Noname manuscript No.
(will be inserted by the editor)

Fast automatic Bayesian cubature using Lattice sampling

R. Jagadeeswaran · Fred. J. Hickernell

Received: date / Accepted: date

Abstract Automatic cubatures provide approximations to multidimensional integrals that satisfy user-specified error tolerances. For multidimensional problems, the sampling density is fixed, but the sample size, n, is determined automatically. Bayesian cubature postulates that the integrand is an instance of a stochastic process. Prior information about mean and covariance of this process is used to form data-driven error bounds. However, the process of inferring the mean and covariance governing the stochastic process from n integrand values involves computing matrix inverses and determinants, which are in general time-consuming $O(n^3)$ operations. Our work employs low discrepancy data sites and matching kernels that lower the computational cost to $O(n \log n)$. The confidence interval for the Bayesian posterior error is used to choose n automatically to satisfy the user-defined error tolerance. This approach is demonstrated using rank-1 lattice sequences and shift-invariant kernels.

Keywords Bayesian cubature · Probabilistic numeric methods · GAIL

1 Introduction

Cubature is the problem of inferring a numerical value integral $\mu = \int g(x)\,\mathrm{d}x$ of a multi-dimensional function f where μ has no closed form analytic expression. Typically, g is only accessible in the form of a black-box function

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

S. Author
second address

routine. Cubature is a key component of many problems in scientific computing, finance, statistical modeling, and machine learning.

The integral μ is typically expressed in the form

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

where $f : \mathcal{X} \to \mathbb{R}$ is the integrand and ν is a probability measure defined on the measurable set $\mathcal{X} \subseteq \mathbb{R}^d$. The problem of choosing f and ν well to correspond to the original integrand, g, is the matter of importance sampling, which we do not address here. The cubature algorithm is defined as

$$\hat{\mu}(f) := w_0 + \sum_{i=1}^{n} f(\boldsymbol{x}_i) w_i = \int_{\mathcal{X}} f(\boldsymbol{x})\, \hat{\nu}(\mathrm{d}\boldsymbol{x}), \tag{2}$$

where the weights, w_0, and $\boldsymbol{w} = \{w_i\}_{i=1}^n$, and the nodes $\{x_i\}_{i=1}^n$ are chosen to make the error, $|\mu(f) - \hat{\mu}(f)|$, small.

Our concern is constructing 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, i.e.,

$$|\mu(f) - \hat{\mu}(f)| \le \epsilon$$

Rather than relying on strong assumptions about the integrand, such as its variance or total variation, we construct a stopping criterion that is data-driven, and based on a credible interval arising by a Bayesian assumption on the problem. We build upon the ideas of Diaconis [Dia88], O'Hagan [O'H91], Ritter [Rit00], Rasmussen and Ghahramani [RG03], and others. Our contribution here is to demonstrate how the choice of a family of kernels that match the low discrepancy sampling nodes facilitates fast computation of the data-driven stopping criterion. If n function values are obtained for cubature purposes, then $\mathcal{O}(n \log(n))$ additional operations are required to check whether the error tolerance is satisfied. This is significantly fewer operations than the $\mathcal{O}(n^3)$ typically required for Bayesian cubature.

Traditional cubature methods assume the integrand to be a deterministic function. An alternative to this approach is to assume that the integrand is a *stochastic process*. Bayesian cubature (BC) methods assume the integrand is an instance of a Gaussian process. BC methods approximate the multivariate integrals that cannot be evaluated analytically, using weighted sums of integrand values at carefully chosen nodes over the domain $\mathcal{X}$ as shown in eqn (2).

Traditional cubature methods may not provide any guaranteed error accuracy. The goal of this work is to develop a guaranteed, BC algorithm using the confidence intervals given by Bayesian posterior error. Our algorithm strives to find the minimum sample size 'n' - *number of integrand values* required so that the error $|\mu(f) - \hat{\mu}(f)|$ is less than the user-defined error threshold ϵ, i.e,

$$|\mu(f) - \hat{\mu}(f)| \le \epsilon$$

But the true error $|\mu(f) - \hat{\mu}(f)|$ will not be available for problems that do require cubature methods. So we compute an approximate error bound err_n

using the confidence interval obtained from the posterior error which meets,

$$\mathrm{err}_n \le |\mu(f) - \hat{\mu}(f)|$$

By calculating a data-driven error bound, our algorithm adaptively determines 'n'. An added advantage of our approach is being data-driven, so it does not require any other parameter of the integrand like the 'total variation'.

The "algorithms for numerical tasks" that return uncertainties in their calculations are called *probabilistic numeric methods*. Our BC algorithm can be categorized as a probabilistic numeric method, due to the nature of assumptions made and the confidence interval it provides.

Section 2 introduces the Bayesian approach to estimate the posterior error and develops the concepts to derive error bound. Then formulates the Automatic cubature algorithm using the error bound. Finally demonstrates why directly using Bayesian cubature algorithm is computationally very expensive. Section 3 Introduces the concept of Fast transform kernel and develops the concepts to make the Bayesian cubature faster which is the major contribution of this research. Section 4 Demonstrates the realization of Fast transform kernel using a shift-invariant kernel and Rank-1 Lattice points. Section 5 covers further enhancements to the algorithm to avoid cancellation error and make it faster. Finally, numerical examples are shown using the faster algorithm developed.

2 Bayesian cubature

This section introduces the concepts of the Bayesian cubature and derives all the basic results which will be used in the next section for further speedup of the algorithm. *Bayesian posterior error* [Hic17] is used to estimate an error bound for the cubature.

2.1 Bayesian posterior error

Random f postulated by Diaconis [Dia88], O'Hagen [O'H91], Ritter [Rit00], Rasmussen and Ghahramani [RG03] and others: $f \sim \mathcal{GP}(m, s^2 C_{\boldsymbol{\theta}})$, a Gaussian process from the sample space $\mathcal{F}$ with mean m and covariance function $s^2 C_{\boldsymbol{\theta}}$, $C_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. In this work, we use $\mathcal{X} = [0,1]^d$ and uniform measure. The scale parameter s^2 and shape parameter $\boldsymbol{\theta}$ should be estimated.

For a Gaussian process, all vectors of linear functionals of f have a multivariate Gaussian distribution. Sometimes we drop $\boldsymbol{\theta}$ in the writings to simplify the notation. With this assumption,

$$\mu(f) \sim \mathcal{N}(m\mu(1), s^2 c_0), \quad c_0 = \int_{[0,1]^2} C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t})\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{t},$$
$$\boldsymbol{f} = (f(\boldsymbol{x}_i))_{i=1}^n \sim \mathcal{N}(m\mathbf{1}, s^2\mathsf{C}), \quad \mathsf{C} = (C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1}^n,$$
$$\hat{\mu}(f) = w_0 + \boldsymbol{w}^T \boldsymbol{f} \sim \mathcal{N}(w_0 + m\boldsymbol{w}^T\mathbf{1}, s^2 \boldsymbol{w}^T \mathsf{C}\boldsymbol{w}), \quad \mathbf{1} = (1, ..., 1)^T.$$

Let us consider a fixed node set design $\{\boldsymbol{x}_i\}_{i=1}^n$, then we observe $\boldsymbol{y} = \{y_i = f(\boldsymbol{x}_i)\}_{i=1}^n$. With this, the conditional probability density of error $\mu - \hat{\mu}$ given observed data $\boldsymbol{y}$:

$$\mu - \hat{\mu} \,|\boldsymbol{y} \;\sim\; \mathcal{N}\begin{pmatrix} -w_0 + m(\mu(1) - \mathbf{1}^T \mathsf{C}^{-1}\boldsymbol{c}) + \boldsymbol{y}^T(\mathsf{C}^{-1}\boldsymbol{c} - \boldsymbol{w}), \\ s^2(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c}) \end{pmatrix}, \tag{3}$$

$$\text{where} \quad \boldsymbol{c} = \left(\int_{[0,1]} C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}) \mathrm{d}\boldsymbol{x} \right)_{i=1}^{n}$$

By choosing the weights w_0 and $\boldsymbol{w}$ carefully:

$$w_0 = m(\mu(1) - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c}), \quad \boldsymbol{w} = \mathsf{C}^{-1}\boldsymbol{c},$$

we can force the posterior error $\mu - \hat{\mu} \,|\boldsymbol{y}$ to have zero mean:

$$\mu - \hat{\mu} \,|\boldsymbol{y} \;\sim\; \mathcal{N}\left(0,\; s^2(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})\right),$$

which leads to unbiased solution:

$$\hat{\mu}(f) = w_0 + \boldsymbol{w}^T\boldsymbol{y} = m(\mu(1) - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c}) + \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}$$

If we can assume the observed data comes from the source of Gaussian process with zero mean $m = 0$ then $w_0 = 0$; This fact can be used where applicable. For the integration problem as defined eqn (1), which is the focus of this work, $\mu(1) = 1$. So, further in this work, we use the actual value instead.

