2025.112694

Publication date
2026

Document Version
Final published version

Published in
Automatica

Citation (APA)
Menzen, C., Kok, M., & Batselier, K. (2026). Tensor network square root Kalman filter for online Gaussian process regression. *Automatica*, *183*, Article 112694. https://doi.org/10.1016/j.automatica.2025.112694

Automatica 183 (2026) 112694

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Tensor network square root Kalman filter for online Gaussian process regression☆

Clara Menzen *, Manon Kok, Kim Batselier

Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628 CD, Delft, The Netherlands

ARTICLE INFO

Article history:
Received 3 September 2024
Received in revised form 18 September 2025
Accepted 30 September 2025

Keywords:
Square root Kalman filtering
Tensor network
Gaussian processes
Recursive estimation

ABSTRACT

The state-of-the-art tensor network Kalman filter lifts the curse of dimensionality for high-dimensional recursive estimation problems. However, the required rounding operation can cause filter divergence due to the loss of positive definiteness of covariance matrices. We solve this issue by developing, for the first time, a tensor network square root Kalman filter, and apply it to high-dimensional online Gaussian process regression. In our experiments, we demonstrate that our method is equivalent to the conventional Kalman filter when choosing a full-rank tensor network. Furthermore, we apply our method to a real-life system identification problem where we estimate 4^{14} parameters on a standard laptop. The estimated model outperforms the state-of-the-art tensor network Kalman filter in terms of prediction accuracy and uncertainty quantification.

1. Introduction

In a time when data-driven AI models are trained on an exponentially growing amount of data, it is crucial that the models can be adapted to newly observed data without retraining from scratch. These online or recursive settings are present in many fields including system identification (Batselier, Chen, & Wong, 2017b; Doyle, Pearson, & Ogunnaike, 2002), sensor fusion (Solin, Kok, Wahlström, Schön, & Särkkä, 2018; Viset, Helmons, & Kok, 2022), robotics (Liu, Chowdhary, Castra da Silva, Liu, & How, 2018; Nguyen-Tuong, Peters, & Seeger, 2008), and machine learning (Hartikainen & Särkkä, 2010; Ranganathan, Yang, & Ho, 2010; Stanton, Maddox, Delbridge, & Wilson, 2021).

While Bayesian algorithms, like widely-used Gaussian processes (GPs) (Rasmussen & Williams, 2006) are well-suited for an online setting, they are associated with potentially high computational costs. Standard GP regression using a batch of N observations has a cubic cost in N, i.e., $\mathcal{O}(N^3)$. The number of observations is growing in an online setting, so the cost increases each time step and can become a computational bottleneck.

There are numerous parametric approximations to address scalability in batch settings, including sparse GPs (Quinonero-Candela & Rasmussen, 2005) and reduced-rank GPs (Solin & Särkkä, 2020), which both have a complexity of $\mathcal{O}(NM^2)$, M being the number of inducing inputs and basis function for the respective method. Structured kernel interpolation for sparse GPs (Wilson & Nickisch, 2015) reduces the complexity further to $\mathcal{O}(N + DM^{1+1/D})$, D being the number of input dimensions.

Parametric approximations allow for a straightforward recursive update, where the posterior distribution from the previous time step is used as a prior for the current time step (Särkkä & Svensson, 2023). In this context, online GPs have been used, e.g., for GP state-space models (Berntorp, 2021; Särkkä, Solin, & Hartikainen, 2013; Svensson & Schön, 2017), rank-reduced Kalman filtering (Schmidt, Hennig, Nick, & Tronarp, 2023) and recursive sparse GPs (Stanton et al., 2021).

In this paper, we consider the online parametric GP model given by

$$\begin{aligned} y_t &= \boldsymbol{\phi}(\mathbf{x}_t)^\top \mathbf{w}_t + \epsilon_t, \quad \epsilon_t \sim \mathcal{N}(0, \sigma_y^2), \\ \mathbf{w}_{t-1} &\sim \mathcal{N}(\hat{\mathbf{w}}_{t-1}, \mathbf{P}_{t-1}), \end{aligned} \tag{1}$$

where y_t is a scalar observation at discrete time t, $\boldsymbol{\phi}(\cdot)$ are basis functions that map a D dimensional input vector $\mathbf{x}_t$ to a feature space, $\mathbf{w}_t \in \mathbb{R}^M$ are the parameters at time t, and σ_y^2 denotes the variance of the measurement noise ϵ_t which is assumed to be i.d.d. and zero-mean Gaussian. With (1), the posterior distribution $p(\mathbf{w}_t \mid \mathbf{x}_{1:t}, \mathbf{y}_{1:t}) = \mathcal{N}(\hat{\mathbf{w}}_t, \mathbf{P}_t)$ — i.e., the distribution of $\mathbf{w}_t$ given all inputs and measurements up until time t, $\mathbf{x}_{1:t} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t]$, $\mathbf{y}_{1:t} = [y_1, y_2, \ldots, y_t]$ — is computed at each time step using the estimate $\hat{\mathbf{w}}_{t-1}$ and covariance matrix $\mathbf{P}_{t-1}$ from the previous time step as a prior.

☆ Kim Batselier was supported by the project Sustainable learning for Artificial Intelligence from noisy large-scale data (with project number VI.Vidi.213.017) which is financed by the Dutch Research Council (NWO). The material in this paper was not presented at any conference. This paper was recommended for publication in revised form by Associate Editor Tianshi Chen under the direction of Editor Alessandro Chiuso.

* Corresponding author.
E-mail addresses: cm.menzen@gmail.com (C. Menzen), m.kok-1@tudelft.nl (M. Kok), k.batselier@tudelft.nl (K. Batselier).

https://doi.org/10.1016/j.automatica.2025.112694
0005-1098/

We consider commonly used product kernels with a feature map given by

$$\boldsymbol{\phi}(\mathbf{x}_t) = \boldsymbol{\phi}^{(1)}(\mathbf{x}_t) \otimes \cdots \otimes \boldsymbol{\phi}^{(d)}(\mathbf{x}_t) \otimes \cdots \otimes \boldsymbol{\phi}^{(D)}(\mathbf{x}_t), \tag{2}$$

where $\boldsymbol{\phi}^{(d)}(\mathbf{x}_t) \in \mathbb{R}^I$ with I being the number of basis functions in the dth dimension, and $\otimes$ denoting a Kronecker product. The resulting number of basis functions is $M = I^D$, growing exponentially with the input dimension D. Requiring exponentially many parameters in a high-dimensional setting is a known problem, discussed in the related literature: In Svensson and Schön (2017) separable kernels or a radial basis function expansion are proposed as an alternative with the disclaimer of limiting the space of functions that is possible to describe. In Stanton et al. (2021) dimensionality reduction is applied for all experiments with $D > 3$. Alternatively, several tensor network (TN)- based methods have been proposed to break this curse of dimensionality and achieve a linear computational complexity in D. In the batch setting, Batselier, Chen, and Wong (2017a) and Wesel and Batselier (2021) give solutions for the squared exponential and polynomial kernel, respectively. In the online setting, the state-of-the-art method is the tensor network Kalman filter (TNKF) (Batselier et al., 2017b; Batselier, Ko, & Wong, 2019), where the Kalman filter time and measurement update are implemented in TN format.

While the TNKF lifts the curse of dimensionality, it has a significant drawback. The TNKF requires a TN-specific rounding operation (Oseledets, 2011), which can result in covariance update losing positive (semi-) definiteness (De Rooij, Batselier, & Hunyadi, 2023), resulting in the divergence of the filter.

This paper resolves this issue by computing the square root covariance factor in tensor train (TT) format instead. Our approximation represents the $M \times M$ square root covariance factor as a tensor train matrix (TTm). This is motivated by prior square root covariance factors of product kernels having a Kronecker product structure, which corresponds to a rank-1 TTm. In addition, work by Nickson, Gunter, Lloyd, Osborne, and Roberts (2015) and Izmailov, Novikov, and Kropotov (2018) approximates the covariance matrix as a rank-1 TTm. This work generalizes the rank-1 approximation to higher ranks which results in better prediction accuracy and uncertainty quantification. We call our method the tensor network square root Kalman filter (TNSRKF).

We show in experiments that the TNSRKF is equivalent to the standard Kalman filter when choosing full-rank TTs. In addition, we show how different choices of TT-ranks affect the performance of our method. Finally, we compare the TNSRKF to the TNKF in a real-life system identification problem with 4^{14} parameters and observe that, contrary to the TNKF, our method does not diverge.

