ESAIM: PROCEEDINGS AND SURVEYS, June 2025, Vol. 81, p. 193-218
Didier Auroux, Martin Campos Pinto, Bruno Després, Victorita Dolean, Stéphane Lanteri, Victor Michel-Dansac

MULTIDIMENSIONAL INTEGRATION USING MACHINE LEARNING AND MONTE CARLO METHODS FOR ACOUSTIC PREDICTIONS

Théo Coiffard[1], Victoria Eshomomoh Laoye[2], Yolhan Mannes[3], Shyam Mohan Subbiah Pillai[4] and Nicolas Trafny[5]

Abstract. To predict underwater noise radiated by a ship, various numerical methods are available. In underwater acoustics, the most effective prediction methods consist in solving an acoustic analogy using an integral formulation. In this study, we propose a machine learning surrogate-based method, combined with Monte Carlo integration, to efficiently estimate volume integrals that arise in acoustic analogies. We use three machine learning surrogate models: multi-layer perceptrons, Gaussian processes and gradient-boosted decision trees. For each model, a theoretical background is presented. We conduct numerical experiments to compare the state-of-the-art classical Monte Carlo quadrature method with our new machine learning based method. We first apply our method to simple canonical functions, which are analytically integrable, to evaluate the accuracy of our method. We then use a multi-layer perceptron-based surrogate model to approximate a fabricated function that mimics the characteristics of noise sources found in acoustic prediction models, such as those related to turbulent flows near geometrical singularities. Numerical experiments demonstrate that the proposed machine learning-based approach achieves performance levels comparable to state-of-the-art Monte Carlo quadrature methods, demonstrating the potential of ML techniques in this domain.

1. Introduction

Ambient noise in the ocean caused by anthropogenic and natural sources has been an increasing subject of concern. Over the last decades, the underwater noise pressure spectrum has increased by at least 12dB compared to pre-industrial levels [1]. Underwater noise radiated from large vessels, which is predominantly in the low frequency range (5 - 300Hz), is becoming a growing concern since it has been shown to coincide with frequencies audible by marine wildlife [2]. Hence, it can have a significant impact, especially on species that have evolved to exploit hearing as their primary sense. In order to design vessels meeting regulatory guidelines [3] that define a maximum radiated pressure level, it is necessary to understand the various noise generation mechanisms involved and to develop prediction methods associated to each of these mechanisms.

There are two distinct contributions to the total radiated acoustic field noise caused by a ship: the vibratory contribution, produced by the excitation of the hull by turbulent eddies and machinery noise, and the purely hydrodynamic contribution near the propeller produced by cavitation, turbulence, and their interaction with

[1] La Rochelle Université
[2] University of Ibadan
[3] Université De Toulon
[4] Alexander von Humboldt Chair of Mathematics for Uncertainty Quantification, RWTH Aachen University, Germany
[5] NAVAL Group

Article published online by EDP Sciences and available at https://www.esaim-proc.org or https://doi.org/10.1051/proc/202581193

propeller blades. Many of the existing flow-induced noise prediction methods, based on complete numerical simulation of the turbulent flow, are too expensive for industrial underwater applications [4]. Recently, an approach based on the Lighthill acoustic analogy and the use of a tailored Green's function [5] has been proposed. This approach requires a numerical simulation of the mean flow and enables us to consider realistic geometries.

However, it requires the computation of a 6-dimensional integral, which is computationally expensive. A suitable method for solving high-dimensional integrals is the Monte Carlo method [6]. Monte Carlo methods are a broad class of computational algorithms that rely on independent and identically distributed (iid) random samples to estimate a quantity of interest. The error associated with this probabilistic method does not depend on the number of dimensions, whereas the error made by deterministic quadrature methods increases exponentially with the number of dimensions. It can be shown [7] that the integration error decreases in the worst case (i.e., without using any adaptive method) as $M^{0.5}$, where M is the number of sample evaluations of the integrand. The disadvantage of the Monte Carlo method is that a high integration precision cannot be achieved. In underwater acoustics, however, a high precision is not necessary, since a targeted $\pm$1dB precision on the radiated sound pressure level is already reached for a 25% relative integration error. There are different methods used to improve the performance of Monte Carlo integration [8], such as quasi Monte Carlo, stratified sampling, importance sampling, sequential Monte Carlo and mean-field particle methods. The most popular ones are importance sampling and stratified sampling. On one hand, importance sampling relies on the idea of modifying the sampling probability density, in order to concentrate the integrand evaluation points in the regions where the magnitude of the integrand is the largest. On the other hand, stratified sampling technique consists of subdivision of the integration domain into sub-domains over which the integrals are evaluated.

Despite these various approaches that greatly improve Monte Carlo quadrature performance, the computational cost required to evaluate the underwater radiated noise by a full-scale ship remains prohibitive. Various approaches based on hybrid Monte Carlo algorithms and a machine learning surrogate model have been proposed [9–11]. The objective of this study is to investigate if these new approaches can be used to improve the performance of acoustic prediction tools by reducing the amount of time required to evaluate a 6-dimensional integrand for a given targeted precision. This paper is structured as follows; in section 2, we briefly introduce the acoustic prediction models to present the key features of the integrals we need to evaluate. In section 3, we introduce some theoretical background to Monte Carlo integration and more specifically to the VEGAS algorithm. In Sections 4 to 6, we apply some machine learning-based approaches on a variety of canonical functions that can be integrated analytically. Finally, in Section 7, we apply our surrogate-based integration technique on a fabricated integrand, exhibiting characteristics analogous to those encountered in acoustic prediction models.

2. Context of the problem

The objective of this section is to introduce the equations that can be used to estimate the radiated pressure created by the interaction of a turbulent flow and a solid boundary surface of arbitrary shape. We consider a bounded obstacle with boundary S placed inside a turbulence volume V. The sound radiated by the flow around the obstacle can be determined using Lighthill's wave equation [12], which is expressed in the time-harmonic regime as (assuming an $e^{-i\omega t}$ time dependence):

$$(\nabla^2 + k_0^2)p(\mathbf{x},\omega) = -\frac{\partial^2 T_{ij}}{\partial x_i \partial x_j}, \tag{1}$$

where $k_0 = \omega/c_0$ is the acoustic wave-number, with ω the angular frequency and c_0 the speed of sound, T_{ij} is the incompressible Lighthill tensor. We assume [13] that $T_{ij} = \rho_0 u_i u_j$, where ρ_0 is the fluid density and u_i is the flow velocity. A solution to Lighthill's wave equation can be obtained by introducing a Green's function. Here, among an infinite number of Green's functions, we use a tailored Green function that satisfies the rigid

boundary condition on S:

$$\begin{cases} (\nabla^2 + k_0^2) G_T(\mathbf{x}, \mathbf{y}, \omega) = -\delta(\mathbf{x} - \mathbf{y}), \\ \nabla G_T(\mathbf{x}, \mathbf{y}, \omega) \cdot \mathbf{n} = 0, \quad \forall \mathbf{y} \in S. \end{cases} \tag{2}$$

