Evaluation of Massively Scalable Gaussian Processes

HANNA HULTIN

Master in Applied and Computational Mathematics
Date: June 13, 2017
Supervisor at Department of Mathematics: Henrik Hult
Supervisor at Robotics, Perception and Learning: Hedvig Kjellström
Supervisor at SICS: Theodoros Vasiloudis
Examiner: Boualem Djehiche
Swedish title: Evaluering av massivt skalbara Gaussiska processer
School of Engineering Sciences

Abstract

Gaussian process methods are flexible non-parametric Bayesian methods used for regression and classification. They allow for explicit handling of uncertainty and are able to learn complex structures in the data. Their main limitation is their scaling characteristics: for n training points the complexity is $\mathcal{O}(n^3)$ for training and $\mathcal{O}(n^2)$ for prediction per test data point. This makes full Gaussian process methods prohibitive to use on training sets larger than a few thousand data points.

There has been recent research on approximation methods to make Gaussian processes scalable without severely affecting the performance. Some of these new approximation techniques are still not fully investigated and in a practical situation it is hard to know which method to choose. This thesis examines and evaluates scalable GP methods, especially focusing on the framework Massively Scalable Gaussian Processes introduced by Wilson et al. in 2016, which reduces the training complexity to nearly $\mathcal{O}(n)$ and the prediction complexity to $\mathcal{O}(1)$. The framework involves inducing point methods, local covariance function interpolation, exploitations of structured matrices and projections to low-dimensional spaces. The properties of the different approximations are studied and the possibilities of making improvements are discussed.

Sammanfattning

Gaussiska processmetoder är flexibla icke-parametriska Bayesianska metoder som används för regression och klassificering. De tillåter explicit hantering av osäkerhet och kan lära sig komplexa strukturer i data. Den största begränsningen är deras skalningsegenskaper: för n träningspunkter är komplexiteten $\mathcal{O}(n^3)$ för träning och $\mathcal{O}(n^2)$ för prediktion per ny datapunkt. Detta gör att kompletta Gaussiska processer är för krävande för att använda på träningsdata större än några tusen datapunkter.

Det har nyligen forskats på approximationsmetoder för att göra Gaussiska processer skalbara utan att påverka prestandan allvarligt. Några av dessa nya approximationstekniker är fortfarande inte fullkomligt undersökta och i en praktisk situation är det svårt att veta vilken metod man ska använda. Denna uppsats undersöker och utvärderar skalbara GP-metoder, särskilt med fokus på ramverket Massivt Skalbara Gaussiska Processer introducerat av Wilson et al. 2016, vilket minskar träningskomplexiteten till $\mathcal{O}(n)$ och prediktionskomplexiteten till $\mathcal{O}(1)$. Ramverket innehåller inducerande punkt-metoder, lokal kärninterpolering, utnyttjande av strukturerade matriser och projiceringar till lågdimensionella rum. Egenskaperna hos de olika approximationerna studeras och möjligheterna att göra förbättringar diskuteras.

Contents

Chapter 1

Introduction

Gaussian processes (GPs) are stochastic processes that have been studied and used by mathematicians for centuries [1]. Today they are well established models for spatial and temporal problems. Examples of commonly used GPs are Brownian motion, Langevin processes and Wiener processes. Kalman filters, often used to model speech waveforms, are also GP models [2].

For predictions, GPs were first used in the 1940's for time series predictions and since the 1970's they have been widely used for geostatistical and meteorological predictions. GP models were then eventually used for the general multivariate input regression problem as well [1].

GPs can be interpreted as distributions over functions where the distribution is fully specified by the mean and covariance functions of the process. With this interpretation, they can be used in Bayesian non-parametric models for regression, classification and other machine learning tasks. GP models are Bayesian probabilistic models and as such they allow for explicit handling of the uncertainty present in all real-world scenarios and give full probabilistic predictions. With GP regression there is no need to choose basis functions as is usually the case for parametric approaches, instead the complexity of the model grows automatically with larger datasets which is the typical behaviour of non-parametric models.

The main limitation of using GPs for regression is their scaling characteristics. With n training points the computational complexity is $\mathcal{O}(n^3)$ for inference and learning and the memory complexity is $\mathcal{O}(n^2)$. The computational bottleneck for inference is the inversion of the $n \times n$ covariance function and for learning the logarithm of the determinant of the same matrix has to be computed as well. These computations are usually made by doing a Cholesky decomposition which has the computational complexity $\mathcal{O}(n^3)$. Further more, the computational complexity for making predictions is $\mathcal{O}(n^2)$ per prediction point. These high complexities rule out the possibility to use full GPs in scenarios with more than a few thousand data points.

A large body of research has been dedicated to making GPs scalable, with inducing point methods [3], which approximate the true covariance matrix with a lower rank matrix, being the most popular. Typically these methods attain $\mathcal{O}(m^2 n)$ computational complexity and $\mathcal{O}(mn)$ space complexity for m inducing points, where $m \ll n$. These methods are usually designed to work with any covariance function and any training data, but they trade accuracy for computational complexity. Alternative approaches have been developed that focus on exploiting structured matrices, which allow for faster exact infer-

ence at the cost of limited flexibility for the training points and the covariance function [4].

This thesis evaluates a new framework for scalable GP regression, Massively Scalable Gaussian Processes (MSGP) introduced by Wilson et al. in 2016 [5]. This framework uses a combination of inducing points, local interpolation and structured GP approaches to achieve highly scalable, but also flexible learning of GPs.

The main idea of MSGP is to place the inducing points on a regular grid and approximate the covariance matrix of the training points by using local interpolation on the covariance function of the inducing points. The covariance function is usually a smooth well-behaved function, which often results in the local interpolation having a high accuracy. Since the inducing points are placed on a regular grid, the covariance matrix of the inducing points is a structured matrix which can exploit faster matrix operations. Therefore, the number of inducing points are no longer restricted to be much less than the number of training points, as in the other inducing points method, which allows for near-exact accuracy and expressive covariance function learning.

1.1 Outline

Chapter 2 gives an overview of GPs and how they can be used for regression and classification as well as a review of methods for making GPs scalable. Thereafter the framework MSGP is presented and explained in more detail in Chapter 3. In the following chapter the experiments with the aim of evaluating the different parts of the framework are presented and the results are commented. Lastly there is a discussion in Chapter 5.

Chapter 2

Gaussian Processes for Regression

2.1 Gaussian Processes from a Mathematical Perspective

Three main references have been used for the mathematical background on GPs: Gaussian Hilbert Spaces by Svante Janson [6], Random Fields and Geometry by Robert J. Adler and Jonathan E. Taylor [7] and The Geometry of Random Fields by Robert J. Adler [8].

2.1.1 Gaussian Random Variables

A real-valued random variable X is called Gaussian if it has the probability density function:

$$\varphi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(x-m)^2}{2\sigma^2}\right), \quad x \in \mathbb{R} \tag{2.1}$$

for some $m \in \mathbb{R}$ and $\sigma > 0$. The random variable X is then a Gaussian random variable with mean m and standard deviation σ. This is abbreviated as $X \sim \mathcal{N}(m, \sigma^2)$. If $m = 0$ and $\sigma = 1$ the random variable has a standard normal distribution and in general if $m = 0$ and $\sigma > 0$ the random variable is called centered [7].

For the multivariate case, an $\mathbb{R}^D$-valued random vector X is a multivariate Gaussian random variable if any linear combination of its elements $\sum_{i=1}^{D} a_i X_i$, $a \in \mathbb{R}^D$ is a Gaussian random variable too. Then there is a mean vector $\mathbf{m} \in \mathbb{R}^D$ such that $m_i = \mathbb{E}[X_i]$ and a symmetric positive semi-definite covariance matrix $K \in \mathbb{R}^{D \times D}$ with elements $K_{ij} = \mathrm{Cov}(X_i, X_j) = \mathbb{E}[(X_i - m_i)(X_j - m_j)]$. The probability density function of the multivariate Gaussian X is then given by:

$$\varphi(\mathbf{x}) = (2\pi)^{-D/2} |K|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^\top K^{-1}(\mathbf{x}-\mathbf{m})\right), \quad \mathbf{x} \in \mathbb{R}^D \tag{2.2}$$

This is written as $X \sim \mathcal{N}(\mathbf{m}, K)$ or $X \sim \mathcal{N}_D(\mathbf{m}, K)$ with the dimension explicitly written.

If is easily shown using simple algebra and the probability density function of a Gaussian random variable $X \sim \mathcal{N}_D(\mathbf{m}, K)$ that the distribution of AX is also a Gaussian distribution for any matrix $A \in \mathbb{R}^{d \times D}$. Specifically $AX \sim \mathcal{N}_d(A\mathbf{m}, AKA^\top)$.

This can be used to compute the conditional distributions by making special choices

of A. Consider the partitions:

$$X = \begin{pmatrix} X^1 \\ X^2 \end{pmatrix} = \begin{pmatrix} (X_1, ..., X_d)^\top \\ (X_{d+1}, ..., X_D)^\top \end{pmatrix} \tag{2.3}$$

$$\mathbf{m} = \begin{pmatrix} \mathbf{m}^1 \\ \mathbf{m}^2 \end{pmatrix} = \begin{pmatrix} (m_1, ..., m_d)^\top \\ (m_{d+1}, ..., m_D)^\top \end{pmatrix} \tag{2.4}$$

$$K = \begin{pmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{pmatrix} \tag{2.5}$$

where $K_{11} \in \mathbb{R}^{d \times d}$. Then X^i is $\mathcal{N}(\mathbf{m}_i, K_{ii})$ for $i = 1, 2$ and the conditional distribution of X^i given X^j is $\mathcal{N}(\mathbf{m}_{i|j}, K_{i|j})$ where:

$$\mathbf{m}_{i|j} = \mathbf{m}_i + K_{ij} K_{jj}^{-1} (X^j - \mathbf{m}_j) \tag{2.6}$$

$$K_{i|j} = K_{ii} - K_{ij} K_{jj}^{-1} K_{ji} \tag{2.7}$$

2.1.2 Gaussian Processes

Given a probability space $(\Omega, \mathcal{A}, P)$ and an arbitrary index set T, a real stochastic process indexed by T can be defined in several ways [6]:

1. A collection $\{X_t\}_{t \in T}$ of random variables on $(\Omega, \mathcal{A}, P)$
2. A random function $X(\cdot) \in \mathbb{R}^T$
3. A function $X(t, \omega)$ on $T \times \Omega$ such that $X(t, \cdot)$ is measurable for each $t \in T$

Using the first definition for stochastic processes, Gaussian processes (GPs) can be defined as follows. For any set T, a Gaussian process indexed by T is defined to be a collection $\{X_t\}_{t \in T}$ of jointly Gaussian random variables on $(\Omega, \mathcal{A}, P)$. That the random variables are jointly Gaussian means that any finite vector consisting of random variables from $\{X_t\}_{t \in T}$ is a multivariate Gaussian random variable.

The function $m(t) = \mathbb{E}[X_t]$, $t \in T$ is called the mean function and the Gaussian process is called centered if $m(t) = \mathbb{E}[X_t] = 0$, $\forall t \in T$ [7].

The covariance function $k(s, t)$ of the Gaussian process is defined by $k(s, t) = \mathrm{Cov}(X_s, X_t)$ for $(s, t) \in T \times T$. The covariance function of a Gaussian process is symmetric and positive semi-definite [6].

Conversely, given any set T, a function $m(\cdot) \in \mathbb{R}^T$ and a symmetric positive semi-definite function $k(\cdot, \cdot) \in \mathbb{R}^{T \times T}$ there exists a Gaussian process with mean function $m(\cdot)$ and covariance function $k(\cdot, \cdot)$ [7].

Stationary Stochastic Processes

A stochastic process $X(t)$ with $T = \mathbb{R}^D$ is called strictly stationary or strictly homogeneous if its finite dimensional distributions are invariant under translations in the parameter t. This means that for any set of points $\tau, t_1, ..., t_j$ in $\mathbb{R}^D$ the joint distribution of $X(t_1), X(t_2), ..., X(t_j)$ is the same as the joint distribution of $X(t_1 + \tau), X(t_2 + \tau), ..., X(t_j + \tau)$ [8].

Often it is enough to consider weakly stationary processes which satisfy that their mean function $m(t)$ is a constant function and the covariance function $k(s,t)$ is a function of $s-t$ only. A strictly stationary stochastic process is clearly also weakly stationary, but the reverse is generally not true. However, for real Gaussian processes any weakly stationary GP is also a strictly stationary GP.

Continuity and Differentiability of Stochastic Processes

There are various types of continuity and differentiability for stochastic processes, since we are not studying a regular deterministic function but rather the convergence of a sequence of random variables. One type of stochastic convergence is mean square convergence and this is often used for stochastic processes since it has a natural connection to the covariance function [8].

A stochastic process $X(t)$ is said to be continuous in mean square at $t_* \in \mathbb{R}^D$ if for a sequence of points $t_1, t_2, ...$ such that $|t_j - t_*| \to 0$ as $j \to \infty$, it holds that $\mathbb{E}[|X(t_j) - X(t_*)|^2] \to 0$ as $j \to \infty$. If this is true for all $t_* \in A \subset \mathbb{R}^D$, then $X(t)$ is said to be continuous in mean square (MS) over A.

Using the covariance function, MS continuity can also be studied by using that $X(t)$ is continuous in mean square at $t_* \in \mathbb{R}^D$ if and only if its covariance function $k(s,t)$ is continuous at the point $t = s = t_*$. However, MS continuity does not imply that $X(t)$ is almost surely continuous, also called sample function continuity, which is a stronger condition given by:

$$P(\omega : |X(t_j, \omega) - X(t_*, \omega)| \to 0 \quad \text{as } j \to \infty) = 1 \tag{2.8}$$

where as above the sequence of points $t_1, t_2, ...$ is such that $|t_j - t_*| \to 0$ as $j \to \infty$.

We can also consider differentiability in mean square for a real stochastic process X. If the derivative $\frac{\partial^2 k(s,t)}{\partial s_i \partial t_i}$ exists and is finite at the point $(t,t) \in \mathbb{R}^D \times \mathbb{R}^D$ then the limit:

$$X_i(t) = \lim_{h \to 0} \frac{X(t + he_i) - X(t)}{h} \tag{2.9}$$

exists (in MS) and is called the MS derivative of $X(t)$ at t. Here e_i is the unit vector in direction i. The stochastic process is said to be possess a MS derivative if $X_i(t)$ exists for each $t \in \mathbb{R}^D$. The covariance function of $X_i(t)$ is then given by $\frac{\partial^2 k(s,t)}{\partial s_i \partial t_i}$. This can then be extended to higher order derivatives as well.

For a stationary stochastic process the conditions for MS continuity and differentiability are simplified. It is enough to check if the covariance function is continuous at one single point where $s - t = 0$ to see if the stochastic process is MS continuous. For differentiability we note that if all the $2k$:th order partial derivatives of the covariance function exist and are finite at the origin, then all the k:th order MS partial derivatives of the stochastic process exist as well.

Degenerate Covariance Functions

Suppose that we have a covariance function $k(\mathbf{x}, \mathbf{x}')$ on $\mathbb{R}^D \times \mathbb{R}^D$. We consider the integral equation:

$$\int k(\mathbf{x}, \mathbf{x}')\phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda \phi(\mathbf{x}'), \quad \text{for } \mathbf{x}' \in \mathbb{R}^D \tag{2.10}$$

A function which satisfies both this equation for some $\lambda \neq 0$ and $\int |\phi(\mathbf{x})|^2 d\mathbf{x} < \infty$ is called an eigenfunction of the covariance function with respect to the measure μ and the corresponding λ is called an eigenvalue. In general there are an infinite number of eigenvalues $\lambda_1, \lambda_2, ...$ and corresponding eigenfunctions $\phi_1, \phi_2, ...$. The eigenvalues can be chosen so that the eigenvalue sequence is non-increasing and that the eigenfunctions form an orthonormal sequence, hence $\int \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$. Then Mercer's theorem tells us that the covariance function can be expanded as $k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{x}')$ and $\sum_{j=1}^{\infty} \lambda_j^2 < \infty$ [9].

A covariance function is called degenerate if it has only a finite number of non-zero eigenvalues. If a covariance function is not degenerate it is called non-degenerate.

2.2 Gaussian Processes from a Machine Learning Perspective

The main reference for Gaussian Processes from a Machine Learning perspective has been the book Gaussian Processes for Machine Learning by Carl Edward Rasmussen and Christopher K. I. Williams [9].

2.2.1 Supervised Learning

The problem we consider is the one of supervised learning, which means that we want to learn input-output mappings from a training data set. If the output data are continuous this is called regression and if they are discrete the problem is called classification.

Following the notation of Rasmussen and Williams we have a training data set $\mathcal{D}$ which consists of n observation pairs of inputs and outputs: $\mathcal{D} = \{(\mathbf{x}_i, y_i) \,|\, i = 1, ..., n\}$. Here $\mathbf{x}$ is an input vector of dimension D and y is the scalar output, also called target. Using all our input vectors and outputs we can construct the design matrix X and the target vector $\mathbf{y}$ as follows:

$$X = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \end{pmatrix} \tag{2.11}$$

$$\mathbf{y} = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}^\top \tag{2.12}$$

The training data set can then be written as $\mathcal{D} = (X, \mathbf{y})$. The goal of supervised learning is to make inferences about the general relationship between inputs and targets using the labelled training data.

2.2.2 GP Regression

When using GPs for regression and classification the interpretation of a GP is a distribution over functions, where this distribution is specified by the mean and covariance functions of the GP.

Using the notation of Rasmussen and Williams [9] we write a GP as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \tag{2.13}$$

where $m(\mathbf{x})$ is the mean function and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function as in Section 2.1.2. This uses the second definition of GPs in Section 2.1.2, but the random function $X(\cdot)$ is here denoted $f(\cdot)$. The index set T is in this case the set of possible inputs denoted $\mathcal{X}$ and we will usually use $\mathcal{X} = \mathbb{R}^D$, where $D \in \mathbb{N}^+$. We will also write $f_i = f(\mathbf{x}_i)$

and denote the matrix of covariances evaluated at all pairs of points in two data sets X^1 and X^2 by $K(X^1, X^2)$ or K_{X^1,X^2} where $K(X^1, X^2)_{ij} = k(\mathbf{x}_i^1, \mathbf{x}_j^2)$. For notational simplicity the mean function will usually be assumed to be zero.

To be able to use the GP for regression or classification, we need to assume a model which connects the output $y(\mathbf{x})$ to our GP $f(\mathbf{x})$. Usually $f(\mathbf{x})$ can be seen as a latent function, which is used to infer the predictive distribution of $y(\mathbf{x}_*)$ at a test input $\mathbf{x}_*$ given the data set $\mathcal{D}$ by first inferring a posterior distribution over $f(\mathbf{x}_*)$.

Usually the output values are not the actual function values of f, but noisy observations of them. This is often modelled by additive independent identically distribution Gaussian noise, hence $y(\mathbf{x}) = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$.

Predictive distribution

The covariance between the outputs when using the model described above is given by:

$$\mathrm{Cov}(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq} \tag{2.14}$$
$$\mathrm{Cov}(\mathbf{y}) = K(X, X) + \sigma_n^2 I \tag{2.15}$$

The joint distribution of the observed output values $\mathbf{y}$ and the function values at the test input points $\mathbf{f}_*$ is in this case given by:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{pmatrix}\right) \tag{2.16}$$

The distribution of the function values at the test input point conditioned on the training data set is then as follows:

$$\mathbf{f}_* | X_*, X, \mathbf{y} \sim \mathcal{N}(\mathbb{E}[\mathbf{f}_* | X_*, X, \mathbf{y}], \mathrm{Cov}(\mathbf{f}_* | X_*, X, \mathbf{y})) \tag{2.17}$$
$$\mathbb{E}[\mathbf{f}_* | X_*, X, \mathbf{f}] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}, \tag{2.18}$$
$$\mathrm{Cov}(\mathbf{f}_* | X_*, X, \mathbf{y}) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)) \tag{2.19}$$

This is then called the predictive distribution for the Gaussian process regression. The predictive distribution of the corresponding test targets can then be computed by adding $\sigma_n^2 I$ to the expression for $\mathrm{Cov}(\mathbf{f}_* | X_*, X, \mathbf{f})$:

$$\mathbf{y}_* | X_*, X, \mathbf{y} \sim \mathcal{N}(\mathbb{E}[\mathbf{y}_* | X_*, X, \mathbf{y}], \mathrm{Cov}(\mathbf{y}_* | X_*, X, \mathbf{y})) \tag{2.20}$$
$$\mathbb{E}[\mathbf{y}_* | X_*, X, \mathbf{f}] = K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y}, \tag{2.21}$$
$$\mathrm{Cov}(\mathbf{y}_* | X_*, X, \mathbf{f}) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)) + \sigma_n^2 I \tag{2.22}$$

Marginal likelihood

The log marginal likelihood of the observed output values can be found by noting that the distribution is $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, K(X, X) + \sigma_n^2 I)$, hence:

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^\top [K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y} - \frac{1}{2}\log |K(X, X) + \sigma_n^2 I| - \frac{n}{2}\log 2\pi \tag{2.23}$$
$$\propto -\frac{1}{2}\left(\mathbf{y}^\top [K(X, X) + \sigma_n^2 I]^{-1}\mathbf{y} + \log |K(X, X) + \sigma_n^2 I|\right) \tag{2.24}$$

Deterministic function as mean function

If the mean function would be a deterministic function $m(\mathbf{x})$ instead of the zero function, the zero mean GP can simply be applied to the difference between the observations and the fixed mean. Then the variance remains unchanged and the predictive mean becomes:

$$\mathbb{E}[\mathbf{f}_*|X_*, X, \mathbf{f}] = \mathbf{m}(X_*) + K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1}(\mathbf{y} - \mathbf{m}(X)) \tag{2.25}$$

Bayesian linear model as a GP

A Bayesian linear regression model can be written as $f(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}$ where $\boldsymbol{\phi}(\cdot)$ is a deterministic function which maps a D-dimensional vector into an N dimensional feature space and $\mathbf{w}$ are the weights with prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$. We then have that $f(\mathbf{x})$ and $f(\mathbf{x}')$ are jointly Gaussian with zero mean and covariance $\boldsymbol{\phi}(\mathbf{x})^\top \Sigma_p \boldsymbol{\phi}(\mathbf{x}')$, so $f(\cdot)$ is a GP with mean function zero and covariance $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top \Sigma_p \boldsymbol{\phi}(\mathbf{x}')$.

