Fast Laplace Approximation for Gaussian Processes with a Tensor Product Kernel

Perry Groot [a] Markus Peters [b] Tom Heskes [a] Wolfgang Ketter [b]

[a] *Radboud University Nijmegen*
[b] *Erasmus University Rotterdam*

Abstract

Gaussian processes provide a principled Bayesian framework, but direct implementations are restricted to small data sets due to the cubic time cost in the data size. In case the kernel function is expressible as a tensor product kernel and input data lies on a multidimensional grid it has been shown that the computational cost for Gaussian process regression can be reduced considerably. Tensor product kernels have mainly been used in regression with a Gaussian observation model since key steps in their algorithms do not easily translate to other tasks. In this paper we show how to obtain a scalable Gaussian process framework for gridded inputs and non-Gaussian observation models that factorize over cases. We empirically validate our approach on a binary classification problem and our results shows a major performance improvement in terms of run time.

1 Introduction

Gaussian processes (GPs) provide a rich, principled, and well-established Bayesian framework for tasks such as non-linear non-parametric regression and classification [8]. A direct implementation of GPs, however, limits their applicability to small data sets since the kernel matrix needs to be stored, costing $\mathcal{O}(N^2)$, and inverted, costing $\mathcal{O}(N^3)$, with N the number of data points. In recent years, several approaches have addressed this problem often selecting a subset of the training data (the active set) of size M reducing the computational complexity to $\mathcal{O}(M^2N)$ for $M \ll N$ [6, 10]. An alternative approach, is to exploit additional structure in the GP model to reduce the computational complexity.

The structural assumption that we exploit in this paper is the assumption of input data lying on a multidimensional grid.[1] Such data is often found in regression applications in time and space, for example, regular measurements over time and space from weather stations. This assumption allows the kernel function to be written as a tensor product kernel that allows for efficient computations using Kronecker products [5]. In recent years, the use of multidimensional grids has attracted increasing attention, and has been used in a variety of applications such as image reconstruction [3, 14], point process intensity estimation [2], network reconstruction [11], and density estimation [9].

The Kronecker product has, however, been limited to GP regression with spherical noise. The main reason for this is that a critical step is to compute the inverse $(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}$, which is not a Kronecker product even if $\boldsymbol{K}$ is. In this particular case it is still possible to efficiently compute the inverse using an eigen decomposition $\boldsymbol{K} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$ and the identity $(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{Q}^T\boldsymbol{y}$. This identity, however, no longer holds with non-spherical noise and how to efficiently obtain an eigen decomposition in this case is in fact an open problem [4].

In this paper we propose a scalable GP framework for gridded inputs and non-Gaussian observation models that factorize over cases. Section 2 briefly introduces GPs, the Laplace approximation, and the Kronecker product. Thereafter we describe how to efficiently combine tensor product kernel functions in the Laplace approximation. We provide derivations and pseudocode for obtaining the mode and for hyperparameter learning in Sections 3 and 4. Section 5 reviews the memory and run time requirements of the algorithms. In Section 6 we empirically validate the new methods and Section 7 gives conclusions.

[1] The grid does not necessarily have to be equispaced.

2 Background

We denote vectors $\boldsymbol{x}$ and matrices $\boldsymbol{K}$ with bold-face type and their components with regular type, i.e., x_i, K_{ij}. With $\boldsymbol{x}^T$ we denote the transpose of the vector $\boldsymbol{x}$. Let $\boldsymbol{x} \in \mathbb{R}^D$ be an input, $\boldsymbol{y} \in \mathcal{Y}$ an output. We denote with $\boldsymbol{X}, \boldsymbol{Y}$ the observed data. We denote with $|\boldsymbol{A}|$ the determinant of $\boldsymbol{A}$, $\boldsymbol{A} \otimes \boldsymbol{B}$ the Kronecker product, $\boldsymbol{A} \circ \boldsymbol{B}$ the Hadamard (element wise) product, and $\text{vec}(\boldsymbol{A})$ the vector obtained by stacking all the columns of matrix $\boldsymbol{A}$. Given a vector $\boldsymbol{a}$ and matrix $\boldsymbol{A}$, $\text{diag}(\boldsymbol{a})$ gives a diagonal matrix with $\boldsymbol{a}$ on its diagonal, and $\text{diag}(\boldsymbol{A})$ gives the vector of all diagonal elements of $\boldsymbol{A}$. Hyperparameters are denoted by $\boldsymbol{\theta} = \{\boldsymbol{\theta}^c, \boldsymbol{\theta}^l\}$, with $\boldsymbol{\theta}^c$ the kernel hyperparameters and $\boldsymbol{\theta}^l$ the likelihood hyperparameters.