2. Problem formulation

Similar to the TNKF, we build on standard equations for the measurement update of the Kalman filter, given by

$$\mathbf{S}_t = \boldsymbol{\phi}_t^\top \mathbf{P}_{t-1} \boldsymbol{\phi}_t + \sigma_y^2 \tag{3}$$

$$\mathbf{K}_t = \mathbf{P}_{t-1} \boldsymbol{\phi}_t \mathbf{S}_t^{-1} \tag{4}$$

$$\hat{\mathbf{w}}_t = \hat{\mathbf{w}}_{t-1} + \mathbf{K}_t (y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1}) \tag{5}$$

$$\mathbf{P}_t = (\mathbf{I}_M - \mathbf{K}_t \boldsymbol{\phi}_t^\top) \mathbf{P}_{t-1} (\mathbf{I}_M - \mathbf{K}_t \boldsymbol{\phi}_t^\top)^\top + \sigma_y^2 \mathbf{K}_t \mathbf{K}_t^\top, \tag{6}$$

where $\mathbf{S}_t$ denotes the innovation covariance and $\mathbf{K}_t$ denotes the Kalman gain. Note that for a scalar measurement, $\mathbf{S}_t$ is a scalar and $\mathbf{K}_t$ a vector, whereas in the case of multiple measurements per time step, they are matrices. Without the loss of generality, we present the scalar case, where, beyond the scope of this paper, our approach can easily be extended to vector measurements. We recursively update the posterior distribution of the parametric weights from (1), i.e., $p(\mathbf{w}_t \mid \mathbf{x}_{1:t}, \mathbf{y}_{1:t})$. For product kernels with a feature map given in (2), it is $\mathbf{w}_t \in \mathbb{R}^{I^D}$ and $\mathbf{P}_t \in \mathbb{R}^{I^D \times I^D}$. In this case, the Kalman filter suffers from the curse of dimensionality.

The first tensor-based Kalman filter, the TNKF (Batselier et al., 2017b), solved the curse of dimensionality and implements (3)–(6) in TT format, where the weights are represented as a TT and the covariance matrix as a TTm. During the updates, the algebraic operations in TT format increase the TT-ranks of the involved variables, according to Batselier et al. (2019, Lemma 2). To counteract the rank increase and keep the algorithm efficient, the TNKF requires an additional step called TT-rounding (Oseledets, 2011). This SVD-based operation transforms the TT or TTm to ones with smaller TT-ranks. TT-rounding can result, however, in the loss of positive (semi-) definiteness.

To avoid this issue, we implement the square root formulation of the Kalman filter (SRKF), as described e.g. in Grewal and Andrews (2014, Ch. 7), in TT format. The SRKF expresses (3)–(6) in terms of a square root decomposition $\mathbf{P}_t = \mathbf{L}_t \mathbf{L}_t^\top$, with the square root covariance factor $\mathbf{L}_t$ given by

$$\mathbf{L}_t = \begin{bmatrix} (\mathbf{I}_M - \mathbf{K}_t \boldsymbol{\phi}_t^\top) \mathbf{L}_{t-1} & \sigma_y \mathbf{K}_t \end{bmatrix}. \tag{7}$$

In each update, (7) is computed by concatenating two matrices, such that the number of columns of $\mathbf{L}_t$ increases. For the next update, $\mathbf{L}_t$ needs to be transformed back to its original size. In the SRKF, this is done by computing a thin QR-decomposition (Golub & Van Loan, 2013, p. 248) of $\mathbf{L}_t$ given by

$$\underbrace{\mathbf{L}_t^\top}_{(M+1)\times M} = \underbrace{\mathbf{Q}_t}_{(M+1)\times M} \underbrace{\mathbf{R}_t}_{M \times M} \tag{8}$$

and replacing $\mathbf{L}_t$ by $\mathbf{R}_t^\top$, i.e., by the transpose of $\mathbf{R}_t$.

The orthogonal $\mathbf{Q}_t$-factor can be discarded since

$$\mathbf{P}_t = \mathbf{L}_t \mathbf{L}_t^\top = \mathbf{R}_t^\top \underbrace{\mathbf{Q}_t^\top \mathbf{Q}_t}_{\mathbf{I}_M} \mathbf{R}_t = \mathbf{R}_t^\top \mathbf{R}_t. \tag{9}$$

In TT format, performing the QR-decomposition as in (8) is not possible. We solve this issue by proposing an SVD-based algorithm in TT format that truncates $\mathbf{L}_t$ back to its original size.

3. Background on tensor networks

3.1. Tensor networks

Tensor networks (TNs), also called tensor decompositions, are an extension of matrix decompositions to higher dimensions. There are multiple TN architectures, including the CANDECOMP/PARAFAC decomposition (Kolda & Bader, 2009), the Tucker decomposition (Tucker, 1966), and the tensor train (TT) decomposition (Oseledets, 2011). In this paper, we focus on TTs to approximate the weight vector's mean as discussed in Section 3.1.1, and a TT matrix (TTm) (Oseledets, 2010) to approximate the square root covariance factor, as discussed in Section 3.1.2.

In this context, we denote TTs representing vectors as a lowercase bold letter, e.g. $\mathbf{w}_t$, and their components, called TT-cores, as capital calligraphic bold letters, e.g. $\mathcal{W}^{(d)}$. TT matrices are denoted by upper-case bold letters, e.g. $\mathbf{L}_t$ and their corresponding TTm-cores as capital calligraphic bold letters, e.g. $\mathcal{L}^{(d)}$.

3.1.1. Tensor train vectors

As depicted in Fig. 1(a), a TT vector consists of interconnected three-way tensors, called TT-cores, visualized as nodes with three edges. Each edge corresponds to an index of a TT-core and connected edges are summations over the involved indices. Each TT-core is connected by two edges, called TT-ranks, to its

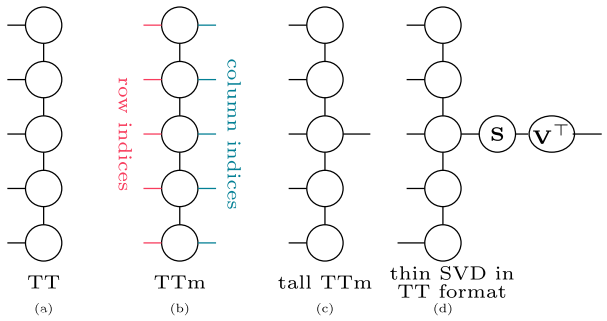

Fig. 1. Visual depiction of tensor diagrams for a (a) TT, (b) TTm, (c) tall TTm and (d) thin SVD.

neighboring TT-cores, except for the first and last TT-core, whose outer TT-ranks are by definition equal to one.

For the purpose of this paper, consider a TT that represents the mean of the weight vector $\mathbf{w}_t \in \mathbb{R}^M$. The TT-cores, denoted by $\mathcal{W}_t^{(1)}, \ldots, \mathcal{W}_t^{(d)}, \ldots, \mathcal{W}_t^{(D)}$ with $\mathcal{W}_t^{(d)} \in \mathbb{R}^{R_d \times I \times R_{d+1}}$ for $d = 1, \ldots D$, where R_d and R_{d+1} are the TT-ranks and I is the size of the non-connected edge such that $M = I^D$. By definition $R_1 = R_{D+1} = 1$. Without the loss of generality, we use TT-cores with equal TT-ranks $R_{\mathbf{w}}$. The storage complexity of $\mathbf{w}_t$ without TNs is $\mathcal{O}(I^D)$ and in TT format $\mathcal{O}(DIR_{\mathbf{w}}^2)$, where lower TT-ranks $R_{\mathbf{w}}$ will result in more efficient representations.

An important characteristic of a TT for numerical stability is that it can be transformed into the site-d-mixed canonical format.

Definition 1 (*Site-d-mixed canonical format Schollwöck (2011)*)**.** A TT $\mathbf{w}_t$ in site-d-mixed canonical format is given by

$$\mathbf{w}_t = \mathbf{G}_{d,t}\mathbf{w}_t^{(d)}, \tag{10}$$

where $\mathbf{G}_{d,t} \in \mathbb{R}^{M \times R_{\mathbf{w}} I R_{\mathbf{w}}}$ is an orthogonal matrix computed from all TT-cores except the dth and $\mathbf{w}_t^{(d)} \in \mathbb{R}^{R_{\mathbf{w}} I R_{\mathbf{w}}}$ is the vectorization of the dth TT-core. In this format, the TT representation is linear in the dth TT-core when all other TT-cores are fixed.

3.1.2. TT matrices and tall TT matrices

A TTm consists of interconnected four-way tensors, as depicted in Fig. 1(b). Analogous to the TT, the TTm components and connected edges are called TTm-cores and TTm-ranks, respectively, where each TTm-core has two free edges, the row and column indices.

For the purpose of this paper, consider a TTm representation of the square root covariance factor $\mathbf{L}_t \in \mathbb{R}^{M \times M}$. The TTm-cores are denoted by $\mathcal{L}_t^{(1)}, \ldots, \mathcal{L}_t^{(d)}, \ldots, \mathcal{L}_t^{(D)}$ with $\mathcal{L}_t^{(d)} \in \mathbb{R}^{R_d \times I \times J \times R_{d+1}}$, where I and J are the number of row and column indices, indicated in Fig. 1(b) as red and blue edges respectively, such that $M = I^D$ and $M = J^D$. By definition, $R_1 = R_{D+1} = 1$, and for this paper, we generally assume that all other TTm-ranks $R_2 = \cdots = R_D = R_{\mathbf{L}}$ are equal. The storage complexity of $\mathbf{L}_t$ without TNs is $\mathcal{O}(I^D \times I^D)$ and in TTm format $\mathcal{O}(DR_{\mathbf{L}}^2 IJ)$.

