FAST AUTOMATIC BAYESIAN CUBATURE USING MATCHING KERNELS AND DESIGN

BY

JAGADEESWARAN RATHINAVEL

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Applied Mathematics
in the Graduate College of the
Illinois Institute of Technology

Approved _____
Advisor

Chicago, Illinois
December 2019

ACKNOWLEDGMENT

I want to thank my advisor Prof. Fred Hickernell for his support and guidance in my completion of this thesis and throughout my studies here at IIT. His support and motivation have given me the confidence to endure through the research.

I would like to also thank the GAIL project collaborators with whom I have worked to add my new algorithms to the GAIL MATLAB toolbox: Professor Sou-Cheng Choi, Yuhan Ding, Lan Jiang, Xin Tong, and Kan Zhang. Especially, Professor Sou-Cheng Chois support and guidance as the project leader helped me to focus of my cubature algorithms.

My special gratitude also goes to my thesis committee members, Professor Jinqiao Duan, Professor Fred J. Hickernell, Professor Shuwang Li, and Professor Geoffrey Williamson. Above all, I want to thank them because they were flexible and willing dedicate time to review my work and attend my comprehensive and defense examinations.

I would like to thank Dirk Nuyens for suggestions, valuable tips and notes when we were researching higher order nets and kernels.

I would like to thank the organizers of the SAMSI-Lloyds-Turing Workshop on Probabilistic Numerical Methods, where a part of preliminary version of this work was discussed. I also thank Chris Oates and Sou-Cheng Choi for valuable comments.

I would like to specifically thank my friend Samuel Davidson for reviewing and suggesting the improvements on the text.

Last but not least, I would not be able to make it without the support of my family. I would like to thank my wife for her continuous support and sacrifice. I also would like to thank my parents for their endless support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Automatic cubatures approximate multidimensional integrals to user-specified error tolerances. In many real-world integration problems, the analytical solution is either unavailable or difficult to compute. To overcome this, one can use numerical algorithms that approximately estimates the value of the integral.

For high dimensional integrals, usage of quasi-Monte Carlo (QMC) methods are very popular. QMC methods are equal-weight quadrature rules where the quadrature points are chosen deterministically, unlike Monte Carlo (MC) methods where the points are chosen randomly. The families of integration lattice node and digital nets are the most popular quadrature points used. These methods consider the integrand a deterministic function. There is an alternate approach called Bayesian cubature methods. Bayesian cubature methods postulate the integrand is an instance of a Gaussian stochastic process.

For high dimensional problems, it is difficult to adaptively change the sampling pattern, but one can automatically determine the sample size, n, given a fixed and reasonable sampling pattern. We take this approach using a Bayesian perspective. We assume a Gaussian process parameterized by a constant mean and a covariance function defined by a scale parameter and a function specifying how the integrand values at two different points in the domain are related. These parameters are estimated from integrand values or are given non-informative priors. This leads to credible interval for the integral. The sample size, n, is chosen to make the credible interval for the Bayesian posterior error no greater than the desired error tolerance.

However, the process just outlined typically requires vector-matrix operations with a computational cost of $O(n^3)$. Our innovation is to pair low discrepancy nodes with matching kernels that lower the computational cost to $O(n \log n)$. This approach is demonstrated using two methods: 1) rank-1 lattice sequences and shift-invariant

kernels, 2) Sobol' sequences and Walsh kernel. They are also implemented in the Guaranteed Automatic Integration Library (GAIL). The Bayesian cubatures, we develop are guaranteed for integrands belonging to cone of functions that belong to the middle of the sample space. The concept of cone of functions is explained in Section 2.3.

CHAPTER 1

INTRODUCTION

1.1 Cubature

Cubature is the problem of inferring a numerical value for a definite integral, $\mu := \int_{\mathbb{R}^d} g(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$, where μ has no closed form analytic expression. Typically, g is accessible through a black-box function routine. Cubature means numerical multi-variate integration and is a key component of many problems in Scientific computing, finance [1], statistical modeling, Imaging [2], uncertainty quantification, machine learning [3] JR: add references.

The integral may often be expressed as

$$\mu := \mu(f) := \mathbb{E}[f(\boldsymbol{X})] = \int_{[0,1]^d} f(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}, \tag{1.1}$$

where $f : [0,1]^d \to \mathbb{R}$ is the integrand, and $\boldsymbol{X} \sim \mathcal{U}[0,1]^d$. The process of transforming the original integral into the form of (1.1) is not addressed here. The cubature may be an affine function of integrand values:

$$\widehat{\mu} := \widehat{\mu}(f) := w_0 + \sum_{i=1}^{n} f(\boldsymbol{x}_i) w_i, \tag{1.2}$$

where the weights, w_0, and $\boldsymbol{w} = (w_i)_{i=1}^n \in \mathbb{R}^n$, and the nodes, $\{\boldsymbol{x}_i\}_{i=1}^n \subset [0,1]^d$, are chosen to make the error, $|\mu - \widehat{\mu}|$, small. The integration domain $[0,1]^d$ is convenient for the low discrepancy node sets that we use. The nodes are assumed to be deterministic. The integral of function f is the same over $[0,1]^d$ or $(0,1)^d$ or $[0,1)^d$, So we use $[0,1]^d$ or $[0,1)^d$ depending on the application. Most often $[0,1)^d$ is preferred especially for extensible node-sets because it partitions easily into congruent subhypercubes. This research focuses on expensive multivariate numerical integrals where the computational cost of evaluating the integrand is a bottleneck.

1.2 Stopping Criterion

We construct a reliable stopping criterion that determines the number of integrand values required, n, to ensure that the error is no greater than a user-defined error tolerance denoted by ε, i.e.,

$$|\mu - \widehat{\mu}| \leq \varepsilon. \tag{1.3}$$

Rather than relying on strong assumptions about the integrand, such as an upper bound on its variance or total variation, we construct a stopping criterion that is based on a credible interval arising from a Bayesian approach to the problem. We build upon the work of Briol et al. [4], Diaconis [5], O'Hagan [6], Ritter [7], Rasmussen and Ghahramani [8], and others. Our algorithm is an example of *probabilistic numerics*. To study numerical algorithms from a statistical point of view, where uncertainty is formally due to the presence of an unknown numerical error is the goal of probabilistic numerics.

JR: Briefly explain Probabilistic numeric ..

Our primary contribution in this research is to demonstrate how the choice of a family of covariance kernels that match the low discrepancy sampling nodes facilitates fast computation of the cubature and the data-driven stopping criterion. Our Bayesian cubature requires a computational cost of

$$\mathcal{O}\big(n\$(f) + N_{\text{opt}}[n\$(C) + n\log(n)]\big), \tag{1.4}$$

where $\$(f)$ is the cost of one integrand value, $\$(C)$ is the cost of a single covariance kernel value, $\mathcal{O}(n\log(n))$ is the cost of a fast Fourier transform, and N_{opt} is an upper bound on the number of optimization steps required to choose the hyperparameters. If function evaluation is expensive, e.g., the output of a computationally intensive simulation, or if $\$(f) = \mathcal{O}(d)$ for large d, then $\$(f)$ might be similar in magnitude to $N_{\text{opt}}\log(n)$ in practice. Typically, $\$(C) = \mathcal{O}(d)$. Note that the $\mathcal{O}(n\log(n))$ contribution is d independent.

By contrast to our fast algorithm, the typical computational cost for Bayesian cubature is

$$\mathcal{O}\big(n\$(f) + N_{\text{opt}}[n^2\$(C) + n^3]\big), \tag{1.5}$$

which is explained in Section 2.4. Note that aside from evaluating the integrand, the computational cost in (1.5) is much larger than that in (1.4).

1.3 Low Discrepancy Points

Low discrepancy points are characterized by how uniformly and randomly the points are distributed which is measured by the *discrepancy* score, especially when the points are projected onto low-dimensions. The goal is to have as much as uniform space filling. The discrepancy measure is defined as below. Let $\mathcal{M}$ be the set of all intervals of the form $\prod_{j=1}^{d}[a_j, b_j) = \{\boldsymbol{x} \in \mathbb{R}^d : a_j \leq x_j \leq b_j, 0 \leq a_j \leq b_j \leq 1\}$, where $|\mathcal{P}|$ is the cardinality of the set $\mathcal{P}$, and λ_L is the Lebesgue measure. Then, the discrepancy of a point set $\mathcal{P}$ is,

$$D(\mathcal{P}) := \sup_{M \in \mathcal{M}} \left| \frac{|M \cap \mathcal{P}|}{|\mathcal{P}|} - \lambda_L(M) \right|$$

The *low discrepancy points* satisfy $D(\mathcal{P}) = \mathcal{O}(\log n^d / n)$. In this work we experiment with two most popular low discrepancy point sets, 1) lattice points, 2) Sobol' points.

1.4 Prior Work

Hickernell [9] compares different approaches to cubature error analysis depending on whether the rule is deterministic or random and whether the integrand is assumed to be deterministic or random. Error analysis that assumes a deterministic integrand lying in a Banach space leads to an error bound that is typically impractical for deciding how large n must be to satisfy (1.3). The deterministic error bound includes a (semi-) norm of the integrand, which is often more complex to compute than the original integral.

Hickernell and Jiménez-Rugama [10, 11] have developed stopping criteria for cubature rules based on low discrepancy nodes by tracking the decay of the discrete Fourier coefficients of the integrand. The algorithms proposed here also relies on discrete Fourier coefficients, but in a different way. Although we only explore automatic Bayesian cubature for absolute error tolerances, the recent work by Hickernell, Jiménez-Rugama, and Li [12] suggests how one might accommodate more general error criteria, such as relative error tolerances.

Chapter 2 explains the Bayesian approach to estimate the posterior cubature error and defines our automatic Bayesian cubature. Although much of this material is known, it is included for completeness. We end Chapter 2 by demonstrating why Bayesian cubature is typically computationally expensive. Chapter 3 introduces the concept of covariance kernels that match the nodes and expedite the computations required by our automatic Bayesian cubature. Chapter 4 implements this concept for shift invariant kernels and rank-1 lattice nodes. Chapter 5 demonstrates another implementation of matching nodes and kernel using Sobol' points and Walsh kernel. It also develops approaches to build shift-invariant kernels of continuous valued kernel order rather than fixing the kernel order to integer values. Chapter 6 describes how to avoid cancellation error for kernels of product form. It also covers some of the additional techniques used in the implementation of our Bayesian Cubature algorithms. Numerical examples are provided in Chapter 7 to demonstrate the performance and advantages of our new algorithms. We conclude with a brief discussion and future work.

We use the terms integrand or function interchangeably to denote the function f being considered for the numerical integration. Also we use the terms, nodes, points, node-sets, design, and data-sites interchangeably to denote the points $\{\boldsymbol{x}_i\}_{i=1}^n \subset [0,1]^d$ used in the cubature.

CHAPTER 2

BAYESIAN CUBATURE

JR: Briefly explain Bayesian approach using the words of diaconis, poincare work ..

The Bayesian approach for numerical analysis was popularized by Diaconis [5]. The earliest reference for such kind of approach dates back to Poincaré, where, the theory of interpolation was discussed. Diaconis motivates the reader by interpreting the most well known numerical methods, 1) Trapezoidal-rule and 2) Splines, from the statistical point of view with whatever is known about the integrand as prior information. For Example, the trapezoidal-rule can be interpreted as a Bayesian method with prior information being modeled as Brownian motion in $\mathcal{C}[0,1]$, the space of continuous functions, with normal prior.

This research is focused on Bayesian approach for numerical integration that is known as Bayesian Cubature as introduced by O'Hagan [13]. The Bayesian Cubature returns a probability distribution $\mathbb{P}_f$, that expresses belief about the true value of integral, $\mu(f)$. The probability distribution $\mathbb{P}_f$ will be based on a prior that depends on f, in our case simply the function values. The $\mathbb{P}_f$ is computed via Bayes' rule using the *data* contained in the function evaluations. The maximum likelihood estimate of $\mathbb{P}_f$ can be interpreted as a point estimate of the integral [4]. The distribution in general captures numerical uncertainty due to the fact that we have only used a finite number of function values to evaluate the integral.

2.1 Bayesian Posterior Error

We assume the integrand, f, is an instance of a stochastic Gaussian process, i.e., $f \sim \mathcal{GP}(m, s^2 C_{\boldsymbol{\theta}})$. Specifically, f is a real-valued random function with constant mean m and covariance function $s^2 C_{\boldsymbol{\theta}}$, where s is a positive scale factor, and $C_{\boldsymbol{\theta}} : [0,1]^d \times [0,1]^d \to \mathbb{R}$ is a symmetric, positive-definite function and parameterized by

$\boldsymbol{\theta}$:

$$\mathsf{C}^T = \mathsf{C}, \quad \boldsymbol{a}^T \mathsf{C} \boldsymbol{a} > 0, \quad \text{where } \mathsf{C} = (C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1}^n \,, \\ \text{for all } \boldsymbol{a} \neq 0,\ n \in \mathbb{N},\ \text{distinct } \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in [0,1]^d. \quad (2.1)$$

The covariance function, C, and the Gram matrix, C depend implicitly on $\boldsymbol{\theta}$, but the notation may omit this for simplicity's sake wherever possible. Procedures for estimating or integrating out the hyperparameters m, s, and $\boldsymbol{\theta}$ are explained later in this section.

For a Gaussian process, all vectors of linear functionals of f have a multivariate Gaussian distribution. For any deterministic sampling scheme with distinct nodes, $\{\boldsymbol{x}_i\}_{i=1}^n$, and defining $\boldsymbol{f} := (f(\boldsymbol{x}_i))_{i=1}^n$ as the multivariate Gaussian vector of function values, it follows from the definition of a Gaussian process that

$$\boldsymbol{f} \sim \mathcal{N}(m\mathbf{1}, s^2\mathsf{C}), \tag{2.2a}$$

$$\mu \sim \mathcal{N}(m, s^2 c_0), \tag{2.2b}$$

$$\text{where } c_0 := \int_{[0,1]^d \times [0,1]^d} C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{t}, \tag{2.2c}$$

$$\operatorname{cov}(\boldsymbol{f}, \mu) = \left(\int_{[0,1]^d} C(\boldsymbol{t}, \boldsymbol{x}_i) \, \mathrm{d}\boldsymbol{t} \right)_{i=1}^n =: \boldsymbol{c}. \tag{2.2d}$$

Here, c and $\boldsymbol{c}$ depend implicitly on $\boldsymbol{\theta}$. We assume the covariance function C is simple enough that the integrals in these definitions can be computed analytically. We need the following lemma to derive the posterior error of our cubature.

Lemma 2.1.1. *[14, (A.6), (A.11–13)] If* $\boldsymbol{Y} = (\boldsymbol{Y}_1, \boldsymbol{Y}_2)^T \sim \mathcal{N}(\boldsymbol{m}, \mathsf{C})$*, where* $\boldsymbol{Y}_1$ *and* $\boldsymbol{Y}_2$ *are random vectors of arbitrary length, and*

$$\boldsymbol{m} = \begin{pmatrix} \boldsymbol{m}_1 \\ \\ \boldsymbol{m}_2 \end{pmatrix} = \begin{pmatrix} \mathbb{E}(\boldsymbol{Y}_1) \\ \\ \mathbb{E}(\boldsymbol{Y}_2) \end{pmatrix},$$

$$\mathsf{C} = \begin{pmatrix} \mathsf{C}_{11} & \mathsf{C}_{21}^T \\ \\ \mathsf{C}_{21} & \mathsf{C}_{22} \end{pmatrix} = \begin{pmatrix} \mathrm{var}(\boldsymbol{Y}_1) & \mathrm{cov}(\boldsymbol{Y}_1, \boldsymbol{Y}_2) \\ \\ \mathrm{cov}(\boldsymbol{Y}_2, \boldsymbol{Y}_1) & \mathrm{var}(\boldsymbol{Y}_2) \end{pmatrix}$$

then

$$\boldsymbol{Y}_1 | \boldsymbol{Y}_2 \ \sim \ \mathcal{N}\big(\boldsymbol{m}_1 + \mathsf{C}_{21}^T \mathsf{C}_{22}^{-1}(\boldsymbol{Y}_2 - \boldsymbol{m}_2), \quad \mathsf{C}_{11} - \mathsf{C}_{21}^T \mathsf{C}_{22}^{-1} \mathsf{C}_{21}\big).$$

Moreover, the inverse of the matrix C *may be partitioned as*

$$\mathsf{C}^{-1} = \begin{pmatrix} \mathsf{A}_{11} & \mathsf{A}_{21}^T \\ \\ \mathsf{A}_{21} & \mathsf{A}_{22} \end{pmatrix},$$

$$\mathsf{A}_{11} = (\mathsf{C}_{11} - \mathsf{C}_{12}\mathsf{C}_{22}^{-1}\mathsf{C}_{21})^{-1}, \qquad \mathsf{A}_{21} = -\mathsf{C}_{22}^{-1}\mathsf{C}_{21}\mathsf{A}_{11},$$

$$\mathsf{A}_{22} = \mathsf{C}_{22}^{-1} + \mathsf{C}_{22}^{-1}\mathsf{C}_{21}\mathsf{A}_{11}\mathsf{C}_{21}^T\mathsf{C}_{22}^{-1}.$$

It follows from Lemma 2.1.1 that the *conditional* distribution of the integral given observed function values, $\boldsymbol{f} = \boldsymbol{y}$ is also Gaussian:

$$\mu | (\boldsymbol{f} = \boldsymbol{y}) \sim \mathcal{N}\big(m(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1}) + \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}, \ \ s^2(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})\big). \tag{2.3}$$

The natural choice for the cubature is the posterior mean of the integral, namely,

$$\widehat{\mu} | (\boldsymbol{f} = \boldsymbol{y}) = m(1 - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c}) + \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}, \tag{2.4}$$

which takes the form of (1.2). Under this definition, the cubature error has zero mean and a variance depending on the choice of nodes:

$$(\mu - \widehat{\mu}) | (\boldsymbol{f} = \boldsymbol{y}) \sim \mathcal{N}\left(0, \quad s^2(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})\right).$$

A credible interval for the integral is given by

$$\mathbb{P}_f\left[|\mu - \widehat{\mu}| \leq \mathrm{err}_{\mathrm{CI}}\right] = 99\%, \tag{2.5a}$$

$$\mathrm{err}_{\mathrm{CI}} = 2.58 s \sqrt{c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}}. \tag{2.5b}$$

Naturally, 2.58 and 99% can be replaced by other quantiles and credible levels.

2.2 Hyperparameter Estimation

The credible interval in (2.5) suggests how our automatic Bayesian cubature proceeds. Integrand data is accumulated until the width of the credible interval, $\mathrm{err}_{\mathrm{CI}}$, is no greater than the error tolerance. As n increases, one expects $c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}$ to decrease for well-chosen nodes, $\{\boldsymbol{x}_i\}_{i=1}^n$. Please note that the credible interval depends on the parameters m, s, and $\boldsymbol{\theta}$

Note that $\mathrm{err}_{\mathrm{CI}}$ has no explicit dependence on the integrand values, even though one would intuitively expect that a larger integrand should imply a larger $\mathrm{err}_{\mathrm{CI}}$. This is because the hyperparameters, m, s, and $\boldsymbol{\theta}$, have not yet been inferred from integrand data. After inferring the hyperparameters, $\mathrm{err}_{\mathrm{CI}}$ does reflect the size of the integrand values. This section describes three approaches to hyperparameter estimation.

2.2.1 Empirical Bayes. The first and a very straight forward approach is to estimate the parameters is via maximum likelihood estimation. The log-likelihood function of the parameters given the function data $\boldsymbol{y}$ is:

$$\begin{aligned} l(s, m, \boldsymbol{\theta} | \boldsymbol{y}) = &-\frac{1}{2} s^{-2} (\boldsymbol{y} - m\mathbf{1})^T \mathsf{C}^{-1} (\boldsymbol{y} - m\mathbf{1}) \\ &- \frac{1}{2} \log(\det \mathsf{C}) - \frac{n}{2} \log(s^2) + \text{constants}. \end{aligned}$$

Maximizing the log-likelihood first with respect to m, then with respect to s, and finally with respect to $\boldsymbol{\theta}$ yields

$$\begin{aligned} m_{\mathrm{MLE}} &= \frac{\mathbf{1}^T \mathsf{C}^{-1} \boldsymbol{y}}{\mathbf{1}^T \mathsf{C}^{-1} \mathbf{1}}, \\ s^2_{\mathrm{MLE}} &= \frac{1}{n} (\boldsymbol{y} - m_{\mathrm{MLE}} \mathbf{1})^T \mathsf{C}^{-1} (\boldsymbol{y} - m_{\mathrm{MLE}} \mathbf{1}) \\ &= \frac{1}{n} \boldsymbol{y}^T \left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1} \mathbf{1} \mathbf{1}^T \mathsf{C}^{-1}}{\mathbf{1}^T \mathsf{C}^{-1} \mathbf{1}} \right] \boldsymbol{y}, \end{aligned}$$

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmin}}\left\{\log\left(\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}\right) + \frac{1}{n}\log(\det(\mathsf{C}))\right\}.$$

The maximum likelihood estimate of $\boldsymbol{\theta}$ balances minimizing the covariance scale factor, s^2_{MLE}, against minimizing $\det(\mathsf{C})$.

Under these estimates of the parameters, the cubature (2.4) and the credible interval (2.5) simplify to

$$\widehat{\mu}_{\text{MLE}} = \left(\frac{(1 - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c})\mathbf{1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} + \boldsymbol{c}\right)^T \mathsf{C}^{-1}\boldsymbol{y},$$
$$\text{err}^2_{\text{MLE}} := \frac{2.58^2}{n}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}),$$

$$\mathbb{P}_f\left[|\mu - \widehat{\mu}_{\text{MLE}}| \leq \text{err}_{\text{MLE}}\right] = 99\%. \tag{2.6}$$

Here c_0, $\boldsymbol{c}$, and C are assumed implicitly to be based on $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{MLE}}$. The empirical Bayes estimate of $\boldsymbol{\theta}$ balances minimizing the covariance scale factor, s^2_{MLE}, against minimizing $\det(\mathsf{C})$.

2.2.1.1 Gradient descent to find optimal shape parameter. The equation of $\boldsymbol{\theta}_{\text{MLE}}$ as defined in (2.15) does not say how the parameter search can be done. There exists empirical algorithms [15] [16] one could use to accomplish the same. Since the objective function is known we could compute the gradient. Using the gradient of $l(s, m, \boldsymbol{\theta}|\boldsymbol{y})$, We can apply optimization techniques such as gradient descent to find the optimal value faster. Let us define the objective function for the same purpose by excluding the negative sign, which modifies the problem as minimization

$$\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) := \frac{1}{n}\log(\det\,\mathsf{C}) + \log\left((\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})\right) + \text{constants}.$$

Taking derivative with respect to $\boldsymbol{\theta}_i$

$$\begin{aligned}\frac{\partial}{\partial\boldsymbol{\theta}_i}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) &= \frac{1}{n}\frac{\partial}{\partial\boldsymbol{\theta}_i}\log(\det\,\mathsf{C}) + \frac{\partial}{\partial\boldsymbol{\theta}_i}\log\left((\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})\right)\\ &= \frac{1}{n}\text{trace}\left(\mathsf{C}^{-1}\frac{\partial\mathsf{C}}{\partial\boldsymbol{\theta}_i}\right) - \frac{\left((\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}\right)^T\left(\frac{\partial\mathsf{C}}{\partial\boldsymbol{\theta}_i}\right)\left((\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}\right)}{(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})}\end{aligned}$$

where we used some of the results from [17]. This can be used with steepest gradient descent as follows,

$$\boldsymbol{\theta}_i^{(j+1)} = \boldsymbol{\theta}_i^{(j+1)} - \nu \frac{\partial}{\partial \boldsymbol{\theta}_i} \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) \tag{2.7}$$

where ν is the stepping constant.

2.2.2 Full Bayes. Rather than use maximum likelihood to determine m and s, one can treat them as hyper-parameters with a non-informative, conjugate prior, namely $\boldsymbol{\rho}_{m,s^2}(\xi, \lambda) \propto 1/\lambda$. Then the posterior density for the integral given the data using Bayes theorem is,

$$\begin{aligned}
&\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}) \\
&\quad \propto \int_0^\infty \int_{-\infty}^\infty \rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}, m = \xi, s^2 = \lambda) \rho_{\boldsymbol{f}}(\boldsymbol{y}|\xi, \lambda) \rho_{m,s^2}(\xi, \lambda)\, \mathrm{d}\xi \mathrm{d}\lambda \\
&\qquad\qquad \text{by the properties of conditional probability} \\
&\quad \propto \int_0^\infty \int_{-\infty}^\infty \rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}, m = \xi, s^2 = \lambda) \rho_{\boldsymbol{f}}(\boldsymbol{y}|\xi, \lambda) \rho_{m,s^2}(\xi, \lambda)\, \mathrm{d}\xi \mathrm{d}\lambda \\
&\qquad\qquad \text{by Bayes' Theorem} \\
&\quad \propto \int_0^\infty \frac{1}{\lambda^{(n+3)/2}} \int_{-\infty}^\infty \exp\Bigg(-\frac{1}{2\lambda} \Bigg\{ \frac{[z - \xi(1 - \boldsymbol{c}^T \mathsf{C}^{-1} \mathbf{1}) - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{y}]^2}{c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}} \\
&\qquad\qquad + (\boldsymbol{y} - \xi \mathbf{1})^T \mathsf{C}^{-1} (\boldsymbol{y} - \xi \mathbf{1}) \Bigg\} \Bigg)\, \mathrm{d}\xi \mathrm{d}\lambda \\
&\qquad\qquad \text{by (2.2), (2.3) and } \rho_{m,s^2}(\xi, \lambda) \propto 1/\lambda \\
&\quad \propto \int_0^\infty \frac{1}{\lambda^{(n+3)/2}} \int_{-\infty}^\infty \exp\left(-\frac{\alpha \xi^2 - 2\beta\xi + \gamma}{2\lambda(c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c})} \right) \mathrm{d}\xi \mathrm{d}\lambda,
\end{aligned}$$

where

$$\begin{aligned}
\alpha &= (1 - \boldsymbol{c}^T \mathsf{C}^{-1} \mathbf{1})^2 + \mathbf{1}^T \mathsf{C}^{-1} \mathbf{1} (c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}), \\
\beta &= (1 - \boldsymbol{c}^T \mathsf{C}^{-1} \mathbf{1})(z - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{y}) + \mathbf{1}^T \mathsf{C}^{-1} \boldsymbol{y} (c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}), \\
\gamma &= (z - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{y})^2 + \boldsymbol{y}^T \mathsf{C}^{-1} \boldsymbol{y} (c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}).
\end{aligned}$$

In the derivation above and below, factors that are independent of ξ, λ, or z can be discarded since we only need to preserve the proportion. But, factors that depend

on ξ, λ, or z must be kept. Completing the square $\alpha\xi^2 - 2\beta\xi + \gamma = \alpha(\xi - \beta/\alpha)^2 - (\beta^2/\alpha) + \gamma$, allows us to evaluate the integrals with respect to ξ and λ:

$$\begin{aligned}
&\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}) \\
&\quad \propto \int_0^\infty \frac{1}{\lambda^{(n+3)/2}} \exp\left(-\frac{\gamma - \beta^2/\alpha}{2\lambda(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right) \cdots \\
&\qquad\qquad \cdots \int_{-\infty}^\infty \exp\left(-\frac{\alpha(\xi - \beta/\alpha)^2}{2\lambda(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right) \mathrm{d}\xi\mathrm{d}\lambda \\
&\quad \propto \int_0^\infty \frac{1}{\lambda^{(n+2)/2}} \exp\left(-\frac{\gamma - \beta^2/\alpha}{2\lambda(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right) \mathrm{d}\lambda \\
&\quad \propto \left(\gamma - \frac{\beta^2}{\alpha}\right)^{-n/2} \propto \left(\alpha\gamma - \beta^2\right)^{-n/2}.
\end{aligned}$$

Finally, we simplify the key term:

$$\begin{aligned}
\alpha\gamma - \beta^2 &= \mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})(z - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y})^2 \\
&\qquad - 2\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})(z - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}) \\
&\qquad + (1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})^2\boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}) \\
&\qquad + [\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}\boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y} - (\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y})^2](c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})^2 \\
&\propto \mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}\left(z - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y} - \frac{(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right)^2 \\
&\qquad - \frac{[(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}]^2}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} + (1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})^2\boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y} \\
&\qquad (c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})[\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}\boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y} - (\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y})^2] \\
&\propto \left(z - \left[\frac{(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})\mathbf{1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} + \boldsymbol{c}\right]^T \mathsf{C}^{-1}\boldsymbol{y}\right)^2 \\
&\qquad + \left[\frac{(1 - \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})^2}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} + (c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})\right] \times \boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y} \\
&\propto (z - \widehat{\mu}_{\text{full}})^2 + (n-1)\sigma^2_{\text{full}} \\
&\propto \left(1 + \frac{1}{n-1}\frac{(z - \mu_{\text{full}})^2}{\widehat{\sigma}^2_{\text{full}}}\right)
\end{aligned}$$

i.e.,

$$\alpha\gamma - \beta^2 \propto \left(1 + \frac{(z - \widehat{\mu}_{\text{full}})^2}{(n-1)\widehat{\sigma}^2_{\text{full}}},\right)$$

where $\widehat{\mu}_{\text{full}} = \widehat{\mu}_{\text{MLE}}$ and

$$\widehat{\sigma}^2_{\text{full}} \;:=\; \frac{1}{n-1}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y} \;\times\; \left[\frac{(1-\boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})^2}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} + (c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})\right].$$

This means that $\mu|(\boldsymbol{f}=\boldsymbol{y})$, properly centered and scaled, has a Student's t-distribution with $n-1$ degrees of freedom. The estimated integral is the same as in the empirical Bayes case, $\widehat{\mu}_{\text{full}} = \widehat{\mu}_{\text{MLE}}$, but the confidence interval is wider:

$$\mathbb{P}_f\left[|\mu - \widehat{\mu}_{\text{MLE}}| \le \text{err}_{\text{full}}\right] = 99\%, \tag{2.8}$$

where

$$\text{err}_{\text{full}} := t_{n_j-1,0.995}\widehat{\sigma}_{\text{full}} > \text{err}_{\text{MLE}}.$$

Here $t_{n-1,0.995}$ denotes the 99.5 percentile of a standard Student's t-distribution with $n-1$ degrees of freedom. This means that $\mu|(\boldsymbol{f}=\boldsymbol{y})$, properly centered and scaled, has a Student's t-distribution with $n-1$ degrees of freedom. The estimated integral is the same as in the empirical Bayes case, $\widehat{\mu}_{\text{full}} = \widehat{\mu}_{\text{MLE}}$, but the credible interval is wider. In other words, the stopping criterion for the full Bayes case is more conservative than that in the empirical Bayes case, (2.6).

Because the shape parameter, $\boldsymbol{\theta}$, enters the definition of the covariance kernel in a non-trivial way, the only way to treat it as a hyperparameter and assign a tractable prior would be for the prior to be discrete. We believe in practice that choosing such a prior involves more guesswork than using the empirical Bayes estimate of $\boldsymbol{\theta}$ in (2.15) or the cross-validation approach described next.