Illustration of GP regression

In Figure 2.1 an example of the prior and the predictive distributions of a GP is illustrated. For this example the squared exponential covariance function was used (see 2.2.4) with parameters $l = \sqrt{2}$ and $\sigma_f = 1$. In Figure 2.1a a plot of the prior mean as well as 95 % confidence intervals are shown and in Figure 2.1b samples from this GP is shown. After observing 5 training data points the posterior is computed and its mean and 95 % confidence intervals are shown in Figure 2.1c. In Figure 2.1d samples of the posterior GP is plotted. It can be noted that at the exact location of the training points the posterior variance is 0 and all the samples have to pass through the training data points, while far away from the observed data points the uncertainty increases.

2.2.3 GP Classification and Non-Gaussian Likelihoods

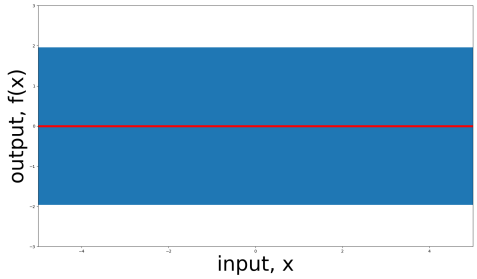
In general we will consider GP regression with Gaussian likelihood, but also regression with non-Gaussian likelihoods as well as classification can be solved with a GP, which is briefly explained in this section.

For classification we need a different model than for regression since in this case the outputs are discrete. Considering binary classification, we assume that we have the two possible labels such that $y \in \{-1, 1\}$. The prediction using GP is then usually done by "squashing" our latent random function $f(\mathbf{x})$ through the logistic function $\sigma(z) = 1/(1 + \exp(-z))$ and setting $p(y(\mathbf{x}) = 1|f(\mathbf{x})) = \sigma(f(\mathbf{x}))$. Other sigmoid functions than the logistic function could also be used for the "squashing", for example the cumulative distribution function of the standard Gaussian distribution.

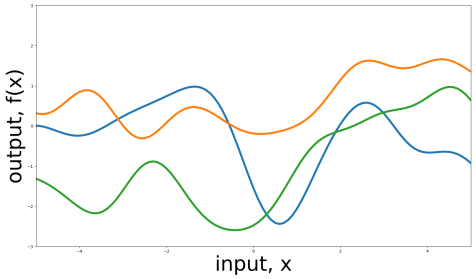
As for the regression case, we want to infer the predictive distribution of $y(\mathbf{x}_*)$ at a test input $\mathbf{x}_*$ given the data set $\mathcal{D}$ by first inferring a posterior distribution over $f(\mathbf{x})$, but in this case the posterior distribution of f_* given by:

$$p(f_*|X, \mathbf{y}, \mathbf{x}_*) = \int p(f_*|X, \mathbf{y}, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f}|X, \mathbf{y}) d\mathbf{f} \tag{2.26}$$

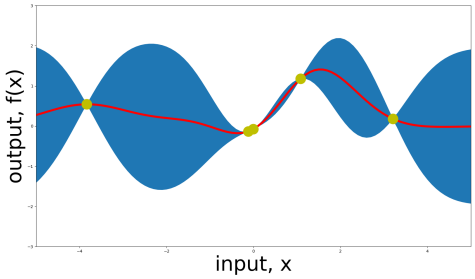
can not be analytically found since $p(\mathbf{f}|X, \mathbf{y}) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|X)/p(\mathbf{y}|X)$ involves the non-Gaussian likelihood $p(\mathbf{y}|\mathbf{f})$.



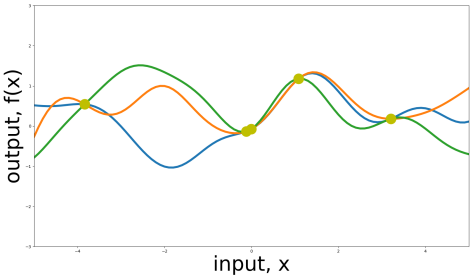
(a) Prior GP



(b) Samples prior GP



(c) Posterior GP



(d) Samples posterior GP

Figure 2.1: Illustration of inference with a GP. a) Mean in red and 95 % confidence intervals in blue of the prior GP. b) Samples from the prior GP. c) Mean in red and 95 % confidence intervals in blue of the posterior GP with training data points marked with yellow circles. d) Samples from the posterior GP with training data points marked with yellow circles.

Also the posterior distribution over y_* uses an analytically intractable integral:

$$p(y_* = 1|X, \mathbf{y}, \mathbf{x}_*) = \int \sigma(f_*)p(f_*|X, \mathbf{y}, \mathbf{x}_*)df_* \tag{2.27}$$

Instead of analytical expressions, approximations of some sort are needed. The likelihood $p(\mathbf{f}|X, \mathbf{y})$ can either be approximated as a Gaussian, using for example the Laplace approximation or expectation propagation (EP), or analytical approximations of integrals based on Monte Carlo sampling can be used. These methods can more generally be used as soon as the likelihood is not Gaussian for any reason.

This binary classification can be generalized to multi class classification, for example by using the softmax functions so that the probability of the training point i belonging to class c is given by:

$$p(y_i^c|\mathbf{f}_i) = \frac{\exp(f_i^c)}{\sum_{c'} \exp(f_i^{c'})} \tag{2.28}$$

where f_i^c is the latent function value at training point i for class c and the latent GPs for different classes are assumed to be uncorrelated.

2.2.4 Covariance Function

The choice of covariance function for the GP predictor is of most importance, since the covariance function encodes our beliefs and assumptions about the underlying function that we want to learn. It is the covariance function that defines nearness or similarity between points for our GP.

For example, a basic assumption is that points with inputs which are close to each other have a high probability to have similar target values, and therefore training points close to a test point should be important for the prediction of that test point. This means that the covariance function should have high values for points close to each other.

From Section 2.1.2 it is clear that we can choose any symmetric positive semi-definite function to be the covariance function. Given a set of input points $\{\mathbf{x}_i|i = 1, ..., n\}$ the Gram matrix or covariance matrix K is given by $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Since the covariance function is a positive semi-definite function, the covariance matrix must be a positive semi-definite matrix and in fact a function is positive semi-definite if all possible Gram matrices for any choice of $n \in \mathbb{N}$ and $\{\mathbf{x}_i|i = 1, ..., n\}$ are positive semi-definite.

There are some basic terms related to covariance functions. A stationary covariance function is a function of $\mathbf{x} - \mathbf{x}'$ while an isotropic covariance function is a function only of $|\mathbf{x} - \mathbf{x}'|$. This means that a stationary covariance function is invariant to translations in the input space and an isotropic covariance function is invariant to all rigid motions. We also have dot product covariance functions which are functions of $\mathbf{x} \cdot \mathbf{x}'$. These are invariant to a rotation of the coordinates about the origin, but not to translations.

Squared Exponential Covariance Function

The squared exponential (SE) covariance function is according to Rasmussen and Williams probably the most widely-used within the covariance function machines field. It has the form:

$$k_{SE}(\mathbf{x}, \mathbf{x}') = \sigma_f \exp\left(-\frac{r^2(\mathbf{x}, \mathbf{x}')}{2l^2}\right) \tag{2.29}$$

where $r(\mathbf{x}, \mathbf{x}') = |\mathbf{x} - \mathbf{x}'|$, the parameter σ_f determines the overall variance of the process, the parameter l is the characteristic length-scale which determines how far apart points must be to become practically uncorrelated. Since it is only a function of $|\mathbf{x} - \mathbf{x}'|$ it is an isotropic covariance function. The SE covariance function has mean square derivatives of all orders and is therefore very smooth, which may be unrealistic for a lot of models.

An anisotropic version of the SE covariance function can be created by setting $r^2(\mathbf{x}, \mathbf{x}') = (\mathbf{x} - \mathbf{x}')^\top M(\mathbf{x} - \mathbf{x}')$ for some positive semi-definite matrix M. If M is diagonal we get different length-scales for different dimensions and such a covariance function implements automatic relevance determination (ARD) since the inverse of the length-scale determine the relevance of the input in that dimension.

The Matérn Class of Covariance Functions

The Matérn class of covariance functions has the form:

$$k_{\text{Matern}}(\mathbf{x}, \mathbf{x}') = \sigma_f \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r(\mathbf{x}, \mathbf{x}')}{l} \right)^\nu K_\nu \left(\frac{\sqrt{2\nu} r(\mathbf{x}, \mathbf{x}')}{l} \right) \tag{2.30}$$

where $\nu > 0$, $l > 0$ and $\sigma_f > 0$ are hyperparameters, K_ν is a modified Bessel function and $\Gamma(\cdot)$ is the Gamma function. The Matérn covariance is isotropic and the hyperparameter l is again the length-scale, while ν determines how smooth the covariance is and as $\nu \to \infty$ we obtain the SE covariance function.

2.2.5 Learning: Choice of Covariance Function and Hyperparameters

As already mentioned in Section 2.2.4 the choice of covariance function is very important for the performance of the GP regression. Except the functional form of the covariance function, the covariance function $k(\cdot, \cdot)$ usually also has some free parameters called hyperparameters, for example the length-scale in the SE function. The model selection or training of a GP includes both choosing the functional form and the values of the hyperparameters for the covariance function.

Usually, the functional form of the covariance function is already chosen (for example based on reasoning about the underlying model and the training data) and only the values of the hyperparameters $\boldsymbol{\theta}$ are left to be determined. According to the Bayesian model selection, we would like to marginalise over the hyperparameters. However, these computations are usually not analytically tractable. Approximations of the marginalisation can be made by using MCMC, but more frequently used is type II maximum likelihood where the hyperparameters are determined by maximizing the marginal likelihood $p(\mathbf{y}|X, \boldsymbol{\theta})$. This can be done by using a gradient based optimizer, but there is no guarantee that we will not end up in only a local maximum.

A problem when using maximum likelihood methods is the risk of overfitting. However, typical covariance functions usually have just a small number of hyperparameters and we are not maximizing the likelihood directly but the marginal likelihood which has an automatic trade-off between data-fit and model complexity. The marginal likelihood conditioned on the hyperparameters is given by:

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) \propto -\frac{1}{2} \left(\mathbf{y}^\top [K(X, X) + \sigma_n^2 I]^{-1} \mathbf{y} + \log |K(X, X) + \sigma_n^2 I| \right) \tag{2.31}$$

Here the first term is the one involving the fit of the data while the second term is the automatic complexity penalty which only depends on the covariance function and not the observed targets.

Another possibility for model selection is cross-validation.

2.3 Scalable Gaussian Processes

A problem with the original GPs is the computational cost. For the inference $(K(X,X) + \sigma_n^2 I)^{-1}\mathbf{y}$ must be computed and for the hyperparameter learning also the log determinant $\log|K(X,X) + \sigma_n^2 I|$ must be computed. The inversion of the covariance matrix as well as the log determinant have a computational cost of $\mathcal{O}(n^3)$ when using the regular Cholesky decomposition. After this is done the predictions have a computational cost of $\mathcal{O}(n)$ for the predictive mean and $\mathcal{O}(n^2)$ for the predictive variance. This means that for large data sets the original GPs are not reasonable to use.

Rasmussen and Williams review a number of fast approximations for large data sets in their book [9], which are summarized below. However, scalable GPs are an active area of research and several additional scalable GP methods have been proposed since their review. The rest of the methods for scalable GPs are taken from various articles, see the references for each method.

2.3.1 Iterative Solution of Linear Systems

The linear system $(K + \sigma_n^2 I)\mathbf{v} = \mathbf{y}$ used in GP regression can be solved by an iterative method. One iterative method that can be used is linear conjugate gradients (LCG), which gives an exact solution after n iterations, but it can be used to give an approximate solution by earlier termination after k iterations. Then the time complexity is $\mathcal{O}(kn^2)$. The conjugate gradients method could also be sped up by using approximate matrix-vector multiplication.

2.3.2 Sparse Approximations

The complexity of inference using a full GP is $\mathcal{O}(n^3)$ due to the inversion of the matrix $K + \sigma_n^2 I$. If the matrix K has rank $m < n$ it can be represented by $K = QQ^\top$ where Q is an $n \times m$ matrix. Then the matrix inversion can be sped up by using $(QQ^\top + \sigma_n^2 I_n)^{-1} = \sigma_n^{-2} I_n - \sigma_n^{-2} Q(\sigma_n^2 I_q + Q^\top Q)^{-1} Q^\top$ where the inversion is of a $m \times m$ matrix instead and the complexity is reduced to $\mathcal{O}(nm^2)$.

If K is not of rank $m < n$ we can still consider reduced-rank approximations of the matrix. This is equivalent to approximating the full non-degenerate GP with a finite m-dimensional degenerate linear model, which is a good approximation if the eigenvalue spectrum of the covariance function decays fairly rapidly and this is the case for typical covariance functions, e.g. the SE covariance function [1].

However, computing the m leading eigenvalues by doing an eigendecomposition is in general of complexity $\mathcal{O}(n^3)$ so this does not decrease the complexity. Instead one can use a subset of input points as a basis for low rank construction. These points are often called inducing points or pseudo-inputs and generally they do not have to be original training points. The different methods for scalable GPs based on low rank approximations are usually called sparse approximations even though the matrices used are not

sparse, this name instead refers to the fact that the number of points used in the methods are sparse.

Subset of Data Points

For the subset of data points (SD) method, we use the original GP predictor, but only use a smaller subset of size m of the data. For this approximation the main question is how to choose the subset of data points before using the GP. One alternative is to choose the next point to include by maximizing the differential entropy score given by $\Delta_j = H[p(f_j)] - H[p^{\text{new}}(f_j)]$ where $H[p(f_j)]$ is the entropy of the Gaussian at point j before including a new point and $H[p^{\text{new}}(f_j)]$ is the entropy of the Gaussian at point j after including point j. This leads to an overall complexity of $\mathcal{O}(m^2 n)$. When using this selection criterium, the method is called informative vector machine (IVM). Another alternative is to use the information gain $\text{KL}(p^{\text{new}}(f_j)||p(f_j)))$ as criterion for the selection of the next point.

Subset of Regressors

It can be shown that the mean GP predictor can be found by considering a finite-dimensional generalized regression model:

$$f(\mathbf{x}_*) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_*, \mathbf{x_i}) \tag{2.32}$$

where the prior over the parameters $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ is given by $\alpha \sim \mathcal{N}(\mathbf{0}, K^{-1})$. The predictive mean of this model will be the same as for the GP regression, but the predictive variance will differ.

As an approximation to the model f we can consider only a subset of the regressors (SR), giving the approximate model:

$$f_{SR}(\mathbf{x}_*) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_*, \mathbf{x_i}) \tag{2.33}$$

where $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_m) \sim \mathcal{N}(\mathbf{0}, K_{mm}^{-1})$. This approximate model has the predictive mean and variance given by:

$$\mathbb{E}[f_{SR}(\mathbf{x}_*)] = \mathbf{k}_m(\mathbf{x}_*)^\top (K_{mn} K_{nm} + \sigma_n^2 K_{mm})^{-1} K_{mn} \mathbf{y} \tag{2.34}$$

$$\mathbb{V}[f_{SR}(\mathbf{x}_*)] = \sigma_n^2 \mathbf{k}_m(\mathbf{x}_*)^\top (K_{mn} K_{nm} + \sigma_n^2 K_{mm})^{-1} \mathbf{k}_m(\mathbf{x}_*) \tag{2.35}$$

The cost required for the needed matrix computations is $\mathcal{O}(m^2 n)$ instead of the original $\mathcal{O}(n^3)$. Thereafter, predicting the mean for a new test point is $\mathcal{O}(m)$ and the variance $\mathcal{O}(m^2)$.

The subset of points used for the approximate method can be chosen in multiple ways. It could simply be chosen randomly from X or it could be the cluster centres obtained from clustering X. A third alternative is to use a greedy forward selection algorithm, for example choosing the next points by minimizing the residual sum of squares or maximizing the marginal likelihood $\log p_{SR}(\mathbf{y}|X)$.

Projected Process Approximation

The projected process (PP) approximation uses only m values of $f(\cdot)$ but makes use of all n data points in the likelihood by projecting the m latent points to n dimensions.

First we need to split the original $\mathbf{f}$ into $\mathbf{f}_m$ and $\mathbf{f}_{n-m}$. We will use the notation K_{mm} for the covariance matrix of the m inducing points, K_{nn} for the original covariance matrix $K_{X,X}$ of all the n training points, $K_{(n-m)m}$ for the covariance matrix between the $n-m$ training points which are not inducing points and the m inducing points and so on.

We then have a conditional distribution $p(\mathbf{f}_{n-m}|\mathbf{f}_m)$ with mean $\mathbb{E}[\mathbf{f}_{n-m}|\mathbf{f}_m] = K_{(n-m)m}K_{mm}^{-1}\mathbf{f}_m$ and replace the true likelihood term for the $(n-m)$ points by $\mathcal{N}(\mathbf{y}_{n-m}|\mathbb{E}[\mathbf{f}_{n-m}|\mathbf{f}_m], \sigma_n^2 I)$. We then get the approximate likelihood by:

$$q(\mathbf{y}|\mathbf{f}_m) = \mathcal{N}(\mathbf{y}|K_{nm}K_{mm}^{-1}\mathbf{f}_m, \sigma_n^2 I) \tag{2.36}$$

Using this likelihood and the prior of $p(\mathbf{f}_m)$ leads to the predictive mean and variance:

$$\mathbb{E}[f(\mathbf{x}_*)] = \mathbf{k}_m(\mathbf{x}_*)^\top (K_{mn}K_{nm} + \sigma_n^2 K_{mm})^{-1} K_{mn}\mathbf{y} \tag{2.37}$$

$$\mathbb{V}[f(\mathbf{x}_*)] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_m(\mathbf{x}_*)^\top K_{mm}^{-1}\mathbf{k}_m(\mathbf{x}_*) + \sigma_n^2 \mathbf{k}_m(\mathbf{x}_*)^\top (K_{mn}K_{nm} + \sigma_n^2 K_{mm})^{-1}\mathbf{k}_m(\mathbf{x}_*) \tag{2.38}$$

This is the same predictive mean as for the SR model, but a different predictive variance. The marginal likelihood of the PP approximation will also be the same as for the SR model. The matrix computations needed take $\mathcal{O}(m^2 n)$ and prediction of a new test point then takes $\mathcal{O}(m)$ for the mean and $\mathcal{O}(m^2)$ for the variance.

For this model as well the question of how to choose the m points arises. One method is to use a point if the novelty is large enough, where the novelty of an input $\mathbf{x}$ is given by $k(\mathbf{x}, \mathbf{x}) - \mathbf{k}_m(\mathbf{x})^\top K_{mm}^{-1}\mathbf{k}(\mathbf{x})$.

Sparse Pseudo-input Gaussian Process

The sparse pseudo-input GP (SPGP) was introduced in 2006 by Snelson and Ghahramani [10]. This methods replaces the real training data set $\mathcal{D} = (X, \mathbf{y})$ by a pseudo data set $\bar{\mathcal{D}} = (\bar{X}, \bar{\mathbf{f}})$ of size $m < n$ which are seen as parameters of the model. The GP predictive distribution from this pseudo data set is then used as a parametrised model likelihood. This leads to the following likelihood for a single point:

$$p(y|\mathbf{x}, \bar{X}, \bar{\mathbf{f}}) \sim \mathcal{N}(K_{\mathbf{x}m}K_m^{-1}\bar{\mathbf{f}}, k_{\mathbf{xx}} - K_{\mathbf{x}m}K_m^{-1}K_{m\mathbf{x}} + \sigma_n^2) \tag{2.39}$$

where $K_{\mathbf{x}m}$ is the covariance matrix between the input $\mathbf{x}$ and all the pseudo-inputs $\bar{X}$ and K_m is the covariance of the pseudo-inputs. Assuming that the output data are generated i.i.d. given the inputs leads to the complete training data likelihood:

$$p(\mathbf{y}|\bar{\mathbf{f}}) = \prod_{i=1}^{n} p(y_i|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{nm}K_m^{-1}\bar{\mathbf{f}}, \mathrm{diag}(K_n - Q_n) + \sigma_n^2 I_n) \tag{2.40}$$

where the explicit conditioning on the inputs is dropped and the low rank covariance matrix $Q_n = K_{nm}K_m^{-1}K_{mn}$ is used.

To find an appropriate pseudo-data set that explains the real data well, a Gaussian prior is placed on the pseudo outputs to be able to integrate these out and then maximum likelihood is used to find the pseudo-inputs. The prior is given by:

$$p(\bar{\mathbf{f}}) \sim \mathcal{N}(\mathbf{0}, K_m) \tag{2.41}$$

This leads to the SPGP marginal likelihood:

$$p(\mathbf{y}) = \int p(\mathbf{y}|\bar{\mathbf{f}})p(\bar{\mathbf{f}}) \sim \mathcal{N}(\mathbf{0}, Q_n + \text{diag}(K_n - Q_n) + \sigma_n^2 I_n) \tag{2.42}$$

The SPGP predictive distribution for a point $(\mathbf{x}_*, y_*)$ is then obtained by first finding the joint distribution $p(y_*, \mathbf{y})$ and then conditioning on $\mathbf{y}$, which leads to:

$$p(y_*|\mathbf{y}) \sim \mathcal{N}(\mu_*, \sigma_*^2) \tag{2.43}$$

$$\mu_* = Q_{*n}[Q_n + \text{diag}(K_n - Q_n) + \sigma_n^2 I_n]^{-1}\mathbf{y} \tag{2.44}$$

$$\sigma_* = k_{**} - Q_{*n}[Q_n + \text{diag}(K_n - Q_n) + \sigma_n^2 I_n]^{-1}Q_{n*} + \sigma_n^2 \tag{2.45}$$

Using the matrix inversion lemma the computational cost of the SPGP is $\mathcal{O}(m^2 n)$. The prediction of a new test point then takes $\mathcal{O}(m)$ for the mean and $\mathcal{O}(m^2)$ for the variance.

The pseudo-inputs can be considered as extra hyperparameters and can be found at the same time as the regular hyperparameters by maximizing the marginal likelihood with respect to both the pseudo-inputs and the hyperparameters by gradient ascent. The derivatives with respect to all the pseudo-inputs can be computed in $\mathcal{O}(nm^2 + nmD)$ time.

Variational Lower Bound

Titsias [11] introduced a variational formulation for sparse approximations in 2009 which infers the inducing points and covariance hyperparameters by maximizing a lower bound of the exact marginal likelihood of the full GP. This method is also called the Variational Free Energy (VFE) approximation [12].

As for the SPGP, a pseudo data set $\bar{\mathcal{D}} = (\bar{X}, \bar{\mathbf{f}})$ of size $m < n$ is used to obtain this approximation.