2.1 Gaussian Processes

A Gaussian process (GP) is a collection of random variables $\{f(\boldsymbol{x}_i)\}_{i \in I}$ for some index set I, any finite number of which have a joint Gaussian distribution [8]. A GP $f \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K})$ results in a finite multivariate Gaussian distribution where each element of the covariance matrix $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is given by a kernel function $k(\cdot, \cdot)_{\boldsymbol{\theta}^c}$ with parameters $\boldsymbol{\theta}^c$. A GP effectively specifies a prior distribution over functions $f(\cdot)$ in a Bayesian framework in which the likelihood model $p(y_i|f(\boldsymbol{x}_i))$ is parameterized by $f(\cdot)$. Given a GP prior and likelihood, Bayes formula gives us the posterior distribution $p(\boldsymbol{f}|\boldsymbol{Y}, \boldsymbol{X}, \boldsymbol{\theta}) = p(\boldsymbol{Y}|\boldsymbol{f}, \boldsymbol{\theta}_l)p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta}_c)/p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\theta})$. In the simplest case, observations are assumed to be generated by an unknown function possibly corrupted with Gaussian noise, i.e., $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ which leads to a model in which inference is analytically tractable. In this case the predictive distribution for a test location $\boldsymbol{x}_*$ is given by $f_*|\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{x}_* \sim \mathcal{N}(\mu_*, v_*)$ with $\mu_* = k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{Y}$ and $v_* = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, \boldsymbol{X})(\boldsymbol{K} + \sigma^2\boldsymbol{I})^{-1}k(\boldsymbol{X}, \boldsymbol{x}_*)$. For non-Gaussian likelihoods one needs to resort to sampling methods or approximations.

2.2 Laplace Approximation

With a non-Gaussian likelihood inference in Gaussian process models leads to analytically intractable integrals when making predictions. For example, the distribution of a latent variable given a test case is given by

$$p(\boldsymbol{f}_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) = \int p(f_*|\boldsymbol{X}, \boldsymbol{x}_*, \boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y})d\boldsymbol{f} \tag{1}$$

The Laplace approximation [13] gives an analytic approximation to these integrals by approximating the true posterior distribution $p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y})$ with a Gaussian $q(\boldsymbol{f})$ centered on the mode of the posterior. The variance is obtained through a second-order Taylor expansion around the posterior mode $\hat{\mathbf{f}}$, which results in

$$q(\boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}|\hat{\mathbf{f}}, (\boldsymbol{K}^{-1} + \boldsymbol{W})^{-1}) \tag{2}$$

with $\hat{\mathbf{f}} = \arg\max_{\boldsymbol{f}} p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\theta})$ and $\boldsymbol{W} = -\nabla\nabla_{\boldsymbol{f}} \log p(\boldsymbol{y}|\boldsymbol{f})|_{\boldsymbol{f}=\hat{\mathbf{f}}}$ which is diagonal since the likelihood factorizes over cases.

2.3 Kronecker Products

In this paper we assume a tensor product kernel function, i.e., $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{d=1}^{D} k_d(\boldsymbol{x}_i^d, \boldsymbol{x}_j^d)$. Assuming a product kernel function together with input data on a multidimensional grid leads to a kernel matrix that decomposes into a Kronecker product of matrices of lower dimensions. The Kronecker product has a convenient algebra [5]:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{11}\boldsymbol{B} & \cdots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\boldsymbol{B} & \cdots & a_{mn}\boldsymbol{B} \end{bmatrix} \qquad \begin{aligned} (\boldsymbol{A} \otimes \boldsymbol{B})\text{vec}(\boldsymbol{X}) &= \text{vec}(\boldsymbol{B}\boldsymbol{X}\boldsymbol{A}^T) \\ \boldsymbol{A}\boldsymbol{B} \otimes \boldsymbol{C}\boldsymbol{D} &= (\boldsymbol{A} \otimes \boldsymbol{C})(\boldsymbol{B} \otimes \boldsymbol{D}) \\ (\boldsymbol{A} \otimes \boldsymbol{B})^{-1} &= \boldsymbol{A}^{-1} \otimes \boldsymbol{B}^{-1} \end{aligned} \tag{3}$$