A TTm can also be written in terms of the site-d-mixed canonical format as defined in Definition 1, but it requires to be transformed into a TT first. This can be done by combining the row and column indexes into one index, which represents a kind of vectorization of the matrix represented by the TTm. Note, however, that the indices are not ordered as in conventional vectorization. A site-d-mixed canonical format of a TTm is given by

$$\mathrm{vec}(\mathbf{L}_t) = \mathbf{H}_{d,t}\mathbf{l}_t^{(d)}, \tag{11}$$

where the orthogonal matrix $\mathbf{H}_{d,t} \in \mathbb{R}^{2M \times R_{\mathbf{L}} I J R_{\mathbf{L}}}$ is computed from all the TTm-cores but the dth, and $\mathbf{l}_t^{(d)} \in \mathbb{R}^{R_{\mathbf{L}} I J R_{\mathbf{L}}}$.

To recompute $\mathbf{L}_t$ in its original size in the QR step of the SRKF (see (8)), here called the re-squaring step, we need a special case of a TTm, the tall TTm, as well as a thin SVD in TTm format.

Definition 2 (*Tall TTm (Batselier et al., 2017a)*)**.** A tall TTm, as depicted in Fig. 1(c), has only one TTm-core with both a row and column index, while all other TTm-cores have only row indices. Then, the TTm represents a tall matrix with many more rows than columns.

Definition 3 (*Thin SVD in TTm format Batselier (2022)*)**.** Consider a TTm in site-d-mixed canonical format, where the dth TTm-core is the one that has the column index, $\mathcal{L}^{(d)} \in \mathbb{R}^{R_{\mathbf{L}} \times I \times J \times R_{\mathbf{L}}}$. The SVD of $\mathcal{L}^{(d)}$ reshaped and permuted in to a matrix of size $R_{\mathbf{L}} I R_{\mathbf{L}} \times J$, is given by

$$\mathbf{U}^{(d)}\mathbf{S}^{(d)}(\mathbf{V}^{(d)})^\top. \tag{12}$$

Now replace the dth TTm-core by $\mathbf{U}^{(d)}$ reshaped and permuted back to the original TTm-core dimensions.

Then the thin SVD is given by the TTm with the replaced TT-core as the orthogonal $\mathbf{U}$-factor, and $\mathbf{S}^{(d)}(\mathbf{V}^{(d)})^\top$ as the $\mathbf{SV}^\top$-factors, as depicted in Fig. 1(d).

4. TNSRKF

We propose our method, combining efficient TN methods with the SRKF formulation for online GP regression. More specifically, we recursively compute the posterior distribution of the parametric weights in (1) from the measurement update of the Kalman filter. To achieve this, we update the mean $\hat{\mathbf{w}}_t \in \mathbb{R}^M$ as a TT (Section 4.1), and the square root factor $\mathbf{L}_t \in \mathbb{R}^{M \times M}$ as a TTm (Section 4.2).

All computations are summarized in Algorithm 1, which outputs the posterior weight distributions $p(\mathbf{w}_t \mid \mathbf{x}_{1:t}, \mathbf{y}_{1:t}) = \mathcal{N}(\hat{\mathbf{w}}_t, \mathbf{P}_t)$, and the prediction for a test input $f_{*,t}$ in terms of a distribution $p(f_{*,t}) = \mathcal{N}(m_{*,t}, \sigma_{*,t}^2)$ with predictive mean $m_{*,t}$ and variance $\sigma_{*,t}^2$. Note that online GP regression refers to ingesting one measurement at a time and updating the weights $\mathbf{w}_t$ recursively. Therefore, in a truly online scenario, where measurements are collected on the fly, the input to Algorithm 1 would not be a batch $\mathbf{y}$, but a single measurement $\mathbf{y}_t$.

4.1. Update of weight mean

The mean of the weights is updated with a new measurement $y_t \in \mathbb{R}$, with (5). In the original tensor-based KF (Batselier et al., 2017b), the two terms in Eq. (5) are summed together in TT format, which increases the TT-ranks. To avoid this rank increase and application of TT-rounding, we propose solving an optimization problem to compute (5) instead: We apply a commonly-used optimization algorithm from the tensor community, called the alternating linear scheme (ALS) (Holtz, Rohwedder, & Schneider, 2012; Rohwedder & Uschmajew, 2013). The ALS computes a TT by updating one TT-core at a time while keeping all other TT-cores fixed. The optimization problem to be solved is given by

$$\begin{aligned} \min_{\mathbf{w}_t} & \|\hat{\mathbf{w}}_{t-1} + \mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1}) - \mathbf{w}_t\|^2 \\ \text{s.t.} \quad & \mathbf{w}_t \text{ being a low-rank TT,} \end{aligned} \tag{13}$$

where $\hat{\mathbf{w}}_{t-1}$ is the estimate from the last time step, playing now the role of the prior for the current time step.

Inserting (10) in (13), thus making use of $\mathbf{G}_{d,t}$ being an orthogonal matrix (see the site-d-mixed canonical format from Definition 1), gives the optimization problem for the update of one TT-core

$$\min_{\mathbf{w}_t^{(d)}} \left\| \mathbf{G}_{d,t}^\top \left(\hat{\mathbf{w}}_{t-1} + \mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1}) \right) - \mathbf{w}_t^{(d)} \right\|^2. \tag{14}$$

In one so-called sweep of the ALS, (14) is solved for each TT-core once. A stopping criterion for the convergence of the residual in (14) determines the total number of sweeps.

4.2. Update of square root covariance factor

To compute the covariance matrix with the standard covariance update in the measurement update, see (6), we recursively compute the square root covariance factor $\mathbf{L}_t$ as defined in (7) such that $\mathbf{P}_t = \mathbf{L}_t\mathbf{L}_t^\top$. To achieve this, we use the ALS to solve (7) (ALS step) and then we transform $\mathbf{L}_t$ as in (8) back to its original size (re-squaring step).

ALS step. In this step, we use the ALS to compute a TTm representing $\mathbf{L}_t$. We solve the optimization problem given by

$$\begin{aligned} &\min_{\mathbf{L}_t} \left\| \begin{bmatrix} (\mathbf{I}_M - \mathbf{K}_t\boldsymbol{\phi}_t^\top)\mathbf{L}_{t-1} & \sigma_y\mathbf{K}_t \end{bmatrix} - \mathbf{L}_t \right|_{\mathrm{F}}^2 \\ &\text{s.t. } \mathbf{L}_t \text{ being a low-rank TTm,} \end{aligned} \tag{15}$$

where $\mathbf{L}_{t-1}$ is the estimated square root covariance factor from time step $t-1$ now serving as the prior. The original ALS algorithm is defined for TTs, so we must adapt it for TT matrices.

For this, it is necessary to use the site-d-mixed canonical form for TT matrices, as described in Section 3.1.2 above (11). In addition, we need to horizontally concatenate two matrices in TTm format, which can be done by summing two matrices of size $M \times 2M$ such that (15) becomes

$$\begin{aligned} &\min_{\mathbf{I}_t^{(d)}} \left| \mathbf{H}_{d,t}^\top \operatorname{vec}\left(\begin{bmatrix} 1 & 0 \end{bmatrix} \otimes (\mathbf{I}_M - \mathbf{K}_t\boldsymbol{\phi}_t^\top)\mathbf{L}_{t-1}\right) \right. \\ &\left. + \mathbf{H}_{d,t}^\top \operatorname{vec}\left(\begin{bmatrix} 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & \mathbf{0}_{M-1} \end{bmatrix} \otimes \sigma_y\mathbf{K}_t\right) - \mathbf{I}_t^{(d)} \right|_{\mathrm{F}}^2, \end{aligned} \tag{16}$$

where vec denotes the vectorization of the involved TT matrices.

Re-squaring step. The optimization problem given by (15) requires concatenating a matrix with a column vector. In TT format, this results in a TTm of size $M \times 2M$. For the TTm-cores of $\mathbf{L}_t$ this means that one TTm-core, which we call the augmented core, is of size $R_{\mathbf{L}} \times I \times 2J \times R_{\mathbf{L}}$. Before serving as a prior for the next time step, a re-squaring step implementing the QR step (see (8)) in TN format is required to transform $\mathbf{L}_t$ back to its original size. Since computing a QR decomposition of a TTm is not directly possible, we present an SVD-based algorithm in TN format to transform $\mathbf{L}_t$ of size $M \times 2M$ back to $M \times M$, as described in Algorithm 2.

