Circuits and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
https://sps.ewi.tudelft.nl/

SPS-2023-5519829

M.Sc. Thesis

Malleable Kernel Interpolation for Scalable Structured Gaussian Process

Hanyuan Ban

Abstract

Gaussian process regression (GPR), a potent non-parametric data modeling tool, has gained attention but is hindered by its high computational load. State-of-the-art low-rank approximations like structured kernel interpolation (SKI)-based methods offer efficiency, yet lack a strategy for determining the number of grid points, a pivotal factor impacting accuracy and efficiency. In this thesis, we tackle this challenge.

We explore existing low-rank approximations that facilitates the computation, dissecting their strengths and limitations, particularly SKI-based methods. Subsequently, we introduce a novel approximation framework, MKISSGP, which dynamically adjusts grid points using a new hyperparameter of the model: density, according to changes in the kernel hyperparameters in each training iteration.

MKISSGP exhibited consistent error levels in the reconstruction of the kernel matrix, irrespective of changes in hyperparameters. This robust performance forms the bedrock for achieving accurate approximations of kernel matrix-related terms. When employing our recommended density value (i.e., 2.7), MKISSGP achieved a comparable level of precision to that of precise GPR, while requiring only 52% of the time compared to the current state-of-the-art method.

Malleable Kernel Interpolation for Scalable Structured Gaussian Process

Hanyuan Ban

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Microelectronics & Computer Engineering
Circuits and Systems Group

Delft University of Technology
Department of
Microelectronics

The undersigned hereby certify that they have read and recommend to the Faculty of Electrical Engineering, Mathematics and Computer Science for acceptance a thesis entitled **"Malleable Kernel Interpolation for Scalable Structured Gaussian Process"** by **Hanyuan Ban** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 07-11-2023

Chairman: _____
dr. R.T.Rajan

Advisor: _____
dr. R.T.Rajan

Committee Members: _____
dr. F. Fioronelli

_____
dr. B. Giovanardi

Abstract

Gaussian process regression (GPR), a potent non-parametric data modeling tool, has gained attention but is hindered by its high computational load. State-of-the-art low-rank approximations like structured kernel interpolation (SKI)-based methods offer efficiency, yet lack a strategy for determining the number of grid points, a pivotal factor impacting accuracy and efficiency. In this thesis, we tackle this challenge.

We explore existing low-rank approximations that facilitates the computation, dissecting their strengths and limitations, particularly SKI-based methods. Subsequently, we introduce a novel approximation framework, MKISSGP, which dynamically adjusts grid points using a new hyperparameter of the model: density, according to changes in the kernel hyperparameters in each training iteration.

MKISSGP exhibited consistent error levels in the reconstruction of the kernel matrix, irrespective of changes in hyperparameters. This robust performance forms the bedrock for achieving accurate approximations of kernel matrix-related terms. When employing our recommended density value (i.e., 2.7), MKISSGP achieved a comparable level of precision to that of precise GPR, while requiring only 52% of the time compared to the current state-of-the-art method.

Acknowledgments

I would like to express my profound gratitude to my supervisor, Dr. Raj Thilak Rajan, and my daily supervisor, Ir. Ellen Riemens, for their unwavering support throughout the process of writing this thesis. Without their guidance and encouragement, this endeavor would not have been possible. Our meetings provided me with invaluable insights into the field of Gaussian Process, as well as valuable suggestions for crafting a high-quality thesis. Despite encountering delays, they graciously assisted me in revising my plan and offered guidance. I am confident that the knowledge and methodologies I have gained from them will continue to enrich my life.

I extend my heartfelt appreciation to my girlfriend, who has been my constant companion throughout the two years of my foreign study life. She has been the sunshine on rainy days, the vibrant colors in the monochrome world, and the driving force propelling me towards the future.

Lastly, I would like to express my deep gratitude to my dearest friends. Sharing this incredible journey with them in this beautiful land has been an invaluable experience. They are the cherished fruits of the past two years of study. While life at TUDelft had its challenges, when I look back on this journey nearing its conclusion, every moment feels vivid and profoundly moving. This thesis marks the culmination of my student life. I really enjoyed every bit of my last journey.

Hanyuan Ban
Delft, The Netherlands
07-11-2023

Contents

List of Figures

List of Tables

Acronyms

CCI cubic convolution interpolation

CG conjugate gradients

dNMLL derivative of negative marginal log-likelihood w.r.t. hyperparameters

DTC deterministic training conditional

DTE Dutch Taylor expansion

ECCI exponential cubic convolution interpolation

FFT fast Fourier transform

FITC fully independent training conditional

GP Gaussian process

GPR Gaussian process regression

GSGP grid-structured Gaussian process

KDE kernel density estimate

KISSGP kernel interpolation for scalable structured Gaussian process

MKISSGP malleable kernel interpolation for scalable structured Gaussian process

MPC model predictive control

MVM matrix-vector multiplication

NMLL negative marginal log-likelihood

NMPC nonlinear model predictive control

PDF probability density function

RBF radial basis function

RMSE root mean square error

SE square exponential

SKI structural kernel interpolation

SMAE standard mean absolute error

SoR subset of regressors

SPD symmetric positive-definite

SSGP sparse spectral Gaussian process

SSKI sparse structured kernel interpolation

Nomenclature

Common

Scalars are represented by plain lowercase letters, e.g. a
Vectors are represented by bold lowercase letters, e.g. $\mathbf{a}$
Matrices are represented by bold uppercase letters, e.g. $\mathbf{A}$

∇	the gradient operator
∇^2	the Laplace operator

Gaussian Process

$\boldsymbol{\phi}(\mathbf{x})$	basis function vector
$\boldsymbol{\theta}$	the hyperparameters of a kernel function
$\mathbf{f}_*$	the predicted test output
$\mathbf{f}$	the noiseless train output
$\mathbf{K}_{\cdot,\cdot}$	kernel matrix of two sources
$\mathbf{U}$	the inducing variables or grid points
$\mathbf{W}$	the interpolation matrix
$\mathbf{X}_*$	the test input
$\mathbf{X}$	the train input
$\mathbf{y}$	the noisy train output
$\mathcal{GP}(m(\mathbf{x}), k(\cdot,\cdot))$	a Gaussian process with mean $m(\mathbf{x})$ and kernel function $k(\cdot,\cdot)$
$k(\cdot,\cdot)$	kernel function
$\mathrm{S}(\boldsymbol{\omega})$	the spectral density of a kernel function

Linear Algebra

$\det(\mathbf{A})$	determinant of matrix $\mathbf{A}$
$\mathbf{0}$	the zero matrix
$\mathbf{A}^T$	transpose of matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of matrix $\mathbf{A}$
$\mathbf{I}$	the identity matrix
$\mathrm{diag}[\cdot]$	the diagonal compoment of a matrix

Other Symbols

δ_{ij}	the Kronecker delta. The value is 1 when $i = j$, otherwise 0
$\mathbb{R}^d$	d-dimensional real space
$\mathcal{F}[\cdot]$	the Fourier transform
$\mathcal{O}(\cdot)$	complexity order
$\mathcal{O}(mvm(\cdot))$	complexity order of calculating the matrix-vector multiplication

$\otimes$ the Cartesian product

Probabilistics

$\mathbb{E}(\cdot)$ the expected value

$\mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$ a (multivariate) normal distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{K}$

$Cov(\mathbf{X})$ the covariance matrix of a vector of inputs $\mathbf{X}$

$cov(\mathbf{x})$ the covariance of a single input$\mathbf{x}$

$p(x)$ the probability of x

Linear Algebra

Matrix Inversion Lemma

For invertible matrices $\mathbf{B}$ and $\mathbf{D}$:

$$(\mathbf{ABC}+\mathbf{D})^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{A}\left(\mathbf{B}^{-1} + \mathbf{CD}^{-1}\mathbf{A}\right)^{-1}\mathbf{CD}^{-1}$$

Matrix Determinant Lemma

For invertible matrices $\mathbf{B}$ and $\mathbf{D}$:

$$\det(\mathbf{ABC}+\mathbf{D}) = \det(\mathbf{B}^{-1} + \mathbf{CD}^{-1}\mathbf{A})\det\mathbf{B}\det\mathbf{D}.$$

Cholesky Decomposition

For a symmetric positive-definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$:

$$\mathbf{A} = \mathbf{LL}^T.$$

where $\mathbf{L}$ is a lower triangular matrix.

This decomposition can be used in solving linear systems $\mathbf{Ax} = \mathbf{b}$ with high efficiency and strong numeric stability [1]. Specifically, first, one can efficiently calculate $\mathbf{y}$ from $\mathbf{Ly} = \mathbf{b}$ by forward substitution, since $\mathbf{L}$ has a lower triangular structure. Next, solve $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ through backward substitution. The Cholesky decomposition has a time complexity of $\mathcal{O}(n^3)$ in general. However, it still takes less computation than the matrix inverse operation.

Furthermore, the log-determinant of $\mathbf{A}$ can be easily acquired by:

$$\log\det(\mathbf{A}) = 2\sum_{i=0}^{n-1}\log\mathbf{L}_{i,i},$$

where $\mathbf{L}_{i,i}$ denotes the i-th element of the diagonal of $\mathbf{L}$.

Lanczos Decomposition

For a symmetric positive-definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$:

$$\mathbf{A} = \mathbf{QTQ}^T,$$

where $\mathbf{Q} \in \mathbb{R}^{n\times m}$ has orthonormal columns, $\mathbf{T} \in \mathbb{R}^{m\times m}$ is a symmetric tridiagonal matrix. m is the degree of the Lanczos decomposition, which is a free parameter of choice. The first column of $\mathbf{Q}$ can also be arbitrary as long as it is unitary.

This decomposition is most commonly used to calculate the first m eigenvalues of $\mathbf{A}$ by performing an eigendecomposition of $\mathbf{T}$. Suppose an eigenvalue and the corresponding eigenvector of $\mathbf{T}$ are λ and $\mathbf{v}$, then λ and $\mathbf{Qv}$ are the eigenvalue and eigenvector of $\mathbf{A}$. The time complexity to perform the Lanczos decomposition is $\mathcal{O}(mn)$. And the time complexity to eigendecompose the tridiagonal matrix $\mathbf{T}$ can be $\mathcal{O}(m^2)$.

Moreover, the decomposition only requires matrix-vector multiplication of $\mathbf{A}$, which can leverage any possible structures in $\mathbf{A}$ that facilitate the multiplication calculation.

In this section, the key iterative algorithms implemented in this thesis are briefly explained with their properties. Pseudocodes are also given along with the explanations.

Iterative Algorithms

Lanczos method for Log-Determinant Estimation

The Lanczos method for log-determinant is an implementation of the stochastic Lanczos quadrature algorithm [2]. The original algorithm generally estimates $\mathrm{tr}(f(\mathbf{A}))$ with $\mathbf{A}$ as an symmetric positive-definite (SPD) matrix and f as any analytical matrix function via stochastic trace estimation [3]. In our application, $\log\det(\mathbf{A}) = \mathrm{tr}(\log(\mathbf{A}))$.

The key idea of stochastic trace estimation is that for a given SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and a set of random probe vectors $\mathbf{z}$ with mean 0 and variance 1 (e.g., vectors with entries as Rademacher random variables), $\mathrm{tr}(\log(\mathbf{A})) = \mathbb{E}(\mathbf{z}^T \log(\mathbf{A})\mathbf{z})$. The terms $\mathbf{z}^T \log(\mathbf{A})\mathbf{z}$ can be approximated via the Lanczos decomposition [4]. First, calculate the m-degree Lanczos decomposition $\mathbf{A} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$ with the first column of $\mathbf{Q}$ initialized as $\mathbf{z}/\|\mathbf{z}\|$. Then eigendecompose $\mathbf{T}$, and we will have the approximation:

$$\mathbf{z}^T \log(\mathbf{A})\mathbf{z} \approx n\mathbf{v}_1^T \log(\mathbf{\Lambda})\mathbf{v}_1,$$

where $\mathbf{v}_1$ is the first eigenvector of $\mathbf{T}$, $\log(\mathbf{\Lambda})$ is the eigenvalue matrix of $\mathbf{T}$ with its diagonal taken the logarithm.

Furthermore, the derivative of $\log\det(\mathbf{A})$ can also be efficiently calculated based on the building blocks we already have. From the Lanczos decomposition, we can derive:

$$\mathbf{A}^{-1}\mathbf{z} \approx n\mathbf{Q}\left(\mathbf{T}^{-1}\right)_{:,1},$$

where $\left(\mathbf{T}^{-1}\right)_{:,1}$ is the first column of $\mathbf{T}^{-1}$. Then, also using the idea of stochastic trace estimation, the derivative is approximated as:

$$\begin{aligned}\frac{\partial \log\det(\mathbf{A})}{\partial \boldsymbol{\theta}_i} = \mathrm{tr}\left(\mathbf{A}^{-1}\frac{\partial \mathbf{A}}{\partial \boldsymbol{\theta}_i}\right) &\approx \frac{1}{n_z}\sum \mathbf{z}^T\mathbf{A}^{-1}\frac{\partial \mathbf{A}}{\partial \boldsymbol{\theta}_i}\mathbf{z} \\ &= \frac{n}{n_z}\sum\left(\mathbf{Q}\mathbf{V}\mathbf{\Lambda}^{-1}\left(\mathbf{V}^T\right)_{:,1}\right)^T\frac{\partial \mathbf{A}}{\partial \boldsymbol{\theta}_i}\mathbf{z},\end{aligned}$$

where $\mathbf{V}$ is the eigenvector matrix of $\mathbf{T}$, n_z is the number of probe vectors. The above Lanczos method for Log-Determinant Estimation is given in the following algorithm:

Algorithm 1 Lanczos method for Log-Determinant Estimation

Input: A SPD matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, degree m, and number of probe vectors n_z.
Output: Estimation of $\log\det(\mathbf{A})$ and $\partial \log\det(\mathbf{A})/\partial\boldsymbol{\theta}_i$

$logdet \leftarrow 0$, $dlogdet \leftarrow [0, 0, \ldots, 0]$
for $k = 1$ to n_z **do**
  Generate a Rademacher random vector $\mathbf{z}_k$, form unit vector $\mathbf{u}_k = \mathbf{z}_k/\|\mathbf{z}_k\|$.
  $\mathbf{Q}, \mathbf{T} \leftarrow$ LanczosDecomposition $(\mathbf{A}, \mathbf{u}_k, m)$, which is the m-degree Lanczos decomposition of $\mathbf{A}$ with $\mathbf{u}_k$ as the initial column of $\mathbf{Q}$.
  $\mathbf{V}, \mathbf{\Lambda} \leftarrow \text{eig}(\mathbf{T})$
  $\mathbf{v}_1 \leftarrow$ first column of $\mathbf{V}$
  $logdet \leftarrow logdet + \frac{n}{n_z}\mathbf{v}_1^T\mathbf{\Lambda}\mathbf{v}_1$
  $\mathbf{r}_1^T \leftarrow$ first column of $\mathbf{V}^T$
  $\mathbf{a} \leftarrow \mathbf{Q}\mathbf{V}\mathbf{\Lambda}^{-1}\mathbf{r}_1^T$
  for i = 1 to the length of hyperparameters **do**
    $dlogdet[i] \leftarrow dlogdet[i] + \frac{n}{n_z}\mathbf{a}^T\frac{\partial\mathbf{A}}{\partial\boldsymbol{\theta}_i}\mathbf{z}$
  end for
end for
return *logdet*, *dlogdet*

The Lanczos method only requires the calculation of matrix-vector multiplication (MVM) operations with respect to $\mathbf{A}$, which leverages any possible structures in $\mathbf{A}$ that facilitate the MVM.

Linear Conjugate Gradients

Linear conjugate gradients (CG) is a method served to calculate the $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a SPD matrix. The difference between Linear CG and the Cholesky decomposition is that the Cholesky Decomposition solves the problem directly, whereas the linear CG approximates the solution by solving the convex optimization problem:

$$\min_x f(x) := \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}. \tag{0.1}$$

The solution of this problem satisfies $\mathbf{A}\mathbf{x} = \mathbf{b}$. Linear CG solves the problem iteratively by decreasing the residual $\mathbf{r_k} = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ every iteration, where the subscript k means the k-th iteration. Starting from an initial guess $\mathbf{x}_0$, the initial descent direction $\mathbf{p}_0$ is set to $-\mathbf{r}_0$, which is the opposite direction of the gradient of 0.1 at the initial point $\mathbf{x}_0$. Then the algorithm iterates through the following steps:

1. Determine the step length α_k along the descending direction $\mathbf{p}_k$.
2. Update the solution $\mathbf{x}_{k+1}$ with $\mathbf{x}_k + \alpha_k\mathbf{p}_k$.
3. Calculate and evaluate whether the residual $\mathbf{r_k} = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ satisfies the convergence criteria. If so, return the found solution, otherwise:

4. Find the next descending direction $\mathbf{p}_{k+1}$ according to the previous direction and residual information as $\mathbf{p}_{k+1} = \beta_k \mathbf{p}_k - \mathbf{r}_{k+1}$, where β_k is calculated to achieve the conjugate property of the new descending direction and all previous descending directions.

Note that, in step 4, the new descending direction should be conjugate to previous steps with respect to $\mathbf{A}$ [5]. That is, $\mathbf{p}_{k+1}^T \mathbf{A} \mathbf{p}_i = 0, \forall i = 0, 1, \ldots, k$. The procedures are elaborated in the following algorithm:

Algorithm 2 Linear Conjugate Gradients

Input: A SPD matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, a vector $\mathbf{b} \in \mathbb{R}^{n \times 1}$, initial guess $\mathbf{x}_0$, max iteration k_{max}, and convergence threshold ϵ.

Output: Estimation of $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

1: $\mathbf{r}_0 \leftarrow \mathbf{A}\mathbf{x}_0 - \mathbf{b}$
2: **if** $\|\mathbf{r}_0\| < \epsilon$ **then return** $\mathbf{x}_0$
3: **end if**
4: $\mathbf{p}_0 \leftarrow -\mathbf{r}_0$
5: **for** $k = 1$ to k_{max} **do**
6: $\quad \alpha_k \leftarrow \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$
7: $\quad \mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k \mathbf{p}_k$
8: $\quad \mathbf{r}_{k+1} \leftarrow \mathbf{A}\mathbf{x}_{k+1} - \mathbf{b}$
9: $\quad$ **if** $\|\mathbf{r}_{k+1}\| < \epsilon$ **then return** $\mathbf{x}_{k+1}$
10: $\quad$ **end if**
11: $\quad \beta_k \leftarrow \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$
12: $\quad \mathbf{p}_{k+1} \leftarrow \beta_k \mathbf{p}_k - \mathbf{r}_{k+1}$
13: **end for**

This method guarantees to converge within n steps. In every iteration, the calculations only involve MVM with respect to $\mathbf{A}$, thus the time complexity is $\mathcal{O}(n^2)$ per iteration. The iteration number in implementation, however, is a variable related to the convergence criteria and is often independent of n and much lesser than n, thus often omitted in recent works [6]. As a result, linear CG has an efficiency advantage over Cholesky Decomposition ($\mathcal{O}(n^3)$) when n is large.

Nonlinear Conjugate Gradients

The nonlinear CG is an optimization method used to solve general nonlinear functions. In the context of Gaussian process regression (GPR), it is used to minimize the negative marginal log-likelihood (NMLL). The difference between linear CG and nonlinear CG is that:

1. The step length α_k in line 7 of algorithm 2 is calculated as a closed-form expression as the exact minimizer along the conjugate directions $\mathbf{p}_k$. In nonlinear CG, this step length is acquired by using Wolfe line search algorithm [7] as an approximated minimizer.

2. The residual r_k in linear CG is the gradient of 0.1. On the other hand, r_k represents the gradient of the true objective function instead.

3. Different formulas for β_k can be used in nonlinear CG. This thesis uses the formula derived by Polak-Ribiere (PR):

$$\beta_k^{PR} = \frac{\mathbf{r}_{k+1}^T(\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{r}_k^T\mathbf{r}_k}.$$

The resulting algorithm is:

Algorithm 3 Nonlinear Conjugate Gradients

Input: A target function f, initial guess $\mathbf{x}_0$, max iteration k_{max}, and convergence threshold ϵ.

Output: Estimation of $\mathbf{x} = \text{argmin} f(\mathbf{x})$.

$\mathbf{r}_0 \leftarrow \nabla f(\mathbf{x}_0)$
if $\|\mathbf{r}_0\| < \epsilon$ **then return** $\mathbf{x}_0$
end if
$\mathbf{p}_0 \leftarrow -\mathbf{r}_0$
for $k = 1$ to k_{max} **do**
    $\alpha_k \leftarrow \alpha$ from Wolfe line search
    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha_k\mathbf{p}_k$
    $\mathbf{r}_{k+1} \leftarrow \nabla f(\mathbf{x}_{k+1})$
    if $\|\mathbf{r}_{k+1}\| < \epsilon$ **then return** $\mathbf{x}_{k+1}$
    end if
    $\beta_k \leftarrow \frac{\mathbf{r}_{k+1}^T(\mathbf{r}_{k+1}-\mathbf{r}_k)}{\mathbf{r}_k^T\mathbf{r}_k}$
    $\mathbf{p}_{k+1} \leftarrow \beta_k\mathbf{p}_k - \mathbf{r}_{k+1}$
end for

In practice, nonlinear CG shows more promising performance than merely line search. It can follow narrow (ill-conditioned) valleys, where the steepest descent method slows down and follows a criss-cross pattern.

1 Introduction

This thesis mainly focuses on the topic of a specific data modeling technique: Gaussian process (GP). GP is a non-parametric modeling technique that assumes a multivariate Gaussian distribution among all input points. The theory of GP has been well-studied in literature [8]. The focus of this thesis is a popular tool provided by GP: the GPR, which, from its name, performs regression based on GP modeling.

GPR has a wide range of applications in the literature. In the field of Chemistry, Chen et al. have shown that using GPR to perform calibration on spectroscopic data exhibited enhanced results than traditional methods such as principal component regression [9]. Krems R. V. successfully applied GPR to predict the quantum mechanical properties of molecules [10]. Burn, Matthew J. and Popelier, Paul L. A. used GPR to develop force fields for atoms and accurately predict the atomic energies. Deringer et al. provide a comprehensive review of the implementation of GPR in computational materials science and chemistry which focuses on various atomistic properties [11]. GP also has a strong theoretical connection with deep neural networks, where infinitely wide deep networks and GPs are equivalent according to Lee et al. [12]. Moreover, Wilson et al. developed GPR networks that combine the structural properties of Bayesian neural networks with the non-parametric flexibility of GP [13]. Other applications include but are not limited to battery health degradation forecast [14], wind speed forecasting for power generation [15], and outlier detection in industry [16]. In the domain of control theory, GPR also appears to enhance the performance of model predictive control (MPC), specifically, the nonlinear model predictive control (NMPC). The first paper that described the possibility of implementing GPR into a NMPC model is by Kocijan et al. where they used GPR to model the nonlinearity and eventually implemented the improved NMPC model on a benchmark pH process control [17]. Later, Klenske et al. managed to use GPR to specifically model periodic time-varying disturbances to improve NMPC [18]. Due to the high computational cost of the standard GPR, different approximation models of the GPR are also implemented in NMPC where there are time constraints. Researchers have paid attention to leveraging low-rank approximations of the GP to facilitate its performance. Pan et al. implemented the sparse spectral Gaussian process (SSGP) approximation of GPR to do high-performance driving [19]. Hewing et al. implemented the fully independent training conditional (FITC) approximation of GPR on the task of autonomous racing and achieved the fastest performance of a NMPC at that time [20]. For a comprehensive tutorial on implementing the GPR, readers could refer to [21].

Though GPR appears to be a powerful modeling technique, the time required to perform such modeling could be exceptionally long for a large amount of data.

In this thesis, we will explore the existing approximations of GPR, typically, the low-rank approximations of GPR, and develop an even faster approximation to overcome the problem of the notoriously high time consumption of performing GPR.

In this chapter, the basics of GPR are explained in detail in Section 1.1. Then, the problem statement will be presented in Section 1.2. Contributions of this thesis are briefly discussed in Section 1.3. Finally, the outline of this thesis is presented in Section 1.4.

1.1 Gaussian Process Regression

Regression tasks deal with the problem of determining the relationship between a given set of input $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]^T$ and output $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_n)]^T$ and leveraging it to make predictions. In the setting of GP, the output $\mathbf{f}$ is assumed to be a set of random variables. Any finite set of $\mathbf{f}$ follows a joint Gaussian distribution [8]:

$$p(\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{f}, \mathbf{K}), \tag{1.1}$$

$\mathbf{K}$ is referred to in existing literature as the covariance matrix, the kernel matrix, or the Gram matrix, where each entry is calculated by a kernel function of two inputs.

$$\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{1.2}$$

Because the GP models the distribution of an arbitrary amount of points, in general, the GP is a non-parametric model that can be defined by a mean function and a covariance function:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}_i, \mathbf{x}_j)), \tag{1.3}$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$, $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}[f(\mathbf{x}_i) - m(\mathbf{x})][f(\mathbf{x}_j) - m(\mathbf{x})]$.

We can also consider GP in a Bayesian linear regression context. Consider the linear model $f(\mathbf{x}) = \phi(\mathbf{x})^T\mathbf{w}$ with prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_n)$. $\phi(\mathbf{x})$ is a set of basis functions, also referred to as feature vectors, with $\mathbf{x}$ as the variable. These basis functions in general could be any functions. If we evaluate the joint distribution of $f(\mathbf{x})$ at different points, we arrive at a GP with

$$f(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\phi}(\mathbf{x})^T\mathbb{E}(\mathbf{w}),\ \boldsymbol{\phi}(\mathbf{x}_i)^T\mathbb{E}(\mathbf{w}^T\mathbf{w})\boldsymbol{\phi}(\mathbf{x}_j)) = \mathcal{GP}(\mathbf{0},\ \boldsymbol{\phi}(\mathbf{x}_i)^T\Sigma_n\boldsymbol{\phi}(\mathbf{x}_j)). \tag{1.4}$$

Notice that $\boldsymbol{\Sigma}_n$ is positive definite, we can rewrite the kernel as $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\Sigma}_n\boldsymbol{\phi}(\mathbf{x}_j) = \boldsymbol{\psi}(\mathbf{x}_i)^T\boldsymbol{\psi}(\mathbf{x}_j)$, where $\boldsymbol{\psi}(\mathbf{x}) = \boldsymbol{\Sigma}_n^{\frac{1}{2}}\boldsymbol{\phi}(\mathbf{x})$. This representation shows that a kernel can be described as an inner product in the feature space and vice versa. In most cases, directly manipulating the kernel representation instead of the feature vectors would be more efficient. This is referred to in the literature as the *kernel trick*.

GPR is the process of learning the mean and kernel function of a GP based on given data. The detailed procedure of GPR is shown in the following subsections.

1.1.1 Assuming Prior Distribution

GPR starts with assuming a prior distribution on the training output. For the mean of the prior, it is usually assumed to be $\mathbf{0}$ in literature with no loss of generality.

Since when calculating the posterior mean, one can subtract the mean values from the observations, which results in a 0-mean data, then perform the GPR and add the mean values back. Thus, in the following studies, the prior mean of $\mathbf{f}$ will always be set to $\mathbf{0}$.

On the other hand, the prior covariance is specified by choosing a set of kernel functions and initializing its hyperparameters, which further determines the kernel matrix as in 1.2. The most commonly used kernel function is the radial basis function (RBF) kernel, also referred to as the square exponential (SE) kernel:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_s^2 \exp(-\frac{1}{2l^2}(\mathbf{x}_i - \mathbf{x}_j)^2). \tag{1.5}$$

This function has two hyperparameters: signal power σ_s^2 and length-scale l. Varying the signal power σ_s^2 will change the amount of consideration for noise. Varying the length-scale l will render the function smoother or more fluctuated.

