**Automatic Bayesian Cubature manuscript No.**
(will be inserted by the editor)

# Fast Automatic Bayesian Cubature Using Sobol Sampling

**R. Jagadeeswaran · Fred J. Hickernell**

Received: date / Accepted: date

**Abstract** Automatic cubatures approximate integrals to user-specified error tolerances. For high dimensional problems, it is difficult to adaptively change the sampling pattern, but one can automatically determine the sample size, $n$, given a reasonable, fixed sampling pattern. We take this approach here using a Bayesian perspective. In **?**, faster Bayesian cubature was introduced and demonstrated using rank-1 lattice points. However it requires the integrands to be periodic, here we propose an extension to overcome that limitation using digital nets and digitally shift invariant kernels. As the Gaussian assumption is fundamental to our work, We also validate the assumption using a simple diagnostics. Another constraint imposed in **?** was the necessity to have kernel order integer valued, here we provide an alternative to have the kernel order continuous valued. We also demonstrate the hyperparameters can be searched using gradient descent instead of heuristic search.

Our algorithm is implemented in the Guaranteed Automatic Integration Library (GAIL).

**Keywords** Bayesian cubature · Fast automatic cubature · GAIL · Probabilistic numeric methods

## 1 Introduction

JR: discuss prior work

R. Jagadeeswaran
Department of Applied Mathematics,
Illinois Institute of Technology
10 W. 32nd St., Room 208, Chicago IL 60616
E-mail: jrathin1@iit.edu

Fred J. Hickernell
Center for Interdisciplinary Scientific Computation and
Department of Applied Mathematics
Illinois Institute of Technology
10 W. 32nd St., Room 208, Chicago IL 60616
E-mail: hickernell@iit.edu

Cubature, or numerical multivariate integration, 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 $\mu$ has no closed form analytic expression. Typically, $g$ is accessible through a black-box function routine. Cubature is a key component of many problems in scientific computing, finance Glasserman [2004], statistical modeling, imaging Keller [2013], uncertainty quantification, and machine learning Goodfellow et al. [2016]. This article extends the fast Bayesian cubature ideas presented in Jagadeeswaran and Hickernell [2019].

After a suitable variable transformation, the integral may often be expressed as

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

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) is addressed in Beckers and Haegemans [1992], Sidi [2008, 1993], Laurie [1996], Cristea et al. [2007]. Our goal is to construct a cubature, $\widehat{\mu} = \widehat{\mu}(f)$, depending only on integrand values, which satisfies the error criterion

$$|\mu - \widehat{\mu}| \le \varepsilon. \tag{2}$$

Following the Bayesian numerics approach of Diaconis [1988], O'Hagan [1991], Rasmussen and Ghahramani Rasmussen and Williams [2003], Briol et al. [2019], and others, we assume that our integrand is an instance of a Gaussian process and use a credible interval for $\mu$ to construct $\widehat{\mu}$. The integrand is sampled until the credible interval becomes small enough to satisfy with high probability. See Jagadeeswaran and Hickernell [2019] for the details of our approach.

The cubature may be an affine function of integrand values:

$$\widehat{\mu} := \widehat{\mu}(f) := w_0 + \sum_{i=1}^{n} f(\boldsymbol{x}_i) w_i, \quad \mathcal{P} := \{\boldsymbol{x}_i\}_{i=1}^n \subset [0,1]^d \tag{3}$$

where the weights, $w_0$, and $\boldsymbol{w} = (w_i)_{i=1}^n \in \mathbb{R}^n$, and the nodes, $\mathcal{P}$, 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. This research focuses on multivariate numerical integrals where the computational cost is a bottleneck.

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 $\varepsilon$, i.e.,

$$|\mu - \widehat{\mu}| \le \varepsilon. \tag{4}$$

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. Briol et al. [2019], Diaconis Diaconis [1988], O'Hagan O'Hagan [1991], Ritter Ritter [2000], Rasmussen and Ghahramani Rasmussen and Williams [2003], 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.

The primary contribution of this work is to extend the fast Bayesian cubature **?** to use digital nets and Walsh kernels. Additional enhancements also provided to have the ability to search the hyperparameters using steepest descent and to

use continuous valued kernel order. We also provide a diagnostics to verify the Gaussian process assumption that was made in developing the Bayesian cubature.

**?** introduces Bayesian cubature and develops the fast Bayesian transform with certain assumptions. The fast Bayesian transform helps to speedup the computations. Using rank-1 lattice points and shift-invariant kernels demonstrates the fast Bayesian cubature.

Hickernell Hickernell [2018] 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 (4). 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 Hickernell and Jiménez Rugama [2016], Jiménez Rugama and Hickernell [2016] have developed stopping criteria for cubature rules based on low discrepancy nodes by tracking the decay of the discrete Walsh coefficients of the integrand. The algorithms proposed here also rely on discrete Walsh coefficients, but in a different way. We only discuss automatic Bayesian cubature for absolute error tolerances in this thesis. The recent work by Hickernell, Jiménez-Rugama, and Li Hickernell et al. [2018] suggests how one might accommodate more general error criteria, such as relative error tolerances which has been adapted in the MATLAB implementation of our algorithms.

Section 2 explains the Bayesian approach to calculate the posterior cubature error and defines our automatic Bayesian cubature. Although much of this material is known, it is included for completeness. We also briefly introduce foundational concepts such as the fast Bayesian transform and the confidence intervals that will be basis of our main contributions. Section 3 provides enhancements to the fast Bayesian cubature introduced in **?**. Section 4 provides a diagnostics using Q-Q plots to verify the Gaussian process assumption used in developing the Bayesian cubature. Section 5 demonstrates our new implementation of matching nodes and kernel using Sobol' points and Walsh kernels. It also shows that the fast Walsh Hadamard as the fast Bayesian transform for this case. Numerical examples are provided in Section 6 to demonstrate the performance and advantages of our new algorithms. We conclude with a brief discussion and potential future work in Section 7.

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, designs, and data-sites interchangeably to denote the points $\mathcal{P}$ used in the cubature.

## 2 Bayesian Cubature

JR: Briefly explain and cite our BayesLattice paper

1. automatic Bayesian cubature
2. fast automatic Bayesian cubature

The Bayesian approach for numerical analysis was popularized by Diaconis [1988]. 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 a Brownian motion in the sample space $\mathcal{C}[0,1)$, the space of continuous functions.

This research is focused on the Bayesian approach for numerical integration that is known as Bayesian cubature as introduced by O'Hagan O'Hagan [1991]. Bayesian cubature returns a probability distribution, that expresses belief about the true value of integral, $\mu(f)$. This posterior probability distribution is based on a prior that depends on $f$, which is computed via Bayes' rule using the *data* contained in the function evaluations Briol et al. [2019]. 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 $\boldsymbol{\theta}$:

$$\mathsf{C}^T = \mathsf{C}, \quad \boldsymbol{a}^T \mathsf{C} \boldsymbol{a} > 0, \quad \text{where } \mathsf{C} = (C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1}^n, \\ \text{for all } \boldsymbol{a} \neq 0,\ n \in \mathbb{N},\ \text{distinct } \boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in [0,1]^d. \quad (5)$$

The covariance function, $C$, and the Gram matrix, $\mathsf{C}$, depend implicitly on $\boldsymbol{\theta}$, but the notation may omit this for simplicity's sake. 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

$$\boldsymbol{f} \sim \mathcal{N}(m\mathbf{1}, s^2 \mathsf{C}), \quad (6a)$$

$$\mu \sim \mathcal{N}(m, s^2 c_0), \quad (6b)$$

$$\text{where } c_0 := \int_{[0,1]^d \times [0,1]^d} C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{t}, \quad (6c)$$

$$\operatorname{cov}(\boldsymbol{f}, \mu) = \left( \int_{[0,1]^d} C(\boldsymbol{t}, \boldsymbol{x}_i) \, \mathrm{d}\boldsymbol{t} \right)_{i=1}^n =: \boldsymbol{c}. \quad (6d)$$

Here, $c_0$ 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.

It follows from Lemma 1 in **?** that the *conditional* distribution of the integral, given observed function values, $\boldsymbol{f} = \boldsymbol{y}$, is also Gaussian:

$$\mu | (\boldsymbol{f} = \boldsymbol{y}) \sim \mathcal{N}\big(m(1 - \boldsymbol{c}^T \mathsf{C}^{-1} \mathbf{1}) + \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{y}, \ \ s^2 (c_0 - \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{c})\big). \quad (7)$$

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

$$\widehat{\mu} | (\boldsymbol{f} = \boldsymbol{y}) = m(1 - \mathbf{1}^T \mathsf{C}^{-1} \boldsymbol{c}) + \boldsymbol{c}^T \mathsf{C}^{-1} \boldsymbol{y}, \quad (8)$$

which takes the form of (3). 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

$$\mathbb{P}_f\left[|\mu - \widehat{\mu}| \le \mathrm{err}_{\mathrm{CI}}\right] = 99\%, \tag{9a}$$

$$\mathrm{err}_{\mathrm{CI}} = 2.58 s \sqrt{c_0 - \boldsymbol{c}^T \mathsf{C}^{-1}\boldsymbol{c}}. \tag{9b}$$

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

Any Bayesian cubature algorithm based on (9) has computational cost that is $\mathcal{O}(n^3)$. As shown in the numerical experiments **?**, the cost increases quickly as the $n$ required to meet the error tolerance increases. This motivates the fast Bayesian cubature algorithm presented in Section 2.2.

## 2.2 Fast Bayesian Transform Kernel

As introduced in **?**, fast Bayesian transform kernel is defined with the following assumptions:

$$\begin{aligned}
\mathsf{C} = \mathsf{C}_{\boldsymbol{\theta}} &= \left(C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)_{i,j=1}^{n} = (\boldsymbol{C}_1, ..., \boldsymbol{C}_n) \\
&= \frac{1}{n}\mathsf{V}\Lambda\mathsf{V}^H, \qquad \mathsf{V}^H = n\mathsf{V}^{-1}, \\
\mathsf{V} &= (\boldsymbol{v}_1, ..., \boldsymbol{v}_n)^T = (\boldsymbol{V}_1, ..., \boldsymbol{V}_n) \\
\mathsf{C}^p &= \frac{1}{n}\mathsf{V}\Lambda^p\mathsf{V}^H, \qquad \forall p \in \mathbb{Z},
\end{aligned} \tag{10}$$

where $\mathsf{V}^H$ is the Hermitian of $\mathsf{V}$, $\boldsymbol{C}_1, \cdots, \boldsymbol{C}_n$ are columns of $\mathsf{C}$, $\boldsymbol{V}_1, \cdots, \boldsymbol{V}_n$ are columns of $\mathsf{V}$, and $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ are rows of $\mathsf{V}$. The columns of matrix $\mathsf{V}$ are eigenvectors of $\mathsf{C}$, and $\Lambda$ is a diagonal matrix of eigenvalues of $\mathsf{C}$. In this and later sections, we drop the notation of $\boldsymbol{\theta}$'s dependence of various quantities for simplicity of notation. The normalization of $\mathsf{V}$ assumed in (10) conveniently allows the first eigenvector, $\boldsymbol{V}_1$, to be the vector of ones in (11b) below. 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 $\widetilde{\boldsymbol{b}}$'s fast computation:

$$\mathsf{V} \text{ may be identified analytically,} \tag{11a}$$

$$\boldsymbol{v}_1 = \boldsymbol{V}_1 = \boldsymbol{1}, \tag{11b}$$

$$\text{Computing } \mathsf{V}^H\boldsymbol{b} \text{ requires only } \mathcal{O}(n \log n) \text{ operations } \forall \boldsymbol{b}. \tag{11c}$$

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* for the matching nodes $\{\boldsymbol{x}_i\}_{i=1}^{\infty}$.

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

$$\int_{[0,1]^d} C(\boldsymbol{t}, \boldsymbol{x})\, \mathrm{d}\boldsymbol{t} = 1 \qquad \forall \boldsymbol{x} \in [0,1]^d, \tag{12}$$

which implies that $c_0 = 1$ (6c) and $\boldsymbol{c} = \boldsymbol{1}$ (6d).

## 2.3 Hyperparameter search

JR: This section explains hyperparameter estimation using gradient descent

The credible interval in (9) 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 $\boldsymbol{\theta}$. The $\mathrm{err}_{\mathrm{CI}}$ depends on the integrand values, which becomes explicit once the hyperparameters, $m, s$, and $\boldsymbol{\theta}$, are inferred from integrand data.

There were three methods developed for hyperparameter estimation in **?** as summarized below. With the assumptions for a fast Bayesian transform kernel, the computation cost including the hyperparameter estimation can be significantly reduced.