Subsequently, since the posterior error $\mu - \hat{\mu} \,|\boldsymbol{y}$ is a Normal random variable, we can obtain a confidence interval or inference using integrand-samples and estimated-parameters. If n is chosen large enough to make

$$\mathrm{err}_n := 2.58\, \sqrt{s^2(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})} \le \epsilon \tag{4}$$

Then

$$\mathbb{P}_f\left[|\mu - \hat{\mu}| \le \epsilon\right] \ge 99\% \tag{5}$$

where the constant 2.58 comes from the fact that 99% standard Normal distribution is comprised within 2.58 times the standard deviation. We call err_n, error bound and $\mathrm{err}_n \le \epsilon$, stopping criterion.

2.2 Maximal likelihood estimation of parameters

The covariance scale parameter s^2, mean m and kernel shape parameter $\boldsymbol{\theta}$ must be estimated. We choose to do this estimation through Maximum likelihood estimation (MLE), by using the observed integrand values for the purpose of estimating the integral. The log-likelihood function of the parameters given the data $\boldsymbol{y} = \{f(\boldsymbol{x}_i)\}_{i=1}^n$ is:

$$\begin{aligned} l(s, m, \boldsymbol{\theta}|\boldsymbol{y}) &= \log\left(\frac{\exp\left(-\frac{1}{2}s^{-2}(\boldsymbol{y} - m\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m\mathbf{1})\right)}{\sqrt{(2\pi)^n \det(s^2\mathsf{C})}} \right) \\ &= -\frac{1}{2}s^{-2}(\boldsymbol{y} - m\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y} - m\mathbf{1}) - \frac{1}{2}\log(\det\,\mathsf{C}) - n\log(s) + \text{constants}. \end{aligned} \tag{6}$$

Maximizing $l(s, m, \boldsymbol{\theta}|\boldsymbol{y})$ with respect to m, provides the MLE of m:

$$m_{\mathrm{MLE}} = \frac{\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} \tag{7}$$

Then using this result,

$$\begin{aligned}(\boldsymbol{y}-m\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y}-m\mathbf{1}) &= \boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y} - \frac{\boldsymbol{y}^T\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{y}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}} \\ &= \boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}.\end{aligned}$$

Maximizing $l(s,m,\boldsymbol{\theta}|\boldsymbol{y})$ with respect to 's' provides the MLE of s:

$$\begin{aligned}s^2_{\text{MLE}} &= \frac{1}{n}(\boldsymbol{y}-m_{\text{MLE}}\mathbf{1})^T\mathsf{C}^{-1}(\boldsymbol{y}-m_{\text{MLE}}\mathbf{1}) \\ &= \frac{1}{n}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}. \qquad (8)\end{aligned}$$

Plug in the results of m_{MLE} and s_{MLE} to simplify the log likelihood:

$$l(s,m,\boldsymbol{\theta}|\boldsymbol{y}) = -\frac{1}{2}n - \frac{1}{2}\log(\det\mathsf{C}) - n\log(s_{\text{MLE}}) + \text{const}$$

By minimizing the negative of $l(s,m,\boldsymbol{\theta}|\boldsymbol{y})$ with respect to '$\boldsymbol{\theta}$', the MLE of $\boldsymbol{\theta}$ can be obtained. This is usually done by numerically searching for the minimum:

$$\boldsymbol{\theta}_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\left[\frac{1}{2n}\log(\det\mathsf{C}) + \log\left(s_{\text{MLE}}\right)\right]. \qquad (9)$$

Using those two MLE results of m and s, the simplified stopping criterion becomes:

$$2.58\sqrt{\frac{1}{n}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})} \le \epsilon.$$

Finally, using the MLE estimated parameters $m_{\text{MLE}}, s_{\text{MLE}}$ and $\boldsymbol{\theta}_{\text{MLE}}$, we compute:

$$\hat{\mu}(f) = \underbrace{m_{\text{MLE}}(1 - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c})}_{w_0} + \underbrace{\boldsymbol{c}^T\mathsf{C}^{-1}}_{\boldsymbol{w}}\boldsymbol{y} = \left(\frac{(1-\mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c})}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\mathsf{C}^{-1}\mathbf{1} + \mathsf{C}^{-1}\boldsymbol{c}\right)^T\boldsymbol{y}$$

2.3 Formulating the automatic Bayesian cubature algorithm

Let the function to integrate be $f \sim \mathcal{GP}(m, s^2C_{\boldsymbol{\theta}})$. Our goal is to compute $\hat{\mu}$ within the error tolerance ϵ, i.e.,

$$|\mu - \hat{\mu}| \le \epsilon.$$

It is not possible to know true error $|\mu - \hat{\mu}|$ for real problems. So, the error-bound err_n is used as the approximate error estimate. Basic principle of our algorithm is to keep adding more function values till the stopping criterion is met, i.e., $\text{err}_n \le \epsilon$. Once the stopping criterion is met, the approximation of μ can be computed:

$$\hat{\mu}_n = w_0 + \boldsymbol{w}^T\boldsymbol{y}, \qquad (10)$$

Where the suffix n is used to imply the number of integrand-samples used. In every iteration of the *automatic cubature algorithm (1)*, we only need to numerically estimate the MLE of $\boldsymbol{\theta}$ and then use it to compute C and err_n.

The following pseudo-code briefly explains the working of the automatic cubature algorithm.

Algorithm 1

```
 1: procedure AutoCubature($f, \epsilon$)                ▷ Integrate within the error tolerance
 2:     $n_0 \leftarrow 2^8$                                  ▷ start with minimum number of points
 3:     $n \leftarrow n_0,\ n' \leftarrow 0$
 4:     while true do                                    ▷ Iterate till error tolerance is met
 5:         Generate $\{\boldsymbol{x}_i\}_{i=n'+1}^{n}$ and sample $\{f(x_i)\}_{i=n'+1}^{n}$
 6:         Compute error bound $err_n$                  ▷ $\mathrm{err}_n$ data driven error bound
 7:         if $err_n \leq \epsilon$ then break
 8:         end if
 9:         $n' \leftarrow n, n \leftarrow 2 \times n'$
10:     end while
11:     Compute cubature weights $\{w_i\}_{i=1}^{n}$
12:     Compute approximate integral $\hat{\mu}_n$
13:     return $\hat{\mu}_n$                               ▷ Integral estimate $\hat{\mu}_n$
14: end procedure
```

As shown, the algorithm continues the iteration loop till the stopping-criterion is met, i.e., err_n is smaller than error threshold ϵ. At the end of every iteration, if the computed error bound err_n is higher than the required ϵ, algorithm doubles the number of points n and repeats. When the error tolerance ϵ is met, exits the loop. Finally using the n and the MLE estimated parameters, computes the $\hat{\mu}_n$.

Accurately computing the data-driven error bound err_n which closely matches the true error is essential for the effectiveness of our algorithm. So, accurate and faster computation of err_n and $\hat{\mu}_n$ are the main objective of the following sections.