The posterior distribution for the full GP on a set of function points $\mathbf{z}$ is given by:

$$p(\mathbf{z}|\mathbf{y}) = \int p(\mathbf{z}|\mathbf{f})p(\mathbf{f}|\mathbf{y})d\mathbf{f} = \int p(\mathbf{z}|\bar{\mathbf{f}}, \mathbf{f})p(\mathbf{f}|\bar{\mathbf{f}}, \mathbf{y})p(\bar{\mathbf{f}}|\mathbf{y})d\mathbf{f}d\bar{\mathbf{f}} \tag{2.46}$$

Suppose that it would hold that $p(\mathbf{z}|\bar{\mathbf{f}}, \mathbf{f}) = p(\mathbf{z}|\bar{\mathbf{f}})$ then we could write the posterior as:

$$q(\mathbf{z}) = \int p(\mathbf{z}|\bar{\mathbf{f}})p(\mathbf{f}|\bar{\mathbf{f}})\phi(\bar{\mathbf{f}})d\mathbf{f}d\bar{\mathbf{f}} = \int p(\mathbf{z}|\bar{\mathbf{f}})\phi(\bar{\mathbf{f}})d\bar{\mathbf{f}} \tag{2.47}$$

where $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{y})$ and $\phi(\bar{\mathbf{f}}) = p(\bar{\mathbf{f}}|\mathbf{y})$. However, in general it is not true that $p(\mathbf{z}|\bar{\mathbf{f}}, \mathbf{f}) = p(\mathbf{z}|\bar{\mathbf{f}})$ and therefore we expect $q(\mathbf{z})$ to be only an approximation of $p(\mathbf{z}|\mathbf{y})$. We can then choose $\phi(\bar{\mathbf{f}})$ to be a free variational Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, A)$ where in general $\phi(\bar{\mathbf{f}}) \neq p(\bar{\mathbf{f}}|\mathbf{y})$. The mean and covariance functions of the approximate GP posterior are then given by:

$$m_{\mathbf{y}}^q(\mathbf{x}) = K_{\mathbf{x}m}K_{mm}^{-1}\boldsymbol{\mu} \tag{2.48}$$

$$k_{\mathbf{y}}^q(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}, \mathbf{x}') - K_{\mathbf{x}m}K_{mm}^{-1}K_{m\mathbf{x}'} + K_{\mathbf{x}m}BK_{m\mathbf{x}'} \tag{2.49}$$

where $B = K_{mm}^{-1} A K_{mm}^{-1}$. As with the other sparse approximations this has complexity $\mathcal{O}(nm^2)$.

Treating $\bar{X}$ as a variational parameter, $(\bar{X}, \phi)$ can be determined by minimizing the KL divergence $\mathrm{KL}(q(\mathbf{f}, \bar{\mathbf{f}}) || p(\mathbf{f}, \bar{\mathbf{f}}|\mathbf{y}))$, which also can be expressed a maximizing the following variational lower bound of the true log marginal likelihood:

$$F_V(\bar{X}, \phi) = \int p(\mathbf{f}|\bar{\mathbf{f}}) \phi(\bar{\mathbf{f}}) \log \frac{p(\mathbf{y}|\mathbf{f}) p(\bar{\mathbf{f}})}{\phi(\bar{\mathbf{f}})} d\mathbf{f} d\bar{\mathbf{f}} \tag{2.50}$$

The optimal choice of the variational distribution ϕ can be found analytically, leading to the bound:

$$F_V(\bar{X}) = \log[\mathcal{N}(\mathbf{y}|\mathbf{0}, \sigma_n^2 I_n + Q_{nn})] - \frac{1}{2\sigma^2} \mathrm{Tr}(\tilde{K}) \tag{2.51}$$

where $Q_n = K_{nm} K_{mm}^{-1} K_{mn}$ and $\tilde{K} = \mathrm{Cov}(\mathbf{f}|\bar{\mathbf{f}}) = K_{nn} - K_{nm} K_{mm}^{-1} K_{mn}$. The trace term can be seen as a regularization term. The optimal ϕ^* is given by $\boldsymbol{\mu} = \sigma_n^{-2} K_{mm} \Sigma K_{mn} \mathbf{y}$, $A = K_{mm} \Sigma K_{mm}$ and $\Sigma = (K_{mm} + \sigma_n^{-2} K_{mn} K_{nm})^{-1}$. The approximate predictive distribution is then exactly the one used by PP, but the variational method differs in how to select the inducing points as well as the hyperparameters. For the variational method, these are found by maximizing the bound F_V which has a different form than the log likelihoods used in PP and SPGP due to the regularization term.

A Unifying View

Quiñonero-Candela and Rasmussen provided a unifying view of the existing sparse approximations in 2005 [3]. The available algorithms had been introduced with widely different motivations, which made it hard to understand how they relate to each other. Therefore, the unifying view was introduced which interprets all algorithms as exact inference with an approximated prior.

The exact joint GP prior is given by:

$$p(\mathbf{f}, \mathbf{f}_*) \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} K_{X,X} & K_{X,X_*} \\ K_{X_*,X} & K_{X_*,X_*} \end{pmatrix}\right) \tag{2.52}$$

Using m inducing variables $\bar{\mathbf{f}}$ corresponding to the input locations $\bar{X}$ this prior can be rewritten as:

$$p(\mathbf{f}, \mathbf{f}_*) = \int p(\mathbf{f}, \mathbf{f}_*, \bar{\mathbf{f}}) d\bar{\mathbf{f}} = \int p(\mathbf{f}, \mathbf{f}_*|\bar{\mathbf{f}}) p(\bar{\mathbf{f}}) d\bar{\mathbf{f}}, \quad \text{where } p(\bar{\mathbf{f}}) \sim \mathcal{N}(\mathbf{0}, K_{\bar{X},\bar{X}}) \tag{2.53}$$

Next the joint prior is approximated by assuming that $\mathbf{f}_*$ and $\mathbf{f}$ are conditionally independent given $\bar{\mathbf{f}}$, giving:

$$p(\mathbf{f}, \mathbf{f}_*) \approx q(\mathbf{f}, \mathbf{f}_*) = \int q(\mathbf{f}_*|\bar{\mathbf{f}}) q(\mathbf{f}|\bar{\mathbf{f}}) p(\bar{\mathbf{f}}) d\bar{\mathbf{f}} \tag{2.54}$$

The SR, PP and SPGP methods all correspond to different additional assumptions about $q(\mathbf{f}_*|\bar{\mathbf{f}})$ and $q(\mathbf{f}|\bar{\mathbf{f}})$. The exact expressions are given by:

$$p(\mathbf{f}|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, K_{X,X} - Q_{X,X}) \tag{2.55}$$

$$p(\mathbf{f}_*|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X_*,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, K_{X_*,X_*} - Q_{X_*,X_*}) \tag{2.56}$$

where $Q_{A,B} = K_{A,\bar{X}} K_{\bar{X},\bar{X}}^{-1} K_{\bar{X},B}$.

For SR the approximate conditional distributions are deterministic and are given by:

$$q_{SR}(\mathbf{f}|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, 0) \tag{2.57}$$

$$q_{SR}(\mathbf{f}_*|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X_*,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, 0) \tag{2.58}$$

Then the effective joint prior is given by:

$$q_{SR}(\mathbf{f}, \mathbf{f}_*) \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} Q_{X,X} & Q_{X,X_*} \\ Q_{X_*,X} & Q_{X_*,X_*} \end{pmatrix}\right) \tag{2.59}$$

Based on this, Quiñonero-Candela and Rasmussen suggest calling the method the Deterministic Inducing Conditional (DIC) approximation instead. It can be noted that the SR approximation is equivalent to exact inference in the degenerate GP with covariance function $k_{SR}(\mathbf{x}, \mathbf{x}') = K_{\mathbf{x}m} K_{mm}^{-1} K_{m\mathbf{x}'}$ which has rank at most m.

The PP method uses a deterministic training conditional approximation but keeps the test conditional exact:

$$q_{PP}(\mathbf{f}|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, 0) \tag{2.60}$$

$$q_{PP}(\mathbf{f}_*|\bar{\mathbf{f}}) = p(\mathbf{f}_*|\bar{\mathbf{f}}) \tag{2.61}$$

The effective joint prior is for PP given by:

$$q_{PP}(\mathbf{f}, \mathbf{f}_*) \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} Q_{X,X} & Q_{X,X_*} \\ Q_{X_*,X} & K_{X_*,X_*} \end{pmatrix}\right) \tag{2.62}$$

Since the covariances for training and test cases are treated differently, the PP method does not correspond exactly to a GP. The name of the method in the unifying framework is the deterministic training conditional (DTC) approximation.

The SPGP method approximates the training conditional distribution by assuming conditional independence. The name of the method in the unifying framework is the fully independent training conditional (FITC) approximation:

$$q_{FITC}(\mathbf{f}|\bar{\mathbf{f}}) = \prod_{i=1}^{n} p(f_i|\bar{\mathbf{f}}) \sim \mathcal{N}(K_{X,\bar{X}} K_{\bar{X},\bar{X}}^{-1} \bar{\mathbf{f}}, \mathrm{diag}[K_{X,X} - Q_{X,X}]) \tag{2.63}$$

$$q_{FITC}(\mathbf{f}_*|\bar{\mathbf{f}}) = p(\mathbf{f}_*|\bar{\mathbf{f}}) \tag{2.64}$$

The effective joint prior is for this approximation given by:

$$q_{FITC}(\mathbf{f}, \mathbf{f}_*) \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} Q_{X,X} - \mathrm{diag}[Q_{X,X} - K_{X,X}] & Q_{X,X_*} \\ Q_{X_*,X} & K_{X_*,X_*} \end{pmatrix}\right) \tag{2.65}$$

If also the test outputs are assumed to be conditionally independent then the method is called the fully independent conditional (FIC) approximation, which has the effective prior:

$$q_{FIC}(\mathbf{f}, \mathbf{f}_*) \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} Q_{X,X} - \mathrm{diag}[Q_{X,X} - K_{X,X}] & Q_{X,X_*} \\ Q_{X_*,X} & Q_{X_*,X_*} - \mathrm{diag}[Q_{X_*,X_*} - K_{X_*,X_*}] \end{pmatrix}\right) \tag{2.66}$$

This method has exactly the same marginal predictive variances at FITC, but the FIC corresponds to exact inference in a non-degenerate GP with covariance function $k_{FIC}(\mathbf{x}, \mathbf{x}') = k_{SR}(\mathbf{x}, \mathbf{x}') + \delta_{\mathbf{x},\mathbf{x}'}[k(\mathbf{x}, \mathbf{x}') - k_{SR}(\mathbf{x}, \mathbf{x}')]$

Comparison of the Sparse Approximations

Snelson compares the sparse approximations SR, PP and FITC in his PhD thesis [1]. He suggests using the FITC approximation over PP and SR in all cases since SR gives rise to too small predictive variances far away from the inducing points, PP has low noise problems and FITC can be seen as a closer approximation to the full GP in the unifying view. He also shows experimentally that SPGP achieves high accuracy.

In 2016 Bauer et al. compared the popular FITC and VFE approximations [12]. They show that the FITC can overestimate the marginal likelihood, severely underestimate the noise parameter σ_n and not recover the true posterior. The VFE is in contrast a true bound to the exact marginal likelihood and behaves as expected. In practice, FITC often performs surprisingly well due to local optima while VFE's objective is harder to optimise. Bauer et al. conclude that their recommendation is using VFE, but performing the optimization carefully.

2.3.3 Bayesian Committee Machine

The Bayesian Committee Machine (BCM) was introduced by Tresp [13] and is a transductive approach which uses a model dependent on the test set input locations [9]. For BCM the training data set $\mathcal{D}$ is split into p parts $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_p$ and approximated by:

$$p(\mathbf{y}_1, ..., \mathbf{y}_p | \mathbf{f}_*, X) \approx \prod_{i=1}^{p} p(\mathbf{y}_i | \mathbf{f}_*, X_i) \tag{2.67}$$

The predictive distribution for test points then becomes:

$$q(\mathbf{f}_* | \mathcal{D}_1, ..., \mathcal{D}_p) \propto p(\mathbf{f}_*) \prod_{i=1}^{p} p(\mathbf{y}_i | \mathbf{f}_*, X_i) \propto \frac{\prod_{i=1}^{p} p(\mathbf{f}_* | \mathcal{D}_i)}{p^{p-1}(\mathbf{f}_*)} \tag{2.68}$$

This is a Gaussian distribution with predictive mean and covariance:

$$\mathbb{E}_q[\mathbf{f}_* | \mathcal{D}] = [\mathrm{Cov}_q(\mathbf{f}_* | \mathcal{D})] \sum_{i=1}^{p} [\mathrm{Cov}_q(\mathbf{f}_* | \mathcal{D}_i)]^{-1} \mathbb{E}[\mathbf{f}_* | \mathcal{D}_i] \tag{2.69}$$

$$[\mathrm{Cov}_q(\mathbf{f}_* | \mathcal{D}_i)]^{-1} = -(p-1) K_{**}^{-1} + \sum_{i=1}^{p} [\mathrm{Cov}_q(\mathbf{f}_* | \mathcal{D}_i)]^{-1} \tag{2.70}$$

where K_{**} is the covariance matrix evaluated at the test points.

A question is how to choose the partitions of the dataset. One possibility is to make random partitions of equal size, another one is to cluster the data to obtain the partitions.

In case all partitions have size m and we make predictions for m test points the computational complexity is $\mathcal{O}(m^2 n)$ for predicting the m test points.

Chapter 3

Massively Scalable Gaussian Processes

Our focus for the review of scalable GPs has been on the so called Massively Scalable Gaussian Processes (MSGP) framework since this is what the Master's thesis will evaluate. This framework was introduced by Wilson et al. in 2016 [5].

3.1 Structure Exploitation

For the special cases when the input points are either on a rectilinear grid and the covariance function is a product kernel or the points are on a one-dimensional equidistant grid and the covariance function is stationary, the covariance matrix K has structures which can be exploited for faster computations.

3.1.1 Kronecker Structure

For the case when the input points are multidimensional and on a rectilinear grid, $\mathbf{x} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_D$ and the covariance function can be written as a product of functions for each dimensions $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x'_d)$, then the covariance matrix K can be written as a Kronecker product $K = K_1 \otimes \cdots \otimes K_D$. The eigendecomposition of $K = QVQ^\top$ can then be computed by separately computing the eigendecompositions of the smaller matrices $K_1, ..., K_D$. The matrices Q and V are given by the Kronecker products $Q = Q_1 \otimes \cdots \otimes Q_D$ and $V = V_1 \otimes \cdots \otimes V_D$ where $K_d = Q_d V_d Q_d^\top$, $d = 1, ..., D$, are the eigendecompositions of each K_d. Given the eigendecomposition of K we can use $(K + \sigma_n^2 I)^{-1}\mathbf{y} = (QVQ^\top + \sigma_n^2 I)^{-1}\mathbf{y} = Q(V + \sigma_n^2 I)^{-1}Q^\top \mathbf{y}$ and $\log|K + \sigma_n^2 I| = \sum_i \log(V_{ii} + \sigma_n^2)$. Since V is a diagonal matrix of the eigenvalues, the computation of $(V + \sigma_n^2 I)^{-1}$ is trivial and Q is an orthogonal matrix of eigenvectors that also can be decomposed as a Kronecker product, which makes fast matrix vector multiplications (MVMs) possible. Overall, for m grid points the training cost is $\mathcal{O}(Dm^{1+\frac{1}{D}})$ and the storage requirements are $\mathcal{O}(Dm^{2/D})$. However, note that these computations are only possible for multidimensional inputs and not for $D = 1$ when no decompositions are possible.

Matrix Vector Multiplication using Kronecker Structure

Kronecker products can be exploited to obtain fast MVMs by using the Kronecker property $(B \otimes C)\text{vec}(X) = \text{vec}(CXB^\top)$. From Appendix D in Wilson [14] an MVM can be computed as follows.

The wanted MVM is given by $K\mathbf{u} = \left(\bigotimes_{p=1}^{P} K_p\right)\mathbf{u}$ where $\mathbf{u} \in \mathbb{R}^m$, $K_p \in \mathbb{R}^{n_p \times n_p}$, $K \in \mathbb{R}^{m \times m}$ and $m = \prod_{p=1}^{P} n_p$. Using $(B \otimes C)\text{vec}(X) = \text{vec}(CXB^\top)$ we get:

$$K\mathbf{u} = \left(\bigotimes_{p=1}^{P} K_p\right)\mathbf{u} = \text{vec}\left(K_P U \left(\bigotimes_{p=1}^{P-1} K_p\right)^\top\right) \tag{3.1}$$

where $U = \text{reshape}(\mathbf{u}, n_P, m/n_P)$. Using the identity $(A^\top)^\top = A$ this can be rewritten as:

$$K\mathbf{u} = \text{vec}\left(\left(\left(\bigotimes_{p=1}^{P-1} K_p\right)(K_P U)^\top\right)^\top\right) \tag{3.2}$$

We now use the fact that:

$$\left(\bigotimes_{p=1}^{P-1} K_p\right)(K_P U)^\top = \text{reshape}\left(\text{vec}\left(\left(\bigotimes_{p=1}^{P-1} K_p\right)(K_P U)^\top\right), m/n_P, n_P\right) \tag{3.3}$$

For the inner component we can use $(B \otimes C)\text{vec}(X) = \text{vec}(CXB^\top)$ again, but backwards:

$$\text{vec}\left(\left(\bigotimes_{p=1}^{P-1} K_p\right)(K_P U)^\top\right) = \text{vec}\left(\left(\bigotimes_{p=1}^{P-1} K_p\right)(K_P U)^\top I_{n_P}\right) \tag{3.4}$$

$$= \left(I_{n_P} \otimes \left(\bigotimes_{p=1}^{P-1} K_p\right)\right)\text{vec}\left((K_P U)^\top\right) \tag{3.5}$$

The MVM can now be written as:

$$K\mathbf{u} = \text{vec}\left(\left(\text{reshape}\left(\left(I_{n_P} \otimes \left(\bigotimes_{p=1}^{P-1} K_p\right)\right)\text{vec}\left((K_P U)^\top\right), m/n_P, n_P\right)\right)^\top\right) \tag{3.6}$$

Repeating this procedure for the MVM $\left(I_{n_P} \otimes \left(\bigotimes_{p=1}^{P-1} K_p\right)\right)\text{vec}\left((K_P U)^\top\right)$ leads to

$$\left(I_{n_P} \otimes \left(\bigotimes_{p=1}^{P-1} K_p\right)\right)\text{vec}\left((K_P U)^\top\right) \tag{3.7}$$

$$= \text{vec}\left(\left(\text{reshape}\left(\left(I_{n_{P-1}} \otimes I_{n_P} \otimes \left(\bigotimes_{p=1}^{P-2} K_p\right)\right)\text{vec}\left((K_{P-1} U_2)^\top\right), m/n_{P-1}, n_{P-1}\right)\right)^\top\right) \tag{3.8}$$

where $U_2 = \text{reshape}\left((K_P U)^\top, m/n_{P-1}, n_{P-1}\right)$. By keeping repeating this procedure for all P components and noting that $\left(\bigotimes_{p=1}^{P} I_{n_p}\right)\mathbf{x} = \mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^m$, the MVM $K\mathbf{u}$ can be computed by Algorithm 1.

For the special case when $n_p = m^{1/P}$ for all $p = 1, ..., P$ the computational complexity is given by $\mathcal{O}(Pm^{\frac{P+1}{P}})$.

Algorithm 1 MVMs for Kronecker product

Input: K, list of matrices $K_p \in \mathbb{R}^{n_p \times n_p}$ for $p = 1, ..., P$, $m = \prod_{p=1}^{P} n_p$ and $\mathbf{u} \in \mathbb{R}^n$

Output: $\mathbf{b} \in \mathbb{R}^m$ where $\mathbf{b} = \left(\bigotimes_{p=1}^{P} K_p\right) \mathbf{u}$

1: **for** $p = P$ to 1 **do**
2: $U = \text{reshape}(u, n_p, m/n_p)$
3: $u = \text{vec}((K_p U)^\top)$
4: **end for**
return u

3.1.2 Toeplitz Structure

Given a stationary covariance function, the covariance matrix for a one-dimensional equidistant spaced grid is a Toeplitz matrix, which means that all the diagonals are constant.

There is no particularly efficient way to obtain a full eigendecomposition of a Toeplitz matrix, but fast operations can be obtained through the relationship with circulant matrices.

An $a \times a$ symmetric circulant matrix C is a symmetric Toeplitz matrix where the first column is given by a circulant vector $\mathbf{c} = [c_1, c_2, c_3, \cdots, c_3, c_2]^\top$ and each subsequent column is shifted one position from the next, i.e. $C_{i,j} = \mathbf{c}_{|j-i| \mod a}$. The eigendecomposition of a circulant matrix is given by $C = F^{-1}\text{diag}(F\mathbf{c})F$ where F is the discrete Fourier transform (DFT), $F_{jk} = \exp(-2jk\pi i/a)$. The log determinant of C can be computed from a single fast Fourier transform (FFT) which has a computational cost of $\mathcal{O}(a \log a)$ and a memory demand of $\mathcal{O}(a)$. Fast MVMs with C can be computed at the same asymptotic cost using two FFTs, one inverse FFT (IFFT) and one inner product.

Fast MVMs with symmetric Toeplitz matrices can be achieved by embedding an $m \times m$ symmetric Toeplitz matrix K into a $(2m-1) \times (2m-1)$ circulant matrix C with first column

$\mathbf{c} = [k_1, k_2, \cdots k_{m-1}, k_m, k_{m-1}, \cdots, k_2]^\top$ so that $K = C_{1:m,1:m}$ and

$$K\mathbf{y} = \left(C \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_{m \times 1} \end{bmatrix} \right)_{i=1:m} \tag{3.9}$$

The complete procedure for fast MVMs with Toeplitz matrices is shown in Algorithm 2 adapted from Tang et al [15].

Solving $K^{-1}\mathbf{y}$ can be achieved by an iterative procedure using only MVMs which has an asymptotic cost of $\mathcal{O}(m \log m)$. The log determinant and predictive variance require $\mathcal{O}(m^2)$ computations.

Algorithm 2 MVMs for Toeplitz matrices

Input: $\mathbf{k} \in \mathbb{R}^m$, first row of a symmetric Toeplitz matrix, and $\mathbf{y} \in \mathbb{R}^m$
Output: $\mathbf{b} \in \mathbb{R}^m$ where $\mathbf{b} = \text{toep}(\mathbf{k})\mathbf{y}$

1: $\mathbf{c} = [k_1, k_2, \cdots k_{m-1}, k_m, k_{m-1}, \cdots, k_2]^\top$
2: $\mathbf{u} = \begin{bmatrix} \mathbf{y} \\ \mathbf{0}_{m\times 1} \end{bmatrix}$
3: $\mathbf{f} = \text{FFT}(\mathbf{u})$
4: $\mathbf{g} = \text{FFT}(\mathbf{c})$
5: $\mathbf{h} = \mathbf{f} . * \mathbf{g}$
6: $\mathbf{x} = \text{IFFT}(\mathbf{h})$
return $\mathbf{z}[1:m]$

3.2 KISS-GP

The MSGP builds on the KISS-GP framework introduced by Wilson and Nickish in 2015 [16]. This method performs local kernel interpolation in combination with inducing point approximations and structure exploiting algebra. Using local kernel interpolation and inducing points, structured matrices can be used without requiring any structure in the input points.

