Koenig, Josie; Pfeffer, Max; Stoll, Martin

Efficient training of Gaussian processes with tensor product structure

Computational Optimization and Applications

Computational Optimization and Applications (2025) 92:563–587
https://doi.org/10.1007/s10589-025-00707-7

Efficient training of Gaussian processes with tensor product structure

Josie Koenig[1] · **Max Pfeffer**[2] · **Martin Stoll**[3]

Received: 11 January 2024 / Accepted: 6 June 2025 / Published online: 12 July 2025

Abstract
To determine the optimal set of hyperparameters of a Gaussian process based on a large number of training data, both a linear system and a trace estimation problem must be solved. In this paper, we focus on establishing numerical methods for the case where the covariance matrix is given as the sum of possibly multiple Kronecker products, i.e., can be identified as a tensor. As such, we will represent this operator and the training data in the tensor train format. Based on the AMEn method and Krylov subspace methods, we derive an efficient scheme for computing the matrix functions required for evaluating the gradient and the objective function in hyperparameter optimization.

Keywords Gaussian process · Tensor train · Trace estimation

✉ Max Pfeffer
m.pfeffer@math.uni-goettingen.de

Josie Koenig
josie.koenig@uni-potsdam.de

Martin Stoll
martin.stoll@mathematik.tu-chemnitz.de

[1] Department of Mathematics, University of Potsdam, Karl-Liebknecht-Str. 24-25, 14476 Potsdam, Germany

[2] Institute of Numerical and Applied Mathematics, University of Göttingen, Lotzestr. 16-18, 37083 Göttingen, Germany

[3] Department of Mathematics, TU Chemnitz, Reichenhainer Str. 41, 09126 Chemnitz, Germany

1 Introduction

Gaussian processes are a well-established method in machine learning and statistics to solve regression or classification problems [1, 2]. Their performance crucially depends on the choice of the hyperparameters of the covariance kernel function and their optimization is computationally expensive and requires advanced tools from large-scale numerical linear algebra. We here focus on the case where the kernel function has additional structure and as such leads to structured covariance matrices of very large dimension. One such example given in [3] are multi-output kernel functions. We discuss the efficient training if the structure of the kernel function and thus the covariance matrix or its approximation can be viewed as a tensor [4]. Here, we consider a tensor to be a multidimensional generalization of a matrix, i.e., an array with D indices:

$$\mathrm{K} \in \mathbb{R}^{n_1 \times \ldots \times n_D}.$$

Since the storage requirements of the tensor K depend on the mode D in an exponential fashion, one typically observes the *curse of dimensionality*. As a result, the tensor will be approximated using a low-rank tensor format [4, 5]. The parameter optimization requires the solution of linear systems, the approximation of a log-determinant, often reformulated as a trace estimation problem, and the computation of the gradient that then again requires linear system solves and trace estimation procedures.

Our paper starts by recalling some of the basics of Gaussian processes in Sect. 1, where also the Kronecker-sum structure of the covariance matrix will be introduced. In Sect. 2, we recall the basics of the tensor train (TT) format and describe the Krylov method in TT format. Finally, in Sect. 3, we derive formulas that are necessary to compute the cost function of the hyperparameter optimization as well as its gradient, before we show numerical experiments in Sect. 4.

Existing work

For linear systems with the kernel matrix K, i.e. $D = 2$, the recent survey [6] contains a list of references on this topic. One of the main ingredients is the acceleration of the matrix vector products with general kernel matrices K. For these matrices much research is devoted to using low-rank approximations [7–11], methods from Fourier analysis [12], or hierarchical matrices [13] in general kernel-based learning.

In more detail, low-rank techniques have been used very successfully in Gaussian process methods often based on a set of inducing points with the *subset of regressors* (SOR) [14] or its diagonal correction, the *fully independent training conditional* (FITC) [15]. Wilson and Nickisch introduce a technique based on kernel interpolation in [11]. The authors in [16] exploit this structured kernel interpolation for approximating the linear system solves as well as providing trace estimators for functions of K based on Krylov subspace methods. A more sophisticated implementation based on PyTorch was given in [17]. The challenging problem in computational Gaussian process learning and its parameter tuning is the trace estimation. This task has received much attention recently [18–22], where the finite approximation of the expectation of $x^T Ax$ is approximated. Recently, variance-reducing techniques for

this approximation have been introduced to give the *Hutch++* trace estimator or via the use of a preconditioner (cf. [23]).

In the case of the underlying structure being based on low-rank tensor approximations only a few results are available, i.e., approximating the eigenfunctions of the kernel via tensor networks [24], approximating the kernel matrix using a rank-one Kronecker product [25] and for inducing point approximations of the covariance matrix [26].

The contribution of our work is to consider the application of a tensor based training of the Gaussian process model as described in Sect. 1. In particular, for Kronecker-structured systems with non-trivial rank we introduce a low-rank tensor approach (cf. Sect. 2) that relies on the solution of a linear system with a tensor train matrix and the function of a tensor applied to a tensor in order to approximate the log-determinant term and the corresponding derivatives for training the parameters (cf. Sect. 3).

The default setting for Gaussian processes is as follows: A set of training inputs $\mathrm{X}_{train} \in \mathbb{R}^{N \times n_{train}}$ with the corresponding outputs $\mathrm{y}_{train} \in \mathbb{R}^{n_{train}}$ is given. We assume the outputs to be polluted by centered Gaussian noise, i.e., $\mathrm{y}_{train} = \mathrm{y}_{true} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathrm{I})$. The relationship between inputs and outputs must be inferred from the structure of the data. Finally, the goal is to predict the outcomes $\mathrm{y}_{test} \in \mathbb{R}^{n_{test}}$ for unseen test inputs $\mathrm{X}_{test} \in \mathbb{R}^{N \times n_{test}}$ as accurately as possible. In this paper we assume that the input–output relationship can be modeled by a Gaussian process.

Gaussian processes are a special subclass of stochastic processes. For $\mathcal{X}$ a parameter space (typically $\mathbb{R}^N$) and $(\Omega, \Sigma, p_\Omega)$ a probability space, a *stochastic process* is a family $\{y(x), x \in \mathcal{X}\}$ of random variables defined on Ω [27, p. 11]. This family can equivalently be described as a family of mappings $y : \mathcal{X} \times \Omega \mapsto \mathbb{R}, (x, \omega) \to y(x, \omega)$. For fixed $\omega \in \Omega$ we obtain functions on $\mathcal{X}$. For fixed $x \in \mathcal{X}$ we obtain random variables $y(x) : \Omega \mapsto \mathbb{R}$.

A *Gaussian process* is a stochastic process where for each finite subset $\mathrm{X} \subset \mathcal{X}$, the points $(y(x))_{x \in \mathrm{X}} \subset \mathbb{R}^{|\mathrm{X}|}$ have a joint Gaussian distribution. Since Gaussian distributions are completely defined by their mean and variance, Gaussian processes are completely defined by their mean function $m(x)$ and their covariance or kernel function $k(x, x')$, where

$$\begin{aligned} m(x) &:= \mathbb{E}[(y(x))], \\ k(x, x') &:= \mathbb{E}[(y(x) - m(x))(y(x') - m(x'))]. \end{aligned}$$

Since we have assumed that the input–output relationship in our data can be modeled by a Gaussian process, we now need to find the Gaussian process, i.e., the mean function and the covariance function, that best describe the training data. Of course, it is not feasible to search among all possible Gaussian processes, but we have to make some restrictions to actually get a solution. Thus, without loss of generality, it is usually assumed that $m(x) = 0 \ \forall x \in \mathcal{X}$(otherwise shift by the mean function). Only the covariance function $k(x, x')$ remains as an interesting property of a Gaussian process.

The learning procedure

Gaussian process learning means finding the optimal hyperparameters of a fixed family of kernel functions $k_\theta(\cdot,\cdot)$. Here we write the parameters of the kernel to be collected in the parameter vector θ. Note that we use $\tilde{\mathrm{K}} := \mathrm{K} + \sigma^2\mathrm{I}$ to include the noise and that the matrix K is obtained by evaluating $k_\theta(\cdot,\cdot)$ on each pair of training data. The kernel K depends on the parameters θ, which we write explicitly in situations where it is crucial.

The first step in the Gaussian process learning procedure is to choose a family of kernel functions to consider (e.g., squared-exponential). The goal is to find hyperparameters θ^* such that the Gaussian distribution $p(\mathrm{y}_{train}|\mathrm{X}_{train}) = \mathcal{N}(\mathbf{0}, \tilde{\mathrm{K}}(\theta)) =: Z(\theta)$ has maximum likelihood for $\theta = \theta^*$. The distribution $p(\mathrm{y}_{train}|\mathrm{X}_{train})$ is derived from the Gaussian process describing the input–output relationship in the training data.

Instead of maximizing the Gaussian likelihood directly, we minimize the negative logarithm of the likelihood function, called the *negative log-likelihood*:

$$f(\theta) = -\log Z(\theta) = \frac{1}{2}\left(n_{train}\log(2\pi) + \underbrace{\log\det(\tilde{\mathrm{K}}(\theta))}_{\substack{=\ \mathrm{tr}(\log\tilde{\mathrm{K}})(\theta) \\ \text{see Sect. (3.1.2)}}} + \mathrm{y}_{train}^{\mathrm{T}} \underbrace{\tilde{\mathrm{K}}(\theta)^{-1}\mathrm{y}_{train}}_{\text{see Sect. (3.1.1)}} \right). \quad (1)$$

In order to minimize the negative log-likelihood function, we must be able to evaluate it frequently and fast. The cost of this evaluation is dominated by solving a linear system $\tilde{\mathrm{K}}^{-1}(\theta)\mathrm{y}_{train}$ and computing its log-determinant $\log\det(\tilde{\mathrm{K}}(\theta)) = \mathrm{tr}(\log\tilde{\mathrm{K}}(\theta))$. This is addressed in Sect. 3.1. To minimize (1) the gradients of the function $f(\theta)$ are also of interest. It is well known that the analytical gradients of the log-determinant and the kernel inverse are given by

$$\begin{aligned} \frac{\partial}{\partial\theta_j}\log\det(\tilde{\mathrm{K}}(\theta)) &= \mathrm{tr}\left(\tilde{\mathrm{K}}(\theta)^{-1}\frac{\partial\tilde{\mathrm{K}}(\theta)}{\partial\theta_j}\right), \text{ see Sect. (3.2.2)} \\ \frac{\partial}{\partial\theta_j}\tilde{\mathrm{K}}(\theta)^{-1} &= -\tilde{\mathrm{K}}(\theta)^{-1}\frac{\partial\tilde{\mathrm{K}}(\theta)}{\partial\theta_j}\tilde{\mathrm{K}}(\theta)^{-1}, \text{ see Sect. (3.2.1).} \end{aligned} \quad (2)$$

These computations necessary for the minimization are typically difficult to handle, especially when the amount of training data is large, and will be addressed in Sect. 3.2.