2.4 Example with Matern kernel

We would like to demonstrate and test numerical accuracy and computational cost of the automatic Bayesian cubature algorithm using the results obtained so far. For this example, we use the Matern kernel:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) = \prod_{k=1}^{d} \exp(-\theta_k |\boldsymbol{x}_k - \boldsymbol{t}_k|)(1 + \theta_k |\boldsymbol{x}_k - \boldsymbol{t}_k|) \tag{11}$$

On our test computer with Intel i7 3630QM and 16GB RAM memory, it took 118 minutes to compute $\hat{\mu}_n$ with $n = 2^{13}$. As shown in Figure 1, computation time increases rapidly with n. Especially, Maximum-likelihood-estimation of $\boldsymbol{\theta}$ which needs the loss function, is the most time consuming of all. Because the loss functions need to be computed multiple times in every iteration to choose the optimal shape parameter till its minimum is found. Not only the complexity increases, also the kernel matrix becomes highly ill-conditioned with increasing n number of data-points. We must use alternative techniques to overcome this problem. So, our algorithm in the current form is not straightaway usable for any practical applications.

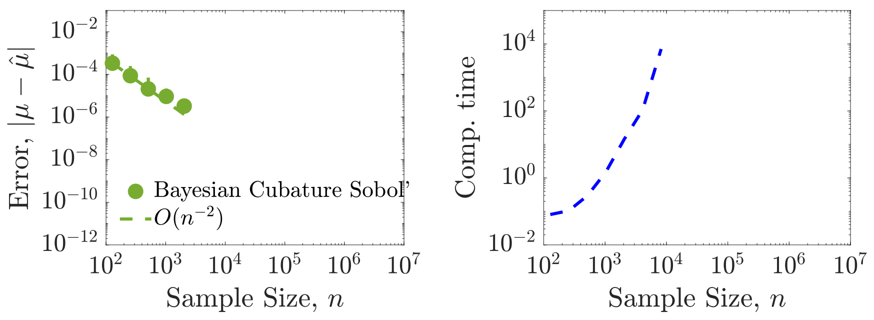

Fig. 1: MVN for d=2 with Matern kernel

3 Fast automatic Bayesian cubature

Automatic Bayesian cubature algorithm described (Section 2.3) is slow due to its computationally intensive nature. This could be a hindrance in real world applications when a larger number of data points are needed to achieve the desired accuracy. This section and further sections explore techniques to make it faster.

Bayesian cubature algorithm uses error-bound err_n to decide when to stop, and uses MLE loss function (9) to find the optimal shape parameter. These computations involve covariance matrix inversions and multiplications, which are time-consuming and prone to numerical error. When the algorithm tries to adapt by increasing n to get better accuracy, the covariance matrix size also grows, along with it, the matrix's condition number also grows significantly causing numerical error in matrix operations. This phenomenon is called ill-conditioning.

One approach to overcome these issues could be to use an efficient method to compute the whole equation without incurring the ill-conditioning, which involves, for example, using *stable computation* [Fas07]. But it is going to be still slower and time-consuming.

Another approach for stable and faster computation is, using a carefully chosen kernel with some special properties which leads to faster matrix operations, like faster matrix inversion and multiplication. This approach can be designed to be faster in computation while avoiding the numerical error due to ill-conditioning. The later approach is the major objective of this research work. Our approach is especially to choose a special form of kernel, which is called *fast transform kernel* along with a suitable pointset, that will avoid expensive matrix operations.

3.1 Fast transform kernel

Suppose the domain is $\mathcal{X} = [0,1]^d$ and the probability measure is uniform, If the kernel $C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t})$ is chosen carefully along with appropriate point set $\{\boldsymbol{x}_i\}_{i=1}^n$, have the special properties, so the resulting Gram matrix has factorization of the form:

$$\begin{aligned} \mathsf{C} &= \Big(C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j)\Big)_{i,j=1}^{n} = (\boldsymbol{C}_1, ..., \boldsymbol{C}_n) \\ &= \frac{1}{n}\mathsf{V}\mathsf{\Lambda}\mathsf{V}^H = \frac{1}{n}\mathsf{V}^*\mathsf{\Lambda}\mathsf{V}^T, \qquad \mathsf{V}^H = n\mathsf{V}^{-1}, \end{aligned} \tag{12}$$

Where V^H is the Hermitian of V,

$$\mathsf{V} = (\boldsymbol{v}_1, ..., \boldsymbol{v}_n)^T = (\boldsymbol{V}_1, ..., \boldsymbol{V}_n), \text{ also } \boldsymbol{v}_1 = \boldsymbol{V}_1 = \boldsymbol{1}$$
$$\mathsf{\Lambda} = \operatorname{diag}(\boldsymbol{\lambda}), \quad \boldsymbol{\lambda} = (\lambda_1, ...\lambda_n),$$
$$\mathsf{V}^{-1} = \frac{1}{n}\mathsf{V}^H, \; \mathsf{C}^{-1} = \frac{1}{n}\mathsf{V}\mathsf{\Lambda}^{-1}\mathsf{V}^H = \frac{1}{n}\mathsf{V}^*\mathsf{\Lambda}^{-1}\mathsf{V}^T.$$

Suppose the transform $\hat{\boldsymbol{z}} = \mathsf{V}^T\boldsymbol{z}$ for any arbitrary $\boldsymbol{z}$ can be computed within the computational cost of $\mathcal{O}(n \, log \, n)$, we call it a *Fast transform*. Consequently $C_{\boldsymbol{\theta}}$ is called a *fast transform kernel*. We use the notation $\hat{\boldsymbol{z}}$ to indicate the transform of $\boldsymbol{z}$. The properties of fast transform can be leveraged to make the computations in Bayesian cubature algorithm faster. To begin with, $\boldsymbol{\lambda}$ can be computed faster by simply,

$$\mathsf{V}^T\boldsymbol{C}_1 = \mathsf{V}^T\left(\frac{1}{n}\mathsf{V}^*\mathsf{\Lambda}\boldsymbol{v}_1\right) = \underbrace{\left(\frac{1}{n}\mathsf{V}^T\mathsf{V}^*\right)}_{\mathsf{I}}\mathsf{\Lambda}\boldsymbol{v}_1 = \mathsf{\Lambda}\boldsymbol{1} = \begin{pmatrix}\lambda_1\\ \vdots \\ \lambda_n\end{pmatrix} = \boldsymbol{\lambda} \tag{13}$$

Most of the computations in the algorithm are of the form $\boldsymbol{a}^T\mathsf{C}^{-1}\boldsymbol{b}$, can be simplified easily:

$$\mathsf{C}V_1 = \lambda_1 V_1, \quad \text{by definition}$$
$$\text{So,} \quad \mathsf{C}^{-1}\boldsymbol{1} = \mathsf{C}^{-1}V_1 = \frac{V_1}{\lambda_1} = \frac{\boldsymbol{1}}{\lambda_1},$$
$$\text{similarly,} \quad \boldsymbol{a}^T\mathsf{C}^{-1}\boldsymbol{b} = \frac{1}{n}\boldsymbol{a}^T\mathsf{V}\mathsf{\Lambda}^{-1}\mathsf{V}^H\boldsymbol{b} = \frac{1}{n}\widehat{\boldsymbol{a}}^T\mathsf{\Lambda}^{-1}\widehat{\boldsymbol{b}}^* = \frac{1}{n}\sum_{i=1}^{n}\frac{\widehat{a_i}\widehat{b_i^*}}{\lambda_i},$$

where $\widehat{\boldsymbol{a}} = \mathsf{V}^T\boldsymbol{a}$ and $\widehat{\boldsymbol{b}} = \mathsf{V}^T\boldsymbol{b}$. By using this, some of the recurring terms in the algorithm are simplified,

$$\begin{aligned} \boldsymbol{1}^T\mathsf{C}^{-1}\boldsymbol{1} &= \boldsymbol{1}^T\left(\frac{\boldsymbol{1}}{\lambda_1}\right) = \frac{n}{\lambda_1} \\ \boldsymbol{1}^T\mathsf{C}^{-1}\boldsymbol{y} &= \boldsymbol{1}^T\left(\frac{\boldsymbol{y}}{\lambda_1}\right) = \frac{\sum_{i=1}^n y_i}{\lambda_1} = \frac{\widehat{y_1}}{\lambda_1} \\ \boldsymbol{y}^T\mathsf{C}^{-1}\boldsymbol{y} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{y_i}|^2}{\lambda_i} \\ \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{1} &= \boldsymbol{c}^T\left(\frac{\boldsymbol{1}}{\lambda_1}\right) = \frac{\widehat{c}_1}{\lambda_1} \\ \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{c_i}|^2}{\lambda_i} \end{aligned}$$