**Theorem 1** *There are at least three approaches to estimating or integrating out the hyperparameters defining a Gaussian process from which the integrand is drawn: empirical Bayes, full Bayes, and generalized cross-validation. Under assumptions* (11) *and* (12)*, the parameters and credible interval half-widths 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:*

$$
\begin{aligned}
m_{\mathrm{EB}} = m_{\mathrm{full}} = m_{\mathrm{GCV}} &= \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^n y_i,\\
s_{\mathrm{EB}}^2 &= \frac{1}{n^2}\sum_{i=2}^n \frac{|\widetilde{y}_i|^2}{\lambda_i},\\
s_{\mathrm{GCV}}^2 &= \frac{1}{n}\sum_{i=2}^n \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\left[\sum_{i=1}^n \frac{1}{\lambda_i}\right]^{-1},\\
\widehat{\sigma}_{\mathrm{full}}^2 &= \frac{1}{n(n-1)}\sum_{i=2}^n \frac{|\widetilde{y}_i|^2}{\lambda_i}\left(\frac{\lambda_1}{n} - 1\right),\\
\boldsymbol{\theta}_{\mathrm{EB}} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^n \frac{|\widetilde{y}_i|^2}{\lambda_i}\right) + \frac{1}{n}\sum_{i=1}^n \log(\lambda_i)\right], \qquad (14a)\\
\boldsymbol{\theta}_{\mathrm{GCV}} &= \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\log\left(\sum_{i=2}^n \frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right) - 2\log\left(\sum_{i=1}^n \frac{1}{\lambda_i}\right)\right], \qquad (14b)\\
\widehat{\mu}_{\mathrm{EB}} = \widehat{\mu}_{\mathrm{full}} = \widehat{\mu}_{\mathrm{GCV}} &= \frac{\widetilde{y}_1}{n} = \frac{1}{n}\sum_{i=1}^n y_i, \qquad (15)
\end{aligned}
$$

*and so $\widehat{\mu}$ is simply the sample mean. Also the credible interval half-widths, $\mathrm{err}_{\mathrm{CI}}$, are given by*

$$
\begin{aligned}
\mathrm{err}_{\mathsf{x}} &= 2.58 s_{\mathsf{x}} \sqrt{1 - \frac{n}{\lambda_1}}, \qquad \mathsf{x} \in \{\mathrm{EB}, \mathrm{GCV}\}, \qquad (16a)\\
\mathrm{err}_{\mathrm{full}} &= t_{n-1,0.995}\widehat{\sigma}_{\mathrm{full}} > \mathrm{err}_{\mathrm{EB}}. \qquad (16b)
\end{aligned}
$$

*The resulting credible intervals are then*

$$
\mathbb{P}_f\left[|\mu - \widehat{\mu}_{\mathsf{x}}| \le \mathrm{err}_{\mathsf{x}}\right] = 99\%, \qquad \mathsf{x} \in \{\mathrm{EB}, \mathrm{full}, \mathrm{GCV}\}. \qquad (17)
$$

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

Equation (17) presents three credible intervals, for the $\mu$, the desired integral. Each credible interval is based on different assumptions about the hyperparameters $m$, $s$, and $\boldsymbol{\theta}$. 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 as explained further in Section 3.1. We apply these results in Section 2.4 to speedup the computations.

### 2.4 The Fast Automatic Bayesian Cubature Algorithm

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; these 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 (Section 3.1) and not the extremes, we expect our automatic Bayesian cubature to approximate the integral within the desired error tolerance with high probability. The examples in Section 6 support that expectation.

**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 $n_{\max}$

$n \leftarrow n_0$, $n' \leftarrow 0$, err $\leftarrow \infty$
**while** err $> \varepsilon$ and $n \leq n_{\max}$ **do**
  Generate $\{\boldsymbol{x}_i\}_{i=n'+1}^{n}$ and sample $\{f(\boldsymbol{x}_i)\}_{i=n'+1}^{n}$
  Compute $\boldsymbol{\theta}$ by (14a) or (14b)
  Compute err according to (16a) or (16b)
  $n' \leftarrow n$, $n \leftarrow 2n'$
**end while**
Sample size to compute $\widehat{\mu}$, $n \leftarrow n'$
Compute $\widehat{\mu}$, the approximate integral, according to (15)
**return** $\widehat{\mu}$, $n$ and err

The results in **?** provide techniques to speedup the computation, but did not provide any explicit technique for parameter search. We provide one such approach using steepest descent in the Section 3.2.

## 3 Enhancements to Bayesian Cubature

This section covers enhancements developed to the Bayesian Cubature proposed in **?**. First, we start with explaining the underlying function space of integrands as desired in Section 2.4.

### 3.1 Cone of Functions and the Credible interval

Bayesian cubature assumes that 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. Suppose that

$$|\mu(f) - \widehat{\mu}_n(f)| \le \mathrm{err}_{\mathrm{CI}}(f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)) \tag{18}$$

for some $f$, which it is 99% of the time under our hypothesis. Also note that our $\mathrm{err}_{\mathrm{CI}}$ (16a) (16b) are positively homogeneous functions, meaning,

$$\mathrm{err}_{\mathrm{CI}}(ay_1, \cdots, ay_n) = |a| \, \mathrm{err}_{\mathrm{CI}}(y_1, \cdots, y_n).$$

One can verify the homogeneity of (16a) and (16b) easily. Thus if $f$ satisfies (18), then

$$\begin{aligned} |\mu(af) - \widehat{\mu}_n(af)| &= |a| \, |\mu(f) - \widehat{\mu}_n(f)| \\ &\le |a| \, \mathrm{err}_{\mathrm{CI}}(f(\boldsymbol{x}_1), \cdots, f(\boldsymbol{x}_n)) \\ &= \mathrm{err}_{\mathrm{CI}}(af(\boldsymbol{x}_1), \cdots, af(\boldsymbol{x}_n)) \end{aligned}$$

for all real $a$. Thus the set of all $f$ satisfying (18) is a *cone*, $\mathscr{C}$. Cones of functions satisfy the property that if $f \in \mathscr{C}$ then $af \in \mathscr{C}$.

In the context of Bayesian cubature, one can explain the cone concept beginning with the definition of credible interval (9). Let $f \sim \mathcal{GP}$, be an instance of a Gaussian stochastic process:

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

This can be interpreted as $|\mu(f) - \widehat{\mu}_n(f)| \le \mathrm{err}_{\mathrm{CI}}(f)$ with 99% confidence. If $f$ is in the 99% middle of the sample space with $f(\boldsymbol{x}_i) = y_i$ then $af$ is also in the middle 99% of the sample space with $af(\boldsymbol{x}_i) = ay_i$.

We demonstrate the credible interval using the following example. For this purpose, choose a smooth and periodic integrand $f_{\mathrm{smooth}}(\boldsymbol{x}) = \exp(\sum_{\ell=1}^{d} \cos(2\pi x_\ell))$ and another integrand $f_{\mathrm{peaky}}(\boldsymbol{x}) = f_{\mathrm{smooth}} + a_{\mathrm{peaky}} f_{\mathrm{noise}}$ where $a_{\mathrm{peaky}} \in \mathbb{R}$. Here $f_{\mathrm{noise}}(\boldsymbol{x}) = (1 - \exp(2\pi\sqrt{-1}\boldsymbol{x}^T\boldsymbol{\zeta}))$, $\boldsymbol{\zeta} \in \mathbb{R}^d$ is some $d$-dimensional vector belonging to the dual space of the lattice nodes for some $\{\boldsymbol{x}_i\}_{i=1}^n$. The $\boldsymbol{\zeta}$ in the dual space of lattice nodes implies that $f_{\mathrm{noise}}(\boldsymbol{x}_i) = 0$ at the sampling nodes $\{\boldsymbol{x}_i\}_{i=1}^n$. The $f_{\mathrm{nice}}$ is obtained by kernel interpolation of the $n$ samples of $f_{\mathrm{smooth}}$ at $\{\boldsymbol{x}_i\}_{i=1}^n$. We chose the Matérn kernel (**??**) for the interpolation. Please note that $f_{\mathrm{peaky}}(\boldsymbol{x}_i) = f_{\mathrm{nice}}(\boldsymbol{x}_i) = f_{\mathrm{smooth}}(\boldsymbol{x}_i)$ for $i = 1, \cdots, n$.

In Figure 1, the sampled function values are shown as dots. One can imagine these samples were obtained from $f_{\mathrm{nice}}$, a moderately smoother function or from $f_{\mathrm{peaky}}$, a highly oscillating function. In this example, we used $a_{\mathrm{peaky}} = 2$.

When using $n = 16$ rank-1 lattice points, and $r = 1$ shift-invariant kernel, we get the posterior distribution of $\mu$ as shown in Figure 2. The true integral value is shown as $\mu_{\mathrm{smooth}}$ which is at the center of the plot. The integral of the peaky function $f_{\mathrm{peaky}}$ lies outside of the 99% of the credible interval given by (**??**), whereas the $\mu_{\mathrm{nice}}$ falls within.

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

### 3.2 Gradient descent to find optimal shape parameter

The equation specifying $\boldsymbol{\theta}_{\mathrm{EB}}$ as defined in (14a) does not say how the parameter search can be done. However there exist empirical algorithms Brent [1973],

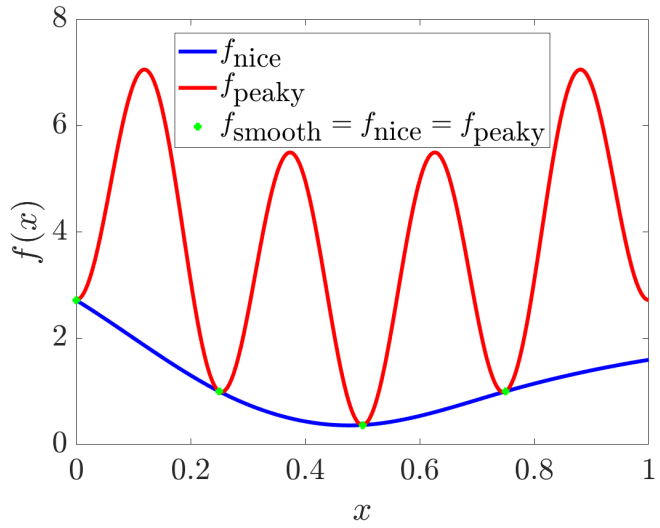


**Fig. 1** Example integrands 1) $f_{\text{nice}}$, a smooth function, 2) $f_{\text{peaky}}$, a peaky function. The function values $f_{\text{peaky}}(\boldsymbol{x}_i) = f_{\text{nice}}(\boldsymbol{x}_i) = f_{\text{smooth}}(\boldsymbol{x}_i)$ for $i = 1, \cdots, n$. This plot can be conditionally reproduced using `DemoCone.m`

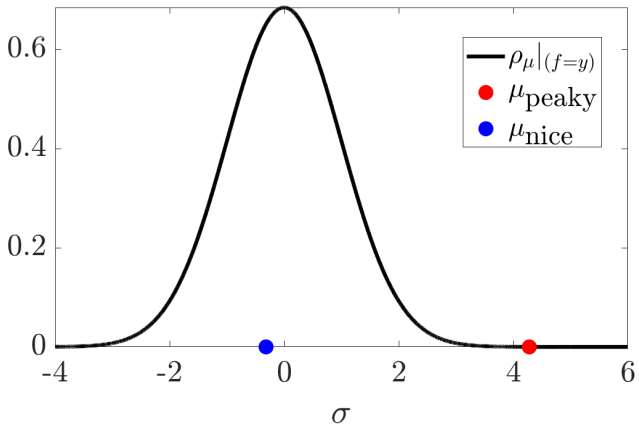


**Fig. 2** Probability distributions showing the relative integral position of a smooth and a peaky function. $f_{\text{nice}}$ lies within the center 99% of the confidence interval, and $f_{\text{peaky}}$ lies on the outside of 99% of the confidence interval. This plot can be conditionally reproduced using `DemoCone.m`

Forsythe et al. [1976] that one could use to accomplish the same. Since the objective function is known we could compute the gradient. Using the gradient of $l(s, m, \boldsymbol{\theta}|\boldsymbol{y})$, one can apply optimization techniques such as gradient descent to find the optimal value faster. Let us define the objective function to guide the search by excluding the negative sign, which modifies the problem to become a

minimization of

$$\mathcal{L}_{\mathrm{EB}}(\boldsymbol{\theta}|\boldsymbol{y}) := \frac{1}{n}\sum_{i=1}^{n}\log(\lambda_i) + \log\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i}\right) + \text{constants}.$$

Taking derivative with respect to $\theta_\ell$, for $\ell = 1, \cdots, d$

$$\frac{\partial}{\partial\theta_\ell}\mathcal{L}_{\mathrm{EB}}(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}\frac{\bar{\lambda}_{i(\ell)}}{\lambda_i} - \left(\sum_{i=2}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2\,\bar{\lambda}_{i(\ell)}}{\lambda_i^2}\right)\left(\sum_{i=2}^{n}\frac{|\tilde{\boldsymbol{y}}_i|^2}{\lambda_\ell}\right)^{-1}, \tag{19}$$

$$\text{where}\bar{\lambda}_{i(\ell)} = \frac{\partial}{\partial\theta_\ell}\lambda_{i(\ell)} \tag{20}$$

where $\bar{\lambda}_{i(\ell)}$ is the derivative of the $i$th eigenvalue of $\mathsf{C}$ in the $\ell$th variable. Here we used some of the results from Dong et al. [2017]. A technique to compute $\bar{\lambda}_{i(\ell)}$ faster is discussed in Section 3.3. This result can be used with gradient descent as follows,

$$\theta_\ell^{(j+1)} = \theta_\ell^{(j)} - \nu_\ell\frac{\partial}{\partial\theta_\ell}\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{y}), \quad j = 0, 1, \cdots \tag{21}$$

where $\nu_\ell$ is the step size for the gradient descent.