Predictions from unseen data

After optimizing the hyperparameters, we obtain the kernel function $k_{\theta^*}(\cdot,\cdot)$ that best describes our training data. We can then obtain predictions y_{test} from new unseen input data X_{test} using the learned kernel in the following way:

The prior joint distribution of the noise-polluted outputs y_{train} and y_{test} using the optimized kernel is again Gaussian:

$$\begin{bmatrix} \mathrm{y}_{train} \\ \mathrm{y}_{test} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \tilde{\mathrm{K}} & \mathrm{K}_* \\ \mathrm{K}_*^\top & \mathrm{K}_{**} + \sigma^2 \mathrm{I} \end{bmatrix}\right),$$

where $\tilde{\mathrm{K}}$ is as above and $\mathrm{K}_* = k_{\theta^*}(\mathrm{X}_{test}, \mathrm{X}_{train})$ and $\mathrm{K}_{**} = k_{\theta^*}(\mathrm{X}_{test}, \mathrm{X}_{test})$ are the matrices obtained by evaluating the learned kernel function on the data. The predictive posterior distribution for the noisy outputs y_{test} from the test points X_{test} can then be computed by

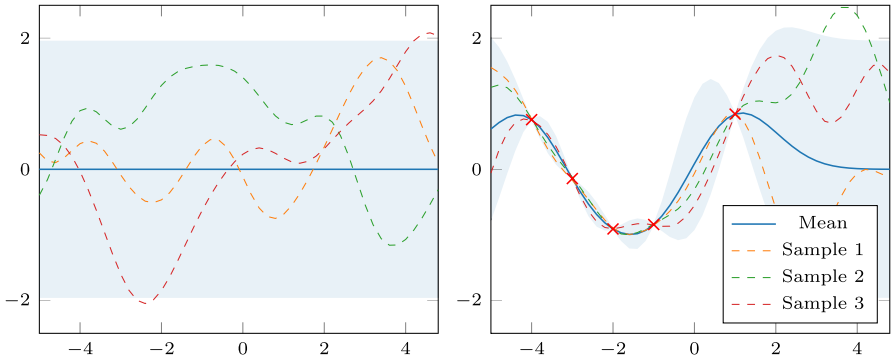
$$\begin{aligned} p(\mathrm{y}_{test} | \mathrm{X}_{test}, \mathrm{X}_{train}, \mathrm{y}_{train}) &= \mathcal{N}(\mathrm{y}_{test} | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \\ \boldsymbol{\mu}_* &= \mathrm{K}_*^\top (\mathrm{K} + \sigma^2 \boldsymbol{I})^{-1} \mathrm{y}_{train} \\ \boldsymbol{\Sigma}_* &= \mathrm{K}_{**} + \sigma^2 \mathrm{I} - \mathrm{K}_*^\top (\mathrm{K} + \sigma^2 \boldsymbol{I})^{-1} \mathrm{K}_*. \end{aligned} \tag{3}$$

Thus, one needs to compute the values of the learned kernel function for each pair of inputs to obtain the mean $\boldsymbol{\mu}_*$ and the covariance $\boldsymbol{\Sigma}_*$ of the test outputs. The desired prediction y_{test} is then obtained by sampling from the posterior Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. Alternatively, the mean $\boldsymbol{\mu}_*$ can be used as a point estimate for y_{test} and the covariance $\boldsymbol{\Sigma}_*$ describes its uncertainty. Figure 1 illustrates this prediction procedure for a one-dimensional example. Figure 1a shows the prior distribution, Fig. 1b shows the posterior distribution after the training data has been incorporated.

The above predictive procedure requires knowledge of the optimal kernel function $k_{\theta^*}(\cdot, \cdot)$, and the quality of the prediction depends crucially on its hyperparameters. From Fig. 2 it can be seen how the choice of parameters affects the posterior distribution and the ability to make meaningful predictions for unseen data. This emphasizes why the hyperparameter optimization described above is essential.

The Kronecker-sum kernel

The numerical methods to be used in hyperparameter optimization depend on the chosen kernel. In this paper, we consider a special setting for input locations on a Cartesian grid, cf. [28, p. 126 ff.]. The test and training data must be of the form



(a) Gaussian process prior (b) Gaussian process posterior with small noise

Fig. 1 **a** Gaussian process prior with no training data. **b** The posterior distribution after having incorporated 5 training points. Means are given in blue, the shaded area indicates the 95% confidence interval and the dashed lines sample from the current distribution

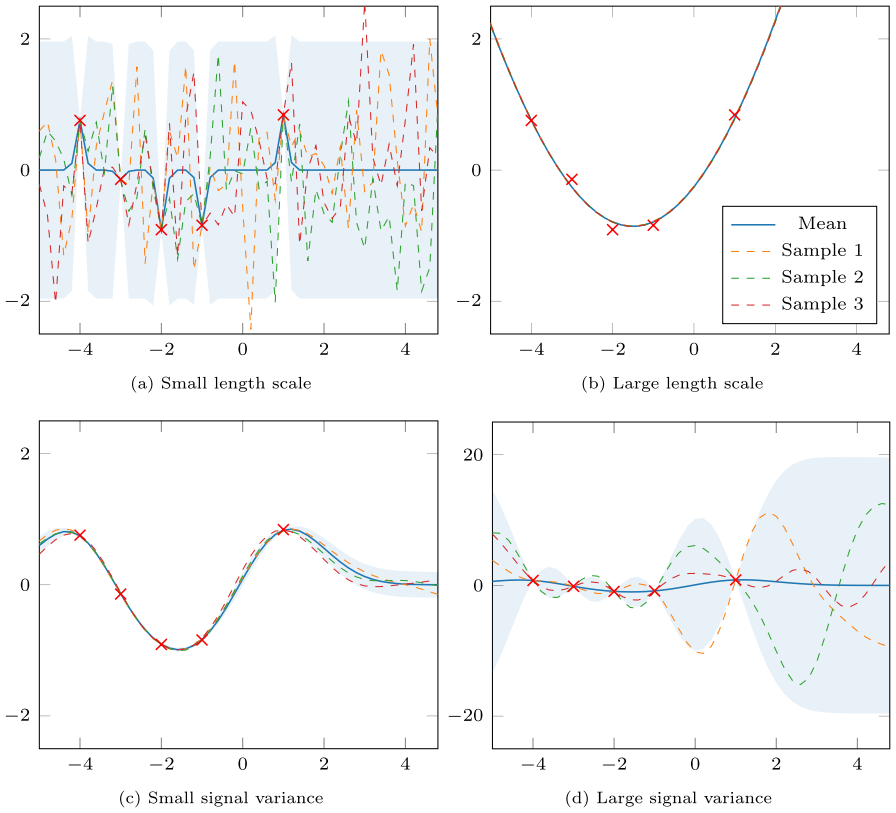


Fig. 2 Gaussian posterior from a squared-exponential kernel function with different length scale and signal variance. Training and test data are the same, given by red crosses. Means are given in blue, the shaded areas indicates the 95% confidence interval and the dashed lines sample from the current distribution. Note the different y-scale in (**d**)

$$\mathrm{X} = \mathrm{X}^{(1)} \times \ldots \times \mathrm{X}^{(D)}, \;\; \mathrm{X}^{(d)} \in \mathbb{R}^{n_d}, \;\; d = 1, \ldots, D,$$

where $\mathrm{X}^{(d)}$ contains the n_d input locations along the dimension d which may vary for each $d = 1, \ldots, D$. In total, there are $n = \prod_{d=1}^{D} n_d$ inputs for $\mathrm{X} \subset \mathbb{R}^D$ leading to outputs $\mathrm{y} \in \mathbb{R}^n = \mathbb{R}^{n_1 \times \cdots \times n_D}$. Outputs of this form can be treated as D-mode tensors and will be used as such in our computations.

With this in mind, we can now define the covariance kernel with the following Kronecker-sum structure:

$$\mathrm{K}(\mathrm{X}, \mathrm{X}) = \sum_{r=1}^{R} \mathrm{K}_r^{(1)} \otimes \mathrm{K}_r^{(2)} \otimes \ldots \otimes \mathrm{K}_r^{(D)}, \;\; \text{where } \mathrm{K}_r^{(d)} := k_r^{(d)}(\mathrm{X}^{(d)}, \mathrm{X}^{(d)}). \tag{4}$$

Such a kernel corresponds to a $2D$-mode tensor with each $\mathrm{K}_r^{(d)} \in \mathbb{R}^{n_d \times n_d}$. In particular, we will focus on the case when $R > 1$, since the case $R = 1$ with moderate individual dimensions of the kernel matrices $\mathrm{K}_1^{(d)}$ can be treated with Cholesky

decompositions of these matrices [29]. In that case, efficient training of the Gaussian process could be performed using standard techniques.