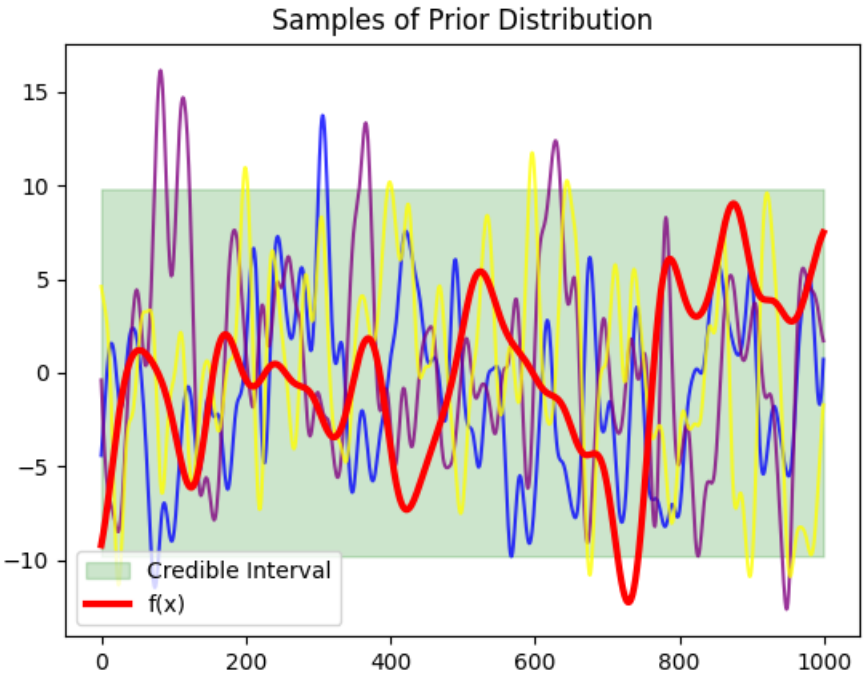


Figure 1.1: 4 samples drawn from a zero-mean prior distribution. The sample in thick red is initialized with a length scale of 30, which will be considered in the following experiments as the target function. Other curves are randomly drawn from initializations with a length scale of 10.

Figure 1.1 is an illustration of some function outputs drawn from a 0-mean Gaussian process using an RBF kernel function with $\sigma_s^2 = 16$. The sample depicted in red is initialized with $l = 30$. **This sample of data points will be taken as the underlying function we aim to model with GPR and other methods throughout the paper**. The samples in other colors are 3 other function samples randomly drawn

from an initialization with $l = 10$. The green rectangular area is the credible interval which is the area between $m(\mathbf{x}) \pm 1.96 * \sigma$, where σ is the standard deviation calculated by the kernel function. Currently, it is a rectangular area since the prior mean is always assumed to be 0, and we used a *stationary* kernel (i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i - \mathbf{x}_j)$) to simulate. It seems that the curves are continuous and assemble normal functions in a general context. However, one should consider them as an infinite amount of points evaluated by a joint Gaussian distribution, instead of points drawn from a certain continuous function.

Formally, consider a training set with inputs $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ ..., \mathbf{x}_n\}$ and outputs $\mathbf{y} = \{y_1, y_2, \ ..., y_n\}$ drawn from the underlying distribution, and a set of test inputs $\mathbf{X}_*$ where we want to predict the output value $\mathbf{f}_*$ at these points. Assume additive independent white Gaussian noise on the training observations, i.e., $\mathbf{y} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I})$, we can write the prior distribution for all data points:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \ \begin{bmatrix} \mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_n^2 \mathbf{I} & \mathbf{K}_{\mathbf{X},*} \\ \mathbf{K}_{\mathbf{X},*}^T & \mathbf{K}_{*,*} \end{bmatrix}\right), \tag{1.6}$$

where $\sigma_n^2 \mathbf{I}$ is a diagonal matrix representing the covariance of all noise. $\mathbf{K}_{\mathbf{X},\mathbf{X}} = \mathbf{K}(\mathbf{X}, \mathbf{X})$, $\mathbf{K}_{\mathbf{X},*} = \mathbf{K}(\mathbf{X}, \mathbf{X}_*)$, $\mathbf{K}_{*,*} = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*)$, $\mathbf{K}(\cdot, \cdot)$ is the kernel matrix with corresponding inputs.

1.1.2 Learning the Hyperparameters

As we intend to learn a GP model towards an underlying target distribution (e.g., the red curve in Figure 1.1), instead of fitting one function that describes the relationship between input and output (e.g., $f(\mathbf{x}) = \mathbf{x}^\mathbf{T}\mathbf{w}$ for a simple linear case), GPR fits the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ and thus the kernel matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}}$. The kernel matrix plays a crucial role in evaluating the posterior distribution as shown in the next subsection.

Though the form of the kernel function is fixed at the beginning, we learn its hyperparameters to adapt the training data. This is done by minimizing the NMLL function through optimization algorithms:

$$-\log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{2}(\mathbf{y}^T {\mathbf{K}_\mathbf{n}}^{-1} \mathbf{y} + \log\det(\mathbf{K}_\mathbf{n}) + n \log 2\pi), \tag{1.7}$$

where $\mathbf{K}_\mathbf{n}$ is the shorthand notation for $\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_n^2 \mathbf{I}$, $\boldsymbol{\theta}$ represents the hyperparameters. Notice that $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ is parameterized by $\boldsymbol{\theta}$. $\det(\cdot)$ is the determinant function, n is the number of training inputs. The maximization process is completed by optimization algorithms such as line search and conjugate gradients. These algorithms require the derivative of negative marginal log-likelihood w.r.t. hyperparameters (dNMLL) to estimate the next iteration step. The gradient information is calculated as follows:

$$\frac{\partial - \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} = \frac{1}{2}\left(-\mathbf{y}^T {\mathbf{K}_\mathbf{n}}^{-1} \frac{\partial \mathbf{K}_{\mathbf{X},\mathbf{X}}}{\partial \boldsymbol{\theta}_i} {\mathbf{K}_\mathbf{n}}^{-1} \mathbf{y} + \mathrm{tr}\left({\mathbf{K}_\mathbf{n}}^{-1} \frac{\partial \mathbf{K}_{\mathbf{X},\mathbf{X}}}{\partial \boldsymbol{\theta}_i}\right)\right), \tag{1.8}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of a matrix. As mentioned in Section 1.2.1, the optimization process requires an iterative calculation of the inverse and log-determinant of

the $n \times n$ matrix $\mathbf{K_n}$. To alleviate the computation burden, one can already implement the Cholesky decomposition on $\mathbf{K_n}$ to efficiently calculate both terms.

1.1.3 Evaluating the Posterior Distribution

After learning the most suitable kernel function, we recompute the kernel matrices using the optimized kernel function and condition the prior on the test inputs. The result is given as [8]:

$$\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\bar{\mathbf{f}}_*, Cov(\mathbf{f}_*)), \tag{1.9}$$

where:

$$\bar{\mathbf{f}}_* = \mathbf{K}_{\mathbf{X},*}^T(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, \tag{1.10a}$$

$$Cov(\mathbf{f}_*) = \mathbf{K}_{*,*} - \mathbf{K}_{\mathbf{X},*}^T(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I})^{-1}\mathbf{K}_{\mathbf{X},*}. \tag{1.10b}$$

That is, we have acquired the expected value of the output $\bar{\mathbf{f}}_*$ at test inputs, along with the confidence $Cov(\mathbf{f}_*)$ we have in the predictions. Figure 1.2 shows the results of the GPR.

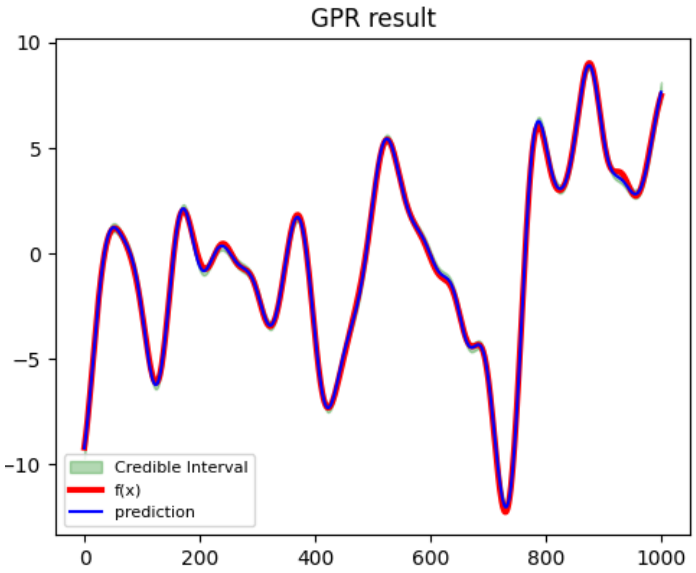


Figure 1.2: Posterior Distribution of the GPR

In Figure 1.2, the red curve is the underlying function. The blue curve is the posterior mean calculated with hyperparameters after training with 1000 noisy training points, which is taken as the prediction of the GPR. The green area, though nearly non-existent, is the credible interval showing that we are very confident about the predicted results. We can clearly see that, compared with Figure 1.1, a GP that is more fit to the underlying function is obtained.

In sum, The crux of GPR is to learn the hyperparameters of a suitable kernel function that embeds the underlying features of the data set. Then we can use the kernel function to form the GP model and calculate the posterior distributions.

However, it is well known that GPR do not scale well to large amounts of data. This is because optimization of hyperparameters requires the calculation of $(\mathbf{K}_n)^{-1}$ and $\log\det(\mathbf{K}_n)$ of the kernel matrix in every iteration as shown in 1.7 and 1.8. Typically, this step requires $\mathcal{O}(n^3)$ time complexity using the Cholesky decomposition. Numerous methods have been proposed to calculate a low-rank approximation of the kernel function or kernel matrix in order to facilitate the calculation of these terms, which will be explained in detail in Chapter 2.

1.2 Problem Statement

In this section, we will formulate the main problem addressed in this thesis. It will begin from the general problem of GPR and narrow down to a specific bottleneck of the state-of-the-art solution.

1.2.1 Bottleneck of GPR

In the context of GPR, modeling a dataset is to study the posterior mean and covariance of the GP. This is accomplished by training hyperparameters in the model to capture the underlying data distribution. The training procedure is an optimization process that requires iterative calculation of the inverse and log-determinant of a $n \times n$ matrix, where n is the number of training points. The calculations of the terms are $\mathcal{O}(n^3)$ time complexity operations. In many real-world applications, e.g., the NMPC model, either the vast quantity of training data or the requirement of frequent updates prevents the direct use of precise GPR. To alleviate the high calculation burden, many approximation techniques have been proposed.

A review of the most popular approximation techniques is conducted by Liu et al. [22]. In their review, approximations are first categorized into global approximations and local approximations. Local approximations are often implemented in distributed systems, thus not suitable for this thesis. Global approximation methods are further classified into subset-of-data, sparse kernels, and sparse approximations. The first two categories deliberately omit part of the training information in order to achieve faster computations which is a naive way of treating the problem. The sparse approximations also referred to as low-rank approximations, are the current state-of-the-art way of achieving fast GP calculations. The general form of low-rank approximations is to approximate the $n \times n$ kernel matrix as:

$$\mathbf{K_{nn}} \approx \mathbf{A}_{mn}^T \mathbf{B}_{mm} \mathbf{A}_{mn}, \tag{1.11}$$

where $\mathbf{A}_{mn} \in \mathbb{R}^{m \times n}$, $\mathbf{B}_{mm} \in \mathbb{B}^{m \times m}$. In such a way, the calculation of the inverse and log-determinant can be downgraded from a $n \times n$ matrix to a $m \times m$ matrix where $m < n$ with a certain loss of accuracy. The existing techniques will be presented in Chapter 2.

Among the existing low-rank approximations, the cutting-edge methods are the structural approximations based on the structural kernel interpolation (SKI). This advanced approach enables the approximate computation of GPR with a remarkably efficient time complexity of only $\mathcal{O}(n + m^2)$. Here, m represents the number of grid

points within the SKI system and is considered a predefined hyperparameter. Nevertheless, presently, there is an absence of well-defined strategies for establishing the optimal value of m. Notably, while there are documented methods in the literature aimed at addressing the exponential growth of m when dealing with an increasing number of data dimensions [23][24][25], it remains unclear how to precisely determine the number of grid points in each dimension. This suggests that the value of m may either be excessively high, leading to increased time consumption without commensurate gains in accuracy, or it may be insufficiently high to attain the desired level of accuracy.

1.2.2 Problem Statement

In this thesis, the main problem is to find a systematic way of determining the number of grid points per each dimension in SKI-based structural approximations of GPR so that m is optimally defined to achieve the desired level of accuracy in the shortest time.

1.2.3 Methodology

The procedures that we take to solve the main problem of this thesis are listed as follows:

1. Study the theory of GPR.
2. Explore the literature and summarize existing low-rank approximations of GPR.
3. Propose a new approximation method that further improves the state-of-the-art approximation method in terms of reaching the same level of accuracy in a shorter time.
4. Experiment on the proposed new approximation to prove its efficiency.

The study of GPR is summarized in the previous section. The rest of the procedures will be allocated in different chapters in this thesis.

1.3 Contributions

In this thesis, we produced the following contributions:

- (**Main**) We present a novel low-rank approximation framework, denoted as malleable kernel interpolation for scalable structured Gaussian process (MKISSGP), which extends the capabilities of the established state-of-the-art SKI-based kernel interpolation for scalable structured Gaussian process (KISSGP) approximation. A key feature of MKISSGP is the introduction of a flexible grid point determination strategy. This strategy effectively minimizes the number of grid points required to achieve a desired level of accuracy, serving as the centerpiece of our innovation.

- (Auxiliary) We studied and summarized the theory of Gaussian process regression (GPR). We confirmed that the bottleneck of the calculation of GPR comes from the calculation of $(\mathbf{K}_{\mathbf{X},\mathbf{X}}+\sigma_n^2\mathbf{I})^{-1}$, $\log\det(\mathbf{K}_{\mathbf{X},\mathbf{X}}+\sigma_n^2\mathbf{I})$, and their derivatives w.r.t. hyperparameters.
- (Auxiliary) We provided a comprehensive summary of the currently available low-rank approximations. Furthermore, we conducted a comparative experiment to assess and contrast their time requirements for achieving accuracy levels akin to those of the precise GPR. Our findings lead to the conclusion that among these approximations, the SKI-based KISSGP approach exhibits the highest efficiency.

1.4 Outline

This thesis is organized as follows.

In Chapter 1, an introduction to GPR and the problem statement are first given. Then the implemented iterative algorithms are provided explicitly for reference. The contributions of this thesis are then summarized.

In Chapter 2, several existing low-rank approximations of GPR are explained in detail and compared with respect to accuracy and time complexity.

In Chapter 3, the proposed low-rank approximation is developed and compared with the state-of-the-art method.

In Chapter 4, the proposed low-rank approximation is tested in different aspects to show its ability to further shrink the time consumption.

In Chapter 5, the conclusion of this thesis will be drawn.

Existing Low-Rank Approximations

2

There is a large body of work devoted to low-rank approximations of the kernel matrix. In this paper, we categorize them into four categories: Nyström approximations, prior approximations, spectral approximations, and structural approximations.

Nyström approximations is a family of methods that typically uses numerical techniques to approximate the kernel matrix. In this thesis, we only elaborate on the most classical Nyström approximation [26] because of its wide popularity in later works citations.

Prior approximations introduce a special set of inputs called *inducing variables* $\mathbf{U}$. The low-rank approximations are formulated by assuming different probabilistic properties between the inducing variables and the training and test inputs. This is a broad family of methods that are still implemented in recent applications. The most classical and popular prior approximations: the subset of regressors (SoR) approximation [27] and the fully independent training conditional (FITC) approximation [28] will be explained in detail, while the deterministic training conditional (DTC) approximation [29] will be briefly explained in transition between these two methods. the

While previous methods focus on numerical techniques and probabilistic assumptions, spectral approximations focus on the frequency analysis of the kernel function. These approximations reconstruct the kernel function by finite sets of basis functions. The sparse spectrum Gaussian process (SSGP) approximation [30] is a method of this family with wide implementation and citations in literature, thus will be elaborated in this chapter. The Hilbert space method [31], though less popular than SSGP, introduces a different perspective to approach the frequency domain of the kernel function, which is also worth explaining.

Structural approximations are currently the state-of-the-art and most popular family of low-rank approximations. There is even work devoted to using GPU to further accelerate the computation of the approximations of this family [32]. The first structural approximation paper [33] proposed the kernel interpolation for scalable structured Gaussian process (KISSGP) approximation that introduced a novel framework called structural kernel interpolation (SKI) that approximates the kernel matrix through interpolation from grid points. Later studies focus on different aspects of SKI to further improve its efficiency [6][23], which results in the grid-structured Gaussian process (GSGP) approximation and the sparse structured kernel interpolation (SSKI) approximation.

Below is a table of the methods showing their corresponding time complexities:

Category	Method Name	Time Complexity
GPR	GPR	$\mathcal{O}(n^3)$
Nyström Approx.	Nyström	$\mathcal{O}(m^2n)$
Prior Approx.	SoR	$\mathcal{O}(m^2n)$
	DTC	$\mathcal{O}(m^2n)$
	FITC	$\mathcal{O}(m^2n)$
Spectral Approx.	SSGP	$\mathcal{O}(m^2n)$
	Hilbert Space	$\mathcal{O}(m^2n)$
Structural Approx.	kissgp	$\mathcal{O}(n+m^2)$
	GSGP	$\mathcal{O}(n+m^2)$
	SSKI	$\mathcal{O}(n+m^2)$

Table 2.1: Overview of existing approximation methods

In the table we see the time complexity is expressed in terms of n and m, where n is the total number of training points, and m is a controllable number smaller than n.

In the following sections, a detailed analysis of the methods is presented in the first four sections. Also, for the methods with wide implementations or significance (the most popular or classical methods), we will give the final approximated form of the kernel function, the posterior for one test point, and the marginal likelihood. At the end of this chapter, the performance and plots of the resulting posterior of these methods will be compared with the results of GPR with explanations.

2.1 Nyström Approximation

The Nyström Approximation is a method proposed to numerically speed up kernel methods such as GPR [26]. It is formulated by selecting m training points from all original training points, and composing the kernel matrix as a product of the sub-matrices corresponding to the selected training points. The approximation for the training kernel matrix is given by

$$\tilde{\mathbf{K}} = \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}, \tag{2.1}$$

where $\tilde{\mathbf{K}}$ is the approximation, $\mathbf{K}_{\mathbf{U},\mathbf{X}} \in \mathbb{R}^{m\times n}$ is the kernel matrix corresponding to the selected training inputs and all the training inputs, $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ is the kernel matrix of the training inputs. In other words, we only have to acquire m rows of the kernel matrix with $m \ll n$ to approximate the original matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ in the form of matrix multiplication. This is also equivalent to approximating the kernel function as:

$$k(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{k}_i^T\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{k}_j, \tag{2.2}$$

where $\mathbf{k}_i$ is the column vector with every entry as the kernel function of $\mathbf{x}_i$ and every selected training input.

Recall that the calculation of GPR involves the inversion and determinant of $\mathbf{K_n}$. Now, using the matrix inversion lemma and the matrix determinant lemma, we acquire the approximations of both the inversion and determinant:

$$\mathbf{K_n}^{-1} \approx \sigma_n^{-2}\mathbf{I} - \sigma_n^{-4}\mathbf{K_{X,U}}\mathbf{A}^{-1}\mathbf{K_{U,X}}, \tag{2.3a}$$

$$\det(\mathbf{K_n}) \approx \sigma_n^{2n}\det(\mathbf{K_{U,U}^{-1}})\det(\mathbf{A}), \tag{2.3b}$$

where $\mathbf{A} = \mathbf{K_{U,U}} + \sigma_n^{-2}\mathbf{K_{U,X}}\mathbf{K_{X,U}}$. Notice that the matrix under the inversion and determinant operation in this altered form is $m \times m$, and m can be much smaller than n.

The main idea behind the Nyström approximation is using an empirical average of m samples to replace the integration over the probability distribution of input samples in the original eigenfunction problem. That is from

$$\int k(\mathbf{y}, \mathbf{x})\phi_i(\mathbf{x})p(\mathbf{x})d\mathbf{x} = \lambda_i\phi_i(\mathbf{y}), \tag{2.4}$$

where $\phi_i(\mathbf{x})$ is the i-th eigenfunction of $k(\cdot, \cdot)$ and λ_i is the corresponding eigenvalue, to

$$\frac{1}{m}\sum_{j=1}^{k} k(\mathbf{y}, \mathbf{x}_j)\phi_i(\mathbf{x}_j) \approx \lambda_i\phi_i(\mathbf{y}). \tag{2.5}$$

Substituting 2.5 in the setting of a $m \times m$ matrix eigenproblem, we arrive at the following approximation

$$\phi_i(\mathbf{x}_j) \approx \sqrt{m}\mathbf{U}_{j,i}^{(m)}\ , \qquad \lambda_i \approx \frac{\lambda_i^{(m)}}{m}, \tag{2.6}$$

where $\mathbf{U}_{j,i}^{(m)}$ is the j-th element of the i-th eigenvector of the $m \times m$ kernel matrix. Denoting the approximation of the full kernel matrix as $\tilde{\mathbf{K}}$. Further plugging 2.6 into its eigendecomposition regarding the eigenfunctions as eigenvectors, we arrive at the approximations for the eigenvectors $\tilde{\mathbf{v}}_i^{(n)}$, and eigenvalues $\tilde{\lambda}_i^{(n)}$ of the original kernel matrix

$$\tilde{\mathbf{U}}_i^{(n)} \approx \sqrt{\frac{m}{n}}\frac{1}{\lambda_i^{(m)}}\mathbf{K_{X,U}}\mathbf{v}_i^{(m)}, \qquad \lambda_i^{(n)} \approx \frac{n}{m}\lambda_i^{(m)}. \tag{2.7}$$

Finally, using the above expressions to evaluate the eigendecomposition $\mathbf{K_{X,X}} = \mathbf{U\Lambda U}^T$, we find the result in 2.1.

Based on the results above, the Nyström approximates the kernel function, posterior mean, and covariance, and the log-likelihood in the following form:

$$\begin{aligned}
&\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{k}_i^T\mathbf{K_{U,U}^{-1}}\mathbf{k}_j,\\
&\textbf{posterior mean} : \bar{f}_* \approx \mathbf{k_{X,*}^T}\mathbf{L}^{-1}\mathbf{y},\\
&\textbf{posterior covariance} : cov(f_*) \approx k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k_{X,*}^T}\mathbf{L}^{-1}\mathbf{k_{X,*}},\\
&\textbf{NMLL} : \log p(\mathbf{y}) \approx \frac{1}{2}[\mathbf{y}^T\mathbf{L}^{-1}\mathbf{y} + \log\det(\mathbf{K_{U,U}^{-1}}) + \log\det(\mathbf{A}) + n\log 2\pi\sigma_n^2],
\end{aligned} \tag{2.8}$$

where $\mathbf{k}_{\mathbf{X},*}$ is the column corresponding to $\mathbf{x}_*$ in $\mathbf{K}_{\mathbf{X},*}$, $\mathbf{L}^{-1} = \sigma_n^{-2}\mathbf{I} - \sigma_n^{-4}\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{A}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}$ is the inverse of $(\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}} + \sigma_n^2\mathbf{I})$ using the matrix inversion lemma, $\mathbf{A} = \mathbf{K}_{\mathbf{U},\mathbf{U}} + \sigma_n^{-2}\mathbf{K}_{\mathbf{U},\mathbf{X}}\mathbf{K}_{\mathbf{X},\mathbf{U}}$. Notice that in 2.8, we now express the posterior mean and posterior covariance in terms of a single output, which will also be the case in other sections. From 2.8 we can see that the determinant and inverse are now directly taken towards a $m \times m$ matrix, thus the time complexity for this term is decreased to $\mathcal{O}(n^3)$ to $\mathcal{O}(m^3)$. However, due to the multiplication with matrix $\mathbf{K}_{\mathbf{X},\mathbf{U}}$, the overall time complexity is $\mathcal{O}(m^2 n)$.

The Nyström method is notorious for having the possibility of producing non-positive definite posterior covariance [34]. This prohibits the implementation of the Nyström method in practical use. We also exclude the results of the Nyström method from our comparison of existing methods, since a negative covariance is incomparable with a valid covariance.

2.2 Prior Approximation

Prior Approximations refer to a set of methods that produce approximations of the GP prior $p(\mathbf{f}, \mathbf{f}_*)$ in a Bayesian perspective with further assumptions. Many works categorize prior approximations to the Nyström approximation family due to the similarity in the approximation formulas. However, we consider them differently because the latter assumes a special probabilistic model, while the former approaches the problem by approximating the eigenproblem of the kernel function. It is also because of this strict probabilistic setting that prior approximations always guarantee a feasible result.

The probabilistic model, specifically, introduces a set of observations called *inducing variables* $\mathbf{U}$. Also, the train output and test output are assumed conditionally independent given $\mathbf{U}$ [34]. That is

$$p(\mathbf{f}, \mathbf{f}_*) = \int p(\mathbf{f}, \mathbf{f}_*|\mathbf{U})p(\mathbf{U})d\mathbf{U} \quad \approx \quad q(\mathbf{f}, \mathbf{f}_*) = \int p(\mathbf{f}|\mathbf{U})p(\mathbf{f}_*|\mathbf{U})p(\mathbf{U})d\mathbf{U}. \tag{2.9}$$

$\mathbf{U}$ is called inducing variables since the dependency between train and test samples is only communicated through $\mathbf{U}$. The precise conditional prior for $\mathbf{f}$ and $\mathbf{f}_*$ are in the same form as 1.10a, 1.10b in a noise-free case

$$p(\mathbf{f}|\mathbf{U}) = \mathcal{N}(\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{K}_{\mathbf{X},\mathbf{X}} - \mathbf{Q}_{\mathbf{X},\mathbf{X}}), \tag{2.10a}$$

$$p(\mathbf{f}_*|\mathbf{U}) = \mathcal{N}(\mathbf{K}_{*,\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}), \tag{2.10b}$$

with notation $\mathbf{Q}_{\mathbf{a},\mathbf{b}} \triangleq \mathbf{K}_{\mathbf{a},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{b}}$.

2.2.1 SoR Approximation

The first emerged method of this family, SoR, assumes a deterministic relationship between $\mathbf{U}$ and the train and test outputs [35] [27]:

$$p_{SoR}(\mathbf{f}|\mathbf{U}) \sim \mathcal{N}(\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{0}), \tag{2.11a}$$

$$p_{SoR}(\mathbf{f}_*|\mathbf{U}) \sim \mathcal{N}(\mathbf{K}_{*,\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{0}). \tag{2.11b}$$

Notice that the covariance is 0 for both training and test inputs. With 2.9, we calculate the joint prior as:

$$q_{SoR}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{Q}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I} & \mathbf{Q}_{\mathbf{X},*} \\ \mathbf{Q}_{*,\mathbf{X}} & \mathbf{Q}_{*,*} \end{bmatrix}\right). \tag{2.12}$$

With the same derivation from 1.6 to 1.9, also considering the fact that $\mathbf{y} \sim \mathcal{N}(\mathbf{f}, \sigma_n^2\mathbf{I})$, we get the approximated prediction formula

$$\begin{aligned} q_{SoR}(\mathbf{f}_*|\mathbf{y}) &\sim \mathcal{N}(\mathbf{Q}_{*,\mathbf{X}}(\mathbf{Q}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y},\ \mathbf{Q}_{*,*} - \mathbf{Q}_{*,\mathbf{X}}(\mathbf{Q}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I})^{-1}\mathbf{Q}_{\mathbf{X}},*) \\ &\sim \mathcal{N}(\sigma_n^{-2}\mathbf{K}_{\mathbf{U},*}^T\mathbf{A}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}\mathbf{y}, \mathbf{K}_{\mathbf{U},*}^T\mathbf{A}^{-1}\mathbf{K}_{\mathbf{U},*}), \end{aligned} \tag{2.13}$$

where $\mathbf{A} = \mathbf{K}_{\mathbf{U},\mathbf{U}} + \sigma_n^{-2}\mathbf{K}_{\mathbf{U},\mathbf{X}}\mathbf{K}_{\mathbf{X},\mathbf{U}}$ is the same as in Nyström approximation. Also, we can see that the prior for the training inputs has the exact same form as in Nyström, i.e., $\mathbf{Q}_{\mathbf{X},\mathbf{X}} = \tilde{\mathbf{K}} = \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}$. This means that we acquire the same log-likelihood function and the same time complexity $\mathcal{O}(m^2n)$.