Similarly the derivative of the objective function in (14b) can be obtained:

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

taking the derivative w.r.t $\theta_\ell$,

$$\begin{aligned}&\frac{\partial}{\partial\theta_\ell}\mathcal{L}_{\mathrm{GCV}}(\boldsymbol{\theta}|\boldsymbol{y})\\ &= -2\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2}{\lambda_i^2}\right)^{-1}\left(\sum_{i=2}^{n}\frac{|\widetilde{y}_i|^2\,\bar{\lambda}_{i(\ell)}}{\lambda_i^3}\right) + 2\left(\sum_{i=1}^{n}\frac{1}{\lambda_i}\right)^{-1}\left(\sum_{i=1}^{n}\frac{\bar{\lambda}_{i(\ell)}}{\lambda_i^2}\right).\end{aligned} \tag{22}$$

where $\bar{\lambda}_{i(\ell)}$ is the derivative of the $i$th eigenvalue of the Gram matrix, $\mathsf{C}$, in the $\ell$th variable.

## 3.3 Product Kernels

The Bayesian cubature in this research uses product kernels for the demonstrations and numerical implementations. They got nice properties which are helpful to obtain analytical results easily. Especially the gradient descent for parameter search can be explicitly expressed with the derivatives of the kernel as shown in Section 3.3.3. Product kernels in $d$ dimensions are of the form,

$$C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = \prod_{\ell=1}^{d}\Big[1 - \eta_\ell\,\mathfrak{C}(x_\ell, t_\ell)\Big] \tag{23}$$

where $\eta_\ell$ is called shape parameter in the $\ell$th variable for $\ell = 1, \cdots, d$, and $\mathfrak{C}$ is chosen such that to ensure $C_{\boldsymbol{\theta}}$ is symmetric and positive definite. Our goal is to compute $\bar{\lambda}_{i(\ell)}$ for which the kernel derivative is necessary. The derivative of the product kernels can be obtained easily. Please note that $\boldsymbol{\theta}$ denotes all the hyper parameters of the kernel $C$ where $\eta$ is one of them and called the shape parameter.

### *3.3.1 Derivative of the product kernel when $\eta_1 = \cdots = \eta_d = \eta$*

It was suggested to use gradient descent to find optimal shape parameter in Section 3.2. In this section, we compute the gradient for product kernels. When the

$\eta_1 = \cdots = \eta_d = \eta$, the derivative of a product kernel w.r.t. $\eta$ can be obtained as below,

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

*3.3.2 When $\eta_\ell$ is different for each $\ell = 1, \cdots, d$*

In this case, we will have a vector of length $d$ shape parameters. Derivative of the kernel, $C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x})$ (23), with respect to $\eta_\ell$ is,

$$\frac{\partial}{\partial \eta_\ell} C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x}) = \frac{\partial}{\partial \eta_\ell} \prod_{j=1}^{d} \Big[1 - \eta_j \mathfrak{C}(x_j, t_j)\Big], \quad \text{where} \quad \ell = 1, \cdots, d$$
$$= \prod_{j=1, j \neq \ell}^{d} \Big[1 - \eta_j \mathfrak{C}(x_j, t_j)\Big] \Big(-\mathfrak{C}(x_\ell, t_\ell)\Big)$$
$$= \frac{1}{\eta_\ell} \underbrace{\left(\prod_{j=1}^{d} \Big[1 - \eta \mathfrak{C}(x_j, t_j)\Big]\right)}_{C_{\boldsymbol{\theta}}(\boldsymbol{t}, \boldsymbol{x})} \left(1 - \frac{1}{1 - \eta_\ell \mathfrak{C}(x_\ell, t_\ell)}\right).$$

Please note that 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 (23). The $\bar{\lambda}_{i(\ell)}$ can be computed now using (24) with the computed kernel derivative, $\frac{\partial}{\partial \eta_\ell} C_{\boldsymbol{\theta}}$.

*3.3.3 Shape parameter search using steepest descent*

Using the obtained derivative of the eigenvalues, $\bar{\lambda}_{i(\ell)}$, one can easily compute the gradient of the objective function (19) or (22). This can be further used to implement the steepest descent search as introduced in Section 3.2

$$\eta_\ell^{(j+1)} = \eta_\ell^{(j)} - \nu \frac{\partial}{\partial \eta_\ell} \mathcal{L}(\boldsymbol{\theta} | \boldsymbol{y}), \quad j = 0, 1, \cdots, \quad \ell = 1, \cdots, d$$

where $\nu$ is the step size for the gradient descent, $j$ is the iteration index, and $\frac{\partial}{\partial \eta_\ell} \mathcal{L}(\boldsymbol{\theta} | \boldsymbol{y})$ is either (19) or (22) depending on the choice of the hyperparameter search method. The parameter $\eta_\ell$ is usually searched in the whole $\mathbb{R}$ by using the simple domain transformation as explained below.

If $\mathsf{V}$ does not depend on $\boldsymbol{\theta}$ then one can fast compute the derivative of Gram matrix $\mathsf{C}$. Starting from the definition (10) and taking derivative w.r.t. $\theta_\ell$,

$$\frac{\partial \mathsf{C}}{\partial \theta_\ell} = \frac{1}{n} \mathsf{V} \frac{\partial \Lambda}{\partial \theta_\ell} \mathsf{V}^H = \frac{1}{n} \mathsf{V} \bar{\Lambda}_{(\ell)} \mathsf{V}^H,$$
$$\text{where} \quad \bar{\Lambda}_{(\ell)} = \operatorname{diag}(\bar{\boldsymbol{\lambda}}_{(\ell)}), \quad \text{and}$$

$$\bar{\boldsymbol{\lambda}}_{(\ell)} = \frac{\partial \boldsymbol{\lambda}}{\partial \theta_\ell} = \left(\frac{\partial \lambda_i}{\partial \theta_\ell}\right)_{i=1}^{n} = \left(\frac{\partial}{\partial \theta_\ell} \mathsf{V}^H \boldsymbol{C}_1\right) = \mathsf{V}^H \left(\frac{\partial}{\partial \theta_\ell} C_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \boldsymbol{x}_i)\right)_{i=1}^{n}, \tag{24}$$

where we used the fast Bayesian transform property $\boldsymbol{\lambda} = \mathsf{V}^H \boldsymbol{C}_1$. We use the notation $\bar{\boldsymbol{\lambda}}_{(\ell)} = \mathsf{V}^H \bar{\boldsymbol{C}}_{1(\ell)}$ to denote the derivative of the eigenvalue $\boldsymbol{\lambda}_{(\ell)}$, where $\bar{\boldsymbol{C}}_{1(\ell)}$ denotes the first row of the gram matrix after taking the derivative in the $\ell$th variable, i.e.

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

### 3.4 Continuous Valued Kernel Order

JR: Need better and more convincing motivation

In the **?**, it was assumed that 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 handpick the kernel order to match the integrand's smoothness. However, it is not possible to know the integrand's smoothness in most of the practical applications. The constraint to have an integer-valued kernel order also limits the ability to continuously vary the kernel's smoothness to match the integrand like the shape parameter is varied to match.

The integer kernel order is not suitable to optimally search by standard optimization algorithm. As a consequence, one usually ends 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 two alternative forms of the kernel which allow the kernel order to be positive continuous value greater than one or a continuous value in the range $(0, 1)$. Let us recall the infinite series expression that was used to construct the kernel **?**:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) := \sum_{\boldsymbol{k} \in \mathbb{Z}^d} \alpha_{\boldsymbol{k},\boldsymbol{\theta}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{x}} e^{-2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{t}}, \quad \text{where } \alpha_{\boldsymbol{k},\boldsymbol{\theta}} = \prod_{\ell=1}^{d} \frac{\eta_\ell}{|k_\ell|^r}$$

and $\boldsymbol{\theta} = (r, \boldsymbol{\eta})$. This form is convenient for analytical derivations. To make the derivations easier to follow, we fix the dimension $d = 1$,

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

#### *3.4.1 Truncated series kernel*

The following variation to the infinite series kernel (**??**) has the kernel order in the interval $(1, \infty)$. This kernel provides algebraic decay but it is more robust in the hyperparameter search. We reuse the original definition of the infinite kernel (**??**) but truncate to a finite length. This allows the kernel order $r$ continuous valued so that it does not have to be an even integer, which was a 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 $\theta = (r, \eta)$. Since the infinite sum cannot be used directly, 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

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

where $n$ is the number of samples. The reason for having the truncation length and the number of samples equal will be obvious as we proceed further. The first column of the Gram matrix is

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

where $d$ is the number of dimensions. However the direct computation involves $n^2$ computations since we have chosen the truncation length to $n$. We can reduce the computations to $\mathcal{O}(n \log n)$ using the FFT. 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 $\mathfrak{C}_r$, rewrite

$$\boldsymbol{C}_{\boldsymbol{\theta},n} = \left( \prod_{l=1}^{d} [1 + \eta \mathfrak{C}_r(|x_{il} - x_{1l}|)] \right)_{i=1}^{n}. \tag{25}$$

One can observe $|x_{i\ell} - x_{1\ell}| \in \{0, \frac{1}{n}, \frac{2}{n}, \ldots \frac{n-1}{n}\}$ by using the definition of lattice points from Rathinavel [2019]. This can be used to rewrite $\mathfrak{C}_r$ in a much simpler form,

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

This notation is very convenient to show that $\widetilde{\mathfrak{C}}_r$, the discrete Fourier transform of $\mathfrak{C}_r$, can be computed analytically

$$\begin{aligned} \widetilde{\mathfrak{C}}_r(m) &= \sum_{j=0}^{n-1} \mathfrak{C}_r(j/n) e^{-2\pi\sqrt{-1}jm/n} \\ &= \sum_{k=-n/2, k\neq 0}^{n/2-1} \sum_{j=0}^{n-1} \frac{1}{|k|^r} e^{2\pi\sqrt{-1}(k-m)j/n}, \quad \text{by (27)} \\ &= \sum_{k=-n/2, k\neq 0}^{n/2-1} \frac{n}{|k|^r}\, \delta_{k-m \bmod n, 0}\,. \end{aligned}$$

This is the reason we have chosen the truncation length to $n$. Based on the above result, it is evident that $\widetilde{\mathfrak{C}}_r$ can be computed analytically,

$$\widetilde{\mathfrak{C}}_r := \Big(\widetilde{\mathfrak{C}}_r(m)\Big)_{m=0}^{n-1}, \quad \text{where} \quad \widetilde{\mathfrak{C}}_r(m) = \begin{cases} 0, & \text{for} \quad m = 0 \\ \frac{n}{|m|^r}, & \text{for} \quad m = 1, \dots, n/2-1 \\ \frac{n}{|n-m|^r}, & \text{for} \quad m = n/2, \dots, n-1 \end{cases} \tag{26}$$

where we used the fact,

$$\sum_{i=0}^{n-1} e^{2\pi\sqrt{-1}ij/n} = \begin{cases} \frac{1-e^{2\pi\sqrt{-1}jn/n}}{1-e^{2\pi\sqrt{-1}j/n}} = 0, & j \neq 0 \bmod n \\ n, & j = 0 \bmod n. \end{cases} \tag{27}$$

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

$$\begin{aligned} &\frac{1}{n}\sum_{m=0}^{n-1} \widetilde{\mathfrak{C}}_r(m) e^{2\pi\sqrt{-1}lm/n} \\ &= \frac{1}{n}\sum_{m=0}^{n-1}\sum_{j=0}^{n-1} \mathfrak{C}_r(j/n) e^{-2\pi\sqrt{-1}jm/n} e^{2\pi\sqrt{-1}lm/n}, \quad \text{by (27)} \\ &= \frac{1}{n}\sum_{j=0}^{n-1} \mathfrak{C}_r(j/n) n \delta_{(l-j) \bmod n, 0} \\ &= \mathfrak{C}_r(l/n), \quad \text{for} \quad l = 0, \dots, n-1 \end{aligned}$$

This implies that to compute $n$ values of $\Big(C_{\boldsymbol{\theta},n}(\boldsymbol{x}_i, \boldsymbol{x}_1)\Big)_{i=1}^{n}$, we need to have the number of samples and the truncation length the same. The above results are summarized as an algorithm to compute $\mathfrak{C}$ using FFT in Algorithm 2.