2.2.3 Full Bayes with generic prior. Rather than use non-informative, conjugate prior one can use generic prior, namely $\boldsymbol{\rho}_{m,s^2}(\xi,\lambda) \propto g(\lambda)$, which can generalize to any generic function. Then the posterior density for the integral given the data using Bayes theorem is,

$$\rho_\mu(z|\boldsymbol{f}=\boldsymbol{y})$$

$$\propto \int_0^\infty \int_{-\infty}^\infty \rho_\mu(z|\boldsymbol{f}=\boldsymbol{y}, m=\xi, s^2=\lambda)\rho_{\boldsymbol{f}}(\boldsymbol{y}|\xi,\lambda)\rho_{m,s^2}(\xi,\lambda)\,\mathrm{d}\xi\mathrm{d}\lambda$$

by the properties of conditional probability

$$\propto \int_0^\infty \int_{-\infty}^\infty \rho_\mu(z|\boldsymbol{f}=\boldsymbol{y}, m=\xi, s^2=\lambda)\rho_{\boldsymbol{f}}(\boldsymbol{y}|\xi,\lambda)\rho_{m,s^2}(\xi,\lambda)\,\mathrm{d}\xi\mathrm{d}\lambda$$

by Bayes' Theorem

$$\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{(n+1)/2}} \int_{-\infty}^\infty \exp\Bigg(-\frac{1}{2\lambda}\Bigg\{\frac{[z-\xi(1-\boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}]^2}{c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}} + (\boldsymbol{y}-\xi\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y}-\xi\mathbf{1})\Bigg\}\Bigg)\,\mathrm{d}\xi\mathrm{d}\lambda$$

by (2.2), (2.3) and $\rho_{m,s^2}(\xi,\lambda) \propto g(\lambda)$

$$\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{(n+1)/2}} \int_{-\infty}^\infty \exp\left(-\frac{\alpha\xi^2-2\beta\xi+\gamma}{2\lambda(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right)\,\mathrm{d}\xi\mathrm{d}\lambda,$$

where

$$\alpha = (1-\boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})^2 + \mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}),$$

$$\beta = (1-\boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1})(z-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}) + \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}),$$

$$\gamma = (z-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y})^2 + \boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y}(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}).$$

In the derivation above and below, factors that are independent of ξ, λ, or z can be discarded since we only need to preserve the proportion. But, factors that depend on ξ, λ, or z must be kept. Completing the square $\alpha\xi^2-2\beta\xi+\gamma = \alpha(\xi-\beta/\alpha)^2 - (\beta^2/\alpha)+\gamma$, allows us to evaluate the integrals with respect to ξ and λ:

$$\begin{aligned}
&\rho_\mu(z|\boldsymbol{f}=\boldsymbol{y}) \\
&\quad\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{(n+1)/2}} \exp\left(-\frac{\gamma-\beta^2/\alpha}{2\lambda(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right)\cdots \\
&\qquad\qquad \cdots\int_{-\infty}^\infty \exp\left(-\frac{\alpha(\xi-\beta/\alpha)^2}{2\lambda(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right)\,\mathrm{d}\xi\mathrm{d}\lambda \\
&\quad\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{n/2}} \exp\left(-\frac{\gamma-\beta^2/\alpha}{2\lambda(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right)\mathrm{d}\lambda.
\end{aligned}$$

This can be interpreted as Laplace transform of $g(\lambda)$.

$$\rho_\mu(z|\boldsymbol{f}=\boldsymbol{y}) \propto \int_0^\infty \frac{g(\lambda)}{\lambda^{n/2}} \exp\left(-\frac{\gamma-\beta^2/\alpha}{2\lambda(c_0-\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\right)\mathrm{d}\lambda$$

$$\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{n/2}} \exp\left(-\frac{1}{\lambda}\chi\right) \mathrm{d}\lambda, \quad \text{where} \quad \chi = \frac{\gamma - \beta^2/\alpha}{2(c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c})}$$

Let $\lambda = \dfrac{1}{w}, \quad \mathrm{d}\lambda = -w^{-2}\mathrm{d}w$ then,

$$\begin{aligned}
\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}) &\propto \int_0^\infty \frac{g(\lambda)}{\lambda^{n/2}} \exp\left(-\frac{1}{\lambda}\chi\right) \mathrm{d}\lambda \\
&= \int_0^\infty \frac{g(1/w)}{w^{-n/2}} \exp\left(-w\chi\right) (-w^{-2}) \mathrm{d}w \\
&= \int_\infty^0 -g(1/w) w^{\frac{n}{2}-2} \exp\left(-w\chi\right) \mathrm{d}w \\
&= \int_0^\infty g(1/w) w^{\frac{n-4}{2}} \exp\left(-w\chi\right) \mathrm{d}w \\
&= \mathcal{LT}\{g(1/\chi)\}^{(\frac{n-4}{2})'}
\end{aligned}$$

where $\mathcal{LT}(.)$ denotes the Laplace transform and $(\frac{n-4}{2})'$ indicates the derivative taken after the transform. Here we used frequency domain derivative property of the Laplace transform. Thus, $\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y})$ is proportional to $(\frac{n-4}{2})$th derivative of the Laplace transform of $g(1/\chi)$.

If $g(\lambda) = \dfrac{1}{\lambda}$ then,

$$\begin{aligned}
\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}) &= \int_0^\infty g(1/w) w^{\frac{n}{2}-2} \exp\left(-w\chi\right) \mathrm{d}w \\
&= \left(\mathcal{LT}(g(1/t))\right)^{(\frac{n}{2}-2)'} |_{t=\chi} \\
&= \left(1/t^2\right)^{(\frac{n}{2}-2)'} |_{t=\chi} \\
&\propto \chi^{-n/2} = \left(\frac{\gamma - \beta^2/\alpha}{2(c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c})}\right)^{-n/2} \\
&\propto \left(\gamma - \frac{\beta^2}{\alpha}\right)^{-n/2} \\
&\propto \left(\alpha\gamma - \beta^2\right)^{-n/2},
\end{aligned}$$

where we used the fact, Laplace transform of $g(1/t)$ is $1/t^2$. After the transform, taking $(\frac{n}{2} - 2)$th derivative gives us the result. This shows when using a generic prior leads to a posterior of the form, $\rho_\mu(z|\boldsymbol{f} = \boldsymbol{y}) \propto \mathcal{LT}\{g(1/\chi)\}^{(\frac{n-4}{2})'}$, with full Bayes approach.

2.2.4 Generalized Cross-Validation. A third parameter optimization technique is *leave-one-out cross-validation* (CV). Let $\widetilde{y}_i = \mathbb{E}[f(\boldsymbol{x}_i)|\boldsymbol{f}_{-i} = \boldsymbol{y}_{-i}]$, where the subscript $-i$ denotes the vector excluding the i^{th} component. This is the conditional expectation of $f(\boldsymbol{x}_i)$ given all data but the function value at $\boldsymbol{x}_i$. The cross-validation criterion, which is to be minimized, is sum of squares of the difference between these conditional expectations and the observed values:

$$\text{CV} = \sum_{i=1}^{n} (y_i - \widetilde{y}_i)^2. \tag{2.9}$$

Let $\mathsf{A} = \mathsf{C}^{-1}$, let $\boldsymbol{\zeta} = \mathsf{A}(\boldsymbol{y} - m\mathbf{1})$, and partition C, A, and $\boldsymbol{\zeta}$ as

$$\mathsf{C} = \begin{pmatrix} c_{ii} & \boldsymbol{C}_{-i,i}^T \\ \boldsymbol{C}_{-i,i} & \mathsf{C}_{-i,-i} \end{pmatrix}, \qquad \mathsf{A} = \begin{pmatrix} a_{ii} & \boldsymbol{A}_{-i,i}^T \\ \boldsymbol{A}_{-i,i} & \mathsf{A}_{-i,-i} \end{pmatrix}, \qquad \boldsymbol{\zeta} = \begin{pmatrix} \zeta_i \\ \boldsymbol{\zeta}_{-i} \end{pmatrix},$$

where the subscript i denotes the i^{th} row or column, and the subscript $-i$ denotes all rows or columns except the i^{th}. Following this notation, Lemma 2.1.1 implies that

$$\begin{aligned} \widetilde{y}_i &= m + \boldsymbol{C}_{-i,i}^T \mathsf{C}_{-i,-i}^{-1} (\boldsymbol{y}_{-i} - m\mathbf{1}) \\ \zeta_i &= a_{ii}(y_i - m) + \boldsymbol{A}_{-i,i}^T (\boldsymbol{y}_{-i} - m\mathbf{1}) \\ &= a_{ii}[(y_i - m) - \boldsymbol{C}_{-i,i}^T \mathsf{C}_{-i,-i}^{-1} (\boldsymbol{y}_{-i} - m\mathbf{1})] \\ &= a_{ii}(y_i - \widetilde{y}_i). \end{aligned}$$

Thus, (2.9) may be re-written as

$$\text{CV} = \sum_{i=1}^{n} \left(\frac{\zeta_i}{a_{ii}} \right)^2, \quad \text{where} \quad \boldsymbol{\zeta} = \mathsf{C}^{-1}(\boldsymbol{y} - m\mathbf{1}).$$

The *generalized cross-validation* criterion (GCV) replaces the i^{th} diagonal element of A in the denominator by the average diagonal element of A [18, 19, 20]:

$$\text{GCV} = \frac{\sum_{i=1}^{n} \zeta_i^2}{\left(\frac{1}{n} \sum_{i=1}^{n} a_{ii} \right)^2}$$

$$= \frac{(\boldsymbol{y} - m\mathbf{1})^T \mathsf{C}^{-2}(\boldsymbol{y} - m\mathbf{1})}{\left(\frac{1}{n}\text{trace}(\mathsf{C}^{-1})\right)^2}.$$

The loss function GCV depends on m and $\boldsymbol{\theta}$, but not on s. Minimizing the GCV yields

$$m_{\text{GCV}} = \frac{\mathbf{1}^T \mathsf{C}^{-2} \boldsymbol{y}}{\mathbf{1}^T \mathsf{C}^{-2} \mathbf{1}},$$

$$\boldsymbol{\theta}_{\text{GCV}} \quad = \quad \underset{\boldsymbol{\theta}}{\text{argmin}} \left\{ \log \left(\boldsymbol{y}^T \left[\mathsf{C}^{-2} - \frac{\mathsf{C}^{-2}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-2}}{\mathbf{1}^T\mathsf{C}^{-2}\mathbf{1}} \right] \boldsymbol{y} \right) \; - \; 2\log\left(\text{trace}(\mathsf{C}^{-1})\right) \right\}.$$

Plugging this value of m into (2.4) yields

$$\widehat{\mu}_{\text{GCV}} = \left(\frac{(1 - \mathbf{1}^T \mathsf{C}^{-1}\boldsymbol{c})\mathsf{C}^{-1}\mathbf{1}}{\mathbf{1}^T\mathsf{C}^{-2}\mathbf{1}} + \boldsymbol{c} \right)^T \mathsf{C}^{-1}\boldsymbol{y}.$$

An estimate for s may be obtained by noting that by Lemma 2.1.1,

$$\text{var}[f(\boldsymbol{x}_i) | \boldsymbol{f}_{-i} = \boldsymbol{y}_{-i}] = s^2 a_{ii}^{-1}.$$

Thus, we may estimate 's' using an argument similar to that used in deriving the GCV and then substituting m_{GCV} for m:

$$\begin{aligned}
s^2 &= \text{var}[f(\boldsymbol{x}_i) | \boldsymbol{f}_{-i} = \boldsymbol{y}_{-i}] a_{ii} \\
&\approx \frac{1}{n} \sum_{i=1}^{n} (y_i - \widetilde{y}_i)^2 a_{ii} \\
&= \frac{1}{n} \sum_{i=1}^{n} \frac{\zeta_i^2}{a_{ii}} \\
&\approx \frac{\frac{1}{n}\sum_{i=1}^{n} \zeta_i^2}{\frac{1}{n}\sum_{i=1}^{n} a_{ii}} \\
&= \frac{(\boldsymbol{y} - m\mathbf{1})^T \mathsf{C}^{-2}(\boldsymbol{y} - m\mathbf{1})}{\text{trace}(\mathsf{C}^{-1})} \quad = \quad s^2_{\text{GCV}},
\end{aligned}$$

where

$$s^2_{\text{GCV}} := \boldsymbol{y}^T \left[\mathsf{C}^{-2} - \frac{\mathsf{C}^{-2}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-2}}{\mathbf{1}^T\mathsf{C}^{-2}\mathbf{1}} \right] \boldsymbol{y} \left[\text{trace}(\mathsf{C}^{-1}) \right]^{-1}.$$

The confidence interval based on generalized cross-validation corresponds to (2.5) with the GCV estimates for m, s, and $\boldsymbol{\theta}$:

$$\mathrm{err}_{\mathrm{GCV}} = 2.58 s_{\mathrm{GCV}} \sqrt{c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c}}, \tag{2.10}$$

$$\mathbb{P}_f \left[|\mu - \widehat{\mu}_{\mathrm{GCV}}| \leq \mathrm{err}_{\mathrm{GCV}} \right] = 99\%. \tag{2.11}$$

The methods developed for hyperparameter estimation from the previous sections are summarized as below:

Theorem 2.2.1. *There are at least three approaches to estimating or integrating out the hyperparameters defining the Gaussian process from which the integrand is drawn: empirical Bayes, full Bayes, and generalized cross-validation. Under these three approaches, we have the following:*

$$m_{\mathrm{MLE}} = \frac{\mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \boldsymbol{y}}{\mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1}}, \qquad m_{\mathrm{GCV}} = \frac{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2} \boldsymbol{y}}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1}}, \tag{2.12}$$

$$s^2_{\mathrm{MLE}} = \frac{1}{n} \boldsymbol{y}^T \left[\mathsf{C}^{-1}{}_{\boldsymbol{\theta}} - \frac{\mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1} \mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}}}{\mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1}} \right] \boldsymbol{y}, \tag{2.13}$$

$$\begin{aligned} \widehat{\sigma}^2_{\mathrm{full}} = {} & \frac{1}{n-1} \boldsymbol{y}^T \left[\mathsf{C}_{\boldsymbol{\theta}}^{-1} - \frac{\mathsf{C}_{\boldsymbol{\theta}}^{-1} \mathbf{1} \mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1}}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \mathbf{1}} \right] \boldsymbol{y} \\ & \times \left[\frac{(1 - \boldsymbol{c}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \mathbf{1})^2}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \mathbf{1}} + (c_0 - \boldsymbol{c}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \boldsymbol{c}) \right], \end{aligned} \tag{2.14}$$

$$s^2_{\mathrm{GCV}} = \boldsymbol{y}^T \left[\mathsf{C}_{\boldsymbol{\theta}}^{-2} - \frac{\mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1} \mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2}}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1}} \right] \boldsymbol{y} \left[\mathrm{trace}(\mathsf{C}_{\boldsymbol{\theta}}^{-1}) \right]^{-1},$$

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \underset{\boldsymbol{\theta}}{\mathrm{argmin}} \left\{ \log \left(\boldsymbol{y}^T \left[\mathsf{C}_{\boldsymbol{\theta}}^{-1} - \frac{\mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1} \mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}}}{\mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1}} \right] \boldsymbol{y} \right) + \frac{1}{n} \log(\det(\mathsf{C}_{\boldsymbol{\theta}})) \right\}, \tag{2.15}$$

$$\boldsymbol{\theta}_{\mathrm{GCV}} = \underset{\boldsymbol{\theta}}{\mathrm{argmin}} \left\{ \log \left(\boldsymbol{y}^T \left[\mathsf{C}_{\boldsymbol{\theta}}^{-2} - \frac{\mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1} \mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2}}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1}} \right] \boldsymbol{y} \right) - \log \left(\mathrm{trace}(\mathsf{C}_{\boldsymbol{\theta}}^{-2}) \right) \right\}, \tag{2.16}$$

$$\widehat{\mu}_{\mathrm{MLE}} = \widehat{\mu}_{\mathrm{full}} = \left(\frac{(1 - \mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \boldsymbol{c}) \mathbf{1}}{\mathbf{1}^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \mathbf{1}} + \boldsymbol{c} \right)^T \mathsf{C}^{-1}{}_{\boldsymbol{\theta}} \boldsymbol{y}, \tag{2.17}$$

$$\widehat{\mu}_{\mathrm{GCV}} = \left(\frac{(1 - \mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \boldsymbol{c}) \mathsf{C}_{\boldsymbol{\theta}}^{-1} \mathbf{1}}{\mathbf{1}^T \mathsf{C}_{\boldsymbol{\theta}}^{-2} \mathbf{1}} + \boldsymbol{c} \right)^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \boldsymbol{y}. \tag{2.18}$$

The credible intervals widths, $\mathrm{err}_{\mathrm{CI}}$*, are given by*

$$\mathrm{err}_{\mathsf{x}} = 2.58 s_{\mathsf{x}} \sqrt{c_0 - \boldsymbol{c}^T \mathsf{C}_{\boldsymbol{\theta}}^{-1} \boldsymbol{c}}, \qquad \mathsf{x} \in \{\mathrm{MLE}, \mathrm{GCV}\}, \tag{2.19}$$

$$\mathrm{err}_{\mathrm{full}} = t_{n-1,0.995}\widehat{\sigma}_{\mathrm{full}} > \mathrm{err}_{\mathrm{MLE}}. \tag{2.20}$$

The resulting credible intervals are then

$$\mathbb{P}_f\left[|\mu - \widehat{\mu}_{\mathsf{x}}| \le \mathrm{err}_{\mathsf{x}}\right] = 99\%, \qquad \mathsf{x} \in \{\mathrm{MLE}, \mathrm{full}, \mathrm{GCV}\}. \tag{2.21}$$

Here $t_{n-1,0.995}$ denotes the 99.5 percentile of a standard Student's t-distribution with $n-1$ degrees of freedom. In the formulas above, $\boldsymbol{\theta}$ is assumed to take on the values $\boldsymbol{\theta}_{\mathrm{MLE}}$ or $\boldsymbol{\theta}_{\mathrm{GCV}}$ as appropriate.

In the theorem above, note that if the original covariance kernel, C, is replaced by bC for some positive constant b, the cubature, $\widehat{\mu}$, the estimates of $\boldsymbol{\theta}$, and the credible interval half-widths, $\mathrm{err}_{\mathsf{x}}$ for $\mathsf{x} \in \{\mathrm{MLE}, \mathrm{full}, \mathrm{GCV}\}$, all remain unchanged. The estimates of s^2 are multiplied by b^{-1}, as would be expected.

2.3 Cone of Functions and the Credible interval

JR: Done: Revise f_{nice} using kernel approximation

We assume the integrand belongs to a cone of well-behaved functions, $\mathscr{C}$, to make the computations bounded in terms of function data. The concept of cone in general for cubature error analysis can be stated using the error bound definition. Given the data driven error bound $\mathrm{err}_{f,n}$

$$|\mu(f) - \widehat{\mu}_n(f)| \le \mathrm{err}_{f,n}(f(x_1), \ldots f(x_n)), \quad \forall f \in \mathscr{C}$$

where the cone of functions is defined as $\mathscr{C} = \{f : f \in \mathscr{C} \Rightarrow af \in \mathscr{C}\}$. More specifically, if a function f belongs to $\mathscr{C}$, then any function obtained by constant multiplication, af, also belongs to the cone. This property is very useful, since

$$\begin{aligned} |\mu(af) - \widehat{\mu}_n(af)| &\le |a|\,|\mu(f) - \widehat{\mu}_n(f)| \\ &\le |a|\,\mathrm{err}_{f,n}(f(x_1), \ldots f(x_n)) \end{aligned}$$

$$= \mathrm{err}_{f,n}(af(x_1), \ldots af(x_n)).$$

If the integrand is multiplied by a constant, say $a = 45$, then the integral error is also multiplied by the same constant $a = 45$, but the factor $\mathrm{err}_{f,n}$ which depends on the characteristic of f does not change. In the context of Bayesian cubature, One can explain the cone concept beginning with the definition of credible interval (2.5). Let $f \sim \mathcal{GP}$, is an instance of a Gaussian stochastic process

$$\mathbb{P}_f \left[|\mu(f) - \widehat{\mu}_n(f)| \leq \mathrm{err}_n(f) \right] \geq 99\%.$$

This can be interpreted as $|\mu(f) - \widehat{\mu}_n(f)| \leq \mathrm{err}_n(f)$ with 99% confidence. If f is in the 99% middle of the functions space such that $f(x_i) = y_i$ then af is also in the center of 99%.

We explain the credible interval using the following example. For this purpose, chosen a smooth and periodic integrand $f_{nice}(\boldsymbol{x}) = \exp(\sum_{j=1}^{d} \cos(2\pi x_j)) + a_{nice} f_{noise}$ and another integrand $f_{peaky}(\boldsymbol{x}) = f_{nice} + a_{peaky} f_{noise}$ where $a_{peaky} >> a_{nice}$. Here the function $f_{noise}(\boldsymbol{x}) = (1 - \exp(2\pi i \sqrt{-1} \boldsymbol{x}^T \boldsymbol{\zeta}))$ is chosen to be zero at the sampling nodes $\{\boldsymbol{x}_i\}_{i=1}^n$, where $a \in \mathbb{R}$ is some constant, $\boldsymbol{\zeta} \in \mathbb{R}^d$ is some d-dimensional vector belonging to the dual space of the lattice nodes $\{\boldsymbol{x}_i\}_{i=1}^n$ chosen to sample the integrand. This helps to satisfy $f_{nice}(\boldsymbol{x}_i) = f_{real}(\boldsymbol{x}_i)$ at the sampling nodes $\{\boldsymbol{x}_i\}_{i=1}^n$

As shown in Figure 2.1, sampled function values $\{f(\boldsymbol{x}_i)\}_{i=1}^n$ from a smooth integrand f_{true} are shown as dots. One can imagine these samples $\{f(\boldsymbol{x}_i)\}_{i=1}^n$ were obtained from f_{nice}, a moderately smoother function or from f_{peaky}, a highly oscillating function.

When using $n = 16$ rank-1 lattice points and $r = 1$ shift-invariant kernel, we get the posterior distribution of μ as shown in Figure 2.2. The true integral value is shown as μ_{true} which is at the center of the plot. The integral of the peaky function f_{peaky} almost lies outside of the 99% of the credible interval given by (2.6), whereas

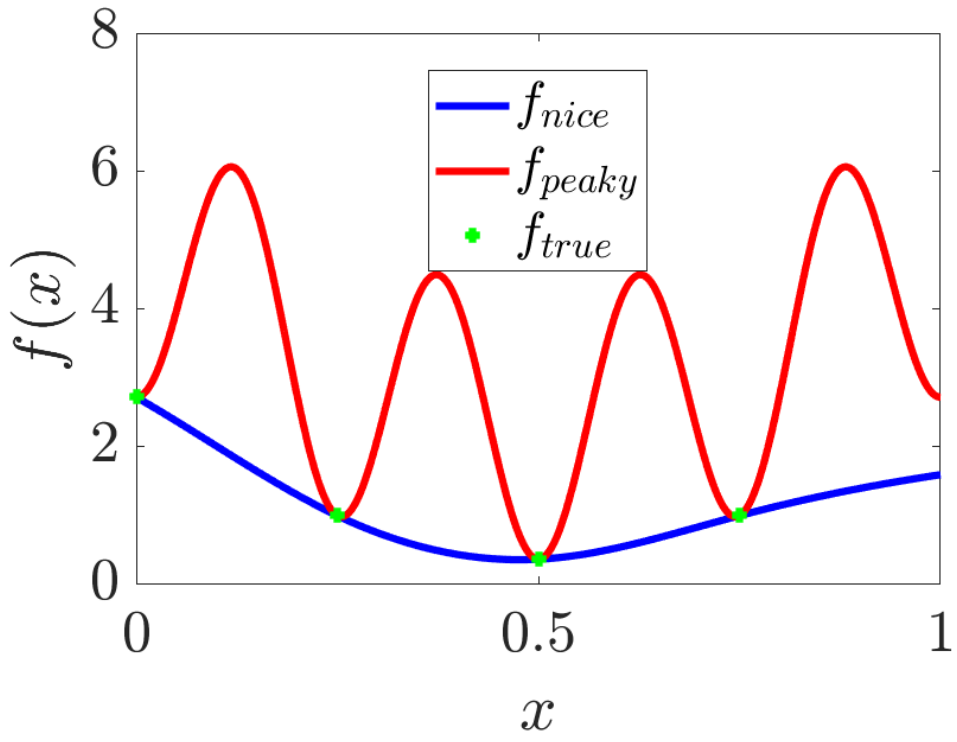


Figure 2.1. Example integrands 1) f_{nice}, a smooth function, 2) f_{peaky}, a peaky function, and samples from f_{true}, the true integrand. All have the same values at $\{\boldsymbol{x}_i\}_{i=1}^n$.

the μ_{nice} falls within.

Our Bayesian Cubature algorithms compute the approximate integral using only the samples of the integrand. Estimated integral value of our algorithm closely match the integral of a smooth function that falls within the middles of the confidence interval. If the true integrand were to resemble the smooth approximate function then the estimated integral will be accurate.

2.4 The Automatic Bayesian Cubature Algorithm

The previous section presents three credible intervals, (2.6), (2.8), and (2.11), for the μ, the desired integral. Each credible interval is based on different assumptions about the hyperparameters m, s, and $\boldsymbol{\theta}$. We stress that one must estimate these

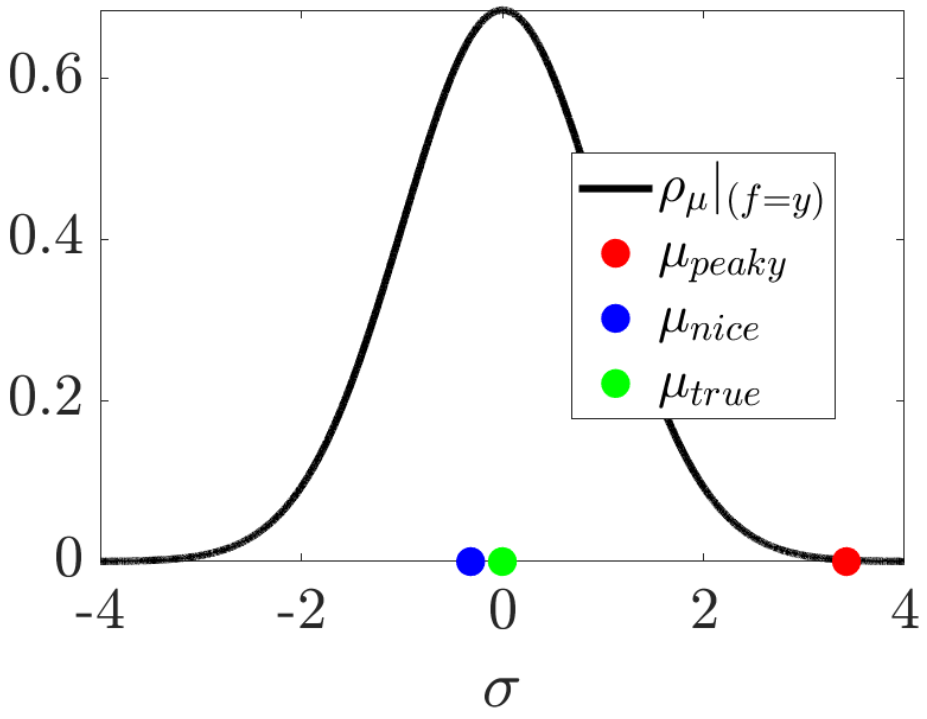


Figure 2.2. Probablity distributions showing the relative position integral of a smooth and a peaky function. f_{nice} lies within the center 99% of the confidence interval, and f_{peaky} lies on the outside of 99% of the confidence interval.

hyperparameters or assume a prior distribution on them because the credible intervals are used as stopping criteria for our cubature rule. Since a credible interval makes a statement about a typical function—not an outlier—one must try to ensure that the integrand is a typical draw from the assumed Gaussian process.

Our Bayesian cubature algorithm increases the sample size until the width of the credible interval is small enough. This is accomplished through successively doubling the sample size. The steps are detailed in Algorithm 1.

We recognize that multiple applications of our credible intervals in one run of the algorithm is not strictly justified. However, if our integrand comes from the middle of the sample space and not the extremes, we expect our automatic Bayesian cubature to approximate the integral within the desired error tolerance with high

probability. The example in the next section and the examples in chapter 7 support that expectation. We also believe that an important factor contributing to the occasional failure of our algorithm is unreasonable parameterizations of the stochastic process from which the integrand is hypothesized to be drawn. Overcoming this latter challenge is a topic for future research.

Algorithm 1 Automatic Bayesian Cubature

Require: a generator for the sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$; a black-box function, f; an absolute error tolerance, $\varepsilon > 0$; the positive initial sample size, n_0; the maximum sample size $n_{\max}$

1: $n \leftarrow n_0,\ n' \leftarrow 0,\ \mathrm{err} \leftarrow \infty$
2: **while** err $> \varepsilon$ and $n \leq n_{\max}$ **do**
3: Generate $\{\boldsymbol{x}_i\}_{i=n'+1}^{n}$ and sample $\{f(\boldsymbol{x}_i)\}_{i=n'+1}^{n}$
4: Compute $\boldsymbol{\theta}$ by (2.15) or (2.16)
5: Compute err according to (2.19), (2.20), or (2.10)
6: $n' \leftarrow n,\ n \leftarrow 2n'$
7: **end while**
8: Sample size to compute $\widehat{\mu}$, $n \leftarrow n'$
9: Compute $\widehat{\mu}$, the approximate integral, according to (2.17) or (2.18)
10: **return** $\widehat{\mu}$, n and err

As described above, the computational cost of Algorithm 1 is the sum of the following:

- $\mathcal{O}\big(n\$(f)\big)$ for the integrand data, where $\$(f)$ is the computational cost of a single $f(\boldsymbol{x})$; $\$(f)$ may be large if it is the result of an expensive simulation; $\$(f)$ is typically proportional to d;
- $\mathcal{O}\big(N_{\mathrm{opt}} n^2 \$(C_{\boldsymbol{\theta}})\big)$ for the evaluation of the Gram matrix $\mathsf{C}_{\boldsymbol{\theta}}$, N_{opt} is the number of optimization steps required, and $\$(C_{\boldsymbol{\theta}})$ is the computational cost of a single

$C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x})$; $\$(C_{\boldsymbol{\theta}})$ is typically proportional to d; and

- $\mathcal{O}\big(N_{\text{opt}} n^3\big)$ for the matrix inversions and determinant calculations; this cost is independent of d.

As we see in the example in the next section, the cost increases quickly as the n required to meet the error tolerance increases. This motivates the fast Bayesian cubature algorithm presented in Chapter 3.

2.5 Example with the Matérn Kernel

To demonstrate automatic Bayesian cubature consider a Matérn covariance kernel:

$$C_\theta(\boldsymbol{x}, \boldsymbol{t}) = \prod_{k=1}^{d} \exp(-\theta|\boldsymbol{x}_k - \boldsymbol{t}_k|)(1 + \theta|\boldsymbol{x}_k - \boldsymbol{t}_k|).$$

Also, consider the integration problem of evaluating *multivariate Gaussian probabilities*:

$$\mu = \int_{(\boldsymbol{a},\boldsymbol{b})} \frac{\exp\big(-\frac{1}{2}\boldsymbol{t}^T \Sigma^{-1} \boldsymbol{t}\big)}{\sqrt{(2\pi)^d \det(\Sigma)}} \, \mathrm{d}\boldsymbol{t}, \tag{2.22}$$

where $(\boldsymbol{a}, \boldsymbol{b})$ is a finite, semi-infinite or infinite box in $\mathbb{R}^d$. This integral does not have an analytic expression for general Σ, so cubatures are required.