Then the MLE estimates can be simplified using these results:

$$m_{\mathrm{MLE}} = \frac{\mathbf{1}^T \mathsf{C}^{-1}\boldsymbol{y}}{\mathbf{1}^T \mathsf{C}^{-1}\mathbf{1}} = \frac{\widehat{y}_1}{n} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

$$s^2_{\mathrm{MLE}} = \frac{1}{n}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right] = \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{y}_i|^2}{\lambda_i} - \frac{|\widehat{y}_1|^2}{n\lambda_1} = \frac{1}{n}\sum_{i=2}^{n}\frac{|\widehat{y}_i|^2}{\lambda_i}$$

3.2 Application of fast transform kernel

Using the properties of the fast transform, matrix multiplications and inversions can be avoided, so the overall computation cost is $\mathcal{O}(n\log n)$. To begin with, MLE estimate of $\boldsymbol{\theta}$ can be made faster:

$$\begin{aligned}\boldsymbol{\theta}_{\mathrm{MLE}} &= \operatorname*{argmin}_{\boldsymbol{\theta}}\left[\frac{1}{2n}\log(\det \mathsf{C}) + \log\left(s_{\mathrm{MLE}}\right)\right]\\ &= \operatorname*{argmin}_{\boldsymbol{\theta}}\left[\frac{1}{2n}\sum_{i=1}^{n}\log(\lambda_i) + \log\left(\frac{1}{n}\sum_{i=2}^{n}\frac{|\widehat{y}_i|^2}{\lambda_i}\right)\right] \qquad (14)\\ &\text{where}\quad \hat{\boldsymbol{y}} = (\hat{y}_i)_{i=1}^n = \mathsf{V}^T\boldsymbol{y}.\end{aligned}$$

Similarly, the error bound err_n computation can be made faster:

$$\begin{aligned}\mathrm{err}_n &= 2.58\sqrt{\frac{1}{n}\boldsymbol{y}^T\left[\mathsf{C}^{-1} - \frac{\mathsf{C}^{-1}\mathbf{1}\mathbf{1}^T\mathsf{C}^{-1}}{\mathbf{1}^T\mathsf{C}^{-1}\mathbf{1}}\right]\boldsymbol{y}(c_0 - \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{c})}\\ &= 2.58\sqrt{\frac{1}{n^2}\sum_{i=2}^{n}\frac{|\widehat{y}_i|^2}{\lambda_i}\left(c_0 - \frac{1}{n}\sum_{i=1}^{n}\frac{|\widehat{c}_i|^2}{\lambda_i}\right)},\\ &\text{where}\quad \widehat{\boldsymbol{y}} = \mathsf{V}^T\boldsymbol{y}, \widehat{\boldsymbol{c}} = \mathsf{V}^T\boldsymbol{c}.\end{aligned}$$

Finally, the computation of $\hat{\mu}_n$ is made faster:

$$\begin{aligned}\hat{\mu}_n(f) &= w_0 + \boldsymbol{w}^T\boldsymbol{y} = m[1 - \mathbf{1}^T\mathsf{C}^{-1}\boldsymbol{c}] + \boldsymbol{c}^T\mathsf{C}^{-1}\boldsymbol{y}\\ &= \frac{\widehat{y}_1}{n}\left[1 - \frac{\widehat{c}_1}{\lambda_1}\right] + \frac{1}{n}\sum_{i=1}^{n}\frac{\widehat{c}_i\widehat{y}_i^*}{\lambda_i}\\ &= \frac{\widehat{y}_1}{n} + \frac{1}{n}\sum_{i=2}^{n}\frac{\widehat{c}_i\widehat{y}_i^*}{\lambda_i} \qquad (15)\end{aligned}$$

It is interesting to note that, with $m \neq 0$ assumption, $\hat{\mu}$ is simply the sample mean. Moreover the choice of kernel or any MLE estimated parameters do not affect the $\hat{\mu}_n$ but influences the accuracy of the error bound err_n.

These simplified computations involve no matrix inversion or multiplications. It just uses scalar divisions and multiplications so the computational cost is $\mathcal{O}(n)$. But computation of fast transform $\widehat{\boldsymbol{y}}$ is $\mathcal{O}(n\log n)$. Consequently the overall computation cost is $\mathcal{O}(n\log n)$.

4 Implementation 1: Using a shift invariant kernel for Bayesian cubature

We have established the concept of fast transform and showed how it can make the computations faster. But we assumed there exist a kernel that meets the requirements. Now we are going to show a example. The following shift invariant kernel satisfies the requirements of the fast transform kernel when combined with Rank-1 lattice points:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{t}) := \sum_{\boldsymbol{k}\in\mathbb{Z}^d} \alpha_{\boldsymbol{k},\boldsymbol{\theta}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{x}} e^{-2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{t}}, \tag{16}$$

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

where d is number of dimensions and $\alpha_{\boldsymbol{k}}$ is a scalar. The Gram matrix formed by this kernel is a Hermitian matrix. With some more simplifications:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{t}) = \prod_{l=1}^{d}\left[1+\sum_{k_l=1}^{\infty}\left|\frac{\theta_l}{k_l}\right|^r 2\cos(2\pi\sqrt{-1}k_l(x_l-t_l))\right].$$

The *shape parameter* $\boldsymbol{\theta}$ is used to make the kernel customizable. To be specific, the shape parameter tweaks the kernel, so that the function space spanned by the kernel closely resembles the space where the integrand function belongs.

This form of the kernel is very convenient to use in any analytical derivations or proofs, but not suitable for use with finite precision computers as this kernel involves infinite sum. It is preferred to have a simpler expression of the kernel without infinite sum for practical computations.

4.1 Using Lattice points

Along with the kernel (16), Rank-1 lattice points $\boldsymbol{x}_i \in \mathsf{L}_{n,\boldsymbol{h}}$ are used to get the *fast transform kernel.* In this work, we use the lattice points as defined in [SD06]:

$$\mathsf{L}_{n,\boldsymbol{h}} := \{\boldsymbol{x}_i := \boldsymbol{h}\frac{i-1}{n} \mod 1;\ \ i=1,\dots,n\},$$

Where $\boldsymbol{h}$ is the generating vector. Its dual lattice is defined as:

$$\mathsf{L}^{\perp}_{n,\boldsymbol{h}} := \{\boldsymbol{k}\in\mathbb{Z}^d : \boldsymbol{h}^T\boldsymbol{k} \equiv 0 \pmod n\},$$

With lattice points, the kernel is written as:

$$\begin{aligned} C(\boldsymbol{x}_i,\boldsymbol{x}_j) &= \sum_{\boldsymbol{k}\in\mathbb{Z}^d} \alpha_{\boldsymbol{k}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\frac{\boldsymbol{h}i}{n}} e^{-2\pi\sqrt{-1}\boldsymbol{k}^T\frac{\boldsymbol{h}j}{n}} \\ &= \sum_{\boldsymbol{k}\in\mathbb{Z}^d} \alpha_{\boldsymbol{k}} e^{2\pi\sqrt{-1}\boldsymbol{k}^T\boldsymbol{h}\frac{(i-j)}{n}} \end{aligned}$$

Please note that ' mod 1' need not be used explicitly in $C(\boldsymbol{x}_i,\boldsymbol{x}_j)$ since $e^{\pm 2\pi\sqrt{-1}} = 1$. The kernel (16), when used with Rank-1 Lattice points, leads to symmetric and circulant Gram matrix C. We can demonstrate the resulting Gram matrix satisfies the conditions of a fast transform.