We can use the Green's theorem, that yields [13]:

$$p(\mathbf{x}, \omega) = -\int_V \frac{\partial^2 T_{ij}}{\partial x_i \partial x_j} G_T(\mathbf{x}, \mathbf{y}, \omega) dV - \oint_S G_T(\mathbf{x}, \mathbf{y}, \omega) \nabla p(\mathbf{y}, \omega) \cdot \mathbf{n} dS \tag{3}$$

where $\mathbf{n}$ is the normal to the surface taken in the outward direction from the solid. Using the momentum equation, the pressure gradient over the surface can be related to the surface velocity fluctuation $\mathbf{v}$, so that:

$$p(\mathbf{x}, \omega) = -\int_V \frac{\partial^2 T_{ij}}{\partial x_i \partial x_j} G_T(\mathbf{x}, \mathbf{y}, \omega) dV - i\omega\rho_0 \oint_S G_T(\mathbf{x}, \mathbf{y}, \omega) v_j(\mathbf{y}, \omega) n_j dS \tag{4}$$

The first integral in (4) is the hydro-acoustic radiation due to the direct acoustic radiation of the turbulent flow and its amplification by the surface. The second integral in (4) is the vibro-acoustic radiation, due to the vibration of the surface S. We cannot use (4) directly because it requires knowledge of the Lighthill stress tensor, whose computation is too expensive at low Mach number. Therefore, it is convenient in the frequency domain to use a statistical description of the turbulence. The analysis splits into two parts. We define the radiated pressure spectrum $S_{pp}(\mathbf{x}, \omega) = \overline{p(\mathbf{x}, \omega) p^*(\mathbf{x}, \omega)}$, where the over-line denotes the ensemble average operator. This spectrum has two components: the hydro-acoustic contribution $S_{pp}^{(H)}(\mathbf{x}, \omega)$ and the vibroacoustic contribution $S_{pp}^{(V)}(\mathbf{x}, \omega)$. The total spectrum is the sum of these two components. First, $S_{pp}^{(H)}$ can be expressed, for a given observation point $\mathbf{x} \in V$:

$$S_{pp}^{(H)}(\mathbf{x}, \omega) = 4\rho_0^2 \int_V \int_V U_i U_j \phi_{kl}(\mathbf{y}, \mathbf{z}, \omega) \frac{\partial^2 G_T(\mathbf{x}, \mathbf{y}, \omega)}{\partial y_i \partial y_k} \frac{\partial^2 G_T^*(\mathbf{x}, \mathbf{z}, \omega)}{\partial z_j \partial z_l} dV(\mathbf{y}) dV(\mathbf{z}) \tag{5}$$

where U_i is the mean-flow velocity and ϕ_{kl} is the turbulence velocity fluctuation spectrum [14]. One important feature of equation (5) is that the turbulence volume V is defined as the exterior volume surrounding the rigid boundary surface. To compute the 6-dimensional integral in (5), it is convenient to consider V as a cube and to consider an integrand that is zero-valued inside the boundary surface. This may seem a trivial point but, as it is discussed in section 4, it appears to be of great interest when the integrand is approximated by a surrogate model. To compute the vibro-acoustic radiated pressure, we need to proceed in two steps. First, we need to estimate the surface velocity fluctuations cross-spectrum for $\mathbf{x} \in S$ and $\mathbf{x'} \in S$:

$$\phi_{vv}(\mathbf{x}, \mathbf{x'}, \omega) = \int_S \int_S H_v(\mathbf{x}, \mathbf{y}, \omega) \phi_{pp}(\mathbf{y}, \mathbf{z}, \omega) H_v^*(\mathbf{x'}, \mathbf{z}, \omega) dS(\mathbf{y}) dS(\mathbf{z}), \tag{6}$$

with ϕ_{pp} being the wall pressure fluctuation cross-spectrum that represents the excitation of the surface S by the turbulent flow and where H_v is a transfer function that represents the displacement response of the structure at $\mathbf{x}$ caused by a point force at $\mathbf{y}$ varying sinusoidally over time. Finally, the vibro-acoustic radiated pressure spectrum can be expressed:

$$S_{pp}^{(V)}(\mathbf{x}, \omega) = \omega^2 \rho_0^2 \int_S \int_S \phi_{vv}(\mathbf{y}, \mathbf{z}, \omega) G_T(\mathbf{x}, \mathbf{y}, \omega) G_T^*(\mathbf{x}, \mathbf{z}, \omega) dS(\mathbf{y}) dS(\mathbf{z}). \tag{7}$$

The calculation of equations (5) to (7) requires the knowledge of the tailored Green's function and the mean-flow characteristics. The tailored Green's function is known analytically only for canonical geometries and must be computed numerically for geometries of arbitrary shape [5]. It is important to note that, in both cases, it is a computationally expensive step. The mean-flow characteristics (mean-flow velocity, turbulent kinetic energy,

turbulence dissipation rate ...) must be calculated using a Computational Fluid Dynamics (CFD) simulation and stored in a three dimensional mesh. The interpolation of the required physical quantities inside the CFD mesh can also be computationally expensive for large data sets. In order to derive an efficient quadrature method for these equations, it is important to note some other key features of the integrands in equations (5) to (7). Firstly, they are complex functions with oscillatory real and imaginary parts. How much the function oscillate over the integration domain V depends on the frequency: the higher the frequency, the more challenging the integration is. Secondly, the Green function $G_T(\mathbf{x}, \mathbf{y}, \omega)$ being singular for $\mathbf{x} = \mathbf{y}$, the integrands of these functions are nearly singular when $\mathbf{y}$ or $\mathbf{z}$ get close to $\mathbf{x}$.