**Algorithm 2** The kernel with continuous valued order

**Require:** Number of points to use, $n$;

1: Analytically compute $\widetilde{\mathfrak{C}}_r$ in (25), the discrete Fourier transform of $\mathfrak{C}_r$ using (26)
2: Take the inverse FFT of $\widetilde{\mathfrak{C}}_r$ to get $\mathfrak{C}_r$
3: Using $\mathfrak{C}_r$ compute the truncated series of kernel of truncation length $n$ using (25)

In Algorithm 2, the computational cost of computing $\mathfrak{C}_r$ is $\mathcal{O}(n \log n)$ instead of $\mathcal{O}(n^2)$. Plugging-in the values of $\mathfrak{C}_r$ in (25) gives the kernel. Another major benefit is that the FFT approach in Algorithm 2 is the computations are numerically more stable than the direct sum approach. Please note that these kernels evolve with the truncation length $n$. The larger $n$ value the closer the kernel resembles the original infinite series kernel. One disadvantage is, the truncated series kernels obtain algebraic order decay at best. The infinite series kernel with little modification can be enhanced to obtain exponential decay as shown next.

### *3.4.2 Exponentially decaying kernel*

We propose the following alternative form of the kernel. This kernel can 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 used to denote the kernel order to distinguish it from the notation in (25). This can be rewritten as

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

Let us focus on the first term $(*)$ within the parenthesis in the previous equation,

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

Using this result

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

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 3.5 to map the values to or from $\mathbb{R}$, where the search is usually done. One disadvantage of this kernel is that it is very sensitive to the changes in kernel order $q \in (0,1)$, for even small values, which might cause the hyperparameter search to miss the global minima.

## 3.5 Kernel Hyperparameters Search

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

The various hyperparameters introduced and used by our algorithms need to be optimally chosen. 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 scenarios, the search interval is unknown. So the natural choice is to use

unbounded search over the unbound domain such as `fminsearch` provided by MATLAB. However hyperparameters need to live in a domain that is bounded or semi-bounded. There are some simple domain transformations available to achieve this.

### *3.5.1 Positive kernel shape parameter*

The following parameter map is used to ensure that the shape parameter values are positive real numbers. For $\eta > 0$ as introduced in Section 3.3.1, let

$$\eta(t_1) = \quad e^{t_1}, \qquad\qquad \eta : (-\infty, \infty) \to (0, \infty).$$

Instead of searching for $\eta \in (0, \infty)$, we 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 back to the $(0, \infty)$ interval using

$$\eta_{\text{opt}} = e^{t_{1,\text{opt}}}.$$

### *3.5.2 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 3.4.1,

$$r(t_2) = \quad 1 + e^{t_2}, \qquad\qquad r : (-\infty, \infty) \to (1, \infty).$$

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

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

### *3.5.3 Kernel order $0 < q < 1$*

The following multivariate 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 3.4.2,

$$q(t_3) = \quad \frac{1}{1 + e^{t_3}}, \qquad\qquad q : (-\infty, \infty) \to (0, 1).$$

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

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

### *3.5.4 Combined searching of kernel order $r$ and shape parameter $\boldsymbol{\eta}$*

Instead of searching $\boldsymbol{\eta}$ and $r$ separately one would prefer to search them together so that the most optimal values can be obtained, where $\boldsymbol{\eta} = (\eta_1, \cdots, \eta_d)$ such that $\eta_l \neq \eta_k$, for $l \neq k$. We can combine the parameter maps used above to ensure that the kernel order values in $(1, \infty)$ and shape parameter $\boldsymbol{\eta}$ in $(0, \infty)^d$ as required in Section 3.4.1,

$$\boldsymbol{\theta}(\boldsymbol{t}) = \begin{pmatrix} r(t_1) \\ \eta(t_2) \\ \vdots \\ \eta(t_{d+1}) \end{pmatrix} = \begin{pmatrix} 1 + e^{t_1} \\ e^{t_2} \\ \vdots \\ e^{t_{d+1}} \end{pmatrix}, \quad \boldsymbol{\theta} : \mathbb{R}^{d+1} \to (1, \infty) \times (0, \infty)^d.$$

So instead of searching for $\boldsymbol{\theta}_{\text{opt}}$ in $(1,\infty)\times(0,\infty)^d$, one may search for the optimal

$$\boldsymbol{t} = \boldsymbol{t}^{-1}(\boldsymbol{\theta}) = \begin{pmatrix} \log(r-1) \\ \log(\eta_1) \\ \vdots \\ \log(\eta_d) \end{pmatrix}$$

in the whole real line $\mathbb{R}^{d+1}$. The optimal value $\boldsymbol{t}_{\text{opt}}$ can be transformed back to the desired interval $(1,\infty)\times(0,\infty)^d$ using

$$\boldsymbol{\theta}_{\text{opt}} = \begin{pmatrix} 1+e^{t_{1,\text{opt}}} \\ e^{t_{2,\text{opt}}} \\ \vdots \\ e^{t_{d+1,\text{opt}}} \end{pmatrix}.$$

Similarly one can map the kernel order $q \in (0,1)$ Section 3.4.2, and $\boldsymbol{\eta}$ in to a multivariate hyperparameter search.

## 4 Diagnostics for the Gaussian Process Assumption

The starting point for our Bayesian cubature is the assumption that the integrand arises from a Gaussian process, $f \sim \mathcal{GP}(m, s^2 C_{\boldsymbol{\theta}})$. This implies that the function values, $\boldsymbol{f}$, shall satisfy a multivariate Gaussian distribution, as in (6). If we you could transform it to the standard Gaussian, we can visually verify with readily available tools such as Q-Q plot. We propose a transform to the given function data $\boldsymbol{f}$ to get standard Gaussian, resulting in $\boldsymbol{Z}$:

$$\boldsymbol{Z} = \frac{1}{n} s^{-1} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H (\boldsymbol{f} - m\mathbf{1}). \tag{28}$$

We can show that the transformed data, $\boldsymbol{Z}$, has zero mean and is also uncorrelated because

$$\begin{aligned}
\mathbb{E}[\boldsymbol{Z}] &= \frac{1}{n} s^{-1} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H (\mathbb{E}[\boldsymbol{f}] - m\mathbf{1}) \\
&= 0, \qquad \text{because } \mathbb{E}[\boldsymbol{f}] = m \\
\operatorname{cov}(\boldsymbol{Z}) &= \frac{1}{n^2} s^{-2} \mathbb{E}\left[ \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H (\boldsymbol{f} - m\mathbf{1})(\boldsymbol{f} - m\mathbf{1})^T \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H \right] \\
&= \frac{1}{n^2} s^{-2} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H \mathbb{E}\left[ (\boldsymbol{f} - m\mathbf{1})(\boldsymbol{f} - m\mathbf{1})^T \right] \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H \\
&= \frac{1}{n^2} s^{-2} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H s^2 \frac{1}{n} \mathsf{V}\mathsf{\Lambda}\mathsf{V}^H \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H \\
&= \frac{1}{n^3} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} (n) \mathsf{\Lambda}(n) \mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H = \mathsf{I}
\end{aligned}$$

Thus, the elements of $\boldsymbol{Z}$ are IID standard Gaussian random variables. In practice, using the estimated $m_{\text{EB}}$, such as empirical Bayes, further simplifies,

$$\begin{aligned}
\boldsymbol{Z} &= \frac{1}{n} s^{-1} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \mathsf{V}^H (\boldsymbol{f} - m\mathbf{1}) \\
&= \frac{1}{n} s^{-1} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} (\mathsf{V}^H \boldsymbol{f} - \frac{\tilde{f}_1}{n} \mathsf{V}^H \mathbf{1}) \\
&= \frac{1}{n} s^{-1} \mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \left( \mathsf{V}^H \boldsymbol{f} - \tilde{f}_1 \left(1, 0, \ldots, 0\right)^T \right)
\end{aligned}$$

To verify this hypothesis, we build a custom function using the known Bernoulli polynomial series,

$$f_{\theta,r}(\boldsymbol{x}) = \widehat{f}_0 + \theta \sum_{\boldsymbol{k}\in\{1,\dots,N\}^d} \left[\widehat{f}_c(\boldsymbol{k})\cos(2\pi \boldsymbol{k}^T\boldsymbol{x}) + \widehat{f}_s(\boldsymbol{k})\sin(2\pi \boldsymbol{k}^T\boldsymbol{x})\right] \tag{29}$$

$$\text{where} \quad \widehat{f}_0, \widehat{f}_c(\boldsymbol{k}), \widehat{f}_s(\boldsymbol{k}) \overset{\text{IID}}{\sim} \mathcal{N}(0, a^{\|\boldsymbol{k}\|_0} b^{d-\|\boldsymbol{k}\|_0} \prod_{k_j\neq 0} k_j^{-r}),$$

$$\theta = a/b$$

The series in (29) is truncated to an arbitrarily chosen length $N$. It has zero mean and covariance

$$\begin{aligned}\operatorname{cov}(f) &= \mathbb{E}[f_{\theta,r}(x) f_{\theta,r}(t)] \\ &= \operatorname{cov}(f_0) + \theta \sum_{k=1}^{N} \left( \mathbb{E}(f_{c,k}^2) \frac{\cos(2\pi kx)\cos(2\pi kt)}{k^{2r}} + \mathbb{E}(f_{s,k}^2) \frac{\sin(2\pi kx)\sin(2\pi kt)}{k^{2r}} \right) \\ &= b^2 + a^2\theta \sum_{k=1}^{N} \frac{\cos(2\pi k(x-t))}{k^{2r}}\end{aligned}$$

Figures 4 and **??** are normal probability plots of the $Z_i$ using empirical Bayes estimates of $m$ and $\boldsymbol{\theta}$. **more goes here**.

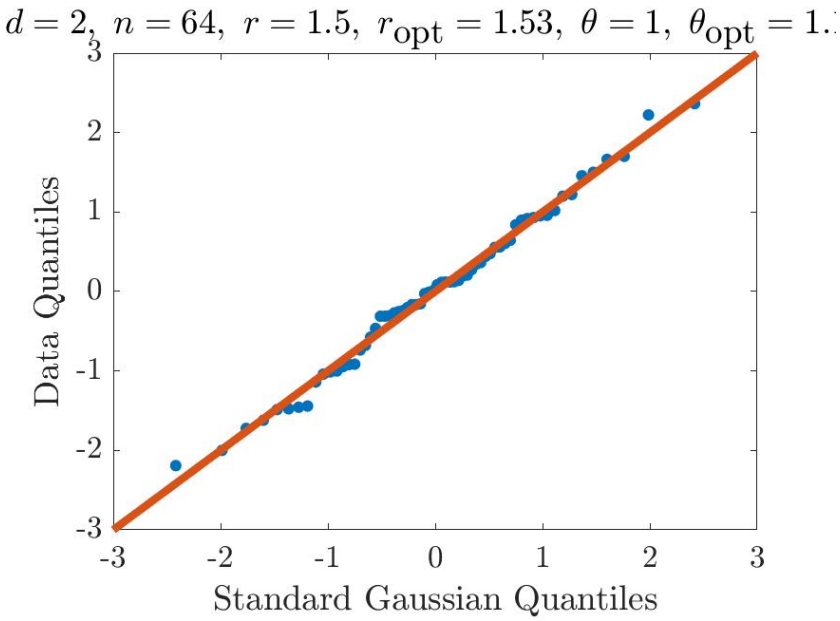


**Fig. 3** Q-Q plot : Random function

4.1 observations

JR: write your observations from the numerical experiments

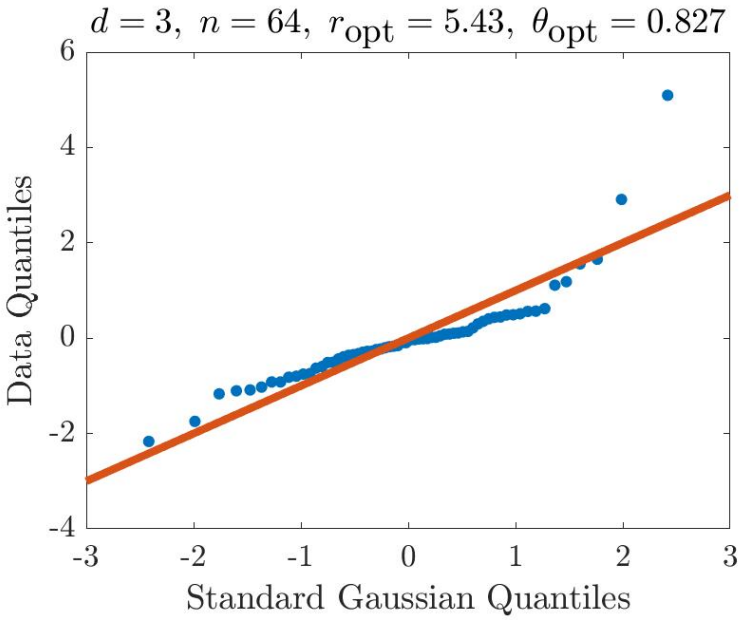


**Fig. 4** Q-Q plot : Keister function integrand

The Q-Q plots in the figures 3, 4 and 5 show that the transformed integrand values $\boldsymbol{Z}$ using (28) behave like standard Gaussian. Also for the random function the Q-Q plot is much nicer than the other functions.