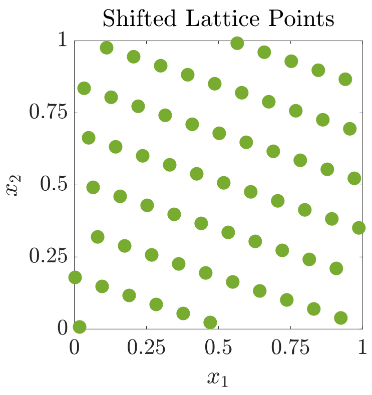

Fig. 2: Randomly shifted Lattice points in d=2

4.2 Computing the kernel using Bernoulli polynomials

The shift-invariant-kernel (16) cannot be computed directly due to it's infinite sum. If the coefficients $\alpha_{\boldsymbol{k}}$ were chosen appropriately, then there exist a direct expression for the kernel in terms of Bernoulli polynomial. We can use the Fourier series expansion properties [DLM12] of Bernoulli polynomial B_r to find the appropriate $\alpha_{\boldsymbol{k}}$. This provides a closed form expression for the kernel without the infinite sum. It also allows to choose the smoothness of kernel, i.e, how fast the coefficients $\alpha_{\boldsymbol{k}}$ in (16) decay. The following very useful expansion is referenced from [DLM12] under eqn (24.8.3):

$$B_r(x) = \frac{-r!}{(2\pi\sqrt{-1})^r} \sum_{\substack{k\neq 0,\\ k=-\infty}}^{\infty} \frac{e^{2\pi\sqrt{-1}kx}}{k^r} \begin{cases} \text{for} \quad r=1, \quad 0<x<1 \\ \text{for} \quad r=2,3,\ldots \quad 0 \le x \le 1 \end{cases} \tag{17}$$

Our goal is to replace the infinite sum of the kernel using Bernoulli polynomials. This would allows the computations to be carried out in any software like Matlab. For 1-D ($d=1$) rewriting (17):

$$\sum_{k\neq 0, k=-\infty}^{\infty} \frac{e^{2\pi\sqrt{-1}k|x|}}{(\frac{k}{\theta})^r} = -\theta^r B_r(|x|) \frac{(2\pi\sqrt{-1})^r}{r!} \quad \text{for} \quad -1 \le x \le 1$$

Comparing this expansion with (16), one can deduce, for $d=1$:

$$C(x,t) = \alpha_0 + \sum_{k\neq 0, k=-\infty}^{\infty} \alpha_k e^{2\pi\sqrt{-1}k|x-t|}, \quad -1 \le x,t \le 1,$$

Where the kernel coefficients $\alpha_{\boldsymbol{k}}$ can be expressed explicitly:

$$\alpha_k = \begin{cases} 1, & k=0 \\ \frac{1}{k^r}, & \text{otherwise} \end{cases},$$

In general for $d>1$, the coefficients $\alpha_{\boldsymbol{k}}$ for $C_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{t})$ is computed using:

$$\alpha_{\boldsymbol{k},\boldsymbol{\theta}} = \frac{1}{\prod_{l=1}^{d} \max(\frac{|k_l|}{\theta_l}, 1)^r},$$

Where $\boldsymbol{\theta}$ is the shape parameter and 'r' is the Bernoulli polynomial. The value of 'r' will be chosen specific to the integrand when using in the cubature algorithm. We keep the value of 'r' to be even positive integer in this work. Using the above-found results, we get the simplified form of the kernel:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{t}) = \prod_{l=1}^{d} 1 - \theta_l^r \frac{(2\pi\sqrt{-1})^r}{r!} B_r(|x_l - t_l|), \qquad 0 \le |x_l - t_l| \le 1, \tag{18}$$

where $\boldsymbol{\theta} \in (0,1]^d$ is the shape parameter, 'r' is the order of the Bernoulli polynomial B_r.

4.3 Eigenvalues and Eigenvectors of C

The kernel's Gram matrix C is circulant. It is generated by the vector:

$$(C_{\boldsymbol{\theta}}(\boldsymbol{x}_1,\boldsymbol{x}_1), C_{\boldsymbol{\theta}}(\boldsymbol{x}_2,\boldsymbol{x}_1), ..., C_{\boldsymbol{\theta}}(\boldsymbol{x}_n,\boldsymbol{x}_1))^T =: \boldsymbol{C}_1, \tag{19}$$

Which is the first row or column of C (since it is a symmetric matrix). By the properties of circulant matrix, the normalized eigenvectors are:

$$\left(1, e^{-2\pi\sqrt{-1}\frac{j}{n}}, e^{-2\pi\sqrt{-1}\frac{2j}{n}}, ..., e^{-2\pi\sqrt{-1}\frac{j(n-1)}{n}}\right)^T, \quad \text{for } j = 0, 1, ..., n-1.$$

The matrix constructed with these eigenvectors is:

$$\mathsf{V} := \left(e^{-2\pi\sqrt{-1}\frac{ij}{n}}\right)_{i,j=0}^{n-1},$$

$$\text{Whereas,} \quad \mathsf{V}^H := \frac{1}{n}\left(e^{2\pi\sqrt{-1}\frac{ij}{n}}\right)_{i,j=0}^{n-1},$$

Where the first column of V, $\boldsymbol{V}_1 = \boldsymbol{1}$. The columns of V are complex exponential vectors independent of the kernel values. But their corresponding eigenvalues are:

$$\mathsf{\Lambda} = (\lambda_j)_{j=1}^n = \mathsf{V}^T\boldsymbol{C}_1 := \mathcal{DFT}\left\{\boldsymbol{C}_1\right\},$$

i.e., eigenvalues of the matrix C are computed by applying DFT over $\boldsymbol{C}_1$. for this kernel, the corresponding fast transform is *Fast Fourier Transform* (FFT). The kernel's Gram matrix C can be written in the factorized form:

$$\mathsf{C} = \frac{1}{n}\mathsf{V}\mathsf{\Lambda}\mathsf{V}^H,$$

This is the exact factorization used for the construction of fast transform. Another requirement $\boldsymbol{V}_1 = \boldsymbol{1}$ is also met. Additionally, computational cost of FFT is $\mathcal{O}(n \log n)$. Thus we can conclude the shift invariant kernel in eqn (16), obeys all the requirements to be considered as a fast transform kernel. So the operation $\mathsf{V}^T z$ is a fast transform. Numerical experiments using this kernel will be shown in the section (4.9).

4.4 Additional assumptions

The shift invariant kernel (16) has some more desirable properties, leading to get $c_0 = 1$ and $\boldsymbol{c} = \boldsymbol{1}$, which will help to make the computations even faster:

$$c_0 = \int_{[0,1]^d \times [0,1]^d} C_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{t}) \mathrm{d}\boldsymbol{x} \mathrm{d}\boldsymbol{t} = 1, \quad \boldsymbol{c} = \left(\int_{[0,1]^d} C_{\boldsymbol{\theta}}(\boldsymbol{x}_i, \boldsymbol{t}) \mathrm{d}\boldsymbol{t} \right)_{i=1}^n = \boldsymbol{1}.$$

This gives simplified error bound:

$$\begin{aligned} \mathrm{err}_n &= 2.58 \sqrt{\frac{1}{n^2} \sum_{i=2}^n \frac{|\widehat{y}_i|^2}{\lambda_i} \left(c_0 - \frac{1}{n} \sum_{i=1}^n \frac{|\widehat{c}_i|^2}{\lambda_i} \right)}, \\ &= 2.58 \sqrt{\frac{1}{n^2} \sum_{i=2}^n \frac{|\widehat{y}_i|^2}{\lambda_i} \left(1 - \frac{n}{\lambda_1} \right)}, \\ &\text{where} \quad \widehat{\boldsymbol{c}} = (n, 0, ..., 0)^T, \quad \widehat{\boldsymbol{y}} = \mathsf{V}^T \boldsymbol{y}, \widehat{\boldsymbol{c}} = \mathsf{V}^T \boldsymbol{c}. \end{aligned}$$

And the cubature:

$$\begin{aligned} \hat{\mu}_n(f) = w_0 + \boldsymbol{w}^T \boldsymbol{y} &= \frac{\widehat{y}_1}{n} + \frac{1}{n} \sum_{i=2}^n \frac{\widehat{c}_i \widehat{y}_i^*}{\lambda_i} \\ &= \frac{\widehat{y}_1}{n}, \quad \text{since} \quad \widehat{\boldsymbol{c}}_i = 0 \ \forall \ i \neq 0 \end{aligned}$$