4.3. Predictions

To perform GP predictions we compute the predictive distribution for a test output $f_{*,t} = \boldsymbol{\phi}(\mathbf{x}_*)^\top\mathbf{w}_t$ with mean and variance given by

$$\begin{aligned} m_{*,t} &= \boldsymbol{\phi}(\mathbf{x}_*)^\top\hat{\mathbf{w}}_t \\ \sigma_{*,t}^2 &= \boldsymbol{\phi}(\mathbf{x}_*)^\top\mathbf{L}_t\mathbf{L}_t^\top\boldsymbol{\phi}(\mathbf{x}_*). \end{aligned} \tag{17}$$

Given $\hat{\mathbf{w}}_t$ as a TT and $\mathbf{L}_t$ as a TTm, we can compute (17) directly in TN format without explicitly reconstructing the mean vector and square root factor. For a test input $\mathbf{x}_*$, Fig. 2 illustrates the computation of (a) the predictive mean $m_{*,t}$, (b) the predictive covariance $\sigma_{*,t}^2$. The corresponding equation to Fig. 2(a) is given by

$$m_{*,t} = \sum_{r^{(2)}}^{R^{(2)}} \cdots \sum_{r^{(D)}}^{R^{(D)}} \prod_{d}^{D} \sum_{i^{(d)}}^{I^{(d)}} \boldsymbol{\phi}_{i^{(d)}}^{(d)}(\mathbf{x}_*)\hat{\mathcal{W}}_{r^{(d)},i^{(d)},r^{(d+1)}}^{(d)},$$

where the lowercase letters in the subscript, i.e. $r^{(d)}$, $r^{(d+1)}$ and $i^{(d)}$, denote the indices of size $R^{(d)} = R^{(d+1)} = R$ and $I^{(d)} = I$,

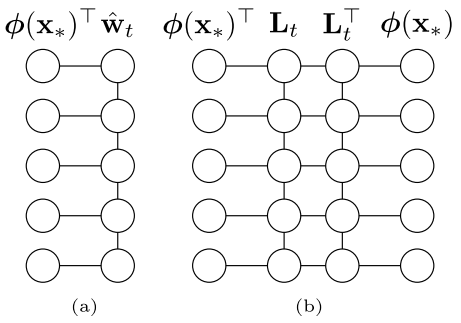

Fig. 2. Visual depiction of (a) predictive mean and (b) predictive covariance for $D = 5$.

Table 1
Computational complexities for one TT-core mean and covariance update. We denote the TT-ranks of $\mathbf{K}_t$ by $R_{\mathbf{K}}$.

Term	Complexity
$\mathbf{G}_{d,t}^\top\hat{\mathbf{w}}_{t-1}$	$\mathcal{O}(R_{\mathbf{w}}^4 I)$
$\mathbf{G}_{d,t}^\top\mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top\hat{\mathbf{w}}_{t-1})$	$\mathcal{O}(R_{\mathbf{w}}^2 R_{\mathbf{K}}^2 I)$
(20)–(22)	$\mathcal{O}(R_{\mathbf{L}}^4 IJ)$

respectively. Fig. 2(b) can be written in a similar way. Moving forward, we provide only the TN diagrams, since the equations can become lengthy.

Algorithm 1 Online GP regression in terms of SRKF in TT format (TNSRKF)

Input: Measurements $\mathbf{y} = y_1, y_2, \ldots, y_N$,
basis functions for inputs $\boldsymbol{\phi}(\mathbf{x}_t)$, $t = 1, \ldots, N$,
prior $\hat{\mathbf{w}}_0$ in TN format (Lemma 5)
prior $\mathbf{L}_0$ in TN format (Lemma 7),
noise variance σ_y^2,
basis functions for prediction point $\boldsymbol{\phi}(\mathbf{x}_*)$.
Output: $p(\mathbf{w} \mid \mathbf{y}_{1:t}) = \mathcal{N}(\hat{\mathbf{w}}_t, \mathbf{P}_t)$ and
$p(f_* \mid \mathbf{y}_{1:t}) = \mathcal{N}(m_{*,t}, \sigma_{*,t}^2)$, for $t = 1, \ldots, N$.
1: Initialize $\mathbf{w}_1 = \hat{\mathbf{w}}_0$ and $\mathbf{L}_1$ as a random TTm in site-d-mixed canonical format.
2: **for** $t = 1, \ldots, N$ **do**
3: Compute $\hat{\mathcal{W}}_t^{(1)}, \hat{\mathcal{W}}_t^{(2)}, \ldots, \hat{\mathcal{W}}_t^{(D)}$ with Eq. (14).
4: Compute $\mathcal{L}_t^{(1)}, \mathcal{L}_t^{(2)}, \ldots, \mathcal{L}_t^{(D)}$ with Eq. (16).
5: Save TT and TTm from step 3 and 4 as initializations for the next time step.
6: Re-square $\mathbf{L}_t$ with Algorithm 2.
7: Compute $m_{*,t}$ with Eq. (17) as depicted in Fig. 2(a).
8: Compute $\sigma_{*,t}^2$ with Eq. (17) as depicted in Fig. 2(b).
9: **end for**

5. Implementation

In this section, we give a detailed description of the non-straightforward TN operations to update the mean estimate $\hat{\mathbf{w}}_t$ and square root covariance factor $\mathbf{L}_t$ as described in Algorithm 1. The leading complexities of the mean and square root covariance factor update are given in Table 1.

5.1. Updating $\hat{\mathbf{w}}_t$ in TN format

In the following sections, we discuss the implementation of (14) for the mean update (Algorithm 1, line 3), and we describe how the mean is initialized in TT format (Algorithm 1, line 1).

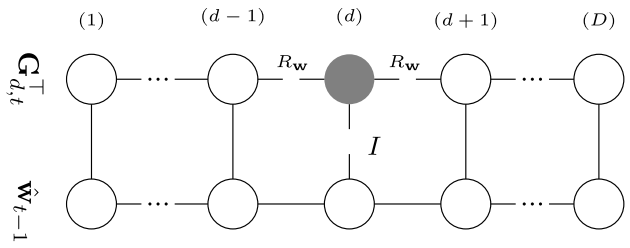

Fig. 3. Visual depiction of computation of $\mathbf{G}_{d,t}^\top \hat{\mathbf{w}}_{t-1}$, resulting in three-way tensor of size $R_\mathbf{W} \times I \times R_\mathbf{W}$ (gray node). The indices are summed over from left to right, alternating between the vertical and horizontal ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.1.1. Implementation of $\mathbf{G}_{d,t}^\top(\hat{\mathbf{w}}_{t-1} + \mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1}))$

To compute the TT representing the mean estimate $\hat{\mathbf{w}}_t$, we implement the ALS to solve (14) (Algorithm 1, line 3).

The following example illustrates the update of one TT-core during the ALS.

Example 4 (*TT-Core Update with ALS*)**.** Take a $D = 5$ dimensional weight vector in TT format with $I = 10$ basis functions in each dimension, resulting in 10^5 parameters and uniform TT-ranks of $R_2 = R_3 = R_4 = 4$. Say, we are currently updating the third TT-core $\mathcal{W}_t^{(3)} \in \mathbb{R}^{4\times 10\times 4}$ using

$$\underbrace{\mathbf{w}_t^{(3)}}_{160\times 1} = \underbrace{\mathbf{G}_{3,t}^\top}_{160\times 10^5} \left(\underbrace{\hat{\mathbf{w}}_{t-1}}_{10^5\times 1} + \underbrace{\mathbf{K}_t}_{10^5\times 1} \underbrace{(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1})}_{1\times 1} \right). \tag{18}$$

We first multiply over the large dimension of 10^5 in $\mathbf{G}_{3,t}^\top \hat{\mathbf{w}}_{t-1}$ and $\mathbf{G}_{3,t}^\top \mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1})$. In TT format, this matrix–vector multiplication is done core by core, thus avoiding the explicit multiplication. Finally, we sum two vectors of size 160.

Fig. 3 illustrates the multiplication of $\mathbf{G}_{d,t}^\top \hat{\mathbf{w}}_{t-1}$ in TT format, resulting in a tensor $\mathcal{W}_t^{(d)}$ of size $R_\mathbf{W} \times I \times R_\mathbf{W}$.

The multiplication of between $\mathbf{G}_{d,t}^\top$ and $\mathbf{K}_t(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1})$ works in the same way as depicted in Fig. 3, after firstly computing $\boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1}$ in TN format and secondly multiplying one arbitrary TT-core of $\mathbf{K}_t$ by the scalar $(y_t - \boldsymbol{\phi}_t^\top \hat{\mathbf{w}}_{t-1})$.

During the update of the dth TT-core, the TT is in site-d-mixed canonical format. Before updating the next TT-core, either the $(d-1)$th or the $(d+1)$th, the site-$(d-1)$-mixed or site-$(d+1)$-mixed canonical format is computed. Note that because of the recursive property, updating every TT-core once with a new measurement is usually sufficient for the residual of (13) to converge.