Notice that the training prior only depends on $\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}$, but not all the training points, SoR has the problem of yielding very high confidence in prediction variance due to the degeneracy of the model [34] [35].

The overall SoR approximations are given as follows:

$$\begin{aligned} &\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{k}_i^T\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{k}_j, \\ &\textbf{posterior mean} : \bar{f}_* \approx \sigma_n^{-2}\mathbf{k}_*^T\mathbf{A}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}\mathbf{y}, \\ &\textbf{posterior covariance} : cov(f_*) \approx \mathbf{k}_*^T\mathbf{A}^{-1}\mathbf{k}_*, \\ &\textbf{NMLL} : \log p(\mathbf{y}) \approx \frac{1}{2}[\mathbf{y}^T\mathbf{L}^{-1}\mathbf{y} + \log\det(\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}) + \log\det(\mathbf{A}) + n\log 2\pi\sigma_n^2], \end{aligned} \tag{2.14}$$

where the symbols have the same meanings as in 2.8.

An improved method called DTC slightly alleviates the strictness of SoR by only imposing a deterministic training conditional, but leaving the test conditional intact [29]. That is

$$p_{DTC}(\mathbf{f}|\mathbf{U}) = \mathcal{N}(\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{0}), \tag{2.15a}$$

$$p_{DTC}(\mathbf{f}_*|\mathbf{U}) = \mathcal{N}(\mathbf{K}_{*,\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{U}, \mathbf{K}_{*,*} - \mathbf{Q}_{*,*}). \tag{2.15b}$$

Notice that the training conditional and test conditional are under different distributions, the DTC method does not refer to a strict Gaussian Process. Given the conditioned distributions, the joint prior of the DTC method is

$$q_{DTC}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{Q}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I} & \mathbf{Q}_{\mathbf{X},*} \\ \mathbf{Q}_{*,\mathbf{X}} & \mathbf{K}_{*,*} \end{bmatrix}\right). \tag{2.16}$$

2.2.2 FITC Approximation

Another variation of the prior approximation is the FITC approximation [28]. The difference between FITC and the previous two prior approximations is that, instead of imposing a deterministic relationship, FITC assumes that the training conditionals have a distribution that is independent of each other:

$$p_{FITC}(\mathbf{f}|\mathbf{U}) = \prod_{1}^{n} p(f_i|\mathbf{U}) = \mathcal{N}(\mathbf{K_{X,U}}\mathbf{K_{U,U}^{-1}}\mathbf{U}, \text{diag}[\mathbf{K_{X,X}} - \mathbf{Q_{X,X}}]), \tag{2.17a}$$

$$p_{FITC}(f_*|\mathbf{U}) = p_{DTC}(f_*|\mathbf{U}) = p(f_*|\mathbf{U}), \tag{2.17b}$$

where diag[$*$] means remaining the original matrix's diagonal component while setting all other elements to 0. We consider FITC with only one test input since the original authors didn't mention whether a diagonal correction is also implemented for the test inputs. We can also interpret FITC as an improved version of DTC by adding a diagonal correction term $\text{diag}[\mathbf{K_{X,X}} - \mathbf{Q_{X,X}}]$. If we extend this independent assumption to the test conditionals, we will arrive at the following joint prior:

$$q_{FITC}(\mathbf{f}, \mathbf{f}_*) = \mathcal{N}\left(0, \begin{bmatrix} \mathbf{Q_{X,X}} + \mathbf{\Lambda} & \mathbf{Q_{X,*}} \\ \mathbf{Q_{*,X}} & \mathbf{K_{*,*}} \end{bmatrix}\right), \tag{2.18}$$

where $\mathbf{\Lambda} = \sigma_n^2\mathbf{I} + \text{diag}[\mathbf{K_{X,X}} - \mathbf{Q_{X,X}}]$ is noise plus the diagonal correction. Due to the introduced diagonal correction, the training prior finally considers some information from the original training data, thus a much richer posterior covariance is acquired. With respect to calculations, $\mathbf{\Lambda}$ is a diagonal matrix, thus preserving the benefits from the matrix inversion and determinant lemmas.

The approximations for FITC are given as follows:

$$\begin{aligned}
&\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{k}_i^T\mathbf{K_{U,U}^{-1}}\mathbf{k}_j + \boldsymbol{\delta}_{ij}\left[k(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{k}_i^T\mathbf{K_{U,U}^{-1}}\mathbf{k}_j\right], \\
&\textbf{posterior mean} : \bar{f}_* \approx \mathbf{k}_*^T\mathbf{A}^{-1}\mathbf{K_{U,X}}\mathbf{\Lambda}^{-1}\mathbf{y}, \\
&\textbf{posterior covariance} : cov(f_*) \approx k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T\mathbf{K_{U,U}^{-1}}\mathbf{k}_* + \mathbf{k}_*^T\mathbf{A}^{-1}\mathbf{k}_*, \\
&\textbf{NMLL} : \log p(\mathbf{y}) \approx \frac{1}{2}[\mathbf{y}^T\mathbf{L}^{-1}\mathbf{y} + \log\det(\mathbf{K_{U,U}^{-1}}) + \log\det(\mathbf{A}) + \log\det(\mathbf{\Lambda}) + n\log 2\pi],
\end{aligned} \tag{2.19}$$

where in the kernel function, $\boldsymbol{\delta}$ is the Kronecker's delta, $\mathbf{L}^{-1} = \mathbf{\Lambda}^{-1} - \mathbf{\Lambda}^{-1}\mathbf{K_{X,U}}\mathbf{A}^{-1}\mathbf{K_{U,X}}\mathbf{\Lambda}^{-1}$ and $\mathbf{A} = \mathbf{K_{U,U}} + \mathbf{K_{U,X}}\mathbf{\Lambda}^{-1}\mathbf{K_{X,U}}$ are slightly different than previous forms due to the change in the diagonal. However, the time complexity of FITC is still $\mathcal{O}(m^2n)$, since $\mathbf{K_{U,U}}$ and $\mathbf{K_{x,U}}$ are the same as SoR.

2.2.3 Selection of Inducing Variables

In all the prior approximation techniques, the choice of inducing variables plays a crucial part in the accuracy of the approximations, due to the information they contain. The first option for the choice of inducing variables $\mathbf{U}$ is within the training set or test set

[34]. That is selecting points from points being trained and being evaluated. However, these are discrete sets, so using optimization techniques to learn the inducing variables is not feasible. Another method is to relax the constraint and assume inducing variables could be chosen from any point in the input space. In other words, the choices could be continuous, meaning they could be learned through optimizing the log-likelihood:

$$\log q(\mathbf{y}|\mathbf{U}) = -\frac{1}{2}\mathbf{y}^T(\mathbf{Q}_{\mathbf{X},\mathbf{X}} + \mathbf{\Lambda})^{-1}\mathbf{y} - \frac{1}{2}\log\det(\mathbf{Q}_{\mathbf{X},\mathbf{X}} + \mathbf{\Lambda}) - \frac{n}{2}\log 2\pi. \tag{2.20}$$

Different prior optimization methods differ in the calculation of $\mathbf{\Lambda}$: $\mathbf{\Lambda}_{SoR} = \mathbf{\Lambda}_{DTC} = \sigma_n^2 I$, $\mathbf{\Lambda}_{FITC} = \sigma_n^2\mathbf{I} - \text{diag}[\mathbf{Q}_{\mathbf{X},\mathbf{X}} - \mathbf{K}_{\mathbf{X},\mathbf{X}}]$. However, for simplicity, the inducing variables chosen for SoR and FITC in this thesis are selected randomly from the training inputs.

2.2.4 Summary of Prior Approximations

Prior approximations introduce a set of inducing variables $\mathbf{U}$ that describes the relationship between training and testing inputs via different probabilistic assumptions. Specifically, the SoR approximation assumes a deterministic relationship between both the training inputs and the inducing variables, and the testing inputs and the inducing variables. As a consequence, the covariance of the posterior is too small which represents over-confident predictions. The DTC alleviates the restrictions by only imposing the deterministic relationship between the training inputs and the inducing variables. The FITC further improves by assuming that the training conditionals are independent of each other by introducing a diagonal correction term. This improvement of FITC renders the approximated kernel matrix full-rank, which makes it the most popular prior approximation in application. The time complexity of all prior approximations is $\mathcal{O}(m^2 n)$, where m is the number of inducing variables and n is the number of training points.

Regarding the choice of inducing variables, either choosing from training inputs, testing inputs, or learning by optimization has support in the literature. For simplicity, we follow the practice of choosing from training inputs in our thesis.

2.3 Spectral Approximations

Spectral approximations are a family of methods that approximate a stationary kernel function. A kernel function is *stationary* if the terms only appeared in the form of their distance:

$$k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i - \mathbf{x}_j) = k(\mathbf{r}). \tag{2.21}$$

An improvement from the prior approximations is that spectral approximations can approximate any stationary kernel functions without having an initial assumption of the kernel type as long as it is stationary.

For any stationary kernel function, a *spectral density* $S(\boldsymbol{\omega})$ expresses how the power is distributed over the frequency domain. The frequency vector $\boldsymbol{\omega}$ has the same dimensions as the input $\mathbf{x}$. According to the Wiener-Khintchine theorem [36], the spectral density and the stationary kernel function form a Fourier transform pair:

$$k(\mathbf{r}) = \int_{\mathbb{R}^d} S(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T \mathbf{r}} \mathrm{d}\boldsymbol{\omega}, \tag{2.22a}$$

$$S(\boldsymbol{\omega}) = \int_{\mathbb{R}^d} k(\mathbf{r}) e^{-j\boldsymbol{\omega}^T \mathbf{r}} \mathrm{d}\mathbf{r}. \tag{2.22b}$$

From approximating the spectral density $S(\boldsymbol{\omega})$, typically by using a sum of a finite number of basis functions $\boldsymbol{\phi}_k(\mathbf{x})$ either trigonometric or not, one can acquire the following approximated form of the original kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sum_k a_k \boldsymbol{\phi}_k(\mathbf{x}_i)^T \boldsymbol{\phi}_k(\mathbf{x}_j). \tag{2.23}$$

In general, the spectral approximations give a decomposition of the original kernel function into an infinite number of basis functions. With different requirements for precision and time, the approximation truncates the terms to different finite numbers.

The posterior mean and covariance are calculated differently according to the actual representation of the approximation, which will be given in the following subsections.

2.3.1 SSGP Approximation

The SSGP approximates the spectral density from different frequency samples. We first assume a representation of a stationary kernel function as a weighted sum of trigonometric functions:

$$f(\mathbf{x}) \approx \sum_{r=1}^{m} a_r \cos \boldsymbol{\omega}_r^T \mathbf{x} + b_r \sin \boldsymbol{\omega}_r^T \mathbf{x}. \tag{2.24}$$

$\boldsymbol{\omega}_r$ is a vector parameter that specifies the spectral frequencies and is considered deterministic. a_r and b_r are the amplitudes, and are treated as independent random variables with prior:

$$a_r \sim \mathcal{N}(0, \frac{\sigma_0^2}{m}), \tag{2.25a}$$

$$b_r \sim \mathcal{N}(0, \frac{\sigma_0^2}{m}). \tag{2.25b}$$

Some may argue that a sum of periodic functions is still a periodic function. However, just like the Fourier expansion, by including different frequency terms, the overall period could be much longer than the input spans, thus allowing approximation within a single period. Also, the original authors claim that this decomposition is feasible in practice for general stationary functions.

Under this assumption of variables, the original function is Gaussian with zero mean and covariance function:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}(f(\mathbf{x}_i) f(\mathbf{x}_j)) = \frac{\sigma_0^2}{m} \sum_{r=1}^{m} \cos \boldsymbol{\omega}_r^T (\mathbf{x}_i - \mathbf{x}_j). \tag{2.26}$$

2.26 can also be written as

$$k(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sigma_0^2}{m} \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j), \tag{2.27}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is the vector collecting the $2m$ basis functions in pairs evaluated at $\mathbf{x}$:

$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \cos \boldsymbol{\omega}_1^T \mathbf{x} & \sin \boldsymbol{\omega}_1^T \mathbf{x} & ... \cos \boldsymbol{\omega}_m^T \mathbf{x} & \sin \boldsymbol{\omega}_m^T \mathbf{x} \end{bmatrix}^T. \tag{2.28}$$

Notice that from 2.26 we can see the kernel is stationary, i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{r})$. Moreover, the signal power is no longer parameterized by a_r and b_r which are $2m$ parameters in total, but parameterized by one parameter σ_0^2 instead. We now have a specific trigonometric form of a kernel, but the meaning of the signal frequencies is still unclear.

We can also approach the kernel function from the spectral view. Bochner's theorem states that $k(\mathbf{r})$ can be represented by the Fourier transform of a positive finite measure [30]. In particular:

$$S(\boldsymbol{\omega}) = \sigma_0^2 p_S(\boldsymbol{\omega}), \tag{2.29}$$

where $p_S(\boldsymbol{\omega})$ is a the probability density of $\boldsymbol{\omega}$. Substitute 2.29 into 2.22, we can write the expression of the kernel function as an expectation:

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= \int_{\mathbb{R}^D} S(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T(\mathbf{x}_i - \mathbf{x}_j)} \mathrm{d}\boldsymbol{\omega} = \sigma_0^2 \int_{\mathbb{R}^D} S(\boldsymbol{\omega}) e^{j\boldsymbol{\omega}^T \mathbf{x}_i} \left(e^{j\boldsymbol{\omega}^T \mathbf{x}_j}\right)^* p_S(\boldsymbol{\omega}) \mathrm{d}\boldsymbol{\omega} \\ &= \sigma_0^2 \mathbb{E}\left[e^{j\boldsymbol{\omega}^T \mathbf{x}_i} \left(e^{j\boldsymbol{\omega}^T \mathbf{x}_j}\right)^*\right], \end{aligned} \tag{2.30}$$

where $(\cdot)^*$ means the complex conjugate of the value inside. This expectation is approximated by Monte Carlo by taking the average of a certain amount of frequency vector samples from $p_S(\boldsymbol{\omega})$, referred to as *spectral points*. Because the spectral density is symmetric around zero, *spectral points* are taken as pairs $[\boldsymbol{\omega}_r, -\boldsymbol{\omega}_r]$. Suppose we take m pairs of such, then the expectation can be approximated with:

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &\simeq \frac{\sigma_0^2}{2m} \sum_{r=1}^{m} \left[e^{j\boldsymbol{\omega}_r^T \mathbf{x}_i} \left(e^{j\boldsymbol{\omega}_r^T \mathbf{x}_j}\right)^* + \left(e^{j\boldsymbol{\omega}_r^T \mathbf{x}_i}\right)^* e^{j\boldsymbol{\omega}_r^T \mathbf{x}_j}\right] \\ &= \frac{\sigma_0^2}{m} \sum_{r=1}^{m} \cos \boldsymbol{\omega}_r^T (\mathbf{x}_i - \mathbf{x}_j). \end{aligned} \tag{2.31}$$

Notice that this final approximation matches the expression of 2.26, which means that the frequencies can be seen as Monte Carlo samples of $p_S(\boldsymbol{\omega})$. The original spectrum $S(\boldsymbol{\omega})$ is thus sparsified and replaced by using a set of Dirac Deltas with amplitude σ_0^2.

In practice, the spectral points are also learned along with other hyperparameters via the log-likelihood, thus there is no need to compute the distribution $p_S(\boldsymbol{\omega})$. However, this means that at least m hyperparameters should be optimized in the learning process.

We give the forms of the important functions throughout the GP process as follows:

$$
\begin{aligned}
&\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) \approx \sigma_n^2 b \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j), \\
&\textbf{posterior mean} : \bar{f}_* \approx b\boldsymbol{\phi}(\mathbf{x}_*)^T \mathbf{A}^{-1} \boldsymbol{\Phi}_{\mathbf{X}} \mathbf{y}, \\
&\textbf{posterior covariance} : cov(f_*) \approx \sigma_n^2 \boldsymbol{\phi}(\mathbf{x}_*)^T \mathbf{A}^{-1} \boldsymbol{\phi}(\mathbf{x}_*), \\
&\textbf{NMLL} : \log p(\mathbf{y}) \approx \frac{1}{2}\left[\mathbf{y}^T \mathbf{L}^{-1} \mathbf{y} + \log\det(\mathbf{A}) + n \log 2\pi\sigma_n^2\right],
\end{aligned}
\tag{2.32}
$$

where $b = \frac{\sigma_0^2}{m\sigma_n^2}$, $\boldsymbol{\Phi}_{\mathbf{X}} = [\boldsymbol{\phi}(\mathbf{x}_1)\ \boldsymbol{\phi}(\mathbf{x}_2)\ ...\boldsymbol{\phi}(\mathbf{x}_n)]$ is a $2m \times n$ matrix, $\mathbf{A} = b\boldsymbol{\Phi}_{\mathbf{X}}\boldsymbol{\Phi}_{\mathbf{X}}^T + \mathbf{I}$, $\mathbf{L}^{-1} = \sigma_n^{-2}\mathbf{I} - \sigma_n^{-2} b \boldsymbol{\Phi}_{\mathbf{X}}^T \mathbf{A}^{-1} \boldsymbol{\Phi}_{\mathbf{X}}$. We can assume that $2m < n$, thus the computations are facilitated by using the Cholesky decomposition to $\mathcal{O}(m^2 n)$ time complexity.

2.3.2 Hilbert Space Method Approximation

The Hilbert space method [31] is also based on an approximation of the covariance function. The covariance functions in their study are further assumed to be *isotropic*, which states that the difference of inputs occurs only in Euclidean norm, i.e., only with terms $\|\mathbf{r}\| = \|\mathbf{x}_i - \mathbf{x}_j\|$, where $\mathbf{x}_i$ and $\mathbf{x}_j$ are two arbitrary inputs.

2.22 for isotropic kernels can be written as:

$$
k(\|\mathbf{r}\|) = \int S(\|\boldsymbol{\omega}\|) e^{j\|\boldsymbol{\omega}\|^T \|\mathbf{r}\|} \mathrm{d}\|\boldsymbol{\omega}\|, \tag{2.33a}
$$

$$
S(\|\boldsymbol{\omega}\|) = \int k(\|\mathbf{r}\|) e^{-j\|\boldsymbol{\omega}\|^T \|\mathbf{r}\|} \mathrm{d}\|\mathbf{r}\|. \tag{2.33b}
$$

We define a covariance operator $\mathcal{K}$ associated to each covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$:

$$
\mathcal{K}\boldsymbol{\phi} = \int k(\cdot, \mathbf{x}_j) \boldsymbol{\phi}(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j. \tag{2.34}
$$

Because of the assumed stationary nature of covariance functions, the operator is translation invariant, and thus we can use the Fourier representation as the transfer function. Furthermore, the transfer function is the spectral density.

Recall that we assume the covariance functions to be isotropic, and further assume that the spectral density can be written in a polynomial expansion (e.g., the Taylor series):

$$
S(\|\omega\|) = a_0 + a_1\|\omega\|^2 + a_2(\|\omega\|^2)^2 + a_3(\|\omega\|^2)^3 + \ldots \tag{2.35}
$$

For a regular function f, we have the following transform:

$$
\mathcal{F}[\boldsymbol{\nabla}^2 f](\omega) = -\|\omega\|^2 \mathcal{F}[f](\omega), \tag{2.36}
$$

where $\mathcal{F}[\cdot]$ denotes the Fourier transform of a function, and $\boldsymbol{\nabla}^2$ denotes the Laplace operator. By implementing this formula in the inverse Fourier transformation of 2.35, the covariance operator can be written as a series of Laplace operators:

$$
\mathcal{K} = a_0 + a_1(-\boldsymbol{\nabla}^2) + a_2(-\boldsymbol{\nabla}^2)^2 + a_3(-\boldsymbol{\nabla}^2)^3 + \ldots \tag{2.37}
$$

The Hilbert space approximation comes by considering the eigenvalue problem of the Laplace operators with Dirichlet boundary conditions. Because the Laplacian operator is a positive definite Hermitian operator, the set of eigenfunctions $\boldsymbol{\phi}_k(\cdot)$ is orthonormal with respect to the inner product. Also, the eigenvalues λ_k are real-positive numbers. Thus we can write the Laplacian operator as:

$$-\boldsymbol{\nabla}^2 f(\mathbf{x}) = \int l(\mathbf{x}_i, \mathbf{x}_j) f(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j, \tag{2.38}$$

where

$$l(\mathbf{x}_i, \mathbf{x}_j) = \sum_k \lambda_k \boldsymbol{\phi}_k(\mathbf{x}_i) \boldsymbol{\phi}_k(\mathbf{x}_j). \tag{2.39}$$

Due to the orthonormality of the eigenfunctions, the arbitrary powers of the Laplacian operators can also be represented with similar expressions:

$$(-\boldsymbol{\nabla}^2)^n f(\mathbf{x}) = \int l^n(\mathbf{x}_i, \mathbf{x}_j) f(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j, \tag{2.40}$$

where the superscript n of $l(\mathbf{x}_i, \mathbf{x}_j)$ is the shorthand notation of:

$$l^n(\mathbf{x}_i, \mathbf{x}_j) = \sum_k \lambda_k^n \boldsymbol{\phi}_k(\mathbf{x}_i) \boldsymbol{\phi}_k(\mathbf{x}_j). \tag{2.41}$$

Thus we have the covariance operated at $f(\mathbf{x}_i)$ as:

$$\begin{aligned} \mathcal{K} f(\mathbf{x}_i) &= [a_0 + a_1(-\boldsymbol{\nabla}^2) + a_2(-\boldsymbol{\nabla}^2)^2 + ...] f(\mathbf{x}_i) \\ &= \int [a_0 + a_1 l(\mathbf{x}_i, \mathbf{x}_j) + a_2 l^2(\mathbf{x}_i, \mathbf{x}_j) + ...] f(\mathbf{x}_j) \mathrm{d}\mathbf{x}_j. \end{aligned} \tag{2.42}$$

Comparing this result with 2.34 we have the approximation of the kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) \approx \sum_k [a_0 + a_1 \lambda_k + a_2 \lambda_k^2 + ...] \boldsymbol{\phi}_k(\mathbf{x}_i) \boldsymbol{\phi}_k(\mathbf{x}_j). \tag{2.43}$$

Finally, substituting $\|\omega\|^2 = \lambda_k$ in 2.35 and plugging the result into 2.43, we arrive at the final approximation of the kernel function:

$$k(\mathbf{x}_i, \mathbf{x}_j) \approx \sum_k S(\sqrt{\lambda_k}) \boldsymbol{\phi}_k(\mathbf{x}_i) \boldsymbol{\phi}_k(\mathbf{x}_j), \tag{2.44}$$

where $S(\cdot)$ is the spectral density of the kernel function, λ_k is the k-th eigenvalue and $\boldsymbol{\phi}_k(\cdot)$ the k-th eigenfunction of the Laplace operator in a given domain.

Because the eigenvalues of the Laplace operator increase with k and the spectral density decays to zero quickly along frequencies, a limited amount of k can produce good approximations.

To translate the approximation of the kernel function into the approximation of the Kernel matrix, we project the training outputs to a truncated set of m basis functions of the Laplacian as given in 2.44:

$$f(\mathbf{x}) = \sum_{k=1}^{m} f_k \boldsymbol{\phi}_k(\mathbf{x}), \tag{2.45}$$

where $f_k \sim \mathcal{N}(0, S(\sqrt{\lambda_k}))$. We can then approximate the eigendecomposition as $\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix with elements $S(\sqrt{\lambda_k}), k = 1, \ldots, m$. The corresponding eigenvectors are specified by the eigenvectors of the Laplacian operator such that $\boldsymbol{\Phi}_{ik} = \boldsymbol{\phi}_k(\mathbf{x}_i)$. Now, we can write out the posterior distribution with the matrix inversion lemma, and also the log-likelihood:

$$\begin{aligned}
&\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) \approx \sum_k S(\sqrt{\lambda_k}) \boldsymbol{\phi}_k(\mathbf{x}_i) \boldsymbol{\phi}_k(\mathbf{x}_j), \\
&\textbf{posterior mean} : \bar{f}_* \approx \sigma_n^{-2} \boldsymbol{\phi}_*^T \mathbf{A}^{-1} \boldsymbol{\Phi}^T \mathbf{y}, \\
&\textbf{posterior covariance} : cov(f_*) \approx \boldsymbol{\phi}_*^T \mathbf{A}^{-1} \boldsymbol{\phi}_*, \\
&\textbf{NMLL} : \log p(\mathbf{y}) \approx \frac{1}{2} \left[\mathbf{y}^T \mathbf{L}^{-1} \mathbf{y} + \log\det(\mathbf{A}) + \sum_k \log S(\sqrt{\lambda_k}) + n \log 2\pi\sigma_n^2 \right],
\end{aligned} \tag{2.46}$$

where $\boldsymbol{\phi}_* = [\boldsymbol{\phi}_1(\mathbf{x}_*)\ \boldsymbol{\phi}_2(\mathbf{x}_*)\ \ldots\ \boldsymbol{\phi}_m(\mathbf{x}_*)]^T$, $\mathbf{A} = \sigma_n^{-2}\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \boldsymbol{\Lambda}^{-1}$, $\mathbf{L}^{-1} = \sigma_n^{-2}\mathbf{I} - \sigma_n^{-4}\boldsymbol{\Phi}\mathbf{A}^{-1}\boldsymbol{\Phi}^T$. The reason for the reduced time is that the eigenfunctions used in the approximation are independent of the hyperparameters of the kernel matrix. Thus, we only need to calculate $\boldsymbol{\Phi}$ and $\boldsymbol{\Phi}^T\boldsymbol{\Phi}$ once, which is everything required for both posterior calculation and hyperparameter optimization, making the whole system's time complexity to asymptotically be $\mathcal{O}(m^2 n)$.

2.3.3 Summary of Spectral Approximations

Spectral approximations typically facilitate the calculation by using a finite sum of basis function products to represent the precise spectral density $S(\omega)$. In SSGP, the basis functions are taken as sinusoids with the amplitude, length scales, and frequencies trained from optimization. Though having the advantage of representing any stationary kernel without initially assuming the kernel function, it suffers from learning too many hyperparameters (i.e., the frequencies). The Hilbert space method approximation takes the basis functions as eigenfunctions of the Laplace operator in a given domain. The speed-up comes from the fact that the eigenfunctions are independent of the choice of the covariance hyperparameters, thus it needs only be calculated one time. The time complexities for both methods are also $\mathcal{O}(m^2 n)$.

2.4 Structural Approximations

Structural approximation is a family of methods that exploits the relationship between the kernel matrix formed by a set of equispace grid points $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ and the kernel matrix of data points $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ that the later could be interpolated from the former. These grid points could be viewed as a special set of inducing variables in the context of the prior

approximation, thus we use the same notation $\mathbf{U}$ in this section to refer to the grid points.

Structural approximations, in general, can have a dramatic boost in calculation speed due to the matrix properties of its decomposition of the original kernel matrix:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}\mathbf{W}^T, \tag{2.47}$$

where $\mathbf{W}$ is a spare matrix, $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ is the grid kernel matrix that may possess special structures, which all accelerate the matrix-vector multiplication (MVM) operation. Thus, structural methods typically implement the linear CG algorithm and the Lanczos method that only requires MVM of matrices to calculate the inverse and log-determinant terms. Though they are iterative methods, the number of iterations needed for linear CG and Lanczos method to reach a certain tolerance is often considered a constant that is much smaller than n [6].

The special matrix structures of the grid kernel matrix could be Kronecker or Toeplitz structures. For multidimensional grids, if the kernel is a product kernel, i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = \prod_{d=1}^{D} k(\mathbf{x}_i^d, \mathbf{x}_j^d$, where $\mathbf{x}_i^d$ means the input value on the d-th dimension, then we have the kernel matrix as a Kronecker product: $\mathbf{K} = \mathbf{K}_1 \otimes \mathbf{K}_2 \otimes \cdots \otimes \mathbf{K}_D$, where $\mathbf{K}_d$ means the kernel matrix calculated by the grid on the d-th dimension [37]. The number of grid points will be $m = \prod_{d=1}^{D} m_d$, where m_d is the number of grid points on the d-th dimension, and D is the total number of dimensions. The Kronecker structure of the grid kernel matrix has several benefits:

- Fast eigenvalue calculation: The eigenvalues of a Kronecker product could be calculated as the outer product of the eigenvalues of each matrix factor. Thus we only need to calculate the eigenvalues of all $\mathbf{K}_d$. The total time complexity is thus $\sum_{d=1}^{D} \mathcal{O}(m_d^3)$, much smaller than directly calculating the eigendecomposition of $\mathbf{K}_{U,U}$, $\mathcal{O}((\prod_{d=1}^{D} m_d)^3)$. This property is implemented in one of the approximations of $\log\det(\mathbf{K}_\mathbf{n})$.

