Article

Efficient Gaussian Process Calculations Using Chebyshev Nodes and Fast Fourier Transform

Adrian Dudek and Jerzy Baranowski *

Department of Automatic Control & Robotics, AGH University of Science & Technology, 30-059 Kraków, Poland; addudek@agh.edu.pl
* Correspondence: jb@agh.edu.pl

Abstract: Gaussian processes have gained popularity in contemporary solutions for mathematical modeling problems, particularly in cases involving complex and challenging-to-model scenarios or instances with a general lack of data. Therefore, they often serve as generative models for data, for example, in classification problems. However, a common problem in the application of Gaussian processes is their computational complexity. To address this challenge, sparse methods are frequently employed, involving a reduction in the computational domain. In this study, we propose an innovative computational approach for Gaussian processes. Our method revolves around selecting a computation domain based on Chebyshev nodes, with the optimal number of nodes determined by minimizing the degree of the Chebyshev series, while ensuring meaningful coefficients derived from function values at the Chebyshev nodes with fast Fourier transform. This approach not only facilitates a reduction in computation time but also provides a means to reconstruct the original function using the functional series. We conducted experiments using two computational methods for Gaussian processes: Markov chain Monte Carlo and integrated nested Laplace approximation. The results demonstrate a significant reduction in computation time, thereby motivating further development of the proposed algorithm.

Keywords: gaussian process; Chebyshev polynomials; fast Fourier transform; integrated nested Laplace approximations; Markov chain Monte Carlo

Citation: Dudek, A.; Baranowski, J. Efficient Gaussian Process Calculations Using Chebyshev Nodes and Fast Fourier Transform. *Electronics* **2024**, *13*, 2136. https://doi.org/10.3390/electronics13112136

Academic Editor: Olivier Sename

Received: 20 February 2024
Revised: 17 May 2024
Accepted: 24 May 2024
Published: 30 May 2024

1. Introduction

Gaussian processes (GPs) have gained significant popularity in the fields of machine learning and statistics, owing to their remarkable adaptability and capacity to capture intricate, non-linear associations. These non-parametric Bayesian models empower the inference of complex, non-linear functions from data while furnishing valuable uncertainty assessments. Essentially, GPs embody a stochastic process that posits a probability distribution over functions, ensuring that any finite collection of function values conforms to a joint Gaussian distribution.

In essence, a GP lays down a prior distribution over functions, subsequently refined through Bayesian inference as more data become available. Notably, GPs exhibit the unique capability of accommodating irregularly spaced and noisy data, as well as capturing inherent correlation structures within the data [1–3].

The utility of GP spans an array of tasks, encompassing regression [4–6], classification [7,8], and optimization [9,10]. These versatile models have found successful applications in diverse domains, including support for bike-sharing systems [11], computer vision [12], and even diesel engine optimization [13]. A prominent feature of GP is its ability to yield probabilistic predictions and effectively account for data noise, as previously noted.

However, the predominant drawback of employing GPs lies in their computational demands [3]. Considerable efforts have been dedicated to enhancing the computational efficiency of GPs, particularly when dealing with large datasets [14,15]. Prominent techniques in this regard include sparse approximation methods and scalable algorithms, which aim to make GPs more accessible and computationally tractable [16,17].

In recent years, the integrated nested Laplace approximation (INLA) framework has emerged as a powerful and efficient approach for modeling GPs in various statistical and machine learning applications. INLA combines elements of Bayesian modeling with numerical approximations, offering a computationally tractable alternative to traditional Markov chain Monte Carlo (MCMC) methods for estimating GP parameters and making predictions [18]. INLA excels in providing accurate and rapid inference for GPs, particularly when dealing with large datasets or complex models [19]. It leverages a combination of deterministic numerical integration techniques and Laplace approximations to approximate the posterior distribution of GP parameters, such as the length scale and amplitude hyperparameters. This approach significantly reduces the computational burden associated with GP modeling, making it feasible to work with extensive datasets and intricate models [20].

Less attention is given to improving the efficiency of using GPs as generative models. In general, GPs are utilized as generative models within the larger algorithm framework. For instance, the authors of [21] demonstrate the technique for embedding unlabeled categorical data into a continuous space using GP. As a probabilistic generative model, it is capable of estimating densities. They present a generative classification model that operates as a supervised approach for labeled categorical data. Moreover, the authors of [22] developed a Gaussian process density sampler, an exchangeable model for nonparametric Bayesian density estimation. This model generates independent samples from a transformed function, derived from a GP prior. Their approach facilitates the inference of an unknown density from data through MCMC methods, which gives samples from both the posterior distribution over density functions and the predictive distribution in the data space. A good representation of the GP generative model, which was optimized through approximation, is presented in [23], where the authors address the problem of Koopman mode decomposition. Their generative GP model allows for the concurrent estimation of specific quantities and latent variables controlled by an unidentified dynamical system. Moreover, they propose an effective method for parameter estimation within the model, which utilizes low-rank approximations of covariance matrices. The optimization of the generative model in the mentioned case primarily relies on reducing the dimension of the covariance matrix, a commonly employed method that facilitates computation time reduction. An underexplored aspect of such optimizations involves decreasing the dimensions of the mentioned matrix in a manner that additionally provides further information for the generative model. Our proposed approach for selecting computation points for GP is based on the premise of not only reducing computation time but also enhancing the generative model's performance through the application of output in the form of functional series.

In this paper, we aim to explore the feasibility and efficiency of leveraging the Chebyshev property for efficient GP calculations. Our main contributions to this paper are as follows:

- **Selecting computational points for GP by using the properties of Chebyshev nodes.** The proposed approach seeks to mitigate the computational challenges associated with GPs by reducing the number of computation points and adopting a specific method to gather information for generating functions with series representation
- **An algorithm output in the form of a functional series, achieving low computational complexity with minimal loss of accuracy.** Normally, it is difficult to ensure the series representation, but our approach allows us to address this challenge with the use of fast Fourier transform (FFT) formulas to convert function values at Chebyshev points into coefficients of the Chebyshev series. This can be done with minimal loss of accuracy (1 bit) while facilitating effective interpolation of the series.
- **An algorithm that is especially susceptible to heavy computational methods.** Alleviating computational burden by reducing the covariance matrix within Chebyshev node calculations can enable the use of high-cost computational methods like MCMC sampling on a wider scale.

- **Improving the usability of GP generative models.** Ensuring the function series output with FFT conversion, with low cost and accuracy loss can lead to overall improved generative model capabilities, for example, in terms of classification.

The rest of the paper is organized as follows. We begin by presenting the fundamental theory behind GPs, generative models, and Chebyshev interpolation that we use in our solution as well as a description of computational methods used in the experiment section (MCMC and INLA framework). Subsequently, we detail the experiments conducted and the application of the proposed methodology to GP computations. Finally, we conclude with a discussion and summary of our findings.

2. Materials and Methods

This section of the article is dedicated to explaining the fundamental theory behind Gaussian processes (GPs), including their covariance functions and practical applications. Furthermore, we describe the theory and use of generative models in the context of GPs. We also describe generative models to highlight the application context of the method. Moreover, we focus on the components of Chebyshev interpolation, which we employ in our proposed process. This starts with a description of Chebyshev nodes and extends to the convergence of functional series, detailing the specifics of the algorithm for converting values at Chebyshev nodes into Chebyshev series coefficients using FFT. It also describes the computational methods used in the experimental section, such as Markov chain Monte Carlo (MCMC) methods and the integrated nested Laplace approximation (INLA) framework, which are employed for efficient Bayesian inference. Additionally, this section provides an overview of the algorithm based on the aforementioned theory, explaining how these elements are bound to address the problem.

2.1. Gaussian Process

We need to provide a concise definition of GPs and explore their fundamental properties. A widely accepted and succinct definition of a GP comes from [2] and can be summarized as follows:

A Gaussian process is a collection of random variables, any finite number of which jointly follow a Gaussian distribution.

To define a GP mathematically, let us introduce the mean function $m(x)$ and the covariance function $k(x, x')$ for a real process $f(x)$ as follows:

$$m(x) = \mathbb{E}[f(x)] \tag{1}$$

$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \tag{2}$$

With these definitions, we can formally express a GP as follows:

$$f(x) \sim GP(m(x), k(x, x')) \tag{3}$$

The mean and covariance functions provide a complete characterization of the GP prior, describing the expected behavior of the function, f.

To define GP in another way, we can consider the following set of observations: $\boldsymbol{f} = \{f(\boldsymbol{x}_n)\}_{n=1}^N$; then the prior distribution of $\boldsymbol{f}$ follows a multivariate Gaussian distribution: $\boldsymbol{f} \sim \text{Normal}(\boldsymbol{\mu}, \boldsymbol{K})$, where $\boldsymbol{\mu} = \{\mu(\boldsymbol{x}_n)\}_{n=1}^N$, and $\boldsymbol{K}$ is the covariance matrix with elements $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The joint distribution of f and a new variable $\tilde{f}$, assuming $\boldsymbol{\mu} = 0$, is expressed as follows:

$$p(\boldsymbol{f}, \tilde{f}) = \text{Normal}\left(\begin{bmatrix} \boldsymbol{f} \\ \tilde{f} \end{bmatrix} \middle| \boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}_{f,f} & \boldsymbol{k}_{f,\tilde{f}} \\ \boldsymbol{k}_{\tilde{f},f} & k_{\tilde{f},\tilde{f}} \end{bmatrix} \right) \tag{4}$$

where $\boldsymbol{k}_{f,\tilde{f}}$ represents the covariance between $\boldsymbol{f}$ and $\tilde{f}$ (creating a covariance matrix), and $k_{\tilde{f},\tilde{f}}$ is the prior variance of $\tilde{f}$ [2].

In a broader context, GPs are stochastic processes commonly used for modeling data observed over time, space, or both [1]. They can be viewed as generalizations of normal probability distributions, where each random variable (scalar or vector in the case of multivariate distributions) is described by a GP. Formally, a GP on a set $\mathcal{T}$ is a collection of random variables indexed by a continuous variable, denoted as $\{f(t) : t \in \mathcal{T}\}$, with the property that any finite subset of these random variables follows a multivariate Gaussian distribution [24,25].

To fully specify a GP, one needs to define the mean (often set to "0" for simplicity) and covariance functions. However, variations in the mean function can be considered for interpretability or prior specification purposes [1,2,26]. The covariance function, also known as the kernel function, quantifies the similarity between data points and is typically chosen from a set of predefined functions [27]. While choosing the appropriate covariance function is crucial, any function generating a positive definite covariance matrix can be used [28]. Nevertheless, creating new covariance functions that are both mathematically and practically useful can be challenging.

In this research, our attention is primarily directed toward exploring the intricacies and applications of two distinct kernels: the radial basis function (RBF) and the Matérn kernel. These kernels were chosen for their unique properties and relevance in the field of study, offering us the opportunity to delve deeper into their theoretical foundations and practical implementations. The RBF kernel is defined as follows:

$$k(x_i, x_j) = \exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right) \tag{5}$$

where l represents the characteristic length scale and $d(\cdot,\cdot)$ is the Euclidean distance. The RBF kernel's key property is its dependence, primarily on the distance from the specified point. In the case of a twice differentiable function, we use a kernel from the Matérn family. The Matérn kernel is a kind of generalization of the RBF function. The general formula for Matérn covariance functions is given by (6):

$$k(x_i, x_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}d(x_i, x_j)\right) \tag{6}$$

where K_ν is a modified Bessel function, l is a length-scale parameter and v is a parameter that allows changing smoothness of the function, which gives an ability to flexibly control the function in relation to the one we want to model. Of particular note are the frequently used parameter, v, with values of 3/2 used for learning functions that are at least once differentiable, and 5/2 for functions that are at least twice differentiable [2].

The kernel choice plays a pivotal role in determining the regularity of the function from the GP. Specifically, the Matérn kernel exhibits a notable characteristic where it is k times differentiable. This differentiability property is integral in defining the smoothness of the function modeled by the Gaussian process. A higher degree of differentiability implies a smoother function, which can be crucial in various applications where the smoothness of the function has significant implications on the accuracy and reliability of the model's predictions or interpretations [2].

GPs play a significant role in the advancement of modern technologies and sciences, where they are used for prediction, optimization, or classification [3]. In practice, they are useful, for instance, in industrial anomaly detection [29], and crucial for ensuring production quality and continuity. GPs predict battery state parameters [30], which are relevant in electric vehicles and energy management [4]. In bike-sharing systems, they optimize bike allocation, enhancing efficiency [31]. Furthermore, GPs are applied in tracking and location, especially where GPSs fail [32]. In optimization areas, they aid in resource management, like in data center designs [9]. These processes are key in data analysis and statistical modeling, used across various fields, from image processing [12] to biomedical data analysis [33], demonstrating their versatility and importance in contemporary applications.

2.2. Generative Models

A generative model is a type of statistical algorithm designed to generate new data similar to the data it was trained on, but not identical. These models learn the probability distribution of the training data, allowing them to create new instances that retain the statistical characteristics of the training set. Generative models are widely used in various fields such as natural language processing, image generation, speech synthesis, and others, where they can create realistic text, images, sounds, or other types of data [34].

In the context of GPs, we utilize function sampling from posterior distributions, enabling statistical inference based on available data. This method relies on data influencing the distribution of hyperparameters, allowing the generation of new data samples and creating a generative model [2]. An example of a generative model can be found in [29], where the described instance of diagnosing malfunctions in industrial processes. In this case, the GP model is trained with measurement data from both normal and healthy operation of equipment as well as from moments when faults occur. The model then becomes capable of generating more time series data, which consequently leads to the ability to classify new situations based on data depth. This approach exemplifies the application of GPs in predictive maintenance, where they assist in interpreting complex data patterns to identify potential issues before they become critical failures, enhancing both the efficiency and safety of industrial operations. Another example can be found in [35]. In this problem, we use generative models within a Gaussian Mixture to classify the faulty or healthy states of induction motor startups.