Genz [21] introduced a variable transformation to transform (2.22) into an integral on the unit cube. Not only does this variable transformation accommodate domains that are (semi-)infinite, it also tends to smooth out the integrand better, which expedites the cubature. Let $\Sigma = \mathsf{L}\mathsf{L}^T$ be the Cholesky decomposition where $\mathsf{L} = (l_{jk})_{j,k=1}^d$ is a lower triangular matrix. Iteratively define

$$\alpha_1 = \Phi(a_1), \qquad \beta_1 = \Phi(b_1),$$
$$\alpha_j(x_1, ..., x_{j-1}) = \Phi\left(\frac{1}{l_{jj}}\left(a_j - \sum_{k=1}^{j-1} l_{jk} \Phi^{-1}(\alpha_k + x_k(\beta_k - \alpha_k))\right)\right), \quad j = 2, ..., d,$$

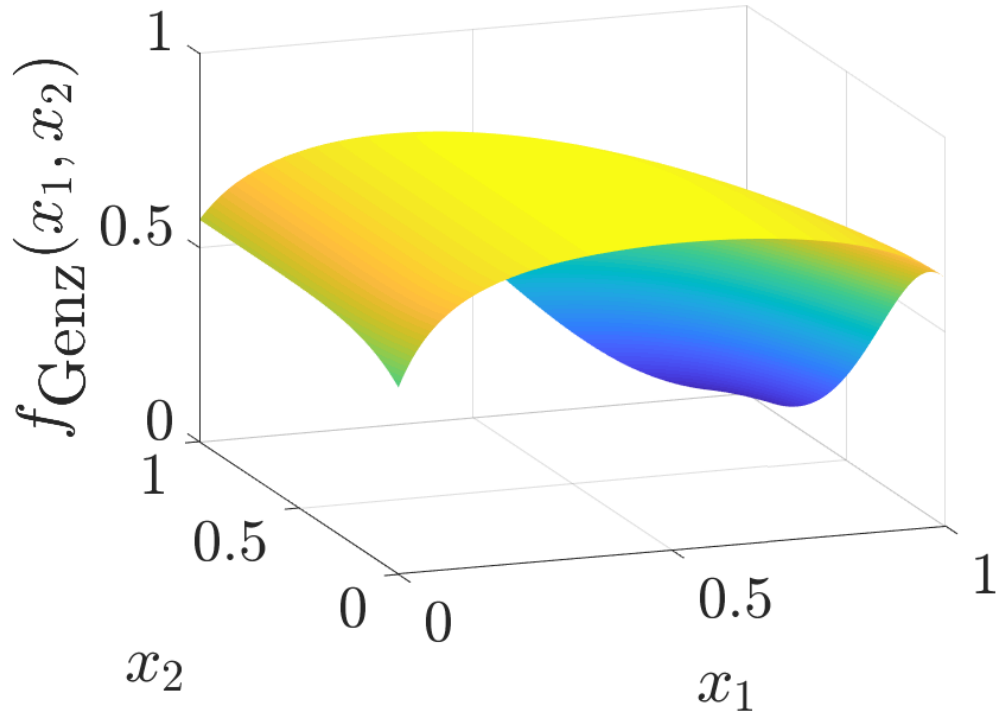


Figure 2.3. The $d = 3$ multivarivate normal probability transformed to an integral of f_{Genz} with $d = 2$. This plot can be reproduced using `IntegrandPlots.m` in GAIL.

$$\beta_j(x_1, ..., x_{j-1}) = \Phi\left(\frac{1}{l_{jj}}\left(b_j - \sum_{k=1}^{j-1} l_{jk}\Phi^{-1}(\alpha_k + x_k(\beta_k - \alpha_k))\right)\right), \quad j = 2, ..., d,$$

$$f_{\text{Genz}}(\boldsymbol{x}) = \prod_{j=1}^{d}[\beta_j(\boldsymbol{x}) - \alpha_j(\boldsymbol{x})]. \tag{2.23}$$

where Φ is the cumulative standard normal distribution function. Then,

$$\mu = \int_{[0,1]^{d-1}} f_{\text{Genz}}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}.$$

This approach transforms a d' dimensional integral into a $d = d' - 1$ dimensional integral.

We use the following parameter values in the simulation:

$$d = 3, \quad \boldsymbol{a} = \begin{pmatrix} -6 \\ -2 \\ -2 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 5 \\ 2 \\ 1 \end{pmatrix}, \quad \mathsf{L} = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 1 & 0.5 \\ 0 & 0 & 0.25 \end{pmatrix}.$$

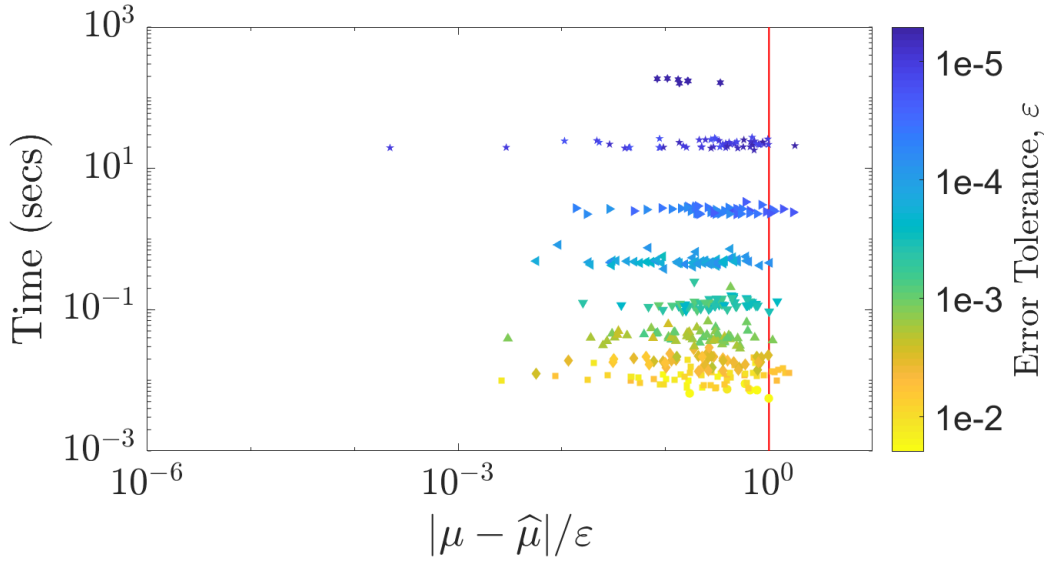


Figure 2.4. Multivariate Gaussian probability: Guaranteed integration using Matérn kernel in $d = 2$ using empirical Bayes stopping criterion within error tolerance ε. This figure can be conditionally reproduced using `matern_guaranteed_plots.m` in GAIL.

The node sets are randomly scrambled Sobol' points [22, 23]. The results are for 400 randomly chosen ε in the interval $[10^{-5}, 10^{-2}]$ as shown in Figure 2.4. In each run, the nodes are randomly scrambled. We observe the algorithm meets the error criterion 95% of the time even-though we used 99% credible intervals. One possible explanation is that the matrix inversions in the algorithm are ill-conditioned leading to numerical inaccuracies. Another possible explanation is that this Matérn covariance kernel is not a good match for the integrand.

On our test computer, it took more than an hour to compute $\widehat{\mu}_n$ with $n = 2^{14}$. As shown in Figure 2.5, the computation time increases rapidly with n. The empirical

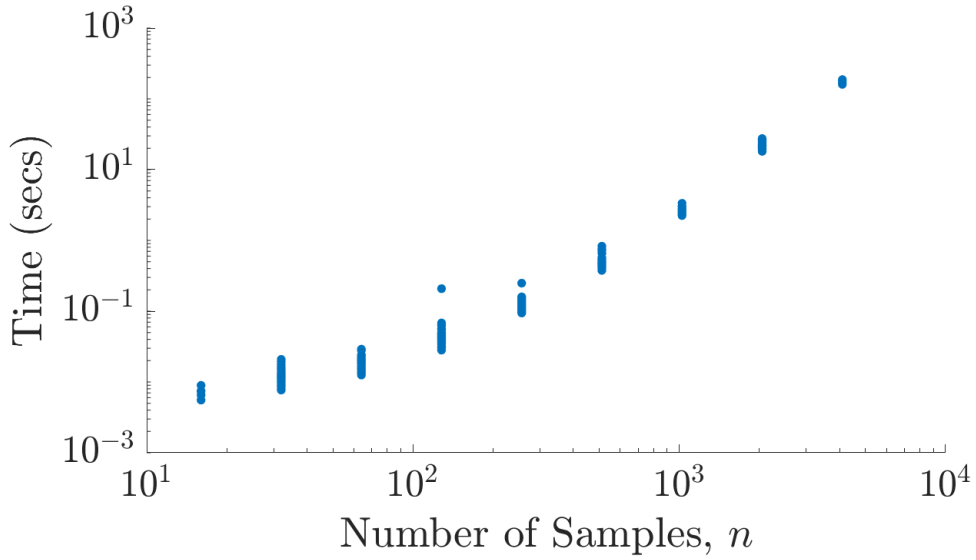


Figure 2.5. Multivariate Gaussian probability estimated using Matérn kernel in $d = 2$ using empirical Bayes stopping criterion. Computation time rapidly increases with increase of n. This figure can be conditionally reproduced using `matern_guaranteed_plots.m` in GAIL.

Bayes estimation of $\boldsymbol{\theta}$, which requires repeated evaluation of the objective function, is the most time consuming of all. Because the objective function needs to be computed multiple times in every iteration to find its minimum. It takes tens of seconds to compute $\widehat{\mu}_n$ with $\varepsilon = 10^{-5}$. In contrast, this example in Chapter 7 take less than a hundredth of a second to compute $\widehat{\mu}_n$ with the same ε using our new algorithm. Not only is the Bayesian cubature with the Matérn kernel slow, but also $\mathsf{C}_{\boldsymbol{\theta}}$ becomes highly ill-conditioned as n increases. So, Algorithm 1 in its current form is impractical when n must be large.

CHAPTER 3

FAST AUTOMATIC BAYESIAN CUBATURE

The generic automatic Bayesian cubature algorithm described in the previous section requires $\mathcal{O}\big(n\$(f) + N_{\text{opt}}[n^2\$(C_{\boldsymbol{\theta}}) + n^3]\big)$ operations to compute the cubature. Now we explain how to speed up the calculations. A key is to choose covariance kernels that match the nodes, $\{\boldsymbol{x}_i\}_{i=1}^n$, so that the vector-matrix operations required by Bayesian cubature can be accomplished using fast Bayesian transforms at a computational cost of $\mathcal{O}\big(n\$(f) + N_{\text{opt}}[n\$(C_{\boldsymbol{\theta}}) + n\log(n)]\big)$.

3.1 Fast Bayesian Transform Kernel

We make some assumptions about the relationship between the covariance kernel and the nodes. In chapter 4 these assumptions are shown to hold for rank-1 lattices and shift-invariant kernels and again in chapter 5 to hold for Sobol' nodes and Walsh kernels. Although the integrands and covariance kernels are real, it is convenient to allow related vectors and matrices to be complex. A relevant example is the fast Fourier transform (FFT) of a real-valued vector, which is a complex-valued vector.

We introduce some further notation

$$\begin{aligned}
\mathsf{C} = \mathsf{C}_{\boldsymbol{\theta}} &= \Big(C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big)_{i,j=1}^n = (\boldsymbol{C}_1, ..., \boldsymbol{C}_n) \\
&= \frac{1}{n}\mathsf{V}\mathsf{\Lambda}\mathsf{V}^H, \qquad \mathsf{V}^H = n\mathsf{V}^{-1}, \\
\mathsf{V} &= (\boldsymbol{v}_1, ..., \boldsymbol{v}_n)^T = (\boldsymbol{V}_1, ..., \boldsymbol{V}_n) \\
\mathsf{C}^p &= \frac{1}{n}\mathsf{V}\mathsf{\Lambda}^p\mathsf{V}^H, \qquad \forall p \in \mathbb{Z},
\end{aligned} \tag{3.1}$$

where V^H is the Hermitian of V, $\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n$ are columns of C, $\boldsymbol{V}_1, \ldots, \boldsymbol{V}_n$ are columns of V, and $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ are rows of V. In this and later sections, we drop the $\boldsymbol{\theta}$ dependence of various quantities for simplicity of notation. The normalization of V assumed in

(3.1) conveniently allows the first eigenvector, $\boldsymbol{V}_1$, to be the vector of ones in (3.2b) below. The columns of matrix V are eigenvectors of C, and Λ is a diagonal matrix of eigenvalues of C. For any $n \times 1$ vector $\boldsymbol{b}$, define the notation $\widetilde{\boldsymbol{b}} := \mathsf{V}^H \boldsymbol{b}$.

We make three assumptions that allow the fast computation:

$$\mathsf{V} \text{ may be identified analytically,} \tag{3.2a}$$

$$\boldsymbol{v}_1 = \boldsymbol{V}_1 = \mathbf{1}, \tag{3.2b}$$

$$\mathsf{V}^H \boldsymbol{b} \text{ requires only } \mathcal{O}(n \log n) \text{ operations } \forall \boldsymbol{b}. \tag{3.2c}$$

We call the transformation $\boldsymbol{b} \mapsto \mathsf{V}^H \boldsymbol{b}$ a *fast Bayesian transform* and $C_{\boldsymbol{\theta}}$ a *fast Bayesian transform kernel*.

Under assumptions (3.2) the eigenvalues may be identified as the fast Bayesian transform of the first column of C:

$$\begin{aligned} \boldsymbol{\lambda} = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{pmatrix} = \Lambda \mathbf{1} = \Lambda \boldsymbol{v}_1^* = \underbrace{\left(\frac{1}{n} \mathsf{V}^H \mathsf{V} \right)}_{\mathsf{I}} \Lambda \boldsymbol{v}_1^* \\ = \mathsf{V}^H \left(\frac{1}{n} \mathsf{V} \Lambda \boldsymbol{v}_1^* \right) = \mathsf{V}^H \boldsymbol{C}_1 = \widetilde{\boldsymbol{C}}_1, \end{aligned} \tag{3.3}$$

where I is the identity matrix and $\boldsymbol{v}_1^*$ is the complex conjugate of the first row of V. Also note that the fast transform of $\mathbf{1}$ has a simple form

$$\widetilde{\mathbf{1}} = \mathsf{V}^H \mathbf{1} = \mathsf{V}^H \boldsymbol{V}_1 = \begin{pmatrix} n, \\ 0, \\ \vdots, \\ 0 \end{pmatrix}.$$

Many of the terms that arise in the calculations in Algorithm 1 take the form $\boldsymbol{a}^T \mathsf{C}^p \boldsymbol{b}$ for real $\boldsymbol{a}$ and $\boldsymbol{b}$ and integer p. These can be calculated via the transforms

$\widetilde{\boldsymbol{a}} = \mathsf{V}^H\boldsymbol{a}$ and $\widetilde{\boldsymbol{b}} = \mathsf{V}^H\boldsymbol{b}$ as

$$\boldsymbol{a}^T\mathsf{C}^p\boldsymbol{b} = \frac{1}{n}\boldsymbol{a}^T\mathsf{V}\Lambda^p\mathsf{V}^H\boldsymbol{b} = \frac{1}{n}\widetilde{\boldsymbol{a}}^H\Lambda^p\widetilde{\boldsymbol{b}} = \frac{1}{n}\sum_{i=1}^{n}\lambda_i^p\widetilde{a}_i^*\widetilde{b}_i,$$

Note that $\widetilde{\boldsymbol{a}}^*$ appears on the right side of this equation because $\boldsymbol{a}^T\mathsf{V} = (\mathsf{V}^H\boldsymbol{a})^* = \widetilde{\boldsymbol{a}}^*$. In particular,

$$\begin{aligned}
\mathbf{1}^T\mathsf{C}^{-p}\mathbf{1} &= \frac{n}{\lambda_1^p}, & \mathbf{1}^T\mathsf{C}^{-p}\boldsymbol{y} &= \frac{\widetilde{y}_1}{\lambda_1^p},\\
\boldsymbol{y}^T\mathsf{C}^{-p}\boldsymbol{y} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^p}, & \boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1} &= \frac{\widetilde{c}_1}{\lambda_1},\\
\boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y} &= \frac{1}{n}\sum_{i=1}^{n}\frac{\widetilde{c}_i^*\widetilde{y}_i}{\lambda_i}, & \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i},
\end{aligned}$$

where $\widetilde{\boldsymbol{y}} = \mathsf{V}^H\boldsymbol{y}$ and $\widetilde{\boldsymbol{c}} = \mathsf{V}^H\boldsymbol{c}$. For any real $\boldsymbol{b}$, with $\widetilde{\boldsymbol{b}} = \mathsf{V}^H\boldsymbol{b}$, it follows that $\widetilde{b}_1$ is real since the first row of V^H is $\mathbf{1}$.

The covariance kernel used in practice also may satisfy an additional assumption:

$$\int_{[0,1]^d} C(\boldsymbol{t},\boldsymbol{x})\,\mathrm{d}\boldsymbol{t} = 1 \qquad \forall\boldsymbol{x}\in[0,1]^d, \tag{3.4}$$

which implies that $c_{0\boldsymbol{\theta}} = 1$ and $\boldsymbol{c}_{\boldsymbol{\theta}} = \mathbf{1}$. Under (3.4), the expressions above may be further simplified:

$$\boldsymbol{c}^T\mathsf{C}^{-1}\mathbf{1} = \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c} = \frac{n}{\lambda_1}.$$

We use the fast Bayesian transform to speedup the computation of hyperparameter $\boldsymbol{\theta}$, the credible interval width $\mathrm{err}_{\mathrm{CI}}$, and interval $\widehat{\mu}$ that we presented in theorem 2.2.1 as shown next.

3.2 Empirical Bayes

Under assumptions (3.2), the empirical Bayes parameters in (2.12), (2.13), (2.15) (2.17), and (2.19) can be expressed in terms of the fast transforms of the function data, the first column of the Gram matrix, and $\boldsymbol{c}$ as follows:

$$m_{\mathrm{MLE}} = \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

$$
\begin{aligned}
s^2_{\mathrm{MLE}} &= \frac{1}{n^2}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i},\\
\boldsymbol{\theta}_{\mathrm{MLE}} &= \operatorname*{argmin}_{\boldsymbol{\theta}}\left[\log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right)+\frac{1}{n}\sum_{i=1}^{n}\log(\lambda_i)\right],\\
\widehat{\mu}_{\mathrm{MLE}} &= \frac{\widetilde{y}_1}{n}+\frac{1}{n}\sum_{i=2}^{n}\frac{\widetilde{c}_i^*\widetilde{y}_i}{\lambda_i},\\
\mathrm{err}_{\mathrm{MLE}} &= \frac{2.58}{n}\sqrt{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(c_0-\frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)},
\end{aligned}
$$

Since all the quantities on the right hand sides can be obtained in $\mathcal{O}(n\log n)$ operations by fast transforms, the left hand sides are all computable using the asymptotic computational cost.

Under the further assumption (3.4) it follows that

$$
\begin{aligned}
\widehat{\mu}_{\mathrm{MLE}} &= \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i,\\
\mathrm{err}_{\mathrm{MLE}} &= \frac{2.58}{n}\sqrt{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(1-\frac{n}{\lambda_1}\right)}.
\end{aligned}
$$

Thus, in this case $\widehat{\mu}$ is simply the sample mean.

3.2.1 Gradient of the objective function using fast transform. In the previous chapter, we discussed about using gradient descent for hyperparameter search but the computational cost is of $\mathcal{O}(N_{\mathrm{opt}}n^3)$. If V does not depend on $\boldsymbol{\theta}$ then one can fast compute the derivative of Gram matrix C. Starting from the definition (3.1) and taking derivative w.r.t. θ_j,

$$
\frac{\partial \mathsf{C}}{\partial\theta_j} = \frac{1}{n}\mathsf{V}\frac{\partial\Lambda}{\partial\theta_j}\mathsf{V}^H = \frac{1}{n}\mathsf{V}\bar{\Lambda}_{(j)}\mathsf{V}^H,
$$

$$
\text{where}\quad \bar{\Lambda}_{(j)} = \mathrm{diag}(\bar{\boldsymbol{\lambda}}_{(j)}),\quad \text{and}
$$

$$
\bar{\boldsymbol{\lambda}}_{(j)} = \frac{\partial\boldsymbol{\lambda}}{\partial\theta_j} = \left(\frac{\partial\lambda_i}{\partial\theta_j}\right)_{i=1}^{n} = \left(\frac{\partial}{\partial\theta_j}\mathsf{V}^H\boldsymbol{C}_1\right) = \mathsf{V}^H\left(\frac{\partial}{\partial\theta_j}C_{\boldsymbol{\theta}}(\boldsymbol{x}_1,\boldsymbol{x}_i)\right)_{i=1}^{n}.
$$

where we used the fast transform property (3.3). We use the notation $\bar{\boldsymbol{\lambda}}_{(j)} = \mathsf{V}^H\bar{\boldsymbol{C}}_{1(j)}$,

where $\bar{\boldsymbol{C}}_{1(j)}$ denotes the first row of the gram matrix after taking derivative, i.e.

$$\bar{\boldsymbol{C}}_{1(j)} = \left(\frac{\partial}{\partial \theta_j} C_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \boldsymbol{x}_i) \right)_{i=1}^{n}.$$

The goal is to find derivative of the objective function. First, let's rewrite the objective function from (3.6a),

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) &= \underbrace{\frac{1}{n}\log(\det\ \mathsf{C})}_{\mathcal{L}_{|\mathsf{C}|}} + \underbrace{\log\left((\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T \mathsf{C}^{-1}(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})\right)}_{\mathcal{L}_{\boldsymbol{y}}}, \\ &=: \mathcal{L}_{|\mathsf{C}|} + \mathcal{L}_{\boldsymbol{y}}.\end{aligned}$$

Now, take the derivative,

$$\frac{\partial}{\partial \theta_j}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{\partial}{\partial \theta_j}\mathcal{L}_{|\mathsf{C}|} + \frac{\partial}{\partial \theta_j}\mathcal{L}_{\boldsymbol{y}}\ .$$

Let's tackle the individual terms,

$$\begin{aligned}\frac{\partial}{\partial \theta_j}\mathcal{L}_{|\mathsf{C}|} &= \frac{\partial}{\partial \theta_j}\frac{1}{n}\log(\det\ \mathsf{C}), \\ &= \frac{1}{n}\text{trace}\left(\mathsf{C}^{-1}\frac{\partial \mathsf{C}}{\partial \boldsymbol{\theta}_j}\right) = \frac{1}{n}\text{trace}\left(\mathsf{V}\boldsymbol{\Lambda}^{-1}\mathsf{V}^H\frac{1}{n}\mathsf{V}\overline{\boldsymbol{\Lambda}}_{(j)}\mathsf{V}^H\right), \\ &= \frac{1}{n}\text{trace}\big(\mathsf{V}\boldsymbol{\Lambda}^{-1}\overline{\boldsymbol{\Lambda}}_{(j)}\mathsf{V}^H\big), \quad \text{where we used } \ \mathsf{V}^H\mathsf{V} = n, \\ &= \frac{1}{n}\text{trace}\left(\mathsf{V}\ \text{diag}\left(\frac{\overline{\lambda}_{i(j)}}{\lambda_i}\right)_{i=1}^{n}\mathsf{V}^H\right) = \frac{1}{n}\sum_{i=1}^{n}\frac{\overline{\lambda}_{i(j)}}{\lambda_i},\end{aligned}$$

where we used the fact from [24],

$$\log(\det\ \mathsf{C}) = \text{trace}(\log(\mathsf{C})).$$

Part of the $\mathcal{L}_{\boldsymbol{y}}$ was already simplified using the fast transform,

$$(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m_{\text{MLE}}\mathbf{1}) = \frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}.$$

Using the above result,

$$\frac{\partial}{\partial \theta_j}\mathcal{L}_{\boldsymbol{y}} = \frac{\partial}{\partial \boldsymbol{\theta}_j}\log\left(\frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right),$$

$$
\begin{aligned}
&= \left(\frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right)^{-1} \frac{\partial}{\partial \boldsymbol{\theta}_j}\left(\frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right), \\
&= \left(\frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right)^{-1} \frac{1}{n}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left(-\frac{\partial \lambda_i}{\partial \boldsymbol{\theta}_j}\right), \\
&= -\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right)^{-1}\left(\sum_{i=2}^{n}|\widetilde{y}_i|^2\frac{\bar{\lambda}_{i(j)}}{\lambda_i^2}\right).
\end{aligned}
$$

Finally, using the above results,

$$
\frac{\partial}{\partial \theta_j}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\bar{\lambda}_{i(j)}}{\lambda_i} - \left(\sum_{i=2}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2\,\bar{\lambda}_{i(j)}}{\lambda_i^2}\right)\left(\sum_{i=2}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2}{\lambda_j}\right)^{-1}.
$$

If $m = 0$ assumption can be made,

$$
\frac{\partial}{\partial \theta_j}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\bar{\lambda}_{i(j)}}{\lambda_i} - \left(\sum_{i=1}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2\,\bar{\lambda}_{i(j)}}{\lambda_i^2}\right)\left(\sum_{i=1}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2}{\lambda_j}\right)^{-1}.
$$

3.3 Full Bayes

For the full Bayes approach the cubature is the same as for empirical Bayes. We also defer to empirical Bayes to estimate the parameter $\boldsymbol{\theta}$. The width of the confidence interval is $\mathrm{err}_{\mathrm{full}} := t_{n_j-1,0.995}\widehat{\sigma}_{\mathrm{full}}$, where $\widehat{\sigma}^2_{\mathrm{full}}$ can also be computed swiftly under assumptions (3.2):

$$
\widehat{\sigma}^2_{\mathrm{full}} = \frac{1}{n(n-1)}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left[\frac{\lambda_1}{n}\left(1-\frac{\widetilde{c}_1}{\lambda_1}\right)^2 + \left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)\right],
$$

Under assumption (3.4) further simplification can be made:

$$
\widehat{\sigma}^2_{\mathrm{full}} = \frac{1}{n(n-1)}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(\frac{\lambda_1}{n}-1\right),
$$

It follows that

$$
\mathrm{err}_{\mathrm{full}} = t_{n_j-1,0.995}\sqrt{\frac{1}{n(n-1)}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(\frac{\lambda_1}{n}-1\right)}.
$$

3.4 Generalized Cross-Validation

GCV yields a different cubature, which nevertheless can also be computed quickly using the fast transform. Under assumptions (3.2):

$$
\begin{aligned}
m_{\mathrm{GCV}} &= m_{\mathrm{MLE}} = \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i, \\
s^2_{\mathrm{GCV}} &:= \frac{1}{n}\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1}, \\
\boldsymbol{\theta}_{\mathrm{GCV}} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right) - 2\log\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)\right], \\
\widehat{\mu}_{\mathrm{GCV}} &= \widehat{\mu}_{\mathrm{MLE}} = \frac{\widetilde{y}_1}{n} + \frac{1}{n}\sum_{i=2}^{n}\frac{\widetilde{c}_i^{*}\widetilde{y}_i}{\lambda_i}, \\
\mathrm{err}_{\mathrm{GCV}} &= \frac{2.58}{n}\left\{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1} \times \left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)\right\}^{1/2}.
\end{aligned}
$$

Moreover, under further assumption (3.4) it follows that

$$
\begin{aligned}
\widehat{\mu}_{\mathrm{GCV}} &= \widehat{\mu}_{\mathrm{MLE}} = \widehat{\mu}_{\mathrm{full}} = \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i, \\
\mathrm{err}_{\mathrm{GCV}} &= \frac{2.58}{n}\left\{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1} \times \left(1 - \frac{n}{\lambda_1}\right)\right\}^{1/2}.
\end{aligned}
$$

In this case too, $\widehat{\mu}$ is simply the sample mean.

3.4.1 Gradient of the objective function. Using the results obtained from the Section Section 3.2.1 with empirical Bayes, one can reduce the computational cost of the derivative of the loss function,

$$
\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})_{\mathrm{GCV}} = \log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right) - 2\log\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)
$$

Using the similar techniques from Section 3.2.1

$$
\begin{aligned}
&\frac{\partial}{\partial\theta_j}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y})_{\mathrm{GCV}} \\
&= \left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right)^{-1}\frac{\partial}{\partial\theta_j}\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right) - 2\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)^{-1}\frac{\partial}{\partial\theta_j}\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)
\end{aligned}
$$

$$= \left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right)^{-1} \left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^3}(-2)\frac{\partial \lambda_i}{\partial \theta_j}\right) - 2\left(\sum_{i=1}^{n} \frac{1}{\lambda_i}\right)^{-1} \left(\sum_{i=1}^{n} \frac{1}{\lambda_i^2}(-1)\frac{\partial \lambda_i}{\partial \theta_j}\right)$$

$$= -2\left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right)^{-1} \left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2 \bar{\lambda}_{i(j)}}{\lambda_i^3}\right) + 2\left(\sum_{i=1}^{n} \frac{1}{\lambda_i}\right)^{-1} \left(\sum_{i=1}^{n} \frac{\bar{\lambda}_{i(j)}}{\lambda_i^2}\right).$$

3.5 Product kernels

In this research, we use product kernels in the demonstrations and numerical implementations. They got nice properties which are helpful to obtain analytical results easily. Product kernels in d dimensions are of the form,

$$C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = \prod_{l=1}^{d} \Big[1 - \eta\, \mathfrak{C}(x_l, t_l)\Big] \tag{3.5}$$

where η is called shape parameter and $\mathfrak{C}$ is some positive definite function. The derivative of the product kernel can be obtained easily.

3.5.1 Derivative of the product kernel. It was suggested to use gradient descent to find optimal shape parameter in Section 2.2.1.1. In this section, we compute the gradient for product kernels. When the η is common across the dimensions, the derivative of a product kernel w.r.t. η can be obtained as below,

$$\begin{aligned}
\frac{\partial}{\partial \eta} C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) &= \frac{\partial}{\partial \eta} \prod_{l=1}^{d} \Big[1 - \eta \mathfrak{C}(x_l, t_l)\Big], \\
&= \sum_{j=1}^{d} \prod_{l=1, l\neq j}^{d} \Big[1 - \eta \mathfrak{C}(x_l, t_l)\Big] \Big(-\mathfrak{C}(x_j, t_j)\Big) \\
&= \prod_{l=1}^{d} \Big[1 - \eta \mathfrak{C}(x_l, t_l)\Big] \sum_{j=1}^{d} \frac{\Big(-\mathfrak{C}(x_j, t_j)\Big)}{1 - \eta \mathfrak{C}(x_j, t_j)} \\
&= C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) \frac{1}{\eta} \sum_{j=1}^{d} \frac{\Big(1 - \eta \mathfrak{C}(x_j, t_j) - 1\Big)}{1 - \eta \mathfrak{C}(x_j, t_j)}
\end{aligned}$$

$$= C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})\frac{1}{\eta}\sum_{j=1}^{d}\left(1-\frac{1}{1-\eta\mathfrak{C}(x_j,t_j)}\right)$$
$$= (d/\eta)\underbrace{\left(\prod_{l=1}^{d}\left[1-\eta\mathfrak{C}(x_l,t_l)\right]\right)}_{C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})}\left(1-\frac{1}{d}\sum_{j=1}^{d}\frac{1}{1-\eta\mathfrak{C}(x_j,t_j)}\right).$$

Thus,

$$\frac{\partial}{\partial\eta}C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x}) = (d/\eta)C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})\left(1-\frac{1}{d}\sum_{j=1}^{d}\frac{1}{1-\eta\mathfrak{C}(x_j,t_j)}\right).$$

3.5.1.1 When η_j is chosen for each dimension j. In this case, we will have a vector of length d shape parameters. Derivative of the kernel with respect to dimension j is,

$$\begin{aligned}
\frac{\partial}{\partial\eta_j}C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x}) &= \frac{\partial}{\partial\eta_j}\prod_{l=1}^{d}\left[1-\eta_l\mathfrak{C}(x_l,t_l)\right]\\
&= \prod_{l=1,l\neq j}^{d}\left[1-\eta_l\mathfrak{C}(x_l,t_l)\right]\left(-\mathfrak{C}(x_j,t_j)\right)\\
&= \prod_{l=1}^{d}\left[1-\eta_l\mathfrak{C}(x_l,t_l)\right]\frac{\left(-\mathfrak{C}(x_j,t_j)\right)}{1-\eta_j\mathfrak{C}(x_j,t_j)}\\
&= C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})\frac{1}{\eta_j}\frac{\left(1-\eta_j\mathfrak{C}(x_j,t_j)-1\right)}{1-\eta_j\mathfrak{C}(x_j,t_j)}\\
&= C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})\frac{1}{\eta_j}\left(1-\frac{1}{1-\eta_j\mathfrak{C}(x_j,t_j)}\right)\\
&= \frac{1}{\eta_j}\underbrace{\left(\prod_{l=1}^{d}\left[1-\eta\mathfrak{C}(x_l,t_l)\right]\right)}_{C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})}\left(1-\frac{1}{1-\eta_j\mathfrak{C}(x_j,t_j)}\right).
\end{aligned}$$

Thus,

$$\frac{\partial}{\partial\eta_j}C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x}) = \frac{1}{\eta_j}C_{\boldsymbol{\theta}}(\boldsymbol{t},\boldsymbol{x})\left(1-\frac{1}{1-\eta_j\mathfrak{C}(x_j,t_j)}\right).$$

Please note the above derivatives do not depend on $\mathfrak{C}(x,t)$ and most importantly these computations are applicable to any product kernel of the form (3.5).