- Fast MVM calculation: The MVM operation of a Kronecker product could be calculated using a Kronecker product MVM algorithm, which is an $\mathcal{O}(Dm^{1+\frac{1}{D}})$ operation instead of $\mathcal{O}(m^2)$ [38]. This is implemented in lines 2,7, and 9 in the linear CG algorithm 2 that requires calculating $\mathbf{Ax}$ or $\mathbf{Ap}$, where $\mathbf{A}$ takes the value of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$.

The kernel could also possess a Toeplitz structure. This requires the kernel function to be stationary, i.e., $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i - \mathbf{x}_j) = k(\mathbf{r})$, and the grids to be equispaced. The Toeplitz structure can be exploited to perform fast MVM using the fast Fourier transform (FFT) algorithm [37]. This can allow linear CG to calculate $(\mathbf{K}_\mathbf{n})^{-1}\mathbf{y}$ within $\mathcal{O}(m \log m)$ time. However, the original authors state in their codes that only with grid numbers on one dimension greater than 500 can we observe a speed advantage using the FFT algorithm than using brute force.

In the following sections, we will first present a founding paper for structural approximations, and then give two alternations to the original method.

2.4.1 KISSGP Approximation

The base framework of all structural approximation methods, the SKI, was first introduced in KISSGP [33]. This method gains its intuition from the prior approximations to form its interpolation scheme but can stand alone from the prior assumptions.

First, consider the form of the approximation of the training kernel matrix of the SoR method:

$$\mathbf{K}_{SoR} = \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}. \tag{2.48}$$

As mentioned in Section 2.1, the major time consumption comes from $\mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}$ and the inverse $\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}$. The main idea of KISSGP is to interpolate $\mathbf{K}_{\mathbf{X},\mathbf{U}}$ via $\mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}$, where $\mathbf{W} \in \mathbb{R}^{n\times m}$ is the interpolation matrix. This interpolation is feasible if there exists a set of weights $\{w_{i,1}, w_{i,2}, \ldots, w_{i,m}\}$ corresponding to all grid points $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$ for every input $\mathbf{x}_i$ that can achieve:

$$k(\mathbf{x}_i, \mathbf{u}) \approx \sum_{j=1}^{m} w_{i,j} k(\mathbf{u}_j, \mathbf{u}). \tag{2.49}$$

This approximation is assumed to be valid taking the second input of the kernel function $\mathbf{u}$ as any possible grid point. The error of this interpolation is tolerable if the kernel function is smooth, e.g., RBF kernel. It is also determined by the interpolation method and the number of grid points. This will be analyzed in Chapter 3. If we collect all the weights in vectors $\mathbf{w}_i$ for every training input $\mathbf{x}_i$ and stack them together horizontally, we arrive at the following approximation:

$$\mathbf{K}_{\mathbf{X},\mathbf{U}} \approx \mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}, \tag{2.50}$$

where $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n]^T$ is the interpolation matrix. Every entry of $\mathbf{K}_{\mathbf{X},\mathbf{U}}$ is thus approximated by the interpolation in 2.49. $\mathbf{W}$ could be very sparse depending on the interpolation method. If we use linear interpolation per dimension, there will only be 2^D non-zero entries per row in $\mathbf{W}$, where D is the number of individual kernels of the product kernel. In the original KISSGP settings, the default interpolation method is the cubic convolution interpolation [39], which is 4^D non-zero entries per row. By substituting 2.50 into the expression in 2.48, we acquire the following approximation:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{K}_{SKI} = \mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}\mathbf{W}^T. \tag{2.51}$$

In other words, every entry in the training kernel matrix is interpolated via the grid kernel matrix. The author refers to this kernel interpolation as SKI. Though this approximation comes from the form of the SoR kernel, one could directly consider it as a series of interpolations. First is the interpolation 2.50. Then is the interpolation:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{W}^T, \tag{2.52}$$

which also comes from the interpolation assumption 2.49 however further generalizing the interpolation to any data point $\mathbf{x}$ as the second input of the kernel function. 2.50 and 2.52 together can form the SKI kernel 2.51 without the intermediate results.

From the SKI kernel we acquire the corresponding forms of the calculation terms in the SKI framework:

$$\begin{aligned}
&\textbf{kernel function}: k(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{w}_{\mathbf{x}_i}^T \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{w}_{\mathbf{x}_j},\\
&\textbf{posterior mean}: \bar{f}_* \approx \mathbf{w}_{\mathbf{x}_*}^T \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{W}^T (\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y},\\
&\textbf{posterior covariance}: cov(f_*) \approx \mathbf{w}_{\mathbf{x}_*}^T \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{w}_{\mathbf{x}_*} - \mathbf{w}_{\mathbf{x}_*}^T \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{W}^T (\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{W} \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{w}_{\mathbf{x}_*},\\
&\textbf{NMLL}: \log p(\mathbf{y}) \approx \frac{1}{2}[\mathbf{y}^T (\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} + \log\det(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I}) + n \log 2\pi],
\end{aligned} \tag{2.53}$$

where $\mathbf{w}_{\mathbf{x}}$ is the corresponding weight vector for input $\mathbf{x}$.

The superiority of KISSGP in time complexity comes from it's calculations of both $(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}$ and $\log\det(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})$.

The first term is calculated via the linear CG algorithm. In the algorithm, the most time-consuming step is calculating $(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})\mathbf{v}$, where $(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})$ takes the place of $\mathbf{A}$ and $\mathbf{v}$ could be $\mathbf{y}$ or the descending direction vector $\mathbf{p}$ (see algorithm 2). The MVM could be rewritten as:

$$(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})\mathbf{v} = \mathbf{W}(\mathbf{K}_{\mathbf{U},\mathbf{U}}(\mathbf{W}^\mathbf{T}\mathbf{v})) + \sigma_n^2 \mathbf{v}. \tag{2.54}$$

$\mathbf{W}^\mathbf{T}\mathbf{v}$ could be quickly calculated since $\mathbf{W}$ is a sparse matrix. $\mathbf{K}_{\mathbf{U},\mathbf{U}}(\mathbf{W}^\mathbf{T}\mathbf{v})$ could then possibly be facilitated by the presence of Kronecker structure or Toeplitz structure of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$. The final multiplication with $\mathbf{W}$ could also leverage the sparsity of $\mathbf{W}$. The time complexity of the grid kernel matrix MVM is thus $\mathcal{O}(mvm(\mathbf{W})) + \mathcal{O}(mvm(\mathbf{K}_{\mathbf{U},\mathbf{U}})) = \mathcal{O}(n + m^2)$ in general, where $\mathcal{O}(mvm(\cdot))$ represents the time complexity of calculating the MVM [32]. When exploiting Kronecker or Toeplitz structure, it could be $\mathcal{O}(n + Dm^{\frac{1}{D}})$ or $\mathcal{O}(n + m \log m)$, respectively. As mentioned in the explanations of algorithm 2, the iteration number of linear CG is usually considered a constant smaller than n, thus the overall time complexity of calculating $(\mathbf{K}_{SKI} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}$ is $\mathcal{O}(n + m^2)$.

The log-determinant term could be approximated by two methods. The first method, as introduced in the original KISSGP paper is to approximate the log-determinant using the eigenvalues of the grid kernel matrix:

$$\log\det(\mathbf{K}_M + \sigma_n^2 \mathbf{I}) \approx \sum_{i=1}^{n} \log\left(\frac{n}{m}\lambda_i + \sigma_n^2\right), \tag{2.55}$$

where λ_i are the n largest eigenvalues of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$. 0 is filled in if $m < n$. As long as the training inputs are bounded by the grid points, this approximation is asymptotically consistent as n increases [40]. The eigenvalues are acquired by performing eigendecomposition on $\mathbf{K}_{\mathbf{U},\mathbf{U}}$. Kronecker algebra could be utilized to quickly calculate the eigenvalues of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ by performing eigendecomposition in each dimension separately. The time complexity of performing eigendecomposition on one dimension is $\mathcal{O}(m_d^3)$, where m_d is the number of grid points on that dimension.

Another method is the Lanczos method for log-determinant estimation (see algorithm 1) which has similar benefits as the linear CG.

Below is an illustration of the training framework of KISSGP:

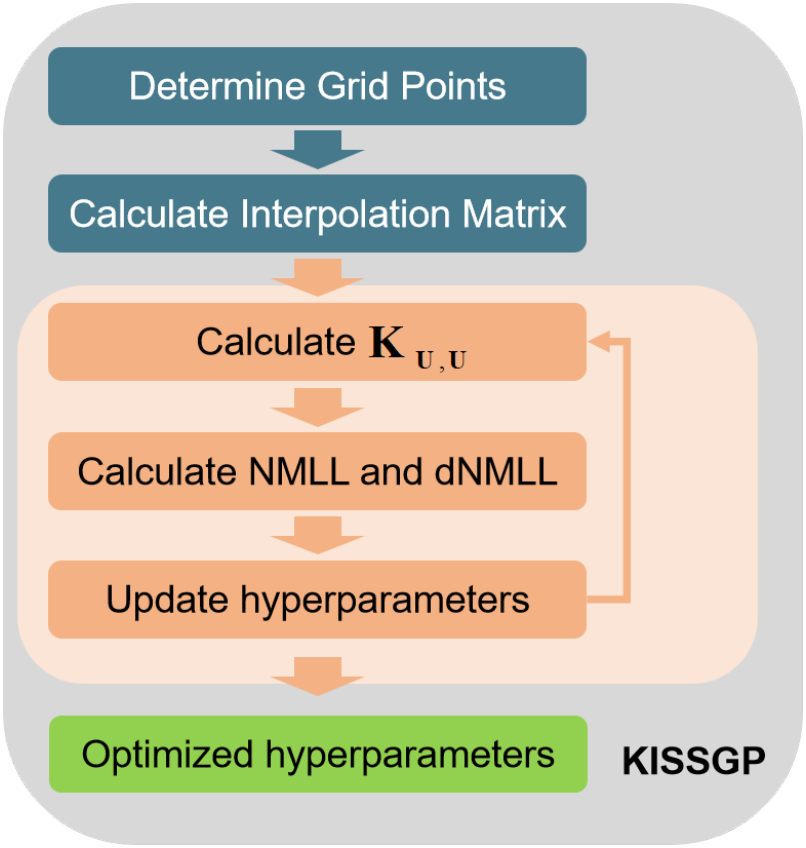


Figure 2.1: Training framework of KISSGP. The blue blocks represent the initialization phase, where the grid points and interpolation matrix $\mathbf{W}$ are only calculated once. The orange blocks are the iterative steps in the algorithm, where the goal is to reach the lowest NMLL. The stopping criteria are set according to the nonlinear CG in algorithm 3, where the optimized hyperparameters are acquired

It is worth noticing that the calculation of the interpolation matrix $\mathbf{W}$ is outside the optimization loop since the grid points are fixed throughout the learning process in KISSGP.

Further implementation details of KISSGP are explained in Section 3.4.

The crucial part for determining the computational cost of SKI is the number of grid points. In the original KISSGP, this number is predetermined by merely guessing and may grow exponentially as the number of dimensions increases (100 grid points to represent one dimension makes 10000 grid points in total for a 2-D grid). In the following two approximations, the first one provides an alternative formulation of the SKI problem to facilitate the calculations by improving the framework. The second approximation directly tackles the problem of dimensions and introduces sparse grids to mitigate the impact of increasing dimensions.

2.4.2 GSGP Approximation

A more recent method that builds on the SKI further improves the efficiency by reframing the SKI to a Bayesian linear regression problem. We call this method GSGP, due to its major design below.

The idea starts by giving the SKI method a Bayesian linear regression formulation. Consider $G = \mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_m \subseteq \mathbb{R}^d$ as a set of d-dimensional grid points. Generate samples $\mathbf{z}$ from the grid covariance matrix according to a traditional GP formulation:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_G), \tag{2.56}$$

where $\mathbf{K}_G$ is the kernel matrix corresponding to G. Then construct a weight vector $\mathbf{w}_\mathbf{x}$ according to a basis function of any interpolation scheme that maps $\mathbf{x} \in \mathbb{R}^d$ to $\mathbf{w}_\mathbf{x} \in \mathbb{R}^m$. Multiply the evaluated points of the original GP and the weights to get the output of the GSGP:

$$f(\mathbf{x}) = \mathbf{w}_\mathbf{x}^T \mathbf{z}. \tag{2.57}$$

Notice that a GSGP is a classical Bayesian linear regression model. The generated function $f(\mathbf{x})$ can be interpreted as an interpolation of the grid values $\mathbf{z}$.

Furthermore, $f(\mathbf{x})$ is a GP with covariance function $cov(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{w}_{\mathbf{x}_i}^T \mathbf{K}_G \mathbf{w}_{\mathbf{x}_j}$. Because, for any finite number of inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, the function values $f(\mathbf{x}_i) = \mathbf{w}_{\mathbf{x}_i}^T \mathbf{z}$ are all linear transformations of the same Gaussian variable $\mathbf{z}$. Therefore $f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_n)$ are jointly Gaussian, $f(\mathbf{x})$ is a Gaussian process. The mean is 0 and the covariance function is given as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)] = \mathbf{w}_{\mathbf{x}_i}^T \mathbb{E}(\mathbf{z}\mathbf{z}^T) \mathbf{w}_{\mathbf{x}_j} = \mathbf{w}_{\mathbf{x}_i}^T \mathbf{K}_G \mathbf{w}_{\mathbf{x}_j}. \tag{2.58}$$

Comparing the forms of 2.58 and the kernel function in 2.53, one can find that they have the same form. In other words, the exact covariance function of the GSGP is the same as the approximation in the SKI framework.

We can then continue on the exact GP procedures for GSGP and arrive at the following functions. They are equivalent to the equations in 2.53 [6].

$$\begin{aligned}
&\textbf{kernel function} : k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{w}_{\mathbf{x}_i}^T \mathbf{K}_G \mathbf{w}_{\mathbf{x}_j}, \\
&\textbf{posterior mean} : \bar{f}_* = \mathbf{w}_{\mathbf{x}_*}^T \mathbf{A}^{-1} \mathbf{K}_G \mathbf{W}^T \mathbf{y}, \\
&\textbf{posterior covariance} : cov(f_*) = \sigma_n^2 \mathbf{w}_{\mathbf{x}_*}^T \mathbf{A}^{-1} \mathbf{K}_G \mathbf{w}_{\mathbf{x}_*}, \\
&\textbf{NMLL} : \log p(\mathbf{y}) = \frac{1}{2}[\log\det(\mathbf{A}) + \sigma_n^{-2} \mathbf{y}^T (\mathbf{y} - \mathbf{W}\mathbf{A}^{-1} \mathbf{K}_G \mathbf{W}^T \mathbf{y}) + \log 2\pi + (n-m)\log \sigma_n^2],
\end{aligned} \tag{2.59}$$

where $\mathbf{A} = \mathbf{K}_G \mathbf{W}^T \mathbf{W} + \sigma_n^2 \mathbf{I}$.

From now on, GSGP and SKI share the same technique: precomputation + iteration.

The pre-computation phase of GSGP consists of storing the computation results of $\mathbf{W}^T\mathbf{W}$, $\mathbf{W}^T\mathbf{y}$, and $\mathbf{y}^T\mathbf{y}$, all of which are sufficient statistics of the regression problem. They can be done within $\mathcal{O}(n)$ time complexity.

The iteration phase consists of the calculation of the matrix-vector product of $(\mathbf{K}_G\mathbf{W}^T\mathbf{W} + \sigma_n^2\mathbf{I})$. Since $\mathbf{W}^T\mathbf{W}$ is sparse, $mvm(\mathbf{W}^T\mathbf{W}) = \mathcal{O}(m)$. As mentioned in KISSGP, $\mathcal{O}(mvm(\mathbf{K}_G)) = \mathcal{O}(m^2)$. Furthermore, considering that the number of iterations of the linear CG itself is much more than n, the overall time complexity per iteration is $\mathcal{O}(m^2)$.

However, due to the asymmetric nature of the matrix $\mathbf{K}_G\mathbf{W}^T\mathbf{W}$, methods suitable for symmetric positive definite matrices, e.g., CG to solve MVM, Lanczos algorithm to tridiagonalize a matrix in order to calculate the $\log\det(\cdot)$ term [4], can not be directly implemented in GSGP.

To solve this problem, a factorized approach of conjugate gradients and the Lanczos algorithm is provided. Taking the conjugate gradients for example, through inspection, we see that the key step in each iteration is the multiplication of $(\mathbf{K}_G\mathbf{W}^T\mathbf{W} + \sigma_n^2\mathbf{I})$ with an iterative vector $\mathbf{v}_i$. One could factorize $\mathbf{v}_i$ as $\mathbf{v}_i = \mathbf{W}\hat{\mathbf{v}}_i + c_i\mathbf{v}_0$, where $\hat{\mathbf{v}}_i \in \mathbb{R}^m$, c_i is a scalar coefficient, and $\mathbf{v}_0$ an initial value. By doing so, each iteration could be facilitated and freed from the symmetric constraint. Detailed implementations are presented in the original paper [6]. In general, this factorized approach grants both SKI and GSGP $\mathcal{O}(m^2)$ time complexity per iteration and the same convergence rates as the original CG or Lanczos algorithm.

In summary, this new method introduces a traditional Bayesian linear regression framework to SKI and leverages a factorized CG or Lanczos to achieve $\mathcal{O}(n)$ pre-computation time and $\mathcal{O}(m^2)$ per-iteration time. However, the problem of having an unlimited number of grid points is still not solved.

2.4.3 Kernel Interpolation with Sparse Grids

Kernel Interpolation with Sparse Grids is a method to deal with the grid point problem when increasing the number of dimensions. We can find that a basic technique used in the previous SKI-based method is that we can interpolate the covariance matrix regarding the input points with the covariance matrix of the on-grid inducing variables. This gives speed to the modern iterative methods (e.g., linear CG and Lanczos) to calculate the terms in 2.53 and 2.59.

However, as the dimension d of the input grows, the grid number scales exponentially in d [23]. Moreover, the default usage of local cubic interpolation also requires the non-zero entries in each row of $\mathbf{W}$ to scale exponentially, which impairs to sparsity of $\mathbf{W}$ (recall that $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ is approximated with $\tilde{\mathbf{K}} = \mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}\mathbf{W}^T$).

The new idea to cope with the increase in dimensions is by introducing sparse grids [41]. Thus we call this method SSKI. Sparse grids are constructed in the following way:

1. Define a one dimensional grid Ω_l with resolution index $l \in \mathbb{N}$ that partitions $[0, 1]$ into 2^l equal parts. The grid points are: $\Omega_l = \{\frac{i}{2^{l+1}} | 1 \leq i \leq 2^{l+1} - 1 \text{where } i \text{ is odd}\}$. Fixing i as an odd number will guarantee that no points in different resolutions overlap with each other [23]. This means that a pair (l, i) uniquely defines a single grid point.

2. Merge multiple one-dimensional grids to a d-dimensional grid. Define a resolution vector $\mathbf{l} = [l_1, l_2, ..., l_d]$ that contains information on the resolution in each dimen-

sion. The d-dimensional grid $\Omega_{\mathbf{l}} := \otimes_{j=1}^{d} \Omega_{l_j}$ is given by the Cartesian product ($\otimes$) of the 1-d grids.

3. Sparse grids are constructed by taking the union of the multi-dimensional grids with the L_1-norm of their resolution vector under a certain threshold. A sparse grid is uniquely specified by the number of dimensions d and the threshold t. Thus a d dimensional sparse grid with threshold t can be denoted as $\mathcal{G}_{d,t}$. One can refer to sections 1 and 2 of the original paper [23] for an illustration of a 2-dimensional sparse grid.

There are certain merits that sparse grids hold:

- The size of a sparse grid $\mathcal{G}_{d,t}$ with $\Theta(2^t)$ point in each dimension is $\mathcal{O}(2^t t^{d-1})$ where the size of a dense grid (only following steps 1 and 2 previously) is $\mathcal{O}(2^{td})$
- A sparse grid with smaller resolution $\mathcal{G}_{d,t_1}$ is contained in one with bigger resolution $\mathcal{G}_{d,t_2}$. i.e., $\mathcal{G}_{d,t_1} \subseteq \mathcal{G}_{d,t_2}$ where $t_1 < t_2$
- A sparse grid $\mathcal{G}_{d,t}$ can be constructed by Cartesian products of 1-d dense grids with $\mathcal{G}_{d-1,t-1}$. i.e., $\mathcal{G}_{d,t} = \bigcup_{i=0}^{t} (\Omega_i \otimes \mathcal{G}_{d-1,t-1})$

However, sparse grids are not Toeplitz anymore, thus we cannot directly implement FFT in the iterative methods in SKI which would take $\mathcal{O}(m \log m)$ time, where m is the number of grid points. To continue to have fast MVM with sparse grids, The original authors [23] provided a recursive algorithm that solves the MVM with sparse grid $\mathcal{G}_{d,t}$ in $\mathcal{O}(t^d 2^t)$ time. Notice that the size of a sparse grid $\mathcal{G}_{d,t}$ is $\mathcal{O}(2^t t^{d-1})$, this means near-linear time complexity. More specifically, the time complexity is $\mathcal{O}(m \log m)$. The algorithm is constructed based on two observations: Due to the third merit, one can recursively decompose the calculation $\mathcal{G}_{d,t}$ to several $\mathcal{G}_{d-1,t-1}$s. Due to the second merit, one can avoid certain calculations since the results of a lower-dimension sparse grid could be within the results of a higher dimension. This algorithm solves the problem of $\mathcal{O}(mvm(\mathbf{K}_{\mathbf{U},\mathbf{U}}))$.

The next problem is to solve $\mathcal{O}(mvm(\mathbf{W}))$. To do so, the authors combined simplicial interpolation [42] with a *combination technique* [23]. The simplicial interpolation allows the number of non-zero points in each row of $\mathbf{W}$ to grow quadratically ($\mathcal{O}(d^2)$) instead of exponentially ($\mathcal{O}(4^d)$). The combination technique combines the results of interpolation using each sub-dense grid within the sparse grid ($\binom{t+d-1}{d-1}$ sub-dense grids in total) since sparse grids cannot be directly used to interpolate. Thus the combination of these two methods yields $\mathcal{O}(mvm(\mathbf{W})) = \mathcal{O}(nd^2\binom{t+d-1}{d-1})$, compared with $\mathcal{O}(n4^d)$ using cubic interpolation with dense grids.

In summary, the overall time complexity could still be $O(n + m^2)$ using the same notions as previous methods. However, the constant terms that were previously omitted are decreased in this sparse SKI method.

2.4.4 Summary of Structural Approximations

The structural approximations are all built on the framework of SKI which was initially introduced in KISSGP. It approximates the kernel matrix of training points $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ via

the interpolation result of the kernel matrix of grid points $\mathbf{K}_{\mathbf{U},\mathbf{U}}$. This set of approximations further reduces the time complexity from $\mathcal{O}(m^2 n)$ to $\mathcal{O}(n + m^2)$ by leveraging the sparsity in the interpolation matrix and the Kronecker or Toeplitz structures of the grid kernel matrix. However, the number of grid points on one dimension is unlimited and the total number grows exponentially as the dimension grows. The first alternation GSGP reformulates SKI as a Bayesian linear regression problem which achieves similar time complexity as before but fails to tackle the problem of the number of grid points. The second alternation which uses sparse grids improves the scalability when the dimension grows but does not specify the number of grid points needed in one dimension. This leaves an open problem of how to determine the grid points per each dimension, which will be the main contribution of this thesis.

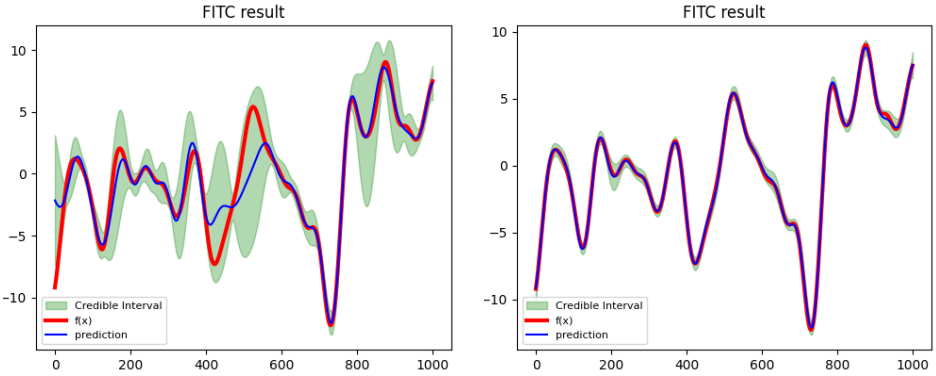
2.5 Comparison of the Approximations

In this section, a set of experiments is conducted on the approximation methods: GPR, FITC, SSGP, and KISSGP. The goal of each approximation method is to use a certain number of training points to learn the distribution that represents an underlying function from which training points are drawn.

For training data, we use the same underlying function as in Section 1.1.3, which is initialized with a RBF kernel with hyperparameters $\sigma_0^2 = 25, l = 30$. A total of $n = 1000$ training points were randomly generated at $x = 0, 1 \cdots, 999$ by adding white noise with $\sigma_n^2 = 0.25$ on the function. Experiments are further divided into three groups. In each group, for FITC, SSGP, and KISSGP, the number of inducing variables, basis functions, and grid points are set to the same value of 50, 100, or 200 respectively. In other words, the m of different methods is set to the same value in a group of experiments, where it varies between groups.

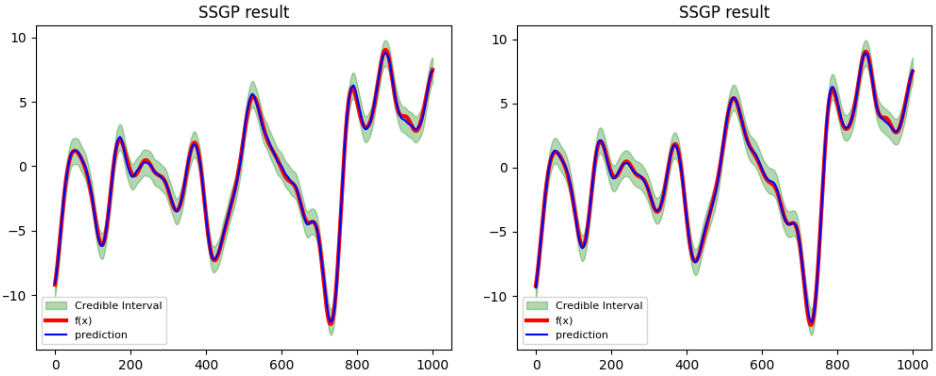
For each approximation method in each experiment group, we will take 100 Monte Carlo samples and calculate the average of 4 metrics: root mean square error (RMSE) between the ground truth $f(x)$ and predictive mean, the time per NMLL calculation, the additional time on top of NMLL calculation for its derivative. The reason for measuring the time per NMLL and dNMLL calculation instead of the time per iteration in the nonlinear CG algorithm is that the former two is a direct reflection of the time complexity of the approximation methods. The latter is unstable due to the initialization of hyperparameters, which can cause an uncontrollable amount of evaluations of the NMLL and dNMLL evaluations in every iteration. Moreover, in the experiments, we observe that some trials converge in very few iterations, but the number of function evaluations per iteration is high. On the other hand, there are also trials that converge in more iterations, but with fewer function evaluations per iteration. Thus we do not use time per iteration as an indicator of the efficiency of methods. We also measure these metrics of the precise GPR for comparison. To perform optimization, we implemented in all experiments the nonlinear CG (see algorithm 3) with maximum iteration of 20 and gradient norm threshold $\epsilon = 0.1$.

The experiments were run on a 2.30GHz Intel Core i7-11800H CPU. Below are examples of the posterior distribution under different settings in 1 Monte Carlo sample.