GPs provide a deeper understanding of data relationships, which is particularly useful in estimating hyperparameters and calculating posterior distributions [2]. In function modeling, it is crucial to focus not only on values at measurement points but also on the overall function trajectory. An important aspect of GPs is that we sample a set of points, not the entire function. The problems we are dealing with involve how to choose specific points at which to calculate the GP and how to generate functions from the model. A simple rule of thumb states that the more data points we have information on, the easier and better the interpolation will be, regardless of the method chosen. However, in the case of GP, significantly increasing the number of computational points (random variables) can lead to computational burdens [3]. Therefore, we must strive for a robust method of interpolating functions while maintaining as few computational points as possible for GP.

2.3. Chebyshev Interpolation

Our proposed solution is to generate functions through a GP model in the form of a functional series, an example of which is the Chebyshev series. GP generates a point from a function. We want to transform those points into a Chebyshev series. Fundamentally, obtaining its coefficients is not a simple task, but each polynomial can be uniquely expressed as a finite Chebyshev series. When the polynomial is defined by its values at Chebyshev points, there is a unique linear relationship between these values and the coefficients of the Chebyshev expansion. This relationship allows for an efficient mapping, executable in $O(nlogn)$ operations using the FFT. This approach offers a computationally effective way of handling polynomial expressions and transformations [36].

In the context of generative modeling, where new functions are generated, the application of the Chebyshev series becomes particularly relevant. In such a case, it is very useful that the Chebyshev series are known for their convergence properties, particularly toward differentiable functions that are generated by the model. Moreover, the degree of approximation plays a crucial role in determining the nature and rate of this convergence [36]. The convergence of the Chebyshev series to the generated functions with the GP model can be mathematically represented by the below formula, where for integer $\nu \geq 1$, let f and all its differentiable functions up to $f^{(\nu-1)}$ and the ν th differentiable f^{ν} be of bounded variation, V, then for any $n > \nu$, we have the following:

$$\|f - f_n\| \leq \frac{2V}{\pi\nu(n-\nu)^{\nu}} \tag{7}$$

Unfortunately, computing the values of the Chebyshev series can be an exceptionally demanding task. In such instances, we might resort to utilizing the Chebyshev interpolant as an alternative to the series itself [36]. The Chebyshev series and interpolants are quite similar in nature, and the convergence of the latter can be articulated through the following expression:

$$\|f - p_n\| \leq \frac{4V}{\pi\nu(n-\nu)^{\nu}} \tag{8}$$

As we can observe in the convergence formulas, the Chebyshev interpolation differs little in terms of accuracy, amounting to only twice the results, which means that in computer calculations, we only lose one bit of accuracy. This minimal loss of precision is compensated by efficiency and numerical stability, which are crucial in many engineering and scientific applications. Thus, we can speak of a significant consistency in the representation of the series and its interpolation using Chebyshev polynomials in terms of our problem. The optimal interpolation nodes are known as Fekete points, which for scalar interpolation are the Legendre nodes. They require solving a large-dimensional eigenvalue problem. Chebyshev nodes have low complexity but are approximately two times worse than Legendre nodes, equating to 1 bit of accuracy [36].

The next problem we must confront is how to obtain the Chebyshev coefficients. This aspect is crucial because the correct determination of these coefficients is fundamental to the effectiveness of the entire method. Thanks to the work of Ahmed and Fisher in 1970 [37], we understand that using interpolation formulas with the FFT on Chebyshev polynomial values can lead to the accurate determination of Chebyshev series coefficients. This process results in minimal loss of accuracy, around the order of one bit [36], due to Chebyshev interpolation. This implies that while efficiently computing the coefficients for a Chebyshev series through FFT, we maintain a high degree of precision, making this method highly valuable for accurate interpolations in various computational applications. Thus, our solution to the problem of function generativity is the use of the Chebyshev series form, where we can quickly calculate the series coefficients by utilizing FFT on Chebyshev node values. An important aspect is selecting the appropriate number of Chebyshev points, ensuring minimal information loss about the original function while avoiding overburdening the GP model with heavy calculations. We can apply a solution similar to that promoted in the modern approach to interpolation [36], where the accuracy of the interpolating polynomial stops at achieving machine precision. For differentiable functions, such as those from the Matérn family, Chebyshev coefficients gradually decrease to a certain level. This allows them to be used to determine the required precision. An example of such behavior can be seen in Figure 1, where we observe that the Chebyshev coefficients decrease as the degree of the Chebyshev series increases until it reaches machine precision. In our case, since we may not achieve machine precision accuracy, where subsequent values cease to be significant, we can use a threshold related to the noise variance, which we can observe as oscillations around a single value of the series coefficient, after a significant drop at the beginning.

The assumptions of our approach involve utilizing knowledge about where to apply specific points for constructing Chebyshev polynomials for the purpose of interpolating the Chebyshev series. This entails strategically selecting points that optimize the polynomial's ability to model the underlying function. Chebyshev polynomials are a sequence of polynomials that satisfy a specific orthogonality condition with respect to a weight function. The most commonly used weight function for Chebyshev polynomials is based on the inverse square root of the difference between the upper and lower bounds of the interval over which the polynomials are defined. The kth Chebyshev polynomial can be expressed as the real part of the function, z^k, on the unit circle [36,37]:

$$\begin{aligned} x &= \frac{1}{2}\left(z + z^{-1}\right) = \cos\beta, \quad \beta = \cos^{-1} x, \\ T_k(x) &= \frac{1}{2}\left(z^k + z^{-k}\right) = \cos(k\beta). \end{aligned} \tag{9}$$

One notable property of Chebyshev polynomials is the presence of equally spaced roots along the defined interval. These roots are known as Chebyshev nodes or Chebyshev points and can be expressed as the real parts of these points [36]:

$$x_j = \operatorname{Re} z_j = \frac{1}{2}\left(z_j + z_j^{-1}\right), \quad 0 \le j \le n. \tag{10}$$

An alternative way to define Chebyshev's points is in terms of the original angles [36]:

$$x_j = \cos\left(\frac{j\pi}{n}\right), \quad 0 \le j \le n. \tag{11}$$

Chebyshev polynomials are valuable for interpolation, particularly due to their effectiveness compared to polynomial interpolants created through equally spaced points. The clustering of points near the ends of the interval plays a significant role. Chebyshev points offer the advantage of having each point at a similar average distance from the others, as measured by the geometric mean. The resulting polynomial interpolant using Chebyshev points is referred to as the Chebyshev interpolant [36].

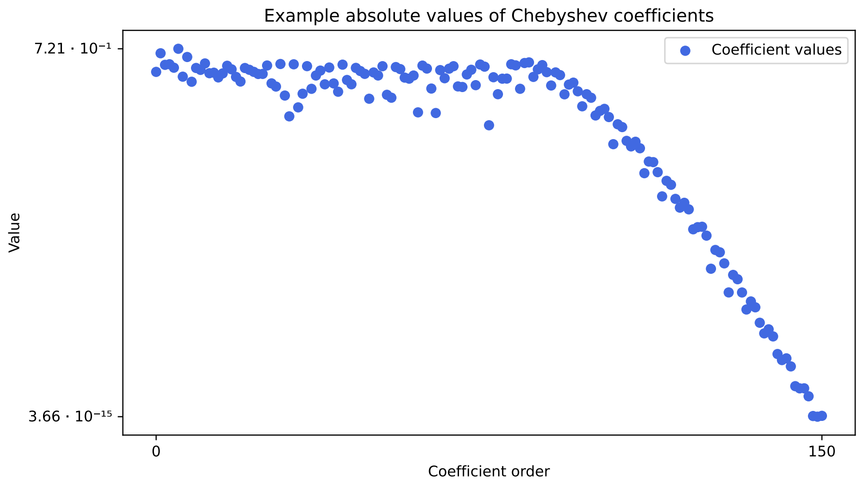

Figure 1. The plot presents Chebyshev series coefficient values compared to the order of the series. It illustrates the point at which the series reaches machine precision values with an interpolation algorithm, indicating which series order is required to achieve the desired accuracy of interpolation. For degrees up to approximately k = 80, the Chebyshev series struggles to accurately represent the function, f, as it cannot adequately capture the oscillations occurring at shorter wavelengths. However, beyond this point, there is rapid convergence in the series. By the time the degree reaches k = 150, the accuracy significantly improves to about 15 digits, leading to the truncation of the computed Chebyshev series at this point. This demonstrates the effectiveness of the Chebyshev series in approximating functions at higher degrees, where it becomes more capable of capturing the intricate details of the function.

Furthermore, Chebyshev polynomials can be exclusively represented in a finite Chebyshev series format. This means that the functions $T_0(x), T_1(x), \ldots, T_n(x)$ form the basis for P_n (the Chebyshev series). The relationship between values at Chebyshev points and Chebyshev expansion coefficients is straightforward and can be achieved using the FFT in $O(n \log n)$ operations. The Chebyshev series can be defined as follows:

$$p_n \approx \sum_{k=0}^{n} a_k T_k(x), \tag{12}$$

where $T_k(x)$ represents the kth Chebyshev polynomial, and a_k is the associated coefficient. The Chebyshev series exhibits several desirable properties, such as uniform convergence over the interval $[-1, 1]$ for any continuous function and rapid convergence for functions that are analytic or possess a smooth derivative [36,37].

2.4. Overview of Proposed Algorithm

The structure of our proposed algorithm, based on the gathered information, comprises several key components:

1. *Data feeding and fitting the model:* Involves using relevant data to train and adjust the Gaussian process (GP) model, ensuring an accurate representation of the system or phenomenon.
2. *Generative Gaussian process model:* The core of the algorithm, utilizing statistical methods for generating predictions or simulations from input data.
3. *Generating values at Chebyshev points:* Calculating values at crucial Chebyshev points for the subsequent transformation process.
4. *FFT transformation to Chebyshev series:* Applying fast Fourier transform to convert values from Chebyshev points into a Chebyshev series for efficient computational processing.
5. *Series degree optimization with noise variance condition:* Optimizing the Chebyshev series degree while considering noise variance (oscillations), balancing accuracy and computational load.

To help understand the algorithm's workflow and the interconnections between different components, Figure 2 presents a flow chart illustrating the general assumptions of the algorithm. Each step in the flow chart represents a specific part of the algorithm.

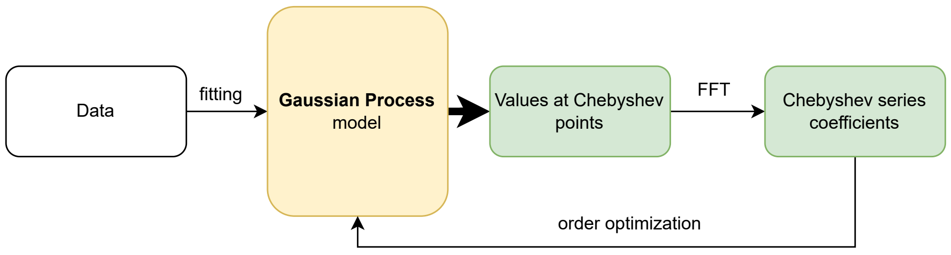

Figure 2. The flow chart visually outlines the structure and flow of the proposed algorithm. It begins with data fitting using a Gaussian process model, followed by generating values at Chebyshev points. The next step involves applying the FFT to these values, leading to the generation of Chebyshev series coefficients. The final stage of the algorithm is the optimization of the order of the series, a crucial step to enhance the algorithm's efficiency and accuracy.

2.5. Computational Methods

The current state of GP modeling reflects both its robust potential in statistical learning and the computational challenges inherent in its implementation. GPs are flexible non-parametric models that offer a principled way of encoding prior beliefs about functions, incorporating uncertainty, and handling various types of data. However, the computational demands of GPs, particularly with large datasets, pose significant challenges.

One of the primary computational challenges in GP modeling arises from the need to compute the inverse and determinant of the covariance matrix, a process that is typically cubic in the number of data points $(O(N))$ [2]. This complexity limits the scalability of standard GP methods to large datasets. Moreover, the problem of direct and analytical calculation of the hierarchical model of Gaussian processes has led to the exploration of alternative solution methods [3]. To address these challenges, several numerical methods and approximations have been developed.

One such method is the Markov chain Monte Carlo (MCMC) sampling method. MCMC is a powerful and widely used computational technique in statistics, machine learning, and various scientific fields. It is particularly valuable when dealing with complex probability distributions and high-dimensional spaces where traditional numerical methods become impractical or inefficient. MCMC provides a way to generate a sequence of random samples from a target probability distribution, allowing us to approximate various characteristics of that distribution [38].

In our research, we are using a Stan model as a probabilistic programming framework that leverages MCMC methods for Bayesian inference. Stan allows us to specify statistical models using a flexible modeling language and then employs MCMC techniques of Hamiltonian Monte Carlo (HMC) to sample from the posterior distribution of model parameters. This combination of Stan's modeling language and MCMC algorithms enables sophisticated and robust Bayesian inference, making it a powerful tool for a wide range of applications. For instance, in our case, it is particularly effective for the hierarchical modeling of GP [39].

The implementation of the GP generative model in Stan involves dealing with multivariate Gaussian processes and generative algorithms. This includes specifying a series of inputs and outputs and defining the probability of outputs conditioned on their inputs. Stan's approach to GPs is characterized by its focus on statistical efficiency and the handling of model conditioning and curvature. This is particularly relevant in Bayesian modeling contexts where the inference about the data-generating process is made using posterior distributions derived from a priori assumptions and observed data. Stan allows direct use of MCMC within its framework and generates data [39].