3.5.2 Shape parameter search using steepest descent. Using the obtained derivative above, one can easily implement the steepest descent search introduced in Section 2.2.1.1

$$\eta^{(j+1)} = \eta^{(j)} - \nu \frac{\partial}{\partial \eta} \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}).$$

3.6 Summary

The assumptions and derivations in this chapter lead to the following theorem.

Theorem 3.6.1. *Under assumptions* (3.2)*, the parameters and credible interval half-widths in Theorem 2.2.1 may be expressed in terms of the fast Bayesian transforms of the integrand data, the first column of the Gram matrix,* c_0*, and* $\boldsymbol{c}$ *as follows:*

$$\begin{aligned}
m_{\text{MLE}} &= m_{\text{full}} = m_{\text{GCV}} = \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i, \\
s^2_{\text{MLE}} &= \frac{1}{n^2}\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i}, \\
\widehat{\sigma}^2_{\text{full}} &= \frac{1}{n(n-1)}\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i}\left[\frac{\lambda_1}{n}\left(1 - \frac{\widetilde{c}_1}{\lambda_1}\right)^2 + \left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)\right], \\
s^2_{\text{GCV}} &= \frac{1}{n}\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1},
\end{aligned}$$

$$\boldsymbol{\theta}_{\text{MLE}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right) + \frac{1}{n}\sum_{i=1}^{n}\log(\lambda_i)\right], \tag{3.6a}$$

$$\boldsymbol{\theta}_{\text{GCV}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right) - 2\log\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)\right], \tag{3.6b}$$

$$\begin{aligned}
\widehat{\mu}_{\text{MLE}} &= \widehat{\mu}_{\text{full}} = \widehat{\mu}_{\text{GCV}} = \frac{\widetilde{y}_1}{n} + \frac{1}{n}\sum_{i=2}^{n}\frac{\widetilde{c}_i^*\widetilde{y}_i}{\lambda_i}, \\
\text{err}_{\text{MLE}} &= \frac{2.58}{n}\sqrt{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)},
\end{aligned}$$

$$\mathrm{err}_{\mathrm{full}} = t_{n-1,0.995}\widehat{\sigma}_{\mathrm{full}},$$

$$\mathrm{err}_{\mathrm{GCV}} = \frac{2.58}{n}\left\{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1}\left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widetilde{c}_i|^2}{\lambda_i}\right)\right\}^{1/2}.$$

Under the further assumption (3.4)*, it follows that*

$$\widehat{\mu}_{\mathrm{MLE}} = \widehat{\mu}_{\mathrm{full}} = \widehat{\mu}_{\mathrm{GCV}} = \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i, \tag{3.7}$$

and so $\widehat{\mu}$ *is simply the sample mean. Also, under assumption* (3.4)*, the credible interval half-widths simplify to*

$$\mathrm{err}_{\mathrm{MLE}} = \frac{2.58}{n}\sqrt{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(1 - \frac{n}{\lambda_1}\right)}, \tag{3.8a}$$

$$\mathrm{err}_{\mathrm{full}} = t_{n-1,0.995}\sqrt{\frac{1}{n(n-1)}\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\left(\frac{\lambda_1}{n} - 1\right)}, \tag{3.8b}$$

$$\mathrm{err}_{\mathrm{GCV}} = \frac{2.58}{n}\left\{\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\frac{1}{n}\sum_{i=1}^{n}\frac{1}{\lambda_i}\right]^{-1}\left(1 - \frac{n}{\lambda_1}\right)\right\}^{1/2}. \tag{3.8c}$$

In the formulas for the credible interval half-widths and $\boldsymbol{\lambda}$ *depends on* $\boldsymbol{\theta}$*, and* $\boldsymbol{\theta}$ *is assumed to take on the values* $\boldsymbol{\theta}_{\mathrm{MLE}}$ *or* $\boldsymbol{\theta}_{\mathrm{GCV}}$ *as appropriate.*

CHAPTER 4

INTEGRATION LATTICES AND SHIFT INVARIANT KERNELS

The preceding sections lay out an automatic Bayesian cubature algorithm whose computational cost is drastically reduced. However, this algorithm relies on covariance kernel functions, $C_{\boldsymbol{\theta}}$ and node sets, $\{\boldsymbol{x}_i\}_{i=1}^n$ that satisfy assumptions (3.2). In this chapter, we demonstrate such a covariance kernel and matching design. When periodic shift-invariant kernels are combined with rank-1 lattice nodes, the resulting Gram matrix is symmetric circulant. This combination also satisfies assumption (3.4). To conveniently facilitate the fast transform, it is assumed in this section and the next that n is power of 2.

4.1 Extensible Integration Lattice Node Sets

We choose set of nodes defined by a shifted extensible integration lattice node sequence, which takes the form

$$\boldsymbol{x}_i = \boldsymbol{h}\phi(i-1) + \boldsymbol{\Delta} \mod \mathbf{1}, \qquad i \in \mathbb{N}. \tag{4.1}$$

Here, $\boldsymbol{h}$ is a d-dimensional generating vector of positive integers, $\boldsymbol{\Delta}$ is some point in $[0,1)^d$, often chosen at random, and $\{\phi(i)\}_{i=0}^n$ is the van der Corput sequence, defined by reflecting the binary digits of the integer about the decimal point, i.e.,

i	0	1	2	3	4	5	6	7	$\cdots$
i	0_2	1_2	10_2	11_2	100_2	101_2	110_2	111_2	$\cdots$
$\phi(i)$	$_2.0$	$_2.1$	$_2.01$	$_2.11$	$_2.001$	$_2.101$	$_2.011$	$_2.111$	$\cdots$
$\phi(i)$	0	0.5	0.25	0.75	0.125	0.625	0.375	0.875	$\cdots$

(4.2)

Note that

$$n\phi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$$

$$\text{is one-to-one,} \quad (4.3)$$

assuming n is a power of 2.

These node sets called rank-1 lattices. A random shift $\boldsymbol{\Delta}$ is added to $\boldsymbol{x}_i$ to avoid the origin zero in the node sets. However, this shift will preserve the discrepancy properties of $\boldsymbol{x}_i$. The rank-1 lattices with the module one addition have a very desirable group structure that helps to satisfy fast Bayesian transform kernel assumptions.

An example of 64 nodes is given in Figure 4.1. The even coverage of the unit cube is ensured by a well chosen generating vector. The choice of generating vector is typically done offline by computer search. See [22] and [25] for more on extensible integration lattices. Lattice rules are designed to integrate the class of sinusoidal functions without error.

4.2 Shift Invariant Kernels

The covariance functions $C_{\boldsymbol{\theta}}$ that match integration lattice node sets have the form

$$C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = K_{\boldsymbol{\theta}}(\boldsymbol{t} - \boldsymbol{x} \bmod \mathbf{1}). \quad (4.4)$$

This is called a *shift invariant kernel* because shifting both arguments of the covariance function by the same amount leaves the value unchanged. By a proper scaling of the kernel $K_{\boldsymbol{\theta}}$ it follows that assumption (3.4) is satisfied. Of course, $K_{\boldsymbol{\theta}}$ is periodic and must be of the form that ensures that $C_{\boldsymbol{\theta}}$ is symmetric and positive definite, as assumed in (2.1).

A family of shift invariant kernels is constructed via even degree Bernoulli polynomials. Symmetric, periodic, positive definite kernels of this form appear in [22]

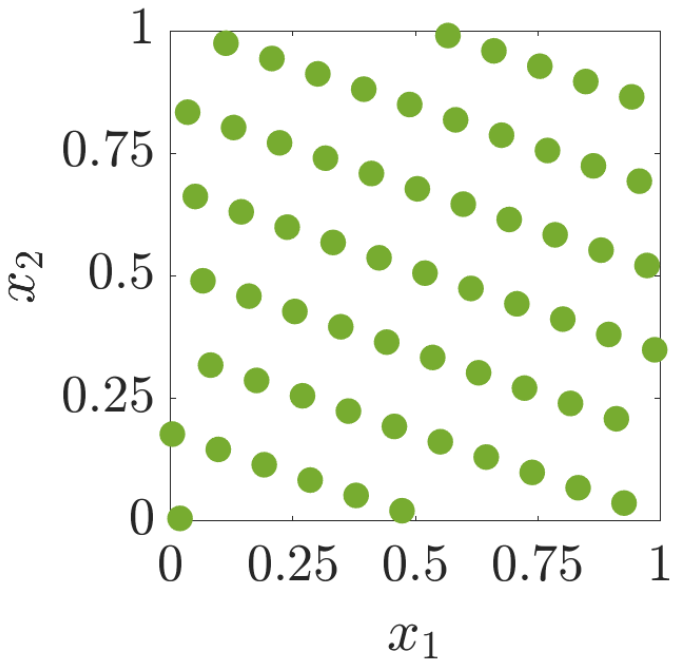


Figure 4.1. Example of a shifted integration lattice node set in $d = 2$

and [26]:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) := \sum_{\boldsymbol{k} \in \mathbb{Z}^d} \alpha_{\boldsymbol{k},\boldsymbol{\theta}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{x}} e^{-2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{t}},$$

where d is number of dimensions and $\alpha_{\boldsymbol{k}}$ is a scalar. The Gram matrix formed by this kernel is Hermitian. The *shape parameter* $\boldsymbol{\theta}$ changes the kernel's shape, so that the function space spanned by the kernel closely resembles the space bearing the integrand. This form of the kernel is very convenient to use in any analytical derivations, but not suitable for use with finite precision computers as this involves infinite sum. If the coefficients are chosen as

$$\alpha_{\boldsymbol{k},\boldsymbol{\theta}} := \prod_{l=1}^{d} \frac{1}{\max(\frac{|k_l|}{\eta_l}, 1)^r_{\eta_l \leq 1}}, \quad \text{with } \alpha_{\boldsymbol{0},\boldsymbol{\theta}} = 1,$$

then there exists a simpler closed form expression without infinite sum

$$K_{\boldsymbol{\theta}}(\boldsymbol{x}) = \prod_{l=1}^{d} \Big[1 - (-1)^r \eta B_{2r}(x_l) \Big], \quad \forall \boldsymbol{x} \in [0,1]^d, \tag{4.5}$$

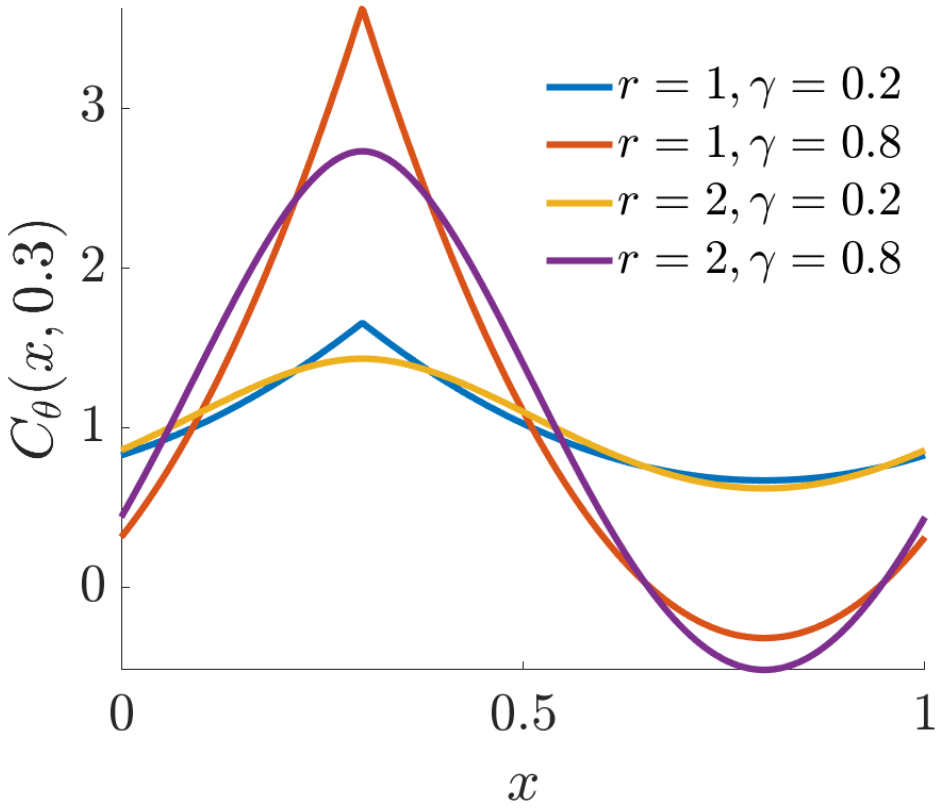


Figure 4.2. Shift invariant kernel in $d = 1$ shifted by 0.3 to show the discontinuity

$$\boldsymbol{\theta} = (r, \eta),\ r \in \mathbb{N},\ \eta > 0. \tag{4.6}$$

The Bernoulli polynomials $B_r(x)$ are described in [27, Chapter 24]

$$B_r(x) = \frac{-r!}{(2\pi\sqrt{-1})^r} \sum_{\substack{k \neq 0, \\ k=-\infty}}^{\infty} \frac{e^{2\pi\sqrt{-1}kx}}{k^r} \quad \begin{cases} \text{for } \ r = 1, & 0 < x < 1 \\ \text{for } \ r = 2, 3, \ldots & 0 \leq x \leq 1 \end{cases}$$

Larger r implies a greater degree of smoothness of the kernel. Larger η implies greater fluctuations of the output with respect to the input. Plots of $C(\cdot, 0.3)$ are given in Figure 4.2 for various r and η values.

Lattice cubature rules are known to have convergence rates that depend on the smoothness of the integrands, but that are rather independent of the choice of the

integration lattice [22]. Thus, we expect integration lattice node sets to perform well regardless of the smoothness of the covariance kernel. The bigger concern is whether the derivatives of the integrand are as smooth as the covariance kernel implies. This topic is touched upon again in Section 4.5.

4.2.1 Eigenvectors. For general shift-invariance covariance functions the Gram matrix takes the form

$$\mathsf{C}_{\boldsymbol{\theta}} = \big(C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\big)_{i,j=1}^{n} \tag{4.7}$$

$$= \Big(K_{\boldsymbol{\theta}}(\boldsymbol{h}(\phi(i-1) - \phi(j-1)) \bmod \mathbf{1})\Big)_{i,j=1}^{n}. \tag{4.8}$$

We now demonstrate that the eigenvector matrix for C is

$$\mathsf{V} = \Big(\mathrm{e}^{2\pi n\sqrt{-1}\phi(i-1)\phi(j-1)}\Big)_{i=1}^{n}. \tag{4.9}$$

Assumption (3.2b) follows automatically. Now, note that the k, j element of $\mathsf{V}^H\mathsf{V}$ is

$$\sum_{i=1}^{n} \mathrm{e}^{2\pi n\sqrt{-1}\phi(i-1)[\phi(j-1)-\phi(k-1)]}.$$

Noting that the sequence $\{\phi(i-1)\}_{i=1}^{n}$ is a re-ordering of $0, \ldots, 1-1/n$ for n a power of 2, this sum may be re-written by replacing $\phi(i-1)$ by $(i-1)/n$:

$$\sum_{i=1}^{n} \mathrm{e}^{2\pi\sqrt{-1}(i-1)[\phi(j-1)-\phi(k-1)]}.$$

Since $\phi(j-1) - \phi(k-1)$ is some integer multiple of $1/n$, it follows that this sum is $n\delta_{j,k}$, where δ is the Kronecker delta function. This establishes that $\mathsf{V}^H = n\mathsf{V}^{-1}$ as in (3.1).

Next, let $\omega_{k,\ell}$ denote the k, ℓ element of $\mathsf{V}^H\mathsf{C}\mathsf{V}$, which is given by the double sum

$$\omega_{k,\ell} = \sum_{i,j=1}^{n} K(\boldsymbol{h}(\phi(i-1) - \phi(j-1)) \bmod \mathbf{1}) \times \mathrm{e}^{-2\pi n\sqrt{-1}\phi(k-1)\phi(i-1)}\mathrm{e}^{2\pi n\sqrt{-1}\phi(j-1)\phi(l-1)}$$

Noting that the sequence $\{\phi(i-1)\}_{i=1}^{n}$ is a re-ordering of $0,\ldots,1-1/n$ for n a power of 2, this sum may be re-written by replacing $\phi(i-1)$ by $(i-1)/n$ and $\phi(j-1)$ by $(j-1)/n$:

$$\omega_{k,\ell} = \sum_{i,j=1}^{n} K\left(\boldsymbol{h}\left(\frac{i-j}{n}\right) \bmod \mathbf{1}\right) \times \mathrm{e}^{-2\pi\sqrt{-1}\phi(k-1)(i-1)}\mathrm{e}^{2\pi\sqrt{-1}(j-1)\phi(\ell-1)}.$$

This sum also remains unchanged if i is replaced by $i+m$ and j is replaced by $j+m$ for any integer m:

$$\begin{aligned}\omega_{k,\ell} &= \sum_{i,j=1}^{n} K\left(\boldsymbol{h}\left(\frac{i-j}{n}\right) \bmod \mathbf{1}\right) \times \mathrm{e}^{-2\pi\sqrt{-1}\phi(k-1)(i+m-1)}\mathrm{e}^{2\pi\sqrt{-1}(j+m-1)\phi(\ell-1)} \\ &= \omega_{k,\ell}\mathrm{e}^{2\pi\sqrt{-1}m(\phi(\ell-1)-\phi(k-1))}.\end{aligned}$$

For this last equality to hold for all integers m, we must have $k=\ell$ or $\omega_{k,\ell}=0$. Thus,

$$\begin{aligned}\omega_{k,\ell} &= \delta_{k,\ell}\sum_{i,j=1}^{n} K\left(\boldsymbol{h}\left(\frac{i-j}{n}\right) \bmod \mathbf{1}\right) \times \mathrm{e}^{-2\pi\sqrt{-1}(i-j)\phi(k-1)} \\ &= n\delta_{k,\ell}\sum_{i=1}^{n} K\left(\left(\frac{i\boldsymbol{h}}{n}\right) \bmod \mathbf{1}\right) \mathrm{e}^{-2\pi\sqrt{-1}i\phi(k-1)}.\end{aligned}$$

This establishes $\mathsf{V}^H\mathsf{C}\mathsf{V}$ as a diagonal matrix whose diagonal elements are n times the eigenvalues, i.e., $\lambda_k = \omega_{k,k}/n$. Furthermore, V is the matrix of eigenvectors, which satisfies assumption (3.2a).

One can interpret the sequence reordering from $\{\phi(i-1)\}_{i=1}^{n}$ to $(0,\ldots,1-1/n)$, for n a power of 2, as a permutation, consequently as a permutation matrix acting upon C in 4.7. With out the sequence reordering, the matrix $\Big(K(\boldsymbol{h}((i-1)-(j-1)) \bmod \mathbf{1})\Big)_{i,j=1}^{n}$ is nothing but a circulant matrix. Thus it can be shown that the Gram matrix C is the permutation of a circulant matrix. By the properties of ϕ in (4.3), it follows that

$$\mathsf{P} = \left(\delta_{n\phi(i-1),j-1}\right)_{i,j=1}^{n} \tag{4.10}$$

is a permutation matrix, where $\delta_{\cdot,\cdot}$ is the Kronecker delta function. Then,

$$\mathsf{C}_{\boldsymbol{\theta}} = \left(C_{\boldsymbol{\theta}}(\boldsymbol{x}_i,\boldsymbol{x}_j)\right)_{i,j=1}^{n}$$

$$= \Big(K_{\boldsymbol{\theta}}\big(\boldsymbol{h}(\phi(i-1)-\phi(j-1)) \bmod \mathbf{1}\big)\Big)_{i,j=1}^{n} \qquad \text{by (4.1) and (4.4)}$$

$$= \Bigg(\sum_{i',j'=1}^{n} \delta_{n\phi(i-1),i'-1}\, K_{\boldsymbol{\theta}}\big(\boldsymbol{h}(i'-j')/n \bmod \mathbf{1}\big)\delta_{j'-1,n\phi(j-1)}\Bigg)_{i,j=1}^{n}$$

$$= \mathsf{P}\mathsf{K}_{\boldsymbol{\theta}}\mathsf{P}^{T}, \tag{4.11}$$

where

$$\mathsf{K}_{\boldsymbol{\theta}} = \big(K_{\boldsymbol{\theta}}\big(\boldsymbol{h}(i-j)/n \bmod \mathbf{1}\big)\big)_{i,j=1}^{n}. \tag{4.12}$$

Because $\mathsf{K}_{\boldsymbol{\theta}}$ is circulant, we know the form of it's eigenvector-eigenvalue decomposition:

$$\mathsf{K}_{\boldsymbol{\theta}} = \frac{1}{n}\mathsf{W}\mathsf{\Lambda}_{\boldsymbol{\theta}}\mathsf{W}^{H}, \;\; \mathsf{W} = \Big(\mathrm{e}^{2\pi\sqrt{-1}(i-1)(j-1)/n}\Big)_{i,j=1}^{n}. \tag{4.13}$$

By (4.11) we then have the eigenvector-eigenvalue decomposition for $\mathsf{C}_{\boldsymbol{\theta}}$ assumed in (3.1), namely

$$\mathsf{C}_{\boldsymbol{\theta}} = \frac{1}{n}\mathsf{V}\mathsf{\Lambda}_{\boldsymbol{\theta}}\mathsf{V}^{H}, \qquad \mathsf{V} = \mathsf{P}\mathsf{W}, \tag{4.14}$$

where the eigenvalues of $\mathsf{C}_{\boldsymbol{\theta}}$ and $\mathsf{K}_{\boldsymbol{\theta}}$ are identical. This can be performed in $\mathcal{O}(n\log(n))$ operations by the FFT.

4.2.2 Iterative Computation of the Fast Transform. Assumption (3.2a) is that computing $\mathsf{V}^{H}\boldsymbol{b}$ requires only $\mathcal{O}(n\log n)$ operations. Recall that we assume that n is a power of 2. This can be accomplished by an iterative algorithm. Let $\mathsf{V}^{(n)}$ denote the $n\times n$ matrix V defined in (4.9). We show how to compute $\mathsf{V}^{(2n)H}\boldsymbol{b}$ quickly for all $\boldsymbol{b}\in\mathbb{R}^{2n}$ assuming that $\mathsf{V}^{(n)H}\boldsymbol{b}$ can be computed quickly for all $\boldsymbol{b}\in\mathbb{R}^{n}$.

From the definition of the van der Corput sequence in (4.2) it follows that

$$\phi(2i) = \phi(i)/2,\; \phi(2i+1) = [\phi(i)+1]/2, \quad i\in\mathbb{N}_0 \tag{4.15}$$

$$\phi(i+n) = \phi(i) + 1/(2n), \qquad i = 0,\ldots,n-1, \tag{4.16}$$

$$n\phi(i)\in\mathbb{N}_0, \qquad i = 0,\ldots,n-1, \tag{4.17}$$

still assuming that n is an integer power of two. Let $\widetilde{\boldsymbol{b}} = \mathsf{V}^{(2n)H}\boldsymbol{b}$ for some arbitrary $\boldsymbol{b} \in \mathbb{R}^{2n}$, and define

$$\boldsymbol{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_{2n} \end{pmatrix}, \quad \boldsymbol{b}^{(1)} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}, \quad \boldsymbol{b}^{(2)} = \begin{pmatrix} b_{n+1} \\ \vdots \\ b_{2n} \end{pmatrix},$$

$$\widetilde{\boldsymbol{b}} = \begin{pmatrix} \widetilde{b}_1 \\ \vdots \\ \widetilde{b}_{2n} \end{pmatrix}, \quad \widetilde{\boldsymbol{b}}^{(1)} = \begin{pmatrix} \widetilde{\boldsymbol{b}}_1 \\ \widetilde{\boldsymbol{b}}_3 \\ \vdots \\ \widetilde{\boldsymbol{b}}_{2n-1} \end{pmatrix}, \quad \widetilde{\boldsymbol{b}}^{(2)} = \begin{pmatrix} \widetilde{\boldsymbol{b}}_2 \\ \widetilde{\boldsymbol{b}}_4 \\ \vdots \\ \widetilde{\boldsymbol{b}}_{2n} \end{pmatrix}.$$

It follows from these definitions and the definition of V in (4.9) that

$$\begin{aligned}
\widetilde{\boldsymbol{b}}^{(1)} &= \left(\sum_{j=1}^{2n} \mathrm{e}^{-4\pi n\sqrt{-1}\phi(2i-2)\phi(j-1)} b_j \right)_{i=1}^{n} \\
&= \left(\sum_{j=1}^{2n} \mathrm{e}^{-2\pi n\sqrt{-1}\phi(i-1)\phi(j-1)} b_j \right)_{i=1}^{n} \qquad \text{by (4.15)} \\
&= \left(\sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}\phi(i-1)\phi(j-1)} b_j \right)_{i=1}^{n} + \left(\sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}\phi(i-1)\phi(n+j-1)} b_{n+j} \right)_{i=1}^{n} \\
&= \mathsf{V}^{(n)H}\boldsymbol{b}^{(1)} + \left(\mathrm{e}^{-\pi\sqrt{-1}\phi(i-1)} s \sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}\phi(i-1)\phi(j-1)} b_{n+j} \right)_{i=1}^{n} \qquad \text{by (4.16)} \\
&= \mathsf{V}^{(n)H}\boldsymbol{b}^{(1)} + \left(\mathrm{e}^{-\pi\sqrt{-1}\phi(i-1)} \right)_{i=1}^{n} \odot \left(\mathsf{V}^{(n)H}\boldsymbol{b}^{(2)} \right),
\end{aligned}$$

where $\odot$ denotes the Hadamard (term-by-term) product. By a similar argument,

$$\begin{aligned}
\widetilde{\boldsymbol{b}}^{(2)} &= \left(\sum_{j=1}^{2n} \mathrm{e}^{-4\pi n\sqrt{-1}\phi(2i-1)\phi(j-1)} b_j \right)_{i=1}^{n} \\
&= \left(\sum_{j=1}^{2n} \mathrm{e}^{-2\pi n\sqrt{-1}[\phi(i-1)+1]\phi(j-1)} b_j \right)_{i=1}^{n} \qquad \text{by (4.15)} \\
&= \left(\sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}[\phi(i-1)+1]\phi(j-1)} b_j \right)_{i=1}^{n} + \left(\sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}[\phi(i-1)+1]\phi(n+j-1)} b_{n+j} \right)_{i=1}^{n}
\end{aligned}$$

$$= \mathsf{V}^{(n)H}\boldsymbol{b}^{(1)} + \left(\mathrm{e}^{-\pi\sqrt{-1}[\phi(i-1)+1]} \sum_{j=1}^{n} \mathrm{e}^{-2\pi n\sqrt{-1}\phi(i-1)\phi(j-1)} b_{n+j} \right)_{i=1}^{n}$$

$$\text{by (4.16) and (4.17)}$$

$$= \mathsf{V}^{(n)H}\boldsymbol{b}^{(1)} - \left(\mathrm{e}^{-\pi\sqrt{-1}\phi(i-1)} \right)_{i=1}^{n} \odot \left(\mathsf{V}^{(n)H}\boldsymbol{b}^{(2)} \right).$$

The computational cost to compute $\mathsf{V}^{(2n)H}\boldsymbol{b}$ is then twice the cost of computing $\mathsf{V}^{(n)H}\boldsymbol{b}^{(1)}$ plus $2n$ multiplications plus $2n$ additions/subtractions. An inductive argument shows that $\mathsf{V}^{(n)H}\boldsymbol{b}$ requires only $\mathcal{O}(n \log n)$ operations.

4.3 Continuous Valued Kernel Order

JR: Need better and more convincing motivation

We assumed in previous sections, the shift-invariant kernel's order is an even valued integer and also fixed. It requires the practitioner to be aware of the integrand's smoothness to precisely hand pick the kernel order to match the integrand's smoothness. However, It is not possible to know the the integrand's smoothness in most of the practical applications. This constraint limits the ability to continuously vary the kernel's smoothness to match the integrand like the shape parameter is chosen to match.

Integer kernel order is not suitable to optimally search by an optimization algorithm. As a consequence, one usually end up choosing a higher kernel order when the integrand is not smooth or lower kernel order when the integrand is very smooth. Often it leads to longer computation time or poor accuracy in the numerical integration. Here we explore an alternative form of the kernel which allows the kernel order to be a positive continuous greater than one. Let us recollect the infinite series expression that was used to construct the kernel(4.5),

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) := \sum_{\boldsymbol{k} \in \mathbb{Z}^d} \alpha_{\boldsymbol{k},\boldsymbol{\theta}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{x}} e^{-2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{t}}, \quad \alpha_{\boldsymbol{k},\boldsymbol{\theta}} = \prod_{l=1}^{d} \frac{1}{\max(\frac{|k_l|}{\eta_l}, 1)^r_{\eta_l \leq 1}}$$

where $\theta = (\eta, r)$. This form is convenient for analytical derivations. To make the derivations easier to follow, let us fix the dimension $d = 1$,

$$C_{\boldsymbol{\theta}}(x,t) = 1 + \eta \sum_{k\in\mathbb{Z},k\neq 0} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}kx} e^{-2\pi\sqrt{-1}kt}$$

4.3.1 Exponentially decaying kernel. We propose the following alternative form of the kernel. This kernel can also provide exponential decay,

$$C_{\boldsymbol{\theta}}(x,t) = 1 + \eta \sum_{k\in\mathbb{Z},k\neq 0} q^{|k|} e^{2\pi\sqrt{-1}k(x-t)}, \quad \text{with} \quad 0 < q < 1$$

where q is the kernel order. This can be rewritten as

$$\begin{aligned}
C_{\boldsymbol{\theta}}(x,t) =& 1 + \eta \sum_{k\in\mathbb{Z},k\neq 0} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} \\
=& 1 + \eta \left(\sum_{k=1}^{\infty} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} + \sum_{k=\infty}^{-1} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} \right), \\
=& 1 + \eta \left(\sum_{k=1}^{\infty} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} + \sum_{k=-\infty}^{-1} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} \right), \\
=& 1 + \eta \left(\underbrace{\sum_{k=1}^{\infty} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)}}_{*} + \sum_{k=1}^{\infty} e^{-2\pi\sqrt{-1}k(x-t)+|k|\log(q)} \right),
\end{aligned}$$

Let us focus on the term $(*)$,

$$\begin{aligned}
(*) &= \sum_{k=1}^{\infty} e^{2\pi\sqrt{-1}k(x-t)+|k|\log(q)} = \sum_{k=1}^{\infty} \left[e^{2\pi\sqrt{-1}(x-t)+\log(q)} \right]^k \\
&= \frac{e^{2\pi\sqrt{-1}(x-t)+\log(q)}}{1 - e^{2\pi\sqrt{-1}(x-t)+\log(q)}} = \frac{1}{e^{-2\pi\sqrt{-1}(x-t)-\log(q)} - 1} \\
&= \frac{1}{q^{-1}e^{-2\pi\sqrt{-1}(x-t)} - 1}
\end{aligned}$$

Using this result

$$\begin{aligned}
C_{\boldsymbol{\theta}}(x,t) &= 1 + \eta \left(\frac{1}{q^{-1}e^{-2\pi\sqrt{-1}(x-t)} - 1} + \frac{1}{q^{-1}e^{2\pi\sqrt{-1}(x-t)} - 1} \right), \\
&= 1 + \eta \left(\frac{q^{-1}\left(e^{2\pi\sqrt{-1}(x-t)} + e^{-2\pi\sqrt{-1}(x-t)} \right) - 2}{q^{-2} - q^{-1}\left(e^{2\pi\sqrt{-1}(x-t)} + e^{-2\pi\sqrt{-1}(x-t)} \right) + 1} \right),
\end{aligned}$$

$$= 1 + \eta \left(\frac{2q^{-1}\cos(2\pi\sqrt{-1}(x-t)) - 2}{q^{-2} - 2q^{-1}\cos(2\pi\sqrt{-1}(x-t)) + 1} \right),$$
$$= 1 + 2\eta q \left(\frac{\cos(2\pi\sqrt{-1}(x-t)) - q}{q^2 - 2q\cos(2\pi\sqrt{-1}(x-t)) + 1} \right),$$

Using the fact $\cos^2(t) + \sin^2(t) = 1$

$$C_{\boldsymbol{\theta}}(x,t) = 1 + 2\eta q \left(\frac{\cos(2\pi\sqrt{-1}(x-t)) - q}{\left[\cos(2\pi\sqrt{-1}(x-t)) - q\right]^2 + \sin^2(2\pi\sqrt{-1}(x-t))} \right),$$

which shows that the kernel order q can be continuously varied while searching for the optimal value. The hyperparameters need to be $\eta > 0$ and $0 < q < 1$ while searching for the optimum value, so we use the transformations demonstrated in Section 6.2 to map the values to or from $\mathbb{R}$, where the search is usually done. One disadvantage of this kernel is, it is very sensitive to the changes in kernel order $q \in (0,1)$, to even small vales. So the hyperparameter search sometimes misses the global minima.