Algorithm 1 Laplace mode finding

1: **function** LAPLACEMODE(covariance matrix $\boldsymbol{K}$, observations $\boldsymbol{y}$, likelihood function $p(\boldsymbol{y}|\boldsymbol{f})$)
2: $\quad\boldsymbol{f} \leftarrow \boldsymbol{0}$ ▷ Initialization
3: $\quad$**repeat** ▷ Newton iteration
4: $\quad\quad\boldsymbol{W} \leftarrow -\nabla\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
5: $\quad\quad\boldsymbol{b} \leftarrow \boldsymbol{W}\boldsymbol{f} + \nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
6: $\quad\quad$Iteratively solve: $(\boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}})\boldsymbol{v} = \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{b}$
7: $\quad\quad\boldsymbol{a} \leftarrow \boldsymbol{b} - \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{v}$ ▷ Eq. (4) using Conjugate Gradient
8: $\quad\quad\boldsymbol{f} \leftarrow \boldsymbol{K}\boldsymbol{a}$
9: $\quad$**until** convergence ▷ Objective $-\frac{1}{2}\boldsymbol{a}^T\boldsymbol{f} + \log p(\boldsymbol{y}|\boldsymbol{f})$
10: $\quad$**return** $\hat{\mathbf{f}} \leftarrow \boldsymbol{f}$ ▷ Mode
11: **end function**

The Kronecker product can be used to construct memory and computationally efficient algorithms. For example, for N data points a full $N \times N$ matrix needs $\mathcal{O}(N^2)$ memory storage and a matrix vector product is of order $\mathcal{O}(N^2)$ whereas for a Kronecker matrix in which each dimension d has cardinality N_d, this is $\mathcal{O}(\sum_{d=1}^{D} N_d^2)$ and $\mathcal{O}(N(\sum_{d=1}^{D} N_d))$, respectively [14]. Furthermore, the Cholesky and singular value decompositions both reduce from $\mathcal{O}(N^3)$ to $\mathcal{O}(\sum_{d=1}^{D} N_d^3)$ since these operations need only be applied to each component.

3 Maximum a Posteriori Estimation

The Laplace approximation approximates the posterior by a Gaussian centered on the mode of the posterior. The mode $\hat{\mathbf{f}}$ of the posterior is the maximizer of the log posterior, which can be found by setting the first derivative of the log posterior to zero and solving for $\hat{\mathbf{f}}$. We can find the maximum iteratively by using Newton's algorithm for which we need to compute in each iteration the following update step:

$$\begin{aligned}\boldsymbol{f}^{new} &= (\boldsymbol{K}^{-1} + \boldsymbol{W})^{-1}\boldsymbol{b} \\ &= \boldsymbol{K}(\boldsymbol{I} - \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}})^{-1}\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K})\boldsymbol{b}\end{aligned} \tag{4}$$

with $\boldsymbol{b} = \boldsymbol{W}\boldsymbol{f} + \nabla \log p(\boldsymbol{y}|\boldsymbol{f})$. In the last line, we used the matrix inversion lemma to rewrite the Newton iteration in a numerically advantageous way. Conjugate Gradient (CG) can now be used iteratively to solve the linear system $(\boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}})\boldsymbol{v} = \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{b}$, avoiding the need to directly compute any expensive matrix inverses. The mode is then found by setting $\boldsymbol{a} = \boldsymbol{b} - \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{v}$ and $\boldsymbol{f} = \boldsymbol{K}\boldsymbol{a}$. Pseudocode is given in Algorithm 1. In line 6 one can obtain an exact answer for a quadratic system if the iterated solver is run for N steps, however, reasonable results are usually already obtained in a small fraction of iterations. Since Kronecker matrix vector products are efficient this greatly reduces the cost of mode finding compared to a standard Cholesky implementation.

4 Model Selection

After obtaining the mode $\hat{\mathbf{f}}$ for a given hyperparameter $\boldsymbol{\theta}$, the next problem is to learn the value of $\boldsymbol{\theta}$. This can be done by minimizing the negative marginal likelihood

$$-\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) \approx \frac{1}{2}\hat{\mathbf{f}}^T\boldsymbol{K}^{-1}\hat{\mathbf{f}} - \log p(\boldsymbol{y}|\hat{\mathbf{f}}) + \frac{1}{2}\log|\boldsymbol{B}| \tag{5}$$

with $\boldsymbol{B} = \boldsymbol{I} + \boldsymbol{K}\boldsymbol{W}$. The first two terms are easily computed using the results of Newton's algorithm discussed in the previous paragraph, but the $\log|\boldsymbol{B}|$ and its gradients present difficulties. The $\boldsymbol{B}$ matrix is not a Kronecker product and a direct implementation of the log determinant has order $\mathcal{O}(N^3)$.