3.2.1 Structured Kernel Interpolation

Given a set of m inducing points U the covariance matrix $K_{X,U}$ is approximated as $\tilde{K}_{X,U} = W_X K_{U,U}$ where W_X is an $n \times m$ matrix of interpolation weights. Given this approximation, any covariance matrix $K_{X,Z}$ can be approximated by $K_{X,Z} \approx \tilde{K}_{X,U} W_Z^\top$. This structured kernel interpolation (SKI) method leads to the fast approximate covariance:

$$k_{SKI}(\mathbf{x}, \mathbf{z}) = \mathbf{w_x} K_{U,U} \mathbf{w_z}^\top \tag{3.10}$$

The training covariance matrix has the approximation $K_{X,X} \approx W_X K_{U,U} W_X^\top = \tilde{K}_{X,X}$.

By performing local kernel interpolation, the interpolation matrices become extremely sparse. The inducing points can be chosen so that $K_{U,U}$ has either Toeplitz or Kronecker structure. This leads to close to linear scaling with the number of inducing points m for training the GP.

Several different local interpolation techniques can be used for one-dimensional inputs. Wilson and Nickish [16] tested local linear and local cubic interpolation for equidistant grids as well as linear inverse distance weighting for grids without equidistant spacing which can be combined with a k-means strategy for choosing the inducing points.

For a point u located between x_i and x_{i+1} the interpolation coefficients for piecewise linear interpolation are given by:

$$c_j = \begin{cases} \frac{|x_{j+1}-u|}{|x_{j+1}-x_j|}, & j = i \\ \frac{|x_{j-1}-u|}{|x_j-x_{j-1}|}, & j = i+1 \\ 0, & j \neq i, i+1 \end{cases} \tag{3.11}$$

This interpolation can be used both for non-equidistant and equidistant grids.

The local cubic interpolation used for equidistant grids is given by [17]:

$$c_j = \begin{cases} \frac{3}{2}|x_j - u|^3 - \frac{5}{2}|x_j - u|^2 + 1 & 0 < |x_j - u| < 1 \\ -\frac{1}{2}|x_j - u|^3 + \frac{5}{2}|x_j - u|^2 - 4|x_j - u| + 2, & 1 < |x_j - u| < 2 \\ 0, & |x_j - u| > 2 \end{cases} \tag{3.12}$$

In their experiments Wilson and Nickish find that SKI with cubic interpolation is more accurate for any given runtime than SKI with linear interpolation, FITC and SoR. They also conclude that cubic interpolation generally outperforms inverse distance weighting with k-means except for a small number of inducing points m.

In multiple dimensions, interpolation gets more complicated. For the product kernel case, the interpolation can be done separately for each dimension and then combined into one matrix. If the covariance between an input point $\mathbf{x}$ and an inducing point $\mathbf{u}^j$ is given by $k(\mathbf{x}, \mathbf{u}^j) = \prod_{d=1}^{D} k_d(x_d, u_d^j)$ and each one-dimensional function is approximated by $k_d(x_d, u_d^j) \approx \sum_{i=1}^{m} w_d^i k_d(u_d^i, u_d^j)$ where w_d^i are the corresponding interpolation weights, then the whole covariance is approximated by:

$$k(\mathbf{x}, \mathbf{u}^j) = \prod_{d=1}^{D} k_d(x_d, u_d^j) \approx \prod_{d=1}^{D} \sum_{i=1}^{m} w_d^i k_d(u_d^i, u_d^j) \tag{3.13}$$

3.2.2 Combining Kernel Interpolation and Structure Exploitation

As noted in the previous section, the inducing points can be chosen so that $K_{U,U}$ has Toeplitz or Kronecker structure. However, the structure exploitations described in Section 3.1 cannot be used directly since K is approximated by $K_{X,X} \approx W_X K_{U,U} W_X^\top$, hence the structure is not in the whole matrix as before, but only in the factor $K_{U,U}$. Therefore, the structure exploitations must be adapted to this new case.

Matrix Vector Multiplications

Fast MVMs with $\tilde{K}_{X,X} = W_X K_{U,U} W_X^\top$ can be obtained by first computing $\mathbf{y}' = W_X^\top \mathbf{y}$ and exploiting the sparseness of W_X, then computing $\mathbf{y}'' = K_{U,U}\mathbf{y}'$ by using Kronecker and/or Toeplitz structure and finally computing $\mathbf{y}''' = W_X \mathbf{y}''$ by again using the sparseness of W_X. According to Wilson et al. MVMs with sparse W cost $\mathcal{O}(n)$ and MVMs with $K_{U,U}$ exploiting Kronecker or Toeplitz structure are roughly linear in m.

Approximating Eigenvalues using Kronecker structure

There is no specially efficient way to compute the exact eigendecomposition of $\tilde{K}_{X,X} = W_X K_{U,U} W_X^\top$. However, the eigenvalues of $K_{U,U}$ can still be used to approximate the eigenvalues of $K_{X,X}$ by considering the eigenfunctions of the covariance function.

The eigenfunctions $\phi(\mathbf{x})$ of the covariance function $k(\mathbf{x}, \mathbf{x}')$ with respect to the measure μ were introduced in Section 2.1.2 and satisfies the following integral equation:

$$\int k(\mathbf{x}, \mathbf{x}')\phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda\phi(\mathbf{x}'), \quad \text{for } \mathbf{x}' \in \mathbb{R}^D \tag{3.14}$$

The measure μ corresponds to the probability density of the input points $p(\mathbf{x})$, which we can write as:

$$\int k(\mathbf{x}, \mathbf{x}')p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} = \lambda\phi(\mathbf{x}'), \quad \text{for } \mathbf{x}' \in \mathbb{R}^D \tag{3.15}$$

Williams and Seeger [18] observed that:

$$\int k(\mathbf{x}, \mathbf{x}')p(\mathbf{x})\phi(\mathbf{x})d\mathbf{x} \approx \frac{1}{n}\sum_{i=1}^{n} k(\mathbf{x}_i, \mathbf{x}')\phi(\mathbf{x}'), \quad \text{for } \mathbf{x}' \in \mathbb{R}^D \tag{3.16}$$

when each $\mathbf{x}_i$ is a sample from $p(\mathbf{x})$. This suggests that if $\lambda_1^X, \lambda_2^X, ..., \lambda_n^X$ is the eigenvalue spectrum of the covariance matrix K for the samples $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, then $1/n$ times the spectrum is an estimator of the n largest eigenvalues of the continuous covariance function. Indeed theory of numerical solutions of eigenvalue problems shows that $\frac{1}{n}\lambda_i^X \to \lambda_i$ for $i = 1, 2, 3....$ as $n \to \infty$.

If we assume that the m inducing points are samples from $p(\mathbf{x})$ as well, the eigenvalues λ_i^U of $K_{U,U}$ can also be used to estimate the m largest eigenvalues of the continuous problem by using $1/m$ times the eigenvalues spectrum of $K_{U,U}$. We can then use the approximation:

$$\lambda_i^X \approx \frac{n}{m}\lambda_i^U \tag{3.17}$$

for $i = 1, ..., \min(n, m)$ if both n and m are reasonably large and the assumption that the inducing points are samples from $p(\mathbf{x})$ is not too out of place.

To compute $\log|K_{X,X} + \sigma_n^2 I|$ we can then use:

$$\log|K_{X,X} + \sigma_n^2 I| \approx \sum_{i}^{\min(n,m)} \log\left(\frac{n}{m}\lambda_i^U + \sigma_n^2\right) \tag{3.18}$$

Note that this approximation has nothing to do with the choice of interpolation, but only with the choice of inducing points.

3.3 Fast Test Predictions

The predictive mean is approximated by $\mathbb{E}[\mathbf{f}_*] \approx K_{X_*,X}\tilde{\boldsymbol{\alpha}}$ where $\tilde{\boldsymbol{\alpha}} = (\tilde{K}_{X,X} + \sigma_n^2 I)^{-1}\mathbf{y}$ which is computed already during the training. Furthermore, structured kernel interpolation can be applied to $K_{X_*,X}$ to give:

$$\mathbb{E}[\mathbf{f}_*] \approx \mathbb{E}[\tilde{\mathbf{f}}_*] = \tilde{K}_{X_*,X}\tilde{\boldsymbol{\alpha}} = W_{X_*}K_{U,U}W_X^\top\tilde{\boldsymbol{\alpha}} \tag{3.19}$$

Since $K_{U,U}W_X^\top\tilde{\boldsymbol{\alpha}}$ is computed during training, the only computation needed at test time is the multiplication with sparse W_{X_*} which leads to $\mathcal{O}(1)$ operations per test point.

The predictive variance of the GP is given by:

$$\mathbf{v}_* = \text{diag}[\text{Cov}(\mathbf{f}_*)] = \text{diag}(K_{X_*,X_*}) - \boldsymbol{\nu}_* \tag{3.20}$$

where $\boldsymbol{\nu}_* = \text{diag}(K_{X_*,X}[K_{X,X} + \sigma_n^2 I]^{-1}K_{X,X_*})$ can be approximated by local interpolation:

$$\boldsymbol{\nu}_* \approx W_*\tilde{\boldsymbol{\nu}}_U, \quad \text{where } \tilde{\boldsymbol{\nu}}_U = \text{diag}(\tilde{K}_{U,X}A^{-1}\tilde{K}_{X,U}) \text{ and } A = \tilde{K}_{X,X} + \sigma_n^2 I \tag{3.21}$$

As for the mean, once $\tilde{\boldsymbol{\nu}}_U$ is precomputed, the only computation needed is the multiplication with sparse W_* which leads to $\mathcal{O}(1)$ operations per test point. To efficiently compute $\tilde{\boldsymbol{\nu}}_U$ a stochastic estimator explained in [19] can be used since $\tilde{\boldsymbol{\nu}}_U$ is the variance of $\tilde{K}_{U,X}\mathbf{r}$ where $\mathbf{r} \sim \mathcal{N}(\mathbf{0}, A^{-1})$. The stochastic estimator is computed by sampling n_s samples $\mathbf{g}_i^m \sim \mathcal{N}_m(\mathbf{0}, I)$ and $\mathbf{g}_i^n \sim \mathcal{N}_n(\mathbf{0}, I)$ for $i = 1, ..., n_s$ and then solving $A\mathbf{r}_i = W_X V\sqrt{E}V^\top \mathbf{g}_i^m + \sigma_n \mathbf{g}_i^n$ for $\mathbf{r}_i$ using LCG. The eigendecomposition $K_{U,U} = VEV^\top$ is computed efficiently by exploiting structure in the matrix. The estimate is finally computed by:

$$\mathbf{v}_* \approx \hat{\mathbf{v}}_* = \max\left[\mathbf{0}, \mathbf{k}_* - W_*\left(\frac{1}{n_s}\sum_{i=1}^{n_s}(\tilde{K}_{U,X}\mathbf{r}_i)^2\right)\right] \tag{3.22}$$

where the square is taken element-wise. The relative error is according to Papandreou and Yuille [19] given by $r = \sqrt{2/n_s}$ which gives a relative error of approximately 32% for $n_s = 20$.

3.4 Circulant Approximation

If U is a regularly spaced multidimensional grid and a stationary product covariance function is used, then the covariance matrix $K_{U,U}$ can be decomposed as a Kronecker product of Toeplitz matrices $K_{U,U} = T_1 \otimes \cdots \otimes T_D$.

As mentioned in Section 3.1.2, there is no particularly efficient way to obtain a full eigendecomposition of a Toeplitz matrix, but fast operations can be obtained through the relationship with circulant matrices.

LCG for solving linear Toeplitz systems can be sped up by using circulant pre-conditioners which act as approximate inverses. To get a good approximation the distance between the pre-conditioner C and the Toeplitz matrix K should be minimized. Three classical pre-conditioners are $C_{Strang} = \arg\min_C ||I - C^{-1}K||_1$, $C_{T.Chan} = \arg\min_C ||C - K||_F$ and $C_{Tyrtyshnikov} = \arg\min_C ||I - C^{-1}K||_F$. Another alternative is to use the Whittle approximation $circ_{Whittle}(\mathbf{k})$:

$$[circ_{Whittle}(\mathbf{k})]_i = \sum_{j=-w}^{w} k_{i+jm} \tag{3.23}$$

$$c(t) = \sum_{j=-w}^{w} k(t + jm\Delta u) \tag{3.24}$$

In [5] the log determinant is then approximated by:

$$\log|\text{toep}(\mathbf{k}) + \sigma_n^2 I| \approx \mathbf{1}^\top \log(F\mathbf{c} + \sigma_n^2 \mathbf{1}) \tag{3.25}$$

where $\mathbf{c} = F^H \max(F\text{circ}(\mathbf{k}), \mathbf{0})$, F^H is the conjugate transpose of the Fourier transform matrix and circ($\mathbf{k}$) is one of the proposed circulant approximations. The choice of circulant approximation can have a large effect on the performance and Wilson and Nickish verified in an empirical comparison that the Whittle approximation yields consistently accurate approximation results.

However, to be able to combine the circulant approximation with the eigenvalue spectrum approximation and the Kronecker structure in the multidimensional case, we can instead use the eigenvalues of the circulant approximation given by Fcirc($\mathbf{k}$) directly to

compute the approximate eigenvalues of the complete covariance matrix. To make sure the approximation is still positive definite $\max[\text{Re}(F\text{circ}(\mathbf{k})), \mathbf{0}]$ can be used as approximate eigenvalues.

3.4.1 Extension to Multivariate Data

The circulant approximation can be extended to multivariate data. When using a translation invariant covariance function on input points on a regular grid of size $n_1 \times n_2 \times \cdots \times n_D$ the covariance matrix is a symmetric block-Toeplitz matrix with Toeplitz blocks (BTTB). Using a dimension-wise circulant embedding of size $(2n_1 - 1) \times (2n_2 - 1) \times \cdots \times (2n_D - 1)$, fast MVMs can be achieved using the multi-dimensional Fourier transform $F = F_1 \otimes F_2 \otimes \cdots \otimes F_D$ by applying Fourier transforms F_d along each dimension. The Whittle approximation can also be generalised by summing over $(2w + 1)^D$ terms which leads to $C_{U,U}$ being a block-circulant matrix with circulant blocks (BCCB). Such a matrix has eigendecomposition $C_{U,U} = F^H(F\mathbf{c})F$, where $\mathbf{c} \in \mathbb{R}^n$ is the Whittle approximation to $\mathbf{k} \in \mathbb{R}^n$, $n = n_1 \cdot n_2 \cdot ... \cdot n_D$. Hence, exploiting the BTTB structure allows for efficiently dealing with multivariate data without requiring a factorizing covariance function.

3.5 Projections

Placing the inducing points onto a multidimensional Cartesian product grid so that the covariance matrix has Kronecker structure leads to the total number of inducing points m growing exponentially with the number of grid dimensions. To still be able to use the approach for high dimensional data input projections can be used.

Here we assume that the inducing points live in a lower $d < D$ dimensional space and are related to the input points by $\mathbf{u} = h(\mathbf{x}, \boldsymbol{\omega})$ where the parameters of the mapping $\boldsymbol{\omega}$ can be determined by maximizing the marginal likelihood of the GP. In the case of linear projections we get $P\mathbf{x} = \mathbf{u}$ where $P \in \mathbb{R}^{d \times D}$. The covariance functions effectively become $k(\mathbf{x}, \mathbf{x}') \to k_d(P\mathbf{x}, P\mathbf{x}')$, $k(\mathbf{x}, \mathbf{u}') \to k_d(P\mathbf{x}, \mathbf{u}')$ and $k(\mathbf{u}, \mathbf{u}') \to k_d(P\mathbf{u}, P\mathbf{u}')$ where $k_d(\cdot, \cdot)$ is the covariance function in the d dimensional space.

The entries of P are treated as hyperparameters of the marginal likelihood and can be learned through marginal likelihood maximization. To avoid optimization issues caused by degeneracies between P and the other hyperparameters, P can be constrained to be orthonormal or have unit scaling.

Chapter 4

Experiments

The purpose of the experiments is to evaluate MSGP, and to this aim different parts of the framework are tested independently and then the complete framework is compared to other GP methods.

MSGP was implemented in Python using the libraries TensorFlow [20] and GPflow [21]. TensorFlow is an open source software library for numerical computation using data flow graphs which was originally developed by the Google Brain Team and GPflow is a package for building GP models in Python using TensorFlow. MSGP was implemented as a new GP model in GPflow which is trained by using the optimization in GPflow. This optimization maximizes the marginal likelihood with respect to the hyperparameters by using either a SciPy [22] or a TensorFlow optimizer. The gradients of the objective function are computed by TensorFlow.

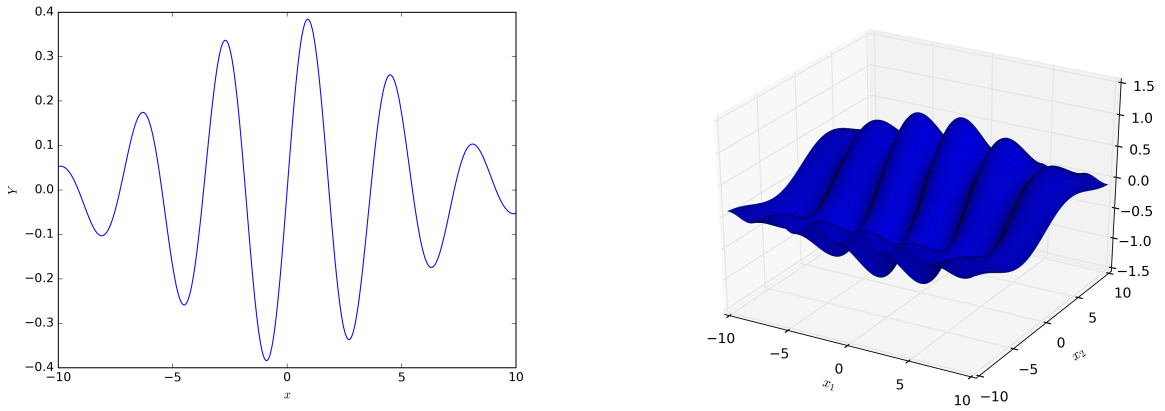
4.1 Datasets

For the tests of the different parts of MSGP synthesized generated data are used since this provides more flexibility in the choice of data. The data is generated by first sampling the input data X from the uniform distribution $\mathcal{U}(-10, 10)$ and then the output data $\mathbf{y}$ is determined by transforming the input data using an analytical function and adding Gaussian noise. The function was chosen to be:

$$f(x) = 4\left(\sum_{d=1}^{D} w_{1,d}\sin(4w_{2,d}x_d)\right)\exp\left(-||x||^2/(50D)\right) \tag{4.1}$$

where $x \in \mathbb{R}^D$ and $w_{i,j} \sim \mathcal{U}(-1, 1)$ for $i = 1, 2$ and $j = 1, ..., D$. To get the targets $y(x)$ independent Gaussian noise was added giving $y(x) = f(x) + e_x$ where $e_x \sim \mathcal{N}(0, 0.01^2)$. This non-linear function was chosen to be similar to the one-dimensional function used by Wilson et al. [5], $f(x) = \sin(x)\exp\left(\frac{-x^2}{2\times 5^2}\right)$, for $D = 1$ but is also generalizable to the multidimensional case $D > 1$. In Figure 4.1 examples of the function $f(x)$ are shown in one and two dimensions.

For the comparison with other GP models real datasets will be used. The chosen real datasets are `SARCOS`, `KIN40K` and `Abalone` which all have been used in the literature of GP methods. The properties of the datasets are summarized in Table 4.1 adapted from [1].



(a) Plot of $f(x)$ in one dimension

(b) Plot of $f(x)$ in two dimensions

Figure 4.1: Plots of the function used to generate data

Dataset	SARCOS	KIN40K	Abalone
Input dimension (D)	21	8	8
Training set size (n)	44 484	10 000	3 133
Test set size (n_*)	4 449	30 000	1 044

Table 4.1: Input dimension, training set size and test set size for the real datasets

SARCOS is a dataset that represents the inverse dynamics of a robot arm. It has 21 input dimensions and is a highly non-linear regression problem. This dataset has been used by Rasmussen and Williams [9], Snelson [1] as well as Titsias [11].

KIN40K is another highly non-linear dataset where the object is to predict the distance of a robot arm head from a target given 8 input dimensions. This dataset has been used by Schwaighofer and Tresp [23], Seeger et al. [24], Snelson [1] and Titsias [11].

Abalone is a smaller dataset with the object to predict the age of abalone given 8 physical measurements. This dataset has been used by Willams and Seeger [25], Snelson [1] and Titsias [11].

4.2 Performance Criteria

To evaluate the performance of the predictions made by the models, three different performance metrics are used: the standardized mean squared error (SMSE) [9], the standardized mean absolute error (SMAE) [5] and the mean standardized log loss (MSLL) [9]. MSLL is also called the standardized negative log probability density (SNLP) [11].

SMSE is given by:

$$SMSE = \frac{1}{n_*} \frac{||\mathbf{y}_* - \bar{\mathbf{f}}_*||^2}{\text{Var}(\mathbf{y}_*)} \tag{4.2}$$

where n_* is the number of test points, $\bar{\mathbf{f}}_*$ are the mean predictions at the test points and $\mathbf{y}_*$ are the true targets at the test points.

SMAE is computed by:

$$SMAE = \frac{\sum_{i=1}^{n_*} |y_*^i - \bar{f}_*^i|}{\sum_{i=1}^{n_*} |y_*^i - \frac{1}{n_*} \sum_{j=1}^{n_*} y_*^i|} \tag{4.3}$$

Both SMSE and SMAE are approximately 1 for the trivial method of always guessing the mean of the training points and below 1 for better methods but never less than zero.