4.3.2 Truncated series kernel. There is an another form to the previous kernel, which has the kernel order in $(1,\infty)$, making it more robust in the hyperparameter search. If we use the original definition of the kernel, we could make the kernel order r explicit such that it does not have to be an even integer, which was the constraint previously. For $d = 1$,

$$C_{\boldsymbol{\theta}}(x,t) = 1 + \eta \sum_{k\in\mathbb{Z}, k\neq 0} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}k(x-t)}$$

where $\theta = (\eta, r)$. But it has the infinite sum which cannot be used directly, so we truncate to length N,

$$C_{\boldsymbol{\theta},N}(x,t) = 1 + \eta \sum_{k=-N/2}^{N/2-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}k(x-t)}$$

The Gram matrix is written as

$$\mathsf{C}_{\boldsymbol{\theta},N} = \Big(C_{\boldsymbol{\theta},N}(\boldsymbol{x}_i, \boldsymbol{x}_j) \Big)_{i,j=1}^{n}.$$

The first column of the Gram matrix is

$$\begin{aligned}\boldsymbol{C}_{\boldsymbol{\theta},N} &= \left(C_{\boldsymbol{\theta},N}(\boldsymbol{x}_i,\boldsymbol{x}_1)\right)_{i=1}^{n} \\ &= \left(\prod_{l=1}^{d}\left[1+\eta_l \sum_{k=-N/2,k\neq 0}^{N/2-1} \frac{1}{|k_l|^r} e^{2\pi\sqrt{-1}k_l(x_{il}-x_{1l})}\right]\right)_{i=1}^{n},\end{aligned}$$

where d is number of dimensions and n is the number of points. But the direct computation involves nN computations, or n^2 computations if $N \approx n$. We can reduce the computations to $\mathcal{O}(n \log n)$ using the FFT. Let us define

$$\mathfrak{C}_r(t) := \sum_{k=-N/2,k\neq 0}^{N/2-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}k\,t}.$$

Using the notation $\mathfrak{C}_r$, rewrite

$$\boldsymbol{C}_{\boldsymbol{\theta},N} = \left(\prod_{l=1}^{d}\left[1+\eta\mathfrak{C}_r(|x_{il}-x_{1l}|)\right]\right)_{i=1}^{n}.$$

By the definition of lattice points from (4.1), we can observe $x_{il}-x_{1l} \in \{0, \frac{1}{n}, \frac{2}{n}, \ldots \frac{n-1}{n}\}$. This can be used to rewrite $\mathfrak{C}_r$ in a much simpler form,

$$\mathfrak{C}_r\left(\frac{j}{N}\right) = \sum_{k=-N/2,k\neq 0}^{N/2-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}k(\frac{j}{N})}, \quad \text{where } j = 0,1,\ldots N-1.$$

These notations were introduced to help us to show that $\widetilde{\mathfrak{C}}_r$, the discrete Fourier transform of $\mathfrak{C}_r$ can be computed analytically,

$$\begin{aligned}\widetilde{\mathfrak{C}}_r(m) &= \sum_{j=0}^{N-1} \mathfrak{C}_r(j/n) e^{-2\pi\sqrt{-1}jm/N} \\ &= \sum_{k=-N/2,k\neq 0}^{N/2-1} \sum_{j=0}^{N-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}kj/N} e^{-2\pi\sqrt{-1}jm/N} \\ &= \sum_{k=-N/2,k\neq 0}^{N/2-1} \sum_{j=0}^{N-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}(k-m)j/N}, \quad \text{by 4.19} \\ &= \sum_{k=-N/2,k\neq 0}^{N/2-1} \frac{N}{|k|^r}\ \delta_{k-m \bmod N,0}\ .\end{aligned}$$

By simply observing the above result, it is evident one can analytically compute $\widetilde{\mathfrak{C}}_r$,

$$\widetilde{\mathfrak{C}}_r(m) = \begin{cases} 0, & \text{for} \quad m = 0, \\ \frac{N}{|m|^r}, & \text{for} \quad m = 1, \dots, N/2 - 1, \\ \frac{N}{|N-m|^r}, & \text{for} \quad m = N/2, \dots, N-1 \end{cases} \tag{4.18}$$

where we used the fact,

$$\sum_{i=0}^{N-1} e^{2\pi\sqrt{-1}ij/N} = \begin{cases} \frac{1-e^{2\pi\sqrt{-1}jN/N}}{1-e^{2\pi\sqrt{-1}j/N}} = 0, & j \neq 0 \bmod N \\ N, & j = 0 \bmod N \end{cases}. \tag{4.19}$$

Having these result, we can easily back-compute $\mathfrak{C}$ using inverse discrete Fourier transform. It can be easily shown that inverse DFT of $\widetilde{\mathfrak{C}}_r$ returns $\mathfrak{C}$ indeed,

$$\begin{aligned} &\frac{1}{N}\sum_{m=0}^{N-1} \widetilde{\mathfrak{C}}_r(m) e^{2\pi\sqrt{-1}lm/N} \\ &= \frac{1}{N}\sum_{m=0}^{N-1}\sum_{j=0}^{N-1} \mathfrak{C}_r(j/n) e^{-2\pi\sqrt{-1}jm/N} e^{2\pi\sqrt{-1}lm/N}, \quad \text{by 4.19} \\ &= \frac{1}{N}\sum_{j=0}^{N-1} \mathfrak{C}_r(j/n) \sum_{m=0}^{N-1} e^{2\pi\sqrt{-1}(l-j)m/N} \\ &= \frac{1}{N}\sum_{j=0}^{N-1} \mathfrak{C}_r(j/n) N \delta_{(l-j) \bmod N, 0} \\ &= \mathfrak{C}_r(l/n), \quad \text{for} \quad l = 0, \dots, N-1 \end{aligned}$$

This shows that to compute n values of $\left(C_{\boldsymbol{\theta},N}(\boldsymbol{x}_i, \boldsymbol{x}_1)\right)_{i=1}^{n}$, setting $N = n$ is very convenient. The above results are summarized as an algorithm to compute $\mathfrak{C}$ using FFT:

1. Analytically compute $\left(\widetilde{\mathfrak{C}}_r(m)\right)_{m=0}^{N-1}$, the discrete Fourier coefficients of $(\mathfrak{C}_r(j/N))_{j=0}^{N-1}$ using (4.18)

2. Take inverse FFT of $\widetilde{\mathfrak{C}}_r$ to get $\mathfrak{C}$

where the computational cost is $\mathcal{O}(n \log n)$ instead of $\mathcal{O}(n^2)$. Another major advantage is, the FFT approach is numerically stable than the direct sum approach.

4.4 Summary

We summarize the results of this Chapter and the previous one as a theorem below.

Theorem 4.4.1. *Let* $\mathsf{C}_{\boldsymbol{\theta}}$ *be any symmetric, positive definite, shift-invariant covariance kernel of the form* (4.4)*, where* $K_{\boldsymbol{\theta}}$ *has period one in every variable. Furthermore, let* $K_{\boldsymbol{\theta}}$ *be scaled to satisfy* (3.4)*. When matched with rank-1 lattice data-sites,* $\mathsf{C}_{\boldsymbol{\theta}}$ *must satisfy assumptions* (3.2)*. The cubature,* $\widehat{\mu}$*, is just the sample mean. The* fast Fourier transform *(FFT) can be used to expedite the estimates of* $\boldsymbol{\theta}$ *in* (6.1) *and the credible interval widths* (6.2) *in* $\mathcal{O}(n \log n)$ *operations.*

Although the third part of the computational cost has the largest dependence on n, in practice it need not be the largest contributor to the computational cost. If function values are the result of an expensive simulation, then the first part may consume most of the computation time.

We have implemented the fast adaptive Bayesian cubature algorithm in MATLAB as part of the Guaranteed Adaptive Integration Library (GAIL) [28] as `cubBayesLattice_g`. This algorithm uses the kernel defined in (4.5) with $r = 1, 2$ and the periodizing variable transforms in Section 4.5. The rank-1 lattice node generator is taken from [29] (`exod2_base2_m20`).

4.5 Periodizing Variable Transformations

The shift-invariant covariance kernels underlying our `cubBayesLattice_g` Bayesian cubature assume that the integrand has a degree of periodicity, with the smoothness assumed depending on the smoothness of the kernel. In other-words, non-

periodic functions do not live in the space spanned by the shift-invariant covariance kernels. While integrands arising in practice may be smooth, they might not be periodic. Variable transformation or periodization transform techniques are typically used to enforce the periodicity in multi-dimensional numerical integrations where boundary conditions needs to be enforced. These transformations could be either polynomial, exponential and also trigonometric in nature. Some of the most popular transformation are listed here for reference. Suppose that the original integral has been expressed as

$$\mu := \int_{[0,1]^d} g(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t}$$

where g has sufficient smoothness, but lacks periodicity. The goal is to transform the integral above to the form of (1.1), where the integrand f—and perhaps its derivatives—are periodic.

The Baker's transform, also called tent transform,

$$\boldsymbol{\Psi} : \boldsymbol{x} \mapsto (\Psi(x_1), \ldots, \Psi(x_d)), \quad \Psi(x) = 1 - 2\,|x - 1/2|\,, \tag{4.20}$$

allows us to write μ in the form of (1.1), where $f(\boldsymbol{x}) = g(\boldsymbol{\Psi}(\boldsymbol{x}))$. Since $\Psi'(x)$ is not continuous, f does not have continuous derivatives.

A family of smoother variable transforms, i.e., with derivatives take the form

$$\boldsymbol{\Psi} : \boldsymbol{x} \mapsto (\Psi(x_1), \ldots, \Psi(x_d)), \quad \Psi : [0,1] \mapsto [0,1],$$

which allows us to write μ in the form of (1.1) with

$$f(\boldsymbol{x}) = g(\boldsymbol{\Psi}(\boldsymbol{x})) \prod_{\ell=1}^{d} \Psi'(x_l).$$

For $r \in \mathbb{N}_0$, if the following hold:

- $\Psi \in C^{r+1}[0,1]$,
- $\lim_{x \downarrow 0} x^{-r-1}\Psi'(x) = \lim_{x \uparrow 1} (1-x)^{-r-1}\Psi'(x) = 0$, and

- $g \in C^{(r,\ldots,r)}[0,1]^d$,

then f has continuous, periodic mixed partial derivatives of up to order r in each direction. Examples of this kind of transform include [30]:

$$
\begin{aligned}
C^0 : \Psi(x) &= 3x^2 - 2x^3, \quad \Psi'(x) = 6x(1-x), \\
C^1 : \Psi(x) &= x^3(10 - 15x + 6x^2), \\
\Psi'(x) &= 30x^2(1-x)^2 \\
\text{Sidi's } C^1 : \Psi(x) &= x - \frac{\sin(2\pi x)}{2\pi}, \\
\Psi'(x) &= 1 - \cos(2\pi x), \\
\text{Sidi's } C^2 : \Psi(x) &= \frac{8 - 9\cos(\pi x) + \cos(3\pi x)}{16}, \\
\Psi'(x) &= \frac{3\pi[3\sin(\pi x) - \sin(3\pi x)]}{16}.
\end{aligned}
$$

These transforms vary in terms of computational complexity and accuracy and shall be chosen to match the covariance kernel and integrand. Choosing an optimal periodizing is a topic of future research. Baker's transform is the least complex of all which is a tent map in each coordinate. It preserves only continuity but it is easier to compute and it does not include product term up to the length dimension of the integrand, making it more numerically stable. C^0 is a polynomial transformation only and ensures periodicity of function. C^1 is a polynomial transformation and preserving the first derivative. Sidi's C^1, a transform which uses trigonometric Sine, preserves the first derivative and is, in general, a better option than C^1. Sidi's C^2, also a transform which uses trigonometric Sine, preserves up to second derivative. We use this when smoothness of Sidi's C^1 is not sufficient and need to preserve up to second derivative.

Periodizing variable transforms are used in the numerical examples in Section 7. In some cases, they can speed the convergence of the Bayesian cubature

because they allow one to take advantage of smoother covariance kernels. However, there is a trade-off. Smoother periodizing transformations tend to give integrands f with larger inferred s values and thus wider credible intervals.

CHAPTER 5

SOBOL' NETS AND WALSH KERNELS

5.1 Sobol' Nets

The previous section showed an automatic Bayesian cubature algorithm using rank-1 lattice nodes and shift-invariant kernels. In this chapter, we demonstrate a second approach to formulate fast transform using matching kernel and point sets. Scrambled Sobol' nets and Walsh kernels are paired to achieve $\mathcal{O}(N^{-1+\epsilon})$ order error convergence. Sobol' nets [31] are low discrepancy quasi-random points, used extensively in numerical integration, simulation, and optimization.

Nets were developed to provide deterministic sample points for quasi-Monte Carlo rules [32]. The (t, m, d)-nets in base b introduced by Niederreiter are such point sets consisting of b^m points in $[0, 1)^d$, whose quality is governed by t, in particular, lower values of t correspond to (t, m, d)-nets of higher quality [33]. Digital (t, m, d)-nets are a special case of (t, m, d)-nets, constructed using matrix-vector multiplications over finite fields.

A (t, m, d)-net in base b is a sequence of z_i of b^m points of $[0, 1]^d$ with the property that every elementary interval in base b of volume b^{t-m} contains precisely b^t points from x_i. Here $d \geq 1$, $b \geq 2$ and given two integers $0 \leq t \leq m$. Sobol' [34] nets are special case of (t, m, d)-nets when base $b = 2$. Digital sequences are infinite length digital nets.

Definition 1. *For any non-negative integer* $i = \dots i_3 i_2 i_1 (\text{base}\, b)$*, define the* $\infty \times 1$ *vector* $\vec{i}$ *as the vector of its digits, that is,* $\vec{i} = (i_1, i_2, \dots)^T$*. For any point* $z = 0.z_1 z_2 \dots (\text{base}\, b) \in [0, 1)$*, let* $\vec{z} = (z_1, z_2, \dots)^T$ *denote the* $\infty \times 1$ *vector of the digits of* z*. Let* $\mathsf{G}_1, \dots, \mathsf{G}_d$ *denote predetermined* $\infty \times \infty$ *generator matrices. The digital sequence in* base b *is* $\{\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, \dots\}$*, where each* $\boldsymbol{z}_i = (z_{i1}, \dots, z_{id})^T \in [0, 1)^d$ *is*

defined by

$$\vec{z}_{i\ell} = \mathsf{G}_\ell \, \vec{\imath}, \quad \ell = 1, \ldots, d, \quad i = 0, 1, \ldots \, .$$

Digital nets have a group structure under the digit-wise addition, which is a very useful property exploited in our algorithm, especially to develop fast Bayesian transform to speedup computations. Digitwise addition, $\oplus$, subtraction $\ominus$, are defined in terms of b-ary expansions of points in $[0,1)^d]$,

$$\boldsymbol{x} \oplus \boldsymbol{y} = \left(\sum_{l=1}^{\infty} [x_{jl} + y_{jl} \bmod b] b^{-l} \bmod 1 \right)_{j=1}^{d},$$

where

$$\boldsymbol{x} = \left(\sum_{l=1}^{\infty} x_{jl} b^{-l} \right)_{j=1}^{d}, \quad \boldsymbol{y} = \left(\sum_{l=1}^{\infty} y_{jl} b^{-l} \right)_{j=1}^{d}, \quad x_{jl}, y_{jl} \in \{0, \ldots, b-1\},$$

Let $\{\boldsymbol{x}_i\}_{i=0}^{b^m-1}$ be a digital net, then

$$\forall i_i, i_2 \in \{0, \ldots, 2^m - 1\}, \quad \boldsymbol{x}_{i_1} \oplus \boldsymbol{x}_{i_2} = \boldsymbol{x}_{i_3}, \quad \text{for some } i_3 \in \{0, \ldots, b^m - 1\}.$$

The following very useful result arises from the fundamental property of digital nets.

Lemma 5.1.1. *Let $\boldsymbol{x}_i, \boldsymbol{x}_j$ are digitally shifted digital nets and $\boldsymbol{z}_i, \boldsymbol{z}_j$ are the corresponding un-shifted digital nets, then,*

$$\boldsymbol{x}_i \ominus \boldsymbol{x}_j = \boldsymbol{z}_i \ominus \boldsymbol{z}_j = \boldsymbol{z}_{i \ominus j}, \quad \forall i, j \in \mathbb{N}_0, \tag{5.1}$$

i.e.,

$$\vec{x}_{il} = \vec{z}_{il} + \vec{\Delta}_l \bmod 1,$$

where $\vec{x}_{il}$ is the lth component of ith digital net and $\vec{\Delta}_l$ is the digital shift for lth component. Also note that the digital subtraction is symmetric,

$$\boldsymbol{x}_i \ominus \boldsymbol{x}_i = \boldsymbol{0}, \qquad \boldsymbol{x}_i \ominus \boldsymbol{x}_j = \boldsymbol{x}_j \ominus \boldsymbol{x}_i, \quad \forall i, j \in \mathbb{N}_0. \tag{5.2}$$

Proof. The proof of Lemma 5.1.1 can be obtained from the definition of digital nets which stated that the digital nets are obtained using generator matrices, $\vec{z}_{il} = \mathsf{G}_l \, \vec{\imath} \bmod 2$. Rewriting the subtraction using the generating matrix provides the result,

$$\begin{aligned}
\vec{z}_{il} - \vec{z}_{jl} \bmod b &= (\mathsf{G}_l \vec{\imath} \bmod b) - (\mathsf{G}_l \vec{\jmath} \bmod b) \\
&= (\mathsf{G}_l \vec{\imath} - \mathsf{G}_l \vec{\jmath}) \bmod b \\
&= \mathsf{G}_l (\vec{\imath} - \vec{\jmath}) \bmod b \\
&= \mathsf{G}_l (\overrightarrow{i \ominus j}) \bmod b \\
&= \vec{z}_{i \ominus j \, l}.
\end{aligned}$$

The rest of the lemma is obvious from the definition of digital nets. □

We use digitally shifted and scrambled nets in this research [35]. Digital shift helps to avoid having nodes at origin, similar to the random shift used with lattice nodes. Let $\{\boldsymbol{x}_0, \boldsymbol{x}_1, \boldsymbol{x}_2, ...\}$ denote the randomly scrambled version of the original sequence $\{\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2, \dots\}$. Let x_{ijk} denote the kth digit of the jth component of $\boldsymbol{x}_i$, and similarly for z_{ijk}. Then

$$x_{ij1} = \pi_j(z_{ij1}), \quad x_{ij2} = \pi_{z_{ij1}}(z_{ij2}), \quad x_{ij3} = \pi_{z_{ij1}, z_{ij2}}(z_{ij3}), \quad \dots,$$

$$x_{ijk} = \pi_{z_{ij1}, z_{ij2}, \dots, z_{ijk-1}}(z_{ijk}), \quad \dots,$$

where the $\pi_{a_1 a_2 \dots}$ are random permutations of the elements in $\{0, \dots, b-1\}$ chosen uniformly and mutually independently. A proof that a randomized net preserves the property of (t, m, d)-net almost surely can be found in Owen [36].

An example of 64 Sobol' points in $d = 2$ is given in Figure 4.1. The even coverage of the unit cube is ensured by a well chosen generating matrix. The choice of generating vector is typically done offline by computer search. See [37] and [38] for more on generating matrices. We use randomly scrambled Sobol' sequences in this research [39] which eliminates bias while retraining the low-discrepancy properties.

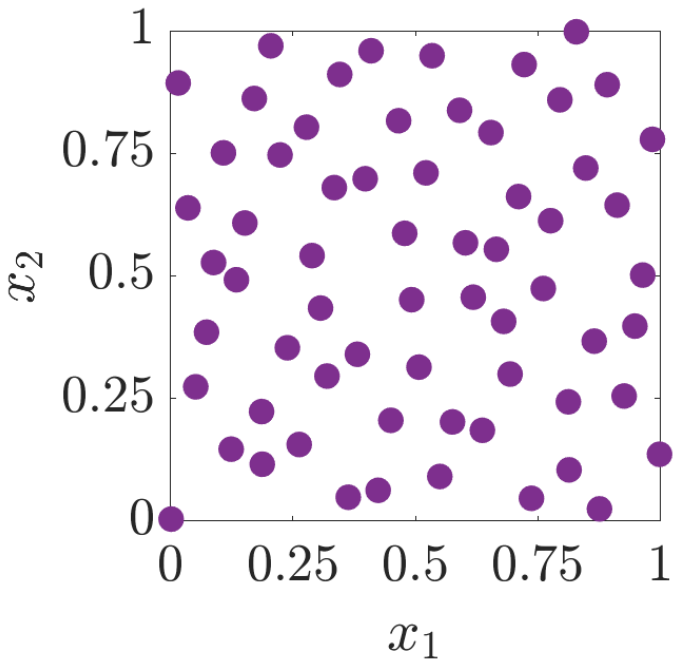


Figure 5.1. Example of a scrambled Sobol' node set in $d = 2$

5.2 Walsh Kernels

Walsh kernels are product kernels based on the Walsh functions. We introduce the necessary concepts in this section.

5.2.1 Walsh functions. Like Fourier transform used with lattice points (Section 4.2), Hadamard-Walsh transform used for the digital nets, which we will simply call Walsh transform. Walsh transform is defined using Walsh functions. Define $\mathbb{N} := \{1, 2, \dots\}$ and $\mathbb{N}_0 := \{0, 1, 2, \dots\}$. The one-dimensional Walsh functions in base b are defined as

$$\mathrm{wal}_{b,k}(x) := e^{2\pi\sqrt{-1}(x_1k_0+x_2k_1+\cdots+x_nk_{n-1})/b} = e^{2\pi i\vec{k}_n^T\vec{x}_n/b} \tag{5.3}$$

for $x \in [0, 1)$ and $k \in \mathbb{N}_0$ and the unique base b expansions $x = \sum_{i\geq 1} x_i b^{-i} = (0.x_1x_2\dots)_b$, and $k = \sum_{i\geq 0} k_i b^i = (k_n \dots k_0)_b$, $\vec{k} = (k_0, \dots k_n)^T$, with n at least as large as the number of digits to represent x or k. Multivariate Walsh functions are

defined as the product of the one-dimensional Walsh functions,

$$\mathrm{wal}_{b,\boldsymbol{k}}(\boldsymbol{x}) := \prod_{j=1}^{d} \mathrm{wal}_{b,k_j}(x_j)$$

As shown in (5.3), the Walsh functions only take the values in $\{1, -1\}$, i.e., $\mathrm{wal}_{b,k} : [0,1] \to \{-1,1\},\ k \in \mathbb{N}_0$. Walsh functions form an orthonormal basis of the Hilbert space $L^2[0,1]$,

$$\int_{[0,1]} \mathrm{wal}_{b,i}(\boldsymbol{x})\mathrm{wal}_{b,j}(\boldsymbol{x})\mathrm{d}x = \delta_{i,j}$$

Digital nets are designed to integrate the Walsh functions without error. Thus our Bayesian cubature algorithm could integrate linear combinations of these ideal integrands without error. Functions that are well approximated by such linear combinations are then integrated with small errors.

In this research we use Sobol' nodes which are digital nets with base $b = 2$. So here afterwards base $b = 2$ is assumed if not specified or the notation is dropped.

5.2.2 Walsh kernels. Consider the kernels of the form [40]

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) = K(\boldsymbol{x} \ominus \boldsymbol{t}) = 1 + \eta\omega_r(\boldsymbol{x} \ominus \boldsymbol{t}), \quad \boldsymbol{\theta} = (r, \eta) \tag{5.4}$$

where $\ominus$ is bitwise subtraction,

$$\omega_r(\boldsymbol{x}) = \prod_{l=1}^{d} \sum_{k=1}^{\infty} \frac{\mathrm{wal}_{b,k}(x_l)}{b^{2r\lfloor \log_b k \rfloor}}$$

where r is kernel order. Explicit expression available for ω_r [40] in case of $b = 2$, order $r = 1$,

$$\omega_1(\boldsymbol{x}) = \prod_{l=1}^{d} \sum_{k=1}^{\infty} \frac{\mathrm{wal}_{b,k}(x_l)}{b^{2\lfloor \log_b k \rfloor}} = 6^d \prod_{l=1}^{d} \left(\frac{1}{6} - 2^{\lfloor \log_2 x_l \rfloor - 1} \right). \tag{5.5}$$

The Figure 5.2 shows $r = 1$ order Walsh kernel in the interval $[0,1]$. Unlike the shift-invariant kernels used with lattice nodes, low order Walsh kernels are not

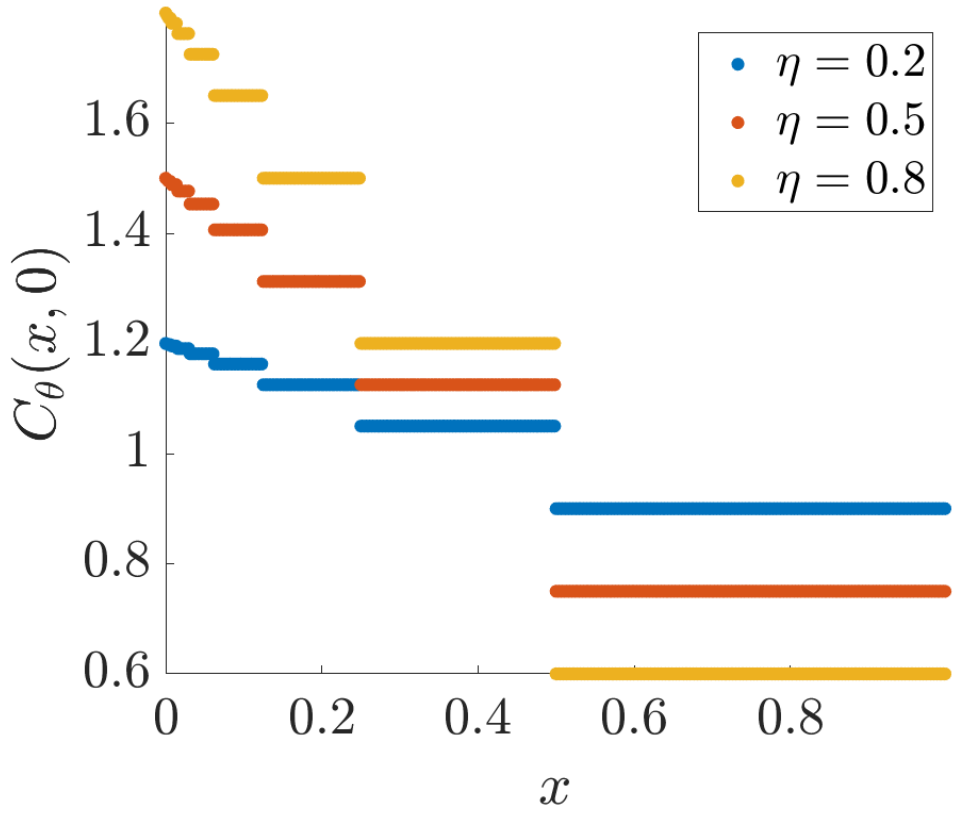


Figure 5.2. Walsh kernel of order $r = 1$ in dimension $d = 1$. This figure can be reproduced using `plot_walsh_kernel.m`.

very smooth and only piecewise constant. Smaller η implies lesser variation in the amplitude of the kernel. Also the Walsh kernels are digital shift invariant but not periodic.

5.2.3 Walsh transform. The Walsh-Hadamard transform (WHT) is a generalized class of discrete Fourier transform (DFT) that was used with lattice nodes, and is much simpler to compute. The Walsh-Hadamard transform matrices comprises of only $\{1, -1\}$ values, so the computation usually involves only additions and subtractions. Hence, the WHT is also sometimes called the integer transform. In comparison DFT uses complex exponential functions and the computation involves complex, non-integer multiplications.

The Walsh-Hadamard transforms are $2^m \times 2^m$ Walsh-Hadamard matrices, which are constructed recursively, starting with $\mathsf{H}^{(0)} = 1$,

$$\mathsf{H}^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

$$\mathsf{H}^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix},$$

$$\vdots$$

$$\mathsf{H}^{(m+1)} = \begin{pmatrix} \mathsf{H}^{(m)} & \mathsf{H}^{(m)} \\ \mathsf{H}^{(m)} & -\mathsf{H}^{(m)} \end{pmatrix} = \mathsf{H}^{(1)} \bigotimes \mathsf{H}^{(m)} \tag{5.6}$$

where $\bigotimes$ is Kronecker product. Alternatively for base $b = 2$, these matrices can be

directly obtained by,

$$\mathsf{H}^{(m)} = \left((-1)^{(\vec{i}_m^T \vec{j}_m)}\right)_{i,j=0}^{n-1},$$

where the notation $\vec{i}_m^T \vec{j}_m$ indicates bitwise dot product.

5.2.4 Eigenvectors of C are columns of Hadamard matrix. The Gram matrix C formed by Walsh kernels and Sobol' nodes have a special structure called block-Toeplitz matrix which can be used to construct the fast Bayesian transform. A Toeplitz matrix is a diagonal-constant matrix in which each descending diagonal from left to right is constant. A block Toeplitz matrix is a special block matrix, which contains blocks that are repeated down the diagonals of the matrix, as a Toeplitz matrix has elements repeated down the diagonal. We prove the eigenvectors of C are columns of Hadamard matrix in two theorems.

Theorem 5.2.1. *Let $(\boldsymbol{x}_i)_{i=0}^{n-1}$ be Sobol' nodes and, K, any positive definite kernel function such as* (5.5) *which matches Sobol' nodes. The Gram matrix,*

$$\mathsf{C} = \big(C(\boldsymbol{x}_i, \boldsymbol{x}_j)\big)_{i,j=0}^{n-1} = \big(K(\boldsymbol{x}_i \ominus \boldsymbol{x}_j)\big)_{i,j=0}^{n-1},$$
$$\text{where} \quad C(\boldsymbol{x}, \boldsymbol{t}) = K(\boldsymbol{x} \ominus \boldsymbol{t}) \quad \boldsymbol{x}, \boldsymbol{t} \in [0,1)^d,$$

is a block-Toeplitz matrix and all the sub-blocks and its sub-sub-blocks are also block-Toeplitz.