Building a Gaussian process (GP) model in Stan involves several key steps, from specifying the model to executing the simulation. Here is a general overview of the process we used:

1. Define the data.
2. Specify the GP prior.
3. Model the likelihood.
4. Define hyperparameters.
5. Parameter estimation.
6. Posterior inference.
7. Making predictions and generation of new samples.

Initially, we define both the input and output data for the model. Given that the model is generative in nature, we declare how many samples are required. In selecting priors, a crucial aspect is the choice of the covariance function, which is employed in constructing the covariance matrix. A common practice, also adopted in our approach, is the use of Cholesky decomposition. The modeling of our likelihood is based on declaring a normal distribution, complete with appropriate variance and potential noise. The hyperparameters of our covariance function are treated as priors over hyperparameters and are inferred from the data. The estimation of these parameters is carried out through MCMC sampling. Once the entire model is defined, posterior inference from the model becomes possible. This allows us to obtain the required parameters along with their uncertainties, enabling the use of the model to generate new responses and their associated uncertainties.

MCMC methods within the Stan framework are known for their high precision and computational complexity. These methods, while detailed and accurate, demand considerable computational resources. In contrast, there are alternative approaches that offer faster computational performance with a trade-off in the amount of information extracted from the model. One prominent alternative is the INLA method. The tool employed in our second modeling framework is the R-INLA package, which provides comprehensive implementations for solving problems using sparse matrices, as described in the INLA methodology. INLA, or the integrated nested Laplace approximation, was introduced by Rue, Martino, and Chopin in 2009 as an alternative to traditional MCMC methods for approximate Bayesian inference [40].

INLA is specifically tailored to models that can be expressed as latent Gaussian Markov random fields (GMRFs) due to their advantageous computational properties. Without going into detail, GPs (especially Matérn ones) are solutions to certain stochastic partial differential equations. Those solutions obtained with finite element methods can be considered GMRFs [41]. In the INLA framework, the observed variables $y = (y_1, \ldots, y_N)$ are modeled using an exponential family distribution. The mean μ_i (for observation y_i) is linked to the linear predictor η_i through an appropriate link function. The linear predictor encompasses terms related to covariates and various types of random effects, while the distribution of the observed variables y depends on certain hyperparameters θ.

The distribution of the latent effects x is assumed to follow a GMRF with a zero mean and a precision matrix $\mathbf{Q}(\theta)$, which is dependent on the hyperparameter vector θ [42]. The likelihood of the observations, given the vector of latent effects and the hyperparameters, can be expressed as a product of likelihoods for individual observations:

$$\pi(\mathbf{y} \mid \mathbf{x}, \theta) = \prod_{i \in \mathcal{I}} \pi(y_i \mid \eta_i, \theta) \tag{13}$$

In this context, the latent linear predictor is denoted as η_i, which is one of the components of the vector x containing all latent effects. Set $\mathcal{I}$ includes the indices of all the observed values of y, although some of these values may not have been observed.

The primary goal of the INLA approach is to estimate the posterior marginal distributions of the model's effects and hyperparameters. Model effects are the components of the model that capture the underlying structure and relationships within the data. This includes fixed effects, which are the deterministic part of the model related to the covariates, and random effects, which account for spatial or temporal correlations, incorporating variation produced by variables in the model that are not the primary focus of the study. Hyperparameters are the parameters of the kernel used in the model. This goal is achieved by leveraging the computational advantages of GMRF and utilizing the Laplace approximation for multidimensional integration. The joint posterior distribution of the effects and hyperparameters can be expressed as [42]:

$$\pi(\mathbf{x}, \theta \mid \mathbf{y}) \propto \pi(\theta) |\mathbf{Q}(\theta)|^{1/2} \exp\left\{ -\frac{1}{2} \mathbf{x}^\top \mathbf{Q}(\theta) \mathbf{x} + \sum_{i \in \mathcal{I}} \log(\pi(y_i \mid x_i, \theta)) \right\} \tag{14}$$

Here, the precision matrix of the latent effects is denoted as $\mathbf{Q}(\theta)$, and its determinant is represented by $|\mathbf{Q}(\theta)|$. Additionally, $x_i = \eta_i$ for values of i that are in the set $\mathcal{I}$ [42].

The computation of the marginal distributions for the latent effects and hyperparameters involves integrating over the hyperparameter space, requiring a reliable approximation of the joint posterior distribution of the hyperparameters [40]. This is accomplished by approximating $\pi(\theta \mid \mathbf{y})$ as $\tilde{\pi}(\theta \mid \mathbf{y})$ and using it to approximate the posterior marginal distribution of the latent parameter x_i:

$$\tilde{\pi}(x_i \mid \mathbf{y}) = \sum_k \tilde{\pi}(x_i \mid \theta_k, \mathbf{y}) \times \tilde{\pi}(\theta_k \mid \mathbf{y}) \times \Delta_k \tag{15}$$

The weights Δ_k correspond to a vector of hyperparameter values θ_k in a grid, and the specific method for calculating the approximation $\tilde{\pi}(\theta_k \mid \mathbf{y})$ can vary depending on the chosen approach [42].

The primary distinction for Gaussian process (GP) generative models lies in the nature of the output from the modeling approach. In the case of an MCMC-based sampling model in Stan, complete information is obtained in the form of samples at specific points, which are then used to determine the underlying distribution. This approach in Stan allows for a direct and comprehensive understanding of the data distribution, as it provides specific instances (samples) of the model's output [39].

On the other hand, INLA provides information primarily about the distribution of the random variables. Instead of generating samples directly from the distribution values, INLA approximates the posterior distribution of both the latent field and hyperparameters using Laplace approximation methods. This approximation leads to a more efficient computational process but offers a less direct insight into the specific instances of the model's output [41].

3. Results

Numerical experiments were conducted using the efficient GP computation algorithm we developed. The process for each scenario consists of the following steps:

1. Selecting and defining the tested function.
2. Generating a small subset of samples with specified noise from the function as data input.
3. Creating a Gaussian process model.
4. Fitting of the Gaussian process model.
5. Generating samples (distributions) at Chebyshev points from the GP model.
6. Converting samples (distributions) into Chebyshev series coefficients.
7. Optimizing the degree of the Chebyshev series.

Approaches vary between the MCMC (HMC) sampling method and the INLA framework. With MCMC, we obtain 4000 samples at each random variable, from which we gather information about the distribution at each point, while with INLA, we only obtain information about the distribution at certain points (for this reason, INLA can run more efficiently). A detailed chart for the algorithm used in the numerical experiments can be seen in Figure 3.

In the initial stage, two different functions are defined for testing purposes. The first function is a simple—at least twice differentiable, i.e., used for Matérn kernels—function and is defined as follows:

$$f(x) = |\sin(x)|^3 \tag{16}$$

The second one is an infinitely differentiable function (used for RBF kernels) and defined as follows:

$$f(x) = e^x + e^{x^2} \tag{17}$$

For these defined functions, a small number of points (20 in our experiments) was generated and then joined with random values from a Gaussian distribution (mean $\mu = 0$; standard deviation $\sigma = 0.1$) to introduce noise. Figures 4 and 5 illustrate the defined functions and the noisy samples generated for them, used in the presented experiment. To simplify subsequent operations with Chebyshev nodes, the domain was set to a range of $[-1, 1]$.

The next stage involves defining and creating a Gaussian process model. At first, we use the GP model created in the Stan, which allows us to create an accurate model based on defined priors and generate samples using MCMC tailored to the specifics of the problem. In our case, the GP model is characterized by a mean of 0. We also use two different covariance functions, one for each case of the function. For the case of an infinitely differentiable function, we use the radial basis function (RBF), also known as the squared exponential (SE) kernel. Also, as we deal with the twice differentiable function, we set the Matérn value to $5/2$. Both of these kernels are the most common in terms of GP usage. Our model in Stan was set up based on general guidelines available in [39]. Our priors were set as normal distributions, we used Cholesky decomposition for our covariance matrix, and generated results for random variables (computation points) in the sample generating block.

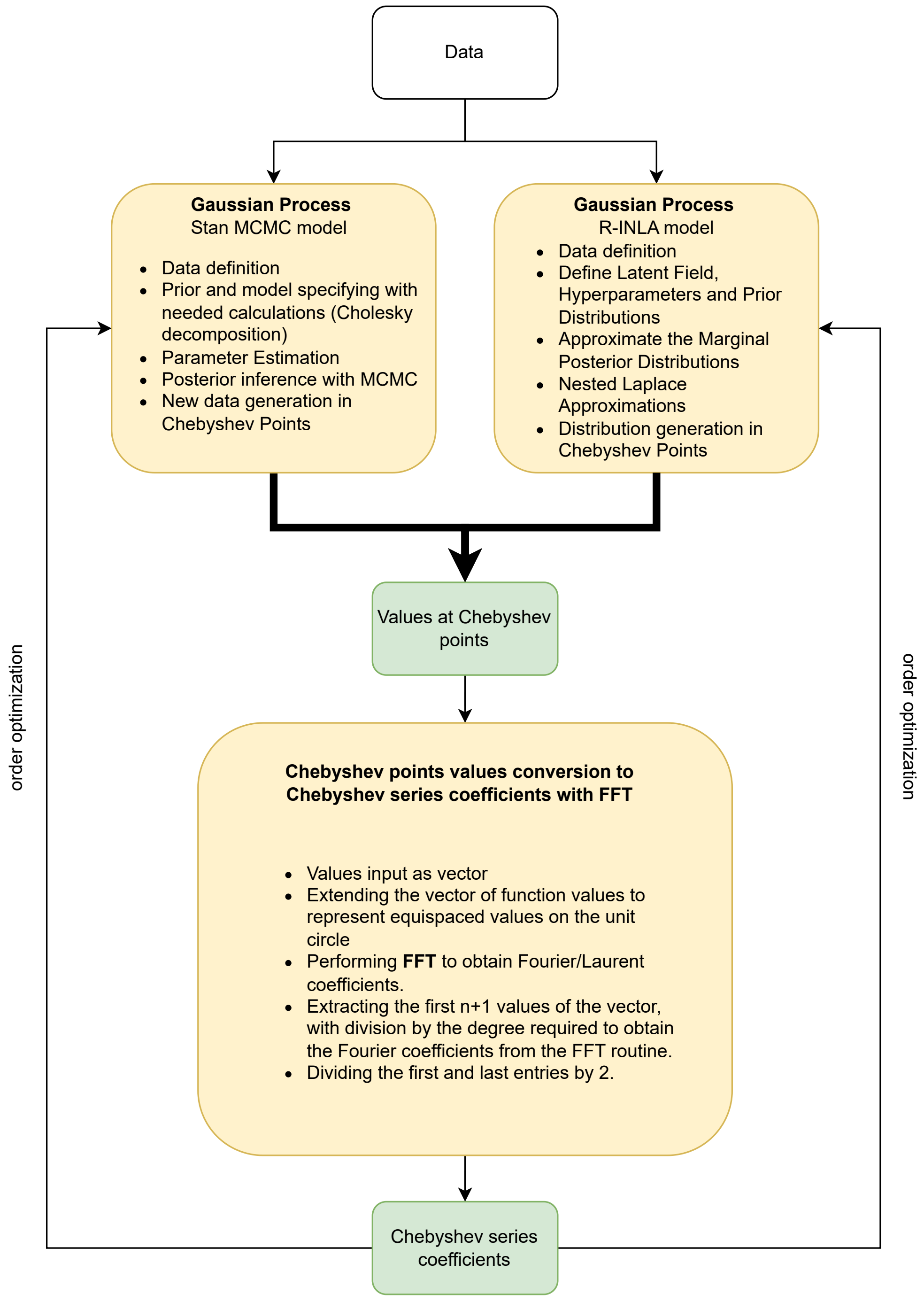
Data
Gaussian Process
Stan MCMC model
• Data definition
• Prior and model specifying with needed calculations (Cholesky decomposition)
• Parameter Estimation
• Posterior inference with MCMC
• New data generation in Chebyshev Points
Gaussian Process
R-INLA model
• Data definition
• Define Latent Field, Hyperparameters and Prior Distributions
• Approximate the Marginal Posterior Distributions
• Nested Laplace Approximations
• Distribution generation in Chebyshev Points
Values at Chebyshev points
Chebyshev points values conversion to Chebyshev series coefficients with FFT
• Values input as vector
• Extending the vector of function values to represent equispaced values on the unit circle
• Performing **FFT** to obtain Fourier/Laurent coefficients.
• Extracting the first n+1 values of the vector, with division by the degree required to obtain the Fourier coefficients from the FFT routine.
• Dividing the first and last entries by 2.
Chebyshev series coefficients
order optimization
order optimization

Figure 3. The provided flowchart effectively illustrates the intricate steps and progression of the designated algorithm. It initiates with the phase of data fitting, which is conducted using a Gaussian process model. This model is executed in two distinct manners: firstly, by employing the integrated nested Laplace approximation (INLA) method, and secondly, through the use of the Stan framework with Markov chain Monte Carlo (MCMC) techniques. The primary objective of these generative models is to facilitate the generation of values at specific Chebyshev points. Following this, the algorithm advances to the implementation of the fast Fourier transform (FFT). This transformative process is integrated into the script and is pivotal in deriving the Chebyshev series coefficients from the aforementioned values. Concluding the algorithm's sequence is the optimization phase, where the series order is meticulously adjusted. This final step is instrumental in elevating the overall efficacy and precision of the algorithm, ensuring optimal performance and accurate outcomes.