To reduce the computational complexity, we propose to approximate the prior covariance matrix

Algorithm 2 Laplace negative approximate marginal likelihood

function LAPLACEMARGLIK(covariance matrix $\boldsymbol{K}$, observations $\boldsymbol{y}$, likelihood function $p(\boldsymbol{y}|\boldsymbol{f})$)
  $[\hat{\mathbf{f}}, \boldsymbol{a}] \leftarrow$ Obtain from Algorithm 1
  for $d \leftarrow 1, \ldots, D$ **do**
    $\boldsymbol{Q}_d, \boldsymbol{S}_d \leftarrow \text{SCHUR}(\boldsymbol{K}_d)$ ▷ Eigen decomposition $\boldsymbol{K}_d = \boldsymbol{Q}_d \boldsymbol{S}_d \boldsymbol{Q}_d^T$
    $\boldsymbol{Q}_d, \boldsymbol{S}_d \leftarrow \text{REDUCERANK}(\boldsymbol{Q}_d, \boldsymbol{S}_d)$ ▷ Select largest eigenvalues
  end for
  $\boldsymbol{W} \leftarrow -\nabla\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$
  $\boldsymbol{\Lambda}_1 \leftarrow \text{diag}(\text{diag}(\boldsymbol{K}) - (\boldsymbol{Q} \circ \boldsymbol{Q}) \cdot \text{diag}(\boldsymbol{S}))$ ▷ Eq. (6)
  $\boldsymbol{\Lambda}_2 \leftarrow \boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}} \boldsymbol{\Lambda}_1 \boldsymbol{W}^{\frac{1}{2}}$
  $\boldsymbol{\Lambda}_3 \leftarrow \boldsymbol{W}^{\frac{1}{2}} \boldsymbol{\Lambda}_2^{-1} \boldsymbol{W}^{\frac{1}{2}}$
  $\boldsymbol{L} \leftarrow \text{CHOLESKY}(\boldsymbol{S}^{-1} + \boldsymbol{Q}^T \boldsymbol{\Lambda}_3 \boldsymbol{Q})$ ▷ Eq. (9)
  $logdetB \leftarrow \sum_i \log \Lambda_{2ii} + \sum_i \log S_{ii} + 2 * \sum_i \log L_{ii}$
  return $-\log q(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) \leftarrow \frac{1}{2} \boldsymbol{a}^T \hat{\mathbf{f}} - \log p(\boldsymbol{y}|\hat{\mathbf{f}}) + \frac{1}{2} logdetB$ ▷ neg. approx. log. marg. lik.
end function

using a reduced-rank approximation:

$$\boldsymbol{K} \approx \boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T + \boldsymbol{\Lambda}_1, \text{ with } \boldsymbol{Q} = \bigotimes_{d=1}^{D} \boldsymbol{Q}_d, \boldsymbol{S} = \bigotimes_{d=1}^{D} \boldsymbol{S}_d, \boldsymbol{\Lambda}_1 = \text{diag}(\text{diag}(\boldsymbol{K}) - \text{diag}(\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T)) \tag{6}$$

and where $\boldsymbol{K}$ is an $N \times N$ matrix, $\boldsymbol{Q}$ is an $N \times R$ matrix, and $\boldsymbol{S}$ is an $R \times R$ diagonal matrix with $R \ll N$. The $\boldsymbol{\Lambda}_1$ matrix is a diagonal matrix that preserves the exact full-rank diagonal to obtain a better approximation without raising the computational costs. The sum of a diagonal matrix and a low rank matrix allows for efficient computations using the Woodbury formula and matrix determinant lemma

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{W}\boldsymbol{V})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{W}^{-1} + \boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1} \tag{7}$$

$$|\boldsymbol{A} + \boldsymbol{U}\boldsymbol{W}\boldsymbol{V}^T| = |\boldsymbol{W}^{-1} + \boldsymbol{V}^T\boldsymbol{A}^{-1}\boldsymbol{U}||\boldsymbol{W}||\boldsymbol{A}| \tag{8}$$

and has been used widely before in statistics and machine learning [1, 7]. The *key insight* here is that the Kronecker product allows the decomposition $\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T$ to be *efficiently* obtained in $\mathcal{O}(\sum_{d=1}^{D} N_d^3)$.

4.1 Evaluating the Marginal Likelihood

With the reduced-rank approximation the log determinant in the negative marginal likelihood can be evaluated more efficiently using the relation:

$$\begin{aligned} |\boldsymbol{B}| &= |\boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}}| \approx |\boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_1\boldsymbol{W}^{\frac{1}{2}} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}}| = |\boldsymbol{\Lambda}_2 + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}}| \\ &= |\boldsymbol{\Lambda}_2||\boldsymbol{S}||\boldsymbol{S}^{-1} + \boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_2^{-1}\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}| = |\boldsymbol{\Lambda}_2||\boldsymbol{S}||\boldsymbol{S}^{-1} + \boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q}| \end{aligned} \tag{9}$$

with $\boldsymbol{\Lambda}_2 = \boldsymbol{I} + \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_1\boldsymbol{W}^{\frac{1}{2}}$ and $\boldsymbol{\Lambda}_3 = \boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_2^{-1}\boldsymbol{W}^{\frac{1}{2}}$. The final formula can be evaluated efficiently since it consists of two diagonal matrices and one low-rank matrix. To evaluate the last term we use the Cholesky decomposition $\boldsymbol{L} = \text{CHOLESKY}(\boldsymbol{S}^{-1} + \boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})$, which can be reused in the gradients. Pseudocode is given in Algorithm 2.