3. Monte Carlo Integration

In this Section, we aim to introduce the Monte Carlo method to estimate definite integrals of type (7). For a given function $f : \mathbb{R}^d \to \mathbb{R}$, we wish to estimate the following integral using the Monte Carlo method:

$$g = \int_{[0,1]^d} f(\mathbf{x}) \mathrm{d}\mathbf{x}, \quad \mathbf{x} \in \mathbb{R}^d. \tag{8}$$

We can write g in terms of an expected value as follows:

$$g = \int_{[0,1]^d} f(\mathbf{x}) \mathrm{d}\mathbf{x} = \int_{[0,1]^d} f(\mathbf{x}) \cdot 1 \mathrm{d}\mathbf{x} = \int_{[0,1]^d} f(\mathbf{x}) \cdot \rho_x(\mathbf{x}) \mathrm{d}\mathbf{x} = \mathbb{E}\left[f(X)\right] \cdot \tag{9}$$

Here ρ_x is the uniform probability density function in $[0,1]^d$ and X is a random variable uniformly distributed in $[0,1]^d$, i.e. $X \sim \mathcal{U}([0,1]^d)$. Monte Carlo integration is based on the following approximation.

$$g \approx \mathcal{A}(M) = \frac{1}{M} \sum_{m=1}^{M} f(X^{(m)}), \tag{10}$$

where $X^{(m)}$ is the m^{th} iid realization of the random variable $X \sim \mathcal{U}([0,1]^d)$ and M denotes the total number of samples of X used in the approximation (10). One can look at (10) as a quadrature formula with uniformly random points in $[0,1]^d$ and equal weights $\frac{1}{M}$. We would like to quantify the error in the approximation in (10). Denoting this error as $\epsilon(M)$, we have:

$$\epsilon(M) = |\mathcal{A}(M) - g| = \left| \frac{1}{M} \sum_{m=1}^{M} \Big(f(X^{(m)}) - g \Big) \right|. \tag{11}$$

Since $\mathcal{A}(M)$ is a random variable, the central limit theorem [15] gives a probabilistic bound on the error $\epsilon(M)$. This theorem states that, in the limit $M \to \infty$,

$$\sqrt{M}\epsilon(M) \to \eta \sim \mathcal{N}(0, \sigma^2), \tag{12}$$

where $\mathcal{N}(0, \sigma^2)$ is a normal distribution with mean 0 and variance $\sigma^2 = \mathrm{Var}\left[f(X)\right]$. Let TOL be the prescribed accuracy of the approximation (10). Since $\epsilon(M)$ is a random variable, we would like to achieve this accuracy in the following sense.

$$\mathbb{P}\left[\epsilon(M) \leq \mathrm{TOL}\right] \geq 1 - \alpha, \tag{13}$$

for some $0 < \alpha \ll 1$, that determines the confidence level of our estimate. From (12), we know that, in the limit $M \to \infty$, $\sqrt{M}\frac{\epsilon(M)}{\sigma} \to \nu \sim \mathcal{N}(0,1)$. Using this, we can rewrite (13) as,

$$\mathbb{P}\left[\nu \leq \frac{\sqrt{M}\mathrm{TOL}}{\sigma}\right] \geq 1-\alpha. \tag{14}$$

We can now use the cumulative distribution function of the standard normal distribution to deduce that

$$\mathrm{TOL} = C_\alpha \frac{\sigma}{\sqrt{M}}, \tag{15}$$

where C_α is the $\left(1-\frac{\alpha}{2}\right)$-quantile of the standard normal distribution. For example, $C_\alpha \approx 1.96$ corresponds to $\alpha = 0.05$. This implies that, $\mathcal{A}(M)$ approximates g with an error $\epsilon(M)$ which is less than $C_\alpha \frac{\sigma}{\sqrt{M}}$ with confidence $1-\alpha$. That is $\epsilon(M) = \mathcal{O}\left(M^{-0.5}\right)$. Consequently, the error convergence rate is independent of the dimension of the problem, making Monte Carlo suitable for high-dimensional integrals. Conversely, from (15), one would need

$$M = C_\alpha^2 \frac{\sigma^2}{\mathrm{TOL}^2} = \mathcal{O}\left(\mathrm{TOL}^{-2}\right) \tag{16}$$

samples to achieve an error $\epsilon(M) \leq \mathrm{TOL}$. However, if $\sigma^2 = \mathrm{Var}\left[f(X)\right]$ is large, then one needs large M to achieve an accuracy of TOL. To overcome this, we investigate two variance reduction methods in this work.

3.1. **Importance Sampling**

Importance sampling is a technique that changes the sampling distribution to reduce variance, without changing the expected value. This can be shown as follows.

$$g = \mathbb{E}\left[f(X)\right] = \int_{[0,1]^d} f(\mathbf{x})\rho_x(\mathbf{x})\mathrm{d}\mathbf{x} = \int_{[0,1]^d} \underbrace{f(\mathbf{x})\frac{\rho_x(\mathbf{x})}{\hat{\rho}_z(\mathbf{x})}}_{=\hat{f}(\mathbf{x})} \hat{\rho}_z(\mathbf{x})\mathrm{d}\mathbf{x} = \mathbb{E}\left[\hat{f}(Z)\right]. \tag{17}$$

However, this does not mean that $\mathrm{Var}\left[f(X)\right]$ and $\mathrm{Var}\left[\hat{f}(Z)\right]$ are the same. By a clever choice of sampling distribution $\hat{\rho}_z(\mathbf{x})$, one can get an estimator with significantly reduced variance. For example, let $\hat{\rho}_z(\mathbf{x}) = Cf(\mathbf{x})\rho_x(\mathbf{x})$ where $C = \left(\int_{[0,1]^d} f(\mathbf{x})\rho_x(\mathbf{x})\mathrm{d}\mathbf{x}\right)^{-1}$. Then, we see that

$$\mathbb{E}\left[\hat{f}(Z)^2\right] = \int_{[0,1]^d} \hat{f}(\mathbf{x})^2\hat{\rho}_z(\mathbf{x})\mathrm{d}\mathbf{x} = \left(\int_{[0,1]^d} f(\mathbf{x})\rho_x(\mathbf{x})\right)^2 = \mathbb{E}\left[f(X)\right]^2 = \mathbb{E}\left[\hat{f}(Z)\right]^2. \tag{18}$$

This implies that we have a zero-variance estimator with this choice of $\hat{\rho}_z$. However, this choice is not practical because we would need knowledge of $\int_{[0,1]^d} f(\mathbf{x})\rho_x(\mathbf{x})\mathrm{d}\mathbf{x}$, which is basically our quantity of interest. However, this indicates that if we choose a sampling distribution that closely follows the product $f \cdot \rho_x$, we could achieve significant variance reduction. This implies that a good importance sampling estimator would sample more from the regions where the function f has higher values, for the case ρ_x is a uniform distribution. The Monte Carlo estimator with importance sampling is written as

$$g = \mathbb{E}\left[f(X)\right] = \mathbb{E}\left[\hat{f}(Z)\right] \approx \mathcal{A}^{\mathrm{IS}}(M) = \frac{1}{M}\sum_{m=1}^{M} f(Z^{(m)})\frac{\rho_x(Z^{(m)})}{\hat{\rho}_z(Z^{(m)})}. \tag{19}$$

Equation (19) can be seen as a quadrature formula with random points distributed according to $\hat{\rho}_z$ and unequal weights $\frac{1}{M}\frac{\rho_x}{\hat{\rho}_z}$. By a clever choice of $\hat{\rho}_z$, we could drastically reduce the number of samples required to estimate g to an accuracy of TOL.