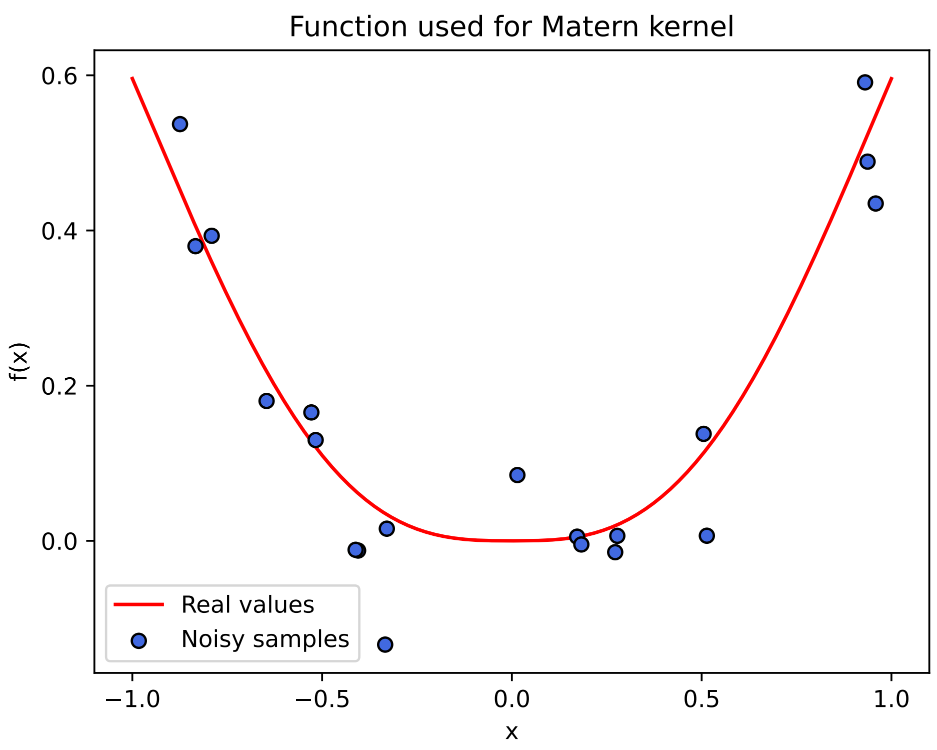

Figure 4. The plot shows an example of the twice differentiable function (red line) with sampled noisy values (blue) used for model fitting.

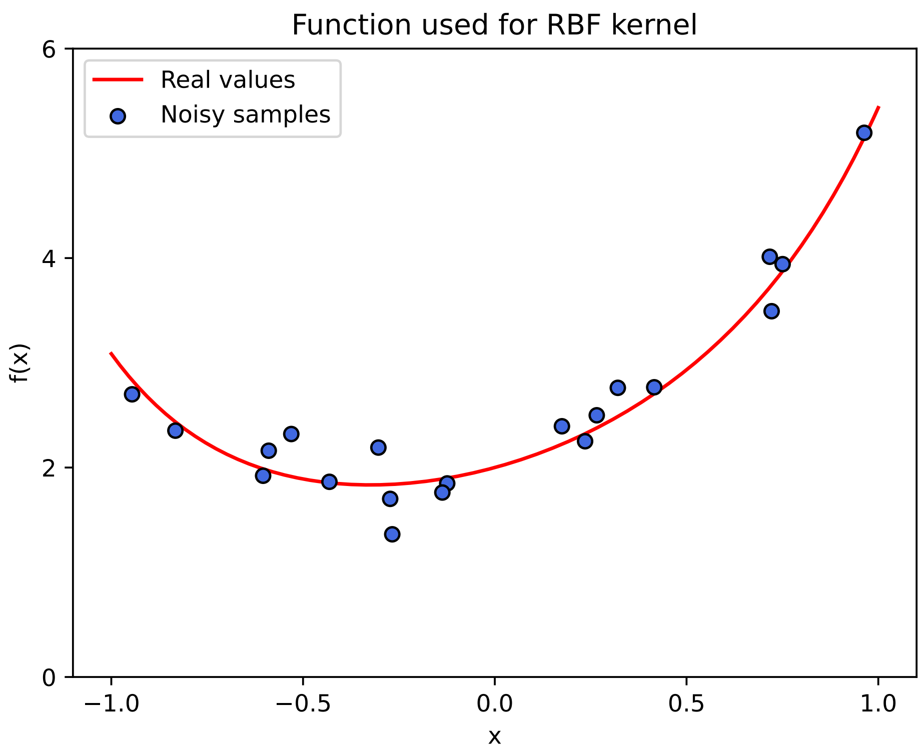

Figure 5. The plot shows an example of the infinitely differentiable function (red line) with sampled noisy values (blue) used for model fitting.

In the case of the INLA framework, since we use the stochastic partial differential equation (SPDE) approach, we can only set covariance parameters to use the Matérn family function. But as Matérn is a variation of the RBF function, we can set the parameter v to high values to approximate RBF function behavior. In the implementation of R-INLA, we first generate a one-dimensional mesh based on which we generate matrix A (the operation matrix on the mesh). As a prior for the similarity matrix, we choose the identity matrix (for random effects). Priors for hyperparameters, as in the case of Stan, are also normal distributions. All created objects are enclosed in what is called a stack, and then a model is created from which we infer the distributions. More on this can be found in [41].

The models were fitted using a small subset of generated noisy data from the original functions. The standard deviation of added noise was $\sigma = 0.1$. It is important to note that the number of Chebyshev nodes used for sample generation directly impacts the degree of the Chebyshev series obtained later, which is closely related to the optimization of this degree, as discussed below. To obtain appropriate preliminary conclusions, the number of Chebyshev nodes we used varied from 10 to 200. Such a discrepancy in values allowed us to examine not only the optimized degree of the Chebyshev series but also, for example, the computational efficiency.

Once the function values were calculated at the Chebyshev nodes, we utilized the FFT to convert them into Chebyshev series by calculating their coefficients, and a script was developed for this purpose. The kth Chebyshev polynomial on the unit interval can be interpreted as the real part of the monomial z^k on the unit disc in the complex plane, given by $z^k = Re(z^k) = \frac{1}{2}(z^k + z^{-k})$. Thus, when adding n+1 Chebyshev polynomials, a truncated Laurent series in the variable z was obtained, with the same coefficients for the z^k and z^{-k} terms. Similarly, the Chebyshev points on the interval $[-1, 1]$ can be seen as the real parts of equispaced nodes on the unit circle.

This connection between the unit interval and the unit circle allowed us to utilize Fourier analysis [37]. A truncated Laurent series with equal coefficients for the z^k and z^{-k} terms is equivalent to a Fourier series in the variable s, where $z = \exp(i \cdot s)$. Therefore, Fourier and Laurent coefficients are identical, and the coefficient vector is symmetric since the same factor multiplies the z^k and z^{-k} terms. In this context, the Chebyshev coefficients correspond to the first n + 1 terms of this vector, with the first and last coefficients divided by 2.

The script based on information introduced in [36,43] was implemented and performed via the following steps:

1. We gathered function values from the model results into vectors.
2. We extended the vectors of function values to represent equispaced values on the unit circle, progressing anticlockwise, starting from an x value of 1.
3. We performed FFT to obtain Fourier/Laurent coefficients. Since the Chebyshev coefficients are real-valued, the real part of this vector was taken to eliminate any spurious imaginary components due to rounding errors.
4. We extracted the first $n + 1$ values of the vector, adjusting by the degree required to obtain the Fourier coefficients from the FFT routine.
5. We divided the first and last entries by 2.

The script can also be seen in Figure 3 as a part of the whole algorithm.

With the calculated Chebyshev series coefficients for the generated values (4000 for each node in the case of MCMC) from the model, we obtained the marginal distribution of the coefficients. In the case of the INLA framework, our conversion script was based on the mean values of the distribution in each node, resulting in single coefficient values from the FFT, in contrast to the MCMC distribution. The previously mentioned challenge was selecting the optimal degree of the Chebyshev series, which subsequently determined the calculation of the GP in the appropriate number of Chebyshev nodes. The method for estimating the optimal degree involved evaluating the behavior of coefficient values based on the degree. Consequently, we assessed the significance of the coefficients to determine a cutoff threshold beyond which the remaining coefficients could be considered negligible. This threshold could be reached when values approached machine precision or oscillated around a value. In our case, we decided to use the value based on our noise variance and set the threshold to $t = 0.1 \cdot \sigma^2$. This step reduces the Chebyshev series degree. Since we were dealing with distributions, we used the expected values of the coefficients to optimize the series order needed to accurately represent the original function.

The results of conducted tests for different orders, ranging from 10 to 200, can be seen below. We gathered the plots of results from models (mean values with confidence interval) and coefficient values together. In Figures 6 and 7, we can see results for the MCMC sampling approach with the use of the Matérn kernel with at least twice differentiable functions. Figures 8 and 9 show the results of the same model but for RBF kernels and infinitely differentiable functions. The results from the INLA framework can be seen in Figures 10 and 11 for at least twice differentiable functions and in Figures 12 and 13 for infinitely differentiable functions. We also gathered information about the average computing time for each experiment in different combinations of kernels, frameworks, and Chebyshev node counts; they can be seen in Table 1.

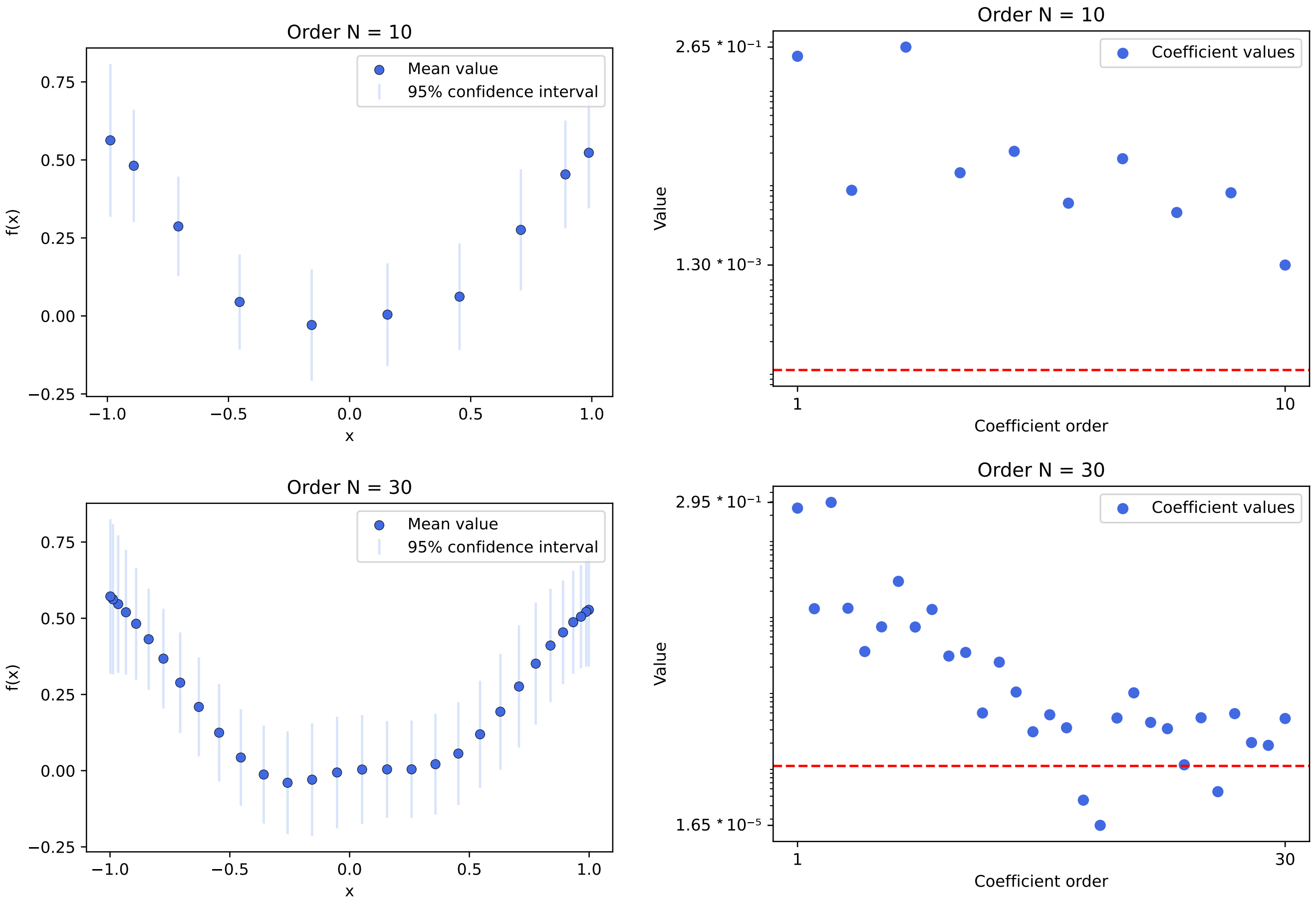

Figure 6. The plots present results for the experiment conducted using the *MCMC sampling* method, with a Gaussian process model that employs a *Matérn kernel*. Experiments use sample data from selected functions and the random variables are set in *10 and 30 Chebyshev nodes*. The left plots show the mean values of calculated samples as well as the 95% confidence interval received from the sampled distribution. The right plots show the behavior of Chebyshev series coefficient values compared to their order. The red line shows the selected threshold for optimization $(0.1 \cdot \sigma^2)$. We can deduce that both results are nowhere near machine precision, so we should rely on the selected threshold. In the case of only 10 Chebyshev nodes, we did not even reach the threshold value, whereas with 30 nodes, sample number 18 did. As a result of our optimization, we should take the value around 18 as the order of the Chebyshev series. Moreover, the computation times for the small number of nodes were acceptable for real-life cases, ranging from several seconds to a dozen seconds.