Proof. We prove this by induction. The relation between sub-block matrices can be deciphered using the properties of digital nets. To help with the proof of block-Toeplitz structure, consider the digital net properties 5.1 and 5.2. Let us introduce some notation,

$$\mathsf{C}^{(l)} := \left(K(z_{i \ominus j})\right)_{i,j=0}^{2^l-1},$$

$$\mathsf{C}^{(l,q)} := \left(K(z_{i\ominus j+q2^l}) \right)_{i,j=0}^{2^l-1}, \qquad \text{furthermore,} \quad \mathsf{C}^{(l)} = \mathsf{C}^{(l,0)}.$$

This notation is convenient for the proof as shown below.

As the first step, we verify the property holds for $l = 0$,

$$\mathsf{C}^{(0)} = \left(K(\boldsymbol{z}_{0\ominus 0}) \right) = \left(K(\boldsymbol{z}_0) \right), \quad \text{by 5.1,}$$

thus by definition $\mathsf{C}^{(0)}$ is a block-Toeplitz.

Please note if $\mathsf{C}^{(m)}$ is block-Toeplitz then $\mathsf{C}^{(m,1)}$ is also a block-Toeplitz. This is due to the fact that $i \oplus 2^m \ominus j = i \ominus j \ominus 2^m = i \ominus j \oplus 2^m$ for $i, j = 0, \ldots, 2^{m-1}$ since $i \ominus j < 2^{m-1}$. Assuming C^m is a block-Toeplitz, we need to prove C^{m+1} is also a block-Toeplitz. Let $n = 2^m$,

$$\mathsf{C}^{(m+1)} = \begin{pmatrix} K(\boldsymbol{z}_{0\ominus 0}) & \ldots & K(\boldsymbol{z}_{0\ominus n-1}) & K(\boldsymbol{z}_{0\ominus n}) & \ldots & K(\boldsymbol{z}_{0\ominus 2n-1}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{n-1\ominus 0}) & \ldots & K(\boldsymbol{z}_{n-1\ominus n-1}) & K(\boldsymbol{z}_{n-1\ominus n}) & \ldots & K(\boldsymbol{z}_{n-1\ominus 2n-1}) \\ K(\boldsymbol{z}_{n\ominus 0}) & \ldots & K(\boldsymbol{z}_{n\ominus n-1}) & K(\boldsymbol{z}_{n\ominus n}) & \ldots & K(\boldsymbol{z}_{n\ominus 2n-1}) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{2n-1\ominus 0}) & \ldots & K(\boldsymbol{z}_{2n-1\ominus n-1}) & K(\boldsymbol{z}_{2n-1\ominus n}) & \ldots & K(\boldsymbol{z}_{2n-1\ominus 2n-1}) \end{pmatrix}$$

$$= \begin{pmatrix} \begin{pmatrix} K(\boldsymbol{z}_0) & \ldots & K(\boldsymbol{z}_{n-1}) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{n-1}) & \ldots & K(\boldsymbol{z}_0) \end{pmatrix} & \begin{pmatrix} K(\boldsymbol{z}_n) & \ldots & K(\boldsymbol{z}_{2n-1}) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{2n-1}) & \ldots & K(\boldsymbol{z}_n) \end{pmatrix} \\ \begin{pmatrix} K(\boldsymbol{z}_n) & \ldots & K(\boldsymbol{z}_{2n-1}) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{2n-1}) & \ldots & K(\boldsymbol{z}_n) \end{pmatrix} & \begin{pmatrix} K(\boldsymbol{z}_0) & \ldots & K(\boldsymbol{z}_{n-1}) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{z}_{n-1}) & \ldots & K(\boldsymbol{z}_0) \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} \mathsf{C}^{(m)} & \mathsf{C}^{(m,1)} \\ \\ \mathsf{C}^{(m,1)} & \mathsf{C}^{(m)} \end{pmatrix}$$

is a block-Toeplitz, where we used the properties 5.1, 5.2 and facts $2n-1 \ominus n = n-1$, $2n-1 \ominus n-1 = n$, and $n \ominus n-1 = 2n-1$. Thus C of size 2^m, for $m \in \mathbb{N}$, is a block-Toeplitz and every block and it's sub-blocks of size 2^p, $p \in \mathbb{N}$, $p \leq m$ are also block-Toeplitz. □

Theorem 5.2.2. *The Hadamard matrix* H *diagonalizes* C *so the columns of Hadamard matrix are the eigenvectors of* C.

Proof. Again we use proof-by-induction technique to show that the Hadamard matrix diagonalizes C.

We can easily see the Hadamard matrix H^1 diagonalizes C^1,

$$\begin{aligned} \mathsf{H}^{(1)}\mathsf{C}^{(1)} &= \begin{pmatrix} 1 & 1 \\ \\ 1 & -1 \end{pmatrix} \begin{pmatrix} K(\boldsymbol{z}_0) & K(\boldsymbol{z}_1) \\ \\ K(\boldsymbol{z}_1) & K(\boldsymbol{z}_0) \end{pmatrix}, \\ &= \begin{pmatrix} K(\boldsymbol{z}_0) + K(\boldsymbol{z}_1) & 0 \\ \\ 0 & K(\boldsymbol{z}_0) - K(\boldsymbol{z}_1) \end{pmatrix} \begin{pmatrix} 1 & 1 \\ \\ 1 & -1 \end{pmatrix}, \\ &= \Lambda^{(1)}\mathsf{H}^{(1)}. \end{aligned}$$

Where $\Lambda^{(1)}$ is a diagonal matrix, thus $\mathsf{H}(1)$ diagonalizes $\mathsf{C}^{(1)}$.

Let us assume H^m diagonalizes C^m, so $\mathsf{H}^{(m)}\mathsf{C}^{(m)} = \Lambda^{(m)}\mathsf{H}^{(m)}$ where $\Lambda^{(m)}$ is diagonal, we need to prove H^{m+1} diagonalizes C^{m+1},

$$\mathsf{H}^{(m+1)}\mathsf{C}^{(m+1)} = \begin{pmatrix} \mathsf{H}^{(m)} & \mathsf{H}^{(m)} \\ \\ \mathsf{H}^{(m)} & -\mathsf{H}^{(m)} \end{pmatrix} \begin{pmatrix} \mathsf{C}^{(m)} & \mathsf{C}^{(m,m)} \\ \\ \mathsf{C}^{(m,m)} & \mathsf{C}^{(m)} \end{pmatrix},$$

$$= \begin{pmatrix} \mathsf{C}^{(m)} + \mathsf{C}^{(m,m)} & 0 \\ 0 & \mathsf{C}^{(m)} - \mathsf{C}^{(m,m)} \end{pmatrix} \begin{pmatrix} \mathsf{H}^{(m)} & \mathsf{H}^{(m)} \\ \mathsf{H}^{(m)} & -\mathsf{H}^{(m)} \end{pmatrix},$$

$$= \Lambda^{(m+1)} \mathsf{H}^{(m+1)}.$$

Thus $\mathsf{H}^{(m+1)}$ diagonalizes $\mathsf{C}^{(m+1)}$ and $\Lambda^{(m+1)}$ is diagonal. Here we used the fact that both H and C are symmetric positive definite. □

5.2.5 Fast transform. We can easily show that the Walsh-Hadamard matrices satisfy the assumptions of fast Bayesian transform (3.2) V^H. It is important to note that the columns/rows of Walsh-Hadamard matrices are mutually orthogonal. As shown in Section 5.2.4 the eigenvectors are,

$$\mathsf{V} = \mathsf{H}^{(m)} = \underbrace{\mathsf{H}^{(1)} \bigotimes \ldots \bigotimes \mathsf{H}^{(1)}}_{m \text{ times}}, \tag{5.7}$$

$$\text{where} \quad \mathsf{C}^{(m)} = \frac{1}{n} \mathsf{H}^{(m)} \Lambda^{(m)} \mathsf{H}^{(m)}.$$

Assumption (3.2b) follows automatically by the fact that Walsh-Hadamard matrices can be constructed analytically. Assumption (3.2a) can also be verified as the first row/column are one vectors. Finally, assumption (3.2c) is satisfied due to the fact that fast Walsh transform can be computed in $\mathcal{O}(n \log n)$ operations using fast Walsh-Hadamard transform. Thus the Walsh-Hadamard transform is a fast Bayesian transform as per the (3.2).

We have implemented a fast adaptive Bayesian cubature algorithm using the kernel (5.4) with $r = 1$ and Sobol' points [41] in MATLAB as part of the Guaranteed Adaptive Integration Library (GAIL) [28] as `cubBayesNet_g`. The Sobol' points used in this algorithm are generated using MATLAB's builtin function `sobolset` and scrambled using MATLAB function `scramble` [39]. The fast Walsh-Hadamard transform (5.7) is computed using MATLAB's builtin function `fwht` with *hadamard*

ordering.

5.2.6 Extensible Sobol' points. Similar to the extensible lattice points, Sobol' points could be extended to improve the integration accuracy till the required error tolerance is met. Extensible Sobol' nodes can be combined with Hadamard matrices as demonstrated here. Using the same notations as in Section 4.2.2, let Let $\widetilde{\boldsymbol{y}} = \mathsf{H}^{(m+1)}\boldsymbol{y}$ for some arbitrary $\boldsymbol{y} \in \mathbb{R}^{2n}$, $n = 2^m$. Define,

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_{2n} \end{pmatrix}, \quad \boldsymbol{y}^{(1)} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \quad \boldsymbol{y}^{(2)} = \begin{pmatrix} y_{n+1} \\ \vdots \\ y_{2n} \end{pmatrix},$$

$$\widetilde{\boldsymbol{y}}^{(1)} = \mathsf{H}^{(m)}\boldsymbol{y}^{(1)} = \begin{pmatrix} \widetilde{y}_1^{(1)} \\ \widetilde{y}_2^{(1)} \\ \vdots \\ \widetilde{y}_n^{(1)} \end{pmatrix}, \quad \widetilde{\boldsymbol{y}}^{(2)} = \mathsf{H}^{(m)}\boldsymbol{y}^{(2)} = \begin{pmatrix} \widetilde{y}_1^{(2)} \\ \widetilde{y}_2^{(2)} \\ \vdots \\ \widetilde{y}_n^{(2)} \end{pmatrix}.$$

Then,

$$\begin{aligned} \widetilde{\boldsymbol{y}} &= \mathsf{H}^{(m+1)}\boldsymbol{y} \\ &= \begin{pmatrix} \mathsf{H}^{(m)} & \mathsf{H}^{(m)} \\ \mathsf{H}^{(m)} & -\mathsf{H}^{(m)} \end{pmatrix} \begin{pmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \end{pmatrix}, \qquad \text{by (5.6)} \\ &= \begin{pmatrix} \mathsf{H}^{(m)}\boldsymbol{y}^{(1)} + \mathsf{H}^{(m)}\boldsymbol{y}^{(2)} \\ \mathsf{H}^{(m)}\boldsymbol{y}^{(1)} - \mathsf{H}^{(m)}\boldsymbol{y}^{(2)} \end{pmatrix}, \\ &= \begin{pmatrix} \widetilde{\boldsymbol{y}}^{(1)} + \widetilde{\boldsymbol{y}}^{(2)} \\ \widetilde{\boldsymbol{y}}^{(1)} - \widetilde{\boldsymbol{y}}^{(2)} \end{pmatrix} =: \widetilde{\boldsymbol{y}} \quad . \end{aligned}$$

As before with the lattice nodes, the computational cost to compute $\mathsf{V}^{(m+1)H}\boldsymbol{y}$ is then twice the cost of computing $\mathsf{V}^{(m)H}\boldsymbol{y}^{(1)}$ plus $2n$ additions, where $n = 2^m$. An inductive argument shows that for any $m \in \mathbb{N}$, $\mathsf{V}^{(m)H}\boldsymbol{y}$ requires only $\mathcal{O}(n \log n)$ operations. Usually the multiplications in $\mathsf{V}^{(m)H}\boldsymbol{y}^{(1)}$ are multiplications by -1 which are simply accomplished using sign change or negation, requiring no multiplications at all.

5.2.7 Higher Order Nets. Higher order digital nets are a modified (t, m, d)-nets, introduced in [42] which can be used to numerically integrate smoother functions which are not necessarily periodic, but have square integrable mixed partial derivatives of order α, at a rate of $\mathcal{O}(N^{-\alpha})$ multiplied by a power of a $\log N$ factor using rules corresponding to the modified (t, m, d)-nets. We want to emphasizes that quasi-Monte Carlo rules based on these point sets can achieve convergence rates faster than $\mathcal{O}(N^{-1})$.

Higher order digital nets are constructed using matrix-vector multiplications over finite fields. Bayesian cubatures using higher order digital nets are the topic for future research.

5.3 Summary

We summarize the results of this chapter as a theorem,

Theorem 5.3.1. *Any symmetric, positive definite, digital shift-invariant covariance kernel of the form* (5.4) *scaled to satisfy* (3.4)*, when matched with digital net data-sites, must satisfy assumptions* (3.2)*. The* fast Walsh-Hadamard transform *(FWHT) can be used to expedite the estimates of* $\boldsymbol{\theta}$ *in* (6.1) *and the credible interval widths* (6.2) *in* $\mathcal{O}(n \log n)$ *operations. The cubature,* $\widehat{\mu}$*, is just the sample mean.*

CHAPTER 6

NUMERICAL IMPLEMENTATION

6.1 Overcoming the Cancellation Error

For the covariance kernels used in our computation, it may happen that n/λ_1 is close to 1. Thus, the term $1 - n/\lambda_1$, which appears in the credible interval widths, $\mathrm{err}_{\mathrm{MLE}}$, $\mathrm{err}_{\mathrm{full}}$, and $\mathrm{err}_{\mathrm{GCV}}$, may suffer from cancellation error. We can avoid this cancellation error by modifying how we compute the Gram matrix and its eigenvalues.

Any shift-invariant or digital shift-invariant covariance kernel satisfying (3.4) can be written as $C_{\boldsymbol{\theta}} = 1 + \mathring{C}_{\boldsymbol{\theta}}$, where $\mathring{C}_{\boldsymbol{\theta}}$ is also symmetric and positive definite. The associated Gram matrix for $\mathring{C}_{\boldsymbol{\theta}}$ is then $\mathring{\mathsf{C}}_{\boldsymbol{\theta}} = \mathsf{C}_{\boldsymbol{\theta}} - \mathbf{1}\mathbf{1}^T$, and the eigenvalues of $\mathring{\mathsf{C}}_{\boldsymbol{\theta}}$ are $\mathring{\lambda}_1 = \lambda_1 - n, \lambda_2, \ldots, \lambda_n$, which follows because $\mathbf{1}$ is the first eigenvector of both $\mathsf{C}_{\boldsymbol{\theta}}$ and $\mathring{\mathsf{C}}_{\boldsymbol{\theta}}$. Note that $\mathring{C}_{\boldsymbol{\theta}}$ inherits the shift-invariant properties of $C_{\boldsymbol{\theta}}$. Then,

$$1 - \frac{n}{\lambda_1} = \frac{\lambda_1 - n}{\lambda_1} = \frac{\mathring{\lambda}_1}{\mathring{\lambda}_1 + n},$$

where now the right hand side is free of cancellation error.

We show how to compute $\mathring{C}_{\boldsymbol{\theta}}$ without introducing round-off error. The covariance functions that we use are of product form, namely,

$$C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = \prod_{\ell=1}^{d} \left[1 + \mathring{C}_{\boldsymbol{\theta},\ell}(t_\ell, x_\ell)\right], \qquad \mathring{C}_{\boldsymbol{\theta},\ell} : [0,1]^2 \to \mathbb{R}.$$

Direct computation of $\mathring{C}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) - 1$ introduces cancellation error if the $\mathring{C}_\ell$ are small. So, we employ the iteration

$$\begin{aligned}
\mathring{C}^{(1)}_{\boldsymbol{\theta}} &= \mathring{C}_{\boldsymbol{\theta},1}(t_1, x_1), \\
\mathring{C}^{(\ell)}_{\boldsymbol{\theta}} &= \mathring{C}^{(\ell-1)}_{\boldsymbol{\theta}}[1 + \mathring{C}_{\boldsymbol{\theta},\ell}(t_\ell, x_\ell)] + \mathring{C}_{\boldsymbol{\theta},\ell}(t_\ell, x_\ell), \qquad \ell = 2, \ldots, d, \\
\mathring{C}_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) &= \mathring{C}^{(d)}_{\boldsymbol{\theta}}.
\end{aligned}$$

In this way, the Gram matrix $\mathring{\mathsf{C}}_{\boldsymbol{\theta}}$, whose i,j-element is $\mathring{C}_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ can be constructed with minimal round-off error.

Computing the eigenvalues of $\mathring{\mathsf{C}}_{\boldsymbol{\theta}}$ via the procedure given in (3.3) yields $\mathring{\lambda}_1 = \lambda_1 - n, \lambda_2, \ldots, \lambda_n$. The estimates of $\boldsymbol{\theta}$ are computed in terms of the eigenvalues of $\mathring{\mathsf{C}}_{\boldsymbol{\theta}}$, so (3.6a) and (3.6b) become

$$\boldsymbol{\theta}_{\mathrm{MLE}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i} \right) + \frac{1}{n} \sum_{i=1}^{n} \log(\lambda_i) \right], \tag{6.1a}$$

$$\boldsymbol{\theta}_{\mathrm{GCV}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2} \right) - 2\log\left(\sum_{i=1}^{n} \frac{1}{\lambda_i} \right) \right], \tag{6.1b}$$

where $\lambda_1 = n + \mathring{\lambda}_1$. The widths of the credible intervals in (3.8a), (3.8b), and (3.8c) become

$$\mathrm{err}_{\mathrm{MLE}} = \frac{2.58}{n} \sqrt{\frac{\mathring{\lambda}_1}{\lambda_1} \sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i}}, \tag{6.2a}$$

$$\mathrm{err}_{\mathrm{full}} = \frac{t_{n_j-1,0.995}}{n} \sqrt{\frac{\mathring{\lambda}_1}{n-1} \sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i}}, \tag{6.2b}$$

$$\mathrm{err}_{\mathrm{GCV}} = \frac{2.58}{n} \sqrt{\frac{\mathring{\lambda}_1}{\lambda_1} \sum_{i=2}^{n} \frac{|\widetilde{y}_i|^2}{\lambda_i^2} \left[\frac{1}{n} \sum_{i=1}^{n} \frac{1}{\lambda_i} \right]^{-1}}. \tag{6.2c}$$

Since $\mathring{\lambda}_1 = \lambda_1 - n$ and $\lambda_1 \sim n$ it follows $\mathring{\lambda}_1/\lambda_1 \approx \mathring{\lambda}_1/(n-1)$ and is small for large n. Moreover, for large n, the credible intervals via empirical Bayes and full Bayes are similar, since $t_{n_j-1,0.995}$ is approximately 2.58.

The computational steps for the improved, faster, automatic Bayesian cubature are detailed in Algorithm 2.

In comparison to Algorithm 1, the second and third components of the computational cost of Algorithm 2 are substantially reduced. The Algorithm 2 has a computational cost which is the sum of the following:

- $\mathcal{O}\big(n\$(f)\big)$ for the integrand data, where $\$(f)$ is the computational cost of a

Algorithm 2 Fast Automatic Bayesian Cubature

Require: a generator for the rank-1 Lattice sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ and a matching shift-invariant kernel, $C_{\boldsymbol{\theta}}$ or a generator for Sobol sequence and a matching digital shift-invariant kernel, $C_{\boldsymbol{\theta}}$; a black-box function, f; an absolute error tolerance, $\varepsilon > 0$; the positive initial sample size, n_0, that is a power of 2; the maximum sample size $n_{\max}$

1: $n \leftarrow n_0,\ n' \leftarrow 0,\ \mathrm{err}_{\mathrm{CI}} \leftarrow \infty$

2: **while** $\mathrm{err}_{\mathrm{CI}} > \varepsilon$ and $n \le n_{\max}$ **do**

3: Generate $\{\boldsymbol{x}_i\}_{i=n'+1}^{n}$ and sample $\{f(\boldsymbol{x}_i)\}_{i=n'+1}^{n}$

4: Compute $\boldsymbol{\theta}$ by (3.6a) or (3.6b)

5: Compute $\mathrm{err}_{\mathrm{CI}}$ according to (6.2a), (6.2b), or (6.2c)

6: $n' \leftarrow n,\ n \leftarrow 2n'$

7: **end while**

8: Update sample size to compute $\widehat{\mu}$, $n \leftarrow n'$

9: Compute $\widehat{\mu}$, the approximate integral, according to (3.7)

10: **return** $\widehat{\mu}$, n and $\mathrm{err}_{\mathrm{CI}}$

single $f(\boldsymbol{x})$;

- $\mathcal{O}\big(N_{\text{opt}} n \$(C_{\boldsymbol{\theta}})\big)$ for the evaluations of the vector $\boldsymbol{C}_1$, where N_{opt} is the number of optimization steps required, and $\$(C_{\boldsymbol{\theta}})$ is the computational cost of a single $C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x})$; and
- $\mathcal{O}\big(N_{\text{opt}} n \log(n)\big)$ for the FFT calculations; there is no d dependence in these calculations.

6.2 Kernel Parameters Search

JR: Explain the transformation used to make the search range positive, > 0 , etc.

The various parameters introduced and used by our algorithms need to be optimally chosen. We have not discussed how that will be done. The parameter search can be done in two major ways. Bounded minima search if the search interval is known else unbounded search. Most of the time, the search interval is unknown. So the natural choice is to use unbounded search over the unbound domain such as is done by our algorithms using `fminsearch` provided by MATLAB. Our parameters need to live in a domain that is bounded or semi-bounded. There are some simple domain transformations available to achieve such constraints.

6.2.1 Positive kernel shape parameter. The following parameter map is used to ensure that the shape parameter values are positive real numbers,

$$\eta(t) = \quad e^t, \qquad\qquad \eta : (-\infty, \infty) \to (0, \infty).$$

So one may search for the optimal $t_1 = \log(\eta)$ over the whole real line $\mathbb{R}$. The optimal value $t_{1,\text{opt}}$ can be transformed to back the $(0, \infty)$ interval using

$$\eta_{\text{opt}} = e^{t_{\text{opt}}}.$$

6.2.2 Kernel order $1 < r < \infty$. The following map is used to ensure that the kernel order values are positive real number and greater than one, i.e., in the $(1, \infty)$ interval as required in Section 4.3.2,

$$r(t) = \quad 1 + e^{-t}, \qquad\qquad r : (-\infty, \infty) \to (1, \infty).$$

So one may search for the optimal $t_2 = -\log(r - 1)$ in the whole real line $\mathbb{R}$. The optimal value $t_{2,\text{opt}}$ can be transformed back to the desired interval $(0, 1)$ using

$$r_{\text{opt}} = 1 + e^{-t_{2,\text{opt}}}.$$

6.2.3 Kernel order $0 < q < 1$. The following map is used to ensure that the kernel order values are positive real and less than one, i.e., in the $(0, 1)$ interval to use with exponentially decaying kernel, as introduced in Section 4.3.1,

$$q(t) = \quad \frac{1}{1 + e^{-t}}, \qquad\qquad q : (-\infty, \infty) \to (0, 1).$$

So one may search for the optimal $t_3 = -\log(q^{-1} - 1)$ in the whole real line $\mathbb{R}$. The optimal value $t_{3,\text{opt}}$ can be transformed back to the desired interval $(0, 1)$ by using

$$q_{\text{opt}} = \frac{1}{1 + e^{-t_{3,\text{opt}}}}.$$

CHAPTER 7

NUMERICAL RESULTS AND OBSERVATIONS

JR: use uniformly randomly chosen ε instead 4 fixed

Bayesian cubature algorithms developed in this work are demonstrated using three commonly used integration examples. These integrals were evaluated using both the algorithms `cubBayesLattice_g` and `cubBayesNet_g`. The first example shows evaluating a multivariate Gaussian probability given the interval. The second example shows integrating the Keister's function, and the final example shows computing an Asian arithmetic option pricing.

7.1 Testing Methodology

Four hundred different randomly chosen error tolerances, ε, were set for each example, with the tolerances chosen from a fixed interval. The error tolerance intervals were chosen depending on the difficulty of the problem. The nodes used in `cubBayesLattice_g` were the randomly shifted lattice rules supplied by GAIL. Where as the nodes used in `cubBayesNet_g` were the randomly scrambled Sobol' nodes supplied by MATLAB's Sobol' sequence generator.

For each integral, and each of our stopping criteria—empirical Bayes, full Bayes, and generalized cross-validation—our algorithm is run 400 times with each randomly chosen error tolerance as mentioned above. For each test, the execution time is plotted against $|\mu - \widehat{\mu}|/\varepsilon$. We expect $|\mu - \widehat{\mu}|/\varepsilon$ to be no greater than one, but hope that it is not too much smaller than one, which would indicate a stopping criterion that is too conservative.

Periodization variable transforms are used in the examples with `cubBayesLattice_g` as the algorithm assumes the integrands to be periodic in $[0,1]^d$. But the `cubBayesNet_g` does not need this additional requirement, so the integrands are used as such.

7.2 Multivariate Gaussian Probability

This example was already introduced in Section 2.5, where we used the Matérn covariance kernel. We reuse f_{Genz} (2.23) and apply periodization transform when required.

7.2.1 Using `cubBayesLattice_g`. As required by the algorithm, we apply Sidi's C^2 periodization to f_{Genz} (2.23), and chose $d = 3$ and $r = 2$. The simulation results for this example function are summarized in Figures 7.1, 7.2, and 7.3. In all cases the Bayesian cubature returns an approximation within the prescribed error tolerance. We used the same setting as before with generic slow Bayesian cubature in Section 2.5 for comparision. For error threshold $\varepsilon = 10^{-5}$ with empirical stopping criterion, our fast algorithm takes 0.001 seconds as shown in Figure 7.1 whereas the basic algorithm takes 30 seconds as shown in Figure 2.4. Amongst the three stopping criteria, GCV achieved the results faster than others. One can also observe from the figures, the credible intervals are wider, causing the true error much smaller than requested. This could be due to the periodization transformed f_{Genz} is smoother than the $r = 2$ kernel could approximate. Using a kernel of matching smoothness could produce right credible intervals.

7.2.2 Using `cubBayesNet_g`. Here we use f_{Genz} (2.23) without any periodization, and chose $d = 3$ and $r = 1$. The simulation results for this example function are summarized in Figures 7.4, 7.5, and 7.6. In all cases the `cubBayesNet_g` returns an approximation within the prescribed error tolerance. We used the same setting as before with generic slow Bayesian cubature in Section 2.5 for comparision. For error threshold $\varepsilon = 10^{-5}$ with empirical stopping criterion, our fast algorithm takes about 2 seconds as shown in Figure 7.1 whereas the basic algorithm takes 30 seconds as shown in Figure 2.4. `cubBayesNet_g` uses fast Walsh transform which is slower in MATLAB due to the way it was implemented. This is reason it takes more time than the

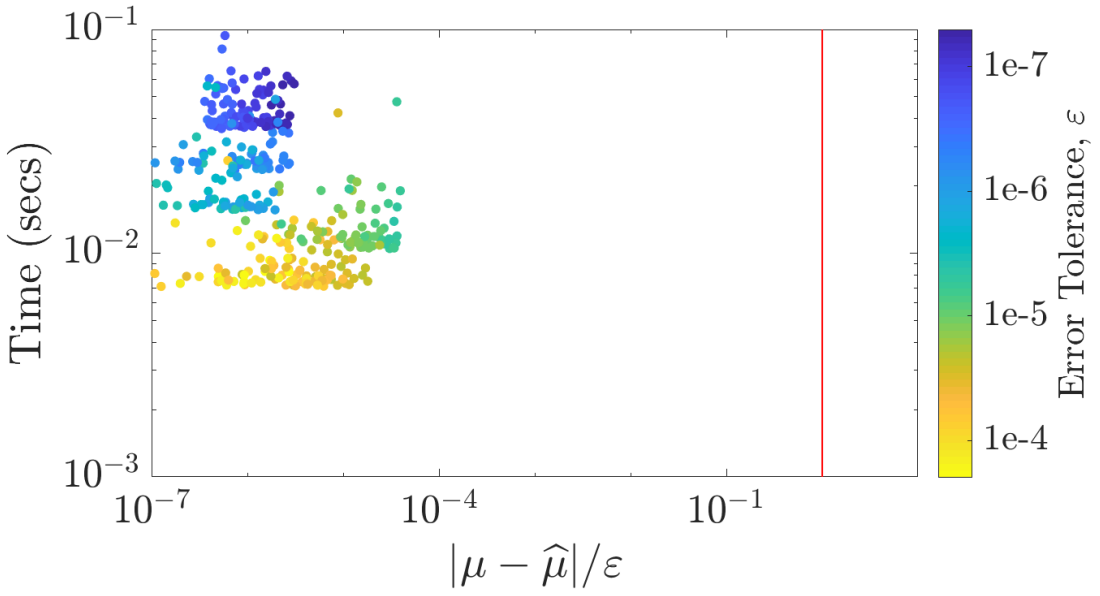


Figure 7.1. `cubBayesLattice_g`: Multivariate normal probability example using the empirical Bayes stopping criterion.

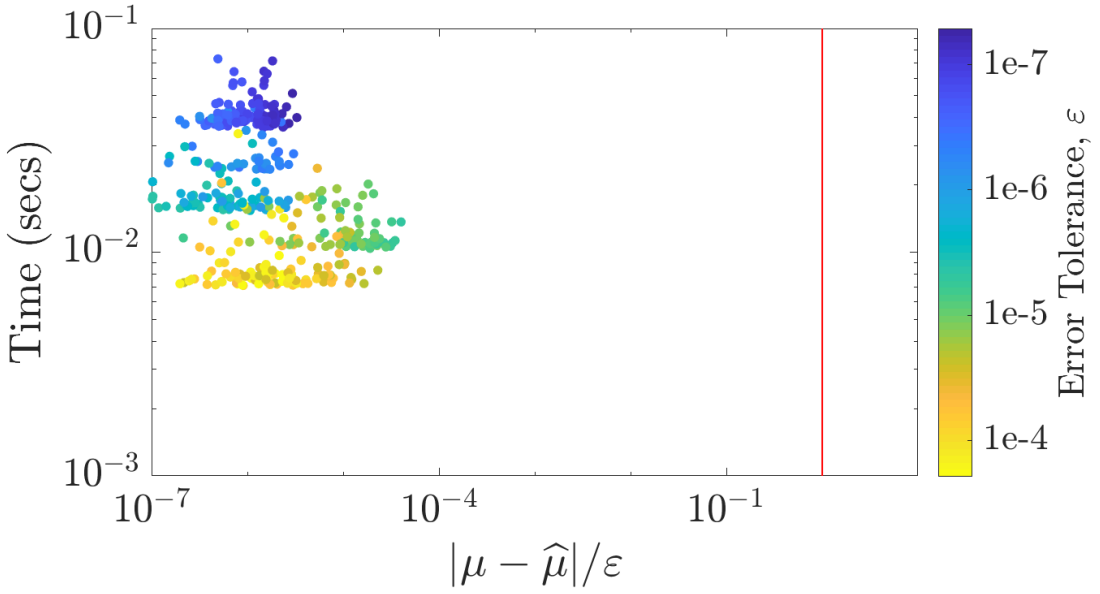


Figure 7.2. `cubBayesLattice_g`: Multivariate normal probability example using the full Bayes stopping criterion.