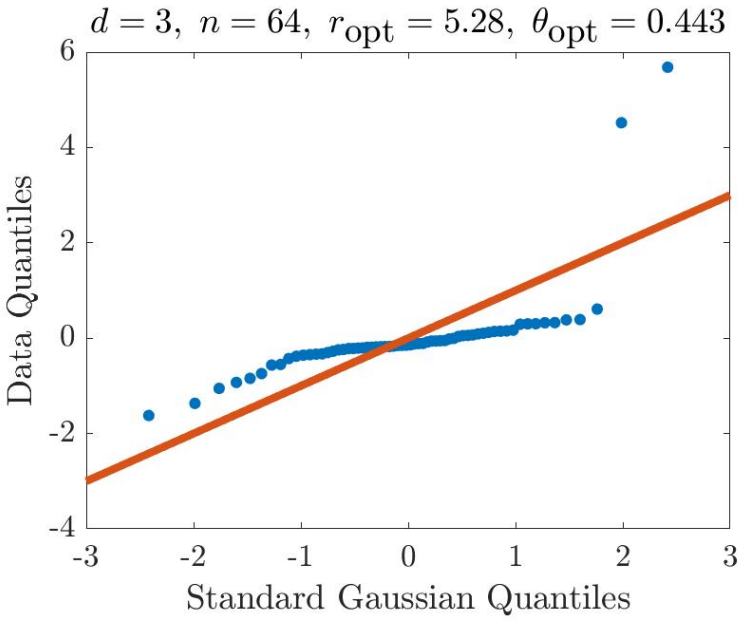


**Fig. 5** Q-Q plot : ExpCos

## 5 Sobol' Nets and Walsh Kernels

The previous section showed an automatic Bayesian cubature algorithm without mentioning any specific nodes or kernels. In this chapter, we demonstrate another approach to formulate fast Bayesian transform using matching kernels and point sets. **?** demonstrated Bayesian cubature using rank-1 lattice points and shift-invariant kernels. Scrambled Sobol' nets and Walsh kernels are paired to achieve $\mathcal{O}(n^{-1+\epsilon})$ order error convergence where $n$ is the sample size. Sobol' nets Sobol' [1967] are low discrepancy points, used extensively in numerical integration, simulation, and optimization. The results of this chapter can be summarized as a theorem,

**Theorem 2** *Any symmetric, positive definite, digital shift-invariant covariance kernel of the form* (34) *scaled to satisfy* (12)*, when matched with digital net data-sites, satisfies assumptions* (11)*. The* fast Walsh-Hadamard transform *(FWHT) can be used to expedite the estimates of* $\boldsymbol{\theta}$ *in* (14) *and the credible interval widths* (16) *in* $\mathcal{O}(n \log n)$ *operations. The cubature,* $\widehat{\mu}$*, is just the sample mean.*

We introduce the necessary concepts and prove this theorem in the remaining of this chapter.

### 5.1 Sobol' Nets

Nets were developed to provide deterministic sample points for quasi-Monte Carlo rules Niederreiter [2005]. Nets are defined geometrically using elementary intervals,

which are subintervals of the unit cube $[0,1)^d$. The $(t,m,d)$-nets in base $b$, introduced by Niederreiter, whose quality is governed by $t$. Lower values of $t$ correspond to $(t,m,d)$-nets of higher quality Baldeaux [2010].

**Definition 1** *Let $\mathcal{A}$ be the set of all elementary intervals $\mathcal{A} \subset [0,1)^d$ where $\mathcal{A} = \prod_{\ell=1}^{d}[\alpha_\ell b^{-\gamma_\ell}, (\alpha_\ell+1)b^{-\gamma_\ell})$, with $d, b, \gamma_\ell \in \mathbb{N}, b \geq 2$ and $b^{\gamma_\ell} > \alpha_\ell \geq 0$. For $m, t \in \mathbb{N}, m \geq t \geq 0$, the point set $\mathcal{P}_m \in [0,1)^d$ with $n = b^m$ points is a $(t,m,d)$ – net in base $b$ if every $\mathcal{A}$ with volume $b^{t-m}$ contains $b^t$ points of $\mathcal{P}_m$.*

Digital $(t,m,d)$-nets are a special case of $(t,m,d)$-nets, constructed using matrix-vector multiplications over finite fields. Digital sequences are infinite length digital nets, i.e., the first $n = b^m$ points of a digital sequence comprise a digital net for all integer $m \in \mathbb{N}_0$.

**Definition 2** *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 $z = 0.z_1 z_2 \dots (\text{base } b) \in [0,1)$, define the $\infty \times 1$ vector of the digits of $z$, that is, $\vec{z} = (z_1, z_2, \dots)^T$. 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*

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

*The value of $t$ as mentioned in Definition 1 depends on the choice of $\mathsf{G}_\ell$.*

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

$$\boldsymbol{z} \oplus \boldsymbol{y} = \left( \sum_{j=1}^{\infty} [z_{\ell j} + y_{\ell j} \bmod b] b^{-j} \bmod 1 \right)_{\ell=1}^{d},$$

$$\boldsymbol{z} \ominus \boldsymbol{y} = \left( \sum_{j=1}^{\infty} [z_{\ell j} - y_{\ell j} \bmod b] b^{-j} \bmod 1 \right)_{\ell=1}^{d},$$

where

$$\boldsymbol{z} = \left( \sum_{j=1}^{\infty} z_{\ell j} b^{-j} \right)_{\ell=1}^{d}, \quad \boldsymbol{y} = \left( \sum_{j=1}^{\infty} y_{\ell j} b^{-j} \right)_{\ell=1}^{d}, \quad z_{\ell j}, y_{\ell j} \in \{0, \cdots, b-1\}.$$

Similarly for integer values in $\mathbb{N}_0^d$, the digitwise addition, $\oplus$, and subtraction $\ominus$, are defined in terms of their $b$-ary expansions,

$$\boldsymbol{k} \oplus \boldsymbol{l} = \left( \sum_{j=0}^{\infty} [k_{\ell j} + l_{\ell j} \bmod b] b^{j} \bmod 1 \right)_{\ell=1}^{d},$$

$$\boldsymbol{k} \ominus \boldsymbol{l} = \left( \sum_{j=0}^{\infty} [k_{\ell j} - l_{\ell j} \bmod b] b^{j} \bmod 1 \right)_{\ell=1}^{d},$$

where

$$\boldsymbol{k} = \left( \sum_{j=0}^{\infty} k_{\ell j} b^{j} \right)_{\ell=1}^{d}, \quad \boldsymbol{l} = \left( \sum_{j=0}^{\infty} l_{\ell j} b^{j} \right)_{\ell=1}^{d}, \quad \boldsymbol{k}_{\ell j}, \boldsymbol{l}_{\ell j} \in \{0, \cdots, b-1\}.$$

Let $\{\boldsymbol{z}_i\}_{i=0}^{b^m-1}$ be a digital net. Then
$\forall i_1, i_2 \in \{0, \cdots, b^m-1\}, \quad \boldsymbol{z}_{j_1} \oplus \boldsymbol{z}_{i_2} = \boldsymbol{z}_{i_3}, \quad$ for some $i_3 \in \{0, \cdots, b^m-1\}$.

The following very useful result, which will be further used to obtain the fast Bayesian transform, arises from the fundamental property of digital nets.

**Lemma 1** *Let $\{\boldsymbol{z}_i\}_{i=0}^{b^m-1}$ be the digital-net and the corresponding digitally shifted net be $\{\boldsymbol{x}_i\}_{i=0}^{b^m-1}$, i.e.,*

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

*where $\vec{x}_{i\ell}$ is the $\ell$th component of ith digital net and $\vec{\Delta}_\ell$ is the digital shift for the $\ell$th component. Then,*

$$\boldsymbol{x}_i \ominus \boldsymbol{x}_j = \boldsymbol{z}_i \ominus \boldsymbol{z}_j = \boldsymbol{z}_{i \ominus j}, \quad \forall i, j \in \mathbb{N}_0. \tag{30}$$

*Also the digital subtraction is symmetric,*

$$\boldsymbol{x}_i \ominus \boldsymbol{x}_i = \boldsymbol{0}, \qquad \boldsymbol{x}_i \ominus \boldsymbol{x}_j = \boldsymbol{x}_j \ominus \boldsymbol{x}_i, \quad \forall i, j \in \mathbb{N}_0. \tag{31}$$

A simple proof can be found in Rathinavel [2019].

We chose digitally shifted and scrambled nets Hickernell and Yue [2000] for our Bayesian cubature algorithm. Digital shifts help to avoid having nodes at the origin, similar to the random shift used with lattice nodes. Scrambling helps to eliminate bias while retaining the low-discrepancy properties. A proof that a scrambled net preserves the property of $(t, m, d)$-net almost surely can be found in Owen Owen [1995]. The scrambling method proposed by Matoušek Matoušek [1998] is preferred since it is more efficient than the Owen's scrambling.

Sobol' nets Sobol' [1976] are a special case of $(t, m, d)$-nets when base $b = 2$. An example of 64 Sobol' nets in $d = 2$ is given in Figure 6. 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 Kuo and Nuyens [2016] and Nuyens for more on generating matrices. We use randomly scrambled and digitally shifted Sobol' sequences in this research Hong and Hickernell [2003].

## 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 the Fourier transform used with lattice points (**?**), the Walsh-Hadamard transform, which we will simply call Walsh transform, is used for the digital nets. The Walsh transform is defined using Walsh functions. Recall $\mathbb{N}_0 := \{0, 1, 2, \cdots\}$. The one-dimensional Walsh functions in base $b$ are defined as

$$\mathrm{wal}_{b,k}(x) := e^{2\pi\sqrt{-1}(x_1k_0 + x_2k_1 + \cdots)/b} = e^{2\pi\sqrt{-1}\vec{k}^T\vec{x}/b}, \tag{32}$$

for $x \in [0, 1)$ and $k \in \mathbb{N}_0$ and the unique base $b$ expansions $x = \sum_{j\geq 1} x_j b^{-i} = (0.x_1x_2\cdots)_b$, $\vec{x} = (x_1, x_2, \cdots)^T$ $k = \sum_{j\geq 0} k_j b^j = (\cdots k_1k_0)_b$, $\vec{k} = (k_0, k_1, \cdots)^T$, and $\vec{k}^T\vec{x} = x_1k_0 + x_2k_1 + \cdots$ where the number of digits used in (32) are limited to the length required to represent $x$ or $k$, i.e., $\max\left(\lceil -\log_b x\rceil, \lceil \log_b k\rceil\right)$. Multivariate

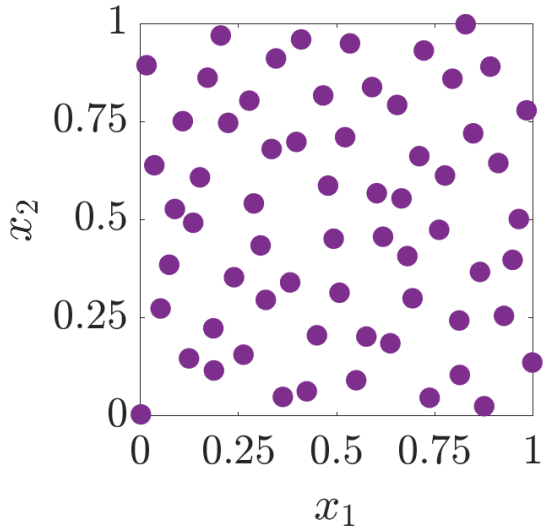


**Fig. 6** Example of a scrambled Sobol' node set in $d = 2$. This plot can be reproduced using `PlotPoints.m`.

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

$$\mathrm{wal}_{b,\boldsymbol{k}}(\boldsymbol{x}) := \prod_{\ell=1}^{d} \mathrm{wal}_{b,k_\ell}(x_\ell)$$

As shown in (32), for the case of $b = 2$, the Walsh functions only take the values in $\{1, -1\}$, i.e., $\mathrm{wal}_{b,\boldsymbol{k}} : [0,1)^d \to \{-1, 1\}$, $k \in \mathbb{N}_0^d$. Walsh functions form an orthonormal basis of the Hilbert space $L^2[0,1)^d$,

$$\int_{[0,1)^d} \mathrm{wal}_{b,\boldsymbol{l}}(\boldsymbol{x})\mathrm{wal}_{b,\boldsymbol{k}}(\boldsymbol{x})\mathrm{d}x = \delta_{\boldsymbol{l},\boldsymbol{k}}, \quad \forall \boldsymbol{l}, \boldsymbol{k} \in \mathbb{N}_0^d$$

Digital nets are designed to integrate certain Walsh functions without error. Thus our Bayesian cubature algorithm integrates linear combinations of certain Walsh functions 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. In this case, the Walsh function is simply

$$\mathrm{wal}_{2,\boldsymbol{k}}(\boldsymbol{x}) = (-1)^{\vec{\boldsymbol{k}}^T \vec{\boldsymbol{x}}}.$$