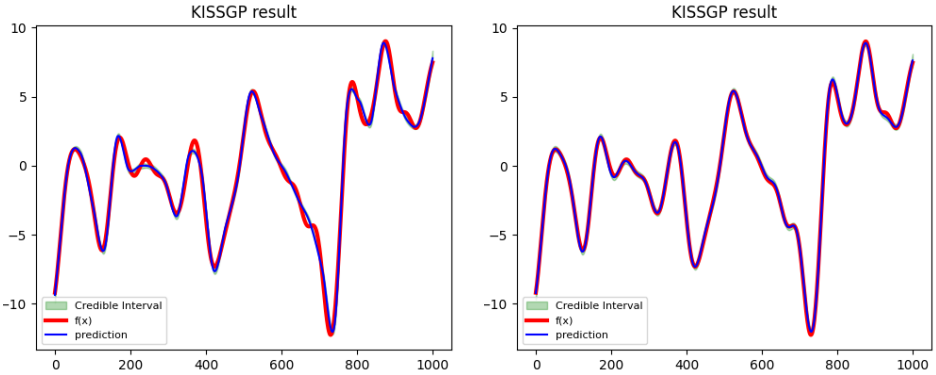
(a) FITC with 50 grid points

(b) FITC with 200 grid points



(c) SSGP with 50 basis functions

(d) SSGP with 200 basis functions



(e) KISSGP with 50 grid points

(f) KISSGP with 200 grid points

Figure 2.2: Comparison of the posterior distributions of different approximation methods. The first column contains the results with $m = 50$, whereas the second column contains the results with $m = 200$.

In Figure 2.2, the posterior means are drawn in blue curves, while the exact underlying function is drawn in red. The credible interval associated with the prediction, which is calculated by the posterior covariance is depicted as a green corridor with the same meaning as in Section 1.1.

First, let us analyze the results vertically. From the first column of results, we observe that with only 50 inducing variables, basis functions, or grid points, FITC has the worst performance among the three methods. This implies that, though the time complexities are at the same level for FITC and SSGP, the number m required for both methods to reach the same level of accuracy is different, where FITC requires more. This is most likely due to the fact that SSGP approximates the RBF kernel in the frequency domain, where it captures the dominant frequencies within a few m. Also, comparing FITC with KISSGP, where the two methods approximate the RBF kernel from the same perspective, we see KISSGP extracts more information from the full knowledge of 50 giving points. Next, comparing the results horizontally, we see that all three methods had an improvement in accuracy when $m = 200$, especially FITC. This illustrates the fact that having a higher m will lead to higher accuracy, though sacrificing time.

In the following table, the average error, the average time spent per NMLL, its derivative dNMLL, and the two together are listed and compared. The best data of each metric in each experiment is depicted in either green or blue.

Method	RMSE	t_{NMLL}(ms)	t_{dNMLL}(ms)	$t_{\text{NMLL}} + t_{\text{dNMLL}}$(ms)
GPR	0.108	73.67	49.76	123.43
FITC-50	42.66	11.74	3.73	15.47
SSGP-50	49.40	4.96	8.30	13.26
KISSGP-50	0.250	7.29	0.14	7.43
FITC-100	0.144	22.26	5.72	27.98
SSGP-100	0.310	13.72	19.08	32.80
KISSGP-100	0.126	26.38	0.75	27.13
FITC-200	0.115	50.35	16.30	66.66
SSGP-200	0.205	31.64	42.51	74.15
KISSGP-200	0.108	45.12	1.00	46.12

Table 2.2: Accuracy and time of different approximations

From the table, we observe that in all three groups of experiments, KISSGP achieved the lowest error. It is worth noticing that KISSGP is the most efficient algorithm in terms of m since it reached a level of accuracy similar to GPR first at 200 grid points. SSGP has the fastest NMLL calculation. However, in each iteration of the optimization algorithm, the NMLL and dNMLL are evaluated together, thus the more valuable information is $t_{\text{NMLL}}+t_{\text{dNMLL}}$, which KISSGP is also superior to other methods due to its little effort in evaluating dNMLL. Some may notice that the SSGP acquired exceptionally worse results compared with the other two in terms of t_{dNMLL}. This is due to the fact that the SSGP needs to optimize $m+2$ hyperparameters in total, compared with 2 for FITC and KISSGP. Moreover, we observe the highest error from SSGP as m increases, though it presented relatively better qualities when m is low.

In conclusion, the FITC approximation has improved performance compared with precise GPR, however, it has worse results than KISSGP in every aspect. The SSGP approximation has its advantage in the freedom of model, as it can approximate any stationary kernel without further assumptions. It also has the fastest NMLL calcula-

tion. On the other hand, due to the increase of the number of hyperparameters as the number of basis functions grows, it suffers from long-duration derivative calculation. Finally, the KISSGP approximation requires the least amount of m to reach near-GPR accuracy and also has the fastest calculation speed. We thus regard KISSGP as the state-of-the-art method that shows superior quality to other methods in various aspects. Our proposed low-rank approximation will be built on the foundation of KISSGP.

2.6 Summary of Existing Low-Rank Approximations

In this chapter, we explained the existing low-rank approximations in four different categories: Nyström approximations, prior approximations, spectral approximations, and structural approximations.

The Nystroöm approximation is one of the earliest low-rank approximations. The main idea is to approximate the integration in the original eigenfunction problem of the kernel function using an empirical average of finite samples. In terms of the kernel matrix, it approximates the training kernel matrix by only MVMs of its submatrices:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}, \tag{2.60}$$

where $\mathbf{K}_{\mathbf{X},\mathbf{U}}$ is the kernel matrix formed by the input sets $\mathbf{X}$ where $\mathbf{U}$ and $\mathbf{U}$ is a subset of the complete training data $\mathbf{X}$. Since $\mathbf{U}$ is a subset of $\mathbf{X}$, it succeeded in the task of decomposing the training kernel matrix into low-rank kernel matrices. However, it often produces negative covariance values on the posterior covariance matrix's diagonal, which is invalid (covariance with oneself should be non-negative). Nonetheless, it approximates the posterior mean and has a time complexity of $\mathcal{O}(m^2n)$.

The prior approximations, which are the most classical low-rank approximations approach the low-rank approximation problem by proposing additional probabilistic assumptions between the training input, testing input, and a set of special inputs: the inducing variables $\mathbf{U}$. This set of inducing variables need not be a subset of training inputs anymore. Due to this additional assumption, the covariance structure between the training inputs and the test inputs also changes, which results in that the posterior covariance is now always valid. The most popular method in this family is the FITC approximation, which decomposes the training kernel matrix as:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}} + \mathbf{\Lambda}, \tag{2.61}$$

where $\mathbf{\Lambda} = \operatorname{diag}\left[\mathbf{K}_{\mathbf{X},\mathbf{X}} - \mathbf{K}_{\mathbf{X},\mathbf{U}}\mathbf{K}_{\mathbf{U},\mathbf{U}}^{-1}\mathbf{K}_{\mathbf{U},\mathbf{X}}\right]$ is the diagonal correction term. The time complexity of FITC is also $\mathcal{O}(m^2n)$, where m is the number of inducing variables.

The methods in the previous two categories all have to assume a specific form of kernel function (e.g., RBF kernel) beforehand. The structural approximations loosen this constraint and work for any stationary (or furthermore isotropic) kernels. This is achieved because the structural approximation directly approximates the spectrum of the kernel function by using a finite sum of basis functions. The most popular spectral approximation, SSGP, uses sinusoid functions as the basis functions and has its decomposition of the kernel matrix as:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \frac{\sigma_0^2}{m} \mathbf{\Phi}_{\mathbf{X}}^T \mathbf{\Phi}_{\mathbf{X}}, \tag{2.62}$$

where $\mathbf{\Phi}_{\mathbf{X}} = [\boldsymbol{\phi}(\mathbf{x}_1)\ \boldsymbol{\phi}(\mathbf{x}_2)\ \ldots \boldsymbol{\phi}(\mathbf{x}_n)]$.

$\boldsymbol{\phi}(\mathbf{x}_n) = \left[\cos \boldsymbol{\omega}_1^T \mathbf{x}_n\ \sin \boldsymbol{\omega}_1^T \mathbf{x}_n\ \ldots \cos \boldsymbol{\omega}_m^T \mathbf{x}_n\ \sin \boldsymbol{\omega}_m^T \mathbf{x}_n\right]^T$ is the vector of basis functions evaluated at $\mathbf{x}_n$ and $\boldsymbol{\omega}_1$ to $\boldsymbol{\omega}_m$ are the frequencies of the basis functions that should be learned as hyperparameters in training. The time complexity is also $\mathcal{O}(m^2 n)$, where m is the number of basis functions. However, since these m frequencies have to be learned, which is much more compared with Nyström approximation and prior approximation which only has to learn the number of hyperparameters in the kernel function, the training process may be unstable.

The structural approximations are the newest approximations that approximate the training kernel matrix by interpolation. Typically in the KISSGP approximation:

$$\mathbf{K}_{\mathbf{X},\mathbf{X}} \approx \mathbf{W} \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{W}^T, \tag{2.63}$$

where $\mathbf{U}$ is a set of equispaced grid points and $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ is its kernel matrix. $\mathbf{W}$ is the interpolation matrix. In structural approximations, the inverse and log-determinant terms of the noisy training kernel matrix are mainly calculated using iterative algorithms that only require MVM (but not inverting or taking the determinant) of $\mathbf{K}_{\mathbf{X},\mathbf{X}}$. The algorithms leverage the sparsity in $\mathbf{W}$ and the Toeplitz and possible Kronecker structure in $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ when considering multi-dimensions which allows faster MVM operations than normal matrices. This gives the structural approximations a substantial speed boost with a time complexity of only $\mathcal{O}(n + m^2)$, where m is the number of grid points. Experiments show that indeed the KISSGP reaches the same level of accuracy as the precise GPR within the shortest time compared with other methods. However, there is still the problem of m being unlimited. There are improvements made to KISSGP to deal with the problem of m growing exponentially as the number of dimensions grows, but the necessary number of grid points in one dimension is still unclear.

In the next chapter, we will focus on finding the methodology to calculate the number of grid points in one dimension, and develop a new approximation method from it.

Proposed Low-Rank Approximation

3

Kernel interpolation for scalable structured Gaussian process (KISSGP) approximation presented an invaluable framework: structural kernel interpolation (SKI) to extract information of the grid kernel matrix. From the expression in its time complexity and from the experiments in Section 2.5, we observe that as we elevate the number of grid points, the time consumed is also increased with an exchange of higher accuracy. However, as we can see from Table 2.2, the accuracy will converge to the accuracy of GPR, which means that further increase of grid points will be futile. This calls for the need to limit the amount of grid points in order to conserve time. However, it is not possible to know in advance the required number of grid points to reach a near-GPR accuracy for a new problem.

In the original KISSGP paper [33], we only know that grid points are fixed throughout learning. The number of grid points, however, is unknown. Efforts have been made in the literature to overcome the problem of exponential growth of grid points as the dimension increases [23][24][25]. However, the effective number of grid points in every single dimension is still unclear.

In our proposed approximation method malleable kernel interpolation for scalable structured Gaussian process (MKISSGP), we introduce a new parameter: *density*, to calculate the number of grid points per dimension according to the length scale of kernels. In this thesis, we use the most popular RBF kernel to calculate density and thus the number of grid points. Furthermore, we propose a framework where the number of grid points change along with the length scale during the training process, thus the name MKISSGP is coined. This new change in SKI can trivially fit with existing algorithms to calculate the inverse and log-determinant. The time complexity remains to be $\mathcal{O}(n + m^2)$, however, the number of m is potentially reduced compared with KISSGP.

In this chapter, the first section will present the detailed problem formulation of MKISSGP. The second section will discuss several possible interpolation methods with respect to their accuracy, time consumption, and compatibility with the SKI framework. The third section will focus on determining the necessary amount of grid points needed. At last, we propose the new interpolation scheme MKISSGP that integrates the results of the previous two sections into the standard KISSGP.

For the following discussions, due to we only focus on one dimension, the entries for functions will be scalars.

3.1 Problem Formulation

In KISSGP, the most essential assumption is that the kernel function $k(x_i, x_j)$ can be interpolated by $\sum_i w_i k(u_i, x_j)$, where u_i are the grid points surrounding x_i. It is worth

noticing that this relationship, on the one hand, holds for:

$$k(x_i, u) \approx \sum_i w_i k(u_i, u), \tag{3.1}$$

where u is an arbitrary grid point. This leads to $\mathbf{K_{X,U}} \approx \mathbf{W K_{U,U}}$.

On the other hand, it should also satisfy:

$$k(x_i, x) \approx \sum_i w_i k(u_i, x), \tag{3.2}$$

where x is an arbitrary input point. Assuming an isotropic kernel function, this leads to the final approximation $\mathbf{K_{X,X}} \approx \mathbf{K_{X,U}} \mathbf{W}^T \approx \mathbf{W K_{U,U}} \mathbf{W}^T$.

Noticing the fact that all x lies within the span of grid points, 3.1 and 3.2 together implies that:

$$\forall x_i, a \in [u_1, u_m], \quad \exists\ \mathbf{w}_i \quad \text{s.t.} \quad k(x_i, a) \approx \mathbf{w}_i^T k(\mathbf{u}, a), \tag{3.3}$$

where $[u_1, u_m]$ is the span of grid points, $\mathbf{w}_i$ is the weight vector, $k(\mathbf{u}, a)$ is a column formed by the kernel values of each grid point and a.

To limit the number of grid points used, we have to tackle with the basic problem of interpolating the kernel function. The accuracy of this interpolation is mainly determined by three factors: choice of interpolation methods, sampling density, and smoothness of data [43]. The third factor is naturally satisfied since we use the RBF kernel which guarantees smoothness.

By either implementing an interpolation method with higher accuracy or controlling the sampling density, we can have a shrink in the number of grid points needed to reach a certain level of accuracy. Currently, KISSGP only utilizes the information of kernel value at grid points during the interpolation. Much more information is actually present if we assume the RBF kernel: the actual form of the function and its infinite orders of derivatives. Moreover, not only the kernel value at grid points are available, but the value at training points could also enhance the interpolation.

Thus the following two sections will focus on the exploration of interpolation methods that integrate more information and a way to control the sampling density.

3.2 Exploration of Interpolation Methods

In this section, the procedure, result, and feasibility of several interpolation methods are analyzed. The function being interpolated as shown in Figure 3.1 will be the RBF kernel $k(x; 5)$ with hyperparameters $\sigma_0^2 = 100$ and $l = 0.5$. The x being interpolated will be in the range of $[0, 10]$. We use 8 grid points to cover the span and 2 additional grid points at the two ends to ensure the feasibility of most interpolation methods. The setting aims to emulate a grid with low density to test the ability of the interpolations to provide accurate results with sparse grids. When testing the interpolation methods, we will compare their interpolation result with this kernel function and measure the RMSE.

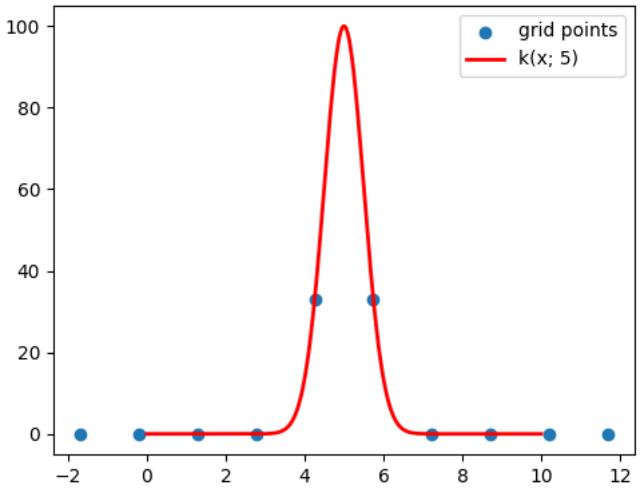


Figure 3.1: The ground truth kernel function

The original interpolation method used in KISSGP is the cubic convolution interpolation (CCI), where the only information required are the relative distances from the interpolated points and the grid points and the function values at the grid points [39]. However, assuming an RBF kernel in the SKI settings, much more information could be provided, which are the actual form of the function, its infinite orders of derivatives, and the function values at all data points. In the following exploration of interpolation methods, we will explore more interpolation methods that try to exploit this additional information and compare the results with the CCI in terms of accuracy and feasibility in the SKI framework.

Due to the fact that the goal of interpolating a kernel function in the framework of SKI is with respect to the first entry as shown in 3.1 and 3.2, in this section, we refer to the function being interpolated as $f(x)$, or equivalently, $k(x; a)$, where the second entry a is an arbitrary point within the span of the training points, and considered as a parameter. The interpolation function is noted as $g(x; a)$, with similar meanings as $k(x; a)$.

3.2.1 Cubic Convolution Interpolation

The CCI is the standard interpolation method implemented in the original KISSGP paper [39]. To interpolate the kernel value $k(x; a)$ at x using an interpolated function $g(x; a)$, the interpolation uses four grid points closest to x, that is $k(x; a) \approx g(x; a) = w_1 k(u_1; a) + w_2 k(u_2; a) + w_3 k(u_3; a) + w_4 f(u_4; a)$. The weights $\{w_1, w_2, w_3, w_4\}$ are determined by a set of functions with 8 free parameters:

$$w(s) = \begin{cases} A_1|s|^3 + B_1|s|^2 + C_1|s| + D_1 & 0 < |s| < 1 \\ A_2|s|^3 + B_2|s|^2 + C_2|s| + D_2 & 1 < |s| < 2 \\ 0 & 2 < |s| \end{cases}, \tag{3.4}$$

where $s = \frac{x-u}{u_{i+1}-u_i}$ is the relative distance from a grid point to the interpolation location. The eight free parameters are further determined by imposing certain properties of the interpolation:

1. Accurate result when interpolating at grid points, i.e., $k(x;a) = g(x;a), \quad x \in \mathcal{U}$. This requires $w(0) = 1$, $w(1^-) = 0$, $w(1^+) = 0$, $w(2) = 0$
2. Smoothness, i.e., the function and its first derivative are continuous. This requires $w'(0^-) = w'(0^+)$, $w'(1^-) = w'(1^+)$, $w'(2^-) = w'(2^+)$.
3. The first three terms of the Taylor expansion of $f(x;a)$ and $g(x;a)$ around the nearest grid point coincide, i.e., $f(x;a) - g(x;a) = \mathcal{O}(d_u)^3$, where $d_u = u_{i+1} - u_i$. This means that the interpolation can accurately approximate any quadratic function.

These constraints together give a unique set of parameters for the function:

$$w(s) = \begin{cases} 1.5|s|^3 - 2.5|s|^2 + 1 & 0 < |s| < 1 \\ -0.5|s|^3 + 2.5|s|^2 - 4|s| + 2 & 1 < |s| < 2 \\ 0 & 2 < |s| \end{cases}. \tag{3.5}$$

The CCI generates the following interpolation shown in blue:

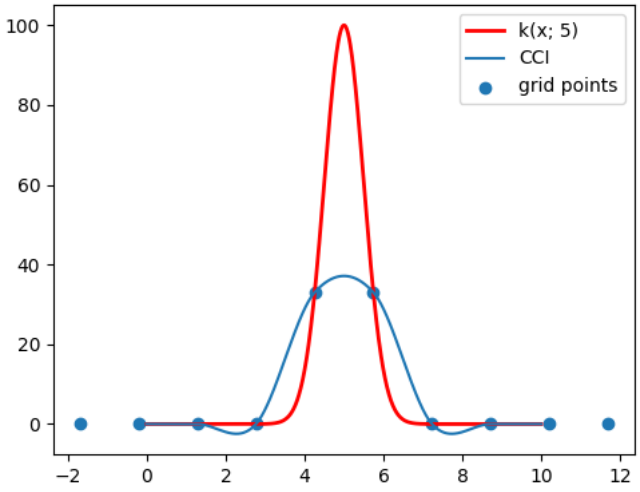


Figure 3.2: Interpolation results of CCI. The blue curve is the CCI which flattened the original peak due to the insufficient density of grids.

From Figure 3.2 we see that the interpolation captured the trend of the kernel function and produced a smooth flat peak. The RMSE of the CCI is 17.60, together with the plot they demonstrate that CCI doesn't perform well with this level of grid sparsity.

The CCI provides a fixed set of parameters regardless of the second entry of the kernel function. This is because the weights are only determined by the relative distances s from the training points to the grid points, which are fixed after selecting the

grids from the beginning. In other words, in KISSGP we can simply calculate and use only one interpolation matrix $\mathbf{W}$ throughout the training process.

3.2.2 Exponential Cubic Convolution Interpolation

We know the form of the RBF kernel from 1.5 that it is an exponential quadratic function. Combining this information with the third property of the CCI, we know that the function can be fully recovered by first taking the logarithm, then interpolating, and taking the exponential back. Thus, this gives us the intuition to combine exponential interpolation [44] with CCI to form a new interpolation method: exponential cubic convolution interpolation (ECCI).

First, we implement the CCI on the logarithms of the kernel values to acquire the interpolation result in the logarithm space. That is:

$$g_{log}(x; a) = \sum_{i=1}^{4} w_i \log k(u_i; a), \tag{3.6}$$

where the weights w_i are the same as in the original CCI. Notice that by taking the logarithm of 1.5, the remaining term is a quadratic function, i.e., , $\log k(u_i; a) = \log \sigma_s^2 - \frac{(u_i - a)^2}{2l^2}$, thus $g_{log}(x; a) = k_{log}(x; a)$. Then we take the result back to linear space by taking the exponential:

$$g(x; a) = \exp(g_{log}(x; a)). \tag{3.7}$$

We also have the property that $g(x; a) = \exp(g_{log}(x; a)) = \exp(k_{log}(x; a)) = k(x; a)$. The resulting interpolation is guaranteed to completely reconstruct the original kernel function regardless of the density of grid points, which is the highest level of accuracy that any interpolation method can achieve.

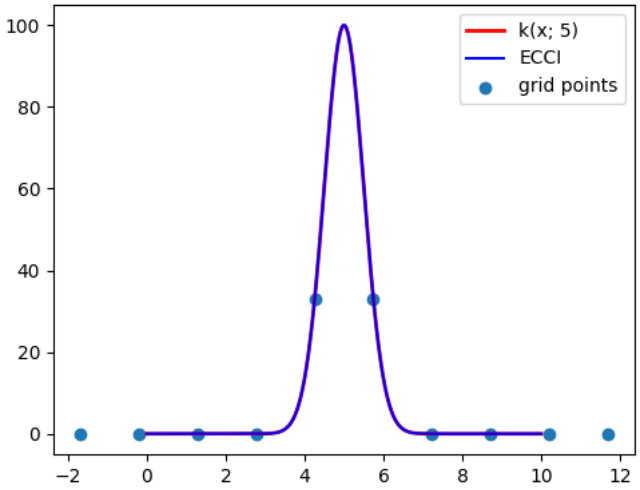


Figure 3.3: Interpolation results of ECCI. The interpolation and ground truth overlap because the RBF kernel is fully reconstructed by ECCI.

Figure 3.3 shows that indeed the exponential CCI has no error in interpolating the RBF kernel. We can have an even more exaggerated plot to demonstrate its accuracy:

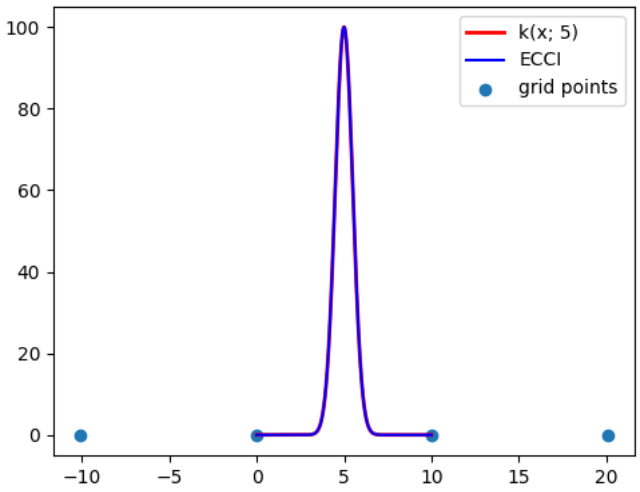


Figure 3.4: ECCI with only 4 grid points. In this plot, we reset the number and position of grid points so that all the interpolated points are within the center interval of the 4 grid points.

where only the necessary amount of grid points to perform ECCI is used. We see that ECCI still fully reconstructs the original function.

Despite the maximum accuracy, a critical fact for the exponential CCI is that it fails to provide speed-up for the GPR approximation because the approximation of the kernel matrix is still full rank:

$$\mathbf{K}_{ecci} = exp(\mathbf{W} \log \mathbf{K}_{\mathbf{U},\mathbf{U}} \mathbf{W}^T), \tag{3.8}$$

where the exponential and logarithm of the matrices are taken entry-wise, thus $\mathbf{K}_{ecci}$ is still a full-rank matrix. From another aspect, since we will have no error in reconstructing the original kernel matrix, the resulting expression must have the same rank as before. Thus, using the form of $\mathbf{K}_{ecci}$ will have the same time complexity as using the exact $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ in inverse and determinant operations. Thus we consider this powerful interpolation method for the RBF kernel unsuitable for the SKI framework.

3.2.3 Dutch Taylor Expansion Interpolation

In [45], the author devised a new interpolation mechanism to perform ray field mapping. Compared with the CCI where derivative information is implicitly leveraged, i.e., no explicit calculation of the function's derivative, this new method can utilize the kernel's derivative information at any order. This meets the fact that the derivative information of the RBF kernel is fully known.

The form of the interpolation is similar to linear interpolation, as it considers information from only 2 nearby grid points:

$$g(x) = \frac{u_2 - x}{u_2 - u_1}\mathcal{D}_n[f(x)](x; u_1) + \frac{x - u_1}{u_2 - u_1}\mathcal{D}_n[f(x)](x; u_2), \tag{3.9}$$

where u_1 and u_2 are the nearby grid points. $\mathcal{D}_n[f(x)](x;\xi)$ is the n-order Dutch Taylor expansion (DTE) [45] of $f(x)$ around point ξ, which has the explicit formula:

$$\mathcal{D}_n[f(x)](x;\xi) = \sum_{k=0}^{n}\left(1 - \frac{k}{n+1}\right)\frac{1}{k!}(x-\xi)^k f^{(k)}(\xi). \tag{3.10}$$

$f^{(k)}(\xi)$ is the k-order derivative of $f(x)$ evaluated at ξ. $f^{(0)}(\xi)$ is defined as the function value itself. In this case, a 0-order DTE interpolation is the linear interpolation, i.e., $\mathcal{D}_n[f(x)](x;\xi) = x - \xi$. The difference between the traditional Taylor expansion and the DTE is that the DTE has an additional $(1 - \frac{k}{n+1})$ factor in the formula. This alternation from the traditional Taylor expansion allows the method to decrease interpolation error to a polynomial of order $n+1$ while the traditional Taylor expansion could not. Considering only using the first-order expansion to interpolate a general quadratic function as an example:

The general quadratic function is given by:

$$f(x) = c_0 + c_1 x + c_2 x^2. \tag{3.11}$$

Using the traditional Taylor expansion in the interpolation gives:

$$\begin{aligned} g(x) &= \frac{u_2 - x}{u_2 - u_1}\left(f(u_1) + (x - u_1)f^{(1)}(u1)\right) + \frac{x - u_1}{u_2 - u_1}\left(f(u_2) + (x - u_2)f^{(1)}(u2)\right) \\ &= \frac{u_2 - x}{u_2 - u_1}(c_0 + c_1 x + c_2(2x - u_1)u_1) + \frac{x - u_1}{u_2 - u_1}(c_0 + c_1 x + c_2(2x - u_2)u_2) \\ &= f(x) + c_2(x - u_1)(x - u_2), \end{aligned} \tag{3.12}$$

which the error term is a second-order polynomial. On the other hand, using the DTE in the interpolation gives:

$$\begin{aligned} g(x) &= \frac{u_2 - x}{u_2 - u_1}\left(f(u_1) + \frac{1}{2}(x - u_1)f^{(1)}(u1)\right) + \frac{x - u_1}{u_2 - u_1}\left(f(u_2) + \frac{1}{2}(x - u_2)f^{(1)}(u2)\right) \\ &= \frac{u_2 - x}{u_2 - u_1}(c_0 + \frac{1}{2}c_1 u_1 + \frac{1}{2}c_1 x + c_2 u_1 x) + \frac{x - u_1}{u_2 - u_1}(c_0 + \frac{1}{2}c_1 u_2 + \frac{1}{2}c_1 x + c_2 u_2 x) \\ &= c_0 + c_1 x + c_2 x^2 \\ &= f(x), \end{aligned} \tag{3.13}$$

where the error term vanishes. This shows that by mitigating the effect of derivatives on both sides, the compromised result of DTE is more accurate than using traditional Taylor expansion. This is the reason this expansion is called "Dutch" Taylor expansion [45].