3.2. **Stratified Sampling**

Suppose we divide the domain $[0,1]^d$ into K volumes, which we call *strata*. Let us denote the k^{th} stratum as $\mathcal{S}_k$. The strata are constructed in a manner such that

- $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset, \forall i, j = 1, \ldots, K$, where $\emptyset$ denotes the null set,
- $\cup_{k=1}^{K} \mathcal{S}_k = [0,1]^d$.

Then, stratified sampling is based on the following identity.

$$g = \int_{[0,1]^d} f(\mathbf{x}) \mathrm{d}\mathbf{x} = \sum_{k=1}^{K} \int_{\mathcal{S}_k} f(\mathbf{x}) \mathrm{d}\mathbf{x}. \tag{20}$$

Using the expected value representation of the integral g and the law of total expectation, we can re-write (20) as

$$\begin{aligned} g = \int_{[0,1]^d} f(\mathbf{x}) \mathrm{d}\mathbf{x} &= \mathbb{E}\left[f(X)\right], \quad X \sim \mathcal{U}([0,1]^d) \\ &= \sum_{k=1}^{K} \mathbb{E}\left[f(X) \mid X \in \mathcal{S}_k\right] \mathbb{P}\left[X \in \mathcal{S}_k\right]. \end{aligned} \tag{21}$$

We then estimate each of the expectations in (21) using a Monte Carlo estimator as follows.

$$\mathbb{E}\left[f(X) \mid X \in \mathcal{S}_k\right] \approx \frac{1}{M_k} \sum_{m=1}^{M_k} f(X^{(m)}), \quad X \sim \mathcal{U}(\mathcal{S}_k). \tag{22}$$

Since X is uniformly distributed in $[0,1]^d$, we know that $\mathbb{P}\left[X \in \mathcal{S}_k\right] = V_k$, where V_k is the volume of the k^{th} stratum. Therefore, the Monte Carlo estimator with stratified sampling can be written as

$$\mathcal{A}^{\text{SS}}(M) = \sum_{k=1}^{K} \frac{1}{M_k} \sum_{m=1}^{M_k} f(X^{(k,m)}) V_k, \tag{23}$$

where $X^{(k,m)}$ denotes the m^{th} iid realisation of the random variable $X \sim \mathcal{U}(\mathcal{S}_k)$ and $M = \sum_{k=1}^{K} M_k$. One can see that $\mathrm{Var}\left[\mathcal{A}^{\text{SS}}(M)\right] = \sum_{k=1}^{K} \frac{1}{M_k} \mathrm{Var}\left[f(X) \mid X \in \mathcal{S}_k\right] V_k^2$. For convenience, let $M_k = \frac{M}{K}$ for all $k = 1, \ldots, K$. Then, the problem of stratified sampling is to find the optimal division of strata $\{\mathcal{S}_k\}_{k=1}^{K}$ that minimizes $\mathrm{Var}\left[\mathcal{A}^{\text{SS}}(M)\right]$. The minimum is achieved when the product $\mathrm{Var}\left[f(X) \mid X \in \mathcal{S}_k\right] \cdot V_k^2$ is constant for all $k = 1, \ldots, K$ [16]. This implies that a stratified sampling algorithm would concentrate samples in the strata where $\mathrm{Var}\left[f(X) \mid X \in \mathcal{S}_k\right]$ is large, that is, where the function f has large gradients.

3.3. VEGAS

The VEGAS Algorithm was first introduced in [8] for efficient high-dimensional ($d > 4$) integration. It uses Monte Carlo integration, enhanced by adaptive importance sampling and stratified sampling (as explained in Sections 3.1 and 3.2), to estimate multidimensional integrals efficiently. It has the following characteristics.

- Error estimate for the integral must be readily computable,
- Non-smooth f should pose no significant problem,
- Error convergence rate should be independent of d,
- Algorithm should automatically concentrate sample evaluations in regions of importance (importance sampling) and regions with high variation (stratified sampling).

The algorithm has a computable error estimate (15), where $\sigma = \sqrt{\mathrm{Var}\left[f(X)\right]}$ is approximated by the sample variance using the same samples used to compute the expected value. Since the algorithm is a Monte Carlo integrator, the error convergence rate is independent of both the smoothness and dimensionality of the integrand f (as seen in (15)). To evaluate high-dimensional integrals efficiently, VEGAS uses importance sampling and stratified sampling (Sections 3.1 and 3.2) to concentrate sample points in regions of importance and high variation. We use the VEGAS algorithm developed by the authors in [8] for automated and efficient Monte Carlo integration in the rest of this work.

4. Machine learning surrogates

In our exploration of approximation methodologies, we intend to rigorously test three main approximators: Gaussian processes (GP), gradient-boosted decision trees (GBDT), and multi-layer perceptrons (MLP). The motivation behind this testing is to gauge the accuracy of these approximators when faced with diverse and challenging mathematical functions.

Following [9], the functions selected for this analysis are deliberately chosen to reflect a broad range of difficulties typically encountered in approximation problems. We list these test functions in Table 1. These functions vary in their complexity, ranging from oscillatory patterns to sharp peaks and even discontinuities. Such a selection is pivotal in ensuring that our approximators are not just tailored for a specific type of function but have a more general application.

A crucial element in the chosen functions is the parameter $\boldsymbol{c}$. The value of $\boldsymbol{c}$ plays a pivotal role in determining the difficulty of the function's numerical integrability. Larger the magnitudes of $\boldsymbol{c}$, more difficult it is to numerically integrate the function. By modulating $\boldsymbol{c}$, we can simulate different degrees of challenges for our approximators, thereby getting a more comprehensive view of their performance capabilities.

In the subsequent sections, we will delve deeper into each surrogate model and the results yielded by these models when integrating the functions listed in Table 1.