### *5.2.2 Walsh kernels*

Consider the covariance kernels of the form,

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) = K_{\boldsymbol{\theta}}(\boldsymbol{x} \ominus \boldsymbol{t}) \tag{33}$$

where $\ominus$ is bitwise subtraction. This is called a *digitally 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 function $K_{\boldsymbol{\theta}}$, it follows that assumption (12) is satisfied. The function $K_{\boldsymbol{\theta}}$ must be of the form that ensures that $C_{\boldsymbol{\theta}}$ is symmetric and positive definite, as assumed in (5). We drop the $\boldsymbol{\theta}$

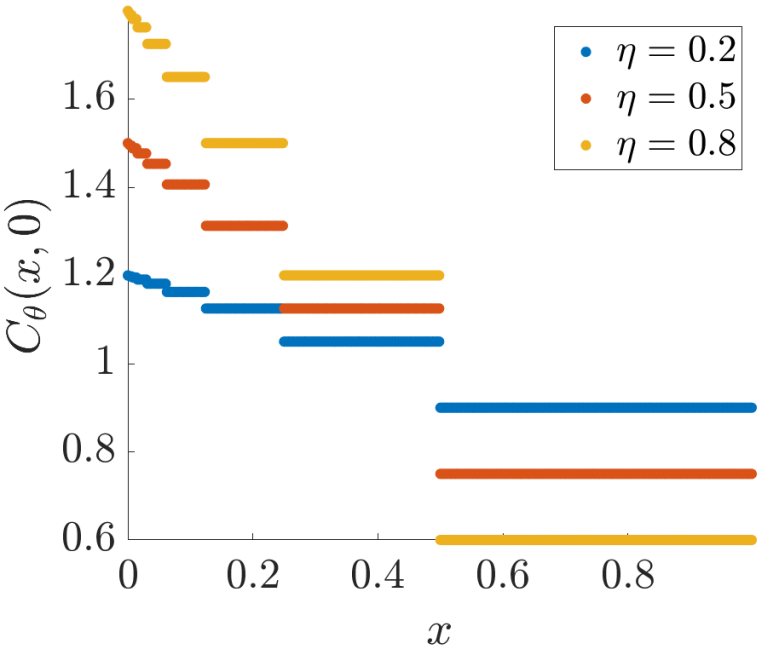


**Fig. 7** Walsh kernel of order $r = 1$ in dimension $d = 1$. This figure can be reproduced using `plot_walsh_kernel.m`.

sometimes to make the notation simpler. The Walsh kernels are of the form,

$$K_{\boldsymbol{\theta}}(\boldsymbol{x} \ominus \boldsymbol{t}) = \prod_{\ell=1}^{d} 1 + \eta_\ell \omega_r(x_\ell \ominus t_\ell), \quad \boldsymbol{\eta} = (\eta_1, \cdots, \eta_d), \quad \boldsymbol{\theta} = (r, \boldsymbol{\eta}) \tag{34}$$

where $r$ is the kernel order, $\boldsymbol{\eta}$ is the kernel shape parameter, and

$$\omega_r(x) = \sum_{k=1}^{\infty} \frac{\mathrm{wal}_{2,k}(x)}{2^{2r\lfloor \log_2 k \rfloor}}.$$

Explicit expression is available for $\omega_r$ in the case of order $r = 1$ Nuyens [2013],

$$\omega_1(x) = 6\left(\frac{1}{6} - 2^{\lfloor \log_2 x \rfloor - 1}\right). \tag{35}$$

The Figure 7 shows the Walsh kernel (34) of order $r = 1$ in the interval $[0, 1)$. Unlike the shift-invariant kernels used with lattice nodes, low order Walsh kernels are discontinuous and are only piecewise constant. Smaller $\eta_\ell$ implies lesser variation in the amplitude of the kernel. Also, the Walsh kernels are digitally shift invariant but not periodic.

### 5.3 Eigenvectors

We show the eigenvectors $\mathsf{V}$ in (10) of the Gram matrix formed by the covariance kernel (34) and Sobol' nets are the columns of the Walsh-Hadamard matrix. First we introduce the necessary concepts.

*5.3.1 Walsh transform*

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

The WHT involves multiplications by $2^m \times 2^m$ Walsh-Hadamard matrices, which is constructed recursively, starting with $\mathsf{H}^{(0)} = 1$,

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

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

$$\vdots$$

$$\mathsf{H}^{(m)} = \begin{pmatrix} \mathsf{H}^{(m-1)} & \mathsf{H}^{(m-1)} \\ \mathsf{H}^{(m-1)} & -\mathsf{H}^{(m-1)} \end{pmatrix} = \underbrace{\mathsf{H}^{(1)} \bigotimes \cdots \bigotimes \mathsf{H}^{(1)}}_{m \text{ times}} = \mathsf{H}^{(1)} \bigotimes \mathsf{H}^{(m-1)} \tag{36}$$

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

$$\mathsf{H}^{(m)} = \left( (-1)^{(\vec{i}^T \vec{j})} \right)_{i,j=0}^{2^m - 1},$$

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

*5.3.2 Eigenvectors of $\mathsf{C}$ are columns of Walsh-Hadamard matrix*

The Gram matrix $\mathsf{C}_{\boldsymbol{\theta}}$ formed by Walsh kernels and Sobol' nodes have a special structure called block-Toeplitz, 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. We prove that the eigenvectors of $\mathsf{C}_{\boldsymbol{\theta}}$ are columns of a Walsh-Hadamard matrix in two theorems.

**Theorem 3** *Let $(\boldsymbol{x}_i)_{i=0}^{n-1}$ be digitally shifted Sobol' nodes and $K$ be any function, then the Gram matrix,*
$\mathsf{C}_{\boldsymbol{\theta}} = \big(C(\boldsymbol{x}_i, \boldsymbol{x}_j)\big)_{i,j=0}^{n-1} = \big(K(\boldsymbol{x}_i \ominus \boldsymbol{x}_j)\big)_{i,j=0}^{n-1}$,

$$\text{where} \qquad n = 2^m, \quad C(\boldsymbol{x}, \boldsymbol{t}) = K(\boldsymbol{x} \ominus \boldsymbol{t}), \quad \boldsymbol{x}, \boldsymbol{t} \in [0,1)^d,$$

*is a $2 \times 2$ block-Toeplitz matrix and all the sub-blocks and their sub-sub-blocks, etc. are also $2 \times 2$ block-Toeplitz.*

**Theorem 4** *The Walsh-Hadamard matrix $\mathsf{H}^{(m)}$ factorizes $\mathsf{C}_{\boldsymbol{\theta}}^{(m)}$, so that the columns of Walsh-Hadamard matrix are the eigenvectors of $\mathsf{C}_{\boldsymbol{\theta}}^{(m)}$, i.e.,*

$$\mathsf{H}^{(m)} \mathsf{C}_{\boldsymbol{\theta}}^{(m)} = \Lambda^{(m)} \mathsf{H}^{(m)}, \quad m \in \mathbb{N}.$$

A detailed proof for Theorem 3 and 4 can be found in Rathinavel [2019].

### 5.4 Fast Bayesian transform

We can easily show that the Walsh-Hadamard matrices satisfy the assumptions of fast Bayesian transform (11). As shown in Section 5.3.2 the columns of $\mathsf{H}^{(m)}$ are the eigenvectors. Since the Gram matrix $\mathsf{C}$ is symmetric, the columns/rows of Walsh-Hadamard matrices are mutually orthogonal. Thus the Gram matrix can be written as

$$\mathsf{C}^{(m)} = \frac{1}{n}\mathsf{H}^{(m)}\Lambda^{(m)}\mathsf{H}^{(m)}, \quad \text{where} \quad \mathsf{H}^{(m)} = \underbrace{\mathsf{H}^{(1)} \bigotimes \cdots \bigotimes \mathsf{H}^{(1)}}_{m \text{ times}}. \tag{37}$$

Assumption (11b) follows automatically by the fact that Walsh-Hadamard matrices can be constructed analytically. Assumption (11a) can also be verified as the first row/column are one vectors. Finally, assumption (11c) 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, $\mathsf{V} := \mathsf{H}$, as per (11).

We have implemented a fast adaptive Bayesian cubature algorithm using the kernel (34) with $r = 1$ and Sobol' points Bratley and Fox [1988] in MATLAB as part of the Guaranteed Adaptive Integration Library (GAIL) Choi et al. [2013–2017] as `cubBayesNet_g`. The Sobol' points used in this algorithm are generated using MATLAB's builtin function `sobolset` and scrambled using MATLAB function `scramble` Hong and Hickernell [2003]. The fast Walsh-Hadamard transform (37) is computed using MATLAB's builtin function `fwht` with *hadamard* ordering.

### 5.5 Iterative Computation of Walsh Transform

In every iteration of our algorithm, we double the number of function values. Using the technique described here, we have to only compute the Walsh transform for the newly added function values. Similar to the lattice points, Sobol' points are extensible by definition. This property is used in our algorithm to improve the integration accuracy till the required error tolerance is met. Sobol' nodes can be combined with Hadamard matrices as demonstrated here for iterative computation. 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,

$$\widetilde{\boldsymbol{y}} = \mathsf{H}^{(m+1)}\boldsymbol{y}$$

$$
\begin{aligned}
&= \begin{pmatrix} \mathsf{H}^{(m)} & \mathsf{H}^{(m)} \\ \mathsf{H}^{(m)} & -\mathsf{H}^{(m)} \end{pmatrix} \begin{pmatrix} \boldsymbol{y}^{(1)} \\ \boldsymbol{y}^{(2)} \end{pmatrix}, \qquad \text{by (36)} \\
&= \begin{pmatrix} \mathsf{H}^{(m)}\boldsymbol{y}^{(1)} + \mathsf{H}^{(m)}\boldsymbol{y}^{(2)} \\ \mathsf{H}^{(m)}\boldsymbol{y}^{(1)} - \mathsf{H}^{(m)}\boldsymbol{y}^{(2)} \end{pmatrix} \\
&= \begin{pmatrix} \widetilde{\boldsymbol{y}}^{(1)} + \widetilde{\boldsymbol{y}}^{(2)} \\ \widetilde{\boldsymbol{y}}^{(1)} - \widetilde{\boldsymbol{y}}^{(2)} \end{pmatrix} =: \widetilde{\boldsymbol{y}} \quad .
\end{aligned}
$$

As before with the lattice nodes, the computational cost to compute $\mathsf{V}^{(m+1)H}\boldsymbol{y}$ is 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.

### 5.6 Higher Order Nets

Higher order digital nets are an extension of $(t, m, d)$-nets, introduced in Dick [2008]. They can be used to numerically integrate smoother functions which are not necessarily periodic, but have square integrable mixed partial derivatives of order $\alpha$, 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 emphasize 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.

One could develop matching digitally shift invariant kernels to formulate the fast Bayesian cubature. Bayesian cubatures using higher order digital nets are a topic for future research.

## 6 Numerical Experiments

JR: use uniformly randomly chosen $\varepsilon$ instead 4 fixed

Fast Bayesian cubature algorithms developed in this research are demonstrated using three commonly used integration examples. These integrals are evaluated using our algorithm `cubBayesNet_g` and compared with `cubBayesLattice_g` which was developed in **?**. The `cubBayesLattice_g` is a fast automatic Bayesian cubature algorithm which uses rank-1 lattice nodes. 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.

### 6.1 Testing Methodology

Four hundred different error tolerances, $\varepsilon$, were randomly chosen from a fixed interval for each example. The intervals for error tolerance were chosen depending

on the difficulty of the problem. The nodes used in `cubBayesNet_g` were the randomly scrambled and shifted Sobol' points supplied by MATLAB's Sobol' sequence generator.

For each integral example, and each stopping criteria—empirical Bayes, full Bayes, and generalized cross-validation—our algorithm is run 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 not used in the examples with `cubBayesNet_g`. Since the `cubBayesNet_g` does not need this requirement, the integrands are used directly.

### 6.2 Multivariate Gaussian Probability

This example was introduced in **?**, where we used with the Matérn covariance kernel and shift-invariant kernel. Here we use $f_{\text{Genz}}$ (**??**) without any periodization as it is not required, and chose $d = 3$ and $r = 1$. The simulation results for this example integrand are summarized in Figures 8, 9, and 10. In all cases, `cubBayesNet_g` returns an approximation within the prescribed error tolerance. We used the same setting as before with generic slow Bayesian cubature in **?** for comparision. For error threshold $\varepsilon = 10^{-5}$ with empirical stopping criterion, our fast algorithm takes about 2 seconds as shown in Figure **??** whereas the basic algorithm takes 30 seconds as shown in Figure **??**. `cubBayesNet_g` uses fast Walsh transform which is slower in MATLAB due to the way it was implemented. This is reason it takes more longer the `cubBayesLattice_g`. But comparing the number of samples, $n$, used for integration provides more insight which directly relates to 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 but it is less conservative. One can also observe from the figures that the credible intervals are narrower than in Figure **??**. This shows that `cubBayesNet_g` with $r = 1$ kernel more accurately approximates the integrand.