In theory, if the original function is infinitely differentiable and we can acquire the exact form of every derivative, we can leverage the DTE interpolation to completely

reconstruct the function between two grid points, which is a nice property satisfied by the RBF kernel. Here we list the corresponding form of $f(x)$ and its first two derivatives [46].

function	form
$k(x;a)$	$\sigma_s^2 \exp(-\frac{1}{2l^2}(x-a))$
$k^{(1)}(x;a)$	$-\frac{x-a}{l^2}k(x;a)$
$k^{(2)}(x;a)$	$\frac{1}{l^2}[\frac{1}{l^2}(x-a)^2-1]k(x;a)$

Table 3.1: RBF kernel and its derivatives

Since we only take the derivative of one variable in the RBF kernel, and the second variable has no relationship with the first, we can consider the second variable as a parameter a of $k(x;a)$, as shown above.

The results of 0 to 2 order DTE interpolation are compared in Figure 3.5.

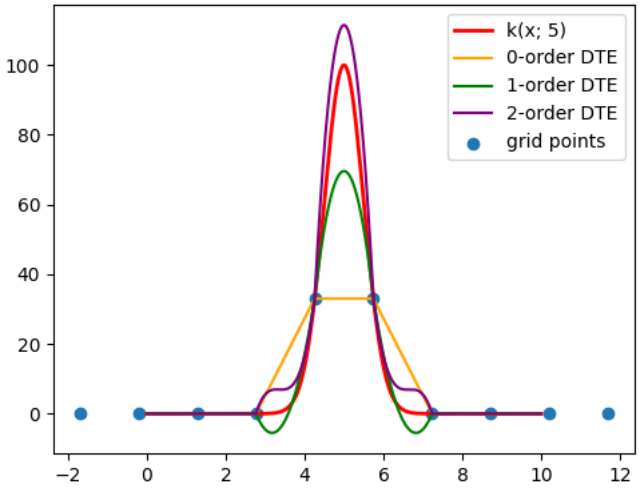


Figure 3.5: Interpolation results of different orders of DTE interpolation. The 1-order and 2-order DTE now generate a higher peak with error shrinking as the order increases. The 0-order DTE is equivalent to the linear interpolation.

The RMSE of each order DTE interpolation compared with the CCI is given in the following table:

method	RMSE
CCI	17.60
0-order DTE	18.59
1-order DTE	7.45
2-order DTE	5.71

Table 3.2: RMSE of different order Dutch Taylor Expansion Interpolations

From Figure 3.5 and Table 3.2 we observe that the interpolation error shrinks as

the order of DTE increases. Moreover, the DTE interpolation only requires 2 nearby grid points to operate.

Despite the improved performance of the 1-order and the 2-order DTE in the experimented interpolation task, it hardly fits the SKI framework. This is because the derivative values of $k(x;a)$ evaluated at x are also subject to the second entry a, as shown in Table 3.1. Thus, weights $\mathbf{w}_i$ acquired through this method are not tailored only for a specific input point $\mathbf{x}_i$, but instead for both entries in the kernel function. In other words, using the set of weights $\mathbf{w}_i$ that achieves small error in $k(\mathbf{x}_i, \mathbf{u}_1) \approx \sum_i w_i k(\mathbf{u}_i, \mathbf{u}_1)$ may completely fail to interpolate $k(\mathbf{x}_i, \mathbf{u}_2)$ with $\sum_i w_i k(\mathbf{u}_i, \mathbf{u}_2)$. This implies that we cannot find a set of weights that achieves all of 3.1 and 3.2 for the DTE approximation. The 0-order DTE, on the other hand, though doesn't require the second entry to calculate, degrades to a simple linear interpolation which has worse results than the CCI. In summary, we say that the DTE interpolation does not fit in the SKI framework.

3.2.4 Optimal Interpolation

Finally, let's explore the interpolation method that utilizes the kernel values we know at all data points. In [47], the authors proposed a general mathematical idea of optimizing spline interpolation coefficients by minimizing the error between interpolated points and observations. The coefficients are thus acquired by solving a linear system. In [48], this method is implemented for image super-resolution.

We can also use this idea to calculate an optimal set of weights for each input. Once again, the problem is to find a fixed set of weights that satisfy the approximations 3.1 and 3.2. We can find this set of weights by forming the problem as a linear system. Suppose that we also use 4 nearby grid points to interpolate the value of $k(x_i, a)$ at x_i, then the 4 weights are optimized by solving:

$$\begin{bmatrix} k(u_{i,1},u_1) & k(u_{i,2},u_1) & k(u_{i,3},u_1) & k(u_{i,4},u_1) \\ k(u_{i,1},u_2) & \dots & & k(u_{i,4},u_2) \\ \vdots & & & \vdots \\ k(u_{i,1},u_m) & \dots & & k(u_{i,4},u_m) \\ k(u_{i,1},x_1) & k(u_{i,2},x_1) & k(u_{i,3},x_1) & k(u_{i,4},x_1) \\ \vdots & & & \vdots \\ k(u_{i,1},x_n) & \dots & & k(u_{i,4},x_n) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} k(x_i,u_1) \\ \vdots \\ k(x_i,u_m) \\ k(x_i,x_1) \\ \vdots \\ k(x_i,u_n) \end{bmatrix}. \tag{3.14}$$

We can further divide 3.14 in to blocks:

$$\begin{bmatrix} \mathbf{K}_{\mathbf{u},i} \\ \mathbf{K}_{\mathbf{x},i} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} \mathbf{k}_{\mathbf{u},i} \\ \mathbf{k}_{\mathbf{x},i} \end{bmatrix}, \tag{3.15}$$

where $\mathbf{K}_{\mathbf{u},i}$ and $\mathbf{k}_{\mathbf{u},i}$ are the blocks consisting the first m rows. $\mathbf{K}_{\mathbf{x},i}$ and $\mathbf{k}_{\mathbf{x},i}$ are the blocks consisting the last n rows. The least-squares solution of 3.15 could be given by the formula:

$$\mathbf{w} = (\mathbf{K}_{\mathbf{ux},i}^T \mathbf{K}_{\mathbf{ux},i})^{-1} \mathbf{K}_{\mathbf{ux},i}^T \mathbf{k}_{\mathbf{ux},i}, \tag{3.16}$$

where $\mathbf{w} = [w_1\ w_2\ w_3\ w_4]^T$, $\mathbf{K}_{\mathbf{ux},i} = [\mathbf{K}_{\mathbf{u},i}\ \mathbf{K}_{\mathbf{x},i}]^T$, $\mathbf{k}_{\mathbf{ux},i} = [\mathbf{k}_{\mathbf{u},i}\ \mathbf{k}_{\mathbf{x},i}]^T$. Because the grid points $\mathbf{U}$ cover the span of $\mathbf{X}$, an approximated solution of the complete linear system is by solving only either $\mathbf{K}_{\mathbf{u},i}\mathbf{w} = \mathbf{k}_{\mathbf{u},\mathbf{i}}$ or $\mathbf{K}_{\mathbf{x},i}\mathbf{w} = \mathbf{k}_{\mathbf{x},\mathbf{i}}$. Since the time complexity of using $\mathbf{K}_{\mathbf{x},i}$ and $\mathbf{K}_{\mathbf{ux},i}$ are similar since $m < n$, we only test the results of using $\mathbf{K}_{\mathbf{u},i}$ or using $\mathbf{K}_{\mathbf{ux},i}$, which are shown in the following plot:

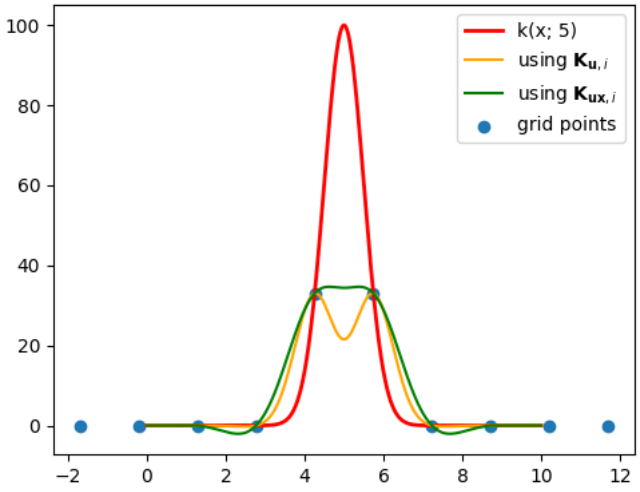


Figure 3.6: Interpolation results of the optimal interpolation. No improved results are delivered by the optimal interpolation when we only consider the results of approximating $k(x; 5)$.

The RMSE of each method is shown in Table 3.3.

method	RMSE
CCI	17.60
Optimal-$\mathbf{K}_{\mathbf{u},i}$	20.74
Optimal-$\mathbf{K}_{\mathbf{ux},i}$	18.12

Table 3.3: RMSE of interpolating $k(x; 5)$ using optimal interpolations of different settings

It seems that the optimal interpolation is not "optimal" compared with the CCI. However, the comparison of the interpolation results in only $k(x; 5)$ is unfair for the optimal interpolation since it solves the least square problem for the system consisting of all grid points or with all data points together. For a single point (i.e., 5) in the system, it might have worse performance than CCI. Thus in the following table, we compare the RMSE of the methods in reconstructing the entire matrix:

$$\mathbf{W}\mathbf{K}_{\mathbf{u},\mathbf{ux}} \approx \mathbf{K}_{\mathbf{x},\mathbf{ux}}, \tag{3.17}$$

where $\mathbf{K}_{\mathbf{u},\mathbf{ux}} = [\mathbf{K}_{\mathbf{u},\mathbf{u}}\ \mathbf{K}_{\mathbf{u},\mathbf{x}}]$ and $\mathbf{K}_{\mathbf{x},\mathbf{ux}}$ has similar meanings. Also, we tested the results when using only 2 nearby grid points in the optimal interpolation.

method	RMSE
Cubic	13.12
Optimal-$\mathbf{K}_{\mathbf{u},i}$-2 points	14.53
Optimal-$\mathbf{K}_{\mathbf{u},i}$-4 points	14.51
Optimal-$\mathbf{K}_{\mathbf{ux},i}$-2 points	12.64
Optimal-$\mathbf{K}_{\mathbf{ux},i}$-4 points	12.50

Table 3.4: RMSE of interpolating $\mathbf{K}_{x,ux}$ using optimal interpolations of different settings

From Table 3.4 we have several observations. First, we see that the improvement of the optimal interpolation from using 2 grid points to 4 grid points is not substantial, which means 2 nearby grid points are enough for optimal interpolation. Second is that only by using the entire $\mathbf{K}_{\mathbf{ux},i}$ can we have only minor improvements in terms of accuracy compared with CCI. This slight accuracy improvement is easily overshadowed by its required time complexity to calculate each weight vector since the calculation of $\mathbf{K}_{\mathbf{ux},i}^T\mathbf{K}_{\mathbf{ux},i}$ is a $\mathcal{O}((n+m)^2)$ calculation, compared with $\mathcal{O}(1)$ for calculating a CCI weight vector. Even if we implement iterative solvers for the linear system to accelerate its calculation, it still is much slower than CCI. Despite its in inefficiency in weight calculation, it is the only new interpolation scheme that fits in the SKI framework.

3.2.5 Summary of the Interpolation Methods

In summary, The CCI and optimal interpolation among the four interpolation methods discussed in this section are qualified candidates that fit in the SKI framework. The optimal interpolation, though having slightly better accuracy in interpolating $\mathbf{K}_{\mathbf{x},\mathbf{u}}$ and $\mathbf{K}_{\mathbf{x},\mathbf{x}}$, is very slow in calculating the weights with the time complexity of $\mathcal{O}((n+m)^2)$ in calculating one weight vector. In comparison, the CCI used only $\mathcal{O}(1)$ time to achieve approximately the theoretical optimal result in terms of accuracy. As for the other two methods, the exponential CCI though has no errors, failed to provide acceleration due to the fact that the decomposed kernel is still a full-rank kernel matrix. The DTE interpolation does not fit in the SKI framework since it cannot find a set of weights that achieves small error in both approximations 3.1 and 3.2. In conclusion, we see the best overall performance from CCI in the SKI framework.

In the proposed low-rank approximation framework, we thus continue to use CCI to approximate the kernel.

3.3 Determination of Grid Points

In the previous section, we selected the CCI in order to achieve high-accuracy interpolation with a limited amount of grid points. In this section, we will discuss the mechanism to determine the number of grid points needed to reach a certain level of accuracy according to the length scale in every iteration. This is the major difference between the proposed approximation method with the standard KISSGP, as it adapts to the kernel function during learning. The content will be divided into two parts. First, a formal definition of the *density* will be given. Then, grid points with a proper *density* will be chosen according to their impact on time and accuracy.

3.3.1 Density

The intuition for finding the number of grid points comes from the observation that the sparsity of grid points that affects the accuracy of interpolation methods is not only determined by the distance between grid points but also influenced by the length scale of the kernel, as depicted in the following figure:

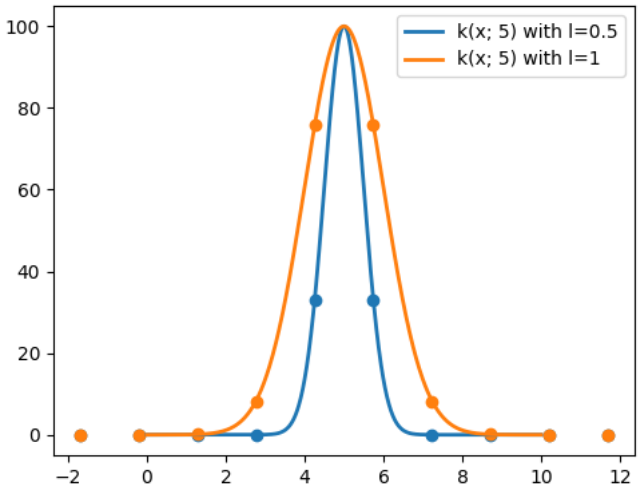


Figure 3.7: Two RBF kernels with $l = 0.5$ and $l = 1$ and same grid point locations. We see that the grid points look denser and provide more information when l is larger.

In Figure 3.7 the number and location of grid points are unchanged, but this set of grid points is denser for the kernel with $l = 1$ than the kernel with $l = 0.5$. Thus, the accuracy of the CCI is influenced by both the distance between nearby grid points and the length scale of the RBF kernel. However, we want to use a single metric *density* ρ that incorporates the impact of both values and uniquely maps to an accuracy regardless of the change of any individual factor. In other words, the design of ρ should meet the following properties:

1. ρ is a function of the distance between nearby grid points $d_u = u_{i+1} - u_i$ and the length scale of the RBF kernel l.
2. The accuracy of interpolation is a function of ρ. The change of d_u or l does not affect accuracy as long as ρ remains unchanged.

This leads to our design of metric ρ simply as:

$$\rho = \frac{l}{d_u}. \tag{3.18}$$

Let us prove that this function for ρ has the second property. Consider the task of interpolating the function $k(x; a)$. It is sufficient to prove property 2 that for any length scale l, and a fixed density ρ, the function value $k(x; a)$ and the interpolated value $g(x; a)$ remain unchanged for all x, given the same grid placement.

Suppose at length scale l_i, we want to interpolate the kernel value $k(x_i; a)$ at x_i where the distance from x_i to a is $x_i - a = s_{xa} d_u = s_{xa} \frac{l}{\rho}$. s_{xa} is the relative distance from x_i to a with the similar definition in CCI. Note that it is valid to represent a point x_i by its relative distance s_{xa} since they are bijective when ρ and l are specified. Similarly, by saying the same grid placement, we imply that the first left grid point of x_i, which we denote as $u_{i,2}$, has the distance $x_i - u_{i,2} = s_{xu} d_u = s_{xu} \frac{l}{\rho}$, where s_{xu} is the relative distance from x_i to $u_{i,2}$. We now prove that the function value $k(x_i; a)$ and the interpolated value $g(x_i; a)$ are only functions of x_i's relative distance s_{xa}, $u_{i,2}$'s relative distance s_{xu} (rigorously speaking, $s_{xu} - s_{xa}$) and density ρ.

The kernel value $k(x_i; a)$ is calculated as:

$$k(x_i; a) = \sigma_s^2 \exp \frac{(x_i - a)^2}{2l^2} = \sigma_s^2 \exp \frac{(s_{xa} \frac{l}{\rho})^2}{2l^2} = \sigma_s^2 \exp \frac{s_{xa}^2}{2\rho^2}. \tag{3.19}$$

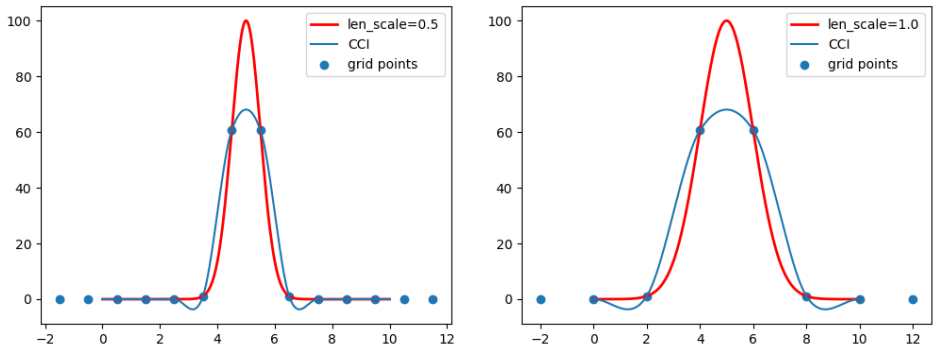
We see that $k(x_i; a)$ is not a function of l.

Since the weights of the CCI is a function of the relative distance s as in 3.5, and the relative distance between grid points is 1 by definition, the assumption that the relative distance between x_i and the left nearest grid point is s_{xu} fixes all the weights as $\mathbf{w}_i = [0 \dots w(s_{xu} - 1)\ w(s_{xu})\ w(s_{xu} + 1)\ w(s_{xu} + 2) \dots 0]^T$. The kernel value on the grid points, on the other hand, is calculated by the RBF kernel. Take the left-most considered grid point $u_{i,1}$ as an example:

$$k(u_{i,1}; a) = k(x_i - (x_i - u_{i,2}) - d_u); a) = \sigma_s^2 \exp \frac{(s_{xa} - s_{xu} - 1)^2}{2\rho^2}. \tag{3.20}$$

With the same derivation, we find that $k(u_{i,j}, a) = \sigma_s^2 \exp \frac{[s_{xa} - s_{xu} + (j-2)]^2}{2\rho^2}$ are not functions of l. Thus, $g(x_i, a) = \sum_{j=1}^{4} w_j k(u_{i,j}, a)$ remains unchanged as ρ is fixed.

Thus, we prove that 3.18 satisfies property 2. Another way to interpret the statement is that the kernel values and interpolated values maintain the same shape as ρ changes. The following plots give an illustrative explanation:



(a) $l = 0.5$ (b) $l = 1.0$

Figure 3.8: Comparison of interpolation result with $\rho = 1$. a) Using 16 grid points. b) Using 8 grid points. It can be observed that the red curves and blue curves are "stretched" along the x-axis.

The above results are acquired by fixing $\rho = 1$ to approximate RBF kernels with $l = 0.5$ and $l = 1$. From the plots, we observe that both the true kernel value distribution and the interpolated value distribution are stretched along the x-axis, but the values remain unchanged.

In summary, the criteria for accuracy can be uniquely controlled by the new parameter: density ρ. Moreover, with ρ we can specify the number of grid points needed at length scale l to reach a certain level of accuracy by adjusting the distance between grid points $d_u = l/\rho$.

3.3.2 Density Driven Grid Point Determination

Previously, we have discussed that density ρ has a direct impact on the accuracy of the interpolation. ρ also directly influences the time complexity of the system, since in every iteration where a specific length scale is decided, the number of grid points m is proportional to ρ, and the time complexity of KISSGP is $\mathcal{O}(n + m^2)$ which could be dominated by m when m is comparable with n. Thus, the balance between accuracy and time consumption can be tuned with this single parameter ρ.

We now draw the RMSE vs. density plot by using different density values to interpolate $k(x; a)$. The RMSE will be taken with respect to the difference between the interpolated function values and the true kernel values at 1000 points. It is worth noticing that the result is a **theoretical** curve since changing either l, d_u, or a does not change the difference of function values at any point as long as density ρ is fixed and the 1000 points are at the same relative distances (i.e., same s_{xa}).

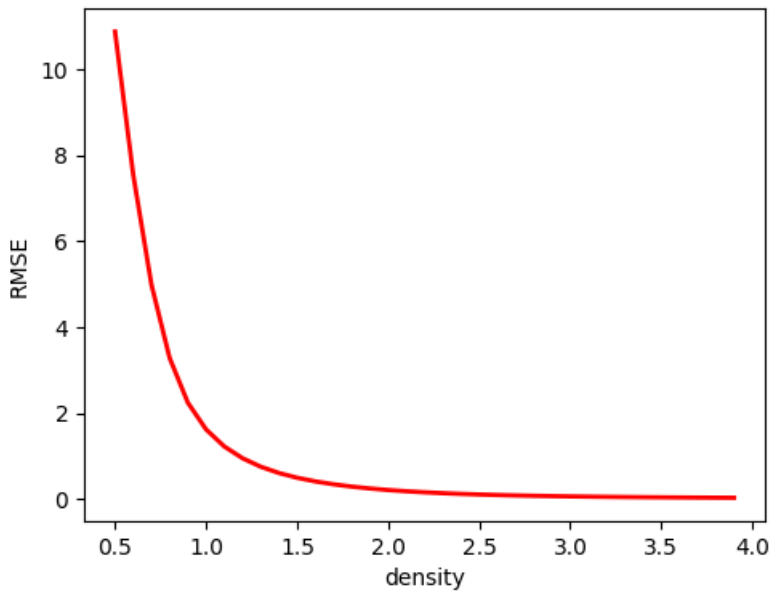


Figure 3.9: density vs. RMSE for RBF kernel reconstruction. When changing l, du, and a with a fixed density, the RMSE value remains unchanged at the same relative positions. Thus this curve can be viewed as a theoretical curve for using CCI to interpolate an RBF kernel.

The choice of density depends on requirements for accuracy and time consumption. Moreover, the characteristics of the accuracy and time consumption of the whole MKISSGP framework when approximating GPR are not exactly the same as in this fundamental interpolation task. Further experiments on how to decide the best density will be given in Section 4.2.

3.4 Proposed Approximation Method

The proposed approximation method is based on KISSGP. However, in our framework, the grid points are updated in every iteration according to ρ to adapt to the change in the length scale l. This gives rise to its name: MKISSGP. This framework is currently specifically tailored to cope with RBF kernels due to the density mechanism. For other kernels, one might have similar parameters as the density to adjust the grid points.

The overall training framework is given in Figure 3.10.

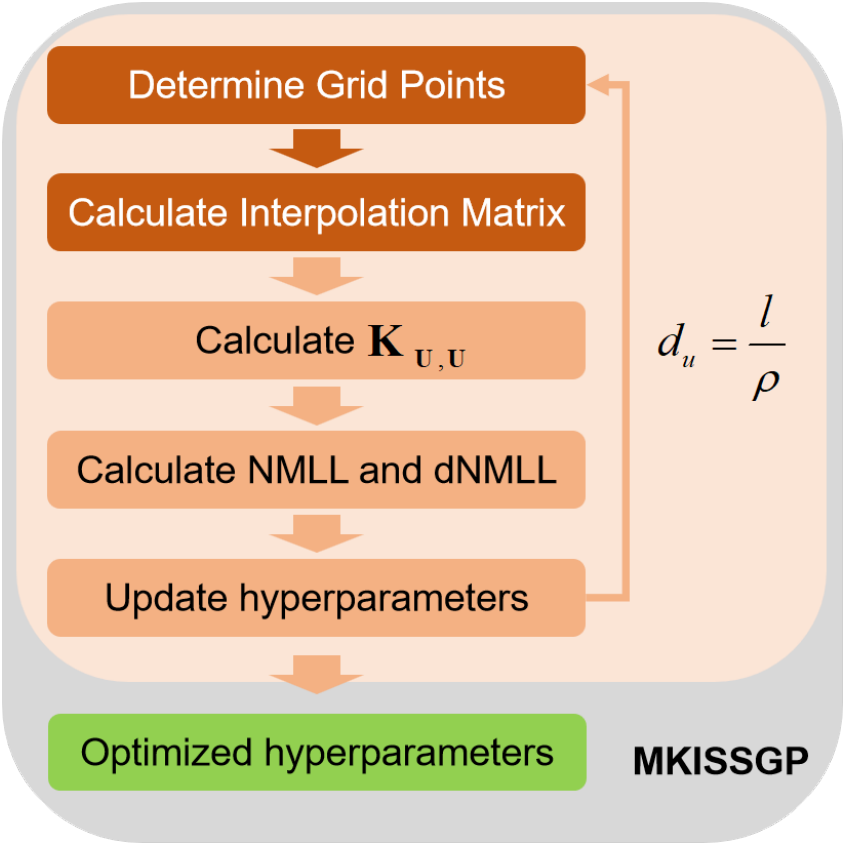


Figure 3.10: Training framework of MKISSGP. The two dark orange blocks were previously outside the optimization loop in the framework of KISSGP. Since the number of grid points is updated according to the length scale, these two steps should be recalculated at the beginning of every loop. The rest of the framework remains the same as in Figure 2.1.

In Figure 3.10, the determination of grid points and the calculation of the interpolation matrix $\mathbf{W}$, which is depicted as dark orange blocks, are moved inside the optimiza-

tion loop compared with KISSGP. This means that the grid points are changed every iteration according to the new length scale, thus we name our method: MKISSGP. The calculation of the NMLL is actually unnecessary for training purpose. However, it is often calculated alongside its derivative due to the fact that they share nearly the same computational resource and for the purpose of logging. The pseudocode of the training algorithm is as follows:

Algorithm 4 Training Process of Mallaeble Kernel Interpolation for Scalable Structured Gaussian Process (MKISSGP)

Input: Training set ($\mathbf{X}$, $\mathbf{y}$) density ρ, initial guess of hyperparameters $\boldsymbol{\theta}_0$, max iteration k_{max}, and convergence threshold ϵ.

Output: The optimized hyperparameters $\boldsymbol{\theta}_{opt}$.

1: Determine grid points according to ρ and $\boldsymbol{\theta}_0$
2: Calculate the initial interpolation matrix $\mathbf{W}$
3: Calculate the initial grid kernel matrix $\mathbf{K}_{\mathbf{U},\mathbf{U}}$
4: $\mathbf{r}_0 \leftarrow \frac{\partial \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}_0)}{\partial \boldsymbol{\theta}_0}$ using $\mathbf{W}$ and $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ ▷ the nonlinear CG approach
5: **if** $\|\mathbf{r}_0\| < \epsilon$ **then return** $\boldsymbol{\theta}_0$
6: **end if**
7: $\mathbf{p}_0 \leftarrow -\mathbf{r}_0$
8: **for** $k = 1$ to k_{max} **do**
9: $\quad \alpha_k \leftarrow \alpha$ from Wolfe line search
10: $\quad \boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k$
11: $\quad$ Determine grid points according to ρ and $\boldsymbol{\theta}_{k+1}$
12: $\quad$ Update the interpolation matrix $\mathbf{W}$
13: $\quad$ Update the grid kernel matrix $\mathbf{K}_{\mathbf{U},\mathbf{U}}$
14: $\quad \mathbf{r}_{k+1} \leftarrow \frac{\partial \log p(\mathbf{y}|\mathbf{X},\boldsymbol{\theta}_{k+1})}{\partial \boldsymbol{\theta}_{k+1}}$ using $\mathbf{W}$ and $\mathbf{K}_{\mathbf{U},\mathbf{U}}$
15: $\quad$ **if** $\|\mathbf{r}_{k+1}\| < \epsilon$ **then return** $\boldsymbol{\theta}_{k+1}$
16: $\quad$ **end if**
17: $\quad \beta_k \leftarrow \frac{\mathbf{r}_{k+1}^T(\mathbf{r}_{k+1}-\mathbf{r}_k)}{\mathbf{r}_k^T \mathbf{r}_k}$
18: $\quad \mathbf{p}_{k+1} \leftarrow \beta_k \mathbf{p}_k - \mathbf{r}_{k+1}$
19: **end for**

A detailed process of every step in the algorithm that are not directly part of the nonlinear CG (i.e., line 1 to 4 and 11 to 14) is given as follows:

- Determine grid points

  Corresponds to line 1 and 11 in algorithm 4.