5.1.2. Initialization of $\hat{\mathbf{w}}_0$ and $\mathbf{w}_1$

For the first time step $t = 1$ of Algorithm 1, we choose a zero-mean assumption for the prior estimate $\hat{\mathbf{w}}_0$. The following Lemma explains how this can be implemented in TT format.

Lemma 5 (*Zero-Mean Prior in TT Format Batselier et al. (2019)*)**.** *Consider a vector with all entries equal to zero. In TT format, such a vector is given by a TT in site-d-mixed canonical format, where the dth TT-core contains only zeros.*

In addition, Algorithm 1 requires an initial guess for $\mathbf{w}_1$ to compute $\mathbf{G}_{d,1}$ from all TT-cores of $\mathbf{w}_1$, except the dth. For this, we set $\mathbf{w}_1 = \hat{\mathbf{w}}_0$.

5.2. Updating $\mathbf{L}_t$ in TT format

To compute the TTm representing $\mathbf{L}_t$, we implement the ALS to solve (16) (Algorithm 1, line 4). The following example illustrates the update of one TTm-core during the ALS.

Example 6 (*TTm-Core Update With ALS*)**.** Take a $D = 5$ dimensional TTm representing $\mathbf{L}_t \in \mathbb{R}^{M\times M}$, where we are currently updating the third TTm-core. We have $I = 10$ and $J = 10$, where the third TTm-core is augmented, and $R_\mathbf{L} = 4$. We update $\mathcal{L}_t^{(3)} \in \mathbb{R}^{4\times 10\times 20\times 4}$ using

$$\begin{aligned} \underbrace{\mathbf{l}_t^{(3)}}_{3200\times 1} = & \underbrace{\mathbf{H}_{d,t}^\top}_{3200\times 2\cdot 10^{10}} \operatorname{vec}\left(\underbrace{\begin{bmatrix}1 & 0\end{bmatrix}}_{1\times 2} \otimes \underbrace{\mathbf{L}_{t-1}}_{10^5\times 10^5} \right) \\ & - \underbrace{\mathbf{H}_{d,t}^\top}_{3200\times 2\cdot 10^{10}} \operatorname{vec}\left(\underbrace{\begin{bmatrix}1 & 0\end{bmatrix}}_{1\times 2} \otimes \underbrace{\mathbf{K}_t \boldsymbol{\phi}_t^\top \mathbf{L}_{t-1}}_{10^5\times 10^5} \right) \\ & + \underbrace{\mathbf{H}_{d,t}^\top}_{3200\times 2\cdot 10^{10}} \operatorname{vec}\left(\underbrace{\begin{bmatrix}0 & 1\end{bmatrix}}_{1\times 2} \otimes \underbrace{\begin{bmatrix}1 & \mathbf{0}_{M-1}\end{bmatrix}}_{1\times 10^5} \otimes \underbrace{\sigma_y \mathbf{K}_t}_{10^5\times 1} \right). \end{aligned} \tag{19}$$

We first multiply over the large dimension of $2\cdot 10^{10}$ in TT format, then sum the three terms of size 3200×1.

From Example 6, it follows that the three terms of (19) need to be implemented. We discuss them separately in the following sections. We distinguish between the update of the augmented TTm-core from all other ones, which result in TTm-cores of size $R_\mathbf{L} \times I \times 2J \times R_\mathbf{L}$ and $R_\mathbf{L} \times I \times J \times R_\mathbf{L}$, respectively. In the tensor diagrams (Figs. 4–6), we depict the update for the augmented TTm-core.

Before diving in, recall from (11) that $\mathbf{H}_{d,t}$ is computed from TTm-cores of $\mathbf{L}_t$, except the dth, where row and column indices are combined. In the tensor diagrams, the indices are depicted not as combined because, in practice, they are generally summed over separately. However, the vectorized format is necessary for writing down the equations in matrix form.

5.2.1. Implementation of first term of (16)

Fig. 4 illustrates the computation of the augmented TTm-core in the first term of (16), given by

$$\mathbf{H}_{d,t}^\top \operatorname{vec}\left(\begin{bmatrix}1 & 0\end{bmatrix} \otimes \mathbf{L}_{t-1}\right). \tag{20}$$

The column indices of $\mathbf{L}_{t-1}$ are indicated by the round edges that are connected to the row indices of $\mathbf{H}_{d,t}^\top$. The edge containing $\mathbf{e}_1 = [1 \quad 0]$ is connected to the dth TTm core of $\mathbf{L}_t$ with a rank-1 connection, which corresponds to the Kronecker product in (20). The summation over the vertical and curved indices has the leading computational complexity of $\mathcal{O}(R_\mathbf{L}^4 IJ)$ per dimension. When updating all TTm-cores except the augmented TTm-core, the additional index of size 2 is summed over resulting in a tensor of size $R_\mathbf{L} \times I \times J \times R_\mathbf{L}$.

5.2.2. Implementation of second term of (16)

Fig. 5 illustrates the computation of

$$\mathbf{H}_{d,t}^\top \operatorname{vec}\left(\begin{bmatrix}1 & 0\end{bmatrix} \otimes \mathbf{L}_{t-1}\mathbf{L}_{t-1}^\top \boldsymbol{\phi}_t \mathbf{S}_t^{-1} \boldsymbol{\phi}_t^\top \mathbf{L}_{t-1}\right), \tag{21}$$

which directly follows from the second term of (16). As shown, the row and column indices of $\mathbf{H}_{d,t}^\top$ are connected separately to the column and row indices of two TT matrices for $\mathbf{L}_{t-1}$, respectively. Like in the previous term, the edge containing $\mathbf{e}_1 = [1 \;\; 0]$ is connected to the augmented TTm-core of $\mathbf{L}_t$ with a rank-1

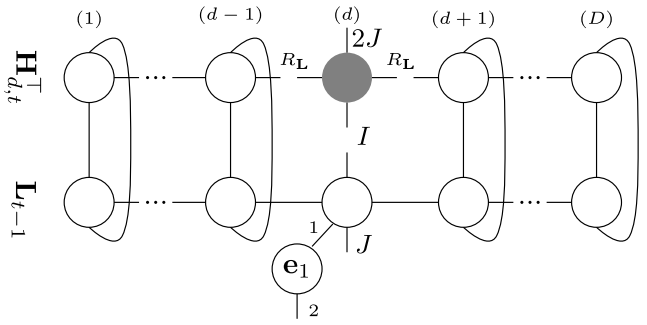

Fig. 4. Visual depiction for computing the augmented TTm-core in (20) resulting in a 4-way tensor of size $R_{\mathbf{L}} \times I \times 2J \times R_{\mathbf{L}}$ (gray node). The combined horizontal and curved indices are summed over and alternating with the horizontal indices.

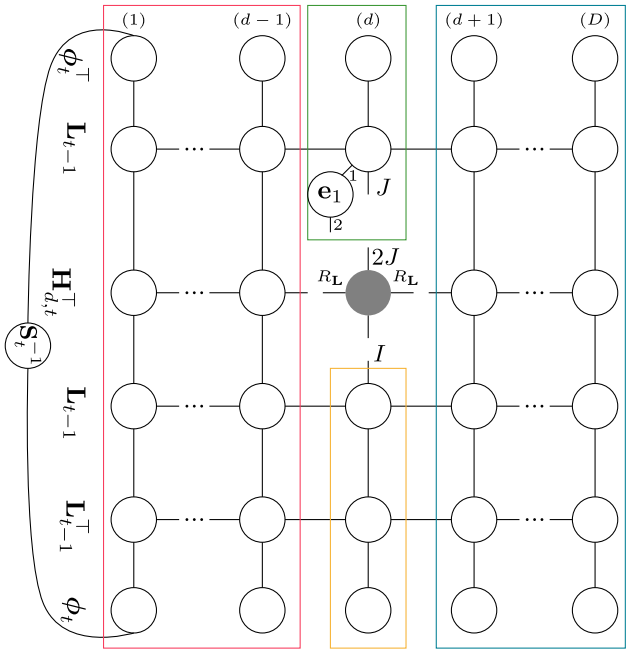

Fig. 5. Visual depiction for computing (21) resulting in a 4-way tensor of size $R_{\mathbf{L}} \times I \times 2J \times R_{\mathbf{L}}$ (gray node). First, the indices in the red and blue boxes are summed over, then the indices between the red, yellow, and blue boxes, and finally, the ones between the red, green, and blue boxes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

connection, which corresponds to the Kronecker product in (21). The leading computational complexity of $\mathcal{O}(R_{\mathbf{L}}^4 IJ)$ per dimension comes from the summation over the vertical indices in the red or blue box indicated in the figure. The most efficient order of doing the computations in Fig. 5 was found with the visual tensor network software by Evenbly (2019), assuming our use case where $D > 3$ and $I, J, R_L < 10$.