The ideas presented in this paper apply in principle to all kinds of kernel functions within the Kronecker-sum kernel given in (4). However, from now on we will assume that the one-dimensional covariance matrices $\mathrm{K}_r^{(d)}$ result from applying a squared-exponential covariance function $k_r^{(d)} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ with length scale parameter $\ell_r^{(d)}$ to each pair of inputs $x, x' \in \mathrm{X}^{(d)}$. The signal variance $\sigma_{f(r)}$ can be shifted into the first kernel function $k_r^{(1)}$, so we use:

$$\begin{aligned} k_r^{(1)}(x, x') &= \sigma_{f(r)}^2 \exp\left(-\frac{(x-x')^2}{2(\ell_r^{(1)})^2} \right), \quad r = 1, \ldots, R, \quad \text{and} \\ k_r^{(d)}(x, x') &= \exp\left(-\frac{(x-x')^2}{2(\ell_r^{(d)})^2} \right), \quad r = 1, \ldots, R, \;\; d = 2, \ldots, D. \end{aligned} \tag{5}$$

For the setting described above we need to learn the hyperparameters collected in $\theta = \{\sigma_{f(r)}, \ell_r^{(d)}\}_{r=1,\ldots,R,\, d=1,\ldots,D}$ and each hyperparameter appears only in one dimension and one summand. This will be especially important when computing the kernel derivative $\frac{\partial \tilde{\mathrm{K}}(\theta)}{\partial \theta_j}$ with respect to the different hyperparameters. For the squared exponential kernel $k_{SE}(x, x') = \sigma_f^2 \exp\left(-\frac{\|x-x'\|_2^2}{2\ell^2} \right)$, it is advantageous to use the logarithmic transformation (*log-transform*) of the hyperparameters [28, p. 138]. It holds

$$\frac{\partial k_{SE}(x, x')}{\partial \sigma_f} = \frac{2k_{SE}(x, x')}{\sigma_f} \quad \text{and} \quad \frac{\partial k_{SE}(x, x')}{\partial \ell} = \frac{\|x-x'\|_2^2}{\ell^3} k_{SE}(x, x'),$$

but due to the chain rule with the derivative of the natural logarithm we get

$$\frac{\partial k_{SE}(x, x')}{\partial \log(\sigma_f)} = 2k_{SE}(x, x') \quad \text{and} \quad \frac{\partial k_{SE}(x, x')}{\partial \log(\ell)} = \frac{\|x-x'\|_2^2}{\ell^2} k_{SE}(x, x'). \tag{6}$$

The main advantages of the log-transform are that the positivity requirements for σ_f and ℓ are automatically satisfied, and that dividing by ℓ^2 instead of ℓ^3 and not dividing by σ_f makes the derivative computation numerically more stable, especially for hyperparameters near zero. The values of the derivatives using the log-transform will be scaled relative to the true values but the roots will remain the same, i.e., $\frac{\partial k_{SE}(x,x')}{\partial \sigma_f} = 0$ iff $\frac{\partial k_{SE}(x,x')}{\partial \log(\sigma_f)} = 0$ and the same for ℓ. This justifies replacing the hyperparameters with their log-transform in the derivative computation.

Kernels with Kronecker-sum structure allow for the modeling of overlapping phenomena, as each summand models an independent stochastic process. Such settings occur, for example, in spatio-temporal magnetoencephalography (MEG) [30] or climate data sets [28]. Computations involving high-dimensional Kronecker-sum

kernels require advanced numerical algebra techniques, which will be discussed in the remainder of this paper.

2 A low-rank tensor train framework

Since our covariance operator is given in tensor product form we can use low-rank tensor formats to perform the necessary operations. Computation and storage of higher order tensors suffer from the *curse of dimensionality* since their complexity grows exponentially with the order of the tensor.

2.1 The TT format

For this reason, tensor decomposition formats have been proposed in the literature [4]. The most well-suited class of decomposition formats for the purpose of optimization are so called *Tree Tensor Networks* [5, 31]. These can be computed efficiently using higher order generalizations of the SVD, quasi-best low-rank approximations are readily available [32], and tensors with given multilinear rank form smooth manifolds that are embedded in the tensor ambient space [33, 34]. In this article, we will focus on the tensor train (TT) decomposition, as it is a good compromise between simplicity (of the format) and complexity (of computation and storage).

The TT decomposition of an order-D tensor $\mathrm{T} \in \mathbb{R}^{n_1 \times \cdots \times n_D}$ can be given elementwise as

$$\mathrm{T}(i_1, \ldots, i_D) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_{D-1}=1}^{r_{D-1}} \mathrm{T}_1(i_1, k_1)\mathrm{T}_2(k_1, i_2, k_2) \cdots \mathrm{T}_D(k_{D-1}, i_D),$$

where $\mathrm{T}_d \in \mathbb{R}^{r_{d-1} \times n_d \times r_d}$ for $d = 1, \ldots, D$ and $r_0 = r_D = 1$. The tuple $\mathrm{r} = (1, r_1, \ldots, r_{D-1}, 1)$ is called the *TT rank* of the tensor T. The tensors T_d of order (up to) three are called the *TT cores* of the TT decomposition.

Following the same idea, we decompose a linear operator $\mathrm{A} : \mathbb{R}^{n_1 \times \cdots \times n_D} \to \mathbb{R}^{m_1 \times \cdots \times m_D}$ in a TT-like format. For a fixed basis and after some reordering of indices, this operator can be given as a tensor of order $2D$ with mode sizes $m_1, n_1, \ldots, m_D, n_D$. Analogously to the TT format, we can therefore define a TT matrix format by

$$\mathrm{A}(i_1, j_1, \ldots, i_D, j_D) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_D=1}^{r_D} \bigotimes_{d=1}^{D} \mathrm{A}_d(k_{d-1}, i_d, j_d, k_d),$$

where $\mathrm{A}_d \in \mathbb{R}^{r_{d-1} \times m_d \times n_d \times r_d}$ for $d = 1, \ldots, D$ and again $r_0 = r_D = 1$.

The complexity of storage of a TT tensor is no longer exponential in the order D but linear in D and n and quadratic in the ranks. Similarly, computations in the TT format have reduced complexity: For two TT tensors of same order and dimension, but with different TT ranks, most standard operations can be computed efficiently.

This includes addition of two tensors, the Hadamard product, and the Frobenius inner product between them (denoted by dot). Furthermore, given a TT matrix of suitable column dimension, we can efficiently compute the matrix-vector product with a TT tensor by computing matrix-vector products of the cores. These operations are implemented in MATLAB, e.g., in the $\mathrm{tt\text{-}toolbox}$ [35]. For the exact definitions of the above operations, we refer the reader to [5].

The complexity of computations and storage is only efficiently reduced if the TT ranks are low. Most of the above operations increase the ranks, sometimes drastically. Therefore, we employ a rounding procedure, known as *TT-SVD*, that truncates the ranks when they become exceedingly large. The TT-SVD effectively relies on successive truncated SVDs of the reshaped TT cores. One sweep through all cores reduces the ranks to a desired magnitude or until a designated error threshold has been reached. We denote this rounding procedure by round and we predefine the desired error tolerance.

Many algorithms prove to be stable with respect to this rank rounding procedure and the errors remain moderate, see for example [36, 37] for approximate solutions to parametric PDE, or [38] for energy computations in quantum chemistry. Furthermore, given a fixed TT rank $\mathrm{r} = (1, r_1, \ldots, r_{D-1}, 1)$, the TT-SVD produces a quasi-best rank-r approximation of T, meaning that the error differs only by a factor $\sqrt{D}$ from the error of the best rank-r approximation of T [5]. For the implementation, we use the round procedure from the $\mathrm{tt\text{-}toolbox}$ in MATLAB [35], see [5, Algorithm 1].

Furthermore, from the same toolbox, we use the AMEn-method for the solution of a linear system in TT format with rank control (denoted by amen, see [39, Algorithm 4]) and also for the summation of a collection of TT tensors (denoted by $\mathrm{amen\_sum}$). These methods consist of updating the TT-cores T_d in an alternating fashion. Additionally, low-rank approximations of the residual are added in every step in order to allow for rank-adaptivity.

Since the training data $\mathrm{y}_{train} \in \mathbb{R}^{n_1 \times \cdots \times n_D}$ is a tensor, we can bring it into TT format using the (possibly truncated) SVD. Each summand $\mathrm{K}_r^{(1)} \otimes \mathrm{K}_r^{(2)} \otimes \ldots \otimes \mathrm{K}_r^{(D)}$ of the kernel matrix K is a rank-1 TT matrix and we can use summation of TT matrices to obtain the kernel in TT format (usually without rounding).

2.2 The TT-Krylov method

In order to compute the log-determinant in the negative log-likelihood $f(\theta)$ (1) and its derivative in the gradient $\nabla f(\theta)$ in the next section, we will need to efficiently evaluate expressions of the form $g(\mathrm{A})\mathrm{b}$ and $\mathrm{v}^T g(\mathrm{A})\mathrm{v}$ for (possibly nonsymmetric) TT matrices A, TT tensors b, v, and g a matrix function. For this, we now introduce the *TT-Krylov method* in this section.

We start with a general introduction for the matrix case. Even then, this problem poses a significant challenge as the evaluation of matrix functions is expensive, especially once the matrix dimensions are large [40, 41]. Many efficient techniques exist for evaluating a matrix function times a vector, i.e., $g(\mathrm{A})\mathrm{b}$. Problems of the form $\mathrm{v}^T g(\mathrm{A})\mathrm{u}$ $(\mathrm{often\,with\,u} = \mathrm{v})$ have been studied in the seminal works of Golub and

Meurant [42–44] that are rooted in the close connection of this expression to Gaussian quadrature.

In this paper we focus on techniques based on Krylov subspaces and briefly illustrate our approach here. The Arnoldi method is used to generate an orthonormal basis for the Krylov subspace

$$\mathcal{K}_\ell(\mathrm{A},\mathrm{b}) = \mathrm{span}\left\{\mathrm{b}, \mathrm{Ab}, \ldots, \mathrm{A}^{\ell-1}\mathrm{b}\right\}$$

via a Gram-Schmidt procedure. One obtains the following decomposition

$$\mathrm{AV}_\ell = \mathrm{V}_\ell \mathrm{H}_\ell + h_{\ell+1,\ell}\mathrm{v}_{\ell+1}\mathrm{e}_\ell^{\mathrm{T}},$$

where the vectors v_k are the basis vectors of the Krylov subspace, $\mathrm{V}_\ell = [\mathrm{v}_1 \cdots \mathrm{v}_\ell]$, and $\mathrm{H}_\ell \in \mathbb{R}^{\ell\times\ell}$ a Hessenberg matrix. With the choice of selecting the seed vector for the Krylov subspace to be

$$\mathrm{v}_1 = \mathrm{b}/\|\mathrm{b}\|$$

we obtain the following approximation

$$g(\mathrm{A})\mathrm{b} \approx \|\mathrm{b}\|\mathrm{V}_\ell g(\mathrm{H}_\ell)\mathrm{e}_1,$$

which is based on the approximation resulting from the Arnoldi procedure. Since the dimensionality of H_ℓ is much smaller than the dimensionality of A, $g(\mathrm{H}_\ell)$ can now be evaluated efficiently [40].