4.2 Gradients

We can optimize the marginal likelihood with respect to the hyperparameters $\boldsymbol{\theta}$. For this we need the partial derivatives $\partial q(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})/\theta_j$. Since $\hat{\mathbf{f}}$ and $\boldsymbol{W}$ are implicit functions of $\boldsymbol{\theta}$ we need to take care of both the explicit and implicit derivatives [8].

$$\frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})}{\partial \theta_j} = \left.\frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})}{\partial \theta_j}\right|_{\text{explicit}} + \sum_{i=1}^{N} \frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta})}{\partial \hat{f}_i} \frac{\partial \hat{f}_i}{\partial \theta_j} \tag{10}$$

4.2.1 Explicit Derivatives – Kernel Hyperparameters

The explicit derivatives are given by:

$$\left.\frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})}{\partial \theta_j^c}\right|_{\text{explicit}} = \frac{1}{2}\hat{\mathbf{f}}^T \boldsymbol{K}^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\boldsymbol{K}^{-1}\hat{\mathbf{f}} - \frac{1}{2}\text{tr}\left((\boldsymbol{W}^{-1}+\boldsymbol{K})^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\right). \tag{11}$$

The first term is easily computed using Kronecker matrix-vector products, but we approximate the second term using the low-rank decomposition. Since

$$\begin{aligned}
(\boldsymbol{W}^{-1}+\boldsymbol{K})^{-1} &= \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{I}+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}})^{-1}\boldsymbol{W}^{\frac{1}{2}} \\
&\approx \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{I}+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_1\boldsymbol{W}^{\frac{1}{2}}+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}})^{-1}\boldsymbol{W}^{\frac{1}{2}} \\
&= \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{\Lambda}_2+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}\boldsymbol{S}\boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}})^{-1}\boldsymbol{W}^{\frac{1}{2}} \\
&= \boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{\Lambda}_2^{-1}-\boldsymbol{\Lambda}_2^{-1}\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{Q}(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3^{-1}\boldsymbol{Q})^{-1}\boldsymbol{Q}^T\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{\Lambda}_2^{-1})\boldsymbol{W}^{\frac{1}{2}} \\
&= \boldsymbol{\Lambda}_3-\boldsymbol{\Lambda}_3\boldsymbol{Q}(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})^{-1}\boldsymbol{Q}^T\boldsymbol{\Lambda}_3
\end{aligned} \tag{12}$$

we can rewrite the trace term as follows

$$\begin{aligned}
\text{tr}\left((\boldsymbol{W}^{-1}+\boldsymbol{K})^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\right) &= \text{tr}\left(\boldsymbol{\Lambda}_3\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\right) - \text{tr}\left(\boldsymbol{\Lambda}_3\boldsymbol{Q}\left(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q}\right)^{-1}\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\right) \\
&= \text{tr}\left(\boldsymbol{\Lambda}_3\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\right) - \text{tr}\left(\boldsymbol{V}_1^T\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\boldsymbol{V}_1\right)
\end{aligned} \tag{13}$$

with $\boldsymbol{V}_1 = \boldsymbol{\Lambda}_3\boldsymbol{Q}\text{CHOLESKY}\left[(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})^{-1}\right]$. In this last line we also used the cyclic property of the trace operator $\text{tr}(\boldsymbol{ABC}) = \text{tr}(\boldsymbol{BCA})$ to obtain a trace of a low-rank $R \times R$ matrix.