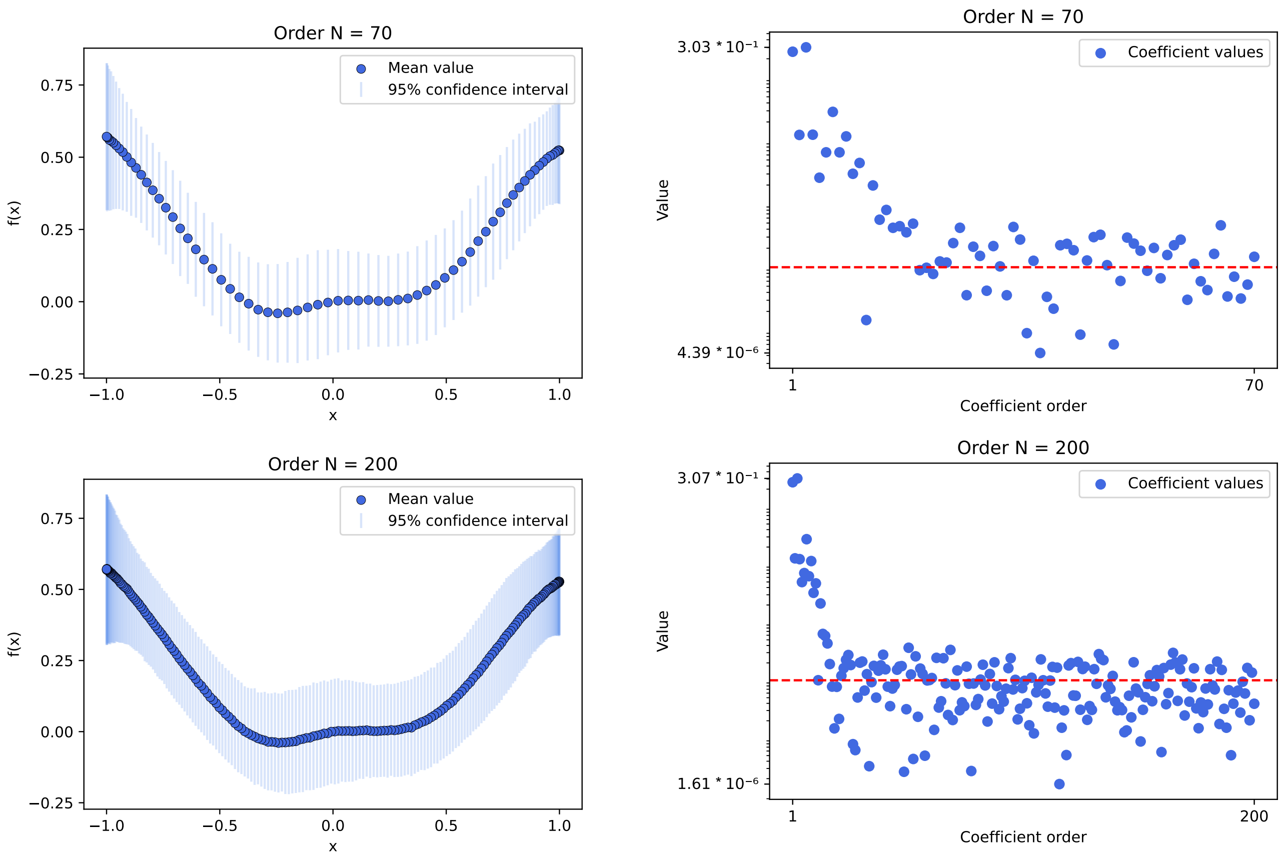

Figure 7. The plots present results for the experiment conducted using the *MCMC sampling* method, with a Gaussian process model that employs a *Matérn kernel*. Experiments use sample data from selected functions and the random variables are set in *70 and 200 Chebyshev nodes*. The left plots show the mean values of calculated samples as well as the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). We can deduce that oscillation around the threshold starts early, around the order of 30, so our previous assumption of selecting the 18th order should be correct. Moreover, the values decrease only slightly up to the order of 10^6, which suggests that they probably won't reach machine precision very soon. These experiments were conducted primarily to observe the behavior after reaching the threshold and to verify if the previous assumption was correct. Additionally, they allowed us to check the computation times for the number of nodes. Unfortunately, these times were unacceptable, ranging from several minutes to 10 min.

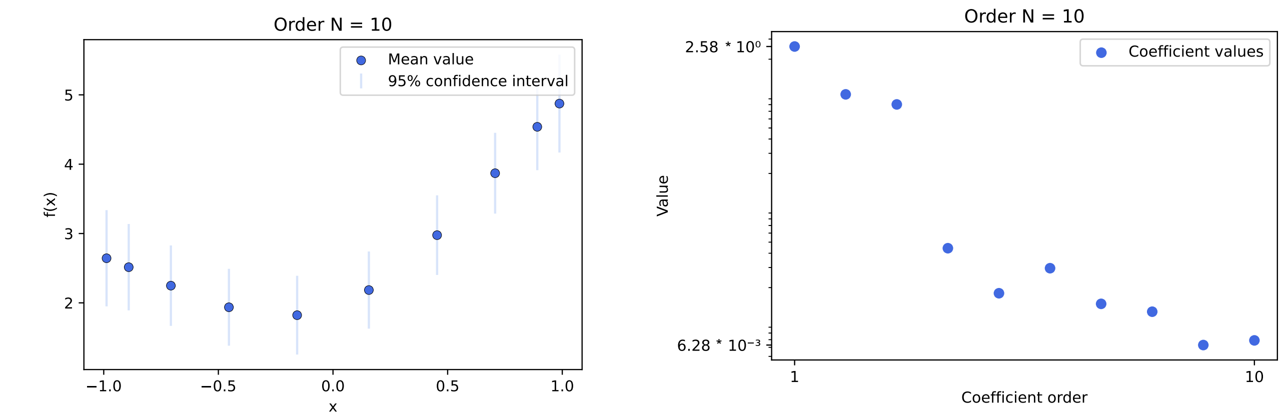

Figure 8. *Cont.*

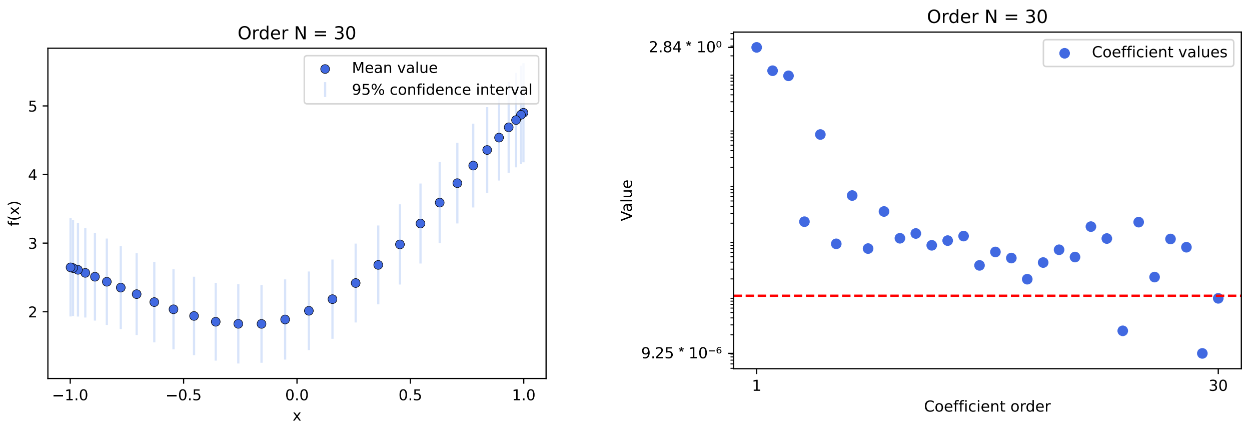

Figure 8. The plots present results for the experiment conducted using the *MCMC sampling* method, with a Gaussian process model that employs an *RBF kernel*. Experiments use sample data from a selected function, with random variables set at *10 and 30 Chebyshev nodes*. The left plots show the mean values of calculated samples as well as the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). The first 10 coefficients are highly informative; none reached the selected threshold. When we increased the number of selected nodes, it was clear that the 24th coefficient crossed the threshold. Contrary to the previous case, this function required more coefficients to gather sufficient information based on our assumptions. Moreover, as with the other MCMC case, computation times for this number of nodes were acceptable and similar to the Matérn ranging from several seconds to a maximum of a dozen seconds.

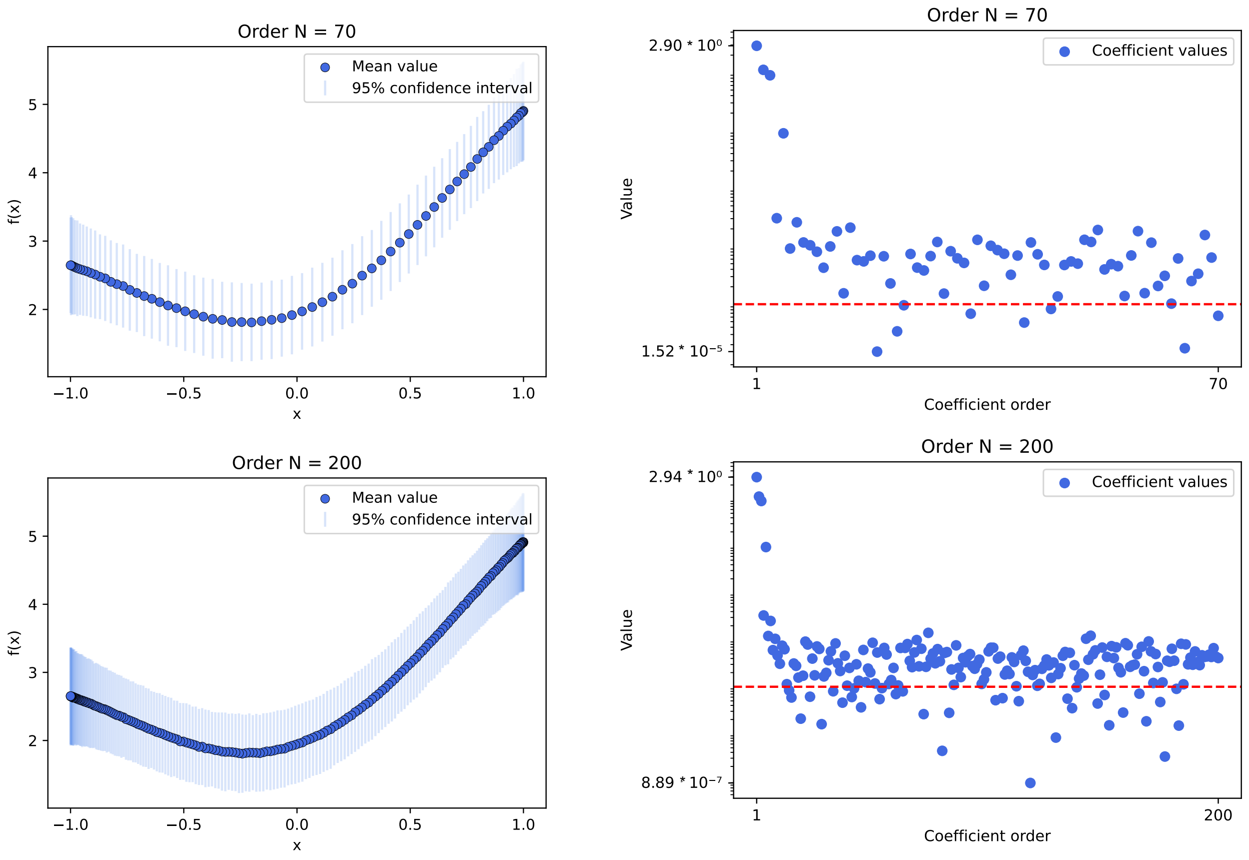

Figure 9. The plots present results for the experiment conducted using the *MCMC sampling* method, with a Gaussian process model that employs an *RBF kernel*. The experiments use sample data from a selected function, with random variables set at *70 and 200 Chebyshev nodes*. The left plots show the

mean values of calculated samples, as well as the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). We can deduce that adding more nodes to the GP model does not actually help gather more information. Oscillations around the threshold start around the order of 30 and do not seem to reach machine precision. Additionally, the computation times for this number of nodes were unacceptable, ranging from several minutes to a dozen minutes.