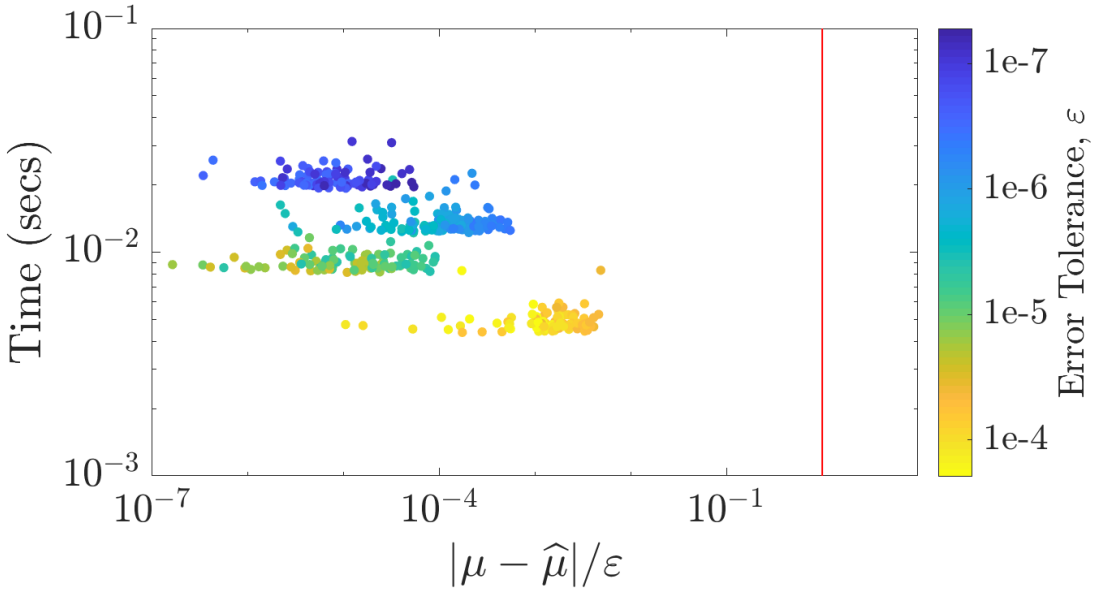


Figure 7.3. `cubBayesLattice_g`: Multivariate normal probability example using the GCV stopping criterion.

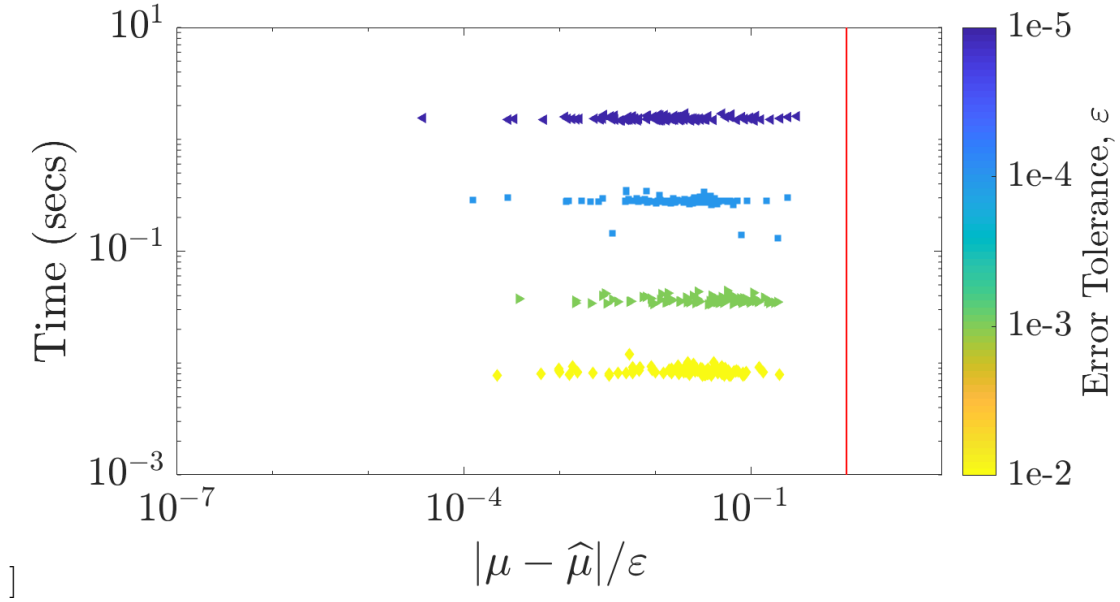
`cubBayesLattice_g`. But comparing the number of samples, n, used for integration provides more insight which directly relates to the alogrithm's computational cost. The `cubBayesLattice_g` used $n = 16384$ samples whereas `cubBayesNet_g` used $n = 32768$ samples even with $r = 1$ order kernel.

Amongst the three stopping criteria, GCV achieved the results faster than others. One can also observe from the figures, the credible intervals are narrower than we observed in Figure 7.1. This shows `cubBayesNet_g` with $r = 1$ kernel more accurately approximates the integrand.

7.3 Keister's Example

This multidimensional integral function comes from [43] and is inspired by a physics application:

$$\begin{aligned} \mu &= \int_{\mathbb{R}^d} \cos(\|\boldsymbol{t}\|) \exp(-\|\boldsymbol{t}\|^2) \, \mathrm{d}\boldsymbol{t} \\ &= \int_{[0,1]^d} f_{\text{Keister}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \end{aligned} \tag{7.1}$$



]

Figure 7.4. `cubBayesNet_g`: Multivariate normal probability example with empirical Bayes stopping criterion.

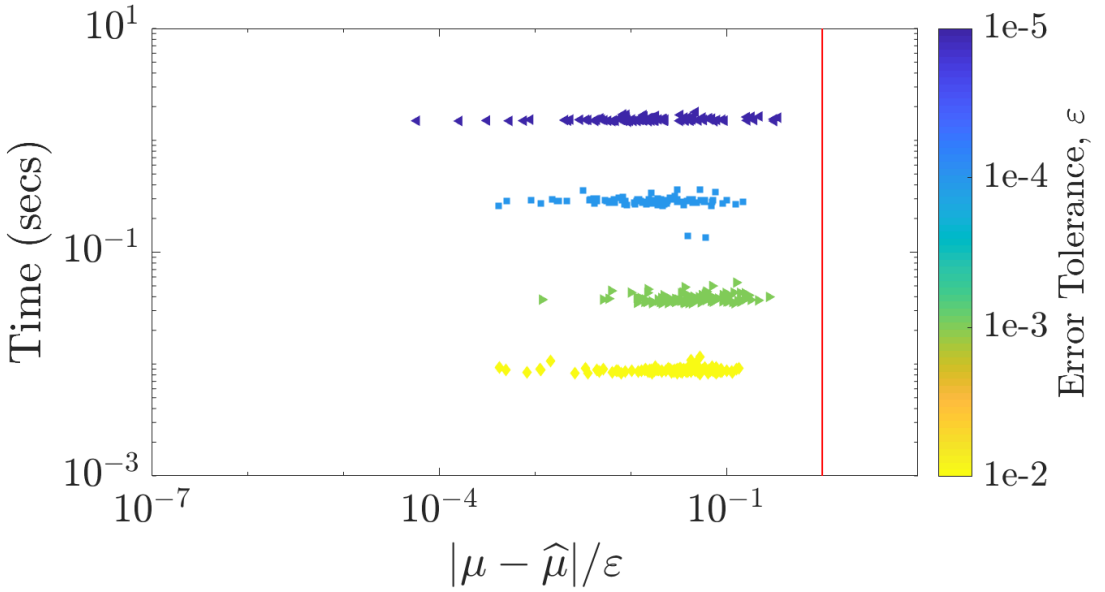


Figure 7.5. `cubBayesNet_g`: Multivariate normal probability example with the full-Bayes stopping criterion.

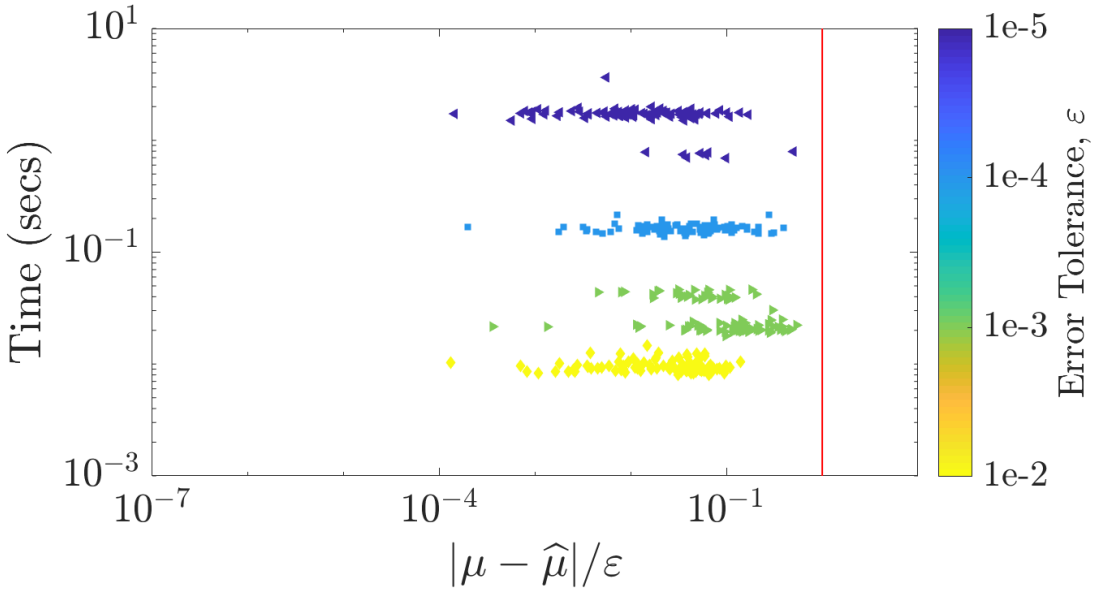


Figure 7.6. `cubBayesNet_g`: Multivariate normal probability example with the GCV stopping criterion.

where

$$f_{\text{Keister}}(\boldsymbol{x}) = \pi^{d/2} \cos\left(\left\|\Phi^{-1}(\boldsymbol{x})/2\right\|\right),$$

and again Φ is the standard normal distribution. The true value of μ can be calculated iteratively in terms of a quadrature as follows:

$$\mu = \frac{2\pi^{d/2} I_c(d)}{\Gamma(d/2)}, \quad d = 1, 2, \ldots$$

where Γ denotes the gamma function, and

$$\begin{aligned}
I_c(1) &= \frac{\sqrt{\pi}}{2\exp(1/4)}, \\
I_s(1) &= \int_{x=0}^{\infty} \exp(-\boldsymbol{x}^T\boldsymbol{x}) \sin(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} \\
&= 0.4244363835020225, \\
I_c(2) &= \frac{1 - I_s(1)}{2}, \qquad I_s(2) = \frac{I_c(1)}{2} \\
I_c(j) &= \frac{(j-2)I_c(j-2) - I_s(j-1)}{2}, \qquad j = 3, 4, \ldots \\
I_s(j) &= \frac{(j-2)I_s(j-2) - I_c(j-1)}{2}, \qquad j = 3, 4, \ldots.
\end{aligned}$$

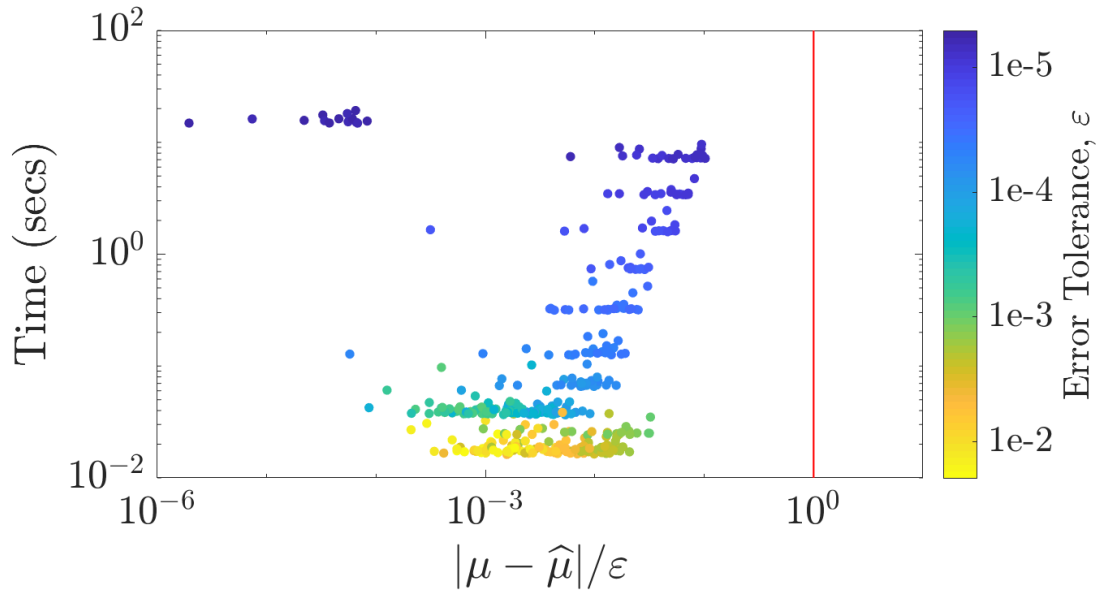


Figure 7.7. `cubBayesLattice_g`: Keister example using the empirical Bayes stopping criterion.

7.3.1 Using `cubBayesLattice_g`. JR: Discuss the big gap between 1e-5 and 1e-4 Figures 7.7, 7.8 and 7.9 summarize the numerical tests for this integral. We used the Sidi's C^1 periodization, dimension $d = 4$, and $r = 2$. As we can see the GCV stopping criterion achieved the results faster than the others similar to the multivariate Gaussian case.

7.3.2 Using `cubBayesNet_g`. Figures 7.10, 7.11 and 7.12 summarize the numerical tests for this case. We used dimension $d = 4$, and $r = 1$. No periodization transform was used as the integrand need not be periodic. In this example, we use $r = 1$ order kernel where as in Section 7.3.1, $r = 2$ kernel was used, which necessitates the `cubBayesNet_g` to use more samples for integration. As we can see the GCV stopping criterion achieved the results faster than the others similar to the multivariate Gaussian case.

7.4 Option Pricing

The price of financial derivatives can often be modeled by high dimensional

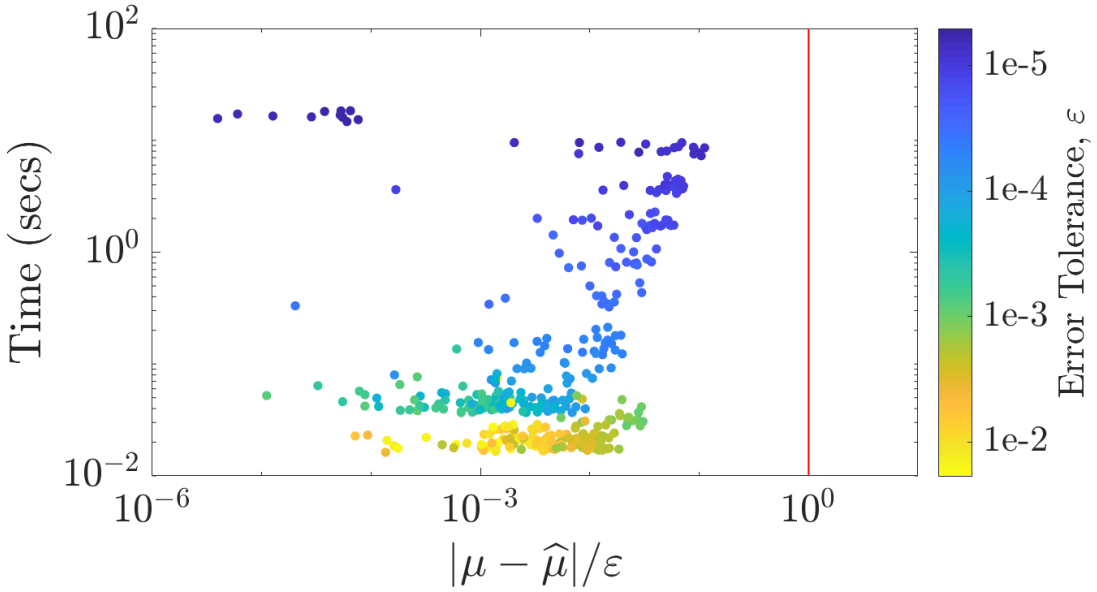


Figure 7.8. `cubBayesLattice_g`: Keister example using the full Bayes stopping criterion.

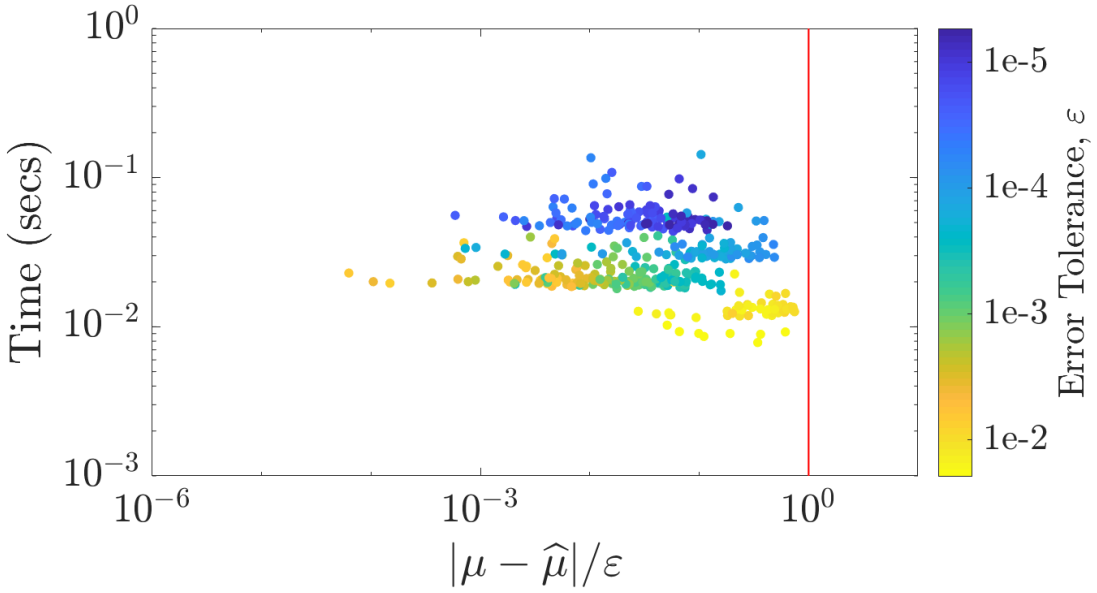


Figure 7.9. `cubBayesLattice_g`: Keister example using the GCV stopping criterion.

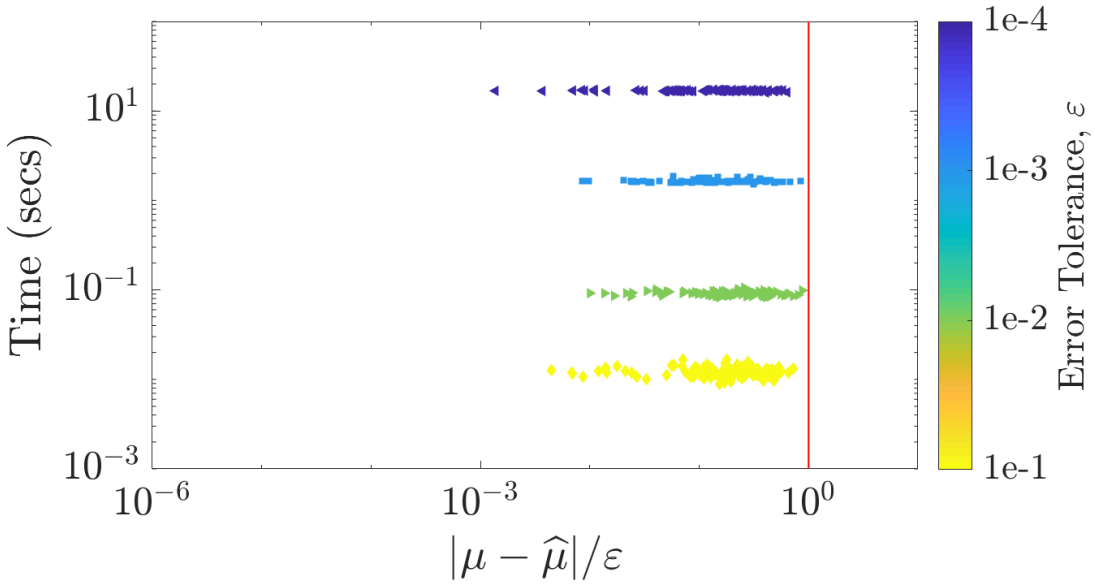


Figure 7.10. `cubBayesNet_g`: Keister example using the empirical Bayes stopping criterion.

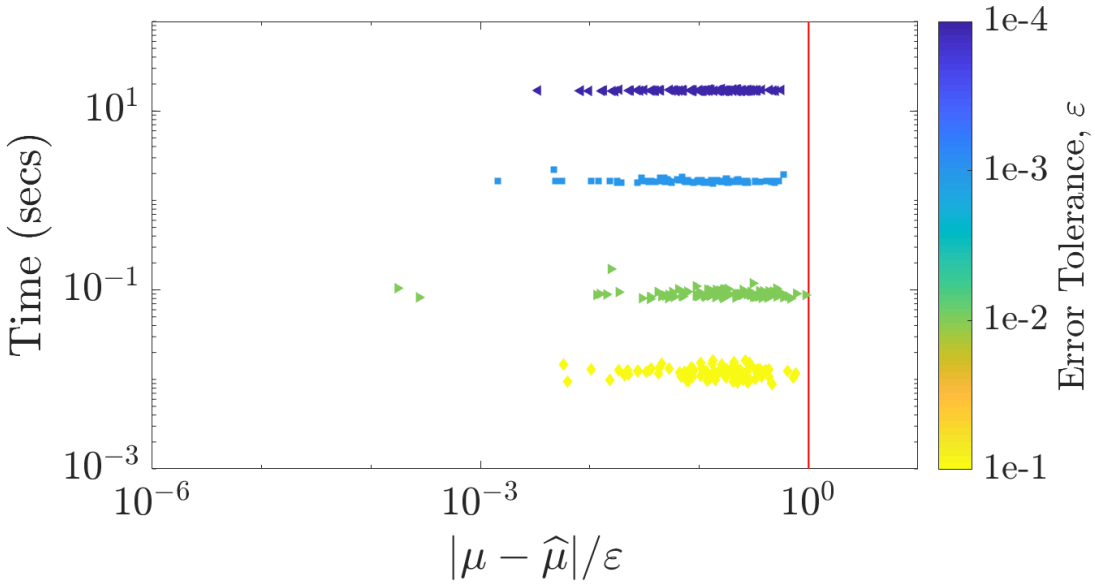


Figure 7.11. `cubBayesNet_g`: Keister example using the full-Bayes stopping criterion.

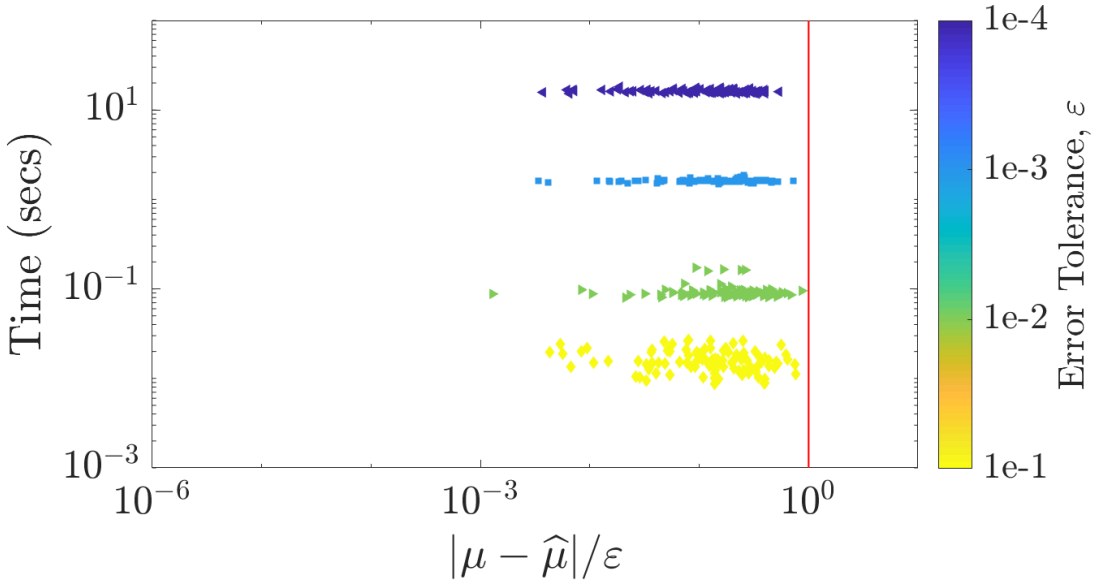


Figure 7.12. `cubBayesNet_g`: Keister example using the GCV stopping criterion.

integrals. If the underlying asset is described in terms of a discretized geometric Brownian motion, then the fair price of the option is:

$$\mu = \int_{\mathbb{R}^d} \text{payoff}(\boldsymbol{z}) \frac{\exp(\frac{1}{2}\boldsymbol{z}^T \Sigma^{-1} \boldsymbol{z})}{\sqrt{(2\pi)^d \det(\Sigma)}} \, \mathrm{d}\boldsymbol{z} = \int_{[0,1]^d} f(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x},$$

where payoff$(\cdot)$ defines the discounted payoff of the option,

$$\Sigma = (T/d)\big(\min(j,k)\big)_{j,k=1}^{d} = \mathsf{L}\mathsf{L}^T,$$

$$f(\boldsymbol{x}) = \text{payoff}\left(\mathsf{L}\begin{pmatrix} \Phi^{-1}(x_1) \\ \vdots \\ \Phi^{-1}(x_d) \end{pmatrix}\right).$$

The Asian arithmetic mean call option has a payoff of the form

$$\text{payoff}(\boldsymbol{z}) = \max\left(\frac{1}{d}\sum_{j=1}^{d} S_j(\boldsymbol{z}) - K, 0\right) \mathrm{e}^{-rT},$$

$$S_j(\boldsymbol{z}) = S_0 \exp\big((r - \sigma^2/2)jT/d + \sigma\sqrt{T/d}\, z_j\big).$$

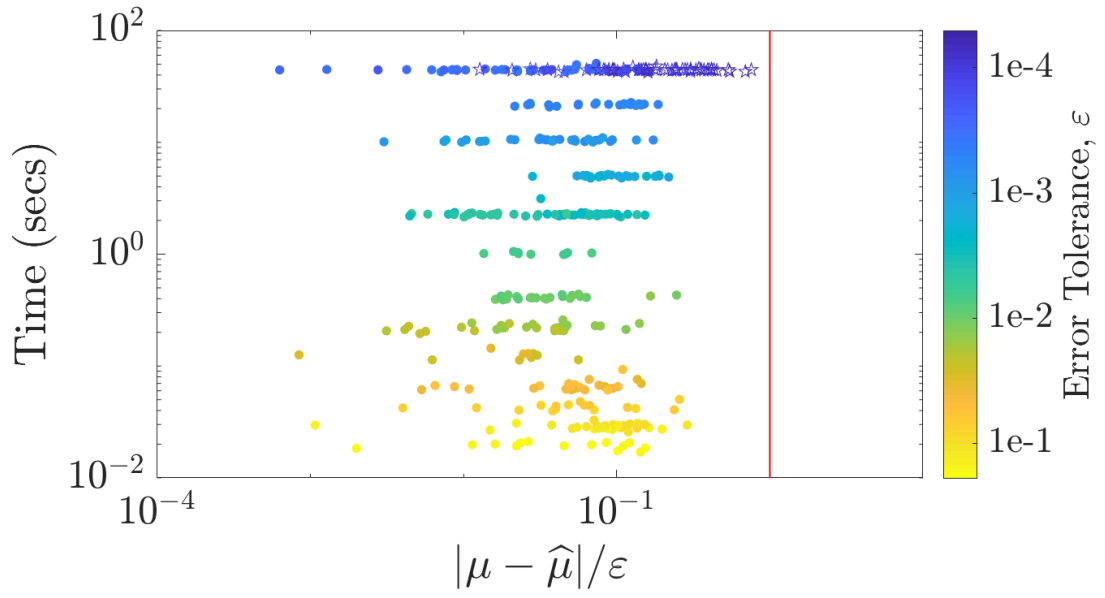


Figure 7.13. `cubBayesLattice_g`: Option pricing using the empirical Bayes stopping criterion.

Here, T denotes the time to maturity of the option, d the number of time steps, S_0 the initial price of the stock, r the interest rate, σ the volatility, and K the strike price.

7.4.1 Using `cubBayesLattice_g`. The Figures 7.13, 7.14 and 7.15 summarize the numerical results for this example using $T = 1/4$, $d = 13$, $S_0 = 100$, $r = 0.05$, $\sigma = 0.5$, $K = 100$. Moreover, L is chosen to be the matrix of eigenvectors of $\mathsf{\Sigma}$ times the square root of the diagonal matrix of eigenvalues of $\mathsf{\Sigma}$. Because the integrand has a kink caused by the max function, it does not help to use a periodizing transform that is very smooth. We choose the baker's transform (4.20) and $r = 1$.

7.4.2 Using `cubBayesNet_g`. The Figures 7.16, 7.17 and 7.18 summarize the numerical results for the option pricing example using the same values as in Section 7.4.1, $T = 1/4$, $d = 13$, $S_0 = 100$, $r = 0.05$, $\sigma = 0.5$, $K = 100$. As mentioned before, this integrand has a kink caused by the max function, So, `cubBayesNet_g` could be more efficient than `cubBayesLattice_g`, as no periodization transform is required.

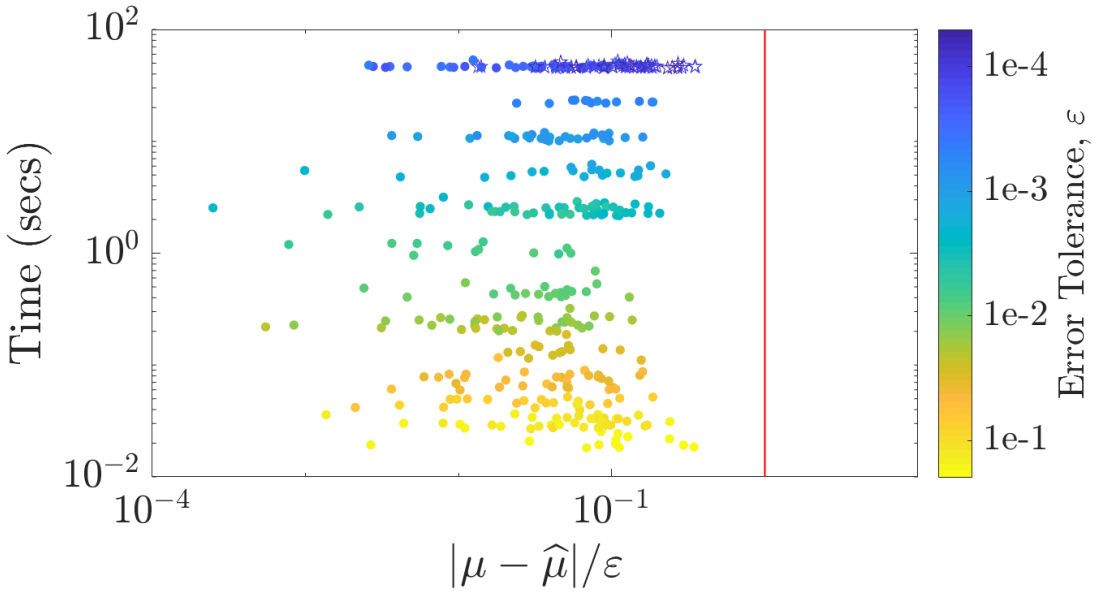


Figure 7.14. `cubBayesLattice_g`: Option pricing using the full Bayes stopping criterion.