MSLL is obtained by considering the negative log probability of the targets given the model:

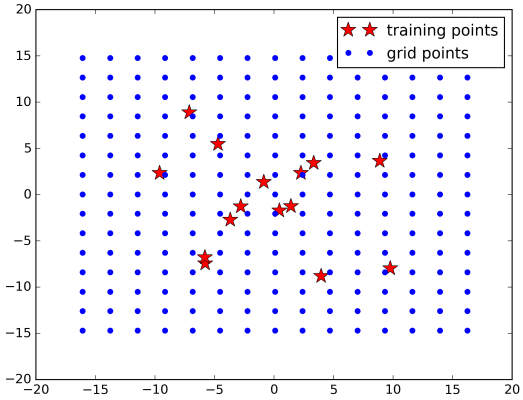
$$-\log p(y_*|\mathcal{D}, \mathbf{x}_*) = \frac{1}{2}\log(2\pi\sigma_*^2) + \frac{(y_* - \bar{f}(\mathbf{x}_*))^2}{2\sigma_*^2} \tag{4.4}$$

where σ_*^2 is the predictive variance of the model. This loss is standardized by subtracting the negative log probability of the target under the trivial model which predicts using a Gaussian distribution with mean and variance of the training data. MSLL is computed by taking the mean value of the negative standardized log likelihood for all test points. This means that MSLL is approximately zero for simple methods and negative for better methods.

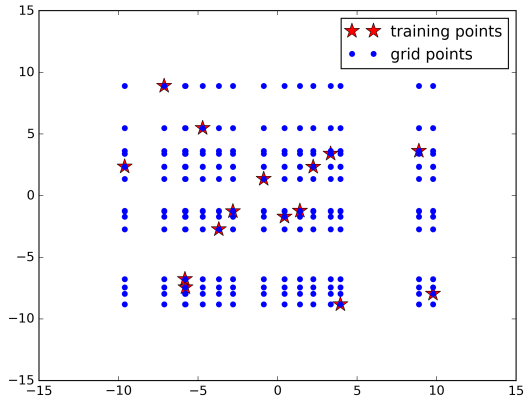
4.3 Choice of Inducing Points

In MSGP, the locations of the inducing points are not optimized, as they are in for example FITC or VFE. Therefore, the choice of the locations of the inducing points has to be made carefully. In the experiments the grid was created so that the end points cover all the training points with some marginal. The intermediate grid points were either placed equidistantly or by using k-means over the location of the training points in that dimension depending on if the interpolation method requires an equidistant grid or not. In the case that the number of inducing points per dimension is larger than the number of training points, the k-means method was replaced since it does not make sense to use k-means with more cluster centres than points. Instead grid points were placed exactly on all the training points and the number of inducing points per dimension was reduced to the number of training points since more inducing points seem unnecessary.

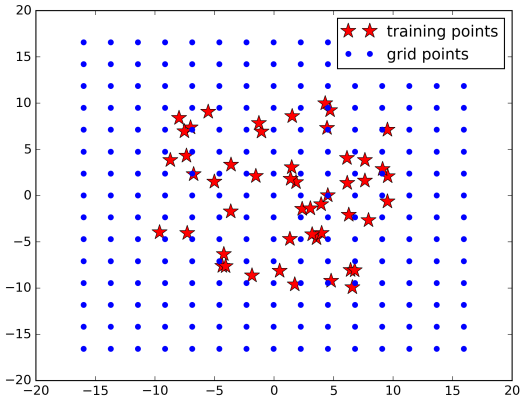
Examples of created grids in two dimensions are shown in Figure 4.2. In Figure 4.2a an equidistant grid with 15 grid points per dimension is created for 15 training points, while in Figure 4.2b an non-equidistant grid with 15 grid points per dimension is created for 15 training points so that every training point lies exactly on some grid point. In Figure 4.2c an equidistant grid with 15 grid points per dimension is created for 50 training points, while in Figure 4.2d an non-equidistant grid with 15 grid points per dimension is created for 50 training points using k-means.



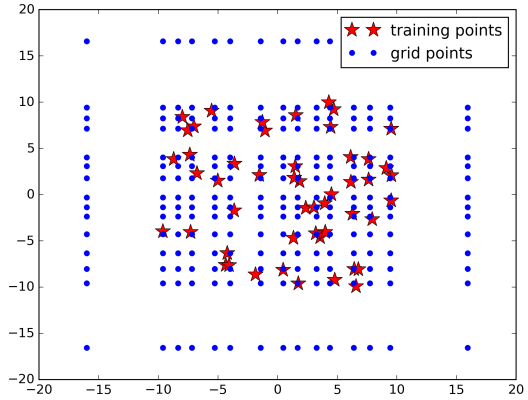
(a) Equidistant grid for 15 training points



(b) Non-equidistant grid for 15 training points



(c) Equidistant grid for 50 training points



(d) Non-equidistant grid for 50 training points

Figure 4.2: Plots of created equidistant and non-equidistant grids in two dimensions when using $n = 15$ or $n = 50$ training points and 15 grid points in each dimension.

4.4 Interpolation Method

For all experiments evaluating the interpolation methods, MSGP is used with the three different kinds of interpolation described in Section 3.2.1: piecewise linear interpolation for non-equidistant grids where the grid points are chosen with k-means, piecewise linear interpolation for equidistant grids and piecewise cubic interpolation for non-equidistant grids. The performances of these versions of MSGP are compared to the performance of a regular full GP.

Three kinds of experiments were performed for the interpolation methods: observing the behaviour when changing the number of inducing points m, observing the behaviour when changing the number of training points n and observing the behaviour when changing the input dimension D.

For all the interpolation experiments with input dimension $D = 1$, the covariance function used was an SE covariance function with length-scale l set to 1 for the regular GP and to max of 1 and the maximum grid space for MSGP. The variance of the covariance function was set to $\sigma_f = 1$ and the variance of the noise was set to $\sigma_n = 0.001$.

Toeplitz MVMs and the Whittle circulant approximation with truncation $w = 3$ were used for the computations as well as the eigenvalue spectrum approximation.

For the experiments where the input dimension was not always one, the covariance function used was an ARD SE covariance function, which can be seen as a generalization of the one-dimensional SE covariance function and can be used with MSGP since it is a stationary product covariance function. The length-scale for each dimension was set to 1 for the regular GP and for the MSGP it was set to max of 1 and the maximum grid space in the corresponding dimension. The variance of the covariance function was set to $\sigma_f = 1$ for each dimension and the variance of the noise was set to $\sigma_n = 0.001$. Kronecker MVMs and the eigenvalue spectrum approximation was used for computations.

Number of inducing points

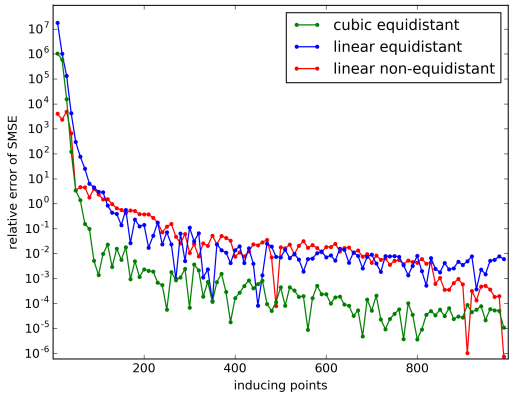
The first part of the experiments for the number of inducing points is smaller problems where it is possible to compute the approximated covariance matrix $K_{SKI} = WK_{U,U}W^\top$ without running out of memory. This was done first for one dimensional input data, $D = 1$, with the number of training points set to $n = 1000$ and the number of test points set to $n_* = 100$. The number of inducing points was increased from 10 to 1000 with a step size of 10.

In Figure 4.3 results are shown for this experiment. From these plots we can see that the non-equidistant linear interpolation with k-means performs better than equidistant linear and cubic interpolation for a very small number of inducing points, since then the covariance matrix is better approximated and the relative error of the SMSE for the test points compared to the regular GP is lower, see Figures 4.3b and 4.3a. However, the performances of all interpolation techniques are very bad compared to the regular GP when using few inducing points. With an increasing number of inducing points, equidistant cubic interpolation approximates the covariance matrix even better than non-equidistant linear interpolation.

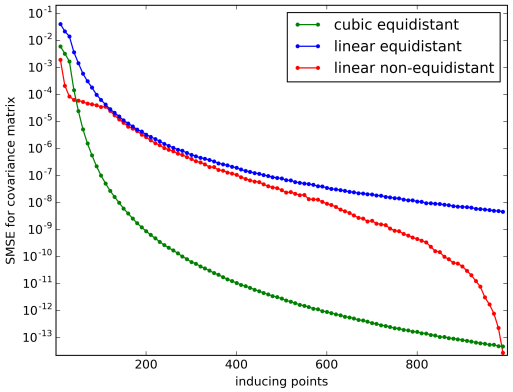
The computed log likelihood shown in Figure 4.3d is severely underestimated for all the interpolation methods when the number of inducing points is much smaller than the number of training points even though the approximation of the covariance matrix is good as seen in Figure 4.3b. The error in the log likelihood estimation does not have to do with the interpolation, but rather with the eigenvalue spectrum approximation which performs badly for few inducing points. The eigenvalue spectrum approximation is examined in more detail in Section 4.5.

A similar experiment was performed for input points in two dimensions. Again the number of training points was set to $n = 1000$ and the number of test points set to $n_* = 100$. The number of inducing points was increased from 10 to 100 per dimension with a step size of 10. This means that the total number of inducing points was increased from $m = 10^2 = 100$ to $m = 100^2 = 10\,000$.

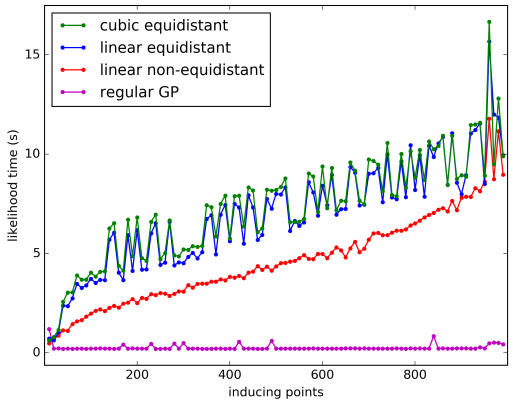
In Figure 4.4 results of this experiment are shown. From the plots of the relative error of the SMSE of the test points and the SMSE of the approximated covariance matrix in Figures 4.4a and 4.4b, we see a similar behaviour to the earlier experiment for $D = 1$. Again the non-equidistant linear interpolation performs better for few inducing points, but the performance is still bad when using few inducing points compared to the regular GP. For a larger number of inducing points, we see again that the cubic interpolation approximates the covariance matrix better than non-equidistant linear interpolation.



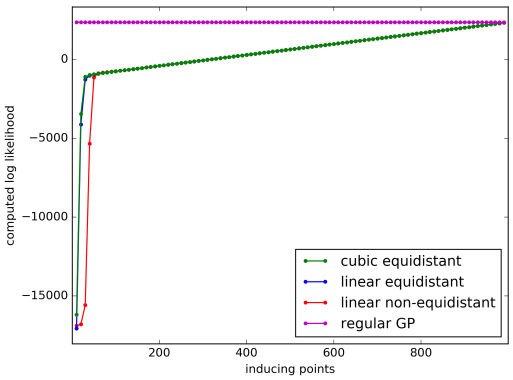
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of SMSE for the approximated covariance matrix



(c) Plot of time for computing the likelihood



(d) Plot of computed log likelihood

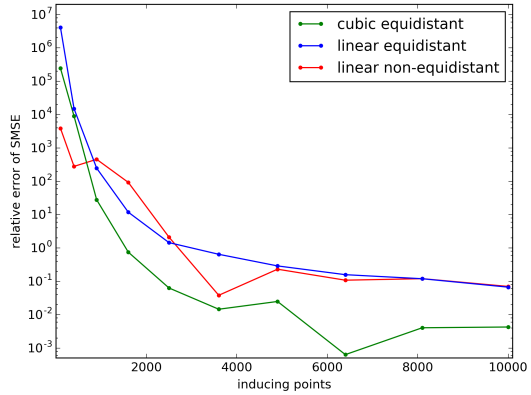
Figure 4.3: Plots of the results using the different interpolation methods when changing the number of inducing points. Here the input dimension was $D = 1$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

For the computed log likelihood in Figure 4.4d we have the same behaviour as for the previous experiment. The log likelihood is again severely underestimated for all the interpolation methods when the total number of inducing points is much smaller than the number of training points due to the eigenvalue spectrum approximation.

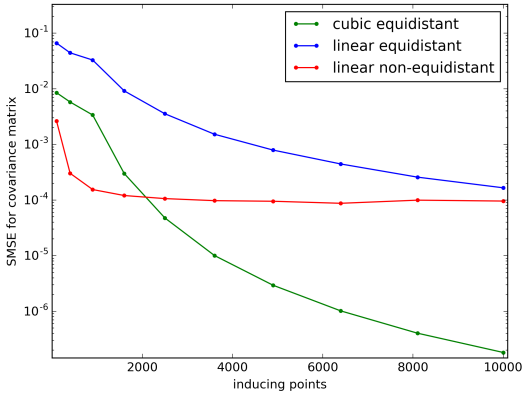
From these smaller experiments, we see that the number of inducing points should be large to get good approximations. The next set of experiments is intended to show the behaviour of the approximations for larger number of inducing points to be able to see how large a number of inducing points is "large enough" to get good performance.

The first results in Figure 4.5 are for $D = 1$, $n = 1000$, $n_* = 100$ and m increasing from 100 to 4000. We can from the figures see that when $m \geq n$ the non-equidistant linear interpolation performs basically as good as the regular GP for SMSE (4.5a) and MSLL (4.5b) as well as the computed log likelihood (4.5d). This is due to the fact that then all the training points are placed exactly at one of the inducing grid points. This is not the case for the equidistant interpolations, but the cubic interpolation still performs very well and considerably better than the linear equidistant interpolation for SMSE and MSLL.

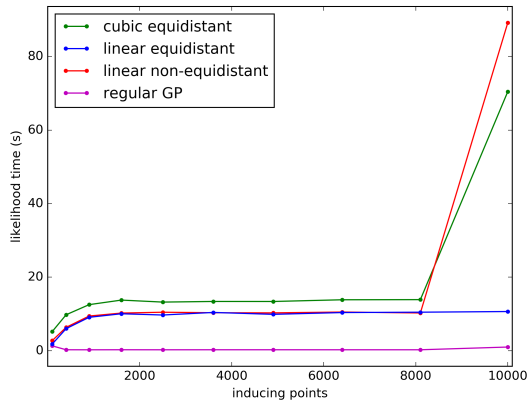
For the computed log likelihood the relative errors for the linear and cubic inter-



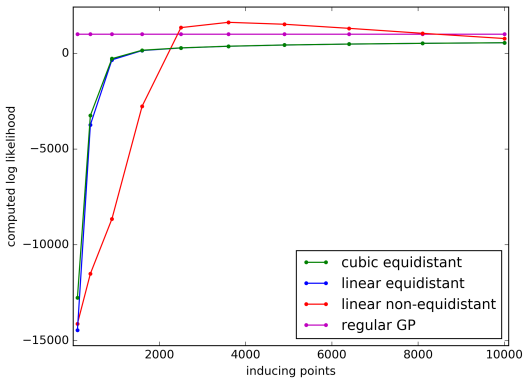
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of SMSE for the approximated covariance matrix



(c) Plot of time for computing the likelihood



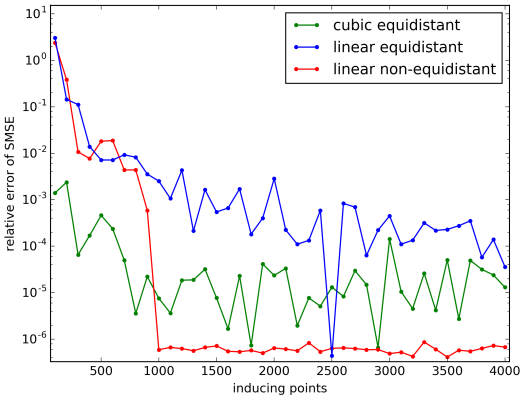
(d) Plot of the computed log likelihood

Figure 4.4: Plots of the results using the different interpolation methods when changing the number of inducing points. Here the input dimension was $D = 2$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

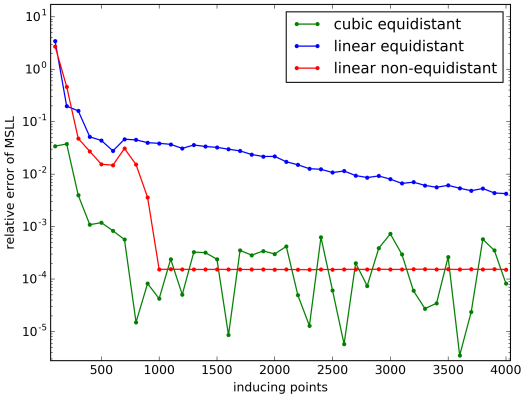
polations are indistinguishable. From Figure 4.5c we see that the time for computing the likelihood is similar for all interpolation methods, and does not increase much with an increasing number of inducing points. The computational time for the regular GP is much lower than for the interpolation methods in this case, but it should be noted that $n = 1000$ is not very large and for this number of training points the regular GP is still reasonable to use.

The last interpolation experiment with a changing number of inducing points is for $D = 2$ input dimensions, the number of training points was set to $n = 1000$ and the number of test points was $n_* = 100$. The number of inducing points was increased from 500 to $4\,000$ points per dimension with a step size of 500. This means that the total number of inducing points was increased from $m = 500^2 = 25\,000$ to $m = 4\,000^2 = 16\,000\,000$.

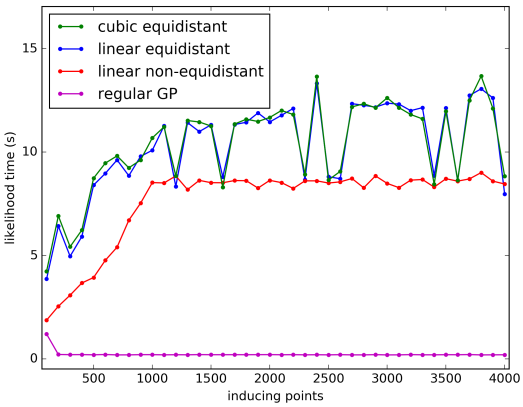
In Figure 4.6 results are shown. In Figures 4.6a and 4.6b the relative errors of the SMSE and MSLL for the MSGP compared to the SMSE and MSLL of the regular GP are shown. We see that the accuracy of the predictions does not depend notably on the number of inducing points m for the cubic equidistant interpolation method. The errors for the linear equidistant interpolation method keeps decreasing with more inducing points,



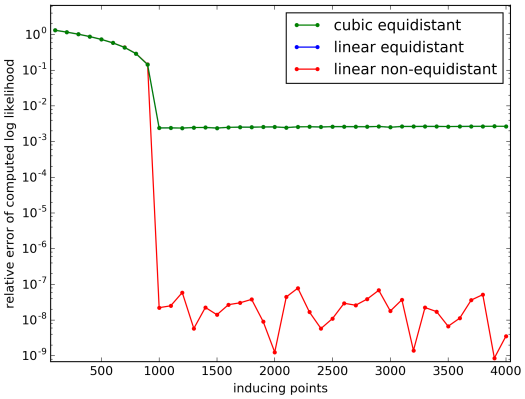
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points



(c) Plot of time for computing the likelihood



(d) Plot of relative error of the computed log likelihood

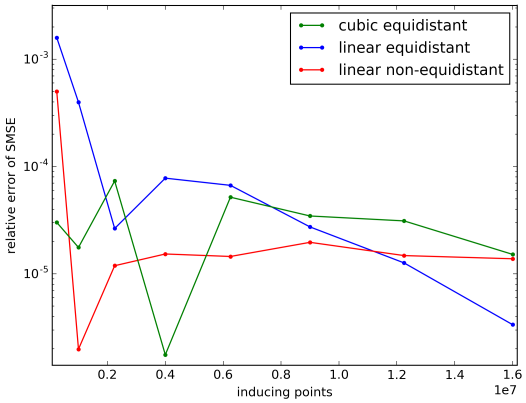
Figure 4.5: Plots of the results using the different interpolation methods when changing the number of inducing points. Here the input dimension was $D = 1$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

but the linear equidistant interpolation is almost always worse than the cubic interpolation, especially for the MSLL. The linear non-equidistant interpolation covers all the training points exactly when we have 1000 inducing points per dimension, and the performance does not get any better than that.

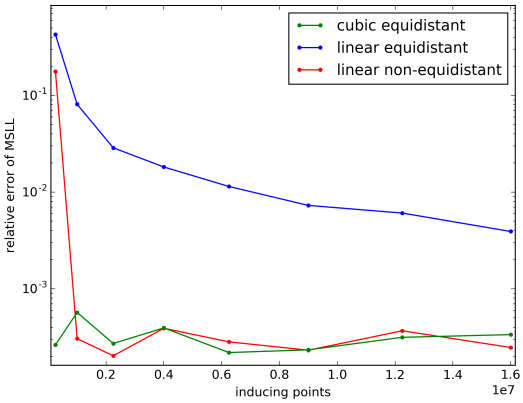
In Figure 4.6c the time for computing the likelihood is shown in a loglog-scale. As expected we can see that the regular GP is close to constant since it does not use the inducing points. The two equidistant interpolations, both linear and cubic, have very similar computational times, while the linear non-equidistant interpolation should be constant for more than 1000 inducing points per dimension, but for some reason, probably connected to the performance of the machine the experiment was run at, two values do not follow this.

The values of the computed log likelihood is shown in Figure 4.6d. The plot for the linear equidistant interpolation is hard to distinguish from the cubic interpolation since the error of the likelihood in a large part comes from the error of the eigenspectrum approximation which is exactly the same for the two equidistant grids. We see that all the

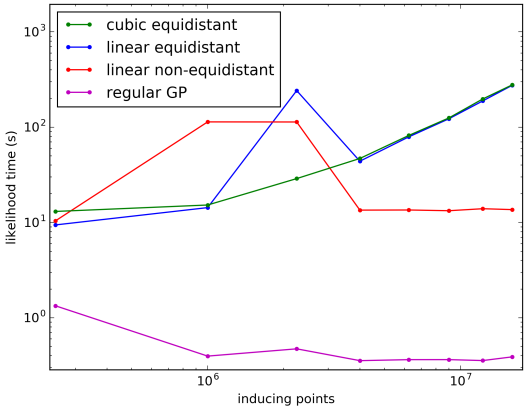
interpolation methods underestimates the log likelihood, this underestimation is due to the fact that the number of training points is always 1000 which is too low to get a better eigenspectrum approximation independent of the number of inducing points.



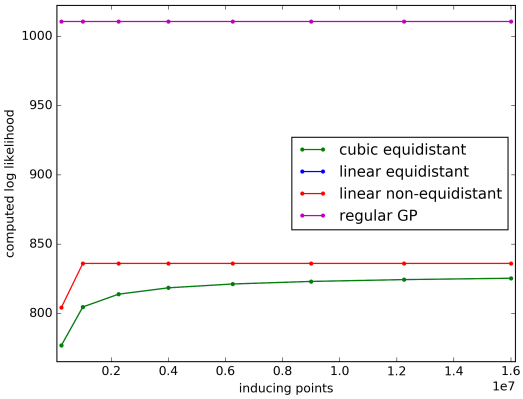
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood



(d) Plot of the computed log likelihood

Figure 4.6: Plots of the results using the different interpolation methods when changing the number of inducing points. Here the input dimension was $D = 2$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

Number of training points

From the previous experiments we have seen that all the interpolation methods seem to give decent results as long as the number of inducing points m is large enough. In this part we now want to see how the performance depends on the number of training points n.