An alternative to standard Krylov methods are rational Krylov methods [41, 45, 46]. These methods approximate the computation of $g(\mathrm{A})\mathrm{b}$ via a rational polynomial $r_\ell(\mathrm{A})$ via $r_\ell(\mathrm{A})\mathrm{b}$, where $r_\ell = \frac{p_{\ell-1}}{q_{\ell-1}}$ and $q_{\ell-1}(z) = \prod_{j=1}^{\ell-1}(1-\frac{z}{\xi_j})$ with poles ξ_j. The rational Krylov space is then given via $\mathcal{Q}_\ell(\mathrm{A},\mathrm{b}) = q_{\ell-1}(\mathrm{A})^{-1}\mathrm{span}\left\{\mathrm{b}, \mathrm{Ab}, \ldots, \mathrm{A}^{\ell-1}\mathrm{b}\right\}$. The corresponding Arnoldi method then reads in matrix form as $\mathrm{AV}_{\ell+1}\mathrm{K}_\ell = \mathrm{V}_{\ell+1}\mathrm{H}_\ell$ where $\mathrm{K}_\ell, \mathrm{H}_\ell$ are $\ell+1$ by ℓ unreduced Hessenberg matrices. The approximation is then obtained via $g(\mathrm{A})\mathrm{b} \approx \mathrm{V}_\ell g(\mathrm{A}_\ell)\mathrm{V}_\ell^T\mathrm{b}$, where the Rayleigh quotient $\mathrm{A}_\ell = \mathrm{V}_\ell^T\mathrm{AV}_\ell$ can also be written as $\mathrm{A}_\ell = \mathrm{H}_\ell\mathrm{K}_\ell^{-1}$. As a result we can get the approximation from the fact that $g(\mathrm{A})\mathrm{b} \approx \|\mathrm{b}\|\mathrm{V}_\ell g(\mathrm{A}_\ell)\mathrm{e}_1$.

In particular, if we are interested in an computing the value of the term $\mathrm{v}^{\mathrm{T}}g(\mathrm{A})\mathrm{v}$ we get as an approximation from the Arnoldi procedure

$$\mathrm{v}^{\mathrm{T}}g(\mathrm{A})\mathrm{v} \approx \|\mathrm{v}\|^2\mathrm{e}_1^{\mathrm{T}}g(\mathrm{H}_\ell)\mathrm{e}_1,$$

and from the rational Arnoldi we see

$$\mathrm{v}^{\mathrm{T}}g(\mathrm{A})\mathrm{v} \approx \|\mathrm{v}\|^2\mathrm{e}_1^{\mathrm{T}}g(\mathrm{A}_\ell)\mathrm{e}_1.$$

One can also rely on approximations based on the nonsymmetric Lanczos process [47, 48] but these are more delicate for the evaluation of expressions of the form $\mathrm{u}^{\mathrm{T}} g(\mathrm{A})\mathrm{v}$ as one typically requires $\mathrm{u}^{\mathrm{T}}\mathrm{v} \neq 0$, which is not satisfied in our case [see (9)].

For the approximation of the gradient and trace estimation we employ the Krylov and rational Krylov method implemented in the tensor train format. For this, the steps of any Krylov method are performed using the tensor train arithmetic

$$\mathrm{KV}_\ell = \mathrm{V}_\ell \mathrm{H}_\ell + h_{\ell+1,\ell} v_{\ell+1} \mathrm{e}_\ell^{\mathsf{T}},$$

where now V_ℓ represents a collection of ℓ TT tensors. The matrix H_ℓ here contains the coefficients resulting from the orthogonalization process within the Arnoldi method. Even though the elements in V_ℓ are TT tensors, H_ℓ remains an $\ell \times \ell$ matrix to which we apply the matrix function. This can be done in the same way for the rational Krylov methods.

However, we need to make sure that the TT ranks do not grow too large. Therefore, we round off the TT ranks after each orthogonalization step. This is done using the round procedure with a predefined truncation tolerance. This affects the orthogonality of the basis vectors v_k and we therefore repeat the orthogonalization (including the rounding) once more. Our experiments have shown that this second orthogonalization is enough and another round does not lead to significant gains. This is akin to a reorthogonalization step in the matrix case. Our experiments show that then, the ranks remain managable and the error is small. Ultimately, this rounding step is the only alteration of the Krylov method in the matrix case. As a stopping criterion we use the difference between two consecutive approximations to the desired quantities.

In the case of approximating

$$\mathrm{v}^{\mathrm{T}} g(\mathrm{A})\mathrm{u}$$

techniques based on the nonsymmetric Lanczos process can be applied [42] but since these might suffer from serious breakdowns we will use the approximation $g(\mathrm{A})\mathrm{u}$ and then compute the inner product with v afterwards.

The procedure is summarized in Algorithm 1, where we show the case for a nonsymmetric TT-matrix A. In the symmetric case, lines 3 to 6 of the Algorithm would only orthogonalize for $j = k-1, k$. If we are interested in the approximation of $\mathrm{v}^{\mathrm{T}} g(\mathrm{A})\mathrm{v}$ we would replace line 12 by

$$\mathrm{v}^{\mathrm{T}} g(\mathrm{A})\mathrm{v} \approx \|\mathrm{v}\|^2 \mathrm{e}_1^{\mathrm{T}} g(\mathrm{H}_\ell)\mathrm{e}_1,$$

and denote the call by $\mathrm{tt\_krylov}(\mathrm{g}, \mathrm{A}, \mathrm{v}, \mathrm{v}, \mathrm{tunctol})$.

Input: $g, \mathbf{A}, \mathbf{b}, \text{trunctol}$
Output: $g(\mathbf{A})\mathbf{b}$
for $k = 1 : \text{maxit}$ **do**
  $\mathbf{w}_k = \mathbf{A}\mathbf{v}_k$
  for two loops **do**
    for $j = 1 : k$ **do**
      $h_{j,k} = h_{j,k} + \texttt{dot}(\mathbf{v}_j, \mathbf{w}_k)$
      $\mathbf{w}_k = \texttt{round}(\mathbf{w}_k - h_{k,j}\mathbf{v}_j, \text{trunctol})$
    end for
  end for
  $h_{k+1,k} = \texttt{dot}(\mathbf{w}_k, \mathbf{w}_k)^{1/2}$
  $\mathbf{v}_{k+1} = \frac{\mathbf{w}_k}{h_{k+1,k}}$
  $\mathbf{gH} = g(\mathbf{H}(1:k, 1:k))$
  $g(\mathbf{A})\mathbf{b} \approx \texttt{amen\_sum}(\mathbf{v}_{1:k}, \mathbf{gH}_{:,1} \|\mathbf{b}\|, \text{trunctol})$
  Check difference between to consecutive approximations and if small **stop.**
end for

Algorithm 1 $\text{tt\_krylov}(\text{g}, \text{A}, \text{b}, \text{trunctol})$: Algorithm for approximating $g(\text{A})\text{b}$ in TT format. The algorithm shows Arnoldi iteration for the basis creation

3 Minimizing the negative log-likelihood in TT format

We recall that the task in Gaussian process learning is to minimize the negative log-likelihood (1) for the parameters θ:

$$\theta^* = \arg\min_{\theta} \frac{1}{2}\left(n_{train}\log(2\pi) + \log\det(\tilde{\mathrm{K}}(\theta)) + \mathrm{y}_{train}^{\mathrm{T}}\tilde{\mathrm{K}}(\theta)^{-1}\mathrm{y}_{train}\right).$$

Since the covariance matrix $\mathrm{K}(\theta)$ and the training data y_{train} can be represented in TT format, all operations in the optimization problem are performed in TT format also. As discussed, this results in a significant reduction of complexity and it allows for the solution of very large problems. For the minimization of the negative log-likelihood, we employ a basic LBFGS-solver provided by MATLAB. For this, all we need is the efficient computation of the cost function and the gradient. This requires the evaluation of the log-determinant and the linear system (in the cost function) and the computation of the trace and another linear system (in the gradient). We describe our procedure in the following subsections.

3.1 Computing the cost function

3.1.1 Solving a linear system in TT format

The first part of the objective function we discuss is the solution of the linear system that comes from the evaluation of the term $\mathrm{y}^{\mathrm{T}}(\mathrm{K} + \sigma^2\mathrm{I})^{-1}\mathrm{y}$. Computing the Cholesky decomposition of the matrix $\tilde{\mathrm{K}} = \mathrm{K} + \sigma^2\mathrm{I}$ is too costly for large D. For this we will employ the AMEN method introduced in [39] without any further modifica-

tions. This method efficiently approximates the solution of a linear system given in TT format, while maintaining a low TT rank. It consists of an alternating procedure that cycles through the TT components and optimizes them separately. In each step, a small fraction of the residual is used in order to adapt the TT ranks. We refer the reader to the original article for more details.

3.1.2 Trace estimation and matrix functions

The required efficient evaluation of the log-determinant becomes intractable for large matrix dimensions and we rely on the equivalent relation

$$\log\det(\tilde{\mathrm{K}}) = \mathrm{tr}(\log(\tilde{\mathrm{K}})).$$

Using this will allow us to avoid the computation of the determinant of $\tilde{\mathrm{K}}$ altogether. Note that again, the problem would be rather easy if we could afford a Cholesky decomposition of the matrix $\tilde{\mathrm{K}}$. Our goal is to estimate the trace of the matrix logarithm. For this, we use the Hutchinson trace estimator

$$\mathrm{tr}(\mathrm{A}) = \mathbb{E}(\mathrm{z}^{\mathrm{T}}\mathrm{Az})$$

for a Rademacher or Gaussian random vector z. Naturally, we approximate this using a Monte-Carlo approach, i.e., with a finite sum

$$\mathrm{tr}(\mathrm{A}) \approx \frac{1}{p}\sum_{i=1}^{p}\left(\mathrm{z}_i^{\mathrm{T}}\mathrm{Az}_i\right) \tag{7}$$

for p i.i.d. Rademacher vectors z_i. For $\mathrm{A} = \log(\tilde{\mathrm{K}}(\theta))$ we can now use the trace estimator for the evaluation of the objective function. However, as is usually the case for naive Monte-Carlo style approaches, the relative error of the trace will be proportional to $\frac{1}{\sqrt{p}}$. In the literature, it is common to choose relatively low numbers of probe vectors ($p < 100$) [16]. The error is then acceptable for the evaluation of the cost function. However, when computing the gradient (where another trace estimation will become necessary), we would need higher accuracy to ensure that the search direction is actually a descent direction of the cost function. For this reason, it is common to replace the cost function by a numerically efficient version

$$f(\theta) \approx \hat{f}(\theta) := \frac{1}{2}\left(N\log(2\pi) + \frac{1}{p}\sum_i(\mathrm{z}_i^{\mathrm{T}}\log(\tilde{\mathrm{K}}(\theta))\mathrm{z}_i) + \mathrm{y}^{\mathrm{T}}(\tilde{\mathrm{K}}(\theta))^{-1}\mathrm{y}\right). \tag{8}$$

In the following, we therefore minimize this modified cost function instead.