5.2.3. Implementation of third term of (16)

Fig. 6 illustrates the computation of

$$\mathbf{H}_{d,t}^{\top} \operatorname{vec}\left(\begin{bmatrix}0 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & \mathbf{0}_{M-1}\end{bmatrix} \otimes \sigma_y \mathbf{L}_t \mathbf{L}_t^{\top} \boldsymbol{\phi}_t \mathbf{S}_t^{-1}\right), \tag{22}$$

which directly follows from the third term of (16). The row of nodes each filled with $\mathbf{e}_1 = [1 \quad \mathbf{0}_{J-1}]$ corresponds to $[1 \quad \mathbf{0}_{M-1}]$ from (22) and their rank-1 connections to the nodes above is the second Kronecker product in (22), which is done dimension-wise in TT format. The node with $\mathbf{e}_2$ corresponds to $[0 \quad 1]$ from (22) and its rank-1 connection is the first Kronecker product

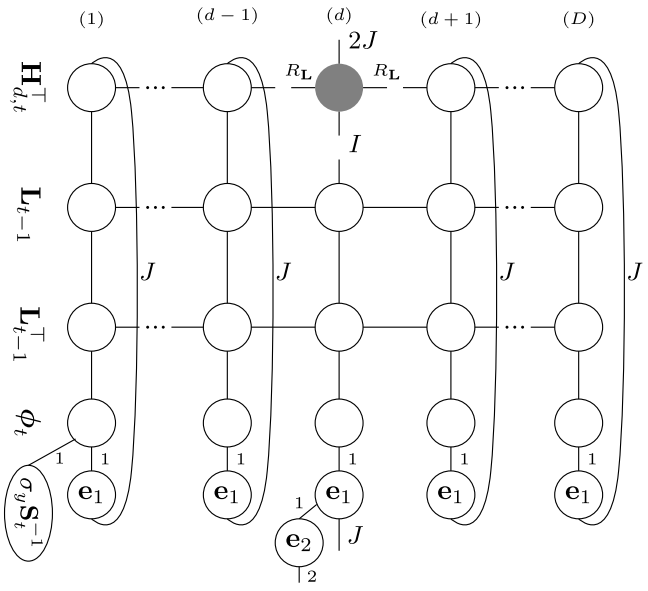

Fig. 6. Visual depiction for computing (22), resulting in a 4-way tensor of size $R_{\mathbf{L}} \times I \times 2J \times R_{\mathbf{L}}$ (gray node). The indices are summed over from left to right by alternating between the vertical and horizontal ones. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

in (22). The summation over the vertical indices is the leading computational complexity of $\mathcal{O}(R_{\mathbf{L}}^4 IJ)$ per dimension.

5.2.4. SVD-based re-squaring step in TTm format

When computing (16), we double the number of columns of $\mathbf{L}_t$ compared to $\mathbf{L}_{t-1}$. For the next time step, however, we need to transform $\mathbf{L}_t$ back to its original size (Algorithm 1, line 6), otherwise its column size will grow with the iterations and slow down the algorithm. The QR step, as in (8), computes a full QR decomposition of $\mathbf{L}_t$, which cannot be done in TT format. Instead, we compute a thin SVD in TTm format (Definition 3) of $\mathbf{L}_t$ transformed into a tall TTm (here also denoted by $\mathbf{L}_t$) with all row indices of size IJ, except the dth which is of size I, and the dth column index of size $2J$. The J-truncated SVD of $\mathbf{L}_t$ is then given by

$$\underbrace{\mathbf{L}_t}_{MJ^{D-1}\times 2J} \approx \underbrace{\mathbf{U}_t \mathbf{S}_t}_{MJ^{D-1}\times J} \underbrace{\mathbf{V}_t^{\top}}_{J\times 2J}, \tag{23}$$

where $\mathbf{U}_t \mathbf{S}_t$ is the new $\mathbf{L}_t$ and $\mathbf{V}_t^{\top}$ can be discarded because of (9). In practice, we compute an SVD of the augmented TTm-core and truncate it back to the size of $R_{\mathbf{L}} \times I \times J \times R_{\mathbf{L}}$.

There is a way to make (23) exact. This is possible if the augmented TTm-core is of size $R_{\mathbf{L}} \times I \times 2JR_{\mathbf{L}}^2 \times R_{\mathbf{L}}$. In this case, the SVD computed of the augmented TTm-core results in a square $\mathbf{U}$-factor. Since the number of columns is doubled every measurement update, the re-squaring step can be skipped p times until $2^p = 2R_{\mathbf{L}}^2$. Choosing smaller values for p reduces computational complexity at the cost of accuracy.

The SVD-based re-squaring step is described in Algorithm 2. The SVD of the reshaped and permuted augmented TTm-core is truncated for $2^p < 2R_{\mathbf{L}}^2$ and exact for $2^p \geq 2R_{\mathbf{L}}^2$.

5.2.5. Initialization of $\mathbf{L}_0$ and $\mathbf{L}_1$

At time $t = 1$, Algorithm 1 requires the prior square root covariance factor $\mathbf{L}_0$ in TTm format. We are considering product kernels that have priors in Kronecker format. The following Lemma describes how these types of priors can be transformed into a TTm for $\mathbf{L}_0$.

Algorithm 2 SVD-based re-squaring step of covariance update

Input: TTm $\mathbf{L}_t$ in site-d-mixed canonical format with $\mathcal{L}^{(d)} \in \mathbb{R}^{R_\mathbf{L} \times I \times 2^p J \times R_\mathbf{L}}$.

Output: TTm $\mathbf{L}_t$ with $\mathcal{L}^{(d)} \in \mathbb{R}^{R_\mathbf{L} \times I \times 2^{p-1} J \times R_\mathbf{L}}$.

1: $\mathbf{L}^{(d)} \leftarrow$ Reshape / permute $\mathcal{L}^{(d)}$ into matrix of size $R_\mathbf{L} I R_\mathbf{L} \times 2^{p+1} J$.
2: Compute thin SVD$(\mathbf{L}^{(d)}) = \mathbf{U}^{(d)} \mathbf{S}^{(d)} (\mathbf{V}^{(d)})^\top$.
3: $\mathcal{L}^{(d)} \leftarrow$ Reshape / permute first $2^{p-1} J$ columns of $\mathbf{U}^{(d)} \mathbf{S}^{(d)}$ of size $R_\mathbf{L} I R_\mathbf{L} \times 2^{p-1} J$ into tensor of size $R_\mathbf{L} \times I \times 2^{p-1} J \times R_\mathbf{L}$.

Lemma 7 (*Prior covariance with Kronecker structure into TTm, follows from Golub & Van Loan, 2013, p.708*)**.** *Given a prior covariance* $\mathbf{P}_0 = \mathbf{P}_0^{(1)} \otimes \mathbf{P}_0^{(2)} \otimes \cdots \otimes \mathbf{P}_0^{(D)}$*, the prior square root covariance in TTm format is given by a TTm with all ranks equal to 1, where the cores are given by* $\mathbf{L}_0^{(1)}, \mathbf{L}_0^{(2)}, \ldots, \mathbf{L}_0^{(D)}$*, each reshaped into a 4-way tensor of size* $1 \times I \times J \times 1$.

In addition, Algorithm 1 requires an initial guess in TTm format for $\mathbf{L}_1 \in \mathbb{R}^{M \times 2M}$. We cannot set $\mathbf{L}_1 = \mathbf{L}_0$ since the prior has TTm-ranks equal to one, and we may want higher TTm-ranks for $\mathbf{L}_t$. This is because the choice of the TTm-ranks of $\mathbf{L}_1$ determines the rank manifold on which the TTm-cores will be optimized. We initialize the TTm-cores as random samples from a zero-mean Gaussian distribution and transform the TTm into site-d-mixed canonical format, where d is the augmented TTm-core.

6. Experiments

In this section, we show how our method works in practice by performing online GP regression on synthetic and real-life data sets. We evaluate our predictions based on the root mean square error (RMSE) for the accuracy of the mean and negative log-likelihood (NLL) for the uncertainty estimation. The metrics after t measurement updates are defined as

$$(\text{RMSE})_t = \sqrt{\sum_{i=1}^{N_*} \frac{(m_{*,t,i} - y_{*,i})^2}{N_*}} \quad \text{and}$$
$$(\text{NLL})_t = 0.5 \sum_{i=1}^{N_*} \log(2\pi\sigma_{*,t,i}^2) + \frac{(m_{*,t,i} - y_{*,i})^2}{\sigma_{*,t,i}^2}, \tag{24}$$

where $y_{*,i}$ is the ith measurement from the test set, $m_{*,t,i}$ and $\sigma_{*,t,i}$ are the predictive mean and variance for the ith test point, and N_* is the number of test points.

First, we show the equivalence of the full-rank TNSRKF and the conventional Kalman filter. Then we show in a synthetic experiment how the choice of $R_\mathbf{w}$ and $R_\mathbf{L}$ impacts the accuracy of the approximation. Finally, we compare our method to the TNKF on a benchmark data set for nonlinear system identification.