4.1. Multi-layer perceptrons

Multi-layer perceptrons (MLPs), a type of feedforward artificial neural network, have been at the forefront of machine learning advancements for several decades. At their core, MLPs are composed of layers of neurons (or nodes) that transmit and transform information from the input to the output. These nodes are interconnected through weights, which are adjusted during the training process to minimize the difference between the predicted and actual outputs.

MLPs can be shown to be able to approximate any function on a compact domain in some topology. This is known as the "Universal Approximation Theorem" [17]. This positions MLPs as a versatile tool in numerous applications, from predicting stock prices to identifying objects in images [18, 19].

Characteristic feature	Function name	Form
Oscillatory	$f_1(\mathbf{x})$	$\cos(2\pi w_1 + \boldsymbol{c}\cdot\boldsymbol{x})$
Product peak	$f_2(\mathbf{x})$	$\prod_{i=1}^{n} \frac{1}{{c_i}^{-2}+(x_i-w_i)^2}$
Corner peak	$f_3(\mathbf{x})$	$\frac{1}{(1+\boldsymbol{c}\cdot\boldsymbol{x})^{n+1}}$
Gaussian	$f_4(\mathbf{x})$	$\exp\left(-\sum_{i=1}^{n} c_i^2(x_i-w_i)^2\right)$
C^0-function	$f_5(\mathbf{x})$	$\exp(-\boldsymbol{c}\cdot\lvert\boldsymbol{x}-\boldsymbol{w}_1\rvert)$
Discontinuous	$f_6(\mathbf{x})$	$\begin{cases}\exp(\boldsymbol{c}\cdot\boldsymbol{x}) & \text{if } x_1 \le w_1 \text{ or } x_2 \le w_2\\ 0 & \text{otherwise}\end{cases}$

TABLE 1. List of test functions

Mathematics of MLPs

From a mathematical perspective, the operation within a neuron in an MLP can be thought of in two primary steps: a weighted sum of inputs and an activation function applied to this sum. Given an input vector $\mathbf{x}$ and weight vector $\mathbf{w}$ for a particular neuron, the weighted sum z is given by:

$$z = \mathbf{w}\cdot\mathbf{x} + b \tag{24}$$

Here, b is a bias term, allowing the neuron to shift the output up or down as needed. Once this weighted sum is calculated, it is passed through an activation function f:

$$y = f(z). \tag{25}$$

Common choices for f include the sigmoid function, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). These functions introduce non-linearities, enabling MLPs to approximate non-linear functions. Considering a single-layer perceptron with a hidden layer, the operations can be represented in a matrix format. If $\mathbf{W}_1$ is the weight matrix from the input to the hidden layer and $\mathbf{W}_2$ is the weight matrix from the hidden layer to the output, then the output $\mathbf{y}$ for input $\mathbf{x}$ is:

$$\begin{cases}\mathbf{h} = f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1),\\ \mathbf{y} = f(\mathbf{W}_2\mathbf{h} + \mathbf{b}_2).\end{cases} \tag{26}$$

Here, $\mathbf{h}$ is the output from the hidden layer, $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias vectors for the hidden and output layers, respectively. Training an MLP involves adjusting the weight matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ (as well as biases) to minimize the difference between predicted output $\mathbf{y}$ and the actual target values. This is often done using the backpropagation algorithm [20], which leverages the chain rule of calculus to distribute the error from the output backwards through the network.

The MLP model used in this work

The neural network model in consideration in this article is a multi-layer perceptron (MLP) designed with four hidden layers, each having 128 neurons. Formally, given an input vector $\mathbf{x} \in \mathbb{R}^6$, the operations within the network can be represented as follows:

- **First Hidden Layer**: $\mathbf{h}_1 = \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$ where $\mathbf{W}_1 \in \mathbb{R}^{128\times 6}$ and $\mathbf{b}_1 \in \mathbb{R}^{128}$.
- **Second Hidden Layer**: $\mathbf{h}_2 = \tanh(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$ where $\mathbf{W}_2 \in \mathbb{R}^{128\times 128}$ and $\mathbf{b}_2 \in \mathbb{R}^{128}$.
- **Third Hidden Layer**: $\mathbf{h}_3 = \tanh(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$ where $\mathbf{W}_3 \in \mathbb{R}^{128\times 128}$ and $\mathbf{b}_3 \in \mathbb{R}^{128}$.
- **Fourth Hidden Layer**: $\mathbf{h}_4 = \tanh(\mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4)$ where $\mathbf{W}_4 \in \mathbb{R}^{128\times 128}$ and $\mathbf{b}_4 \in \mathbb{R}^{128}$.

- **Output Layer**: $y = \mathbf{W}_5\mathbf{h}_4 + b_5$ where $\mathbf{W}_5 \in \mathbb{R}^{1\times 128}$ and $b_5 \in \mathbb{R}$.

To optimize the model's weights and biases, the Mean Squared Error Loss (MSEL) is utilized, defined as:

$$\mathrm{MSEL}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{27}$$

where $\mathbf{y}$ is the vector of true values and $\hat{\mathbf{y}}$ is the vector of predicted values by the model. The parameters of the network are updated using the Adam optimization algorithm [21] with a learning rate of 0.01.

d	$\|c\|$		f_1	f_3	f_4	f_5	f_6
6	8	Training (s)	2.07	2.14	2.18	2.046	2.24
		Integration (s)	4.87	1.67	5.27	4.93	5.67
		Rel. Error (%)	0.6	7.6	0.08	0.21	1
6	1	Training (s)	1.95	2.23	2.10	1.88	2.20
		Integration (s)	4.60	5.19	5.01	4.59	5.51
		Rel. Error (%)	0.01	0.05	0.003	0.016	0.2
4	8	Training (s)	2.32	2.37	2.32	2.36	2.05
		Integration (s)	5.54	2.00	5.84	5.7	5.34
		Rel. Error (%)	0.55	0.7	0.09	0.1	0.6

TABLE 2. Performance of integration on MLP models of various functions for two dimensions d and two integrand difficulties $\|c\|$. (Average over 10 runs)

4.2. **Gaussian processes**

Surrogate modelling using Gaussian processes for numerical simulators was first used in the 1980s. Gaussian process regression has recently gained interest among the machine learning community, as it is a non-parametric model and provides a posterior distribution to the model, enabling uncertainty quantification on predictions. In this section, we aim to briefly demonstrate how a multi-dimensional function can be approximated using Gaussian process regression.