### 6.3 Keister's Example

This multidimensional integral function comes from Keister [1996] and is inspired by a physics application:

$$\begin{aligned} \mu &= \int_{\mathbb{R}^d} \cos(\|\boldsymbol{t}\|) \exp(-\|\boldsymbol{t}\|^2)\, \mathrm{d}\boldsymbol{t} \\ &= \int_{[0,1]^d} f_{\text{Keister}}(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}, \end{aligned} \tag{38}$$

where

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

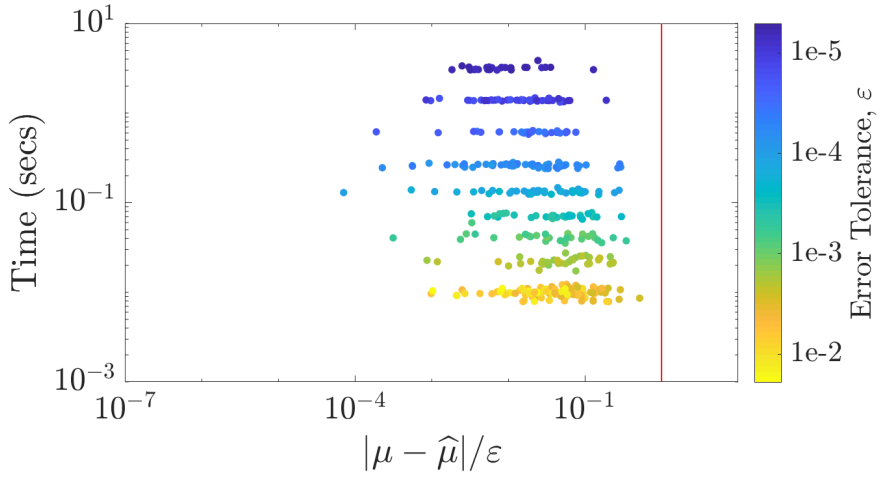


**Fig. 8** Multivariate normal probability example with empirical Bayes stopping criterion.

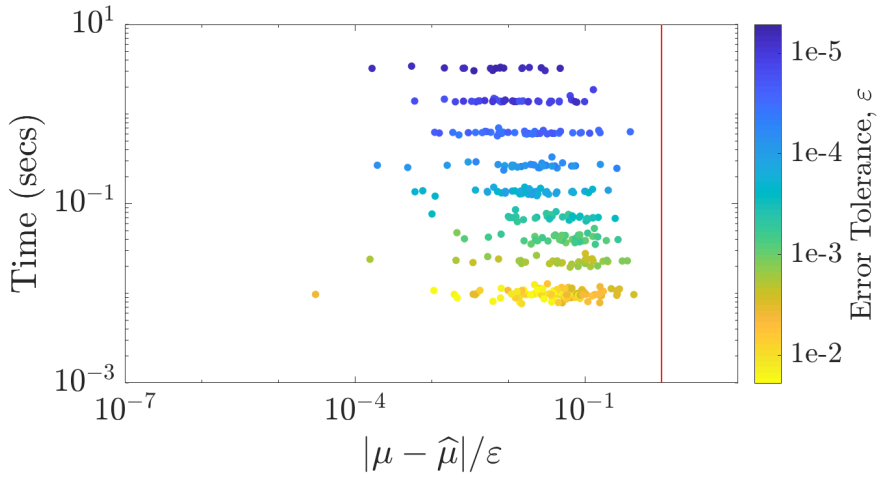


**Fig. 9** Multivariate normal probability example with the full-Bayes stopping criterion.

and $\Phi$ is the standard normal distribution. The true value of $\mu$ 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 $\Gamma$ denotes the gamma function, and

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

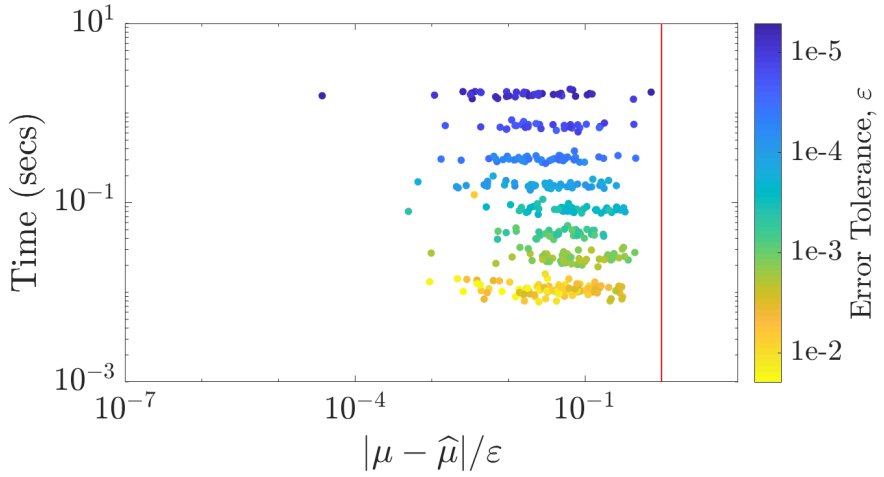


**Fig. 10** Multivariate normal probability example with the GCV stopping criterion.

$$
\begin{aligned}
I_c(2) &= \frac{1 - I_s(1)}{2}, \qquad I_s(2) = \frac{I_c(1)}{2} \\
I_c(j) &= \frac{(j-2)I_c(j-2) - I_s(j-1)}{2}, \qquad j = 3, 4, \ldots \\
I_s(j) &= \frac{(j-2)I_s(j-2) - I_c(j-1)}{2}, \qquad j = 3, 4, \ldots .
\end{aligned}
$$

Figures 11, 12 and 13 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 whereas in Section **??**, $r = 2$ kernel was used. This necessitates `cubBayesNet_g` to use more samples for integration. As observed from the figures, the GCV stopping criterion achieved the results faster than the others but it is less conservative which is also the case with the multivariate Gaussian example.

## 6.4 Option Pricing

The price of financial derivatives can often be modeled by high dimensional integrals. If the underlying asset is described in terms of a discretized geometric Brownian motion, then the fair price of the option is:

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

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

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

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

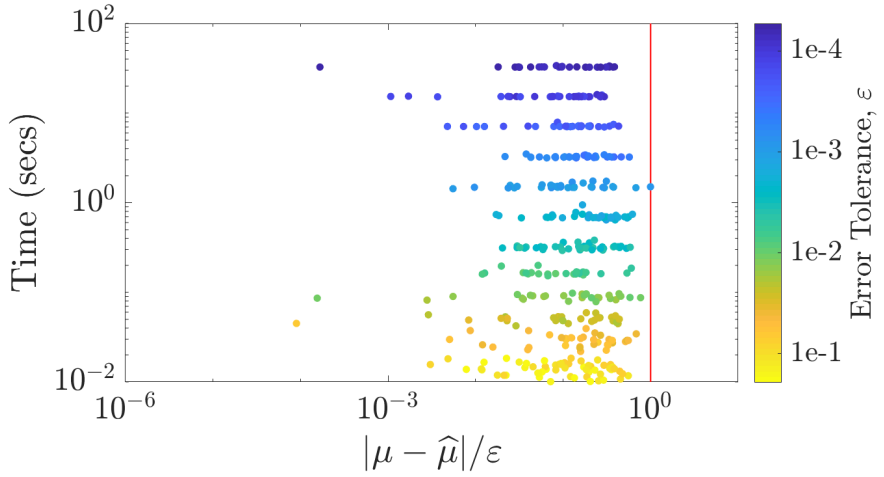


**Fig. 11** Keister example using the empirical Bayes stopping criterion.

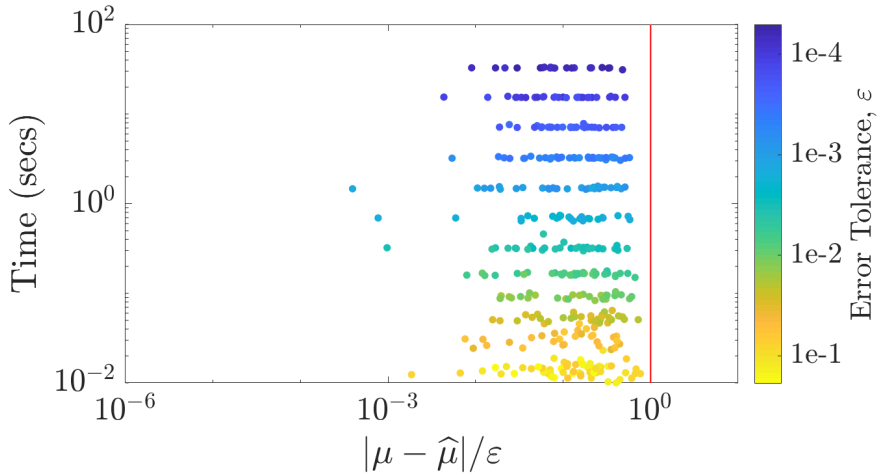


**Fig. 12** Keister example using the full-Bayes stopping criterion.

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

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

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

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, $\sigma$ the volatility, and $K$ the strike price.

The Figures 14, 15 and 16 summarize the numerical results for the option pricing example using the same values for, $T$, $d$, $S_0$, $r$, $\sigma$, $K$, as in Section **??**. As mentioned before, this integrand has a kink caused by the max function, so,

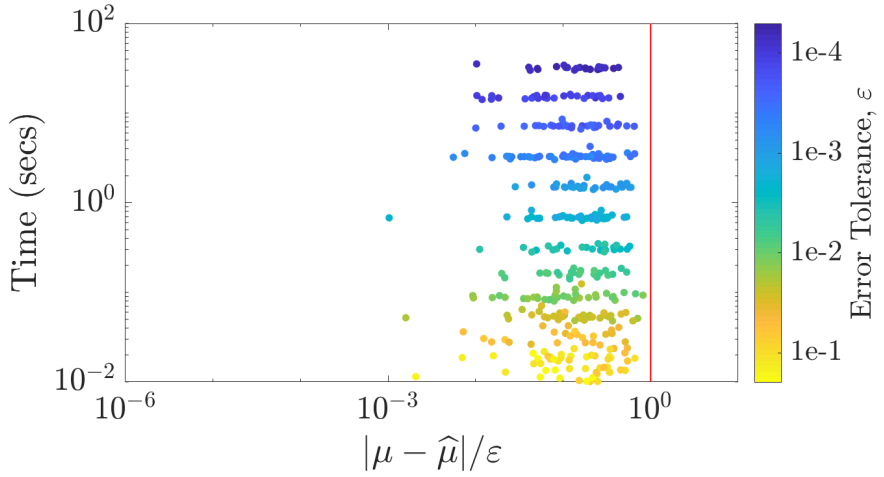


**Fig. 13** Keister example using the GCV stopping criterion.

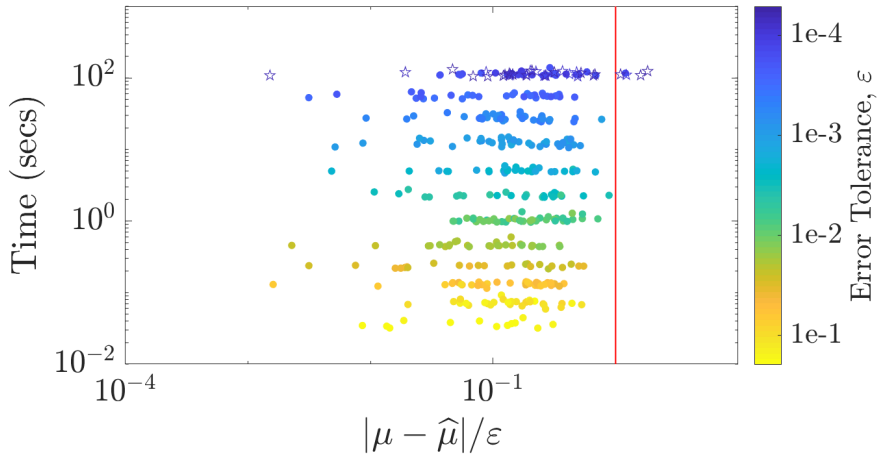


**Fig. 14** Option pricing using the empirical Bayes stopping criterion. The hollow stars indicate the algorithm has not met the error threshold $\epsilon$ even with using maximum $n$.

`cubBayesNet_g` could be more efficient than `cubBayesLattice_g`, as no periodization transform is required. This can be observed from the number of samples used for intgration to meet the same error threshold. For the error tolerance, $\varepsilon = 10^{-3}$, `cubBayesLattice_g` used $n = 2^{20}$ samples, whereas `cubBayesNet_g` used $n = 2^{17}$ samples.