All experiments were performed on an 11th Gen Intel(R) Core(TM) i7 processor running at 3.00 GHz with 16 GB RAM. For reproducibility of the method and the experiments, the code written in Julia programming language is freely available at https://github.com/clarazen/TNSRKF.

6.1. Equivalence of full-rank TNSRKF and Kalman filter

In the first experiment, we show in which case our method is equivalent to the measurement update of the conventional Kalman filter. We generate $D = 3$ dimensional synthetic data sampled from a reduced-rank GP by Solin and Särkkä (2020) with a squared exponential kernel (lengthscale $\ell^2 = 0.1$ and signal variance $\sigma_f^2 = 1$), set the noise variance to $\sigma_y^2 = 0.01$ and use $I = 4$ basis functions per dimension, such that $\mathbf{P}_t \in \mathbb{R}^{64 \times 64}$. The input data lies in a cuboid given by $[-1\ \ 1] \times [-1\ \ 1] \times [-1\ \ 1]$ and $N, N_* = 100$.

Table 2
RMSE and NLL at time $t = N$ for the full-rank setting and different choices of p in comparison to the conventional Kalman filter (KF).

Method	Setting			$(\text{RMSE})_N$	$(\text{NLL})_N$
KF		–		0.07873	−106.864
TNSRKF	$R_\mathbf{w}$	$R_\mathbf{L}$	p		
	4	16	8	0.07873	−106.864
	4	16	4	0.07879	−108.338
	4	16	2	0.07444	−157.716
	4	16	1	0.06765	−166.178

Table 2 shows the RMSE and NLL for test data at time $t = N$ for different choices of p. The TNSRKF is equivalent to the Kalman filter when both $R_\mathbf{w}$ and $R_\mathbf{L}$ are full-rank. In addition, p must be chosen, such that the QR step, discussed in Section 5.2.4, is exact. For settings with lower values for p, the method trade-in accuracy.

In the following sections, we look at scenarios where the Kalman filter can no longer be computed on a conventional laptop because both storage and computational time become unfeasible.

6.2. Influence of the ranks on the approximation

The choice of the TT- and TTm-ranks is not obvious and can be intricate. However, the computational budget often determines how high the ranks can be chosen. In this experiment, we use our method to make online GP predictions on synthetic data while varying the TTm-ranks of $\mathbf{L}_t$, as well as the TT-ranks of $\mathbf{w}_t$.

We consider the Volterra kernel, a popular choice for nonlinear system identification. It is known that the truncated Volterra series suffers from the curse of dimensionality, which was lifted in a TN setting by Batselier et al. (2017a). With the notation of this paper, the basis functions $\boldsymbol{\phi}$ of parametric model (1) are a combination of monomials computed from the input sequence of the given problem. We generate synthetic training and testing data as described in Batselier (2021), where $D = 7$ and $I = 4$ such that the number of parameters is $4^7 = 16\,384$. We set the SNR to 60, corresponding to $\sigma_y^2 = 6.96 \times 10^{-6}$.

Fig. 7 shows the RMSE and NLL on the testing data for $R_\mathbf{w} = 2, 4$ and $R_\mathbf{L} = 2, 4$ over time iterations of the TNSRKF. At $t = N$, the RMSE is lower for $R_\mathbf{w} = 2$ and $R_\mathbf{L} = 4$ than for $R_\mathbf{w} = 4$ and $R_\mathbf{L} = 4$. Thus, it seems that a lower value for the mean estimate represents the data better. Note that although having larger TT-ranks increases the degrees of freedom of the TT, it may not always improve the accuracy of the approximation, e.g. because higher TT-ranks can result in overfitting, while lower ranks can have a regularizing effect. The NLL is the lowest for $R_\mathbf{w} = 4$ and $R_\mathbf{L} = 4$, which is close to the NLL for $R_\mathbf{w} = 4$ and $R_\mathbf{L} = 2$. Note that the NLL for the same $R_\mathbf{L}$ is different for the two settings of $R_\mathbf{w}$, because the NLL also depends on the difference between predicted and actual measurements, thus on the accuracy of $\hat{\mathbf{w}}_t$.

This experiment showed that the choice of $R_\mathbf{w}$ and $R_\mathbf{L}$ influences the performance of the TNSRKF. Since higher values for the ranks also increase the computational complexity, the computational budget will determine the higher limit for the ranks. In addition, an assumption with lower ranks may be fitting the data better in some cases.

6.3. Comparison to TNKF for cascaded tanks benchmark data set

In this experiment, we compare our method to the TNKF on a nonlinear benchmark for system identification, the cascaded tanks data set. A detailed description can be found in Schoukens

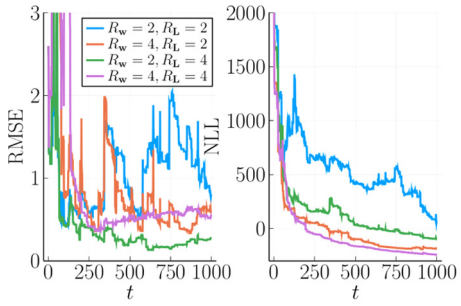

Fig. 7. RMSE and NLL over time iterations for different combinations of R_w and R_L.

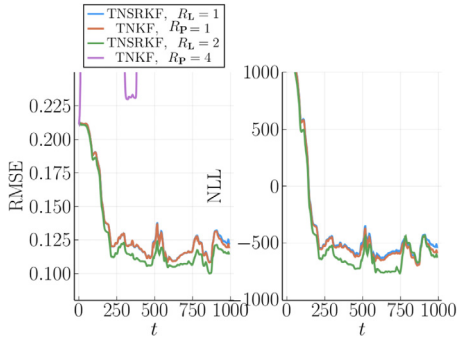

Fig. 8. RMSE and NLL over iterations for TNKF and TNSRKF for $R_L = R_P = 1$ and $R_L = 2$, $R_P = R_L \cdot R_L = 4$. The orange and blue lines mostly overlap because both methods perform similarly for $R_L = R_P = 1$. Also, the violet curve leaves the plot window because the TNKF diverges. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

and Noël (2017). The training and testing data both consist of a data set of 1024 data points each. To train our GP model, we choose lagged inputs and outputs as input to our GP, as described in Karagoz and Batselier (2020), resulting in an input of dimensionality $D = 14$. We use a squared exponential kernel, which hyperparameters we optimize with the Gaussian process toolbox by Rasmussen and Williams (2006), and we choose $I = 4$, such that the model has $M = 4^{14} = 268\,435\,456$ parameters.

For the comparison to the TNKF, we choose the TT-ranks for the mean to be $R_2 = R_{14} = 4$, $R_3 = \cdots R_{13} = 10$, and we vary R_L and the TTm-ranks of the covariance matrix for the TNKF denoted by R_P. Figs. 8 and 9 show the RMSE and NLL over the time iterations of the respective filter. When $R_L = 1$ and $R_P = 1$, both methods perform almost the same, as visualized by the overlapping orange and blue lines. When $R_L^2 = R_P = 4$, our method improves both prediction accuracy and uncertainty estimation compared to the $R_L = 1$. On the contrary, the TNKF diverges and leaves the plotted figure area because the covariance matrix loses positive definiteness. When $R_L^2 = R_P = 16$, the TNKF shows a similar behavior, while the TNSRKF results in lower RMSEs but mostly higher NLL values. This setting shows that higher values for R_L are not always beneficial for the uncertainty estimation.

Finally, Fig. 10 shows the predictions with the TNSRKF on testing data after seeing 100, 200, and 922 data points. Aligned with the plot showing the RMSE and NLL, after 100 data points, the prediction is quite bad and uncertain. After 200 data points, the predictions are better and more certain and further improve after seeing the entire data set.

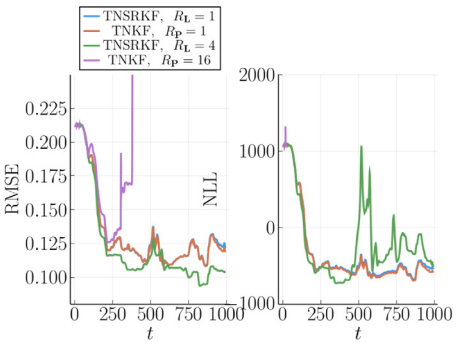

Fig. 9. RMSE and NLL over iterations for TNKF and TNSRKF for $R_L = R_P = 1$ and $R_L = 4$, $R_P = R_L \cdot R_L = 16$. The orange and blue lines mostly overlap because both methods perform similarly for $R_L = R_P = 1$. Also, the violet curve leaves the plot window because the TNKF diverges. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

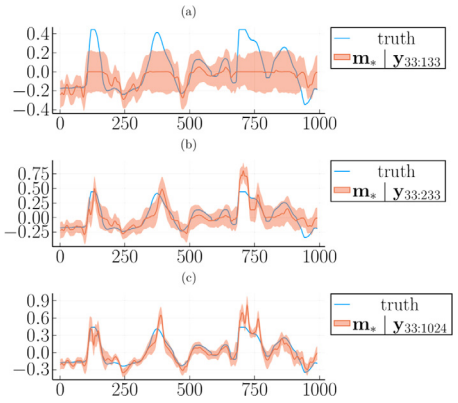

Fig. 10. Predictions on test data with uncertainty bounds after seeing (a) 101, (b) 201, and (3) 992 data points for $R_L = 4$. The measurements start at 33 because the memory goes back 32 time steps.