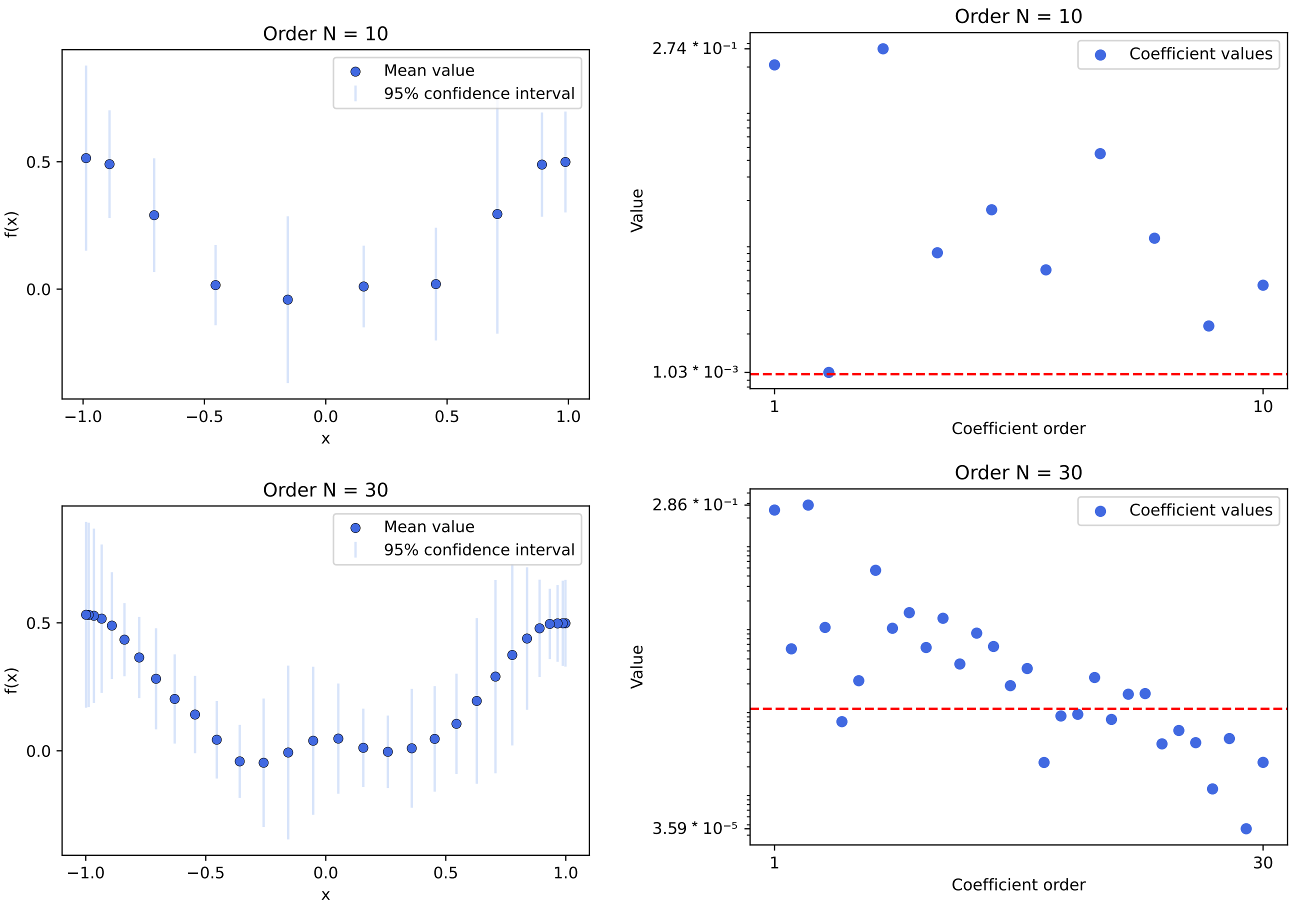

Figure 10. The plots present results for the experiment conducted using the *INLA framework* method, with a Gaussian process model that employs a *Matérn kernel*. Experiments utilized sample data from a selected function with random variables set at *10 and 30 Chebyshev nodes*. The left plots show the mean values of calculated samples as well as the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values in relation to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). Contrary to MCMC experiments, the INLA framework, in this case, managed to catch the threshold at the 2nd order, highlighting the importance of checking higher orders, especially since no other coefficient reached the selected threshold. For the 30th order, the 5th sample could be considered as the cutoff for optimization, but the next value below this is the 16th coefficient, with a clearly visible decreasing trend between the 6th and 16th coefficients. We can deduce that oscillation slightly starts around the order of 17, but we need to verify this with higher orders. The computation times for the node numbers using INLA were excellent and completed within several seconds; this was predictable due to the INLA approximation properties.

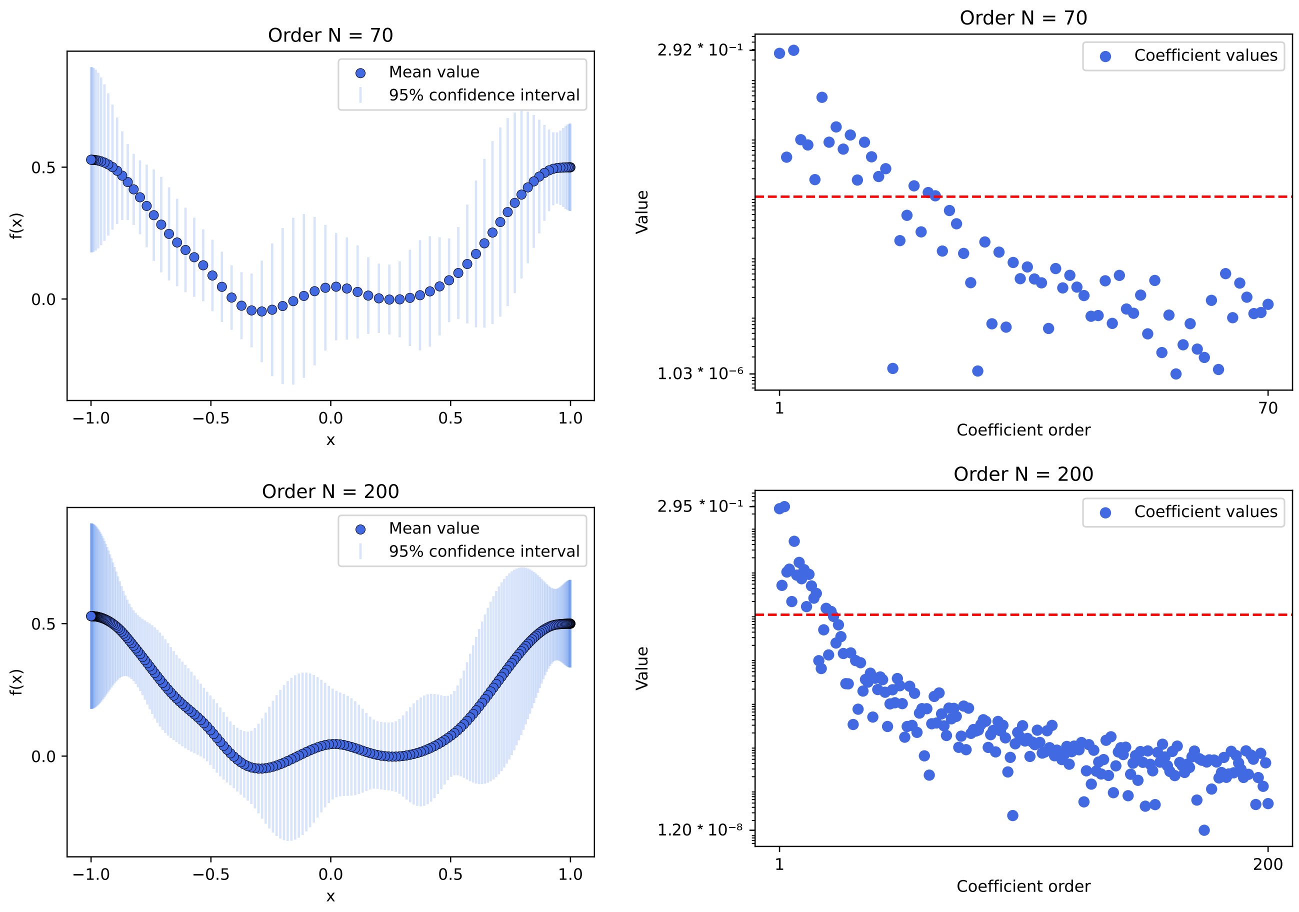

Figure 11. The plots present results for the experiment conducted using the *INLA framework* method, with a Gaussian process model that employs a *Matérn kernel*. Experiments utilize sample data from a selected function, with random variables set at *70 and 200 Chebyshev nodes*. The left plots show the mean values of calculated samples along with the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). Based on previous experiments with a lesser value or order and the information gathered here, we can deduce that values truly cross the threshold around the order of 20. Moreover, coefficient values are slowly decreasing and do not start to oscillate, but in our case, it makes no sense to aim for values below the threshold. As expected, due to the INLA approximation properties, the computation times for any number of nodes were excellent, conducted within several seconds.

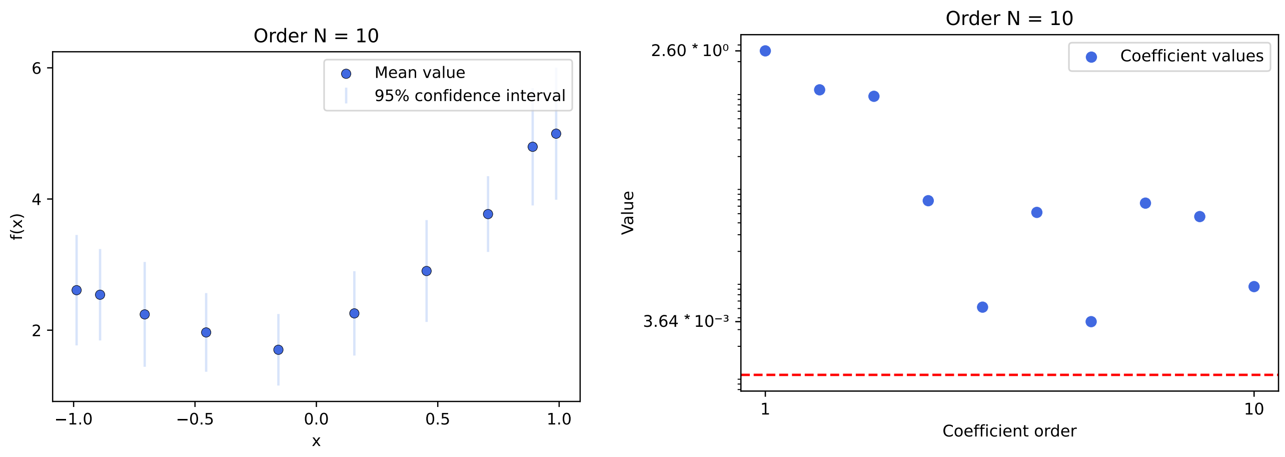

Figure 12. *Cont.*

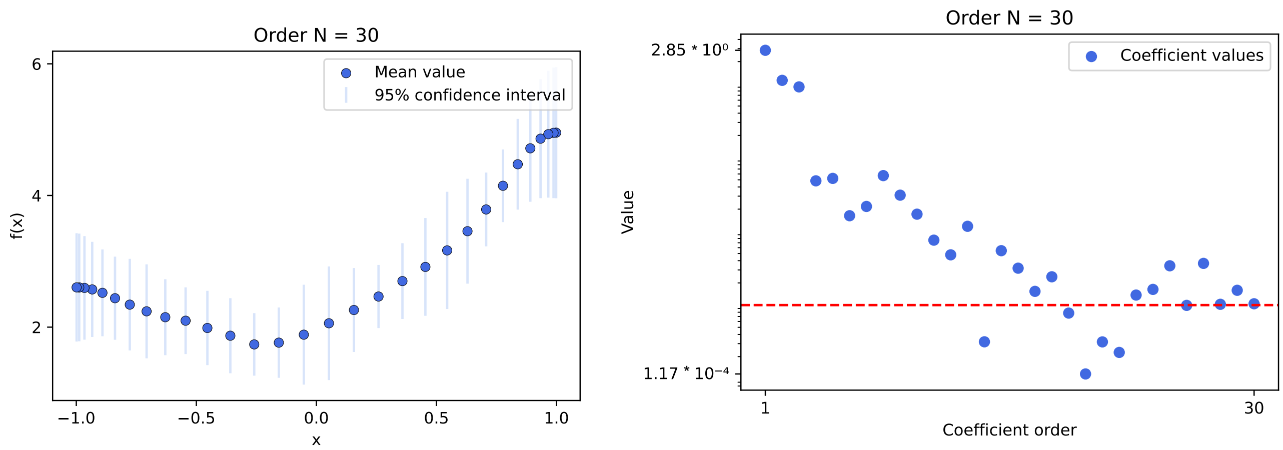

Figure 12. The plots present results for the experiment conducted using the *INLA framework* method, with a Gaussian process model that employs a modified Matérn kernel, simulating the *RBF* one. Experiments utilized sample data from a selected function with random variables set at *10 and 30 Chebyshev nodes*. The left plots show the mean values of calculated samples along with the 95% confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization $(0.1 \cdot \sigma^2)$. In this case, the INLA framework mimics the behavior observed in previous cases with low-order coefficients, where values reach the threshold quickly but continue to show a decreasing trend. We can deduce that the required value is likely around the order of 20. The computation times for this number of nodes were excellent, conducted within several seconds.