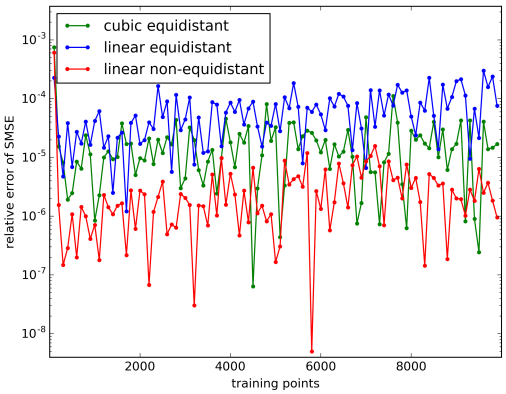
The first experiment is for $D = 1$ input dimension. The number of inducing points was set to $m = 10000$ and the number of test points was $n_* = 100$. The number of training points was increased from 100 to 10 000 with a step size of 100. This means that each training point is always located exactly at a grid point for the non-equidistant grid used in MSGP.

In Figure 4.7 results are shown. In Figure 4.7a and 4.7b the relative error of the SMSE

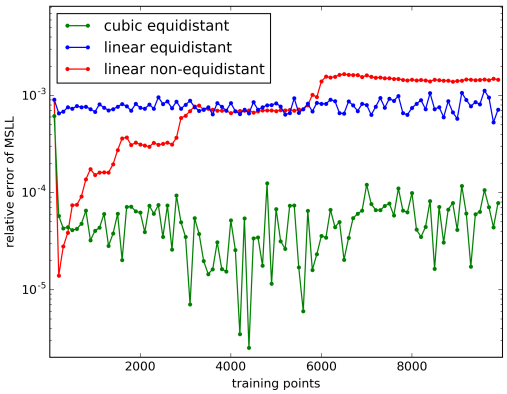
and MSLL for the MSGP compared to the SMSE and MSLL of the regular GP is shown. We can see that the performance does not depend notably on the number of training points for the equidistant interpolations, but for the non-equidistant linear interpolation the relative error of the MSLL actually gets a bit worse with increasing number of training points.

In Figure 4.7c the time for computing the likelihood is shown in a loglog-scale. We can see that the regular GP as well as the MSGP with non-equidistant grid, have a much larger computational complexity than the equidistant grids. This is as expected since only the equidistant grids can exploit Toeplitz structure optimizations.

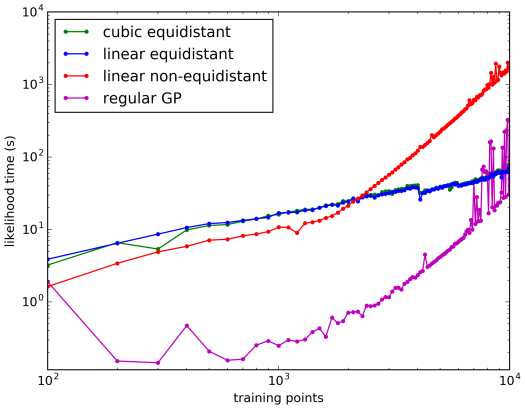
The relative errors of the computed log likelihood for the MSGP compared to the computed log likelihood of the regular GP are shown in Figure 4.7d. As for the previous experiments, the plot for the cubic interpolation is hard to distinguish since the error of the likelihood in a large part comes from the error of the eigenspectrum approximation which is exactly the same for the two equidistant grids. We see that the log likelihood approximation slowly gets better with the number of training points for the equidistant grids.



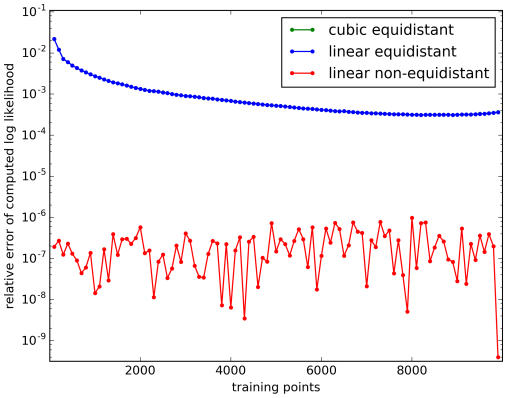
(a) Plot of relative error of SMSE for test points



(b) Plot of relative error MSLL for the test points



(c) Plot of time for computing the likelihood



(d) Plot of relative error of computed log likelihood

Figure 4.7: Plots of the results using the different interpolation methods when changing the number of training points. Here the input dimension was $D = 1$, the number of inducing points $m = 10000$ and the number of test points $n_* = 100$.

The second interpolation experiment with a changing number of training points is for $D = 2$ input dimensions. The number of inducing points was set to 1000 per dimension, so in total $m = 1\,000\,000$ and the number of test points was $n_* = 100$. The number of training points was increased from 100 to $14\,600$ with a step size of 500.

In Figure 4.8 results are shown. In Figures 4.8a and 4.8b the relative errors of the SMSE and MSLL for the MSGP compared to the SMSE and MSLL of the regular GP are shown. We can see that the accuracy of the predictions does not depend particularly on the number of training points n for either of the interpolation methods as long as $n > 1000$. For both the SMSE and the MSLL cubic equidistant interpolation performs considerably better than the two linear interpolations.

The time for computing the likelihood is shown in a loglog-scale in Figure 4.8c. As expected we can see that the regular GP, which can not use Kronecker structure optimizations, has a much larger computational complexity than the interpolation methods which all exploit Kronecker structure optimizations. The two linear interpolations, both equidistant and non-equidistant, have very similar computational times, while the cubic interpolation is a bit slower, due to the fact that the sparse interpolation matrix has more non-zero entries which leads to slower MVMs.

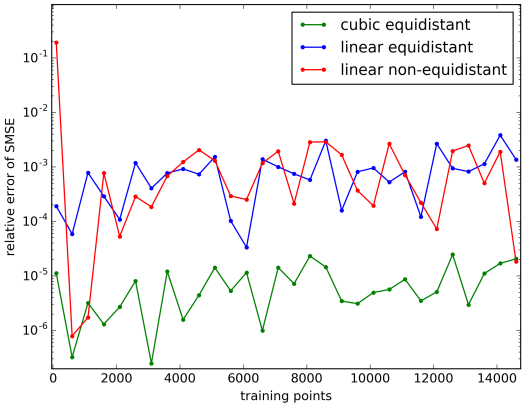
In Figure 4.8d we see the relative error of the computed log likelihood for the MSGP compared to the computed log likelihood of the regular GP in a loglog-scale. As before, the plot for the linear equidistant interpolation is hard to distinguish since the error of the likelihood in a large part comes from the error of the eigenspectrum approximation which is exactly the same for the two equidistant grids. We see that the relative error of the log likelihood approximation decreases with the number of training points for all the interpolation methods, which is due to the fact that the eigenspectrum approximation gets better with more training points.

Input dimension

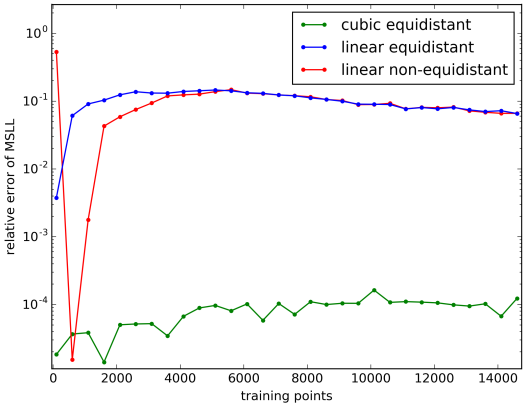
The last interpolation experiment is for changing the input dimension D. The number of training points was set to 500, the number of inducing points was set to 50 per dimension, so in total $m = 50^D$ and the number of test points was $n_* = 100$. The reason so few inducing points per dimension and training points were used, is that the memory requirements are linear in the number of inducing points and the number of training points for MSGP. This means that for $D = 5$ input dimensions we get $m = 50^D = 3.125 \cdot 10^8$ which is already a large number and for more inducing points per dimension this quickly gets too large for our machine.

In Figure 4.9 results are shown. In Figures 4.9a and 4.9b the relative errors of the SMSE and MSLL for the MSGP compared to the SMSE and MSLL of the regular GP are shown. We can see that the accuracy of the predictions gets better when the dimension increases. With a higher dimension more inducing points are used to approximate each training point, 2^D inducing points are used for the linear interpolations and 4^D points for the cubic interpolation. For both the SMSE and the MSLL cubic equidistant interpolation performs considerably better than the two linear interpolations for all dimensions and the linear equidistant interpolation performs better than the non-equidistant one for $D > 2$.

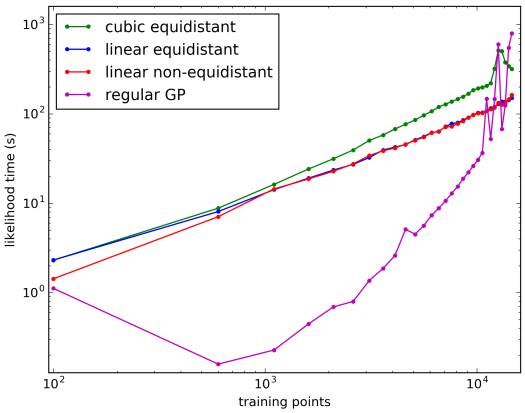
The time for computing the likelihood is shown in Figure 4.9c. As expected we can see that the regular GP is close to constant with the number of input dimensions since



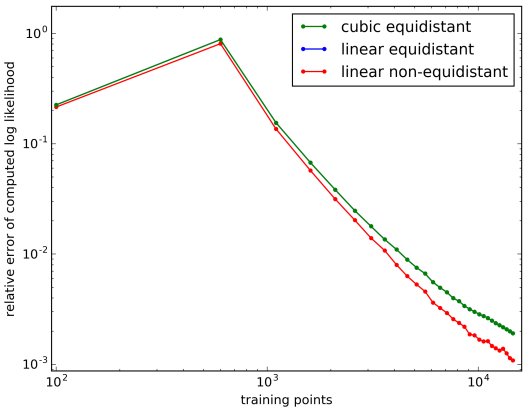
(a) Plot of relative error of SMSE for test points



(b) Plot of relative error MSLL for the test points



(c) Plot of time for computing the likelihood

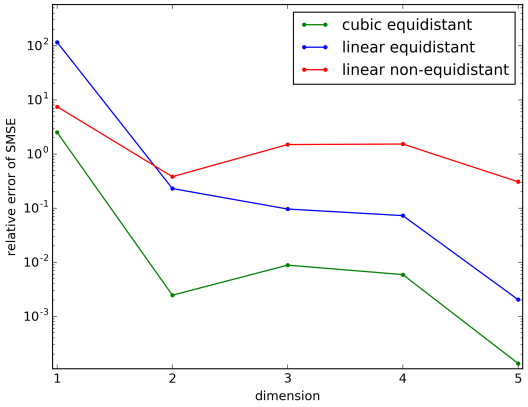


(d) Plot of relative error of computed log likelihood

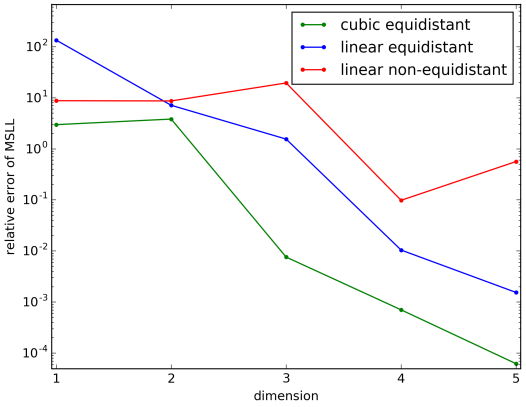
Figure 4.8: Plots of the results using the different interpolation methods when changing the number of training points. Here the input dimension was $D = 2$, the number of inducing points was 1000 per dimension, so in total $m = 1\,000\,000$ and the number of test points $n_* = 100$.

the computational complexity does not depend on the input dimension D for the regular GP. The different interpolation methods all exhibit similar behaviours for the computational time, however it is not possible to draw any conclusions about the exact computational complexity from this experiment.

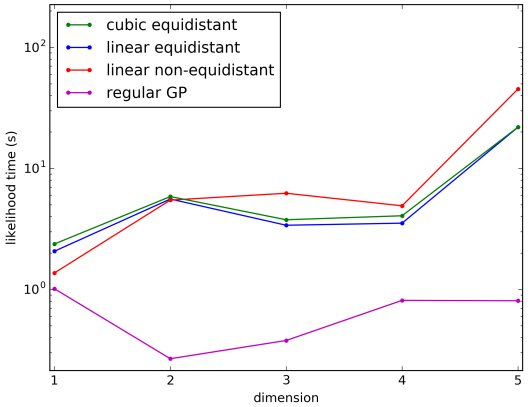
In Figure 4.9d we see the computed log likelihood. The plot for the linear equidistant interpolation is hard to distinguish from the cubic interpolation since the error of the likelihood approximation in a large part comes from the error of the eigenspectrum approximation which is exactly the same for the two equidistant grids. We see that the log likelihood approximation is underestimated for few dimensions, but for higher dimensions the number of inducing points increases, since $m = 50^D$, so that $m \gg n$ and the likelihood approximation is instead overestimated. The linear non-equidistant interpolation performs much better on the likelihood approximation for $D > 1$ than the equidistant interpolation methods in this case for only 500 training points and 50 inducing points per dimension.



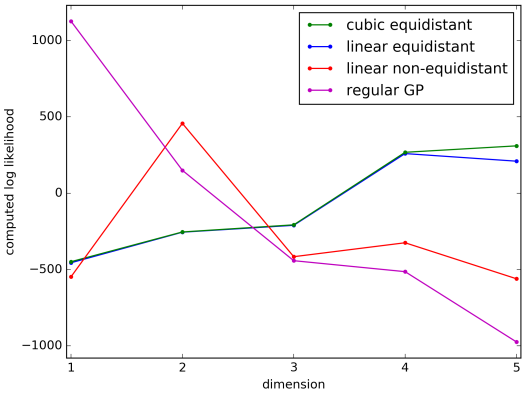
(a) Plot of relative error of SMSE for test points



(b) Plot of relative error MSLL for the test points



(c) Plot of time for computing the likelihood



(d) Plot of computed log likelihood

Figure 4.9: Plots of the results using the different interpolation methods when changing the input dimension D. Here the number of training points was $n = 500$, the number of inducing points was 50 per dimension, so in total $m = 50^D$ and the number of test points $n_* = 100$.

Conclusion

It is clear that to get good performance both for the predictions and the computed log likelihood using any of the interpolation methods, a large number of inducing points is needed, preferably at least as many as the number of training points. However, the number of training points should also be quite large, at least $n \gg 1000$, to get a close approximation of the log determinant.

From the interpolation experiments we see that when using a large number of training points and inducing points, cubic interpolation with equidistant grids achieves the best performance. Equidistant grids also have the advantage of being compatible with Toeplitz methods unlike non-equidistant grids. Therefore, for the rest of the experiment cubic interpolation with equidistant grids will be used for MSGP.

4.5 Structure Optimizations

For the different structure exploitations, we want to find out how much the training is sped up and in case of approximations we want to find out how accurate they are.

4.5.1 Kronecker Structure

For the Kronecker structure we have two parts: the fast exact MVMs and the approximation of the eigenvalue spectrum. Since the Kronecker MVMs are exact they should not influence the accuracy of the model but only the runtime, while for the eigenvalue spectrum approximation the accuracy can also be affected.

For the experiments evaluating the Kronecker structure methods, MSGP is used with different kinds of Kronecker optimizations: no structure at all, fast MVMs, eigenspectrum approximation but without using Kronecker product and finally both fast MVMs and eigenspectrum approximation with Kronecker product. The performances of these versions of MSGP are compared to the performance of a regular full GP.

Two kinds of experiments were performed for the Kronecker optimizations: observing the behaviour when changing the number of inducing points m and observing the behaviour when changing the number of training points n.

For all of the Kronecker experiments, the covariance function used was the same as for the interpolation experiments with $D > 1$. This was an ARD SE covariance function, which can be seen as a generalization of the one-dimensional SE covariance function and can be used with MSGP since it is a stationary product kernel. The length-scale for each dimension was set to max of 1 and the maximum grid space in the corresponding dimension in case of an inducing points grid. The variance of the covariance function was set to $\sigma_f = 1$ for each dimension and the variance of the noise was set to $\sigma_n = 0.001$. The local interpolation used equidistant grids and cubic interpolation.

Number of inducing points

The first part of the experiments for the number of inducing points is smaller problems where it is possible to use local kernel interpolation without structure optimizations. If eigenspectrum approximation and Kronecker MVMs are not used, the whole covariance matrix for the inducing points $K_{U,U}$ has to be computed explicitly for either the MVMs or the log determinant which quickly causes memory problems since $K_{U,U} \in \mathbb{R}^{m \times m}$ gets very large for multidimensional grids.

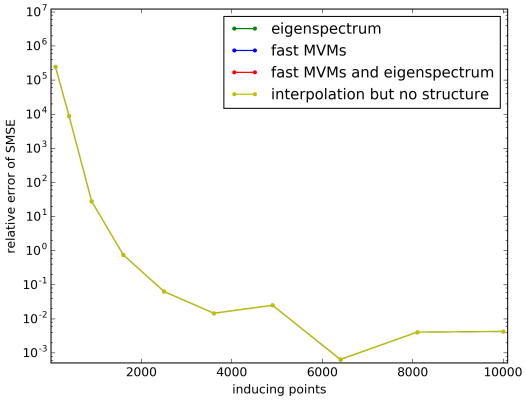
The first experiment is for two dimensional input data, $D = 2$, with the number of training points set to $n = 1000$ and the number of test points set to $n_* = 100$. The number of inducing points was increased from 10 to 100 per dimension with a step size of 10. This means that the total number of inducing points was increased from $m = 10^2 = 100$ to $m = 100^2 = 10\,000$.

In Figure 4.10 results are shown for this experiment. In Figures 4.10a and 4.10b we can see that all the different Kronecker optimizations have indistinguishable results for SMSE and MSLL. This is as expected since for the predictions we do not use the eigenspectrum approximation and the fast MVMs should not affect the accuracy.

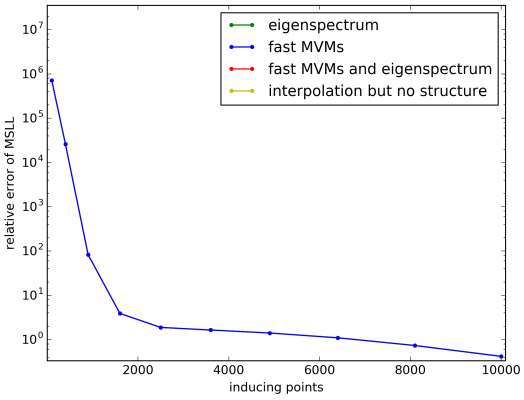
The relative error of the computed log likelihood is shown in a loglog-scale in Figure 4.10d. In this plot the version using only fast MVMs and the one using no Kronecker structure are hard to distinguish from each other and similarly the version using only the

eigenspectrum approximation and the one using both the eigenspectrum approximation and the fast MVMs are hard to distinguish from each other. The versions without the eigenspectrum approximation perform better on the log likelihood since the approximation is only asymptotically exact.

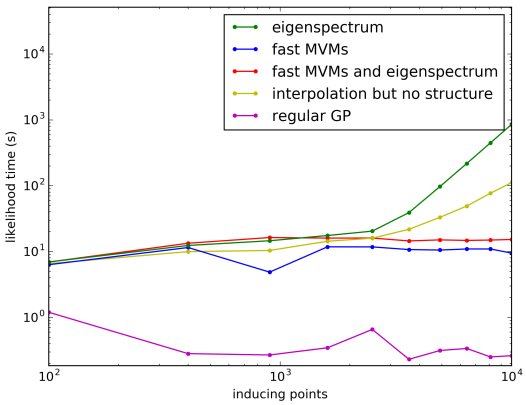
In Figure 4.10c the time for computing the likelihood is shown in a loglog-scale. From this plot it is clear the that versions which use fast MVMs have a much slower increase of the computational time. The affect of using the eigenspectrum approximation is not as clear from this experiment, but here the versions of MSGP which do not use the eigenspectrum approximation compute the log determinant of $K_{SKI} = WK_{U,U}W^\top \in \mathbb{R}^{n\times n}$ and since $n = 1000$ is quite small, this is still fast to compute.



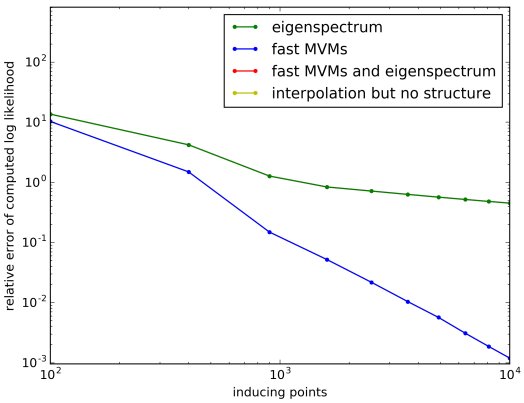
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood



(d) Plot of relative error of the computed log likelihood

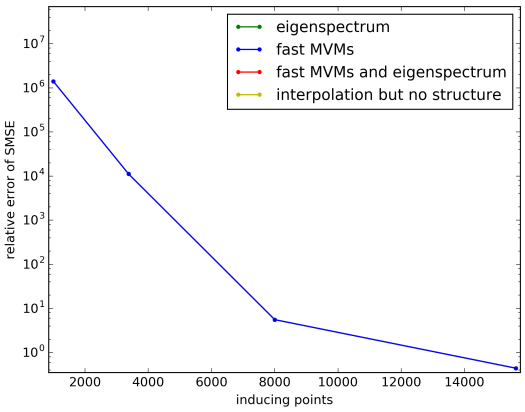
Figure 4.10: Plots of the results using the different Kronecker structure optimizations when changing the number of inducing points. Here the input dimension was $D = 2$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

A similar experiment was done for three dimensional input data, $D = 3$. Again the number of training points was set to $n = 1000$ and the number of test points was set to $n_* = 100$. The number of inducing points was increased from 10 to 25 per dimension with a step size of 5. This means that the total number of inducing points was increased from $m = 10^3 = 1000$ to $m = 25^3 = 15\,625$.