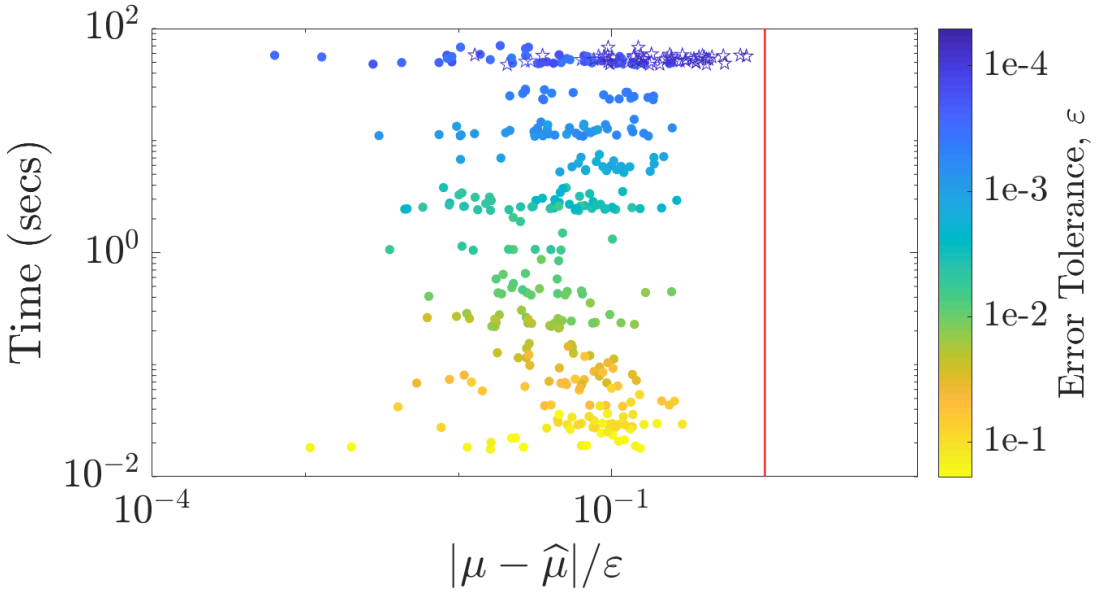


Figure 7.15. `cubBayesLattice_g`: Option pricing using the GCV stopping criterion.

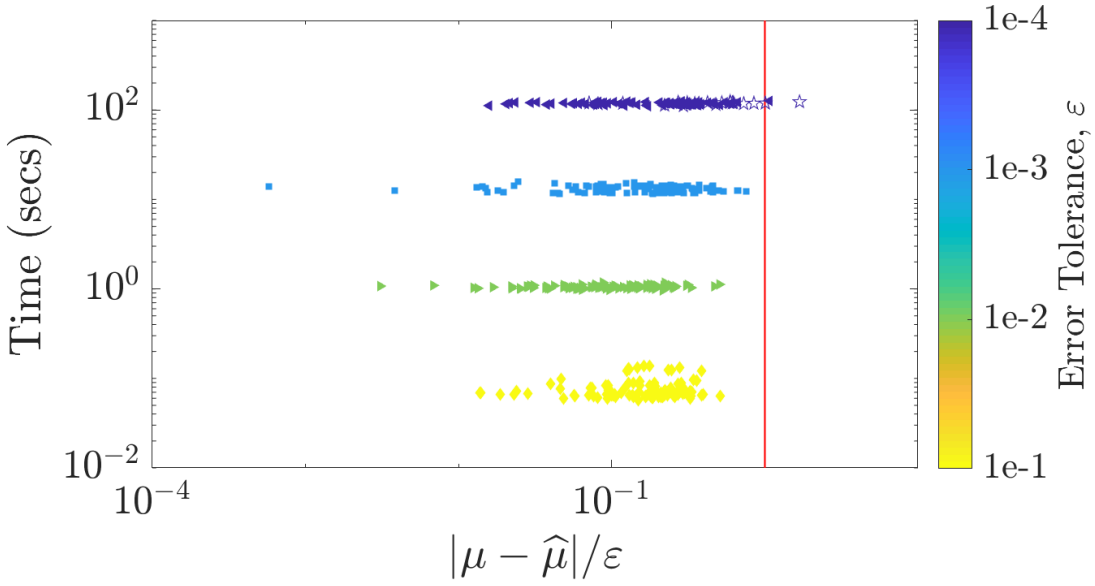


Figure 7.16. `cubBayesNet_g`: Option pricing using the empirical Bayes stopping criterion.

This can be observed from the number of samples used for intgration to meet the same error threshold. For the error tolerance $\varepsilon = 10^{-3}$, the `cubBayesLattice_g` used $n = 2^{20}$ samples, whereas the `cubBayesNet_g` used $n = 2^{17}$ samples.

7.5 Discussion

As shown in Figures 7.1 to 7.18, both the algorithms computed integral within user specified threshold most of the times except on a few occasions. This is especially the case with option pricing example due to the complexity and high dimension of the integrand. Also notice the `cubBayesLattice_g` algorithm finished within 10seconds of time for Keister and multivariate Gaussian. For the option pricing it look closer to 70seconds. This is again due to the complexity of the integrand.

Another noticeable aspect from the plots of `cubBayesLattice_g` is how much the error bounds differ from the true error. For option pricing example, the error bound is not as conservative as it is for the multivariate Gaussian and Keister examples. A possible reason is that the latter integrands are significantly smoother than

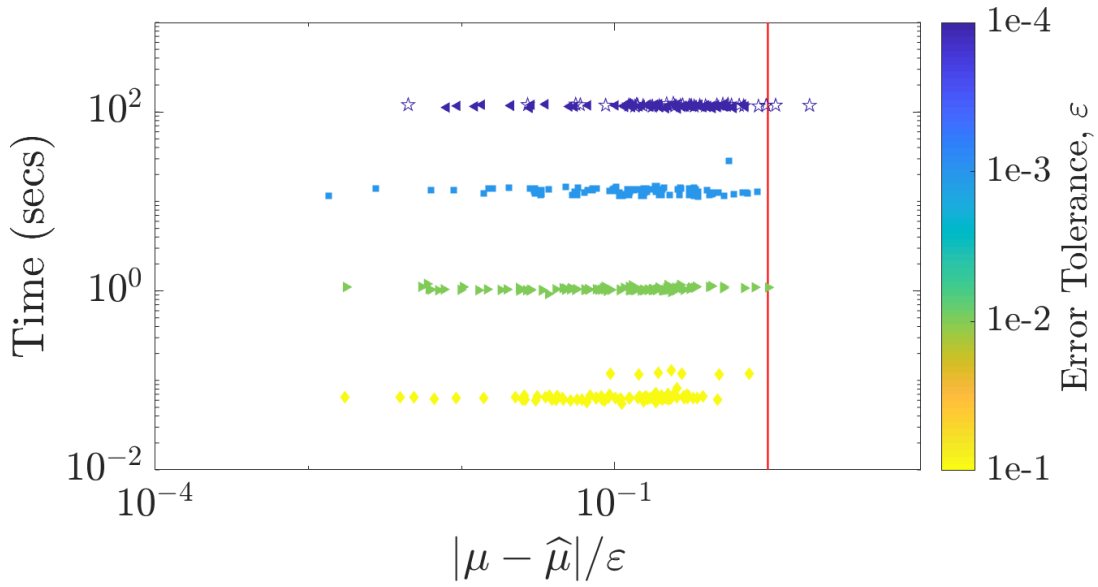


Figure 7.17. `cubBayesNet_g`: Option pricing using the full-Bayes stopping criterion.

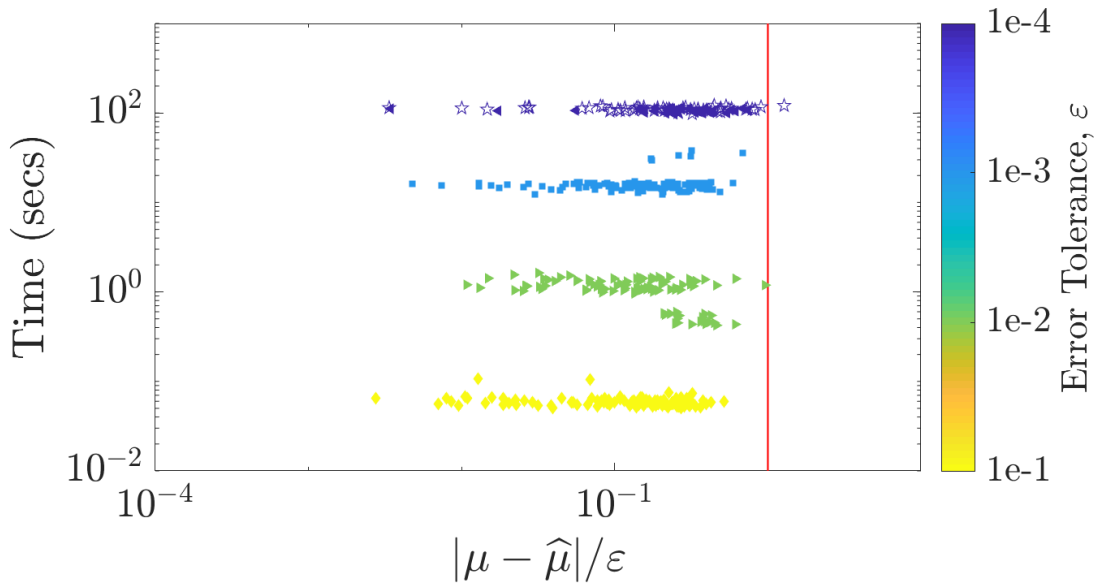


Figure 7.18. `cubBayesNet_g`: Option pricing using the GCV stopping criterion.

is assumed by our covariance kernel. This is a matter for further investigation.

Most noticeable aspect from the plots of `cubBayesNet_g` is how closer the error bounds are to the true error. This shows the `cubBayesNet_g`'s estimation of expected error in the stopping criterion is very accurate. Similar to `cubBayesLattice_g`, it missed meeting the given error threshold for the option pricing example, as marked by the hollow stars, for $\varepsilon = 10^{-4}$. This is because the complexity of the integrand and the algorithm reached max allowed number of samples $n = 2^{20}$.

7.6 Comparison with `cubMC_g`, `cubLattice_g` and `cubSobol_g`

GAIL library provides variety of numerical integration algorithms based on different theoretical foundation, We would like to compare how our algorithms perform relatively. We consider three GAIL algorithms 1) `cubMC_g` a simple Monte-Carlo method for multi-dimensional integration, 2) `cubLattice_g` a quasi-Monte-Carlo method using Lattice points, and 3) `cubSobol_g` a quasi-Monte-Carlo method using Sobol points.

7.6.1 Keister integral. The Table 7.1 summarizes the performance of the methods MC, Lattice, Sobol, BayesLat, and BayesSob—they refer to the GAIL cubatures, `cubMC_g`, `cubLattice_g`, `cubSobol_g`, `cubBayesLattice_g`, `cubBayesNet_g`, respectively for estimating Keister integral defined in (7.1). We conducted two simulations with $d = 3, 8$. In the case of $d = 3$, all five methods succeeded completely meaning the absolute error is less than given tolerance, i.e., $|\mu - \hat{\mu}| \leq \varepsilon$, where $\hat{\mu}$ is a cubature's approximated value. The fastest method was `cubBayesLattice_g`. In the case of $d = 8$, `cubSobol_g` achieved 100% success rate and was the fastest. But `cubBayesLattice_g` was competitive and had the smallest average absolute error. `cubBayesNet_g` used lowest number of samples in case of $d = 8$ but slower than `cubSobol_g`.

JR: avoid Exp notation for 2 decimals

Table 7.1. Comparison of average performance of cubatures for estimating the integral (7.1) for 1000 independent runs. These results can be conditionally reproduced with the script, `KeisterCubatureExampleBayes.m`, in GAIL.

$d = 3,\ \varepsilon = 0.005$					
Method	MC	Lattice	Sobol	BayesLat	BayesSobol
Absolute Error	0.001 100	0.000 510	0.000 520	0.000 430	0.000 560
Tolerance Met	100%	100%	100%	100%	100%
n	2 500 000	4100	3900	1000	1900
Time (seconds)	0.1800	0.0069	0.0054	0.0029	0.0700

$d = 8,\ \varepsilon = 0.050$					
Method	MC	Lattice	Sobol	BayesLat	BayesSobol
Absolute Error	0.012 000	0.015 000	0.007 300	0.001 800	0.008 300
Tolerance Met	100%	99%	100%	100%	100%
n	7 400 000	15 000	16 000	66 000	8200
Time (seconds)	1.2000	0.0220	0.0160	0.2100	0.3500

7.6.2 Multi-variate Normal. The Table 7.2 summarizes the performance of the methods MC, Lattice, Sobol, BayesLat, and BayesSob for estimating the multi-dimensional Normal probability $\mathbf{X} \sim \mathbf{N}(\mu, \mathbf{\Sigma})$. This experiment demonstrates our algorithm's ability to handle very high-dimensional integral.

We conducted two simulations with different Σ and estimation intervals $(\mathbf{a}, \mathbf{b})$ but fixed $\mu = 0$ and required error threshold $\varepsilon = 10^{-3}$. In the first case, all five methods succeeded completely. The fastest method was `cubBayesLattice_g` but codecubBayesNet_g used the lowest number of samples. In the second case also all five methods succeeded, but `cubLattice_g` was the fastest. The `cubBayesNet_g` was competitive and had the smallest average absolute error using lowest number of samples. The `cubBayesLattice_g` achieved the next lowest average error but slower than

`cubSobol_g`.

Table 7.2. Comparison of average performance of cubatures for estimating the $d = 20$ Multi-variate Normal (2.23) for 1000 independent runs with $\varepsilon = 10^{-3}$. These results can be conditionally reproduced with the script, `MVNCubatureExampleBayes.m`, in GAIL.

	$\Sigma = \mathbf{I_d},\ \mathbf{b} = -\mathbf{a} = (\mathbf{3.5}, \ldots, \mathbf{3.5})$				
Method	MC	Lattice	Sobol	BayesLat	BayesSobol
Absolute Error	2.20E−16	2.70E−14	2.70E−14	2.20E−16	2.20E−16
Tolerance Met	100%	100%	100%	100%	100%
n	10 000	1000	1000	1000	260
Time (seconds)	0.0410	0.0820	0.0710	0.0650	0.0790

	$\Sigma = 0.4\mathbf{I_d} + \mathbf{0.6 1 1^T},\ \mathbf{a} = (-\infty, \ldots, -\infty),\ \mathbf{b} = \sqrt{\mathbf{d}}(\mathbf{U_1}, \ldots, \mathbf{U_d})$				
Method	MC	Lattice	Sobol	BayesLat	BayesSobol
Absolute Error	2.30E−4	2.10E−4	4.40E−4	1.00E−4	4.80E−5
Tolerance Met	100%	100%	100%	100%	100%
n	10 000	1000	1000	1000	260
Time (seconds)	0.0350	0.0120	0.0140	0.0150	0.0300

7.7 Shape Parameter Fine-tuning

JR: Numerical examples for the case of shape parameter per dimension

Allowing the kernel shape parameter to vary for each dimension could improve the accuracy of numerical integration when the integrand under consideration has only very low effective dimension such as Option Pricing example we demonstrated. We demonstrate this advantage by integrating a function that is not symmetric across dimensions,

$$f(\boldsymbol{x}) = \sum_{j=1}^{d} v_j \sin(2\pi x_j^2) \tag{7.2}$$

which has known integral

$$\int_{[0,1)^d} f(\boldsymbol{x}) = \frac{1}{2}\,\texttt{fresnels}(d) \sum_{j=1}^{d} v_j$$

where `fresnels` is the Fresnel Sine integral,

$$\texttt{fresnels}(z) = \int_0^z \sin\left(\frac{\pi t^2}{2}\right) \mathrm{d}t.$$

Table 7.3. Comparison of average performance of Bayesian Cubature with common shape parameter vs dimension specific shape parameter for estimating the $d = 3$ Fresnel Sine integral. These results can be conditionally reproduced with the script, `demoMultiTheta.m`, in GAIL.

	Fresnel Sine Integral in $d = 3$	
Method	`OneTheta`	`MultiTheta`
Absolute Error	0.000 23	0.063 00
n	4100	260
Time (seconds)	0.0270	0.0230

The results are summarized from the two different approaches in Table 7.3. The first method, called `OneTheta`, uses common shape parameter across all the

dimensions, whereas the second method, called `MultiTheta`, allows the shape parameters to vary across the dimensions. In the `MultiTheta` method, the shape parameter search is multi-variate so the magnitude of shape parameter depends on the integrand's magnitude in each dimensions. We have chosen a integrand particularly to demonstrate this aspect (7.2) where we used $d = 3$ and the constants $\boldsymbol{\upsilon} = (10^{-4}, 1, 10^{4})$. The choice of magnitude variations in constants $\boldsymbol{\upsilon}$ allows to make the integrand vary significantly across dimensions.

We ran this test for 1000 times. In comparison, both the methods successfully computed integral all the times but the `MultiTheta` is slightly faster. The `MultiTheta` method used less number of samples n but the integration error is bigger than the `OneTheta`. For the same size of n number of samples, `OneTheta` method will be much faster since the shape parameter search is easier. The `MultiTheta` method is useful in scenarios when we want to use smaller size n and the integrand varies significantly across dimensions.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

We have developed a fast, automatic Bayesian cubature that estimates the high dimensional integral within a user defined error tolerance that occur in many scientific computing such as finance, machine learning, or imaging, etc. The stopping criteria arise from assuming the integrand to be a Gaussian process. In Section 2.2, we developed three versions: empirical Bayes, full Bayes, and generalized cross-validation. Empirical-Bayes uses maximum-likelihood to optimally choose the parameters, where posterior of the parameters given the integrand values is maximized. Alternatively, full-Bayes assumes non-informative prior on the parameters and then computes posterior distribution of the integral μ, which leads to a t-distribution to obtain the parameters. Generalized cross-validation extends the concept of cross-validation to construct an objective which in turn maximized.

The computational cost of the automatic Bayesian cubature can be dramatically reduced if the covariance kernel matches the nodes. We have demonstrated two such matches in practice. The first algorithm was based on rank-1 lattice nodes and shift-invariant kernels where the matrix-vector multiplications can be accomplished using the fast Fourier Transform. The second algorithm was based on Sobol' points with first order Walsh kernel where the matrix-vector multiplications can be accomplished using the fast Walsh transform. Three integration problems illustrate the performance of our automatic Bayesian cubature.

For faster computations one could use fixed order kernels in `cubBayesLattice_g`, but for more advanced usage, we have added a kernel variation in Section 4.3 that allows one to optimally choose the kernel order without the constraint of being

even integer.

During the numerical experiments we noticed a computation step that causes inaccuracy due to a cancellation error in the estimation of stopping criterion. We have developed a novel technique in Section 6.1, to overcome this cancellation error using the inherent structure of the shift invariant kernel used in our algorithm.

In Section 3.5.1, we have analytically computed the gradient of the objective function and the shift invariant kernel to use with steepest descent in kernel parameters search. Quasi-Monte Carlo cubature methods are efficient [44] even if the dimension is high given the effective dimension is low. To take advantage of low effective dimension, one should not fix the kernel shape parameter across all the dimensions. In this situation steepest descent method come in handy as one search parameters in multi-dimensions.

8.2 Future Work

We demonstrated the capability our new Bayesian cubature algorithms to successfully compute the integrals faster within the user defined error tolerances. But there are few potential improvements and new areas of applications. Some of the improvement ideas are listed here:

- Higher order digital sequences and digital shift and/or scramble invariant kernels [40] [45]: We could improve the computation speed of `cubBayesNet_g` for smoother integrands using higher order digital sequences and matching kernels which have the potential of being another match that satisfies the conditions in Section 3. The fast transform would correspond to a fast Walsh transform similar to the second algorithm we demonstrated. For such kernels and the first order Walsh kernel we demonstrated, periodicity is not assumed, however, special structure of both the sequences and the kernels are required to take

advantage of integrand smoothness.

- Control variates: Hickernell et.al [12] [46] adapted control variates for Quasi-Montro. Control variates are commonly used to improve the efficiency of IID Monte Carlo integration. One should be able to adapt our Bayesian cubature to control variates, i.e., assuming

$$f = \mathcal{GP}\left(\beta_0 + \beta_1\, g_1 + \cdots + \beta_p\, g_p,\ s^2 C\right),$$

for some choice of vector of functions $\boldsymbol{g} = \{g_1, \ldots, g_p\}$, where $\boldsymbol{g} : [0,1)^d \to \mathbb{R}^p$ whose integrals are known $\mu_{\boldsymbol{g}} := \int_{[0,1)^d} \boldsymbol{g}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$, and some parameters $\beta_0, \ldots, \beta_p$ in addition to the s and C, then

$$\mu := \int_{[0,1)^d} f(\boldsymbol{x}) \mathrm{d}x = \int_{[0,1)^d} h_\beta(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \text{ where } h_\beta(\boldsymbol{x}) := f(\boldsymbol{x}) + \boldsymbol{\beta}^T(\mu_{\boldsymbol{g}} - \boldsymbol{g}(\boldsymbol{x})).$$

Here $\boldsymbol{g}$ are function on which the QMC method does a good job, integrating it without error. The goal is to choose an optimal $\boldsymbol{\beta}$ to make

$$\widehat{\mu}_{\boldsymbol{\beta},n} := \frac{1}{n} \sum_{i=0}^{n-1} h_{\boldsymbol{\beta}}(\boldsymbol{x}_i)$$

sufficiently close to μ with the least expense, n, possible. The efficacy of this approach has not yet been explored.

- Steepest descent: The kernels's optimal shape parameter searched using steepest descent with kernels gradient could sometime get into local minima. This needs more understanding and enhancements. JR: explain why, any suggestions?

- Gaussian diagnosis: We assumed the integrand an instance of a Gaussian process. Is there a way to prove that is a good assumption based on the results we have?

- Parallel Algorithm: For more demanding high performance computing applications where the precision requirements are high, our algorithms will try to use

large number samples n leading to longer computation times. One approach to overcome these constraints is to use Parallel computing techniques to speedup the algorithm. Most time consuming parts of our algorithm are shape parameter search and Fast transform computation. Fast Fourier transform (FFT) and Fast Walsh transform are easily amenable to parallelization, there exist plenty of prior work that can be adapted to work with our algorithms. We use radix-2 FFT, one could use a higher radix FFT to make the computations faster.

Another area of improvement is the parameter search, We explored the steepest descent algorithm but the speedup was not significant. One could explore higher order algorithms such as Newton method which could find the minima faster. Fast transforms are repeatedly computed in every step of the parameter search if it can be avoided by interpolation or other techniques could significantly speedup the algorithm.

One could also GPU to run the whole code of our Bayesian Cubature algorithms or just the FFT/FWHT part of it to get a easier speedup.

BIBLIOGRAPHY

[1] P. Glasserman, *Monte Carlo Methods in Financial Engineering*, ser. Applications of Mathematics. New York: Springer-Verlag, 2004, vol. 53.

[2] A. Keller, "Quasi-Monte Carlo image synthesis in a nutshell," in *Monte Carlo and Quasi-Monte Carlo Methods 2012*, ser. Springer Proceedings in Mathematics and Statistics, J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, Eds., vol. 65. Springer Berlin Heidelberg, 2013, pp. 213–249.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] F.-X. Briol, C. J. Oates, M. Girolami, M. A. Osborne, and D. Sejdinovic, "Probabilistic integration: A role in statistical computation?" *Statist. Sci.*, 2018+, to appear.

[5] P. Diaconis, "Bayesian numerical analysis," in *Statistical Decision Theory and Related Topics IV, Papers from the 4th Purdue Symp., West Lafayette, Indiana 1986*, S. S. Gupta and J. O. Berger, Eds. Springer-Verlag, New York, 1988, vol. 1, pp. 163–175.

[6] A. O'Hagan, "Bayes-Hermite quadrature," *J. Statist. Plann. Inference*, vol. 29, pp. 245–260, 1991.

[7] K. Ritter, *Average-Case Analysis of Numerical Problems*, ser. Lecture Notes in Mathematics. Berlin: Springer-Verlag, 2000, vol. 1733.

[8] C. E. Rasmussen and Z. Ghahramani, "Bayesian Monte Carlo," in *Advances in Neural Information Processing Systems*, S. Thrun, L. K. Saul, and K. Obermayer, Eds. MIT Press, 2003, vol. 15, pp. 489–496.

[9] F. J. Hickernell, "The trio identity for quasi-Monte Carlo error analysis," in *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Stanford, USA, August 2016*, ser. Springer Proceedings in Mathematics and Statistics, P. Glynn and A. Owen, Eds. Springer-Verlag, Berlin, 2018, pp. 13–37, arXiv:1702.01487.

[10] F. J. Hickernelland Ll. A. Jiménez Rugama, "Reliable adaptive cubature using digital sequences," in *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, ser. Springer Proceedings in Mathematics and Statistics, R. Cools and D. Nuyens, Eds., vol. 163. Springer-Verlag, Berlin, 2016, pp. 367–383, arXiv:1410.8615 [math.NA].

[11] Ll. A. Jiménez Rugama and F. J. Hickernell, "Adaptive multidimensional integration based on rank-1 lattices," in *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, ser. Springer Proceedings in Mathematics and Statistics, R. Cools and D. Nuyens, Eds., vol. 163. Springer-Verlag, Berlin, 2016, pp. 407–422, arXiv:1411.1966.

[12] F. J. Hickernell, Ll. A. Jiménez Rugama, and D. Li, "Adaptive quasi-Monte Carlo methods for cubature," in *Contemporary Computational Mathematics — a celebration of the 80th birthday of Ian Sloan*, J. Dick, F. Y. Kuo, and H. Woźniakowski, Eds. Springer-Verlag, 2018, pp. 597–619.

[13] A. OHagan, "Bayes-hermite quadrature," *Journal of Statistical Planning and Inference*, vol. 29(3), p. 245260, 1991.

[14] C. E. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts: MIT Press, 2006, (online version at `http://www.gaussianprocess.org/gpml/`).

[15] R. Brent, *Algorithms for Minimization Without Derivatives*. Prentice-Hall, 1973.

[16] G. Forsythe, M. Malcolm, and C. Moler, *Computer methods for mathematical computations*. Prentice-Hall, 1976.

[17] K. Dong, D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson, "Scalable log determinants for gaussian process kernel learning," *NIPS*, 2017, in press.

[18] P. Craven and G. Wahba, "Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation," *Numer. Math.*, vol. 31, pp. 307–403, 1979.

[19] G. H. Golub, M. Heath, and G. Wahba, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics*, vol. 21, pp. 215–223, 1979.

[20] G. Wahba, *Spline Models for Observational Data*, ser. CBMS-NSF Regional Conference Series in Applied Mathematics. Philadelphia: SIAM, 1990, vol. 59.

[21] A. Genz, "Comparison of methods for the computation of multivariate normal probabilities," *Computing Science and Statistics*, vol. 25, pp. 400–405, 1993.

[22] J. Dick, F. Kuo, and I. H. Sloan, "High dimensional integration — the Quasi-Monte Carlo way," *Acta Numer.*, vol. 22, pp. 133–288, 2013.

[23] J. Dick and F. Pillichshammer, *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge: Cambridge University Press, 2010.

[24] N. J. Higham, *Functions of matrices: theory and computation*. SIAM, 2008.

[25] F. J. Hickernelland H. Niederreiter, "The existence of good extensible rank-1 lattices," *J. Complexity*, vol. 19, pp. 286–300, 2003.

[26] F. J. Hickernell, "Quadrature error bounds with applications to lattice rules," *SIAM J. Numer. Anal.*, vol. 33, pp. 1995–2016, 1996, corrected printing of Sections 3-6 in ibid., **34** (1997), 853–866.

[27] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, C. W. Clark, and A. B. O. Dalhuis, "Digital library of mathematical functions," 2018. [Online]. Available: http://dlmf.nist.gov/

[28] S.-C. T. Choi, Y. Ding, F. J. Hickernell, L. Jiang, Ll. A. Jiménez Rugama, D. Li, R. Jagadeeswaran, X. Tong, K. Zhang, Y. Zhang, and X. Zhou, "GAIL: Guaranteed Automatic Integration Library (versions 1.0–2.2)," MATLAB software, 2013–2017. [Online]. Available: http://gailgithub.github.io/GAIL_Dev/

[29] D. Nuyens. [Online]. Available: https://people.cs.kuleuven.be/~dirk.nuyens/qmc-generators/

[30] A. Sidi, "Further extension of a class of periodizing variable transformations for numerical integration," *J. Comput. Appl. Math.*, vol. 221, pp. 132–149, 2008.

[31] I. M. Sobol', "The distribution of points in a cube and the approximate evaluation of integrals," *U.S.S.R. Comput. Math. and Math. Phys.*, vol. 7, pp. 86–112, 1967.

[32] H. Niederreiter, "Constructions of (t, m, s)-nets and (t, s)-sequences," *Finite Fields Appl.*, vol. 11, pp. 578–600, 2005.

[33] J. F. Baldeaux, "Higher order nets and sequences," Ph.D. dissertation, The School of Mathematics and Statistics at The University of New South Wales, June 2010.

[34] I. M. Sobol', "Uniformly distributed sequences with an additional uniformity property," *Zh. Vychisl. Mat. i Mat. Fiz.*, vol. 16, pp. 1332–1337, 1976.

[35] F. J. Hickernelland R. X. Yue, "The mean square discrepancy of scrambled (t, s)-sequences," *SIAM J. Numer. Anal.*, vol. 38, pp. 1089–1112, 2000.

[36] A. B. Owen, "Randomly permuted (t, m, s)-nets and (t, s)-sequences," pp. 299–317.

[37] F. Y. Kuo and D. Nuyens, "Application of quasi-monte carlo methods to elliptic pdes with random diffusion coefficients a survey of analysis and implementation," *Foundations of Computational Mathematics*, vol. 16(6), pp. 1631–1696, 2016.

[38] D. Nuyens. [Online]. Available: https://people.cs.kuleuven.be/~dirk.nuyens/

[39] H. S. Hong and F. J. Hickernell, "Algorithm 823: Implementing scrambled digital nets," *ACM Trans. Math. Software*, vol. 29, pp. 95–109, 2003.

[40] D. Nuyens, "The construction of good lattice rules and polynomial lattice rules," 08 2013.

[41] P. Bratley and B. L. Fox, "Algorithm 659: Implementing Sobol's quasirandom sequence generator," *ACM Trans. Math. Software*, vol. 14, pp. 88–100, 1988.

[42] J. Dick, "Walsh spaces containing smooth functions an quasi-monte carlo rules of arbitrary high order," *SIAM J. Numer. Anal.*, vol. 46, no. 1519–1553, 2008.

[43] B. D. Keister, "Multidimensional quadrature algorithms," *Computers in Physics*, vol. 10, pp. 119–122, 1996.

[44] I. H. Sloan and H. Woźniakowski, "When are quasi-Monte Carlo algorithms efficient for high dimensional integrals?" *J. Complexity*, vol. 14, pp. 1–33, 1998.

[45] G. L. D. N. F. P. J. Baldeaux, J. Dick, "Efficient calculation of the worst-case error and (fast) component-by-component construction of higher order polynomial lattice rules," *Numerical Algorithms*, vol. 59, pp. 403–431, Mar. 2012.

[46] D. Li, "Reliable quasi-Monte Carlo with control variates," Master's thesis, Illinois Institute of Technology, 2016.

[47] R. Cools and D. Nuyens, Eds., *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, ser. Springer Proceedings in Mathematics and Statistics, vol. 163. Springer-Verlag, Berlin, 2016.