The computational challenge now lies in efficiently evaluating the quantity $\sum_i(\mathrm{z}_i^{\mathrm{T}}\log(\tilde{\mathrm{K}})\mathrm{z}_i)$. This is done using the symmetric TT-Krylov method introduced in Sect. 2.2. We also need to convert the probe vectors z_i to TT format. However, these

vectors will usually have full TT rank, and we therefore use TT tensors of rank 1, where each component is an i.i.d. Rademacher vector:

$$\mathrm{z}_i = \mathrm{z}_i^{(1)} \otimes \cdots \otimes \mathrm{z}_i^{(D)}, \qquad \mathrm{z}_i^{(d)} \sim \mathcal{U}([-1,1]^{n_d}),\ d = 1, \ldots, D.$$

This keeps the TT ranks managable during the TT-Krylov procedure, and we obtain a framework for the efficient computation of the cost function.

We summarize the computation of the cost function in Algorithm 2

Input: $\theta = \{\sigma_{f(r)}, \ell_r^{(d)}\}_{r=1,\ldots,R,\, d=1,\ldots,D}, p, \mathrm{kryltol}, \mathrm{amentol}$
Output: $\hat{f}(\theta)$
1: $t = 0$
2: **for** $i = 1 : p$ **do**
3: $\quad \mathbf{z}_i = \mathbf{z}_i^{(1)} \otimes \cdots \otimes \mathbf{z}_i^{(D)},\ \mathbf{z}_i^{(d)} \sim \mathcal{U}([-1,1]^{n_d}),\ d = 1, \ldots, D$
4: $\quad t = t + \texttt{tt\_krylov}(\texttt{logm}, \tilde{\mathbf{K}}(\theta), \mathbf{z}_i, \mathbf{z}_i, \mathrm{kryltol})$
5: **end for**
6: $\alpha = \texttt{amen}(\tilde{\mathbf{K}}(\theta), \mathbf{y}_{\texttt{train}}, \mathrm{amentol})$
7: $\hat{f}(\theta) = (1/2)(N \log(2\pi) + t/p + \mathbf{y}_{train}^{\mathrm{T}} \alpha)$

Algorithm 2 Algorithm for the computation of the cost function

3.2 Computing the gradient

For the training of the hyperparameters we require the derivative of the approximate negative log-likelihood (8) with respect to the parameters θ. Therefore, we need the derivative of the inverse kernel as well as that of the log-determinant.

3.2.1 Derivative of the inverse kernel

Using the inverse function rule, it holds that

$$\frac{\partial \tilde{\mathrm{K}}(\theta)^{-1}}{\partial \theta_j} = -\tilde{\mathrm{K}}(\theta)^{-1} \frac{\partial \tilde{\mathrm{K}}(\theta)}{\partial \theta_j} \tilde{\mathrm{K}}(\theta)^{-1}.$$

Since $\tilde{\mathrm{K}}(\theta) = \mathrm{K}(\theta) + \sigma^2 \mathrm{I}$, we therefore require the efficient evaluation of the quadratic expression

$$\mathrm{y}_{train}^{\mathrm{T}} \frac{\partial \tilde{\mathrm{K}}^{-1}(\theta)}{\partial \theta_j} \mathrm{y}_{train} = -\mathrm{y}_{train}^{\mathrm{T}} \tilde{\mathrm{K}}^{-1}(\theta) \frac{\partial \mathrm{K}(\theta)}{\partial \theta_j} \tilde{\mathrm{K}}^{-1}(\theta) \mathrm{y}_{train} = -\alpha^{\mathrm{T}} \frac{\partial \mathrm{K}(\theta)}{\partial \theta_j} \alpha,$$

where $\tilde{\mathrm{K}}(\theta)\alpha = \mathrm{y}_{train}$ can be precomputed using again the AMEn method. The computation of the derivative $\frac{\partial \mathrm{K}(\theta)}{\partial \theta_j}$ in TT matrix format will be addressed below.

3.2.2 Derivative of the log-determinant

If we now study (2) it might seem natural to assume that we approximate the derivative of the log-determinant using

$$\frac{\partial \log\det(\tilde{\mathrm{K}}(\theta))}{\partial\theta_j} = \operatorname{tr}\left(\tilde{\mathrm{K}}(\theta)^{-1}\frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}\right) \approx \frac{1}{p}\sum_i \mathrm{z}_i^{\mathrm{T}}\tilde{\mathrm{K}}(\theta)^{-1}\frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}\mathrm{z}_i.$$

But in our scheme we are interested in the gradient of $\hat{f}(\theta)$, which requires the derivative of the expression

$$\mathrm{z}_i^{\mathrm{T}}\frac{\partial \log(\tilde{\mathrm{K}}(\theta))}{\partial\theta_j}\mathrm{z}_i.$$

If $\tilde{\mathrm{K}}(\theta)^{-1}$ and $\frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}$ commute, the derivative of the matrix logarithm with respect to a parameter is (see [49] for a detailed derivation)

$$\frac{\partial \log(\tilde{\mathrm{K}}(\theta))}{\partial\theta_j} = \tilde{\mathrm{K}}(\theta)^{-1}\frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}.$$

However, in our case, $\tilde{\mathrm{K}}(\theta)^{-1}$ and $\frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}$ do not commute, and hence we need to find a way to compute $\frac{\partial \log(\tilde{\mathrm{K}}(\theta))}{\partial\theta_j}$ efficiently. For this, we use a well known result from the theory of matrix functions [40, 50]: The directional derivative $L_{\log}(\mathrm{A},\mathrm{E})$ of the matrix logarithm $\log(\mathrm{A})$ in the direction E can be computed via

$$\log\left(\begin{bmatrix}\mathrm{A} & \mathrm{E}\\ 0 & \mathrm{A}\end{bmatrix}\right) = \begin{bmatrix}\log(\mathrm{A}) & L_{\log}(\mathrm{A},\mathrm{E})\\ 0 & \log(\mathrm{A})\end{bmatrix}.$$

Choosing the direction $E = \frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}$, we obtain

$$\frac{\partial \log(\tilde{\mathrm{K}}(\theta))}{\partial\theta_j} = L_{\log}\left(\tilde{\mathrm{K}}(\theta), \frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}\right).$$

This can can be computed efficiently using the nonsymmetric TT-Krylov method and using the fact that

$$\begin{bmatrix}\mathrm{z}_i^{\mathrm{T}} & 0\end{bmatrix}\log\left(\begin{bmatrix}\tilde{\mathrm{K}}(\theta) & \frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}\\ 0 & \tilde{\mathrm{K}}(\theta)\end{bmatrix}\right)\begin{bmatrix}0\\ \mathrm{z}_i\end{bmatrix} = \mathrm{z}_i^{\mathrm{T}}L_{\log}\left(\tilde{\mathrm{K}}(\theta), \frac{\partial \mathrm{K}(\theta)}{\partial\theta_j}\right)\mathrm{z}_i. \tag{9}$$

We note that $\begin{bmatrix}\mathrm{z}_i^{\mathrm{T}} & 0\end{bmatrix}$ and $\begin{bmatrix}0\\ \mathrm{z}_i\end{bmatrix}$ are TT tensors of order $D+1$ and with TT rank 1. Altogether, we obtain an efficient way to compute the derivative of the trace estimator used in (8).

3.2.3 Derivatives for the tensor parameters

For the above computations, we need to evaluate the derivative $\frac{\partial \mathrm{K}(\theta)}{\partial \theta_j}$ in TT format. We recall that

$$\mathrm{K} = \sum_{r=1}^{R} \bigotimes_{d=1}^{D} \mathrm{K}_r^{(d)}, \text{ where } \mathrm{K}_r^{(d)} := k_r^{(d)}(\mathrm{X}^{(d)}, \mathrm{X}^{(d)}) \in \mathbb{R}^{n_d \times n_d},$$

with the 1-dimensional covariance matrices given by squared-exponential covariance functions and the signal variance shifted to the first covariance matrix, see (5).

Hence, we can compute

$$\begin{aligned}
\frac{\partial \mathrm{K}}{\partial \ell_j^{(i)}} &= \frac{\partial \sum_{r=1}^{R} \bigotimes_{d=1}^{D} \mathrm{K}_r^{(d)}}{\partial \ell_j^{(i)}} = \sum_{r=1}^{R} \frac{\partial \bigotimes_{d=1}^{D} \mathrm{K}_r^{(d)}}{\partial \ell_j^{(i)}} \\
&= \sum_{r=1}^{R} \sum_{d=1}^{D} \frac{\partial \mathrm{K}_r^{(d)}}{\partial \ell_j^{(i)}} \otimes \left(\bigotimes_{k \neq d} \mathrm{K}_r^{(k)} \right) \text{ (property of Kronecker product)} \\
&= \frac{\partial \mathrm{K}_j^{(i)}}{\partial \ell_j^{(i)}} \otimes \left(\bigotimes_{d \neq i} \mathrm{K}_j^{(d)} \right), \text{ as } \frac{\partial \mathrm{K}_r^{(d)}}{\partial \ell_j^{(i)}} = 0 \text{ if } d \neq i \text{ or } r \neq j.
\end{aligned}$$

And similarly, for $r \in \{1, \ldots, R\}$:

$$\frac{\partial \mathrm{K}}{\partial \sigma_{f(r)}} = \frac{\partial \mathrm{K}_r^{(1)}}{\partial \sigma_{f(r)}} \otimes \left(\bigotimes_{d \neq 1} \mathrm{K}_r^{(d)} \right).$$

Here again, $\bigotimes_{d \neq i} \mathrm{K}_r^{(d)}$ can efficiently be computed as the Kronecker product of TT tensors. The matrix derivative $\frac{\partial \mathrm{K}_r^{(d)}}{\partial \theta}$ is now easy to compute for θ any length scale or signal variance parameter. As all covariance matrices result from squared exponential kernels, we use the log-transformed derivatives given by (6) pointwise and obtain

$$\frac{\partial k_r^{(1)}(x, x')}{\partial \log(\sigma_{f(r)})} = 2 k_r^{(1)}(x, x'),$$

and

$$\frac{\partial k_r^{(d)}(x, x')}{\partial \log(\ell_r^{(d)})} = \frac{(x - x')^2}{(\ell_r^{(d)})^2} k_r^{(d)}(x, x'). \tag{10}$$

This computation has to be performed for each pair of inputs $x, x' \in \mathrm{X}^{(d)}$ to construct $\frac{\partial \mathrm{K}_r^{(1)}}{\partial \sigma_{f(r)}}$ and $\frac{\partial \mathrm{K}_r^{(d)}}{\partial \ell_r^{(d)}}$ for all possible $d \in \{1, \ldots, D\}$ and $r \in \{1, \ldots, R\}$. When

the matrix of pointwise squared differences is also given in TT format, $\frac{\partial \mathrm{K}_r^{(d)}}{\partial \ell_r^{(d)}}$ can be computed from (10) by a Hadamard product of TT matrices.