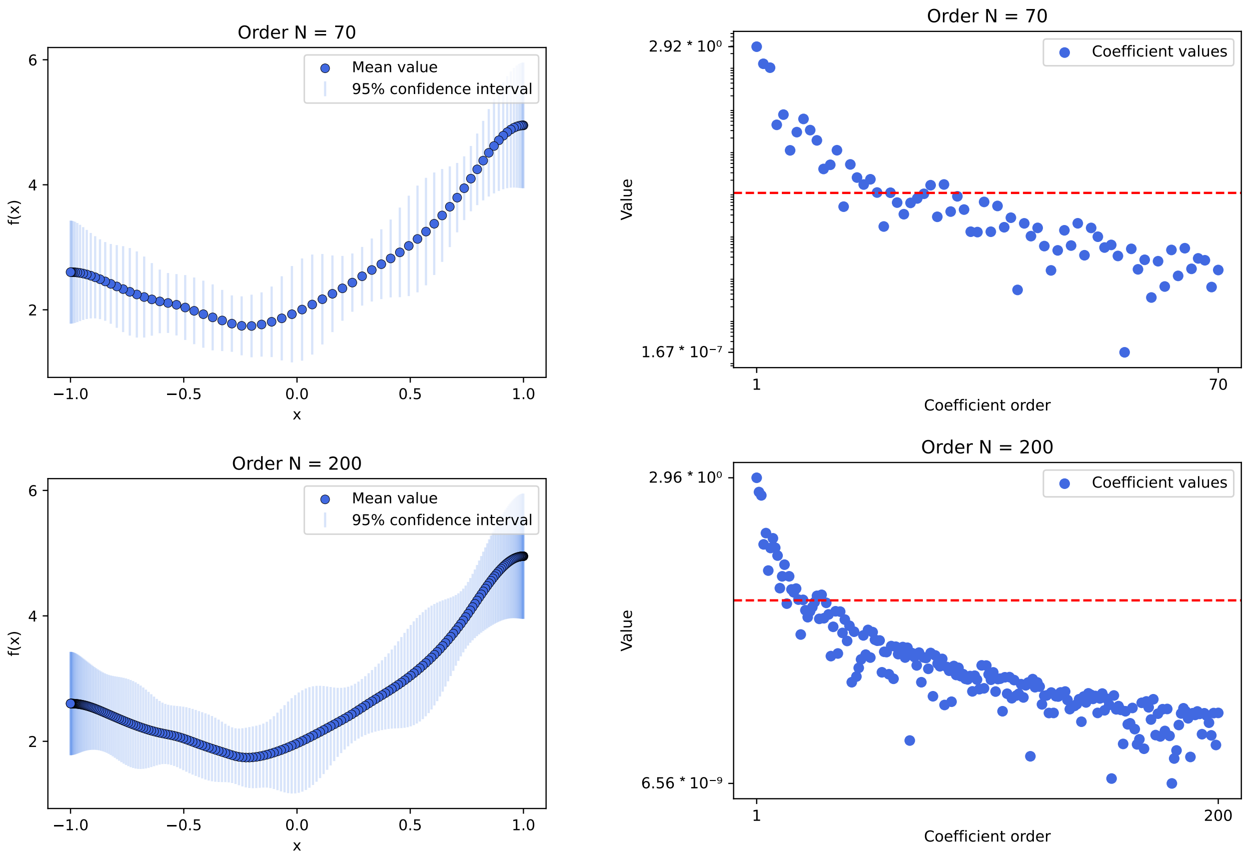

Figure 13. The plots present results for the experiment conducted using the *INLA framework* method, with a Gaussian process model that employs a modified Matérn kernel, simulating the *RBF* one. Experiments utilized sample data from selected functions, with random variables set at *70 and 200 Chebyshev nodes*. The left plots show the mean values of calculated samples along with the 95%

confidence interval derived from the sampled distribution. The right plots illustrate the behavior of Chebyshev series coefficient values relative to their order. A red line indicates the selected threshold for optimization ($0.1 \cdot \sigma^2$). We can deduce that all INLA cases behave similarly: initially, the values decrease rapidly, reaching the threshold, and then the downward trend slows down. The computation times for this number of nodes were excellent, conducted within several seconds.

Table 1. The table presents average computation times for various selected experiments using different numbers of samples: 10, 20, 30, 50, 70, 100, 150, and 200 for Matérn and RBF kernels, using MCMC and INLA frameworks. The time for MCMC methods significantly increases depending on the number of samples for the GP model, whereas for INLA, it remains practically unchanged. This is primarily due to the initial assumption of optimizing computations for INLA. The kernel choice did not have an impact on the computation time.

Samples	**MCMC: Matérn**	**MCMC: RBF**	**INLA: Matérn**	**INLA: RBF**
10	9.6 s	10.1 s	3.1 s	2.4 s
20	15 s	12 s	2.5 s	3 s
30	24.6 s	27.8 s	3.5 s	3.1 s
50	40 s	51 s	2.7 s	3 s
70	1 min 40 s	2 min 10 s	2.7 s	2.8 s
100	3 min 11 s	2 min 51 s	3 s	2.9 s
150	6 min 40 s	7 min 51 s	3.1 s	2.7 s
200	10 min 30 s	13 min 11 s	3.8 s	4 s

The overall conclusion regarding MCMC approaches is that there is a clear barrier to the optimal degree level of the Chebyshev series. A clear cut-off line of the threshold of useful information can be used but can also be achieved by catching the oscillations around the same value of the series coefficient. For the case of the Matérn kernel, the value of oscillations is slightly below the threshold, and for the RBF kernel, it is a bit above the threshold value. Both approaches provide us with the value of nodes we should use; both were around 10–30 orders. It is important to check the optimization of the order because any decrease in these values directly impacts the size of the covariance matrix and, thus, the computation time, losing very little information.

In the case of the R-INLA framework approach, we can conclude that despite achieving high Chebyshev degree values for the coefficients and observing a decreasing tendency, we do not reach machine precision, and the values do not oscillate around any value. In such a situation, we can use the selected threshold, which was useful in the previous experiment with MCMC, where the optimal cut-off values of the necessary coefficients were identified. Moreover, it is important to understand that the INLA framework is optimized to calculate Gaussian random fields (multivariate GPs), so decreasing the number of nodes does not significantly impact execution time.

This difference in results comes from a different calculation method. MCMC uses sampling at each point, so we receive more information, such as the values of each generated sample and chain, from which probability distributions are later calculated. Due to this possibility, MCMC takes into account entire value vectors, not just distribution information, for calculations related to the Chebyshev series. This allows us to optimize the degree of the series with FFT conversion. In the case of R-INLA, the SPDE approach only provides information about the distribution at given points. Therefore, we can only use data on the expected value and standard deviation when calculating values for the Chebyshev series. It is also important to consider the execution times for model fitting and data generation. MCMCs are known for their long and complex calculations but for cases involving 10 to 50 random variables, the execution time did not exceed several dozen seconds. However, for 200 variables, it took almost 15 min on a moderate computer. We did not encounter a similar problem with R-INLA, as it is optimized for calculations on sparse matrices; the execution times for all attempts did not exceed several seconds.

By employing the selected Chebyshev nodes, we obtained the expected results of the GP model in the form of Chebyshev series coefficients, rather than computing the GP at every point. It is also crucial to emphasize that, regardless of the chosen framework or the degree of the results, all are presented in the form of coefficients. This form allows us to encode information in the form of the Chebyshev series itself, which serves as an approximation of the original function and can help in generating more data for generative models. Thus, it carries additional information about the original function, unlike other frequently used methods. It is also important to note that, unlike MCMC, R-INLA is already an approximation method, so the use of subsequent optimizations for simple one-dimensional cases did not bring much improvement. However, in the case of the MCMC approach, our method can facilitate its use in scenarios where it was previously infeasible, especially where generative models are needed.

4. Discussion

In the field of computational process optimization for GPs, one of the more significant and recognizable solutions is sparse GP. This approach involves strategically selecting a subset of informative points, known as inducing points, to represent the underlying data. The main advantage lies in addressing the computational challenges associated with traditional GPs, particularly when dealing with large datasets. By employing sparse GPs, the model's scalability is greatly improved, allowing for efficient processing of extensive datasets while maintaining the flexibility and power of GPs in capturing complex relationships [16,17].

One way to enhance the performance in representing underlying functions through sparse models of GPs is through rectangularization. In situations where data are sparse, avoiding overfitting and optimizing Gaussian process regression (GPR) hyperparameters becomes challenging, especially in high-dimensional spaces where data sparsity cannot be practically resolved by adding more data. The success or failure of GPR applications hinges on the optimal selection of hyperparameters. The authors propose a method that facilitates parameter optimization through the rectangularization of the GPR-defining equation. They use a 15-dimensional molecular potential energy surface as an example; they illustrate the effectiveness of this approach in achieving hyperparameter tuning even with very sparse data [44].

Other works focus on optimizing the performance of online updating for sparse GPs. Research presented in [45] delves into the relationship between a class of sparse-inducing point GP regression methods and Bayesian recursive estimation, enabling Kalman filter-like updates for online learning. Unlike prior focus on the batch setting, this study concentrates on mini-batch training. Leveraging the Kalman filter formulation, the proposed approach recursively propagates analytical gradients of the posterior over mini-batches, resulting in faster convergence and superior performance. The approach presented in [46] was developed for both the fully independent training conditional (FITC) and the partially independent training conditional (PITC) approximations, enabling the inclusion of a new measurement point $(x_n + 1)$ in $O(m^2)$ time, where m represents the size of the set of inducing inputs. The online nature of the algorithms enables the forgetting of earlier measurement data, resulting in a memory space requirement of $O(m^2)$ for both FITC and PITC.

Works that do not rely on sparse GPs can also be found. In [12], the authors propose a way to optimize GPs for time series gap-filling in satellite image processing by replacing the per-pixel optimization step with cropland-based pre-calculations for GPR hyperparameters θ.

In [47], the authors present a novel algorithm in terms of GP optimization: BKB (budgeted kernelized bandit). It is designed for optimization under bandit feedback. BKB achieves near-optimal regret and convergence rates with nearly constant per-iteration complexity. Notably, it makes no assumptions about the input space or GP covariance. The algorithm combines a kernelized linear bandit approach (GP-UCB) with randomized matrix sketching using leverage score sampling. The authors demonstrate that randomly sampling inducing points based on posterior variance accurately provides a low-rank approximation of the GP, maintaining reliable variance estimates and confidence intervals.

In the last example, the authors explore an alternative global optimization framework called optimistic optimization, which is computationally more efficient. This approach leverages prior knowledge of the search space's geometry through a dissimilarity function. The study aims to integrate the conceptual advantages of Bayesian Optimization with the computational efficiency of optimistic optimization. This is achieved by mapping the kernel to a dissimilarity, resulting in the development of an optimistic optimization algorithm [48].

The practical application of the proposed solution can be easily illustrated through existing examples, such as in the previously mentioned problem of anomaly detection in industrial processes [29]. This problem involves several steps: data acquisition, generating new samples using g = Gaussian processes (GPs), and classification through data depth. While other aspects of the issue would remain unchanged, for the generative model, our presented solution could be applied. In the original problem, the authors used a GP implementation based on the ML-II solution. By applying our algorithm, MCMC sampling could be used in such a problem, potentially providing more information about the generation of functions. Another example is the previously discussed issue of using a Gaussian mixture in diagnosing the state of an engine based on its startup characteristics [35]. Here, the authors employ several models aimed at generating new samples for classification. As in the previous case, it would be valuable to explore the application of our proposal, which relies on optimally selecting the degree of the Chebyshev series and Chebyshev points as random variables, thereby enabling the use of MCMC sampling.

As presented by employing various methods, including sparse GP variations, we can address the issue of time-consuming computations in GP modeling. In contrast to the standard sparse approach, our proposal relies on an innovative method characterized by two key features: a reduction in computation time and the ability to reconstruct the original function. The first feature is ensured through the imposition of domain constraints in computations, where we model the GP using a limited set of random variables at selected Chebyshev nodes. The second feature arises from the form of the Chebyshev series, achieved by transforming function values at Chebyshev nodes using FFT, allowing us to obtain coefficients of the Chebyshev series. This representation provides information about the original function. Additionally, by analyzing coefficients of different degrees, we can guarantee a minimal, meaningful number of Chebyshev nodes. Our experiments confirm that our algorithm significantly reduces computation time, especially in the context of sampling-based methods, while maintaining unique properties not encountered in standard sparse approaches.

5. Conclusions

The incorporation of the Chebyshev property to enhance the computation of the GP is a topic frequently overlooked. Over the years, one of the primary challenges associated with GPs has been their computational complexity, particularly when dealing with a large number of points. This complexity is exacerbated when employing the MCMC method for sample generation, often resulting in impractical computation times.

In response to this issue, our proposed approach aims to mitigate the computational burden by reducing the calculations of random variables to the appropriate number of Chebyshev nodes. Moreover, our study focuses on leveraging the values at Chebyshev nodes (and their marginal distribution) gathered from the generative model to construct the Chebyshev series. Chebyshev series are recognized for their strong convergence characteristics, especially in relation to differentiable functions produced by the model. Computing the values of the Chebyshev series can be a highly challenging task so we are using the Chebyshev interpolant as a substitute for the series itself with just one bit loss of accuracy. With conversion based on FFT, we can manage to transform distributions (values) of random variables in Chebyshev nodes to Chebyshev series coefficients. The critical challenge addressed was the judicious selection of the optimal degree for the Chebyshev series, a parameter that influences the computation of the GP within an appropriate number of Chebyshev nodes. To determine the optimal degree, we employed a method that involved assessing the behavior of coefficient values based on the degree, subsequently establishing a cutoff threshold based on the significance of coefficients. This threshold was identified when values approached machine precision or exhibited oscillations around a similar value, leading to a reduction in the Chebyshev series degree.

Our proposed solution is also characterized by the feature of receiving more information about the original function. By transforming the function values at Chebyshev nodes using FFT, we can reduce the amount of Chebyshev points and obtain the Chebyshev interpolation of the generated function. This form of Chebyshev representation can help us generate more variations from the model, with respected uncertainty from the GP model. This approach works very well for generative models. These models are designed to map new data samples based on collected information from the original function. Using MCMC sampling in such cases is useful and our approach allows its application in a reasonable time. Moreover, thanks to the use of the after-mentioned Chebyshev interpolation, we have the possibility of transferring additional information about the function through appropriate coefficients.

From numerical experiments, we can conclude significant differences between the results between MCMC and R-INLA frameworks. These differences arise from their distinct calculation methods. MCMC, using sampling at each point, considers entire value vectors, providing more information for Chebyshev series calculations. This enables the optimization of the series degree. On the other hand, R-INLA, utilizing an SPDE approach, offers information solely about the distribution at given points, limiting the utilization of data to expected values and standard deviations in the context of calculating values for the Chebyshev series.

Furthermore, considering the execution times of the algorithm is crucial. MCMC, known for lengthy and intricate calculations, exhibited significant variations. For 10 random variables, the execution time remained within several seconds, whereas for 200, it extended to about 11 min. In contrast, R-INLA, designed for optimized and approximated calculations on sparse matrices, consistently demonstrated shorter execution times, not exceeding one minute in all attempts. The utilization of the selected Chebyshev nodes in our script facilitated the acquisition of expected results for the GP generative model in the form of Chebyshev series coefficients, avoiding the need for computing the GP at every individual point. Preliminary results indicate a substantial reduction in calculation time in the case of MCMC-based calculations. In both cases, it is possible to utilize a reduced number of nodes for computations without significant loss in the structures of the reconstructed functions from the models. This promising outcome underscores the potential efficiency gains achievable through our proposed method, especially in terms of an MCMC-based approach. Our solution allows us to use the MCMC sampling in cases where it was not possible before due to the problem of computation time. For the R-INLA case, the reduction in computational time is not as spectacular as in the MCMC case, but that outcome should be expected from the already approximate calculations; therefore, the use of the method may seem unnecessary.