A Gaussian stochastic process [22] is a collection of random variables $\{\mathcal{G}(\mathbf{x}) \mid \mathbf{x} \in S_{\text{train}}\}$, the probability distribution of which is fully specified, for every finite subset of variables $\mathcal{G}(\mathbf{x}^{(1)}), \ldots, \mathcal{G}(\mathbf{x}^{(k)})$, by its mean function $\mu(\mathbf{x}) = \mathbb{E}[\mathcal{G}(\mathbf{x})]$ and its covariance function $\Sigma(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(\mathcal{G}(\mathbf{x}) - \mu(\mathbf{x}))(\mathcal{G}(\mathbf{x}') - \mu(\mathbf{x}'))]$. The set S_{train}, in our case, will be the input space with dimension N, the number of inputs. Any finite set of points will have a joint multivariate Gaussian distribution. The training data consists of N pairs of inputs and targets $\{(\mathbf{x}^{(i)}, t^{(i)}), i = 1, \ldots, N\}$. The test input vector is denoted by $\mathbf{x}$, without a superscript. The inputs $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ are d-dimensional and the targets $\{t^{(i)}\}_{i=1}^{N}$ are scalar. Since we assume $\mathcal{G}$ is a Gaussian process, the training outputs are just one sample from a multivariate Gaussian distribution of dimension N. Moreover, the joint distribution of the training outputs plus one test case is an $N+1$-dimensional Gaussian distribution. Hence, the predictive distribution of the test point $\mathbf{x}$, conditioned on the observed targets in the training set of size N, can be proven to be Gaussian. Using this, we can use the properties of conditional Gaussian distributions to derive the following mean and covariance of the predictive distribution.

$$\hat{\mu}(\mathbf{x}) = \mu(\mathbf{x}) + k^{\mathrm{T}}(\mathbf{x})C^{-1}(\boldsymbol{\mu}(\mathbf{x}) - \mathbf{t}) \tag{28}$$

$$\hat{\sigma}^2(\mathbf{x}) = \Sigma(\mathbf{x}, \mathbf{x}) - k^{\mathrm{T}}C^{-1}k(\mathbf{x})\cdot \tag{29}$$

where $k(\mathbf{x}) = \left(\Sigma(\mathbf{x}, \mathbf{x}^{(1)}), \ldots, \Sigma(\mathbf{x}, \mathbf{x}^{(N)})\right)^{\mathrm{T}}$, C is the covariance matrix for the training data with $C_{ij} = \Sigma(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, $\boldsymbol{\mu}(\mathbf{x}) = \{\mu(\mathbf{x}^{(1)}), \ldots, \mu(\mathbf{x}^{(N)})\}$ and $\mathbf{t} = \left(t^{(1)}, \ldots, t^{(N)}\right)^{\mathrm{T}}$. Note that the matrix inversion step in (28) and (29) could be computationally expensive especially for large ($\gg 100$) training data sets. In this work, we choose a mean function $\mu(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} t^{(n)}$ that is constant over all $\mathbf{x} \in \mathbb{R}^d$, and a constant covariance function Σ with radial basis function defined as follows:

$$\Sigma(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left\{-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right\}. \tag{30}$$

The covariance function is parametrized by two hyperparameters σ_f^2 and ℓ. Given a covariance function, the log likelihood of the training data is given by

$$l = -\frac{1}{2}\log\det C - \frac{1}{2}\mathbf{t}^{\mathrm{T}} C^{-1}\mathbf{t} - \frac{N}{2}\log 2\pi. \tag{31}$$

The optimal hyperparameters are adjusted to maximize (31). We initialize the hyperparameters to random values and then use an iterative method to search for optimal values of the hyperparameters. Since there are only two hyperparameters, a relatively small number of iterations is usually sufficient for convergence. However, this approach is susceptible to local minima, hence we try a number of random starting positions in hyperparameter space. The required surrogate function $\bar{f} : \mathbb{R}^d \to \mathbb{R}$ is the updated mean function $\hat{\mu} : \mathbb{R}^d \to \mathbb{R}$ (28). The uncertainty in the prediction is quantified using the variance function $\hat{\sigma}^2 : \mathbb{R}^d \to \mathbb{R}$ (29) at each test point. Figure 1 demonstrates a test example of Gaussian process regression.

Gaussian process regression is a non-parametric approach to constructing surrogate models as it does not assume that the surrogate model belongs to a certain class of functions. This method works well even on small datasets and provides uncertainty quantification on predictions. On the other hand, Gaussian process regression leads to dense matrix calculations, which quickly become computationally expensive for moderately large training datasets. This technique also becomes less efficient in high-dimensional spaces ($d > 10$). To measure the approximation quality of the Gaussian process surrogate $\bar{f}$ to the original function f, we compute the $\mathbb{L}_2$-error of $\bar{f}$ at $\bar{N}$ uniformly sampled test points in the domain.

4.2.1. *Data Scaling*

To increase the robustness of the Gaussian process surrogate model, we perform regression on scaled data. We use a technique called n^{th}-root scaling of the target vector $\mathbf{t}$. That is, for regression, we use the following scaled vector $\bar{\mathbf{t}} = \left(\bar{t}^{(1)}, \ldots, \bar{t}^{(N)}\right)^{\mathrm{T}}$ instead of $\mathbf{t}$.

$$\bar{t}^{(i)} = \mathrm{sgn}(t^{(i)}) \cdot \left|t^{(i)}\right|^{\frac{1}{n}}, \quad i = 1, \ldots, N. \tag{32}$$

The choice of the scaling factor n affects the regression performance. Too small n means that large scale variation still remains whereas too large n leads to too much flattening of data. We choose $n = 3$ [9] for all our numerical experiments in this work. Figure 2 demonstrates the advantage of using scaled data to train our Gaussian process regression model.

4.3. **Gradient-boosted decision tree**

Introduced in 1984, the CART (Classification And Regressor Trees) algorithm was developed by Breiman. L et al [23]. This supervised algorithm allows to predict a class or a quantity according to a training dataset. To explain this concept, a simple example is presented here. This section is subdivided into 2 parts, the first explaining how the decision tree is built, appearing as a weak model, and the second one about the building of strong machine learning models from weak models developed by Friedman et al. [24].