4.5 Variable transforms

Fast transform kernel technique discussed with shift invariant kernel eqn(16) and lattice points so far, assumes the integrand is periodic and has continuous derivatives on the boundaries of the domain $[0, 1]^d$. Non-periodic functions do not live in the space spanned by the kernel. Variable transformation or periodization transform techniques are typically used to improve the accuracy in multi-dimensional numerical integrations where boundary conditions needs to be enforced. These transformations could be either polynomial, exponential and also trigonometric nature.

$$\text{Baker's} : \tilde{f}(\boldsymbol{t}) = f\left(\left(1 - 2\left|t_j - \frac{1}{2}\right|\right)_{j=1}^{d}\right) \tag{20}$$

$$\text{C0} : \tilde{f}(\boldsymbol{t}) = f\left(\boldsymbol{g}_0\right)\prod_{j=1}^{d} g_0'(t_j), \quad \boldsymbol{g}_0 = \left(g_0(t_j)\right)_{j=1}^{d} \tag{21}$$

$$\text{C1} : \tilde{f}(\boldsymbol{t}) = f\left(\boldsymbol{g}_1\right)\prod_{j=1}^{d} g_1'(t_j), \quad \boldsymbol{g}_1 = \left(g_1(t_j)\right)_{j=1}^{d} \tag{22}$$

$$\text{Sidi's C1} : \tilde{f}(\boldsymbol{t}) = f\left(\boldsymbol{\psi}_2\right)\prod_{j=1}^{d} \psi_2'(t_j), \quad \boldsymbol{\psi}_2 = \left(\boldsymbol{\psi}_2(t_j)\right)_{j=1}^{d} \tag{23}$$

$$\text{Sidi's C2} : \tilde{f}(\boldsymbol{t}) = f\left(\boldsymbol{\psi}_3\right)\prod_{j=1}^{d} \psi_3'(t_j), \quad \boldsymbol{\psi}_3 = \left(\boldsymbol{\psi}_3(t_j)\right)_{j=1}^{d} \tag{24}$$

where

$$\begin{aligned}
g_0(t) &= 3t^2 - 2t^3, & g_0'(t) &= 6t(1-t)) \\
g_1(t) &= t^3(10 - 15t + 6t^2), & g_1'(t) &= 30t^2(1-t)^2 \\
\psi_2(t) &= \left(t - \frac{1}{2\pi}\sin(2\pi t)\right), & \psi_2'(t) &= (1 - \cos(2\pi t)) \\
\psi_3(t) &= \frac{1}{16}\left(8 - 9\cos(\pi t) + \cos(3\pi t)\right), & \psi_3'(t) &= \frac{1}{16}\left(9\sin(\pi t)\pi - \sin(3\pi t)3\pi\right)
\end{aligned}$$

These transforms vary in terms of computational complexity and accuracy, shall be chosen on a need basis. Such as:

1. Baker's : Baker's transform or called tent map in each coordinate. It preserves only continuity but it is easier to compute.
2. C0 : Polynomial transformation only ensures periodicity of function.
3. C1 : Polynomial transformation preserving the first derivative.
4. C1sin : Sidi's transform with Sine, preserving the first derivative. This is, in general, a better option than 'C1'.
5. C2sin : Sidi's transform with Sine, preserving upto second derivative. We use this when smoothness of 'C1sin' is not sufficient and need to preserve upto second derivative.

4.6 Iterative Discrete Fourier transform for function values

Automatic cubature algorithms needs to compute Discrete Fourier transform of function values $\boldsymbol{y} = (f(\boldsymbol{x}_i))_{i=1}^{n}$ in every iteration with newly added points. Recomputing the whole Fourier transform in every iteration can be avoided when using Rank-1 Lattice points by using structural properties of the Lattice points. Discrete Fourier transform is defined as :

$$\mathcal{DFT}\{y\} := \hat{y} = \left(\sum_{j=1}^{n} y_j e^{-\frac{2\pi\sqrt{-1}}{n}(j-1)(i-1)} \right)_{i=1}^{n}$$

In essence:

$$\hat{y}_i = \sum_{j=1}^{n} y_j e^{-\frac{2\pi\sqrt{-1}}{n}(j-1)(i-1)}$$

We could rearrange the sum into even indexed $j = 2l$ and odd indexed $j = 2l + 1$:

$$\hat{y}_i = \underbrace{\sum_{l=1}^{n/2} y_{2l} e^{-\frac{2\pi\sqrt{-1}}{n/2}(l-1)(i-1)}}_{\text{DFT of even-indexed part of } y_i} + e^{-\frac{2\pi\sqrt{-1}}{n}(i-1)} \underbrace{\sum_{l=1}^{n/2} y_{2l+1} e^{-\frac{2\pi\sqrt{-1}}{n/2}(l-1)(i-1)}}_{\text{DFT of odd-indexed part of } y_i},$$

Which shows two separately computed DFTs can be combined to produce single output. For example, the odd indexed were the existing DFT and the even indexed are from the new halve of samples, algorithm wants to add to improve accuracy. We use this concept to avoid recomputing the full length DFT of $\boldsymbol{y}$ in every iteration. In other words, DFT is computed only for the newly added samples in every iteration.

4.7 Overcoming the cancellation error

During the numerical experiments, we noticed numerical cancellation error in the computation of the term, especially for the bigger values $n > 2^{15}$. Cancellation error happens because two almost equal values are subtracted, when they differ only in very high decimal values.

$$\left(c_{0,\boldsymbol{\theta}} - \boldsymbol{c}_{\boldsymbol{\theta}}^T \mathsf{C}^{-1} \boldsymbol{c}_{\boldsymbol{\theta}}\right) = \left(1 - \frac{n}{\lambda_1}\right)$$

In this expression $\frac{n}{\lambda_1}$ almost close to 1. We would like to explore techniques to avoid the subtraction in this computation. Let's recollect the definition of the shift invariant kernel:

$$C(\boldsymbol{x}_i, \boldsymbol{x}_j) = \prod_{k=1}^{d} \left[1 + \theta B(\{x_{i_k} - x_{j_k}\})\right].$$

Let's define:

$$\text{modified kernel} \quad \widetilde{C}(\boldsymbol{x}_i, \boldsymbol{x}_j) = C(\boldsymbol{x}_i, \boldsymbol{x}_j) - 1,$$
$$\text{Gram matrix} \quad \widetilde{\mathsf{C}} = \mathsf{C} - \boldsymbol{1}\boldsymbol{1}^T,$$

Let $(\lambda_1, ..., \lambda_n)$ be the eigenvalues of C. Similarly let $(\tilde{\lambda}_1, ..., \tilde{\lambda}_n)$ be the eigenvalues of $\widetilde{\mathsf{C}}$. As per the definition of the Gram matrix C, the eigenvector corresponding to λ_1 is a vector one $\boldsymbol{1}$. Then:

$$\lambda_1 \boldsymbol{1} = \mathsf{C}\boldsymbol{1} = (\widetilde{\mathsf{C}} + \boldsymbol{1}\boldsymbol{1}^T)\boldsymbol{1} = \tilde{\lambda}_1 \boldsymbol{1} + n\boldsymbol{1},$$

This shows $\tilde{\lambda}_1 = \lambda - n$. In fact for the rest of the values are:

$$\tilde{\lambda}_j = \lambda_j, \; \forall j = 1, ..., n-1. \tag{25}$$

Let $\boldsymbol{v}_j, \forall j = 0, ..., n-1$ be the eigenvectors of C, similarly $\tilde{\boldsymbol{v}}_j, \forall j = 0, ..., n-1$ be the eigenvectors of $\widetilde{\mathsf{C}}$,

$$\boldsymbol{v}_j^T \mathsf{C}\mathbf{1} = \lambda_1 \boldsymbol{v}_j^T \mathbf{1} = \mathbf{1}^T \mathsf{C}\boldsymbol{v}_j = \lambda_j \mathbf{1}^T \boldsymbol{v}_j$$

Since $\lambda_1 \neq \lambda_j$, the above statement implies $\boldsymbol{v}_j^T \mathbf{1} = 0$. This interesting property provides the proof of the eqn. (25).