7. Conclusion

In this paper, we presented a TT-based solution for online GP regression in terms of an SRKF. In our experiments, we show that our method is scalable to a high number of input dimensions at a reasonable computational cost such that all experiments could be run on a conventional laptop. In addition, we improve the state-of-the-art method for TN-based Kalman filter: In settings where the TNKF loses positive (semi-)definiteness and becomes numerically unstable, our method avoids this issue because we compute the square root covariance factors instead of the covariance matrix. In this way, we can choose settings for our method that achieve better accuracy than the TNKF.

A future work direction is online hyperparameter optimization. We are looking at a truly online scenario, so future data is not available. Thus, we cannot swipe over mini-batches of data multiple times like other methods, e.g. Schürch, Azzimonti, Benavoli, and Zaffalon (2020), to optimize hyperparameters.

Finally, there is still ongoing research to determine how to choose TT-ranks and TTm-ranks. In the synthetic experiments, we showed the impact of R_L and R_w. Generally, the TT- and TTm-ranks need to be treated as hyperparameters.

References

Batselier, K. (2021). Enforcing symmetry in tensor network MIMO Volterra identification. *IFAC-PapersOnLine*, *54*(7), 469–474.

Batselier, K. (2022). Low-rank tensor decompositions for nonlinear system identification: A tutorial with examples. *IEEE Control Systems Magazine*, *42*(1), 54–74.

Batselier, K., Chen, Z., & Wong, N. (2017a). Tensor network alternating linear scheme for MIMO Volterra system identification. *Automatica*, *84*, 26–35.

Batselier, K., Chen, Z., & Wong, N. (2017b). A tensor network Kalman filter with an application in recursive MIMO Volterra system identification. *Automatica*, *84*, 17–25.

Batselier, K., Ko, C.-Y., & Wong, N. (2019). Extended Kalman filtering with low-rank tensor networks for MIMO Volterra system identification. In *2019 IEEE 58th conference on decision and control* (pp. 7148–7153).

Berntorp, K. (2021). Online Bayesian inference and learning of Gaussian process state–space models. *Automatica*, *129*, Article 109613.

De Rooij, S. J. S., Batselier, K., & Hunyadi, B. (2023). Enabling large-scale probabilistic seizure detection with a tensor-network Kalman filter for LS-SVM. In *2023 IEEE international conference on acoustics, speech, and signal processing workshops* (pp. 1–5). IEEE.

Doyle, F. J., Pearson, R. K., & Ogunnaike, B. A. (2002). *Identification and control using Volterra models*. Springer.

Evenbly, G. (2019). TensorTrace: An application to contract tensor networks. arXiv preprint arXiv:1911.02558.

Golub, G. H., & Van Loan, C. F. (2013). *Matrix computations*. JHU Press.

Grewal, M. S., & Andrews, A. P. (2014). *Kalman filtering: theory and practice with MATLAB*. John Wiley & Sons.

Hartikainen, J., & Särkkä, S. (2010). Kalman filtering and smoothing solutions to temporal Gaussian process regression models. In *2010 IEEE international workshop on machine learning for signal processing* (pp. 379–384). IEEE.

Holtz, S., Rohwedder, T., & Schneider, R. (2012). The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing*, *34*(2), A683–A713.

Izmailov, P., Novikov, A., & Kropotov, D. (2018). Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition. In *International conference on artificial intelligence and statistics* (pp. 726–735). PMLR.

Karagoz, R., & Batselier, K. (2020). Nonlinear system identification with regularized tensor network B-splines. *Automatica*, *122*, Article 109300.

Kolda, T. G., & Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, *51*(3), 455–500.

Liu, M., Chowdhary, G., Castra da Silva, B., Liu, S.-Y., & How, J. P. (2018). Gaussian processes for learning and control: A tutorial with examples. *IEEE Control Systems Magazine*, *38*(5), 53–86.

Nguyen-Tuong, D., Peters, J., & Seeger, M. (2008). Local Gaussian process regression for real time online model learning. In *Advances in neural information processing systems*: *Vol. 21*.

Nickson, T., Gunter, T., Lloyd, C., Osborne, M. A., & Roberts, S. (2015). Blitzkriging: Kronecker-structured stochastic Gaussian processes. arXiv preprint arXiv:1510.07965.

Oseledets, I. V. (2010). Approximation of 2D x 2D matrices using tensor decomposition. *SIAM Journal on Matrix Analysis and Applications*, *31*(4), 2130–2145.

Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, *33*(5), 2295–2317.

Quinonero-Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, *6*, 1939–1959.

Ranganathan, A., Yang, M.-H., & Ho, J. (2010). Online sparse Gaussian process regression and its applications. *IEEE Transactions on Image Processing*, *20*(2), 391–404.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning*. The MIT Press.

Rohwedder, T., & Uschmajew, A. (2013). On local convergence of alternating schemes for optimization of convex problems in the tensor train format. *SIAM Journal on Numerical Analysis*, *51*(2), 1134–1162.

Särkkä, S., Solin, A., & Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing: A look at Gaussian process regression through Kalman filtering. *IEEE Signal Processing Magazine*, *30*(4), 51–61.

Särkkä, S., & Svensson, L. (2023). *Bayesian filtering and smoothing*: *Vol. 17*, Cambridge University Press.

Schmidt, J., Hennig, P., Nick, J., & Tronarp, F. (2023). The rank-reduced Kalman filter: Approximate dynamical-low-rank filtering in high dimensions. *Advances in Neural Information Processing Systems*, *36*, 61364–61376.

Schollwöck, U. (2011). The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, *326*(1), 96–192.

Schoukens, M., & Noël, J. P. (2017). Three benchmarks addressing open challenges in nonlinear system identification. *IFAC-PapersOnLine*, *50*(1), 446–451.

Schürch, M., Azzimonti, D., Benavoli, A., & Zaffalon, M. (2020). Recursive estimation for sparse Gaussian process regression. *Automatica*, *120*, Article 109127.

Solin, A., Kok, M., Wahlström, N., Schön, T. B., & Särkkä, S. (2018). Modeling and interpolation of the ambient magnetic field by Gaussian processes. *IEEE Transactions on Robotics*, *34*(4), 1112–1127.

Solin, A., & Särkkä, S. (2020). Hilbert space methods for reduced-rank Gaussian process regression. *Statistics and Computing*, *30*, 419–446.

Stanton, S., Maddox, W., Delbridge, I., & Wilson, A. G. (2021). Kernel interpolation for scalable online Gaussian processes. In *International conference on artificial intelligence and statistics* (pp. 3133–3141). PMLR.

Svensson, A., & Schön, T. B. (2017). A flexible state–space model for learning nonlinear dynamical systems. *Automatica*, *80*, 189–199.

Tucker, Ledyard R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, *31*(3), 279–311.

Viset, F., Helmons, R., & Kok, M. (2022). An extended Kalman filter for magnetic field SLAM using Gaussian process regression. *Sensors*, *22*(8), 2833.

Wesel, F., & Batselier, K. (2021). Large-scale learning with Fourier features and tensor decompositions. In *Advances in neural information processing systems: Vol. 34*.

Wilson, A., & Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International conference on machine learning* (pp. 1775–1784). PMLR.

Clara Menzen received the M.Sc. degree in engineering science at the Technical University of Berlin, Germany. She is currently finalizing her Ph.D. in large-scale probabilistic modeling using tensor networks at Delft University of Technology, Delft, The Netherlands.

Manon Kok received the dual M.Sc. degrees in philosophy of science, technology, and society and in applied physics from the University of Twente, Enschede, The Netherlands, in 2007 and 2009, respectively, and the Ph.D. degree in automatic control from Linköping University, Linköping, Sweden, in 2017. From 2009 to 2011, she was a Research Engineer with Xsens Technologies. From 2017 to 2018, she was a Postdoctoral with the Computational and Biological Learning Laboratory, University of Cambridge, U.K. She is currently an Associate Professor with the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands. Her research interests include probabilistic inference for sensor fusion, signal processing, and machine learning.

Kim Batselier received the M.S. degree in Electromechanical Engineering and the Ph.D. Degree in Applied Sciences from the KULeuven, in 2005 and 2013 respectively. He worked as a research engineer at BioRICS on automated performance monitoring until 2009. He is currently an associate professor at the Delft University of Technology, The Netherlands. His current research interests include linear and nonlinear system theory/ identication, algebraic geometry, tensors, and numerical algorithms.