We summarize the computation of the gradient in Algorithm 3 and we now have all the ingredients to call MATLAB's fminunc. Later, in Sect. 4.3, Fig. 8, we see that the approximation of the gradient is indeed accurate.

Input: $\theta = \{\sigma_{f(r)}, \ell_r^{(d)}\}_{r=1,\ldots,R,\, d=1,\ldots,D}, p, \mathrm{kryltol}, \mathrm{amentol}$
Output: $\nabla \hat{f}(\theta)$
1: **for** $j = 1 : \texttt{length}(\theta)$ **do**
2: $\quad t = 0$
3: $\quad$ **for** $i = 1 : p$ **do**
4: $\quad\quad \mathbf{z}_i = \mathbf{z}_i^{(1)} \otimes \cdots \otimes \mathbf{z}_i^{(D)},\ \mathbf{z}_i^{(d)} \sim \mathcal{U}([-1,1]^{n_d}),\ d = 1, \ldots, D$
5: $\quad\quad t = t + \begin{bmatrix}\mathbf{z}_i^{\mathsf{T}} & 0\end{bmatrix} \texttt{tt\_krylov}\left(\texttt{logm}, \begin{bmatrix}\tilde{\mathbf{K}}(\theta) & \frac{\partial \mathbf{K}(\theta)}{\partial \theta_j} \\ 0 & \tilde{\mathbf{K}}(\theta)\end{bmatrix}, \begin{bmatrix}0 \\ \mathbf{z}_i\end{bmatrix}, \mathrm{kryltol}\right)$
6: $\quad$ **end for**
7: $\quad \alpha = \texttt{amen}(\tilde{\mathbf{K}}(\theta), \mathbf{y}_{\texttt{train}}, \mathrm{amentol})$
8: $\quad \nabla \hat{f}(\theta)_j = t/p + \alpha^{\mathrm{T}} \frac{\partial \mathbf{K}(\theta)}{\partial \theta_j} \alpha$
9: **end for**

Algorithm 3 Algorithm for the computation of the gradient

4 Numerical experiments

We test our method on two synthetic cases: In one, we generate a random trigonometric function in 3 dimensions. In the second example, we draw a random sample from a Gaussian process with given covariance kernel. In both cases, we learn the parameters of our Gaussian process on some training points and evaluate the results on a range of test data. This is done for an increasing number of probe vectors in the trace estimation.

We start the construction of the test problem by generating a tensor $\mathrm{X}_{train} \in \mathbb{R}^{N \times N \times N}$ as a 3-dimensional grid in $[-1, 1]^3$:

$$\mathrm{X}_{train} = \mathrm{x}_1 \otimes \mathrm{x}_2 \otimes \mathrm{x}_3,$$

where the points are equally spaced via the construction

$$\mathrm{x}_1 = \mathrm{x}_2 = \mathrm{x}_3 = \left[-1 : \frac{2}{N-1} : 1\right].$$

This tensor contains the training points. As test points, we generate the complimentary grid $\mathrm{X}_{test} \in \mathbb{R}^{N-1 \times N-1 \times N-1}$

$$\mathrm{X}_{test} = \tilde{\mathrm{x}}_1 \otimes \tilde{\mathrm{x}}_2 \otimes \tilde{\mathrm{x}}_3,$$

with

$$\tilde{\mathrm{x}}_1 = \tilde{\mathrm{x}}_2 = \tilde{\mathrm{x}}_3 = \Big[-1 + \frac{1}{N-1} : \frac{2}{N-1} : 1 - \frac{1}{N-1} \Big].$$

The training labels are then given as $\mathrm{y}_{train} \in \mathbb{R}^{N \times N \times N}$ and $\mathrm{y}_{test} \in \mathbb{R}^{N-1 \times N-1 \times N-1}$, respectively.

4.1 Trigonometric function with random coefficients

In the first experiment, we set $N = 21$ and generate a random tensor $\mathrm{R} \in \mathbb{R}^{3 \times 3 \times 2}$ with entries uniformly distributed in the interval [0, 1]. The training labels are then given as

$$\begin{aligned} \mathrm{y}_{train}(i,j,k) = \sum_{r=1}^{3} & \sin(\pi \mathrm{R}(r,1,1)\mathrm{x}_1(i) + \frac{\pi}{2}\mathrm{R}(r,1,2)) \sin(\pi \mathrm{R}(r,2,1)\mathrm{x}_2(i) \\ & + \frac{\pi}{2}\mathrm{R}(r,2,2)) \sin(\pi \mathrm{R}(r,3,1)\mathrm{x}_3(i) + \frac{\pi}{2}\mathrm{R}(r,3,2)), \end{aligned}$$

and similarly for the test points y_{test} with the same parameter tensor R. We add the random noise with $\sigma = 0.01$ and the resulting tensors are then converted into TT format with rounding error 10^{-8}.

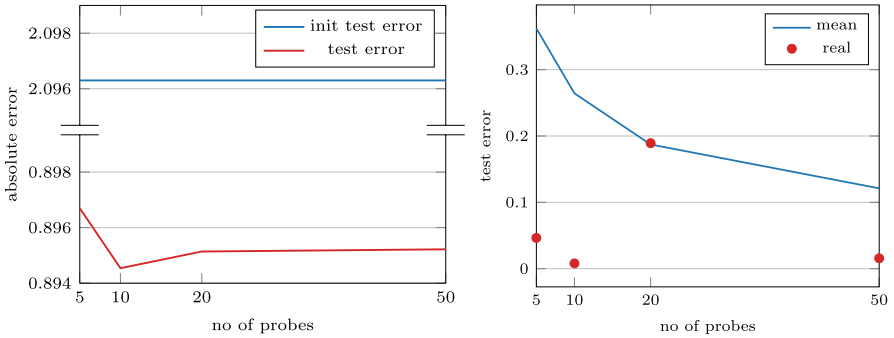
Since the training points are generated by a sum of 3 products of univariate trigonometric functions, we will attempt to reconstruct this function using a Gaussian process with a Kronecker-sum kernel of rank $R = 3$ and order $D = 3$. We initialize our method with parameters $\sigma_{f(r)} = 1 + \varepsilon, r = 1, 2, 3$ and $\ell_r^{(d)} = 0.1 + \varepsilon, r = 1, 2, 3, d = 1, 2, 3$ where ε is random noise in $\mathcal{N}(0, 0.005)$ and different for each parameter.

We set the tolerance for the TT-Krylov method and the AMEn subsolver to 10^{-6} and we run our method using the LBFGS solver in MATLAB's fminunc routine (the tolerance for the norm of the gradient was set to 10^{-10}). We test different numbers of probe vectors in the trace estimation and report on the results.

In Fig. 3a, we show the absolute Frobenius error $\|\mathrm{y}_{test} - \boldsymbol{\mu}_*\|_{\mathrm{F}}$ of the posterior mean tensor $\boldsymbol{\mu}_*$ in (3) to the test tensor y_{test}. We can see that in all cases, the error is significantly reduced as compared to the random initialization. However, the error does not visibly improve for different numbers of probe vectors p.

This is most likely due to the random nature of the trace estimation. In Fig. 3b, we show the relative error of the trace estimation (7) in the first iteration. We can see that this error varies significantly from the mean. This explains the fact that we can get good approximation errors with only a small number of probes.

Finally, in Fig. 4, we show a comparison of one slice of the test tensor and the posterior mean for 10 probes (as this gave the best test error). We chose the 10th slice and compare $\mathrm{y}_{test}(:, 10, :)$ with $\boldsymbol{\mu}_*(:, 10, :)$ as well as the absolute error of the two. We can see that in this experiment, the interpolation works very well and the pointwise errors are small.



(a) Test errors (b) Trace estimation error

Fig. 3 **a** Comparison of the test error of the Gaussian process with the initial hyperparameters (blue) and with optimized hyperparameters (red) plotted against an increasing number of probe vectors for the trace estimation. **b** Error of the trace estimation based on the number of probes with the mean approximation (blue line) and the actual value (red)

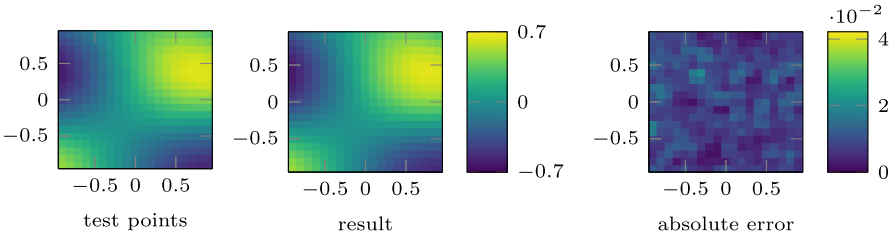


Fig. 4 Slice of a data tensor on the left for the test data and in the middle the same slice of the posterior mean using the optimized parameters. The error is shown on the right

4.2 Random sample from Gaussian process

In the second example, we repeat the previous experiment but with a different training labels y_{train} and also different test points y_{test}. We take the same grid X but generate these tensors by drawing them randomly from the Gaussian process with Kronecker covariance kernel $\mathrm{K} = \sum_{r=1}^{3} \bigotimes_{d=1}^{3} \mathrm{K}_r^{(d)}$, using the parameters

$$\begin{aligned} \sigma_{f(1)} &= 1, & \sigma_{f(2)} &= 0.1, & \sigma_{f(3)} &= 0.01, \\ \ell_1^{(1)} &= 0.06, & \ell_2^{(1)} &= 0.2, & \ell_3^{(1)} &= 0.3, \\ \ell_1^{(2)} &= 0.05, & \ell_2^{(2)} &= 0.19, & \ell_3^{(2)} &= 0.4, \\ \ell_1^{(3)} &= 0.04, & \ell_2^{(3)} &= 0.21, & \ell_3^{(3)} &= 0.5. \end{aligned}$$

The idea is here that short-range interactions ($\ell_r^{(d)} < 1/N$) outweigh the longer interactions in the Kronecker terms 2 and 3.