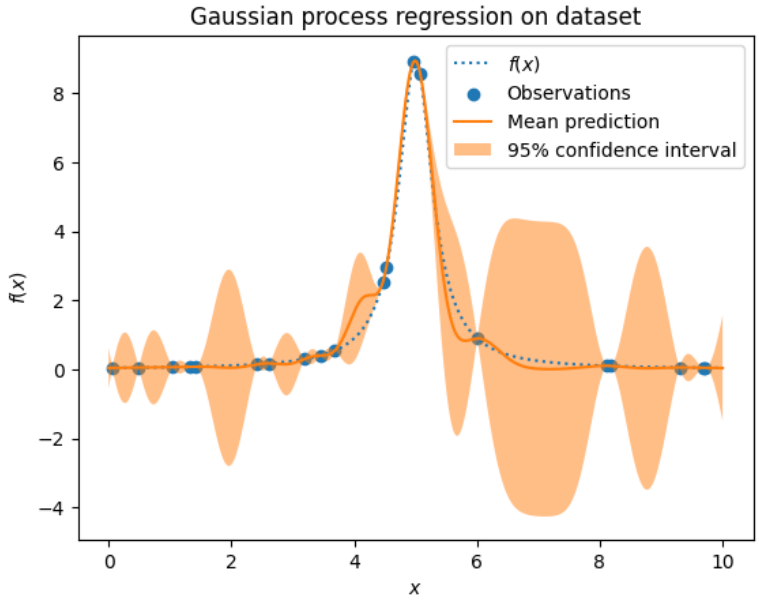
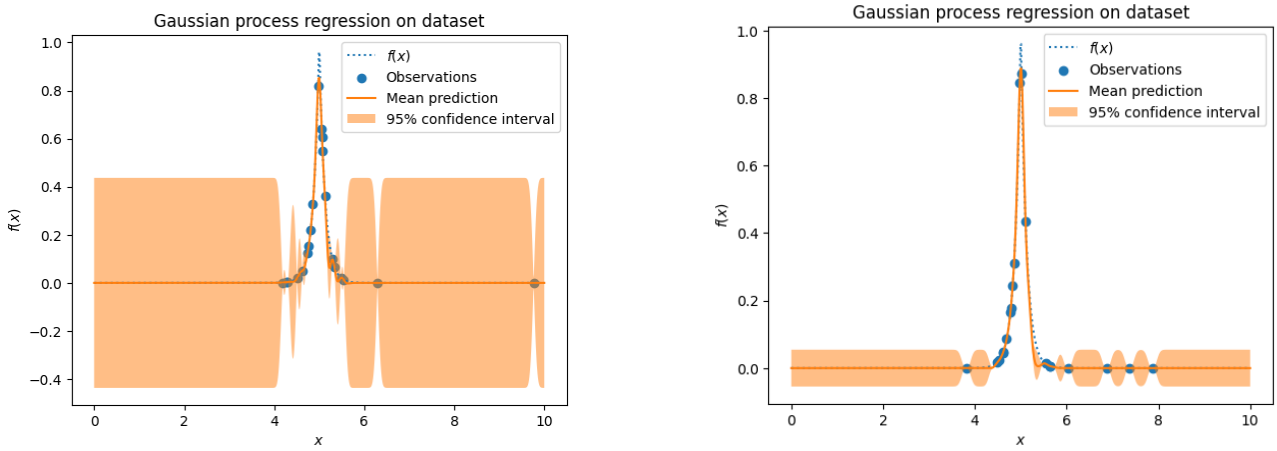


FIGURE 1. A simple Gaussian process regression example in one dimension. We aim to approximate the function $f(x) = \frac{9}{1+9(x-5)^2}$ using 20 randomly distributed training points here. The mean function $\hat{\mu}$ closely follows f and is exact at the training data points. Elsewhere, there is some uncertainty in the prediction which is quantified using the variance function $\hat{\sigma}^2$.



(A) With original training data

(B) With scaled training data

FIGURE 2. Demonstrating the effect of data scaling on a simple Gaussian process regression example in one dimension. We approximate the function $f(x) = \exp\{-8|x-5|\}$ using 20 training points from VEGAS [8]. We observe significantly reduced uncertainty on the surrogate model $\bar{f}$ trained using scaled training data. n^{th}-root scaling with $n = 3$ used here.

Algorithm 1: Gradient Boosting algorithm

Input: $(x_i, y_i)_{i=1,\dots,N}$, ϵ, ν, $\mathcal{M}$: model, i_{max}, $e_{tol} = 1.01\epsilon$

Compute $F_0 \leftarrow \dfrac{1}{N}\sum_{i=0}^{N} y_i$;

while $e_{tol} < \epsilon$ *or* $i < i_{max}$ **do**

$e \leftarrow \dfrac{1}{N}\sum_{i=1}^{N} |F_0 - y_i|$;

$f \leftarrow \mathcal{M}(e)$;

$F_1 \leftarrow F_0 - \nu f$ using (34) and $\mathcal{S} = \{(x_i, y_i)_{i=1,\dots,N}\}$;

$i \leftarrow i + 1$;

$e_{tol} \leftarrow |F_1 - F_0|$;

$F_0 \leftarrow F_1$

end

Output: F_0 : the strong model built

4.3.1. *Decision Tree*

In this part, only the case of the decision tree regressor will be discussed. Let x_i be an $\mathbb{R}^d$ vector, and y_i be a scalar corresponding to the result of the true decision according to x_i. We denote the set of ground truth as $\mathcal{S} = \{x_i, y_i\}_{i=1,\dots,N}$. For simplicity's sake, we consider dimension $d = 1$. Let $\mathcal{M} : \Omega \subset \mathbb{R}^d \longrightarrow \mathbb{R}$ be a decision tree with a depth layer and M subdivisions from the previous node. Mathematically, this model $\mathcal{M}$ can be written as a piecewise constant function as follows :

$$\mathcal{M} : x \to \sum_{i}^{M} \mathbb{1}_{\Omega_i}(x)\gamma_i$$

in which $\Omega = \bigcup_{i=1}^{M} \Omega_i$ and $\gamma_i \in \mathbb{R}$ for all $i \in \{1, \dots, M\}$. We use the following notation:

$$\hat{y}_A = \frac{1}{\#A}\sum_{y \in A} y \tag{33}$$

with $\#A$ corresponding to the cardinality of A. Suppose the Ω space is defined as an interval $[\alpha_0, \alpha_\infty[$. The best fit of the model (with M subdivisions) on the set $\Omega = \bigcup_{i=0}^{M-1} [\alpha_i, \alpha_{i+1}[$, (with $a_\infty = a_M$) according the ground truth $\mathcal{S}$ is defined as follows :

$$\min_{(\alpha_1,\dots,\alpha_{M-1}) \in \mathbb{R}^{M-1}} \sum_{i=1}^{M} \left[\sum_{y \in \mathcal{S}_i} (y - \hat{y}_{\mathcal{S}_i})^2 \right] \tag{34}$$

within which the set $\mathcal{S}_i$ is defined as the set of $y_i \in \mathcal{S}$ such that $x_i \in [\alpha_i, \alpha_{i+1}[$. The concept is similar for dimensions greater than 1, for more details, we refer the readers to [23].