$$\lambda_j \boldsymbol{v}_j = \mathsf{C}\boldsymbol{v}_j = 1_{n\times n}\boldsymbol{v}_j + \widetilde{\mathsf{C}}\boldsymbol{v}_j = \widetilde{\mathsf{C}}\boldsymbol{v}_j = \tilde{\lambda}_j \boldsymbol{v}_j,$$

Thus proven. Using this result, cancellation error in the computation of err_n can be avoided:

$$\begin{aligned}\left(1 - \frac{n}{\lambda_1}\right) &= \left(1 - \frac{n}{\tilde{\lambda}_1 + n}\right) \\ &= \left(\frac{\tilde{\lambda}_1 + n - n}{\tilde{\lambda}_1 + n}\right) = \left(\frac{\tilde{\lambda}_1}{\tilde{\lambda}_1 + n}\right).\end{aligned}$$

The following technique shows an iterative approach to compute $\tilde{C}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for $d > 1$. This iterative technique is developed using induction:

$$\begin{aligned}
d = 1: \qquad & C_1 = 1 + \theta B(\{x_{i_1} - x_{j_1}\}) = 1 + \tilde{C}_1 \\
d = 2: \qquad & C_2 = [1 + \theta B(\{x_{i_2} - x_{j_2}\})][1 + \tilde{C}_1] \\
& \quad = 1 + \theta B(\{x_{i_2} - x_{j_2}\})[1 + \tilde{C}_1] + \tilde{C}_1 \\
& \quad = 1 + \underbrace{\theta B(\{x_{i_2} - x_{j_2}\})C_1 + \tilde{C}_1}_{\tilde{C}_2} \\
d = 3 \qquad & C_3 = [1 + \theta B(\{x_{i_3} - x_{j_3}\})][1 + \tilde{C}_2] \\
& \quad = 1 + \theta B(\{x_{i_3} - x_{j_3}\})[1 + \tilde{C}_2] + \tilde{C}_2 \\
& \quad = 1 + \theta B(\{x_{i_3} - x_{j_3}\})C_2 + \tilde{C}_2 \\
\vdots \qquad & \\
\forall d > 2 \qquad & C_d = [1 + \theta B(\{x_{i_d} - x_{j_d}\})][1 + \tilde{C}_{d-1}] \\
& \quad = 1 + \theta B(\{x_{i_d} - x_{j_d}\})[1 + \tilde{C}_{d-1}] + \tilde{C}_{d-1} \\
& \quad = 1 + \theta B(\{x_{i_d} - x_{j_d}\})C_{d-1} + \tilde{C}_{d-1}
\end{aligned}$$

4.8 Validating Gaussian process assumption

We begin developing the Bayesian cubature with the assumption that the integrand arises from a Gaussian process. Here is an attempt to validate that assumption. Let,

$$\boldsymbol{f} = (f(\boldsymbol{x}_i))_{i=1}^n \sim \mathcal{N}(m\mathbf{1}, \mathsf{C}), \quad \text{where} \quad \mathsf{C} = \frac{1}{n}\mathsf{V}\Lambda\mathsf{V}^H, \quad \mathsf{V}^H\mathsf{V} = n$$

Then,

$$\begin{aligned}\mathbb{E}\left[\frac{1}{\sqrt{n}}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\boldsymbol{f}\right] &= \frac{1}{\sqrt{n}}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\mathbb{E}[\boldsymbol{f}] \\ &= \frac{1}{\sqrt{n}}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H m\mathbf{1} \\ &= m\sqrt{\frac{n}{\lambda_1}}\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix},\end{aligned}$$

And,

$$\begin{aligned}COV\left[\frac{1}{\sqrt{n}}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\boldsymbol{f}\right] &= \frac{1}{n}\mathbb{E}\left[\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H(\boldsymbol{f}-m\mathbf{1})(\boldsymbol{f}-m\mathbf{1})^T\mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}}\right] \\ &= \frac{1}{n}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\mathbb{E}\left[(\boldsymbol{f}-m\mathbf{1})(\boldsymbol{f}-m\mathbf{1})^T\right]\mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \\ &= \frac{1}{n}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\frac{1}{n}\mathsf{V}\mathsf{\Lambda}\mathsf{V}^H\mathsf{V}\mathsf{\Lambda}^{-\frac{1}{2}} \\ &= \mathsf{I}\end{aligned}$$

Let $\boldsymbol{f}' = \frac{1}{\sqrt{n}}\mathsf{\Lambda}^{-\frac{1}{2}}\mathsf{V}^H\boldsymbol{f}$,

$$\boldsymbol{f}' \sim \mathcal{N}\left(m'\mathbf{1}, 1\right),$$

Where $m' = m\sqrt{\frac{n}{\lambda_1}}$. If we can verify the smaple distribution of $\boldsymbol{f}'$ is approximately $\mathcal{N}\left(m'\mathbf{1}, 1\right)$ by using Normal plots , could validate our assumption.

4.9 Numerical Experiments

Having all the tools and optimization done handy, now we shall run the numerical simulations to see the performance.

4.9.1 Test Functions

The following test functions were used to test the integration speed and accuracy of *Fast Bayesian cubature algorithm* that we developed so far

1. Exponential of Cosine:
   This is a very simple function, serves as an example to check the working of the cubature,

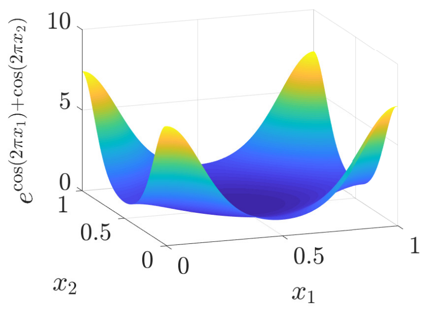

Fig. 3: Exp(Cos) in 2 dimensions

where the function is defined as

$$f(\boldsymbol{x}) = e^{\sum_{i=1}^{d} \cos(2\pi x_d)},$$

$$\int_{[0,1]^d} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \text{BesselI(0,1)}^d$$

where 'BesselI' is the *modified Bessel* function. Exp(Cos) function is periodic in $[0,1]$, so we do not need to use any *transform* to make it periodic.

2. Keister function
The following multidimensional integral example is based on the paper [Kei96], inspired by a physics application.

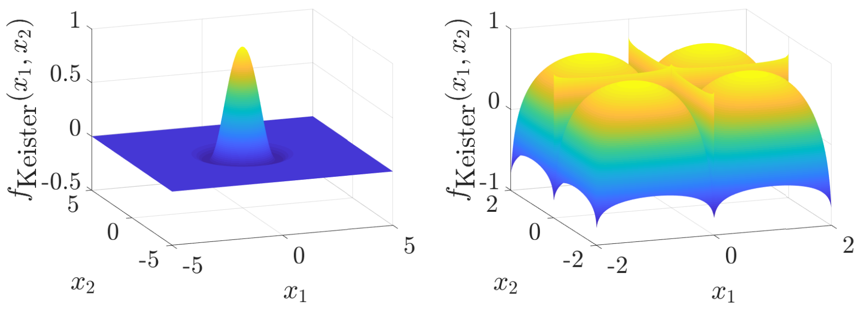

Fig. 4: Keister function in original form and its transform to $[0,1]^2$

$$f(\boldsymbol{x}) = \cos(\|\boldsymbol{x}\|) \exp(-\|\boldsymbol{x}\|^2) \mathrm{d}\boldsymbol{x},$$

$$\int_{\mathbb{R}^d} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} = \frac{2\pi^{\frac{d}{2}}}{\text{gamma}(\frac{d}{2})} \text{cosinteg}(d), \quad d = 1, 2, 3, \ldots$$

where

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

where gamma(.) := gamma function

3. Multivariate Normal:
   We use the Genz's method to compute Multivariate normal probability as explained below. This method reduces the original dimension of the problem by 1.