It is worth noting that our suggested approach shows promise but necessitates further testing and validation on real case scenarios. Future considerations should delve into exploring alternative, less frequently used covariance functions for GPs, adjusting priors and model settings to better align with specific problem nuances, and implementing scalability measures to ensure the versatility and applicability of the proposed solution across diverse domains. As our research progresses, it is imperative to validate and refine the proposed approach to contribute meaningfully to the optimization of GP computations.

Author Contributions: Conceptualization, A.D. and J.B.; methodology, J.B.; resources, A.D. and J.B.; writing—original draft preparation, A.D.; writing—review and editing, A.D. and J.B.; visualization, A.D.; supervision, J.B.; project administration, J.B.; funding acquisition, J.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by AGH's Research University Excellence Initiative under the project "Interpretable methods of process diagnosis using statistics and machine learning" and by the Polish National Science Centre project "Process Fault Prediction and Detection", contract no. UMO-2021/41/B/ST7/03851.

Data Availability Statement: Data are contained within the article

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GP	Gaussian process
MCMC	Markov chain Monte Carlo
INLA	integrated nested Laplace approximation
FFT	fast Fourier transform
GPR	Gaussian process regression
HMC	Hamiltonian Monte Carlo
GMRF	Gaussian Markov random field
RBF	radial basis function
SE	squared exponential
SPDE	stochastic partial differential equation
FITC	fully independent training conditional
PITC	partially independent training conditional
BKB	budgeted kernelized bandit

References

1. Davis, R.A. Encyclopedia of Environmetrics, Gaussian Process. In *Encyclopedia of Environmetrics*; American Cancer Society: Hoboken, NJ, USA, 2006. [CrossRef]
2. Rasmussen, C.; Williams, C.K.I. *Gaussian Processes in Machine Learning*; MIT Press: Cambridge, MA, USA, 2006.
3. Dudek, A.; Baranowski, J. Gaussian Processes for Signal Processing and Representation in Control Engineering. *Appl. Sci.* **2022**, *12*, 4946. [CrossRef]
4. Zeng, A.; Ho, H.; Yu, Y. Prediction of building electricity usage using Gaussian Process Regression. *J. Build. Eng.* **2020**, *28*, 101054. [CrossRef]
5. Gogolashvili, D.; Kozyrskiy, B.; Filippone, M. Locally Smoothed Gaussian Process Regression. *Procedia Comput. Sci.* **2022**, *207*, 2717–2726. [CrossRef]
6. Das, S.; Roy, S.; Sambasivan, R. Fast Gaussian Process Regression for Big Data. *Big Data Res.* **2018**, *14*, 12–26. [CrossRef]
7. Rodrigues, F.; Pereira, F.; Ribeiro, B. Gaussian Process Classification and Active Learning with Multiple Annotators. In Proceedings of the 31st International Conference on Machine Learning, PMLR, Bejing, China, 22–24 June 2014; Xing, E.P., Jebara, T., Eds.; PMLR: New York, NY, USA, 2014; Volume 32, pp. 433–441.
8. Ibna Kaisar, T.; Zaman, K.; Khasawneh, M.T. A New Approach to Probabilistic Classification Based on Gaussian Process and Support Vector Machine. *Comput. Ind. Eng.* **2023**, *174*, 109719. [CrossRef]
9. Gonçalves, G.; Gomes, D.; Leoni, G.; Rosendo, D.; Moreira, A.; Kelner, J.; Sadok, D.; Endo, P. Optimizing the Cloud Data Center Availability Empowered by Surrogate Models. In Proceedings of the 53rd Hawaii International Conference on System Sciences, Maui, HI, USA, 7–10 January 2020; Hawaii International Conference on System Sciences; HICSS 2020. [CrossRef]
10. Sanabria-Borbón, A.C.; Soto-Aguilar, S.; Estrada-López, J.J.; Allaire, D.; Sánchez-Sinencio, E. Gaussian-Process-Based Surrogate for Optimization-Aided and Process-Variations-Aware Analog Circuit Design. *Electronics* **2020**, *9*, 685. [CrossRef]
11. Li, Y.; Zheng, Y. Citywide Bike Usage Prediction in a Bike-Sharing System. *IEEE Trans. Knowl. Data Eng.* **2019**, *32*, 1079–1091. [CrossRef]
12. Belda, S.; Pipia, L.; Morcillo-Pallarés, P.; Verrelst, J. Optimizing Gaussian Process Regression for Image Time Series Gap-Filling and Crop Monitoring. *Agronomy* **2020**, *10*, 618. [CrossRef]
13. Gönül, M.; Kutlar, O.A.; Calik, A.T.; Orcun Parlak, F. Prediction of oil dilution formation rate due to post injections in diesel engines by using Gaussian process. *Fuel* **2021**, *305*, 121608. [CrossRef]
14. Liu, H.; Ong, Y.S.; Shen, X.; Cai, J. When Gaussian Process Meets Big Data: A Review of Scalable GPs. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *31*, 4405–4423. [CrossRef] [PubMed]

15. Liu, H.; Cai, J.; Ong, Y.S.; Wang, Y. Understanding and comparing scalable Gaussian process regression for big data. *Knowl.-Based Syst.* **2019**, *164*, 324–335. [CrossRef]
16. Bottou, L.; Chapelle, O.; DeCoste, D.; Weston, J. Approximation Methods for Gaussian Process Regression. In *Large-Scale Kernel Machines*; MIT Press: Cambridge, MA, USA, 2007; pp. 203–223.
17. Quiñonero-candela, J.; Rasmussen, C.E.; Herbrich, R. A unifying view of sparse approximate Gaussian process regression. *J. Mach. Learn. Res.* **2005**, *6*, 2005.
18. Gómez Rubio, V.; Rue, H. Markov Chain Monte Carlo with the Integrated Nested Laplace Approximation. *Stat. Comput.* **2018**, *28*, 1033–1051. [CrossRef]
19. Lindgren, F.; Rue, H. Bayesian spatial modelling with R-INLA. *J. Stat. Softw.* **2015**, *63*, 1–25. [CrossRef]
20. Van Niekerk, J.; Krainski, E.; Rustand, D.; Rue, H. A new avenue for Bayesian inference with INLA. *Comput. Stat. Data Anal.* **2023**, *181*, 107692. [CrossRef]
21. Lv, F.; Yang, G.; Zhu, W.; Liu, C. Generative classification model for categorical data based on latent Gaussian process. *Pattern Recognit. Lett.* **2017**, *92*, 56–61. [CrossRef]
22. Adams, R.P.; Murray, I.; MacKay, D.J.C. The Gaussian Process Density Sampler. In Proceedings of the 21st International Conference on Neural Information Processing Systems (NIPS'08), Red Hook, NY, USA, 8–10 December 2008; pp. 9–16.
23. Kawashima, T.; Hino, H. Gaussian Process Koopman Mode Decomposition. *Neural Comput.* **2023**, *35*, 82–103. [CrossRef] [PubMed]
24. Riutort-Mayol, G.; Bürkner, P.C.; Andersen, M.R.; Solin, A.; Vehtari, A. Practical Hilbert space approximate Bayesian Gaussian processes for probabilistic programming. *arXiv* **2020**, arXiv:2004.11408. https://doi.org/10.48550/ARXIV.2004.11408.
25. Simpson, D. Un Garçon pas Comme les Autres (Bayes): Yes but What Is a Gaussian Process? or, Once, Twice, Three Times a Definition; or A Descent into Madness. 2021. Available online: https://dansblog.netlify.app/posts/2021-11-03-yes-but-what-is-a-gaussian-process-or-once-twice-three-times-a-definition-or-a-descent-into-madness/yes-but-what-is-a-gaussian-process-or-once-twice-three-times-a-definition-or-a-descent-into-madness.html (accessed on 20 November 2023).
26. Garnett, R. *Bayesian Optimization*; Cambridge University Press: Cambridge, UK, 2022.
27. Blum, M.; Riedmiller, M. Optimization of gaussian process hyperparameters using Rprop. In Proceedings of the ESANN 2013 Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, Bruges, Belgium, 24–26 April 2013; pp. 339–344.
28. Raes, W.; Dhaene, T.; Stevens, N. On The Usage of Gaussian Processes for Visible Light Positioning With Real Radiation Patterns. In Proceedings of the 2021 17th International Symposium on Wireless Communication Systems (ISWCS), Berlin, Germany, 6–9 September 2021; pp. 1–6. [CrossRef]
29. Baranowski, J.; Dudek, A.; Mularczyk, R. Transient Anomaly Detection Using Gaussian Process Depth Analysis. In Proceedings of the 2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR), Amber Baltic Hotel, Miedzyzdroje, Poland, 23–26 August 2021; pp. 221–226. [CrossRef]
30. Dudek, A.; Baranowski, J. Modelling of Li-Ion battery state-of-health with Gaussian processes. *Arch. Electr. Eng.* **2023**, *72*, 643–659. [CrossRef]
31. Gammelli, D.; Rodrigues, F.; Pacino, D.; Kurtaran, H.A.; Pereira, F.C. A Machine Learning Approach to Censored Bike-Sharing Demand Modeling. In Proceedings of the Transportation Research Board, Annual Meeting Proceedings, Transportation Research Board National Cooperative Highway Research Program, Walter E. Washington Convention Center, Washington, DC, USA, 12–16 January 2020; Volume 2020.
32. Zhang, L.; Liu, J.; Jiang, H.; Guan, Y. SensTrack: Energy-Efficient Location Tracking with Smartphone Sensors. *IEEE Sens. J.* **2013**, *13*, 3775–3784. [CrossRef]
33. Cheng, L.; Ramchandran, S.; Vatanen, T.; Lietzén, N.; Lahesmaa, R.; Vehtari, A.; Lähdesmäki, H. An additive Gaussian process regression model for interpretable non-parametric analysis of longitudinal data. *Nat. Commun.* **2019**, *10*, 1798. [CrossRef] [PubMed]
34. Lamb, A. A Brief Introduction to Generative Models. *arXiv* **2021**, arXiv:2103.00265. http://arxiv.org/abs/2103.00265.
35. Jarzyna, K.; Rad, M.; Piatek, P.; Baranowski, J. Bayesian Fault Diagnosis for Induction Motors During Startup in Frequency Domain. In *Lecture Notes in Networks and Systems, Proceedings of the Advanced, Contemporary Control—Proceedings of the XXI Polish Control Conference, Gliwice, Poland, 26–29 June 2023, Volume 2*; Pawelczyk, M., Bismor, D., Ogonowski, S., Kacprzyk, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2023; Volume 709, pp. 14–24. [CrossRef]
36. Trefethen, L.N. *Approximation Theory and Approximation Practice*; SIAM Philadelphia USA: Philadelphia, PA, USA, 2012; pp. I–VII, 1–305.
37. Ahmed, N.; Fisher, P. Study of algorithmic properties of chebyshev coefficients. *Int. J. Comput. Math.* **1968**, *2*, 307–317. [CrossRef]
38. Monterrubio-Gómez, K.; Wade, S. On MCMC for variationally sparse Gaussian processes: A pseudo-marginal approach. *arXiv* **2021**, arXiv:2103.03321. http://arxiv.org/abs/2103.03321.
39. Stan Development Team. Stan User's Guide. 2024. Available online: https://mc-stan.org/users/documentation/ (accessed on 5 January 2024).
40. Rue, H.; Martino, S.; Chopin, N. Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations. *J. R. Stat. Soc. Ser. B* **2009**, *71*, 319–392. [CrossRef]

41. Rue, H.; Held, L. Gaussian Markov Random Fields: Theory and Applications. In *Gaussian Markov Random Fields*; Chapman and Hall/CRC: New York, NY, USA, 2005; Volume 104. [CrossRef]
42. Krainski, E.; Gómez Rubio, V.; Bakka, H.; Lenzi, A.; Castro-Camilo, D.; Simpson, D.; Lindgren, F.; Rue, H. *Advanced Spatial Modeling with Stochastic Partial Differential Equations Using R and INLA*; Chapman and Hall/CRC: New York, NY, USA, 2018. [CrossRef]
43. Mark Richardson. Chebfun and FFT Example. 2024. Available online: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/23972/versions/22/previews/chebfun/examples/approx/html/ChebfunFFT.html (accessed on 6 November 2023).
44. Manzhos, S.; Ihara, M. Rectangularization of Gaussian process regression for optimization of hyperparameters. *Mach. Learn. Appl.* **2023**, *13*, 100487. [CrossRef]
45. Schürch, M.; Azzimonti, D.; Benavoli, A.; Zaffalon, M. Recursive estimation for sparse Gaussian process regression. *Automatica* **2020**, *120*, 109127. [CrossRef]
46. Bijl, H.; van Wingerden, J.W.; B. Schön, T.; Verhaegen, M. Online sparse Gaussian process regression using FITC and PITC approximations. *IFAC-PapersOnLine* **2015**, *48*, 703–708. [CrossRef]
47. Calandriello, D.; Carratino, L.; Lazaric, A.; Valko, M.; Rosasco, L. Gaussian Process Optimization with Adaptive Sketching: Scalable and No Regret. In *Proceedings of Machine Learning Research, Proceedings of the Thirty-Second Conference on Learning Theory, Phoenix, AZ, USA, 25–28 June 2019*; Beygelzimer, A., Hsu, D., Eds.; PMLR: New York, NY, USA, 2019; Volume 99, pp. 533–557.
48. Grosse, J.; Zhang, C.; Hennig, P. Optimistic Optimization of Gaussian Process Samples. *arXiv* **2023**, arXiv:2209.00895.