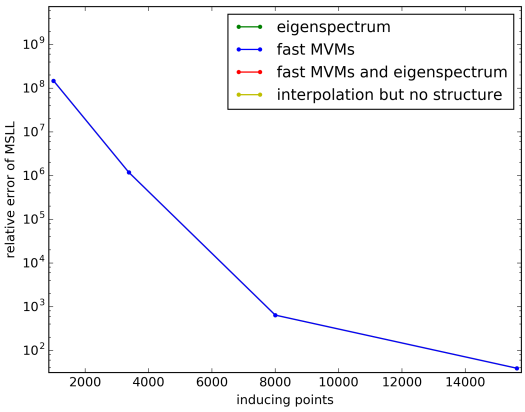
In Figure 4.11 results are shown for this experiment. From these plots we can see that the results are very similar to the ones for $D = 2$ in Figure 4.10. Again we see that all the different Kronecker optimizations have indistinguishable results for SMSE and MSLL in Figures 4.11a and 4.11b.

The relative error of the computed log likelihood is shown in a loglog-scale in Figure 4.11d. As for the previous experiment, the version using only fast MVMs and the one using no Kronecker structure are hard to distinguish from each other and similarly the different versions using eigenspectrum approximation are hard to distinguish from each other. Again the versions without the eigenspectrum approximation perform a bit better on the log likelihood, but for few inducing points per dimension the computed value of the log likelihood is far from the real value for all MSGP versions.

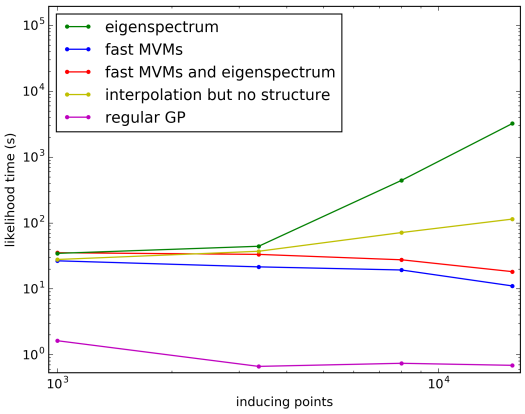
In Figure 4.11c the time for computing the likelihood is shown in a loglog-scale. As for $D = 2$, the MSGP versions which use fast MVMs have a slower increase of the computational time. The affect of using the eigenspectrum approximation is not clear from this experiment either, since we also here use $n = 1000$.



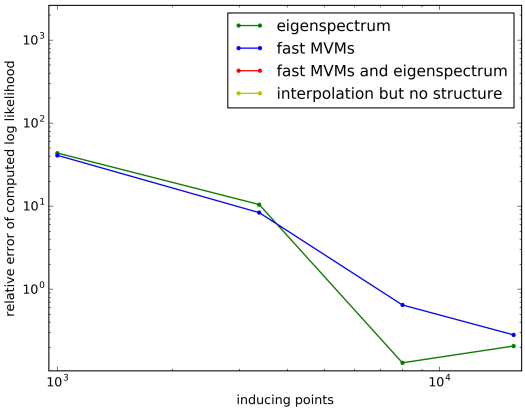
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood



(d) Plot of the relative error of the computed log likelihood

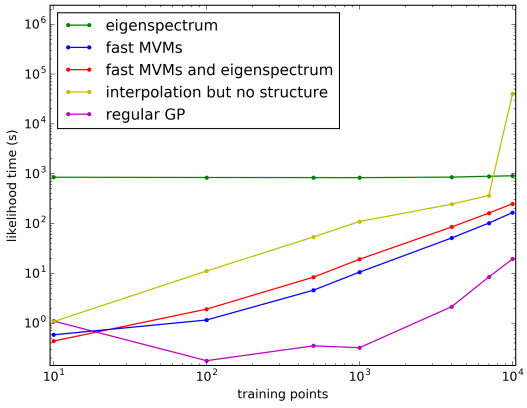
Figure 4.11: Plots of the results using the different Kronecker structure optimizations when changing the number of inducing points. Here the input dimension was $D = 3$, the number of training points $n = 1000$ and the number of test points $n_* = 100$.

Number of training points

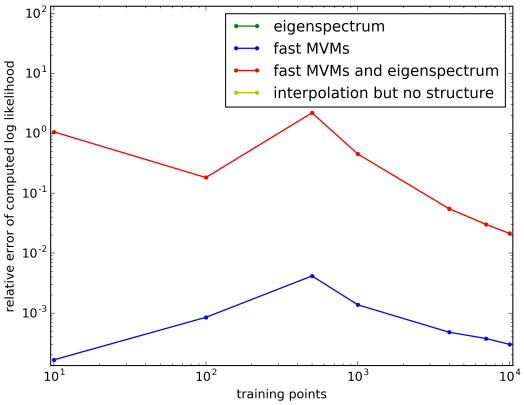
As for the Kronecker experiments when changing the number of inducing points, the first part of the experiments for the number of training points is smaller problems where it is possible to use local kernel interpolation without structure optimizations. As noted earlier, if eigenspectrum approximation and Kronecker MVMs are not used, the whole covariance matrix for the inducing points $K_{U,U}$ has to be computed explicitly for either the MVMs or the log determinant which quickly causes memory problems since $K_{U,U} \in \mathbb{R}^{m \times m}$ gets very large for multidimensional grids.

The first experiment for changing the number of inducing points was done for two dimensional input data, $D = 2$, and the number of training points n was increased from 1000 to $10\,000$. The number of inducing points was set to 100 per dimension so that the total number of inducing points was $m = 100^2 = 10\,000$. No predictions were computed since we have seen that all versions perform equally well on the predictions in the earlier experiments.

In Figure 4.12 results are shown for this experiment. The relative error of the computed log likelihood is shown in a loglog-scale in Figure 4.12b. As for the other Kronecker experiments, the version using only fast MVMs and the one using no Kronecker structure are hard to distinguish from each other and similarly the versions using the eigenspectrum approximation are hard to distinguish from each other. Again we see that the versions without the eigenspectrum approximation perform better on the log likelihood. However, note that the two curves have very similar shapes, even though the values differ with several magnitudes. The eigenvalue approximation should get better with an increasing number of training points n, but another big reason for the decrease of the relative error of the log likelihood with an increasing number of training points n is that the absolute value of the log likelihood increases. In Figure 4.12a the time for computing the likelihood is shown in a loglog-scale.



(a) Plot of time for computing the likelihood



(b) Plot of the relative error of the computed log likelihood

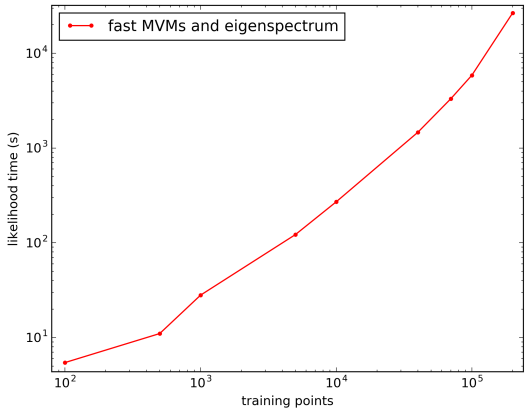
Figure 4.12: Plots of the results using the different Kronecker structure optimizations when changing the number of inducing points. Here the input dimension was $D = 2$, the number of inducing points per dimension 100 so that the total number of inducing points was $m = 100^D = 10\,000$ points.

The last experiment for changing the number of inducing points used two dimen-

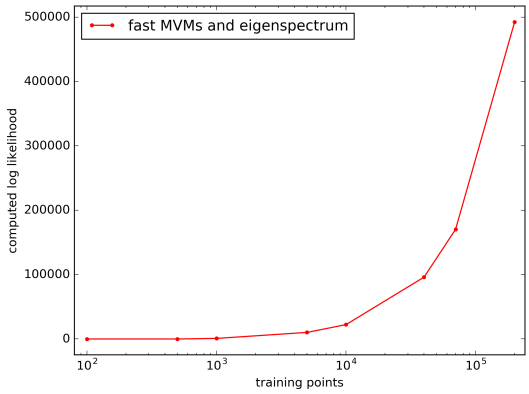
sional input data, $D = 2$, as the previous experiment, but now the number of training points n was increased from 100 to $200\,000$. The number of inducing points was set to 1500 per dimension so that the total number of inducing points was $m = 1500^2 = 2\,250\,000$. No predictions were computed. For this experiment we can not use regular GP which runs into memory problems for $n > 20000$ or the MSGP versions without both Kronecker MVMs and eigenspectrum approximation using the Kronecker product.

In Figure 4.13 results are shown for this experiment. The computed log likelihood is shown in Figure 4.13b. The log likelihood increases with the number of training points, which seems reasonable since it is the same behaviour as we have seen in earlier experiments.

The time for computing the likelihood is shown in a loglog-scale in Figure 4.12a. The curve is close to linear with a slope near 1, which consists with the theoretical result that MSGP which uses all Kronecker optimizations should have the computational complexity $\mathcal{O}(n)$.



(a) Plot of time for computing the likelihood



(b) Plot of the computed log likelihood

Figure 4.13: Plots of the results using the different Kronecker structure optimizations when changing the number of inducing points. Here the input dimension was $D = 2$, the number of inducing points per dimension 1500 so that the total number of inducing points was $m = 1500^D = 2\,250\,000$ points.

4.5.2 Toeplitz Structure

As for the Kronecker structure we have two parts of the Toeplitz structure: the fast exact MVMs and the circulant approximation. Again the MVMs are exact so they should not influence the accuracy of the model but only the runtime, while for the circulant approximation the accuracy can also be affected.

For the experiments evaluating the Toeplitz structure methods, MSGP is used with different kinds of Toeplitz optimizations: no structure at all, fast MVMs, eigenspectrum approximation without circulant approximation, eigenspectrum approximation with circulant approximation and finally both fast MVMs and eigenspectrum approximation with circulant approximation. The performances of these versions of MSGP are compared to the performance of a regular full GP.

Two kinds of experiments were performed for the Toeplitz optimizations: observing the behaviour when changing the number of inducing points m and observing the be-

haviour when changing the number of training points n. All the experiments had one-dimensional inputs, $D = 1$, since then all Toeplitz optimizations can be used.

For all the Toeplitz experiments, the covariance function used was the same as for the interpolation experiments with $D = 1$, that is an SE covariance function with length-scale l set to max of 1 and the maximum grid space in case of an inducing points grid. The variance of the covariance function was set to $\sigma_f = 1$ and the variance of the noise was set to $\sigma_n = 0.001$. The local interpolation used equidistant grids and cubic interpolation.

Number of inducing points

For the first experiment the number of training points was set to $n = 5000$ and the number of test points was set to $n_* = 100$. The number of inducing points was increased from $m = 50$ to $m = 1000$ with a step size of 50.

In Figure 4.14 results are shown for this experiment. From these plots we can see that all the different Toeplitz optimizations have almost indistinguishable results for SMSE and MSLL in Figures 4.14a and 4.14b until the relative errors get very small. This is as expected as for the predictions we do not use the eigenspectrum approximation and the fast MVMs should not affect the accuracy more than numerical errors.

The computed log likelihood is shown in Figure 4.14d. In this plot the version using only fast MVMs, the one using no MVM optimization and the regular GP are hard to distinguish from each other and similarly the versions of MSGP that use the eigenspectrum approximations are hard to distinguish from each other. The versions without the eigenspectrum approximation perform better on the log likelihood since the approximation is only asymptotically exact and for this experiment we always have $m \ll n$.

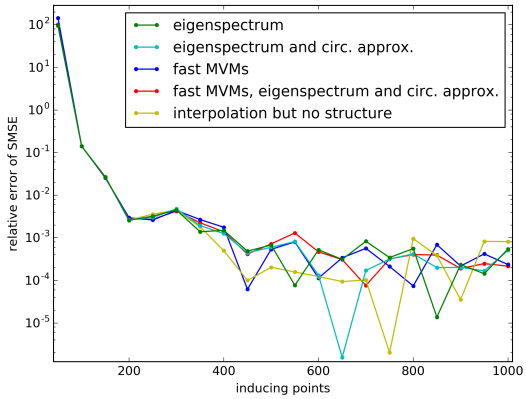
In Figure 4.14c the time for computing the likelihood is shown. From this plot it is not obvious how the computational time for the different structure optimizations scales with the number of inducing points, but in the next experiments a larger number of inducing points and training points will be used to try to make this more clear.

For the next experiment the number of training points was again set to $n = 5000$ and the number of test points was set to $n_* = 100$. This time the number of inducing points was increased from $m = 500$ to $m = 8000$ with a step size of 500.

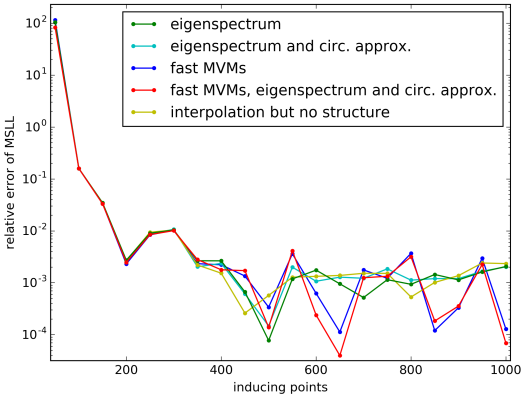
In Figure 4.15 results are shown for this experiment. From these plots we can see that all the different Toeplitz optimizations have comparable results for SMSE and MSLL in Figures 4.15a and 4.15b. Again, this is as expected since the performance of the different MSGP versions should only differ due to numerical errors.

The computed log likelihood is shown in Figure 4.15d. As for the previous experiment, the regular GP and the MSGP versions that do not use the eigenspectrum approximation are hard to distinguish from each other and similarly the versions of MSGP that use the eigenspectrum approximations are hard to distinguish from each other. Again we see that the versions without the eigenspectrum approximation perform much better on the log likelihood until the number of inducing points m is at least as big as the number of training points n, but for $m > n$ all approximation methods perform reasonably well.

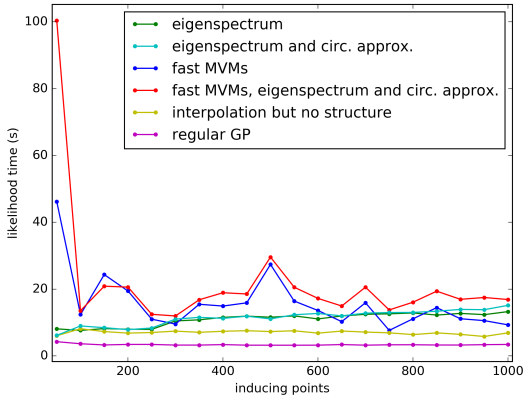
In Figure 4.15c the time for computing the likelihood is shown. From this plot we can see that the computational time increases most using the eigenspectrum approximation without the circulant approximation, which is due to the fact that then the eigendecomposition is performed on the whole matrix $K_{U,U} \in \mathbb{R}^{m \times m}$ which is of complexity $\mathcal{O}(m^3)$. We see that all the other methods have better performance, and in particular when the



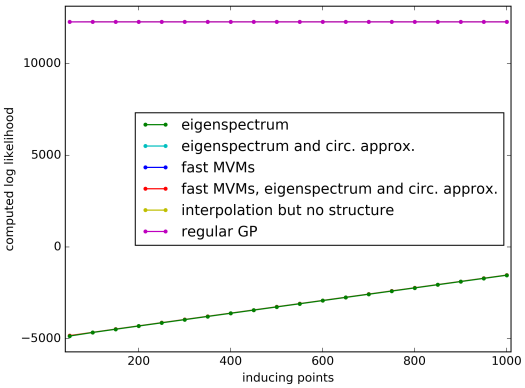
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood



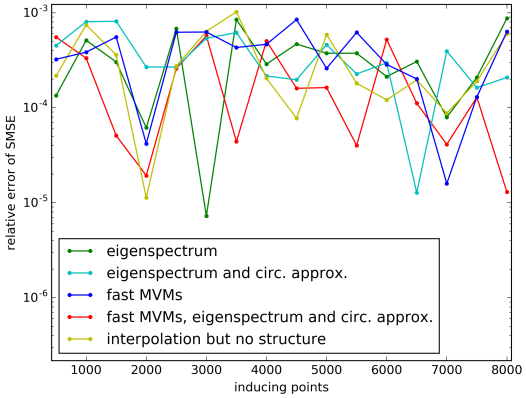
(d) Plot of the computed log likelihood

Figure 4.14: Plots of the results using the different Toeplitz structure optimizations when changing the number of inducing points. Here the input dimension was $D = 1$, the number of training points $n = 5000$ and the number of test points $n_* = 100$.

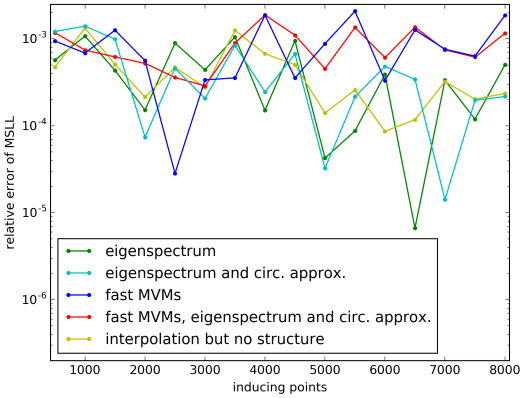
circulant approximation is used as well, the computational complexity gets better.

In the last experiment for the number of inducing points we want to see more clearly how the computational complexity scales with the number of inducing points. Therefore, the number of inducing points was increased from $m = 10$ to $m = 10^5$. However, the only version of MSGP which is able to use $m = 10^5$ inducing points is the one using fast MVMs and eigenspectrum approximation with circulant approximation since otherwise we have to use the whole matrix $K_{U,U} \in \mathbb{R}^{m \times m}$ explicitly, which introduces memory problems. The number of training points was set to $n = 15\,000$ and no predictions were made.

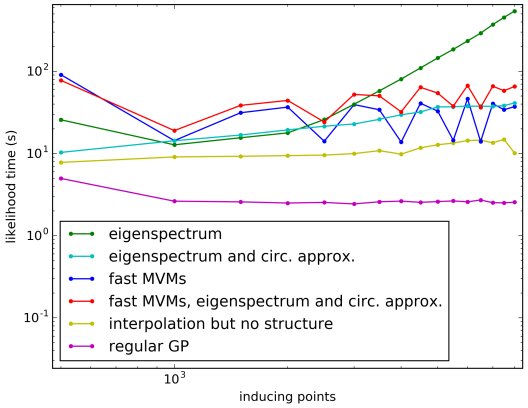
In Figure 4.16 results are shown for this experiment. The relative error of the computed log likelihood is shown in Figure 4.16b. We see that the relative error decreases until $m \geq n$ and for $m > n$ the relative error stays low. In Figure 4.16a the time for computing the likelihood is shown. From this plot we can see that the computational time for the MSGP increases but the graph is not linear at all so the exact computational complexity is not obvious.



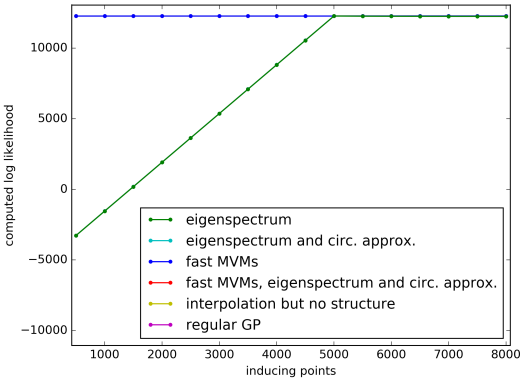
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood



(d) Plot of the computed log likelihood

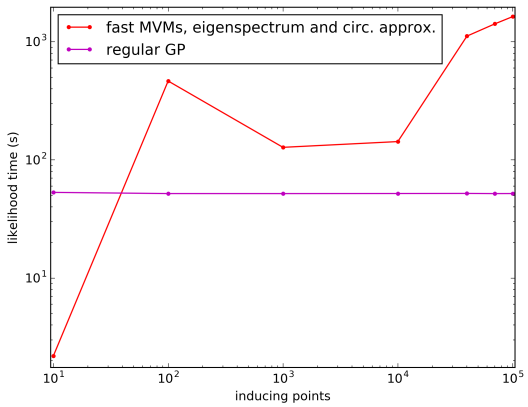
Figure 4.15: Plots of the results using the different Toeplitz structure optimizations when changing the number of inducing points. Here the input dimension was $D = 1$, the number of training points $n = 5000$ and the number of test points $n_* = 100$.

Number of training points

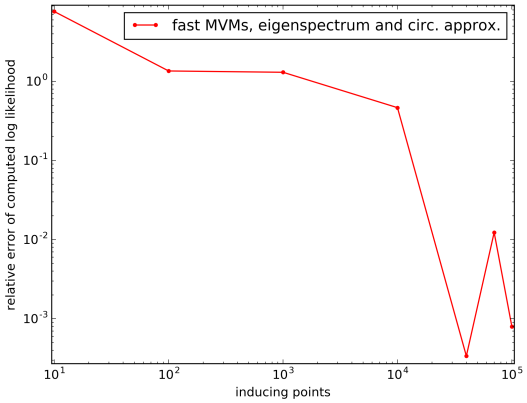
For the first experiment the number of inducing points was set to $m = 1000$ and the number of test points was set to $n_* = 100$. The number of training points was increased from $n = 1000$ to $n = 5000$ with a step size of 1000.

In Figure 4.17 results are shown for this experiment. In Figures 4.17a and 4.17b we see that all the different Toeplitz optimizations have comparable results for SMSE and MSLL. Again, this is as expected since for the predictions we do not use the eigenspectrum approximation and the fast MVMs should not affect the accuracy more than numerical errors.

The computed log likelihood is shown in Figure 4.17d. As before, the MSGP versions without the eigenspectrum approximation and the regular GP are hard to distinguish from each other and similarly the versions of MSGP that use the eigenspectrum approximations behave similarly. As seen in earlier experiments, the versions without the eigenspectrum approximation perform better on the log likelihood, especially for $n \gg m$ since the approximation is only asymptotically exact.



(a) Plot of time for computing the likelihood



(b) Plot of the relative error of the computed log likelihood

Figure 4.16: Plots of the results using the different Toeplitz structure optimizations when changing the number of inducing points. Here the input dimension was $D = 1$ and the number of training points $n = 15000$.