## 6.5 Discussion

JR: Move this to end of chapter

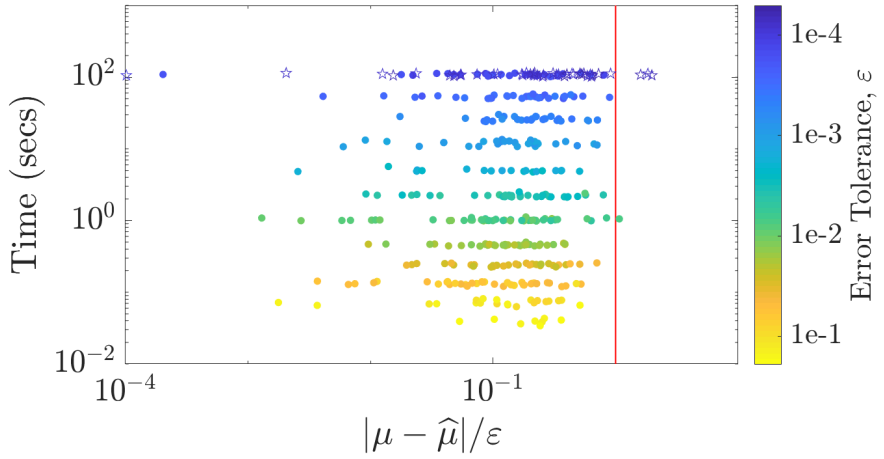


**Fig. 15** Option pricing using the full-Bayes stopping criterion. The hollow stars indicate the algorithm has not met the error threshold $\epsilon$ even with using maximum $n$.

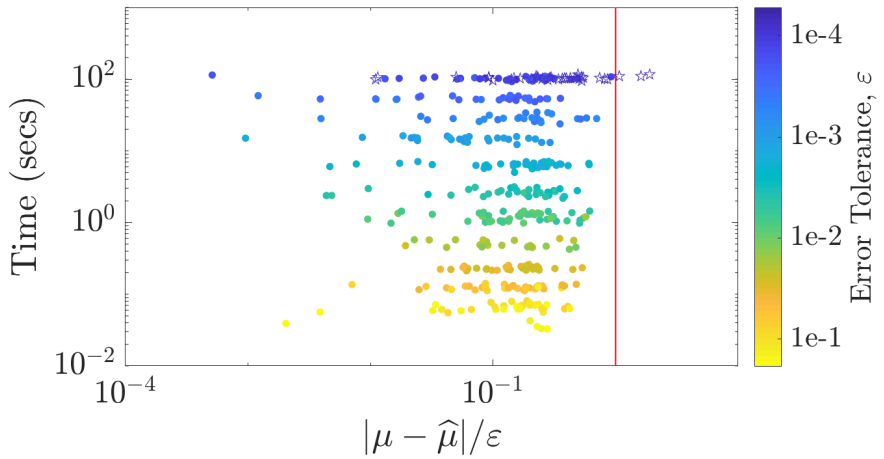


**Fig. 16** Option pricing using the GCV stopping criterion. The hollow stars indicate the algorithm has not met the error threshold $\epsilon$ even with using maximum $n$.

As shown in Figures **??** to 16, both the algorithms computed the integral within user specified threshold most of the time 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 that the `cubBayesLattice_g` algorithm finished within 10 seconds for Keister and multivariate Gaussian. Option pricing took closer to 70 seconds 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 the 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 that 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 $\varepsilon = 10^{-4}$. The algorithm reached max allowed number of samples, $n = 2^{20}$ due to the complexity of the integrand.

### 6.6 Comparison with `cubMC_g`, `cubLattice_g` and `cubSobol_g`

GAIL library provides variety of numerical integration algorithms based on different theoretical foundations, We would like to compare how our algorithms perform relatively to these. 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.

*Keister integral* The Table 1 summarizes the performance of the methods MC, Lattice, Sobol, BayesLat, and BayesSob—which refer to the GAIL cubatures, `cubMC_g`, `cubLattice_g`, `cubSobol_g`, `cubBayesLattice_g`, `cubBayesNet_g`, respectively for estimating Keister integral defined in (38). We conducted two simulations with $d = 3$ and 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 but was slower than `cubSobol_g`.

JR: avoid Exp notation for 2 decimals

**Table 1** Comparison of average performance of cubatures for estimating the Keister integral (38) 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 |

*Multivariate Gaussian* Table 2 summarizes the performance of the methods MC, Lattice, Sobol, BayesLat, and BayesSob for estimating the multi-dimensional Gaussian probability $\mathbf{X} \sim \mathbf{N}(\mu, \mathbf{\Sigma})$. This experiment demonstrates our algorithm's ability to handle high-dimensional integral.

We conducted two simulations with different $\Sigma$ 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 `cubBayesNet_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 was slower than `cubSobol_g`.

**Table 2** Comparison of average performance of cubatures for estimating the $d = 20$ Multivariate Normal (**??**) for 1000 independent runs with $\varepsilon = 10^{-3}$. These results can be conditionally reproduced with the script, `MVNCubatureExampleBayes.m`, in GAIL.

| $\Sigma = \mathsf{I}_d,\ \boldsymbol{b} = -\boldsymbol{a} = (3.5, \cdots, 3.5)$ | | | | | |
|---|---|---|---|---|---|
| Method | MC | Lattice | Sobol | BayesLat | BayesSobol |
| Absolute Error | $2.20 \times 10^{-16}$ | $2.70 \times 10^{-14}$ | $2.70 \times 10^{-14}$ | $2.20 \times 10^{-16}$ | $2.20 \times 10^{-16}$ |
| 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 |

| $\Sigma = 0.4\ \mathsf{I}_d + 0.6\ \mathbf{1}\mathbf{1}^T,\ \boldsymbol{a} = (-\infty, \cdots, -\infty),\ \boldsymbol{b} = \sqrt{d}(U_1, \cdots, U_d)$ | | | | | |
|---|---|---|---|---|---|
| Method | MC | Lattice | Sobol | BayesLat | BayesSobol |
| Absolute Error | $2.30 \times 10^{-4}$ | $2.10 \times 10^{-4}$ | $4.40 \times 10^{-4}$ | $1.00 \times 10^{-4}$ | $4.80 \times 10^{-5}$ |
| 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 |

*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 as in the Option Pricing example we demonstrated. We demonstrate this advantage by integrating a function that is not symmetric across dimensions,

$$f(\boldsymbol{x}) = \sum_{j=1}^{d} \upsilon_j \sin(2\pi x_j^2) \tag{39}$$

which has known integral

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

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 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 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 multivariate, so the magnitude of shape parameter depends on the integrand's magnitude in each dimension. We have chosen an integrand particularly to demonstrate this aspect (39) 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 varies significantly across dimensions.

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

## 7 Conclusion and future work

*TBD*

## References


J. F. Baldeaux. *Higher order nets and sequences*. PhD thesis, The School of Mathematics and Statistics at The University of New South Wales, June 2010.

M. Beckers and A. Haegemans. Transformation of integrands for lattice rules. In T. O. Espelid and A. C. Genz, editors, *Numerical Integration: Recent Developments, Software and Applications*, pages 329–340. Kluwer Academic Publishers, Dordrecht, 1992.

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

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

F.-X. Briol, C. J. Oates, M. Girolami, M. A. Osborne, and D. Sejdinovic. Probabilistic integration: A role in statistical computation? *Statist. Sci.*, 2019. to appear.

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. URL `http://gailgithub.github.io/GAIL_Dev/`.

L. L. Cristea, J. Dick, G. Leobacher, and F. Pillichshammer. The tent transformation can improve the convergence rate of quasi-Monte Carlo algorithms using digital nets. *Numer. Math.*, 105:413–455, 2007.

P. Diaconis. Bayesian numerical analysis. In S. S. Gupta and J. O. Berger, editors, *Statistical Decision Theory and Related Topics IV, Papers from the 4th Purdue Symp., West Lafayette, Indiana 1986*, volume 1, pages 163–175. Springer-Verlag, New York, 1988.

J. Dick. Walsh spaces containing smooth functions an quasi-Monte Carlo rules of arbitrary high order. *SIAM J. Numer. Anal.*, 46(1519–1553), 2008.

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

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

P. Glasserman. *Monte Carlo Methods in Financial Engineering*, volume 53 of *Applications of Mathematics*. Springer-Verlag, New York, 2004.

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

F. J. Hickernell. The trio identity for quasi-Monte Carlo error analysis. In P. Glynn and A. Owen, editors, *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Stanford, USA, August 2016*, Springer Proceedings in Mathematics and Statistics, pages 3–27. Springer-Verlag, Berlin, 2018. doi: 10.1007/978-3-319-91436-7.

F. J. Hickernell and Ll. A. Jiménez Rugama. Reliable adaptive cubature using digital sequences. In R. Cools and D. Nuyens, editors, *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, volume 163 of *Springer Proceedings in Mathematics and Statistics*, pages 367–383. Springer-Verlag, Berlin, 2016. arXiv:1410.8615 [math.NA].

F. J. Hickernell and R. X. Yue. The mean square discrepancy of scrambled $(t, s)$-sequences. *SIAM J. Numer. Anal.*, 38:1089–1112, 2000. doi: 10.1137/S0036142999358019.

F. J. Hickernell, Ll. A. Jiménez Rugama, and D. Li. Adaptive quasi-Monte Carlo methods for cubature. In J. Dick, F. Y. Kuo, and H. Woźniakowski, edi-

tors, *Contemporary Computational Mathematics — a celebration of the 80th birthday of Ian Sloan*, pages 597–619. Springer-Verlag, 2018. doi: 10.1007/978-3-319-72456-0.

H. S. Hong and F. J. Hickernell. Algorithm 823: Implementing scrambled digital nets. *ACM Trans. Math. Software*, 29:95–109, 2003. doi: 10.1145/779359.779360.

R. Jagadeeswaran and F. J. Hickernell. Fast automatic Bayesian cubature using lattice sampling. *Stat. Comput.*, 29:1215–1229, 2019. doi: 10.1007/s11222-019-09895-9.

Ll. A. Jiménez Rugama and F. J. Hickernell. Adaptive multidimensional integration based on rank-1 lattices. In R. Cools and D. Nuyens, editors, *Monte Carlo and Quasi-Monte Carlo Methods: MCQMC, Leuven, Belgium, April 2014*, volume 163 of *Springer Proceedings in Mathematics and Statistics*, pages 407–422. Springer-Verlag, Berlin, 2016. arXiv:1411.1966.

B. D. Keister. Multidimensional quadrature algorithms. *Computers in Physics*, 10:119–122, 1996. doi: 10.1063/1.168565.

A. Keller. Quasi-Monte Carlo image synthesis in a nutshell. In J. Dick, F. Y. Kuo, G. W. Peters, and I. H. Sloan, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2012*, volume 65 of *Springer Proceedings in Mathematics and Statistics*, pages 213—249. Springer Berlin Heidelberg, 2013.

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*, 16(6):1631–1696, 2016.

D. Laurie. Periodizing transformations for numerical integration. *J. Comput. Appl. Math.*, 66:337—344, 1996.

J. Matoušek. On the $L_2$-discrepancy for anchored boxes. *J. Complexity*, 14:527–556, 1998.

H. Niederreiter. Constructions of $(t, m, s)$-nets and $(t, s)$-sequences. *Finite Fields Appl.*, 11:578–600, 2005.

D. Nuyens. URL `https://people.cs.kuleuven.be/~dirk.nuyens/`.

D. Nuyens. The construction of good lattice rules and polynomial lattice rules. Aug 2013.

A. O'Hagan. Bayes-Hermite quadrature. *J. Statist. Plann. Inference*, 29:245–260, 1991. doi: 10.1016/0378-3758(91)90002-V.

A. B. Owen. Randomly permuted $(t, m, s)$-nets and $(t, s)$-sequences. In H. Niederreiter and P. J.-S. Shiue, editors, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 106 of *Lecture Notes in Statistics*, pages 299–317. Springer-Verlag, New York, 1995.

A. O'Hagan. Bayes-hermite quadrature. *Journal of Statistical Planning and Inference*, 29(3):245–260, 1991.

C. E. Rasmussen and C. Williams. Bayesian Monte Carlo. In S. Thrun, L. K. Saul, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 489 – 496. MIT Press, 2003.

J. Rathinavel. *Fast automatic Bayesian cubature using matching kernels and designs*. PhD thesis, Illinois Institute of Technology, 2019.

K. Ritter. *Average-Case Analysis of Numerical Problems*, volume 1733 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 2000.

A. Sidi. A new variable transformation for numerical integration. In H. Brass and G. Hämmerlin, editors, *Numerical Integration IV*, number 112 in International Series of Numerical Mathematics, pages 359–373. Birkhäuser, Basel, 1993.

A. Sidi. Further extension of a class of periodizing variable transformations for numerical integration. *J. Comput. Appl. Math.*, 221:132–149, 2008.

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.*, 7:86–112, 1967.

I. M. Sobol'. Uniformly distributed sequences with an additional uniformity property. *Zh. Vychisl. Mat. i Mat. Fiz.*, 16:1332–1337, 1976.