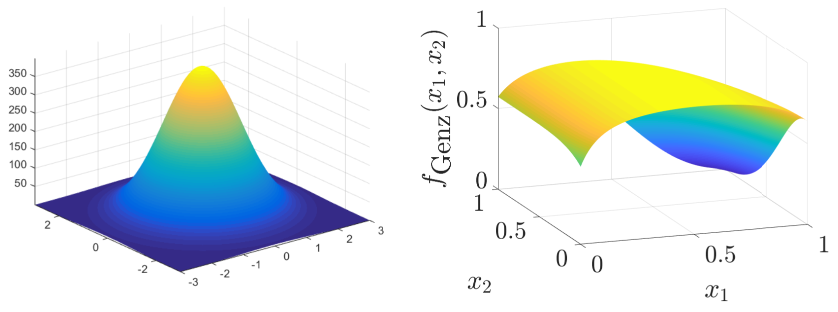

Fig. 5: Mutivariate Normal and Genz function

$$\mu = \int_{[\boldsymbol{a},\boldsymbol{b}]\in\mathbb{R}^d} \frac{\exp\left(-\frac{1}{2}\boldsymbol{t}^T\Sigma^{-1}\boldsymbol{t}\right)}{\sqrt{(2\pi)^d\det(\Sigma)}}\,\mathrm{d}\boldsymbol{t} \overset{\text{[Gen93]}}{=} \int_{[0,1]^{d-1}} f_{\text{Genz}}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

where $\Sigma = \mathsf{L}\mathsf{L}^T$ is the Cholesky decomposition of the covariance matrix, $\mathsf{L} = (l_{jk})_{j,k=1}^d$ is a lower triangular matrix, and

$$\begin{aligned}
\boldsymbol{\alpha}_1 &= \Phi(a_1), \quad \boldsymbol{\beta}_1 = \Phi(b_1) \\
\boldsymbol{\alpha}_j(x_1,...,x_{j-1}) &= \Phi\left(\frac{1}{l_{jj}}\left(a_j - \sum_{k=1}^{j-1} l_{jk}\Phi^{-1}(\boldsymbol{\alpha}_k + x_k(\boldsymbol{\beta}_k - \boldsymbol{\alpha}_k))\right)\right), j = 2,...,d, \\
\boldsymbol{\beta}_j(x_1,...,x_{j-1}) &= \Phi\left(\frac{1}{l_{jj}}\left(b_j - \sum_{k=1}^{j-1} l_{jk}\Phi^{-1}(\boldsymbol{\alpha}_k + x_k(\boldsymbol{\beta}_k - \boldsymbol{\alpha}_k))\right)\right), j = 2,...,d,
\end{aligned}$$

$$f_{\text{Genz}}(\boldsymbol{x}) = \prod_{j=1}^{d}[\boldsymbol{\beta}_j(\boldsymbol{x}) - \boldsymbol{\alpha}_j(\boldsymbol{x})]$$

As we see from the figure, Genz function is not periodic, So we need to make it periodic to get the best accuracy with Bayesian cubature.
We use the following parameter values for the numerical examples

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

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 Brownian motion, B, at time $t_1,...,t_d$, then $Z = (B(t_1),...,B(t_d)) \sim \mathcal{N}(\mathbf{0},\Sigma)$, where $\Sigma = (\min(t_j,t_k))_{j,k=1}^{d}$, and the fair price of the option is

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

where the function payoff(.) describes the discounted payoff of the option,

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

and L is any square matrix satisfying $\Sigma = \mathsf{L}\mathsf{L}^T$.
For the Asian arithmetic mean call option

$$\text{payoff}(\boldsymbol{z}) = \max\left(\frac{1}{d}\sum_{j=1}^{d} S_j - K, 0\right)e^{-rt}, \quad S_j = S_0\exp((r-\sigma^2/2)t_j + \sigma z_j)$$

where d - number of dimensions and T, S, S_0, K, r, σ are the parameters to be specified

5 Conclusion

We developed a generalized technique of *Fast transform kernel.* Using this technique, further developed fast automatic Bayesian cubature algorithm that takes very low computational cost in the order of $\mathcal{O}(n \log n)$ comparing to direct Bayesian cubature of $\mathcal{O}(n^3)$, so it can be used in practical applications. By adjusting the Bernoulli order r of the kernel and using the appropriately smoother variable transformation, we could get higher order of error convergence when the integrand is assumed to have zero mean. In general case without any assumption on the integrand mean , the optimal $\hat{\mu}$ is just the sample mean and so the order of convergence is defined by how smoother the underlying function being integrated and the variable transformation being used. Though the optimal $\hat{\mu}$ is just the sample mean, the error bound err_n still depends on the kernel order. So when we use higher order r, the error bound err_n closely matches the actual error. We could use the algorithm to integrate upto 2^{23} data points on a 16GB of RAM memory and i7-3630QM computer within 5 minutes. Numerical results show that the theoretical error err_n closely matches the actual error but it could still be improved with more tighter error bound especially when the r is smaller.

6 Future work

As an extension to the ideas and techniques established in this work, the following are considered potential future work

1. Sobol points and Fast Walsh Transform (FWT)
   We have shown one example of a special form of kernel with defined requirements to build a *Fast transform kernel.* Using the established generalized requirements for a fast-transform-kernel, we could use the same approach with other kernels with suitable point-sets to achieve similar or better performance and accuracy. One such point-sets that to consider in future is, *Sobol points* and with appropriate choice of kernel, which should lead to *Fast Walsh Transform.*
2. Control variates
   We would like to approximate a function of the form $(f - \beta_1 g_1 -, ..., -\beta_p g_p)$ than
   $$f = \mathcal{N}\left(\beta_0 + \beta_1 g_1 +, ..., +\beta_p g_p, s^2 \mathsf{C}\right)$$
3. Function approximation
   Let us consider approximating a function of the form
   $$\int_{[0,1]^d} \underbrace{f(\boldsymbol{\phi}(\boldsymbol{t})).\left|\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{t}}\right|}_{g(\boldsymbol{t})} \mathrm{d}\boldsymbol{t}$$
   where $\left|\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{t}}\right|$ is Jacobian and then

$$g(\boldsymbol{\psi}(\boldsymbol{x})) = f(\underbrace{\boldsymbol{\phi}(\boldsymbol{\psi}(\boldsymbol{x}))}_{\boldsymbol{x}}).\left|\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{t}}\right|(\boldsymbol{\psi}(\boldsymbol{x}))$$

$$f(\boldsymbol{x}) = g(\boldsymbol{\psi}(\boldsymbol{x})).\frac{1}{\left|\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{t}}\right|(\boldsymbol{\psi}(\boldsymbol{x}))}$$

Finally, the function approximation is

$$\begin{aligned}\tilde{f}(\boldsymbol{x}) &= \tilde{g}(\boldsymbol{\psi}(\boldsymbol{x})) \\ &= \sum w_i C(.,.)\end{aligned}$$

4. Deterministic interpretation of Bayesian cubature

References

[Dia88] P. Diaconis. Bayesian numerical analysis. *Statistical decision theory and related topics IV, Papers from the 4th Purdue symp., West Lafayette, Indiana 1986,*, page 163–175, 1988.

[DLM12] DLMF. Nist digital library of mathematical functions. *http://dlmf.nist.gov/*, Part 2:Release 1.0.5 of 2012–10–01, 2012.

[Fas07] G. E. Fasshauer. *Meshfree Approximation Methods with Matlab.* World Scientific Publishing Co, 2007.

[Gen93] A. Genz. Comparison of methods for the computation of multivariate normal probabilities. *Computing Science and Statistics*, 25:400 – 405, 1993.

[Hic17] Fred J. Hickernell. Error analysis for quasi-monte carlo methods. *arXiv:1702.01487 [math.NA]*, 2017.

[Kei96] B. D. Keister. Multidimensional quadrature algorithms. *Computers in Physics*, 10:119–122, 1996.

[O'H91] A. O'Hagan. Bayes-hermite quadrature. *J. Statist. Plann. Inference*, 29:245–260, 1991.

[RG03] C. E. Rasmussen and Z. Ghahramani. Bayesian monte carlo. *Advances in Neural Information Processing Systems*, pages 489–496, 2003.

[Rit00] K. Ritter. Average-case analysis of numerical problems. *Lecture Notes in Mathematics, Springer-Verlag, Berlin*, vol. 1733:163–175, 2000.

[SD06] Alexander Keller Sabrina Dammertz. Image synthesis by rank-1 lattices. *Monte Carlo and Quasi-Monte Carlo Methods*, Part 2:217–236, 2006.