The training tensor is produced by drawing two random tensors $\mathrm{U}, \mathrm{V} \in \mathbb{R}^{(2N-1)\times(2N-1)\times(2N-1)}$ with entries drawn from the standard normal distribution, converting them to the TT format with rounding error 10^{-8} and computing

$$\mathrm{y}_{full} = \mathrm{K}^{1/2}\mathrm{U} + \sigma\mathrm{V},$$

where $\sigma = 0.01$ is the added observational noise. The matrix square root is computed using our Krylov method for TT tensors and $g(\cdot) = \sqrt{\cdot}$. The resulting tensor is then rounded again with accuracy 10^{-8}. We then compute the training and test tensors by

$$\begin{aligned} \mathrm{y}_{train}(i,j,k) &= \mathrm{y}_{full}(2i-1, 2j-1, 2k-1), & i,j,k &= 1,\ldots,N, \\ \mathrm{y}_{test}(i,j,k) &= \mathrm{y}_{full}(2i, 2j, 2k), & i,j,k &= 1,\ldots,N-1. \end{aligned}$$

We again reconstruct this function using a Gaussian process with covariance kernel of rank $R = 3$ and order $D = 3$ and we initialize the method with the same values used above. All other parameters remain the same as well.

In Fig. 5, we again show the absolute Frobenius error of the posterior mean tensor $\boldsymbol{\mu}_*$ to the test tensor y_{test}. Apart from the error for the posterior mean using the initial parameters we also show the test error for the (known) ground truth parameters set above. We can see that the test error of our method fluctuates. In the case of 20 probes, we get about the same test error as for the ground truth.

However, this experiment is clearly harder. In Fig. 6, we show the 10th slice of the computed tensor in the case of 20 probes, as compared to the test tensor. In this case, we see that the error is much larger, and the resulting tensor is only a blurry approximation of the original function. We cannot expect a much better result, since the test error is the same also for the ground truth. This is due to the random nature of the problem and it likely being more difficult to approximate with a low-rank method given the little smoothness that the data possesses.

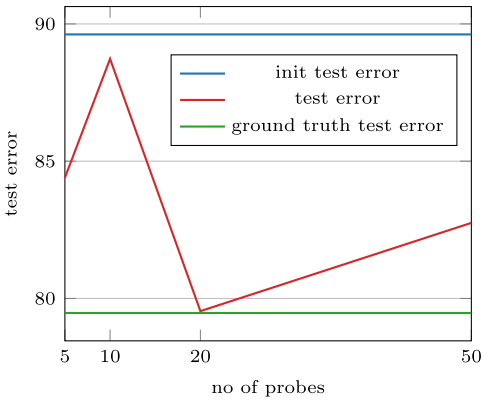


Fig. 5 Comparison of the test error of the Gaussian process with the initial hyperparameters (blue), optimized hyperparameters (red), and with the ground truth parameters (green), plotted against an increasing number of probe vectors for the trace estimation

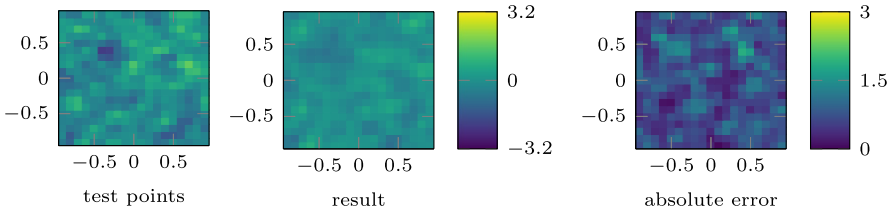


Fig. 6 Slice of the tensor for test points (right), optimized setup (middle) and error (right)

4.3 A high-dimensional case

We now illustrate that our method is capable of learning the parameters of the GP model also in high dimensions and for problems that are too large to be solved without the low-rank approximation. For this, we set $D = 7$ and repeat the experiment of Sect. 4.1: We again set $N = 21$ and generate the training labels y_{train} and test labels y_{test} by a random trigonometric function in 7 variables. We add the random noise with $\sigma = 0.01$ and convert the tensors into TT format with rounding error 10^{-8}. Finally, the tensor is normalized to Frobenius-norm 1.

We fit the parameters of a Gaussian process with a Kronecker-sum kernel of rank $R = 2$ and order $D = 7$. For this, we initialize our method with parameters $\sigma_{f(r)} = \frac{1}{2r} + \varepsilon, r = 1, 2$ and $\ell_r^{(d)} = \frac{1}{20r} + \varepsilon, r = 1, 2, d = 1, \ldots, 7$ where ε is random noise in $\mathcal{N}(0, 0.005)$ and different for each parameter. We test this for 40 probe vectors in the trace estimation and report only on the convergence results, since visualizations of the solution are not possible in high-dimensions.

We set the tolerance for the TT-Krylov method and the AMEn subsolver to 10^{-4}. Since for this large problem, the LBFGS solver in MATLAB's fminunc routine ran into local minima and computations took too long, we used a simple gradient descent method with Armijo line search and initial step size 0.1 instead. We report on the first 100 iterations.

In Fig. 7, we see that the method converges also for the high-dimensional problem. The gradient becomes small initially, but this corresponds only to a plateau of the loss function. It increases again before we see a somewhat linear convergence and a final decrease of the loss.

These 100 iteration steps only took 1692.95 seconds to compute. The system size is $11^7 = 19,487,171$. The storage of the kernel matrix of size $19,487,171 \times 19,487,171$ is $3 \cdot 10^{15}$ bytes. We can see that our method is feasible and accurate in unprecedented regimes.

Finally, we show that the approximation of the gradient using the TT-Krylov method is accurate. For this, we perform the same experiment again, but only with 2 probes (to speed up the computation). Here, we let the gradient descent method run for 10 iterations and then we calculate the gradient at the current iterate. In Fig. 8, we see the absolute approximation error of the cost function at that iterate compared to its first-order Taylor approximation using our computed gradient. One expects that this approximation error grows quadratically with the step size h. We can see that this is indeed the case for the relevant step sizes $h > 10^{-7}$.

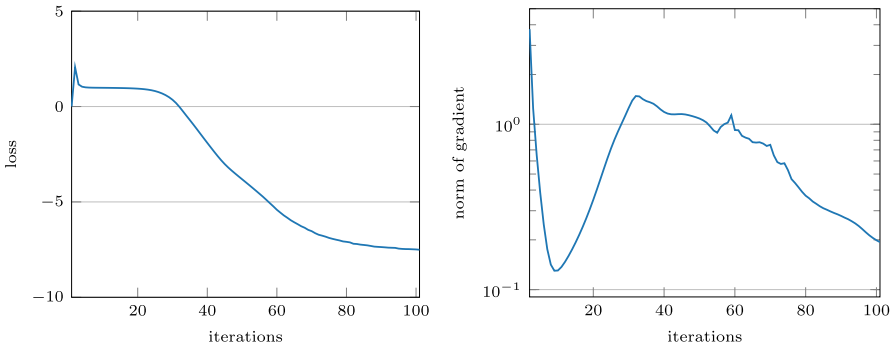


Fig. 7 Loss function value and norm of gradient for 100 iterations of the 7-dimensional problem

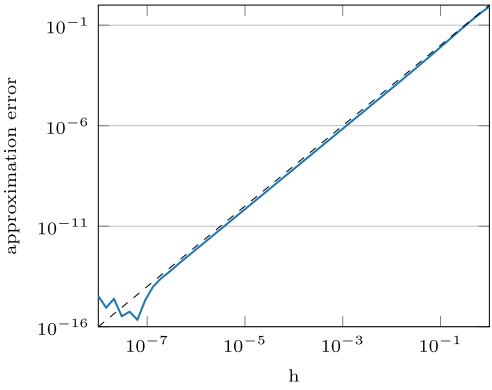


Fig. 8 Approximation error of the cost function f by its first order approximation using the computed gradient, for different step sizes h. The dashed line represents the expected approximation error of order 2

5 Conclusions

In this paper we proposed a numerical scheme for the optimization of the hyperparameters of a Gaussian process model for tensor-valued data that are approximated in the tensor train format. The method relies on the solution of an equation on the AMEn solver and we derived a matrix function analogue for the evaluation of both the objective function and its derivative. The estimation of the trace was done using a matrix logarithm and for its approximation Krylov methods and rational Krylov methods are proposed. For the derivative we require the evaluation of a block matrix function to get the Frechet derivative. We illustrate that the method indeed approximates the theoretical gradients well for the relevant range of parameters and show for two synthetic examples that we produce good approximations.