4.2.2 Implicit Derivatives – Kernel Hyperparameters

To evaluate the implicit derivatives we need the following expression:

$$\frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})}{\partial \hat{f}_i} = -\frac{1}{2}\left[(\boldsymbol{K}^{-1}+\boldsymbol{W})^{-1}\right]_{ii}\frac{\partial^3}{\partial f_i^3}\log p(\boldsymbol{y}|\hat{\mathbf{f}}) \tag{14}$$

The troublesome part can be rewritten using the low-rank decomposition as follows

$$\begin{aligned}
(\boldsymbol{K}^{-1}+\boldsymbol{W})^{-1} &= (\boldsymbol{K}^{-1}+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{I}\boldsymbol{W}^{\frac{1}{2}})^{-1} \\
&= \boldsymbol{K}-\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}}(\boldsymbol{I}+\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K}\boldsymbol{W}^{\frac{1}{2}})^{-1}\boldsymbol{W}^{\frac{1}{2}}\boldsymbol{K} \\
&= \boldsymbol{K}-\boldsymbol{K}(\boldsymbol{W}^{-1}+\boldsymbol{K})^{-1}\boldsymbol{K} \\
&\approx \boldsymbol{K}-\boldsymbol{K}\boldsymbol{\Lambda}_3\boldsymbol{K}+\boldsymbol{K}\boldsymbol{\Lambda}_3\boldsymbol{Q}(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})^{-1}\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{K} \\
&= \boldsymbol{K}-\boldsymbol{K}\boldsymbol{\Lambda}_3\boldsymbol{K}+\boldsymbol{V}_2\boldsymbol{V}_2^T
\end{aligned} \tag{15}$$

with $\boldsymbol{V}_2 = \boldsymbol{K}\boldsymbol{\Lambda}_3\boldsymbol{Q}\text{CHOLESKY}\left[(\boldsymbol{S}^{-1}+\boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})^{-1}\right] = \boldsymbol{K}\boldsymbol{V}_1$. The diagonal $\text{diag}[(\boldsymbol{K}^{-1}+\boldsymbol{W})^{-1}]$ can be easily obtained using Eq. (15). The remaining term

$$\frac{\partial \hat{\mathbf{f}}}{\partial \theta_j^c} = (\boldsymbol{I}+\boldsymbol{K}\boldsymbol{W})^{-1}\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\nabla \log p(\boldsymbol{y}|\hat{\mathbf{f}}) \tag{16}$$

can be solved using conjugate gradient avoiding the need to directly compute the expensive matrix inverse.

4.2.3 Explicit Derivatives – Likelihood Hyperparameters

The explicit derivatives of the likelihood are given by:

$$\left.\frac{\partial \log q(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta})}{\partial \theta_j^l}\right|_{\text{explicit}} = \frac{\partial}{\partial \theta_j^l}\log p(\boldsymbol{y}|\boldsymbol{f}) + \frac{1}{2}\left[(\boldsymbol{K}^{-1}+\boldsymbol{W})^{-1}\right]_{ii}\frac{\partial}{\partial \theta_j^l}\nabla^2 \log p(\boldsymbol{y}|\boldsymbol{f}) \tag{17}$$

in which the second term can be approximated as outlined in Eq. (15).