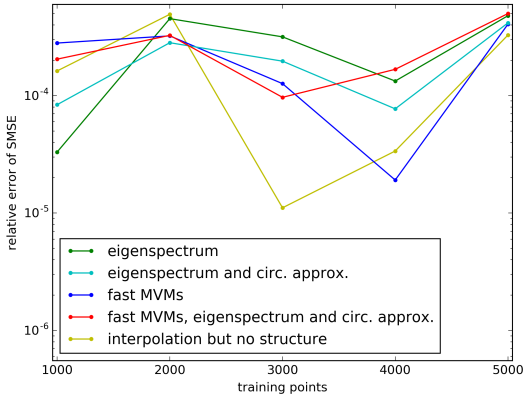
In Figure 4.17c the time for computing the likelihood is shown. From this plot it is not obvious how the computational time for the different structure optimizations scales with the number of training points, but in the next experiments a larger number of inducing points and training points will be used to try to make this more clear.

For the next experiment the number of inducing points was set to $m = 15\,000$ and no predictions were made. The number of training points was increased from $n = 1000$ to $n = 22\,000$ with a step size of 3000.

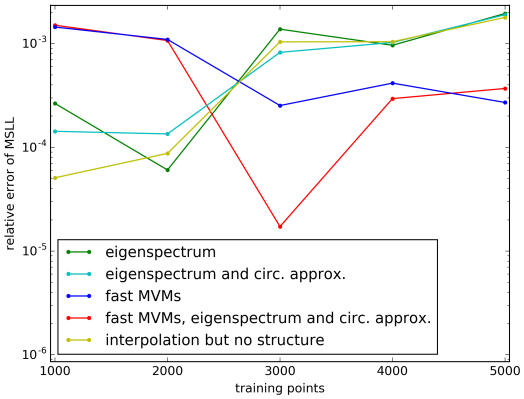
In Figure 4.18 results are shown for this experiment. The relative error of the computed log likelihood is shown in a loglog-scale in Figure 4.18b. The versions with the eigenspectrum approximation performs well on the log likelihood, until $n > m$ when the error starts increasing.

In Figure 4.18a the time for computing the likelihood is shown in a loglog-scale. Here we see that using interpolation without any structure optimizations, has at least the same computational time as using a regular GP. We also see that the computational time for using the eigenspectrum approximation without the circulant approximation is quite large, but does not depend very much on the number of training points. We can also see that for this experiment the fast MVMs are not faster than regular MVMs but actually a bit slower. The fast MVMs should become faster for a larger number of inducing points, since they theoretically have a better computational complexity than the regular MVMs. However, it should be noted that the runtime is not the only important part and another advantage of the Toeplitz MVMs is that they have much lower memory requirements than regular MVMs since for Toeplitz MVMs only one column of the matrix has to be stored.

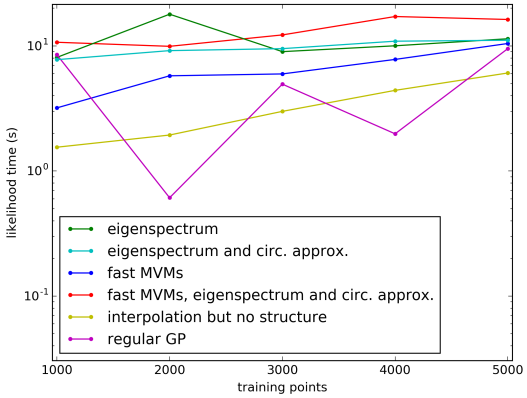
The aim of the last experiment for the Toeplitz optimizations is to see more clearly the computational complexity for the log likelihood. To achieve this the number of inducing points was set to $m = 100\,000$ and no predictions were made. The number of training points was increased from $n = 100$ to $n = 200\,000$. The only version of MSGP that can be used for this experiment is the one using both Toeplitz MVMs and the eigenspectrum approximation with the circulant approximation, due to the large number of



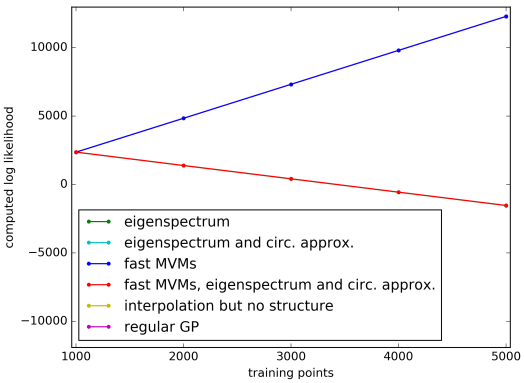
(a) Plot of relative error of the SMSE for test points compared to regular GP



(b) Plot of relative error of the MSLL for the test points compared to regular GP



(c) Plot of time for computing the likelihood

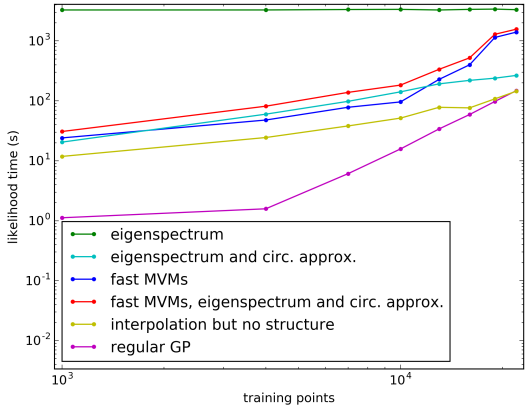


(d) Plot of the computed log likelihood

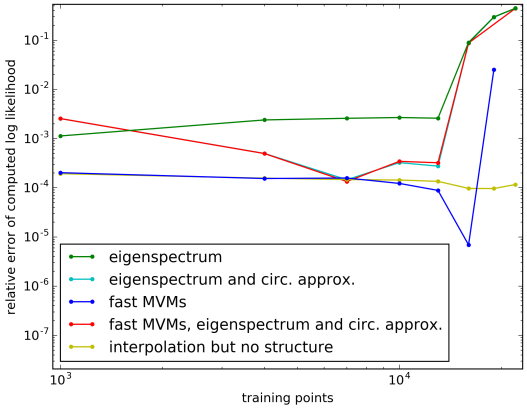
Figure 4.17: Plots of the results using the different Toeplitz structure optimizations when changing the number of training points. Here the input dimension was $D = 1$, the number of inducing points $m = 1000$ and the number of test points $n_* = 100$.

inducing points and training points. The regular GP can not be used for this experiment.

In Figure 4.19 the time for computing the likelihood is shown in a loglog-scale. As in the similar experiment for the Kronecker optimizations, we see that the curve is close to linear with a slope near 1, which consists with the theoretical result that MSGP should have the computational complexity $\mathcal{O}(n)$ for one-dimensional input data when using both Toeplitz MVMs and the eigenspectrum approximation with the circulant approximation.



(a) Plot of time for computing the likelihood



(b) Plot of the computed log likelihood

Figure 4.18: Plots of the results using the different Toeplitz structure optimizations when changing the number of training points. Here the input dimension was $D = 1$ and the number of inducing points $m = 15000$.

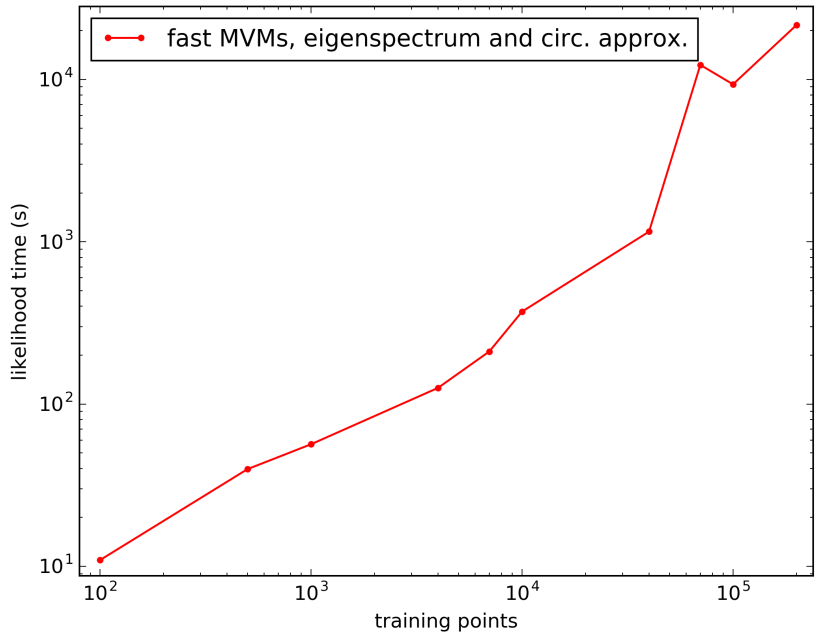


Figure 4.19: Plots of the results using Toeplitz structure optimizations when changing the number of training points. Here the input dimension was $D = 1$ and the number of inducing points $m = 100\,000$.

4.6 Comparison of MSGP to Other GP Methods

For the last part of the experiments, the MSGP framework using local kernel interpolation, Kronecker and Toeplitz optimizations, projections and fast predictions is tested against other GP methods on the real datasets. The regular GP is used when possible, and the other scalable GP methods MSGP is compared to are FITC and VFE, which both already have existing implementations in GPflow.

For all of the datasets, the input points were normalized to have zero mean and unit

variance on the training points and the outputs were centered to have zero mean on the training data.

For MSGP the input points were projected to a two dimensional space since all of the datasets have higher input dimensions than what is reasonable to use with MSGP directly. The values of the elements of the projection matrix as well as the other hyperparameters were determined by optimization of the log likelihood. However, the projection matrix was constrained to have unit scaling. The grid was equidistant and cubic interpolation was used. All of the Kronecker optimizations were used as well as the Whittle circulant approximation with truncation $w = 3$.

For all datasets, MSGP used a two dimensional inducing points grid with 200 points per dimension, which means that the total number of inducing points was $m = 200^2 = 40\,000$.

For both FITC and VFE inducing points are used as well. For the `Abalone` and `KIN40K` datasets $m = 2000$ inducing points were used, while for the `SARCOS` dataset $m = 500$ inducing points were used to avoid memory problems. The inducing points were initialized to the first m training points for both FITC and VFE. The locations of the inducing points were jointly maximized over together with the hyperparameters of the covariance function. For the regular GP, only the hyperparameters of the covariance function are optimized over.

The covariance function used was always an ARD SE covariance function. The lengthscale for each dimension was initialized to 3 and was constrained to be in the interval $[0.1, 10]$ for all GP methods except MSGP where the lower limit was the grid space in that dimension. The variance of the covariance function was initialized to $\sigma_f = 1$ for each dimension. The variance of the noise was set to $\sigma_n = 3$ and constrained to lie in the interval $[0.1, 10]$ for the `Abalone` dataset and in the interval $[10^{-3}, 10]$ for the other two datasets. For all methods except MSGP a White kernel with the fixed variance was added to the covariance as well. This kernel is simply a diagonal matrix added to avoid unsuccessful Cholesky decompositions. The variance of the White kernel was 10^{-2} for the `Abalone` dataset and 10^{-3} for the other two datasets.

To evaluate the performances on the test points SMSE, SMAE and MSLL were computed. Further more, the number of optimizations evaluations performed during the training and the value of the maximized log likelihood are presented to indicate how successful the training was. We also measured the time for computing the log likelihood once, the total training time and the time to predict the mean and variance for all the test points. For MSGP the time for computing $\frac{1}{n_s}\sum_{i=1}^{n_s}(\tilde{K}_{U,X}\mathbf{r}_i)^2$ needed for the fast predictions (see Section 3.3) was included in the prediction time.

4.6.1 `Abalone`

For the `Abalone` dataset, it is possible to use the regular GP as well since the dataset is small enough.

The results for all the GP methods are shown in Table 4.2. All methods have comparable results for the values of SMSE, SMAE and MSLL as well as the maximized log likelihood, but we can note that FITC has even "better" results than the regular GP on the MSLL and the maximized likelihood. Since the regular GP is the true one this indicates that FITC suffers from overfitting.

For the likelihood and prediction times, we see that FITC and VFE have very similar

performance, but for this small dataset the regular GP is actually fastest. MSGP on the other hand performs much worse, but this is likely due to the implementation not being optimized enough. We can also note that even though FITC computes each log likelihood faster than MSGP, it has the highest training time since the optimization evaluates the objective function so many times.

Table 4.2: Results on the `Abalone` dataset for the different scalable GP methods.

	MSGP	FITC	VFE	Regular GP
SMSE	0.4290	0.4428	0.4248	0.4235
SMAE	0.6467	0.6537	0.6445	0.6438
MSLL	−0.4250	−0.4846	−0.4356	−0.4375
Maximized log likelihood	−6898.6	−6064.9	−6812.9	−6806.7
Optimization evaluations	81	741	49	59
Likelihood time (s)	28.62	1.92	1.60	1.45
Prediction time (s)	6.26	2.26	2.15	1.91
Training time (s)	175.5	1479.4	79.6	161.9

4.6.2 KIN40K

The results for the `KIN40K` dataset are shown in Table 4.3. FITC and VFE have similar performances on SMSE, SMAE and MSLL, but FITC finds a much higher value of the log likelihood than VFE. MSGP performs considerably worse than the other methods for all the metrics and also take much longer both for the predictions and the likelihood computations. We can also note that MSGP does not do as many evaluations for the optimizations as the other methods and the maximized log likelihood is considerably smaller, which could indicate that it did not find a good optimum for the log likelihood and therefore does not achieve as good results as the other methods on the performance metrics.

Table 4.3: Results on the `KIN40K` dataset for the different scalable GP methods.

	MSGP	FITC	VFE
SMSE	0.3874	0.0351	0.0221
SMAE	0.6041	0.1252	0.1274
MSLL	−0.4597	−2.2691	−1.8904
Maximized log likelihood	−9775.6	6240.8	1208.3
Optimization evaluations	71	179	317
Likelihood time (s)	96.80	1.50	1.60
Prediction time (s)	111.91	2.22	2.10
Training time (s)	1143.3	294.5	532.0

4.6.3 SARCOS

In Table 4.4 the results are shown for the `SARCOS` dataset. All GP methods achieve low values for the performance metrics, even though MSGP performs a bit worse than FITC and VFE. For the likelihood and prediction times, we see as for the other datasets that FITC and VFE have very similar performance while MSGP performs much worse.

As for the KIN40K dataset the training for MSGP does not perform as many evaluations of the objective function as the other two methods, which could indicate that the optimum found for MSGP is not the best one.

Table 4.4: Results on the SARCOS dataset for the different scalable GP methods.

	MSGP	FITC	VFE
SMSE	0.0596	0.0108	0.0159
SMAE	0.2318	0.1195	0.1171
MSLL	−1.207	−2.258	−2.017
Maximized log likelihood	−142490.5	−100695.2	−112790.8
Optimization evaluations	105	244	410
Likelihood time (s)	401.45	2.59	2.43
Prediction time (s)	608.9	3.34	3.29
Training time (s)	5854.0	151.5	248.9

Chapter 5

Discussion

5.1 Pros and Cons of MSGP

Compared to the regular GP the main advantage of MSGP is the improved computational complexity and memory requirements. In our experiments, we have found that the memory requirements for the regular GP are that first makes an approximation method needed.

However, for $D > 3$ it is instead MSGP that runs into memory problems faster, since the number of points in a regular grid grows exponentially with the dimension. This means that MSGP suffers from the curse of dimensionality, which is not an apparent problem for the regular GP. MSGP can still be used for $D > 3$, as long as it is combined with projections.

Compared to the other two popular approximation methods FITC and VFE, MSGP has better computational complexity and memory requirements with regard to the number of inducing points m. This means that MSGP can use many more inducing points and thereby capture more of the behaviour of the covariance function.

The main disadvantage of MSGP compared to FITC and VFE is that using MSGP comes with a lot of decision making. First of all Toeplitz and/or Kronecker methods have to work with the covariance function and the decision of which structure exploitations to use has to be made. Then the locations of the inducing points have to be chosen, which is not the case for FITC and VFE where the locations usually are chosen by optimization. It must also be decided if projections should be used, and if so in which dimension, and how many samples should be used for the fast predictions.

As seen from the experiments, the implementation of MSGP in TensorFlow is quite slow compared to the other GP methods. This is probably due to the fact that the other GP methods use Cholesky decompositions for the heavy computations which TensorFlow provides a function for. Since the Cholesky decomposition is commonly used, this function is likely to be highly optimized. For MSGP the computations are more complicated with several different parts and it is more challenging to achieve an optimized implementation using TensorFlow in Python.

5.2 When Should MSGP Be Used?

If the training points are few enough to make full regular GP regression possible, the regular GP is the easiest and most accurate method to use. The limit for using the reg-

ular GP is different for each machine, in our case it could be used with up to $\sim 20\,000$ training points for inference to be possible, but less if training of the hyperparameters is to be performed as well.

For more training points, some scalable approximation method has to be used. In general, FITC and VFE are easier to use for non-experienced users since not a lot of choices has be made.

However, if the covariance function used has a more complex behaviour than the SE covariance, it is probable that MSGP catches this behaviour better than FITC and VFE since it can use a much larger number of inducing points. However, for the current version of MSGP, it is only possible to use stationary covariance functions and for $D > 2$ dimensions they must also be product covariance functions if Kronecker structure is to bwe used.

The general advice for using MSGP based on the experiments is to only use it for a large number of training points, always use at least as many inducing points as training points, that is $m \geq n$, and use cubic interpolation and equidistant grids. If the input dimension is larger than 3, projections should be used together with MSGP to make the dimension lower.

For $D = 1$ input dimension Toeplitz methods should be used, both the eigenspectrum approximation with the circulant approximation and the fast MVMs to avoid memory problems. For $D > 1$ input dimensions Kronecker methods should be used instead, again both the eigenspectrum approximation and the fast MVMs to avoid memory problems. It is also possible to combine Toeplitz and Kronecker methods, but this is only needed for the case when the number of inducing points per dimension gets very large.

If optimization of the hyperparameters is performed, the length-scales of the covariance function should be constrained to be at least as large as the grid space, since otherwise the optimization can find bad solutions.

5.3 Future Work

There are still many aspects of MSGP that can be investigated further. As shown in the experiments, the eigenspectrum approximation requires both a large number of training points and a large number of inducing points to perform well. Other approximations of the log determinant can be investigated as well, for example Han et al. have introduced an approximation method for the log determinant using stochastic Chebyshev expansions which relies on fast MVMs [26].

It should also be noted that we only studied the value of the computed log likelihood and not the value of the gradient with respect to the hyperparameters, which would also be interesting to examine further since the optimization is highly dependent on the gradient.

Other interpolation methods could also be investigated, for example multidimensional interpolation which do not treat each dimension separately. Multidimensional interpolation could be combined with the multidimensional extension of Toeplitz methods introduced in Section 3.4.1, which would make it possible to use MSGP for other covariance functions than product functions as well. Furthermore, extrapolation could also be investigated to avoid the problem of having training or test points located outside the grid.

In this thesis the projection method has not been given a lot of attention, but it could

be interesting to examine different projection methods. As noted, MSGP does not work well in higher dimensions and therefore the projection method can be quite important for high-dimensional data. We only used a single constant matrix for the projections in the experiments, but for example additive linear projections or non-linear projections could be used as well, which could lead to better results. For example, Wilson et al. have used deep architectures as non-linear projections [27].

From our experiments the expected nearly-linear computational complexity of MSGP was not always obvious. One reason could be that the LCG actually do not use $j \ll n$ iterations in case the matrix is badly conditioned. A solution for this could be to investigate preconditioners for the LCG since this can speed up the convergence.

Bibliography

[1] E.L. Snelson. Flexible and efficient Gaussian process models for machine learning. *ACM SIGKDD Explorations Newsletter*, 7(2001):1–135, 2007.

[2] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[3] Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Ralf Herbrich. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1935–1959, 2005. ISSN 1533-7928.

[4] Yunus Saatçi. *Scalable inference for structured Gaussian process models*. PhD thesis, Citeseer, 2012.

[5] Andrew Gordon Wilson, Christoph Dann, and Hannes Nickisch. Thoughts on Massively Scalable Gaussian Processes. *EMNLP*, pages 1–25, 2015.

[6] Svante Janson. *Gaussian Hilbert Spaces*, volume 129. Cambridge university press, 1997. ISBN 0521561280.

[7] Robert J. Adler and Jonathan E. Taylor. Random Fields and Geometry. *Science*, 17(3): 448, 2007. ISSN 1439-7382.

[8] Robert J Adler. *The Geometry of Random Fields*, volume 62. SIAM, 1981.

[9] Christopher KI Williams and Carl Edward Rasmussen. Gaussian Processes for Machine Learning. *The MIT Press*, 2(3):4, 2006.

[10] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian Processes using Pseudo-inputs. *Advances in Neural Information Processing Systems 18*, pages 1257–1264, 2006. ISSN 1049-5258.

[11] Michalis Titsias. Variational Learning of Inducing Variables in Sparse Gaussian Processes. *AISTATS*, 5:567–574, 2009. ISSN 15324435.

[12] Matthias Bauer, Mark van der Wilk, and Carl Edward Rasmussen. Understanding Probabilistic Sparse Gaussian Process Approximations. In *Advances in Neural Information Processing Systems*, pages 1525–1533, 2016.

[13] Volker Tresp. A Bayesian committee machine. *Neural computation*, 12(11):2719–2741, 2000.

[14] Andrew Gordon Wilson. Covariance kernels for fast automatic pattern discovery and extrapolation with Gaussian processes. *Dissertation*, 2014.

[15] Z Tang, R Duraiswami, and N A Gumerov. Fast algorithms to compute matrix-vector products for Pascal matrices. 20742, 2004.

[16] Andrew Gordon Wilson and Hannes Nickisch. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). *International Conference on Machine Learning*, 37:1–19, 2015.

[17] Robert Keys. Cubic convolution interpolation for digital image processing. *IEEE transactions on acoustics, speech, and signal processing*, 29(6):1153–1160, 1981.

[18] Christopher K. I. Williams and Matthias Seeger. The Effect of the Input Density Distribution on Kernel-based Classifiers. *International Conference on Machine Learning*, pages 1159–1166, 2000.

[19] George Papandreou and Alan L Yuille. Efficient variational inference in large-scale Bayesian compressed sensing. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1332–1339. IEEE, 2011.

[20] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). Savannah, Georgia, USA*, 2016.

[21] Alexander G de G Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *arXiv preprint arXiv:1610.08733*, 2016.

[22] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`.

[23] Anton Schwaighofer and Volker Tresp. Transductive and inductive methods for approximate Gaussian process regression. *Advances in Neural Information Processing Systems*, pages 977–984, 2003.

[24] Matthias Seeger, Christopher Williams, and Neil Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318, 2003.

[25] Christopher KI Williams and Matthias Seeger. Using the Nyström method to speed up kernel machines. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, pages 661–667. MIT press, 2000.

[26] Insu Han, Dmitry Malioutov, and Jinwoo Shin. Large-scale log-determinant computation through stochastic Chebyshev expansions. In *ICML*, pages 908–917, 2015.

[27] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 370–378, 2016.