  In this step, we have the information on the training input locations $\mathbf{X}$ and the length scale $\mathbf{l}$ for every dimension of $\mathbf{X}$. For every dimension, from l, combined with the pre-determined ρ, we acquire the distance between grid points as $d_u = \frac{l}{\rho}$. Then to guarantee that all training inputs are effectively covered in this dimension. We use $m - 2$ grid points with distance d_u to cover the furthest training points and assign 2 grid points at the beginning and the end with the same distance to ensure that the CCI can function effectively.

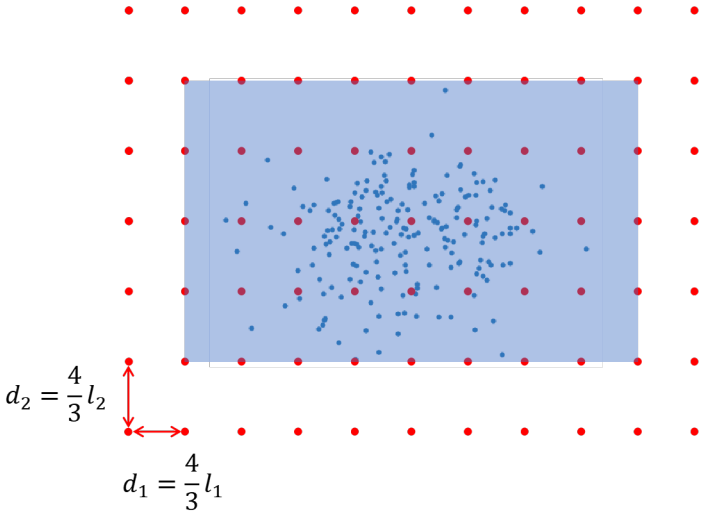


Figure 3.11: Example of a 2-D grid. The blue dots are the training points. The red dots are the 2-D grid points. The blue rectangle is the area that covers all training points. There is an additional padding of grid points around the blue rectangle so that the CCI is applicable at all training points.

Figure 3.11 is a 2-D example of the grid point determination result. The area in blue contains all training points, depicted in blue dots. The grid points are depicted in red dots. Additional padding of grids is outside the blue area to ensure the implementation of the CCI. The gap between grid points in each dimension is determined by $\frac{1}{\rho}$ of the corresponding length scale of the kernel functions. A detailed analysis of the time complexity of this step and the next step will be given in Section 3.4.1.

- Calculate interpolation matrix $\mathbf{W}$.

  Corresponds to line 2 and 12 in algorithm 4.

  Because the grid points now change with the optimization, the interpolation matrix $\mathbf{W}$ should also be recalculated in every iteration. The calculation mechanism is the same as in standard KISSGP. The interpolation weights are first calculated for each dimension, then they are combined by multiplication to form full weights in the multi-dimensional space.

  Also taking a 2-D example, in Figure 3.12, the blue training point $\mathbf{x} = (x_1, x_2)$ will be interpolated using 16 red grid points. In the first dimension, from the distance between x_1 to $u_{11}, u_{12}, u_{13}, u_{14}$, we calculate the weights $w_{11}, w_{12}, w_{13}, w_{14}$. The same happens in the second dimension, we acquire the weights $w_{21}, w_{22}, w_{23}, w_{24}$. Finally, we multiply the corresponding weights together to get the full weights of a grid point. The weight for the grid point (u_{11}, u_{21}) is $w_{11}w_{21}$, the weight for (u_{11}, u_{24}) is $w_{11}w_{24}$, for (u_{14}, u_{21}) is $w_{14}w_{21}$.

  For the 1-d problem, the weights are directly acquired without the multiplication step.

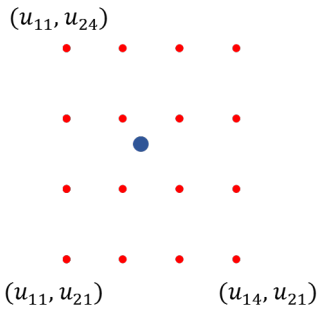


Figure 3.12: 2-D Interpolation Weights Calculation. The CCI for one point requires 4 grid points in 1 dimension. Thus 16 grid points in total are needed for 2-D.

- Calculate $\mathbf{K}_{\mathbf{U},\mathbf{U}}$.

  Corresponds to line 3 and 13 in algorithm 4.

  From this step on, the KISSGP and MKISSGP share the same procedure. The calculation of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ is also conducted in each dimension. Then, the grid kernel matrix factors in each dimension are stored, instead of calculating the Kronecker product. This is because later computations involving the MVM of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ can be facilitated using Kronecker algebra or Toeplitz algebra as shown in Section 2.4.1.

- Calculate dNMLL (and NMLL).

  Corresponds to line 4 and 14 in algorithm 4.

  Since the MKISSGP has the same form of NMLL function as KISSGP in 2.53, we calculate dNMLL of KISSGP as:

$$\frac{\partial - \log p(\mathbf{y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} = \frac{1}{2}\left(-(\mathbf{W}^T\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y})^T\frac{\partial \mathbf{K}_{\mathbf{U},\mathbf{U}}}{\partial \boldsymbol{\theta}_i}(\mathbf{W}^T\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y}) + \frac{\partial \log\det(\tilde{\mathbf{K}}_{\mathbf{n}})}{\partial \boldsymbol{\theta}_i}\right), \tag{3.21}$$

  where $\tilde{\mathbf{K}}_{\mathbf{n}} = \mathbf{K}_{SKI} + \sigma_n^2\mathbf{I} = \mathbf{W}\mathbf{K}_{\mathbf{U},\mathbf{U}}\mathbf{W}^T + \sigma_n^2\mathbf{I}$.

  The first term in the major bracket can be decomposed into several calculations:

  1. $\mathbf{K}_{\mathbf{n}}^{-1}\mathbf{y}$
     This term can be calculated by the linear CG algorithm, leveraging the Toeplitz structure and possibly the Kronecker structure in $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ to facilitate MVM calculations.
  2. $\mathbf{W}^T(\mathbf{K}_{\mathbf{n}}^{-1}\mathbf{y})$
     This step is the MVM of a sparse matrix and the product that we had in the previous step.
  3. $\frac{\partial \mathbf{K}_{\mathbf{U},\mathbf{U}}}{\partial \boldsymbol{\theta}_i}(\mathbf{W}^T\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y})$
     Since the derivative of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ also has the same structure as $\mathbf{K}_{\mathbf{U},\mathbf{U}}$, this step can also be facilitated by exploiting matrix structures and using linear CG.

4. $-(\mathbf{W}^T\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y})^T\frac{\partial \mathbf{K}_{\mathbf{U},\mathbf{U}}}{\partial \boldsymbol{\theta}_i}(\mathbf{W}^T\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y})$
   This step is just the multiplication of two vectors acquired from step 2 and step 3.

The derivative of the log-determinant is calculated differently according to the estimation we use as mentioned in the previous section. When using the Lanczos method, the derivative is calculated by lines 8-12 in algorithm 1. When using the eigen method (see Section 2.4.1), the derivative is calculated according to 3.23 by:

$$\begin{aligned}\frac{\partial \log\det(\tilde{\mathbf{K}}_{\mathbf{n}})}{\partial \boldsymbol{\theta}_i} &= \frac{n}{m}\sum_{j=1}^{n}\frac{\frac{\partial \lambda_j}{\partial \boldsymbol{\theta}_i}}{\frac{n}{m}\lambda_j+\sigma_n^2} \\ &= \frac{n}{m}\begin{bmatrix}\mathbf{v}_1^T\frac{\partial \mathbf{K}_{\mathbf{U},\mathbf{U}}}{\partial \boldsymbol{\theta}_i}\mathbf{v}_1 & \dots \mathbf{v}_n^T\frac{\partial \mathbf{K}_{\mathbf{U},\mathbf{U}}}{\partial \boldsymbol{\theta}_i}\mathbf{v}_n\end{bmatrix}\left(\frac{1}{\frac{n}{m}\boldsymbol{\lambda}+\sigma_n^2}\right),\end{aligned} \tag{3.22}$$

where $\left(\frac{\partial \boldsymbol{\lambda}}{\partial \boldsymbol{\theta}_i}\right)$ us the short-hand notation for the vector $[\frac{\partial \lambda_1}{\partial \boldsymbol{\theta}_i}\ \frac{\partial \lambda_2}{\partial \boldsymbol{\theta}_i}\ \dots \frac{\partial \lambda_n}{\partial \boldsymbol{\theta}_i}]^T$, $1/(\frac{n}{m}\boldsymbol{\lambda}+\sigma_n^2)$ has similar meanings. $\mathbf{v}_n$ is the eigenvector corresponding to the n-th largest eigenvalue λ_n. When $m < n$, $\mathbf{v}_{m+1}$ to $\mathbf{v}_n$ are taken as $\mathbf{0}$. The eigenvalues and eigenvectors could be calculated efficiently leveraging Kronecker algebra when the dimension is higher than 1. Now we have complete knowledge of the calculation of dNMLL of KISSGP.

On the other hand, the dNMLL of MKISSGP is exactly the same as above, because there is no change to the representation of the kernel matrix. One may argue that the interpolation matrix $\mathbf{W}$ also changes as the length scale changes, thus $\frac{\partial \mathbf{W}}{\partial \boldsymbol{\theta}_i}$ should be considered. However, according to the definition of derivatives, we take very small changes in the input dx and evaluate the proportion of function change against this small shift of input $\frac{df(x)}{dx}$. In our scenario, a very small change in the length scale dl will not cause the interpolation matrix $\mathbf{W}$ to change, i.e., $d\mathbf{W} = \mathbf{0}$. This is because when calculating the number of grid points in each dimension, we evaluate the integer part of $(x_{max} - x_{min})/d_u$, where $x_{max} - x_{min}$ is the span of data points. Though $d(d_u) = \frac{dl}{\rho}$, the number of grid points doesn't change. We only recalculate the interpolation matrix $\mathbf{W}$ when the number of grid points changes, thus we have $\frac{\partial \mathbf{W}}{\partial dl} = \mathbf{0}$.

In summary, the derivative calculation of MKISSGP is the same as KISSGP, which is given in 3.21. Calculations of the derivative of the log-determinant is either done by lines 8-12 in algorithm 1 or by 3.22, depending on the method of log-determinant estimation.

The time spent in the calculation of the NMLL time spent in this step is mainly devoted to the calculation of $\mathbf{y}\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y}$ and $\log\det(\tilde{\mathbf{K}}_{\mathbf{n}})$. The first term is calculated using similar approaches as in calculating the dNMLL.

The log-determinant term is approximated by either the formula below:

$$\log\det(\tilde{\mathbf{K}}_{\mathbf{n}}) \approx \sum_{i=1}^{n} \log\left(\frac{n}{m}\lambda_i + \sigma_n^2\right), \tag{3.23}$$

where λ_i are the n largest eigenvalues of $\mathbf{K}_{\mathbf{U},\mathbf{U}}$, or using the Lanczos method for estimating log-determinants.

In our method, we will use the eigen method to estimate the log-determinant term and calculate its derivative. Reasons for this selection will be presented in Section 3.4.2.

3.4.1 Additional Time Complexity Terms of MKISSGP

Compared with the original KISSGP, MKISSGP introduced the procedure of grid point determination and interpolation matrix calculation into the iteration loop, which theoretically introduces additional time complexity terms. We will analyze their impact in this section and show that they are trivial compared with other terms.

First, the calculation of grid points takes $\mathcal{O}(m_i)$ time to evaluate the position of the m_i grid points in dimension i. Thus, for all dimensions, the total time complexity would be $\mathcal{O}(\sum_{d=1}^{D} m_d)$. Due to the fact that in the SKI framework, we assume a multiplicative kernel (see Section 2.4), and the CCI will require at least 4 grid points per dimension, $\mathcal{O}(\sum_{d=1}^{D} m_d) \leq \mathcal{O}(\prod_{d=1}^{D} m_d) = \mathcal{O}(m)$. Equality is satisfied when the number of dimensions is 1.

Second, the calculation of the interpolation matrix also processes per every dimension. For each dimension, we needed to calculate the required grid points and the corresponding weights for the training data, which both take $\mathcal{O}(n)$ time. It seems daunting to introduce this procedure into the iteration loop. However, thanks to parallel processing in the NumPy package [49] of Python, the actual computation time could take much less than $\mathcal{O}(n)$ when processing the array of training data in parallel. Similar mechanisms are also available in other languages, e.g., Matlab builtin optimized matrix operations, the OpenMP package for C, C++, and Fortran.

Furthermore, we tested the average time spent on the important steps in one loop when performing GPR with MKISSGP with density $\rho = 2$. We use the same experimental context as in comparing the existing approximation methods in Section 2.5.

Step	time (ms)
Grid	0.07
$\mathbf{K}_{\mathbf{U},\mathbf{U}}$	0.18
$\mathbf{W}$	0.34
$\log\det(\tilde{\mathbf{K}}_{\mathbf{n}})$	5.62
$\tilde{\mathbf{K}}_{\mathbf{n}}^{-1}\mathbf{y}$	6.95

Table 3.5: The average time spent on the calculation of different terms in MKISSGP

In Table 3.5, we see in green that the additional time introduced by the procedure of MKISSGP is trivial compared with the calculation of the log-determinant and the inverse of the approximated kernel matrix.

3.4.2 Selection of Log-determinant Estimation Methods

In the original KISSGP paper, the author used approximation 3.23 to estimate the log-determinant term [33], we now refer to it as the eigen method. More recent works of structural approximations [24][32][6][23] implemented the Lanczos method for log-determinant estimation (algorithm 1) to leverage fast MVM in iterative calculations. It seems that the latter version of estimation is superior to the former. However, in our experiments, we find the former method is a more efficient approach than the latter.

With the similar settings in the previous section, we tested the average error for both methods. Moreover, to make them comparable, we tuned the degree of Lanczos to 20 and the number of probe vectors to 5 of the Lanczos method to achieve nearly the same computational time for both methods (7.08ms for the "eigen" method and 7.03ms for the Lanczos method). This choice of parameter combination is based on the statement in [4] that it takes only a few probe vectors for the Lanczos method to converge.

Term	Method	SMAE
$logdet$	SKI	0.004
	Eigen	0.016
	Lanczos	0.251
$\mathrm{d}logdet/\mathrm{d}s_0$	Eigen	0.053
	Lanczos	5.525
$\mathrm{d}logdet/\mathrm{d}l$	Eigen	0.096
	Lanczos	10.868

Table 3.6: Average error of different terms in log-determinant estimation from the eigen method and Lanczos method

In Table 3.6, we measure the error between two variables using the standard mean absolute error (SMAE). The choice of using this metric instead of the previously used RMSE is that the log-determinant and its derivatives are just individual scalars. Moreover, the impact of this error should be standardized to depict its impact on the optimization process. (e.g., the error between 1000 and 999 and -0.999 and 0.001 are the same when using the RMSE, but they clearly have an entirely different impact on training when they are the values of derivatives.)

The SKI method in the $logdet$ term is the error between $\log\det(\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma_n^2\mathbf{I})$ and $\log\det(\mathbf{K}_{SKI} + \sigma_n^2\mathbf{I})$. This error is very small, meaning that theoretically, the SKI approximates the kernel matrix very well. Other error terms in this table are all measured using the corresponding precise results of SKI as ground truth.

The comparison of the $logdet$ term between the eigen method and the Lanczos method shows that the former has much higher accuracy. This superiority is even further exaggerated when viewing the derivative terms, where the SMAE of the Lanczos method is more than 100 times larger than the eigen method. In practice, this immense error from the Lanczos method provides incorrect derivative information to the optimizer which leads to failed optimization at a very high chance. On the other hand, if we manage to push the Lanczos method to the same level of accuracy as the eigen

method, which requires Lanczos degree at 80 and the number of probe vectors at 15, it would take 180ms to perform one such estimation. This is unacceptable compared with only 7.03ms required from the eigen method.

Thus, we conclude that the eigen method shows its merit in at least our experimental scenario, thus we re-implement this method to approximate the log-determinant term of SKI in MKISSGP. However, we reserve the freedom of using the Lanczos method as an alternative.

3.5 Summary of the Proposed Approximation Method

In this chapter, we developed our improved approximation method based on KISSGP. We develop our method first assuming using the RBF kernel to model training data. Since we focus on limiting the number of grid points per every dimension, we use 1-D data to experiment. In KISSGP, the core assumption is that the training kernel matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ can be interpolated via the grid kernel matrix $\mathbf{K}_{\mathbf{U},\mathbf{U}}$. This breaks down into two sets of interpolations:

$$k(x_i, u) \approx \sum_i w_i k(u_i, u), \tag{3.24a}$$

$$k(x_i, x) \approx \sum_i w_i k(u_i, x). \tag{3.24b}$$

3.24a means that for any specific training point x_i, there should be a set of weights w_i that can be used to interpolate the kernel function $k(x_i, u)$ with u as any grid point. 3.24b has similar implications.

The accuracy of these interpolations is mainly controlled by two factors:

- The interpolation method used to interpolate the kernel function.
- The density of the grid points.

And the time complexity of the complete approximation algorithm (i.e., $\mathcal{O}(n+m^2)$) is affected by the number of grid points in total (i.e., m). Thus, in order to limit m, we have to choose the most accurate interpolation method that works with relatively sparse grid points, and then choose the necessary amount of grid points that allow the interpolation to reach a certain level of accuracy.

For the interpolation scheme, we explored the following methods. Based on the different drawbacks of the methods, we eventually choose the CCI as the final interpolation method.

- Cubic convolutional interpolation (CCI): this is the default method of KISSGP. The only information it exploits is the kernel values at the grid point location. However, it is able to accurately approximate any quadratic function.
- Exponential convolutional interpolation (ECCI): this method combines the exponential interpolation and the CCI. It leverages the fact that the RBF kernel is an exponential function with a quadratic term as the exponent. It can fully recover

the RBF kernel, but it fails to produce a low-rank decomposition for the kernel matrix.

- Dutch Taylor expansion (DTE) interpolation: this method implements a modified version of the Taylor expansion, the Dutch Taylor expansion, to perform the interpolation. The level of accuracy is controlled by the order of derivatives used. Since we assume the RBF kernel, theoretically we can recover the kernel function by leveraging its infinite orders of derivatives. However, this method hardly fits in the SKI framework due to that the calculated interpolation coefficients are only suitable for a specific pair of inputs of the RBF kernel, but not for the general case when u and x can be any grid point or training point in 3.24a and 3.24b.

- Optimal interpolation: this method acquires the weight through optimization. It exploits the fact that all the kernel values between training points and grid points are known, and thus we optimize the weight by solving a least-square problem. However, the time complexity to acquire the weights is too high ($\mathcal{O}((n+m)^2)$) compared with the CCI ($\mathcal{O}(1)$).

For the number of grid points, we defined a new metric *density* $\rho = \frac{l}{d_u}$, where l is the length scale parameter in the RBF kernel, and d_u is the distance between grid points. We proved that the accuracy for any interpolation method to interpolate the RBF kernel is specified by ρ. In other words, in our scenario, a fixed ρ leads to a fixed accuracy of CCI to interpolate the RBF kernel regardless of the change of length scale l. We can then calculate the necessary number of grid points in every iteration using the new length scale l and the fixed density ρ.

The new approximation method: malleable kernel interpolation for scalable structured Gaussian process (MKISSGP) is thus developed. It builds upon the current KISSGP structure but re-determines the grid points and re-calculates the interpolation matrix $\mathbf{W}$ in every iteration. The additional time introduced per iteration is acceptable due to the presence of parallel computation mechanisms. In addition, the calculation of derivatives with respect to the hyperparameters remains unchanged from the original KISSGP.

4 Experiments

In this chapter, we will test the full capacity of malleable kernel interpolation for scalable structured Gaussian process (MKISSGP) in performing the approximated GPR. To demonstrate the contribution of this thesis, we will conduct experiments in only one dimension to showcase that MKISSGP can maintain the same level of accuracy while using potentially fewer grid points than kernel interpolation for scalable structured Gaussian process (KISSGP). All experiments were run on a 2.30GHz Intel Core i7-11800H CPU.

The following experiments will be conducted:

1. Kernel reconstruction experiment.

   In Section 4.1, the experiment aims to prove that the error level of reconstructing the kernel matrix $\mathbf{K}_{\mathbf{X},\mathbf{X}}$ remains unchanged when density is fixed using MKISSGP, and decreases when using a higher density.

2. Recommended density experiment.

   In Section 4.2, the experiment aims to find a default density for the MKISSGP that balances time and accuracy. However, one can always choose different densities depending on the actual requirements.

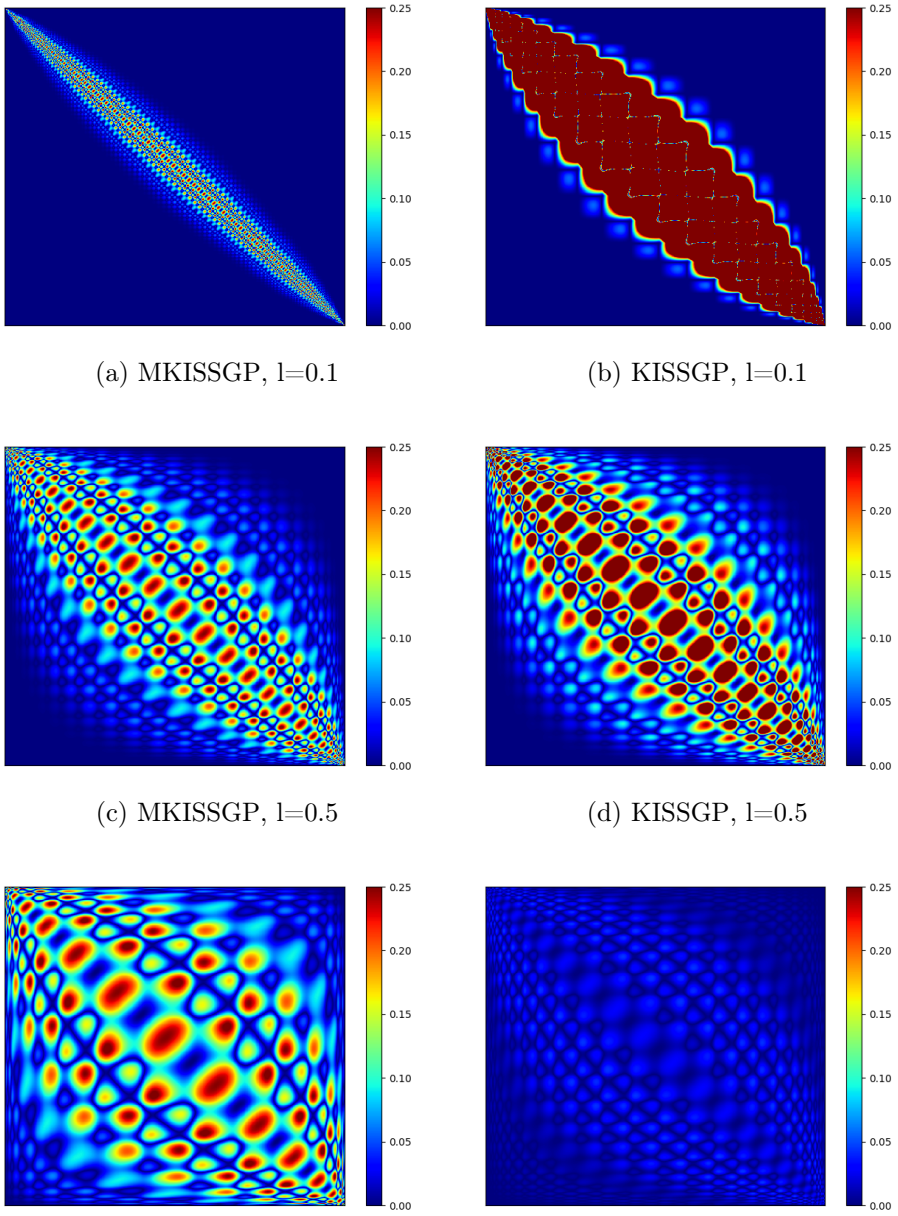
3. Function reconstruction experiment.

   In Section 4.3, the experiment aims to compare MKISSGP with KISSGP with respect to the approximation accuracy and time spent on NMLL and dNMLL calculation to show that MKISSGP requires less time to reach a desired level of accuracy than KISSGP.

4.1 Kernel Reconstruction Experiment

The primary consequence of any low-rank approximation lies in the restoration of the original kernel matrix for training inputs through a kernel matrix of reduced rank. In this experiment, we assess the stability of a MKISSGP with a constant density in reconstructing kernel matrices with varying length scales. For comparison, we employ a KISSGP model with a fixed number of grid points to undertake the same kernel matrix reconstructions.

We generate 3 kernel matrices $\mathbf{K}_{0.1}$, $\mathbf{K}_{0.5}$, $\mathbf{K}_{1.0}$ from 10000 random points drawn from a standard normal distribution while using RBF kernels with $l = 0.1$, $l = 0.5$, and $l = 1.0$, respectively. Then we approximated kernels using MKISSGP with density $\rho = 2.7$ (this choice of density is the result of the next experiment, see Section 4.2) and KISSGP with the number of grid points equivalent to MKISSGP in the $l = 0.5$ case. The results are shown in Figure 4.1.



(a) MKISSGP, l=0.1

(b) KISSGP, l=0.1

(c) MKISSGP, l=0.5

(d) KISSGP, l=0.5

(e) MKISSGP, l=1.0

(f) KISSGP, l=1.0

Figure 4.1: The absolute value of differences between the true kernel matrix and the approximated kernel matrix. The color bar only has its limit at 0.25 to cope with MKISSGP results. **Values higher than 0.25 are also colored dark red.** a), c), and e) are results of using MKISSGP with a $\rho = 2.7$. we see that the reconstruction error is stable. b), d), and f) are results of using KISSGP with the same number of grid points as in c). Clearly, the accuracy fluctuates drastically along with the change in length scale.

The color bar in Figure 4.1 spans the range of $[0, 0.25]$ to encompass the entirety of MKISSGP errors, with a slight surplus. On the left-hand side, Figure 4.1a, Figure 4.1c, and Figure 4.1e illustrate the impact of fixing density. While the length

scale varies, the kernel matrix reconstruction error remains relatively stable within a certain range. In contrast, the right-hand side, Figure 4.1b, Figure 4.1d, and Figure 4.1f, demonstrates significant fluctuations in the reconstruction error of KISSGP as the length scale changes. These results on the right-hand side can also be interpreted as indicative of an increase in accuracy with greater density. Notably, for $l = 0.5$, the reconstruction error differs between MKISSGP and KISSGP despite having the same number of grid points. This disparity arises from differences in localizing grid point positions between the two methods.

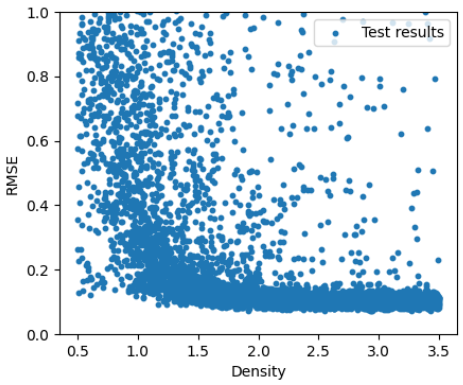
A mistaken assumption can be made from the plots that in Figure 4.1f, KISSGP achieves the lowest error among all cases, suggesting more accurate interpolation compared to MKISSGP. However, it's important to note that this experiment does not aim to establish the maximum accuracy of either method. In practice, one can always employ denser grids to achieve higher accuracy, regardless of the computational time. The primary objective of this experiment is to demonstrate that MKISSGP can maintain a predefined level of accuracy in kernel matrix reconstruction while using a minimal number of grid points, regardless of varying length scales. This is a capability that KISSGP cannot offer.