Algorithm 3 Laplace gradients

1: **function** LAPLACEGRADIENT(covariance matrix $\boldsymbol{K}$, observations $\boldsymbol{y}$, likelihood function $p(\boldsymbol{y}|\boldsymbol{f})$)
2: $[\hat{\mathbf{f}}, \boldsymbol{Q}, \boldsymbol{L}, \boldsymbol{\Lambda}_3] \leftarrow$ Obtain from Algorithm 1 and Algorithm 2
3: $\boldsymbol{V}_1 \leftarrow \boldsymbol{\Lambda}_3\boldsymbol{Q}/\boldsymbol{L}$ ▷ $\boldsymbol{V}_1 = \boldsymbol{\Lambda}_3\boldsymbol{Q}\text{CHOLESKY}\left[(\boldsymbol{S}^{-1} + \boldsymbol{Q}^T\boldsymbol{\Lambda}_3\boldsymbol{Q})^{-1}\right]$
4: $\boldsymbol{V}_2 \leftarrow \boldsymbol{K}\boldsymbol{V}_1$
5: $\boldsymbol{C} \leftarrow \text{diag}(\boldsymbol{K}) - (\boldsymbol{K} \circ \boldsymbol{K}) \cdot \text{diag}(\boldsymbol{\Lambda}_3) + \text{SUMROWS}(\boldsymbol{V}_2 \circ \boldsymbol{V}_2)$ ▷ Eq. (15)
6: $s_2 \leftarrow \frac{1}{2}\boldsymbol{C} \cdot \nabla^3 \log p(\boldsymbol{y}|\boldsymbol{f})$ ▷ Eq. (14)
7: **for** $j \leftarrow 1, \ldots, \dim(\boldsymbol{\theta}^c)$ **do** ▷ loop over kernel hyperparameters
8: $\boldsymbol{s}_1 \leftarrow \frac{1}{2}\boldsymbol{a}^T \frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\boldsymbol{a} - \frac{1}{2}\text{diag}(\boldsymbol{\Lambda}_3)^T \text{diag}(\frac{\partial \boldsymbol{K}}{\partial \theta_j^c}) + \frac{1}{2}\text{tr}(\boldsymbol{V}_1^T \frac{\partial \boldsymbol{K}}{\partial \theta_j^c}\boldsymbol{V}_1)$ ▷ Eqs. (11–13)
9: Iteratively solve: $(\boldsymbol{I} + \boldsymbol{K}\boldsymbol{W})\boldsymbol{s}_3 = \boldsymbol{K}\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$ ▷ Eq. (16) using conjugate gradient
10: $\nabla_j - \log p(\boldsymbol{y}|\boldsymbol{f}) \leftarrow -\boldsymbol{s}_1 - \boldsymbol{s}_2^T\boldsymbol{s}_3$ ▷ Gradient $-\frac{\partial}{\partial \theta_j^c}\log p(\boldsymbol{y}|\boldsymbol{f})$
11: **end for**
12: **for** $j \leftarrow 1, \ldots, \dim(\boldsymbol{\theta}^l)$ **do** ▷ loop over likelihood hyperparameters
13: $\boldsymbol{s}_1 \leftarrow \frac{\partial}{\partial \theta_j^l}\log p(\boldsymbol{y}|\boldsymbol{f}) + \frac{1}{2}\boldsymbol{C}\frac{\partial}{\partial \theta_j^l}\nabla^2 \log p(\boldsymbol{y}|\boldsymbol{f})$ ▷ Eq. (17)
14: Iteratively solve: $(\boldsymbol{I} + \boldsymbol{K}\boldsymbol{W})\boldsymbol{s}_3 = \boldsymbol{K}\frac{\partial}{\partial \theta_j^l}\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$ ▷ Eq. (18) using conjugate gradient
15: $-\nabla_j \log p(\boldsymbol{y}|\boldsymbol{f}) \leftarrow -\boldsymbol{s}_1 - \boldsymbol{s}_2^T\boldsymbol{s}_3$ ▷ Gradient $-\frac{\partial}{\partial \theta_j^l}\log p(\boldsymbol{y}|\boldsymbol{f})$
16: **end for**
17: **return** $-\nabla \log p(\boldsymbol{y}|\boldsymbol{f})$ ▷ Gradient $-\frac{\partial}{\partial \boldsymbol{\theta}}\log p(\boldsymbol{y}|\boldsymbol{f})$
18: **end function**

4.2.4 Implicit Derivatives – Likelihood Hyperparameters

The implicit derivatives of the likelihood (cf. Eq. (10)) are computed by combining Eq. (14) with

$$\frac{\partial \hat{\mathbf{f}}}{\partial \theta_j^l} = (\boldsymbol{I} + \boldsymbol{K}\boldsymbol{W})^{-1}\boldsymbol{K}\frac{\partial}{\partial \theta_j^l}\nabla \log p(\boldsymbol{y}|\boldsymbol{f}) \tag{18}$$

of which the latter is solved using conjugate gradient.

5 Memory and Run Time Complexity of the Algorithms

Algorithms 1, 2, and 3 are both *memory* and *run time* efficient. Care is taken to never evaluate a full $N \times N$ matrix, which may not fit into memory. The matrices $\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2, \boldsymbol{\Lambda}_3$ are diagonal each requiring $\mathcal{O}(N)$ storage, $\boldsymbol{K}$ is a Kronecker matrix requiring $\mathcal{O}(\sum_{d=1}^D N_d^2)$ storage, $\boldsymbol{S} = \otimes \boldsymbol{S}_d$ requires at most $\mathcal{O}(R)$ storage, $\boldsymbol{L}$ requires $\mathcal{O}(R^2)$ storage, and $\boldsymbol{Q} = \otimes \boldsymbol{Q}_d, \boldsymbol{V}_1, \boldsymbol{V}_2$ require (at most) $\mathcal{O}(RN)$ storage. Because all matrix and vector operations operate on small, i.e., $R \times N$, diagonal, or Kronecker matrices, instead of full $N \times N$ matrices, an increase in efficiency is obtained when N becomes large. Note that in Algorithm 2, line 8 and Algorithm 3, line 5 we made use of the identity

$$\text{diag}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{C}^T) = (\boldsymbol{A} \circ \boldsymbol{C}) \cdot \text{diag}(\boldsymbol{B}) \tag{19}$$

which holds when $\boldsymbol{B}$ is a diagonal matrix to speed up computations. For a Kronecker matrix $\boldsymbol{K}$, the Hadamard product $\boldsymbol{K} \circ \boldsymbol{K}$ can be evaluated efficiently and is again a Kronecker matrix.