Acknowledgements The authors would like to thank David Bindel for the insightful discussions. M.S. acknowledges discussions with Kim Batselier on the use of tensor methods for Gaussian processes. The research of J.K. was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)—Project-ID 318763901 - SFB1294. M.P. was partially funded by the DFG - Projektnummer 448293816.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. Adaptive Computation and Machine Learning. The MIT Press, Cambridge (2006). https://doi.org/10.7551/mitpress/3206.001.0001
2. Williams, C.K.I., Rasmussen, C.E.: Gaussian processes for regression. In: Proceedings of the 8th International Conference on Neural Information Processing Systems, pp. 514–520 (1995). https://proceedings.neurips.cc/paper_files/paper/1995/file/7cce53cf90577442771720a370c3c723-Paper.pdf
3. Álvarez, M.A., Rosasco, L., Lawrence, N.D.: Kernels for vector-valued functions: a review. Found. Trends Mach. Learn. **4**(3), 195–266 (2012). https://doi.org/10.1561/2200000036
4. Kolda, T.G., Bader, B.W.: Tensor decompositions and applications. SIAM Rev. **51**(3), 455–500 (2009). https://doi.org/10.1137/07070111X
5. Oseledets, I.V.: Tensor-train decomposition. SIAM J. Sci. Comput. **33**(5), 2295–2317 (2011). https://doi.org/10.1137/090752286
6. Stoll, M.: A literature survey of matrix methods for data science. GAMM-Mitteilungen **43**(3), 202000013 (2020). https://doi.org/10.1002/gamm.202000013
7. Alaoui, A.E., Mahoney, M.W.: Fast randomized kernel ridge regression with statistical guarantees. In: Proceedings of the 28th International Conference on Neural Information Processing Systems—Volume 1, pp. 775–783 (2015). https://proceedings.neurips.cc/paper/2015/file/f3f27a324736617f20abbf2ffd806f6d-Paper.pdf
8. Cai, D., Nagy, J., Xi, Y.: Fast deterministic approximation of symmetric indefinite kernel matrices with high dimensional datasets. SIAM J. Matrix Anal. Appl. **43**(2), 1003–1028 (2022). https://doi.org/10.1137/21M1424627
9. Nakatsukasa, Y., Park, T.: Randomized low-rank approximation for symmetric indefinite matrices. SIAM J. Matrix Anal. Appl. **44**(3), 1370–1392 (2023). https://doi.org/10.1137/22M1538648
10. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: Proceedings of the 20th International Conference on Neural Information Processing Systems, pp. 1177–1184 (2007). https://proceedings.neurips.cc/paper_files/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf
11. Wilson, A., Nickisch, H.: Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In: Proceedings of the 32nd International Conference on International Conference on Machine Learning—Volume 37, pp. 1775–1784 (2015). https://dl.acm.org/doi/10.5555/3045118.3045307
12. Nestler, F., Stoll, M., Wagner, T.: Learning in high-dimensional feature spaces using ANOVA-based fast matrix-vector multiplication. Found. Data Sci. **4**(3), 423–440 (2022). https://doi.org/10.3934/fods.2022012
13. Iske, A., Borne, S.L., Wende, M.: Hierarchical matrix approximation for kernel-based scattered data interpolation. SIAM J. Sci. Comput. **39**(5), 2287–2316 (2017). https://doi.org/10.1137/16M1101167
14. Silverman, B.W.: Some aspects of the spline smoothing approach to non-parametric regression curve fitting. J. R. Stat. Soc. Ser. B Methodol. **47**(1), 1–21 (1985). https://doi.org/10.1111/j.2517-6161.1985.tb01327.x
15. Snelson, E., Ghahramani, Z.: Sparse Gaussian processes using pseudo-inputs. In: Proceedings of the 18th International Conference on Neural Information Processing Systems, pp. 1257–1264 (2005). https://proceedings.neurips.cc/paper_files/paper/2005/file/4491777b1aa8b5b32c2e8666dbe1a495-Paper.pdf

16. Dong, K., Eriksson, D., Nickisch, H., Bindel, D., Wilson, A.G.: Scalable log determinants for Gaussian process kernel learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 6330–6340 (2017). https://dl.acm.org/doi/pdf/10.5555/3295222.3295380
17. Gardner, J., Pleiss, G., Weinberger, K.Q., Bindel, D., Wilson, A.G.: GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 7587–7597 (2018). https://doi.org/10.5555/3327757.3327857
18. Skorski, M.: A modern analysis of Hutchinson's trace estimator. arXiv:2012.12895, https://doi.org/10.48550/arXiv.2012.12895 (2020)
19. Ubaru, S., Chen, J., Saad, Y.: Fast estimation of [CDATA[{{\rm tr}} (f(A))]]$\mathrm{tr}\left(f(A)\right)$ via stochastic Lanczos quadrature. SIAM J. Matrix Anal. Appl. **38**(4), 1075–1099 (2017). https://doi.org/10.1137/16M1104974
20. Meyer, R.A., Musco, C., Musco, C., Woodruff, D.P.: Hutch++: optimal stochastic trace estimation, pp. 142–155 (2021). https://doi.org/10.1137/1.9781611976496.16
21. Cortinovis, A., Kressner, D.: On randomized trace estimates for indefinite matrices with an application to determinants. Found. Comput. Math. **22**(3), 875–903 (2022). https://doi.org/10.1007/s10208-021-09525-9
22. Persson, D., Cortinovis, A., Kressner, D.: Improved variants of the Hutch++ algorithm for trace estimation. SIAM J. Matrix Anal. Appl. **43**(3), 1162–1185 (2022). https://doi.org/10.1137/21M1447623
23. Wenger, J., Pleiss, G., Hennig, P., Cunningham, J., Gardner, J.: Preconditioning for scalable Gaussian process hyperparameter optimization. In: International Conference on Machine Learning, pp. 23751–23780. PMLR (2022). https://proceedings.mlr.press/v162/wenger22a/wenger22a.pdf
24. Menzen, C., Memmel, E., Batselier, K., Kok, M.: Projecting basis functions with tensor networks for Gaussian process regression. IFAC-PapersOnLine **56**(2), 7288–7293 (2023). https://doi.org/10.1016/j.ifacol.2023.10.340
25. Yu, R., Li, G., Liu, Y.: Tensor regression meets Gaussian processes. In: International Conference on Artificial Intelligence and Statistics, pp. 482–490. PMLR (2018). http://proceedings.mlr.press/v84/yu18a/yu18a.pdf
26. Izmailov, P., Novikov, A., Kropotov, D.: Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition. In: International Conference on Artificial Intelligence and Statistics, pp. 726–735. PMLR (2018). http://proceedings.mlr.press/v84/izmailov18a/izmailov18a.pdf
27. Lindgren, G., Rootzen, H., Sandsten, M.: Stationary Stochastic Processes for Scientists and Engineers, 1st edn. Chapman and Hall/CRC, New York (2013). https://doi.org/10.1201/b15922
28. Saatçi, Y.: Scalable inference for structured Gaussian process models. PhD Thesis, University of Cambridge (2011). https://mlg.eng.cam.ac.uk/pub/pdf/Saa11.pdf
29. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003). https://doi.org/10.1137/1.9780898718003
30. Bijma, F., de Munck, J.C., Heethaar, R.M.: The spatiotemporal MEG covariance matrix modeled as a sum of Kronecker products. NeuroImage **27**(2), 402–415 (2005). https://doi.org/10.1016/j.neuroimage.2005.04.015
31. Hackbusch, W., Kühn, S.: A new scheme for the tensor representation. J. Fourier Anal. Appl. **15**(5), 706–722 (2009). https://doi.org/10.1007/s00041-009-9094-9
32. De Lathauwer, L., De Moor, B., Vandewalle, J.: A multilinear singular value decomposition. SIAM J. Matrix Anal. Appl. **21**(4), 1253–1278 (2000). https://doi.org/10.1137/S0895479896305696
33. Holtz, S., Rohwedder, T., Schneider, R.: On manifolds of tensors of fixed TT-rank. Numer. Math. **120**(4), 701–731 (2012). https://doi.org/10.1007/s00211-011-0419-7
34. Uschmajew, A., Vandereycken, B.: The geometry of algorithms using hierarchical tensors. Linear Algebra Appl. **439**(1), 133–166 (2013). https://doi.org/10.1016/j.laa.2013.03.016
35. Oseledets, I.V.: TT-Toolbox (TT=Tensor Train) Version 2.2.2. https://github.com/oseledets/TT-Toolbox. Accessed 14 Nov 2023
36. Eigel, M., Pfeffer, M., Schneider, R.: Adaptive stochastic Galerkin FEM with hierarchical tensor representations. Numer. Math. **136**(3), 765–803 (2017). https://doi.org/10.1007/s00211-016-0850-x
37. Eigel, M., Marschall, M., Pfeffer, M., Schneider, R.: Adaptive stochastic Galerkin FEM for lognormal coefficients in hierarchical tensor representations. Numer. Math. **145**(3), 655–692 (2020). https://doi.org/10.1007/s00211-020-01123-1

38. Szalay, S., Pfeffer, M., Murg, V., Barcza, G., Verstraete, F., Schneider, R., Legeza, O.: Tensor product methods and entanglement optimization for ab initio quantum chemistry. Int. J. Quantum Chem. **115**(19), 1342–1391 (2015). https://doi.org/10.1002/qua.24898
39. Dolgov, S.V., Savostyanov, D.V.: Alternating minimal energy methods for linear systems in higher dimensions. SIAM J. Sci. Comput. **36**(5), 2248–2271 (2014). https://doi.org/10.1137/140953289
40. Higham, N.J.: Functions of Matrices. SIAM, Philadelphia (2008). https://doi.org/10.1137/1.9780898717778
41. Güttel, S.: Rational Krylov approximation of matrix functions: numerical methods and optimal pole selection. GAMM-Mitteilungen **36**(1), 8–31 (2013). https://doi.org/10.1002/gamm.201310002
42. Golub, G.H., Meurant, G.: Matrices, Moments and Quadrature with Applications, vol. 30. Princeton University Press, Princeton (2010). https://doi.org/10.1515/9781400833887
43. Golub, G.H., Meurant, G.: Matrices, moments and quadrature. In: Numerical Analysis 1993, pp. 105–156. CRC Press, Boca Raton (2020). https://doi.org/10.1201/9781003062257
44. Golub, G.H., Meurant, G.: Matrices, moments and quadrature II; How to compute the norm of the error in iterative methods. BIT Numer. Math. **37**(3), 687–705 (1997). https://doi.org/10.1007/BF02510247
45. Güttel, S.: Rational Krylov methods for operator functions. PhD Thesis, Technische Universität Bergakademie Freiberg (2010). https://d-nb.info/1009391631/34
46. Ruhe, A.: Rational Krylov sequence methods for eigenvalue computation. Linear Algebra Appl. **58**, 391–405 (1984). https://doi.org/10.1016/0024-3795(84)90221-0
47. Strakoš, Z., Tichý, P.: On efficient numerical approximation of the bilinear form $c^*A^{-1}b$ [CDATA[c^{*}A^{-1}b]]. SIAM J. Sci. Comput. **33**(2), 565–587 (2011). https://doi.org/10.1137/090753723
48. Schweitzer, M.: A two-sided short-recurrence extended Krylov subspace method for nonsymmetric matrices and its relation to rational moment matching. Numer. Algorithms **76**(1), 1–31 (2017). https://doi.org/10.1007/s11075-016-0239-z
49. Haber, H.E.: Notes on the matrix exponential and logarithm. http://scipp.ucsc.edu/~haber/webpage/MatrixExpLog.pdf (2018). Accessed 14 Nov 2023
50. Al-Mohy, A.H., Higham, N.J.: Computing the action of the matrix exponential, with an application to exponential integrators. SIAM J. Sci. Comput. **33**(2), 488–511 (2011). https://doi.org/10.1137/100788860