In conclusion, this experiment reveals that in MKISSGP, density serves as a crucial factor in controlling the reconstruction error of the kernel matrix. By choosing higher densities, one can enhance the accuracy of kernel matrix reconstruction.

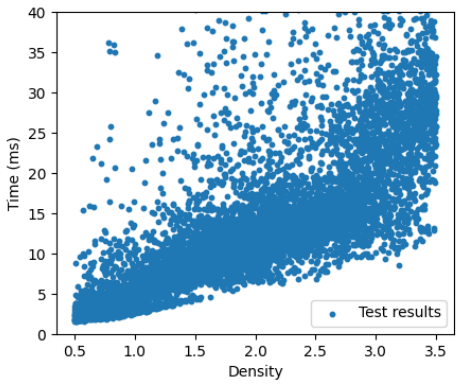
4.2 Recommended Density Experiment

In this experiment, our objective is to determine the optimal default density for MKISSGP, one that is most likely to achieve high accuracy within a short timeframe across various scenarios.

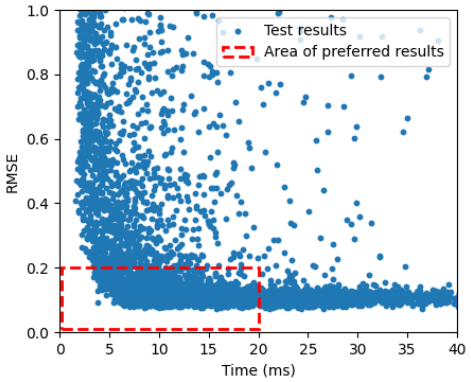
The experiment comprises 8,000 Monte Carlo trials, each characterized by specific parameters. In every trial, we generate 1,000 noisy training points (with $\sigma_n^2 = 0.25$) from a randomly sampled 1-dimensional function governed by a GP using a radial basis function (RBF) kernel. The signal power is drawn from a uniform distribution ranging between 1.0 and 10.0, while the length scale is selected from a uniform distribution within the logarithmic domain spanning from 0.1 to 20.0. Subsequently, for each trial, a MKISSGP model is constructed with densities drawn from a uniform distribution between 0.5 and 3.5. This model performs an approximate GPR on the training data. The efficiency of the algorithm is gauged by recording the average $t_{\text{NMLL}} + t_{\text{dNMLL}}$ per iteration, along with the final RMSE.



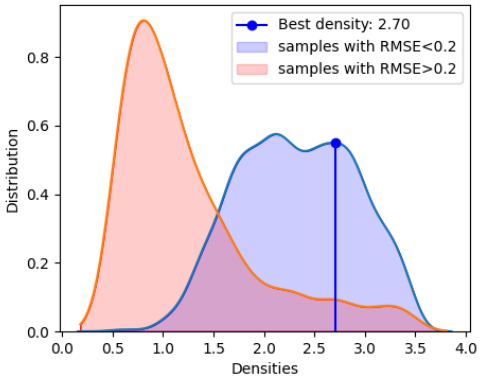
(a) Density vs. RMSE



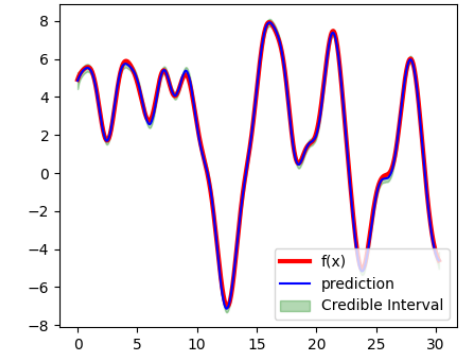
(b) Density vs. Time



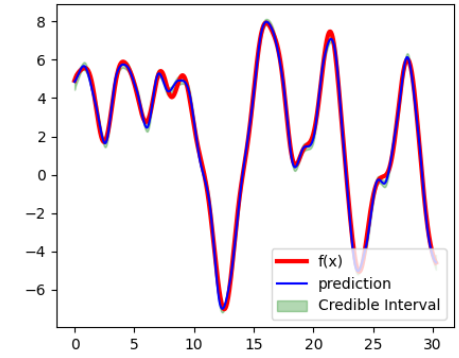
(c) Time vs. RMSE



(d) Distribution of samples with RMSE<0.2 and RMSE>0.2



(e) Prediction Results with RMSE = 0.102



(f) Prediction Results with RMSE = 0.192

Figure 4.2: Recommended density test results. a) RMSE converges with increasing ρ, b) Time generally rises with density; scattered points are attributed to computation variations and hyperparameter initialization. c) RMSE converges with more time; preferred results are indicated by the red box (time $<$ 20ms, RMSE $<$ 0.2). d) The blue KDE plot is plotted from samples with RMSE $<$ 0.2. The red KDE plot is plotted from samples with RMSE $>$ 0.2. The samples are weighted considering error and time. e)-f) Illustrate preferred (e) and non-preferred (f) fits, enforcing an RMSE upper bound of 0.2 in (c).

The figure presented in Figure 4.2a illustrates the distribution of errors across density values. It is intriguing to note that this plot exhibits a similar trend to the curve observed when exclusively considering the reconstruction of the RBF kernel in Figure 3.9. This shows that density plays a significant role in controlling the accuracy of the entire GPR task. Furthermore, as density increases, the covariance of the error distribution diminishes, with only a few outlier samples.

In Figure 4.2b, we examine the distribution of time spent per NMLL and dNMLL calculation with respect to density. Interestingly, the data points exhibit a scattered pattern across the graph, despite the potential presence of a linear lower bound. This scattering of time values can be attributed to the stochastic nature of computational power and variations in hyperparameter initialization during the experimental process.

The relationship between time per iteration and the RMSE is depicted in Figure 4.2c. The data suggests that achieving better results is more likely when there is a higher time allocation per NMLL and dNMLL calculation. This relationship could also be inferred from the patterns observed in the previous two plots. However, the significance of this plot lies in its ability to identify preferred trial outcomes. The region within the red box is considered where the optimal test results are expected to be found. The upper time limit is set at 20ms, as indicated by the graph. Beyond this threshold, minimal improvement in results is observed. Similarly, an upper bound of 0.2 is imposed on the RMSE since errors surpassing this value are considered unacceptable. This is further demonstrated in Figure 4.2e and Figure 4.2f, where the former exhibits a favorable fit, while the latter starts to demonstrate conspicuous errors.

The figure presented in Figure 4.2d showcases the distribution of samples based on their RMSE values. Specifically, it distinguishes between samples with RMSE values less than 0.2 (depicted in blue) and samples with RMSE values greater than 0.2 (depicted in red). This visualization is created using a kernel density estimate (KDE) plot, which is a method for approximating the probability density function (PDF) of the data. It's important to note that the area under the contour in a KDE plot always integrates to 1 and does not indicate the number of samples. In constructing this plot, our primary focus is on accurate approximations, with less weight assigned to the time required.

In essence, the blue curve can be viewed as the PDF of achieving favorable results, while the red curve represents the PDF of obtaining less desirable outcomes, which could be characterized by excessive time consumption or substantial errors.

Our goal is to identify a recommended density value that offers the best likelihood of achieving preferred results while minimizing the chances of experiencing suboptimal outcomes. After a thorough analysis, we have chosen a density value that maximizes the difference between the KDE values, which, as determined, is $\boldsymbol{\rho} = 2.70$. This value is our recommended density setting. However, it's important to acknowledge that individual requirements and the unique characteristics of the data may necessitate adjustments to the chosen density to strike a balance between actual time consumption and accuracy.

It's worth noting that determining the recommended density is not achieved through hypothesis testing based on the two KDE plots. The recommended density is not a threshold used to classify whether a given density will yield good or poor results. Therefore, this question does not fall within the realm of hypothesis testing; rather, it

is a matter of optimizing the density parameter for the best overall performance.

4.3 Function Reconstruction Experiment

Finally, we put the complete capabilities of MKISSGP to the test, evaluating its performance in GPR using the identical dataset outlined in Section 2.5. We proceed to compare the outcomes with those produced by the original KISSGP. In the context of MKISSGP, we conducted tests using three distinct density settings: $\rho = 2.2$, 2.7, and 3.2.

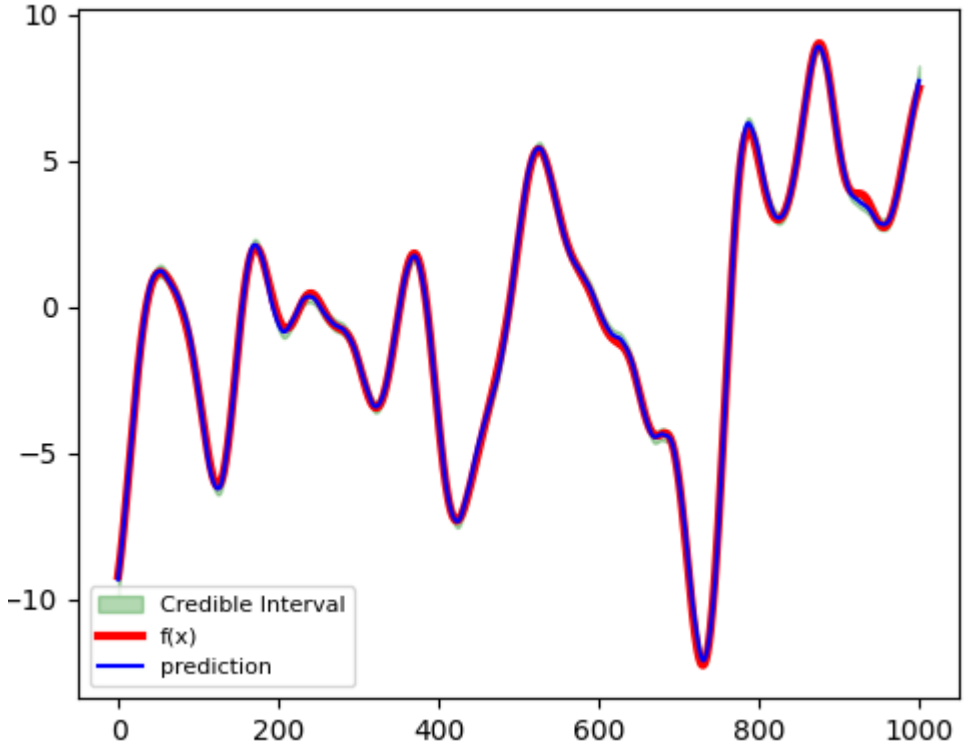


Figure 4.3: Posterior Distribution of MKISSGP with $\rho = 2.7$. The RMSE of this trial is 0.108 which is very close to precise GPR.

Figure 4.3 illustrates the posterior distribution of MKISSGP with the recommended density. We observe that MKISSGP generated a highly accurate approximation of GPR. The following table shows that the computation time of MKISSGP is further reduced from KISSGP.

Method	RMSE	$t_{\text{NMLL}} + t_{\text{dNMLL}}$(ms)	time complexity	density equivalence
GPR	0.108	123.43	$\mathcal{O}(n^3)$	N/A
KISSGP-50	0.250	7.43		1.41
KISSGP-100	0.126	27.13	$\mathcal{O}(n + m^2)$	2.91
KISSGP-200	0.108	46.12		5.91
MKISSGP-2.2	0.148	14.97		2.20
MKISSGP-2.7	**0.111**	**24.07**	$\mathcal{O}(\mathbf{n + m_{min}^2})^*$	2.70
MKISSGP-3.2	0.111	32.00		3.20

Table 4.1: Accuracy and time of KISSGP and MKISSGP

In the table, the $\mathcal{O}(n + m_{min}^2)^*$ means asymptotically reaching the time complexity of $\mathcal{O}(n + m_{min}^2)$. The column "density equivalence" serves as a conversion from the number of grid points in the context of KISSGP to density in the context of MKISSGP using the ground truth length scale $l = 30$ to help compare the two methods.

As evident from Table 4.1, it's apparent that MKISSGP has already attained a level of accuracy on par with GPR when employing the recommended density setting. Furthermore, when compared to KISSGP using 200 grid points, which also exhibits a similar level of accuracy, the time required for both NMLL and dNMLL calculations is approximately halved. This underscores the capability of MKISSGP to significantly expedite GPR by efficiently leveraging information from grid points. In addition, we observe that even though KISSGP with 100 grid points is equivalent to having a density of 2.91 (greater than recommended density), it didn't reach the desired level of accuracy. This is due to the fact that during the initial steps of training, 100 grid points are relatively sparse to acquire accurate estimations of NMLL and dNMLL.

In terms of time complexity, MKISSGP has made further advancements, ultimately achieving an asymptotic time complexity of $\mathcal{O}(n + m_{\text{min}}^2)$. Here, m_{min} represents the minimum number of grid points necessary to reach a specific level of accuracy. This is attributed to the observation that, during nonlinear CG, the changes in the length scale become progressively smaller to the extent that the number of grid points remains unchanged. Consequently, MKISSGP excels in providing both accuracy and efficiency, making it a valuable tool for accelerated GPR applications.

4.4 Summary of Experiments

In our experimental procedures, we initially assessed the capability of MKISSGP to reconstruct the kernel matrix for training points. Our results substantiate the observation that the accuracy of kernel matrix reconstruction is directly correlated with the chosen density. In contrast, KISSGP relies on a fixed number of grid points, which may result in substantial errors in reconstruction or an overabundance of grid points, leading to excessive computational time. In contrast, MKISSGP dynamically adjusts the number of grid points as required to meet specified accuracy criteria.

Having established that density significantly influences the accuracy of kernel approximation, the question arises regarding the optimal choice of density. In our second experiment, we employed the Monte Carlo technique, evaluating a broad range of RBF

kernel instances, to identify a recommended density for MKISSGP. Our prioritization of accuracy over time to a certain extent doesn't block the freedom of choosing different densities based on different time and accuracy requirements.

Finally, we subjected MKISSGP to comprehensive comparative analysis with KISSGP in the same experimental setup, consistently employed throughout the thesis, demonstrating that MKISSGP achieves equivalent levels of accuracy in significantly less time than KISSGP. This validates the efficiency and effectiveness of MKISSGP as a high-performance alternative for various GPR applications.

5 Discussion and Conclusion

In this chapter, we will first discuss the limitations of the current MKISSGP framework. Then, we will discuss the compatibility with other improvements on structural approxiamtions. Finally, we will give a conclusion to this thesis.

5.1 Discussion on Limitations

In our MKISSGP approximation scheme, we developed the idea of density on the RBF kernel and proved that it has a direct relationship with any accuracy metric. We can generalize this notion of density to some other kernels that are **stationary**. For example, the rational quadratic kernel if we assume a fixed α parameter:

$$k_{RQ}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 \exp\left(1 + \frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\alpha l^2}\right)^{-\alpha}. \tag{5.1}$$

Since it is stationary, we can use the same technique as in Section 3.3.1 to represent $(\mathbf{x}_i - \mathbf{x}_j)^2$ as a function of relative distance which then cancels out the l^2 term in the denominator. However, more complicated kernels like the Matérn kernel do not have similar results. Thus, this density-driven MKISSGP method is only valid for some specific stationary kernels, which includes the most popular RBF kernel.

Another limitation and possible drawback of MKISSGP is that when the length scale of the kernel is too small compared with the span of the training points, the theoretical number of grid points can get exceptionally large. In practice, this situation sometimes occurs during the correct process of training even when the underlying distribution has a reasonable length scale. This is due to the randomness of the iterations of the optimization process, in which a large step towards shrinking the length scale could happen. The KISSGP method, however, prevents this problem by deliberately fixing the number of grid points regardless of the length scale. We could borrow this idea by fixing an upper limit for the number of grid points, which is set to a default of 1000 in our software. We observed that this at most times is harmless to the overall training process since such "incautious" steps are often taken in the first few iterations of the optimization. And the accuracy requirement for these iterations is not that high as long as the direction of decent is calculated correctly (which is easy to satisfy). If this happens in the final iterations, the most possible implication is that the data itself consists of a long span of turbulent data points. One might want to adjust the upper bound for the number of grid points accordingly to reach the desired level of accuracy of the final result.

5.2 Discussion on Compatibility

As mentioned first in Section 2.4.1 and in chapter Chapter 3, the number of grid points is the crucial item that controls the time consumption of the SKI-based frameworks. Previous works [23][24][25] have focused on leveraging the impact on the scaling of dimensions while omitting the problem of limiting the number of grid points in each dimension. On the other hand, our MKISSGP method focuses on determining the number of grid points in one dimension to reach a certain level of accuracy. It is a natural question to ask whether these two approaches can combine with each other.

The answer is yes, indeed the other methods can integrate the main idea of this thesis. Take the Kernel Interpolation with Sparse Grids (see Section 2.4.3) as an example, in the first step in constructing a sparse grid, the grid points per each dimension are specified by a resolution index l to partition the span of data in this dimension into 2^l equal parts. The number of l can be taken as the value that makes $2^l - 1$ closest to the number of grid points calculated in the MKISSGP framework in that dimension. The rest of the steps remain the same. One can even impose different accuracy and time requirements on different dimensions. For example, if the requirement for accuracy in the second dimension is not critical, one can choose a lower density in that dimension to shrink overall time consumption. In summary, due to the lack of consideration of the number of grid points in each dimension in every SKI-based approximation method, our method is completely compatible with many other contributions made to the SKI in the literature.

5.3 Conclusion

In this thesis, we solved the problem of finding a systematic way of determining the number of grid points per each dimension in SKI-based approximations. To achieve this accomplishment, we thoroughly studied the theory of GPR, compared and summarized the existing low-rank approximations, proposed our new method Malleable Kernel Interpolation for Scalable Structured Gaussian Process (MKISSGP), and conducted experiments to prove its superiority in main efficiency.

First, we present our main contribution:

- Presented a novel low-rank approximation framework: MKISSGP.

  MKISSGP extends the capabilities of the established state-of-the-art SKI-based KISSGP approximation. A key feature of MKISSGP is the introduction of a flexible grid point determination strategy. Specifically, MKISSGP dynamically adjusts the number of grid points according to the length scale in every training iteration. This strategy effectively minimizes the number of grid points required to achieve a desired level of accuracy, serving as the centerpiece of our innovation. By implementing this strategy, our method reached an asymptotic time complexity of $\mathcal{O}(n + m_{\text{min}}^2)$, where m_{min} denotes the minimum number of grid points necessary to attain the desired accuracy level. By adhering to our recommended density value $\rho = 2.7$, MKISSGP achieved a comparable level of accuracy to precise GPR while significantly reducing computation time—nearly halving it

in comparison to the previous state-of-the-art KISSGP method. The results highlight the superior performance of our approach in terms of both accuracy and efficiency for approximated GPR. In addition, our innovation aligns seamlessly with other advancements in the SKI framework outlined in existing literature, including enhancements aimed at scaling the approach to handle higher dimensions efficiently.

In auxiliary, we also made the following contributions during the analysis of existing works related to GPR:

- Studied the theory of GPR.
  We examined thoroughly the process of GPR, including the calculations of NMLL, dNMLL, and the form of the posterior distribution. We confirmed that the bottleneck of the GPR comes from the calculation of $\mathbf{K_n}^{-1}$ and $\log\det(\mathbf{K_n})$ during the training process and during evaluating the posterior.

- Summarized the existing low-rank approximations.
  We assert that the structural approximations represent state-of-the-art low-rank approximation techniques. These methods leverage specialized matrix structures that greatly simplify matrix-vector multiplications and incorporate iterative algorithms to compute inverse and log-determinant terms. Collectively, these innovations yield a remarkable time complexity of $\mathcal{O}(n + m^2)$, a substantial breakthrough when contrasted with existing approximations at the $\mathcal{O}(m^2 n)$ level. In experiments, the structural approximation achieved equivalent levels of accuracy as precise GPR in a significantly shorter time compared with other approximation methods.

5.4 Future Work

Based on the existing content of MKISSGP, there are remaining topics that are worth further investigation:

- Generalization of scope.

  Within the scope of this thesis, MKISSGP and its density function are specifically tailored to the utilization of the RBF kernel for data modeling. Nevertheless, as elucidated in Section 5.1, it is conceivable that the concept of density could potentially be extended to encompass other stationary kernels. The precise form of the density function for these alternative kernels remains to be explored and empirically verified. Furthermore, given our prior discussion in Section 3.4.1, where we established that the alteration of grid points during iterations is a computationally inexpensive operation, we can potentially provide for every kernel function a kernel-specific mapping from its hyperparameters to the distribution of grid points.

- Integration with dimension-scaling methods.

Numerous studies in the literature have tackled the issue of scaling KISSGP as the dimensionality increases, as evidenced by works such as [23], [24], and [25]. In Section 5.2, we explored the potential integration of MKISSGP with the Kernel Interpolation with Sparse Grids framework. However, the outcomes of this integration, as well as possible combinations with other approaches, remain uncharted and unexamined. Future research endeavors could focus on the realization and comparative analysis of these integrations, thereby forging a more comprehensive framework capable of addressing scalability concerns pertaining to the number of grid points in SKI-based approximations.

- Implement real datasets.

  In this thesis, we did not implement real-world datasets to test the ability of MKISSGP. It is actually significant to conduct such experiments since the data modeling process has a great influence on regression results. The robustness of our findings and conclusions could be substantially enhanced by applying MKISSGP to authentic, real-world datasets.

Bibliography

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[2] S. Ubaru, J. Chen, and Y. Saad, "Fast estimation of tr(f(a)) via stochastic lanczos quadrature," *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1075–1099, 2017.

[3] M. F. Hutchinson, "A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines," *Communications in Statistics-Simulation and Computation*, vol. 18, no. 3, pp. 1059–1076, 1989.

[4] K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson, "Scalable log determinants for gaussian process kernel learning," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] L. M. Adams, J. L. Nazareth *et al.*, *Linear and nonlinear conjugate gradient-related methods*. Siam, 1996, vol. 85.

[6] M. Yadav, D. Sheldon, and C. Musco, "Faster kernel interpolation for gaussian processes," *International Conference on Artificial Intelligence and Statistics*, pp. 2971–2979, 2021.

[7] J. N. S. J. Wright, "Numerical optimization," 2006.

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

[9] T. Chen, J. Morris, and E. Martin, "Gaussian process regression for multivariate spectroscopic calibration," *Chemometrics and Intelligent Laboratory Systems*, vol. 87, no. 1, pp. 59–71, 2007.

[10] R. Krems, "Bayesian machine learning for quantum molecular dynamics," *Physical Chemistry Chemical Physics*, vol. 21, no. 25, pp. 13 392–13 410, 2019.

[11] V. L. Deringer, A. P. Bartók, N. Bernstein, D. M. Wilkins, M. Ceriotti, and G. Csányi, "Gaussian process regression for materials and molecules," *Chemical Reviews*, vol. 121, no. 16, pp. 10 073–10 141, 2021.

[12] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein, "Deep neural networks as gaussian processes," *arXiv preprint arXiv:1711.00165*, 2017.

[13] A. G. Wilson, D. A. Knowles, and Z. Ghahramani, "Gaussian process regression networks," *arXiv preprint arXiv:1110.4411*, 2011.

[14] Y. Shen, "Bi-fidelity informed gaussian process regression methods and their applications," Ph.D. dissertation, The University of Iowa, 2023.

[15] H. Mori and E. Kurata, "Application of gaussian process to wind speed forecasting for wind power generation," in *2008 IEEE International Conference on Sustainable Energy Technologies*, 2008, pp. 956–959.

[16] B. Wang and Z. Mao, "Outlier detection based on gaussian process with application to industrial processes," *Applied Soft Computing*, vol. 76, pp. 505–516, 2019.

[17] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *Proceedings of the 2004 American control conference*, vol. 3. IEEE, 2004, pp. 2214–2219.

[18] E. D. Klenske, M. N. Zeilinger, B. Schölkopf, and P. Hennig, "Gaussian process-based predictive control for periodic error correction," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 1, pp. 110–121, 2015.

[19] Y. Pan, X. Yan, E. A. Theodorou, and B. Boots, "Prediction under uncertainty in sparse spectrum gaussian processes with applications to filtering and control," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2760–2768.

[20] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predictive control using gaussian process regression," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2019.

[21] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018.

[22] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

[23] M. Yadav, D. R. Sheldon, and C. Musco, "Kernel interpolation with sparse grids," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 883–22 894, 2022.

[24] A. G. Wilson, C. Dann, and H. Nickisch, "Thoughts on massively scalable gaussian processes," *arXiv preprint arXiv:1511.01870*, 2015.

[25] P. Izmailov, A. Novikov, and D. Kropotov, "Scalable gaussian processes with billions of inducing inputs via tensor train decomposition," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2018, pp. 726–735.

[26] C. Williams and M. Seeger, "Using the nyström method to speed up kernel machines," *Advances in neural information processing systems*, vol. 13, 2000.

[27] A. Smola and P. Bartlett, "Sparse greedy gaussian process regression," *Advances in neural information processing systems*, vol. 13, 2000.

[28] E. Snelson and Z. Ghahramani, "Sparse gaussian processes using pseudo-inputs," *Advances in neural information processing systems*, vol. 18, 2005.

[29] M. W. Seeger, C. K. Williams, and N. D. Lawrence, "Fast forward selection to speed up sparse gaussian process regression," in *International Workshop on Artificial Intelligence and Statistics*. PMLR, 2003, pp. 254–261.

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

[31] S. Arno and S. Simo, "Hilbert space methods for reduced-rank gaussian process regression," *Statistics and Computing*, vol. 30, no. 2, pp. 419–446, 2020.

[32] J. Gardner, G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson, "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, vol. 31, 2018.

[33] A. Wilson and H. Nickisch, "Kernel interpolation for scalable structured gaussian processes (kiss-gp)," *International conference on machine learning*, pp. 1775–1784, 2015.

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

[35] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When gaussian process meets big data: A review of scalable gps," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 11, pp. 4405–4423, 2020.

[36] E. Hannan, "Stationary time series," *Time Series and Statistics*, pp. 271–276, 1990.

[37] A. G. Wilson, E. Gilboa, A. Nehorai, and J. P. Cunningham, "Fast kernel learning for multidimensional pattern extrapolation," *Advances in neural information processing systems*, vol. 27, 2014.

[38] Y. Saatçi, "Scalable inference for structured gaussian process models," Ph.D. dissertation, University of Cambridge, 2012.

[39] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE transactions on acoustics, speech, and signal processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[40] C. T. Baker, *The numerical treatment of integral equations*. Clarendon Press, 1967.

[41] H.-J. Bungartz and M. Griebel, "Sparse grids," *Acta numerica*, vol. 13, pp. 147–269, 2004.

[42] J. H. Halton, *Simplicial multivariable linear interpolation*. University of North Carolina at Chapel Hill. Department of Computer Science, 1991.

[43] P. J. Davis, *Interpolation and approximation*. Courier Corporation, 1975.

[44] G. Ammar, W. Dayawansa, and C. Martin, "Exponential interpolation: theory and numerical algorithms," *Applied Mathematics and Computation*, vol. 41, no. 3, pp. 189–232, 1991.

[45] D. A. Kraaijpoel, "Seismic ray fields and ray field maps: theory and algorithms," Ph.D. dissertation, Utrecht University, 2003.

[46] J. E. Johnson, V. Laparra, A. Pérez-Suay, M. D. Mahecha, and G. Camps-Valls, "Kernel methods and their derivatives: Concept and perspectives for the earth system sciences," *Plos one*, vol. 15, no. 10, p. e0235885, 2020.

[47] J. Machalová, "Optimal interpolating and optimal smoothing spline," *Journal of Electrical Engineering*, vol. 53, pp. 79–82, 2002.

[48] A. Gilman, D. Bailey, and S. Marsland, "Least-squares optimal interpolation for fast image super-resolution," in *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications*. IEEE, 2010, pp. 29–34.

[49] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2