6 Experiments

In order to compare the run time complexity of a standard Gaussian process with a Gaussian process having a Kronecker product kernel we implemented the algorithms in the publicly available Gaussian process toolbox GPstuff [12] and ran the algorithms on a synthetic dataset. As input we use gridded data on the D-dimensional hypercube $\mathcal{X} = \prod_{d=1}^D [0, 1]$. We consider the binary classification task of a datapoint belonging to the D-dimensional unit ball in $\mathcal{X}$.

For the likelihood model we use a probit model $p(y_i|f_i) = \Phi(y_i f_i)$ with $y_i \in \{-1, 1\}$. We generated inputs with $D = 2$ using varying grid sizes and ran both the standard Laplace approximation and the

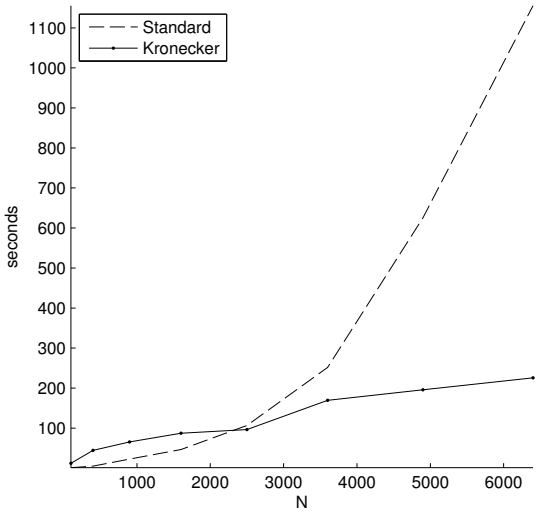

Figure 1: Run time of the standard Laplace algorithm versus the Kronecker Laplace algorithm on a 2D classification problem with a varying number of input data points.

Kronecker Laplace approximation and measured their run-time averaged over five runs. The results are shown in Figure 1. For the low-rank approximation we only selected the eigenvalues higher than $1\text{E-}6$, but if necessary only select from the highest eigen values such that the rank is below 10% of the number of input data points. The tolerance in the conjugate gradient iterations was set to $1\text{E-}10$. Clearly, the increase in computational complexity with increasing data size is much lower for the Kronecker Laplace approximation than for the standard Laplace approximation.

7 Conclusions

In this paper we have presented a scalable Gaussian process framework for gridded inputs and non-Gaussian observation models that factorize over cases. We used the Laplace approximation to perform approximate inference and showed it can efficiently be combined with tensor product kernels using fast matrix-vector operations and an efficiently obtained low-rank approximation of the kernel matrix. We provided derivations and pseudocode of the algorithms and empirically validated our approach on a binary classification problem showing a major performance improvement in terms of run time.

Directions for further work include among others handling missing observations and a more detailed analysis of the computational complexity. The standard Laplace approximation is quite straightforward, but the Kronecker Laplace approximation provides more options that each influence the computational complexity and quality of the algorithms. These include, among others, the rank of the low-rank approximation, the tolerances used in the Conjugate Gradients Iterated solvers, and the number of hyper-parameters.

References

[1] N. Cressie and G. Johannesson. Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society*, 70:209–226, 2008.

[2] J. P. Cunningham, K. V. Shenoy, and M. Sahani. Fast Gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pages 192–199, 2008.

[3] E. Gilboa, Y. Saatci, and J. P. Cunningham. Scaling multidimensional Gaussian processes using projected additive approximations. In *International Conference on Machine Learning*, 2013.

[4] A. Knutson and T. Tao. Honeycombs and sums of Hermitian matrices. *Notices Amer. Math. Soc*, pages 175–186, 2000.

[5] C. F. van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1–2):85–100, 2000.

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

[7] J. Quiñonero-Candela, C.E. Rasmussen, and C.K.I. Williams. Approximation methods for Gaussian process regression. *Advances in Neural Information Processing Systems*, pages 203–223, 2007.

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

[9] J. Riihimäki and A. Vehtari. Laplace approximation for logistic Gaussian process density estimation. *Bayesian Analysis*, in press, 2014.

[10] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 2006.

[11] O. Stegle, C. Lippert, J. M. Mooij, N. D. Lawrence, and K. Borgwardt. Efficient inference in matrix-variate Gaussian models with iid observation noise. In *Advances in Neural Information Processing Systems 24*, pages 630–638, 2011.

[12] J. Vanhatalo, J. Riihimäki, J. Hartikainen, P. Jylänki, V. Tolvanen, and A. Vehtari. GPstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, 14(Apr):1175–1179, 2013.

[13] C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

[14] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham. GPatt: Fast multidimensional pattern extrapolation with Gaussian processes. *arXiv, 2013.*