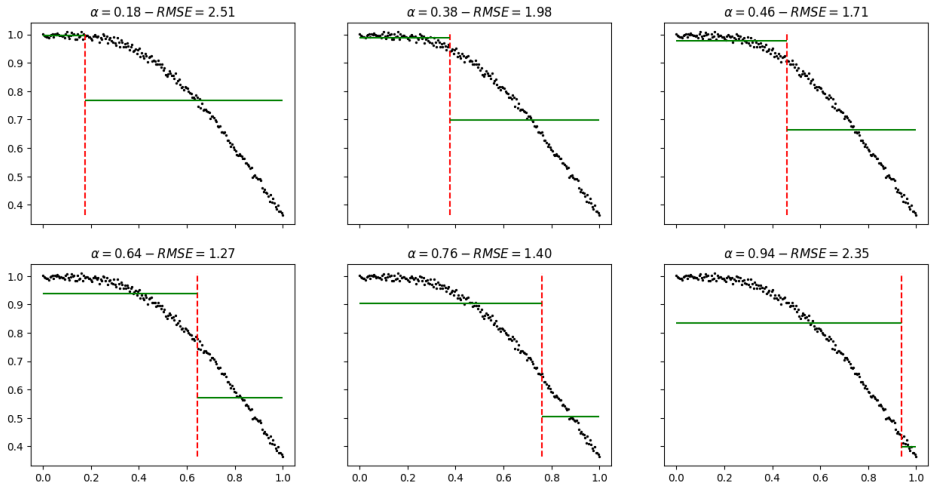


FIGURE 3. Different curves approximating the ground truth dataset (black points), with $x \in \Omega$. In red, the delimitation $x = \alpha$ between the both sides, and in green, the lines correspond to the mean value respectively on their own subdomain.

Figure 3 shows the piecewise constant function according to the value α, and the relative error using the root of the mean squared error. The best model as defined in the equation (34) is achieved for $\alpha = 0.64$, that means the root mean squared error (RMSE) is minimal at this point.

4.3.2. *Gradient Boosting*

The gradient boosting algorithm is a process allowing to build a strong model from a weak one. From an initial state of the strong model $F_0(x)$, which will be considered as being equal to the mean of all known data y_i for all x in Ω, a weak model is developed, in our case. The general algorithm of the gradient boosting is as follows:

$$F_{i+1} = F_i - \nu f_i \tag{35}$$

f_i corresponds to the weak model, this one will be built from the error e_i of the strong model relative to the ground truth. The error will be injected in a weak model $\mathcal{M}$ to predict the error. Otherwise, the equation (35) is rewritten as follows:

$$F_{i+1} = F_i - \nu \mathcal{M}(e_i) \tag{36}$$

Algorithm 1 presents the different steps of the gradient boosting algorithm. Generally, to avoid the problem of non-convergence of the model, a large number of loop iterations is used.

4.3.3. *Model used*

There are a number of parameters to be chosen in Algorithm 1. We specify the parameters we use for the GBDT model in this work. We decide to set the number of maximum layers of the decision tree to 4. This tree is considered binary. The subsampling fraction parameter is set at 0.3, corresponding to the fraction of data used for training. The learning rate for the gradient-boosting algorithm is set at 0.1. The tolerance value for convergence of the model is $\epsilon = 10^{-4}$.

d	$\|c\|$		f_1	f_3	f_4	f_5	f_6
6	8	Training (s)	1.35	1.38	1.19	1.63	1.62
		Integration (s)	13.67	13.97	15.28	15.41	15.20
		Rel. Error (%)	0.6	0.009	0.08	0.28	0.23
6	1	Training (s)	1.12	0.82	1.25	1.25	1.18
		Integration (s)	11.10	17.26	12.32	11.22	10.80
		Rel. Error (%)	0.026	0.0016*	0.042	0.036	1.63
4	8	Training (s)	0.86	1.06	1.93	1.58	1.29
		Integration (s)	12.21	11.81	22.33	17.99	13.87
		Rel. Error (%)	1.5	0.023	0.25	0.07	1.34

TABLE 3. Performance of integration on GBDT models of various functions for two dimensions d and two integrand difficulties $\|c\|$. (Average over 10 runs)

5. An optimal, adaptive machine learning integration algorithm

Given the original integrand $f : \mathbb{R}^d \to \mathbb{R}$ and its corresponding machine learning based surrogate $\bar{f} : \mathbb{R}^d \to \mathbb{R}$, we aim to build an efficient Monte Carlo estimator $\mathcal{A}^{\mathrm{ML}}$ of g such that

$$\mathbb{P}\left[\left|\mathcal{A}^{\mathrm{ML}} - g\right| \leq \mathrm{TOL}\right] \geq 1 - \alpha, \tag{37}$$

for a user-specified accuracy $\mathrm{TOL} > 0$ and constant $0 < \alpha \ll 1$ determining the confidence level. Using the surrogate $\bar{f}$, we can re-write g as:

$$g = \int_{[0,1]^d} f(\mathbf{x})\mathrm{d}\mathbf{x} = \int_{[0,1]^d} \bar{f}(\mathbf{x})\mathrm{d}\mathbf{x} + \underbrace{\int_{[0,1]^d} (f - \bar{f})(\mathbf{x})\mathrm{d}\mathbf{x}}_{\text{surrogate bias}}. \tag{38}$$

Our machine learning integration algorithm approximates both integrals in (38) using independent Monte Carlo averages, giving rise to $\mathcal{A}^{\mathrm{ML}}$.

$$\mathcal{A}^{\mathrm{ML}} = \frac{1}{\bar{M}} \sum_{m=1}^{\bar{M}} \bar{f}(X^{(m)}) + \frac{1}{\tilde{M}} \sum_{m=1}^{\tilde{M}} (f - \bar{f})(X^{(m)}), \tag{39}$$

where $\bar{M}$ and $\tilde{M}$ are the number of samples used in estimating the two integrals in (38). As seen in (15), we need to estimate $\mathrm{Var}\left[\mathcal{A}^{\mathrm{ML}}\right]$ to quantify the error in our approximation of g:

$$\mathrm{Var}\left[\mathcal{A}^{\mathrm{ML}}\right] = \mathrm{Var}\left[\frac{1}{\bar{M}} \sum_{m=1}^{\bar{M}} \bar{f}(X^{(m)}) + \frac{1}{\tilde{M}} \sum_{m=1}^{\tilde{M}} (f - \bar{f})(X^{(m)})\right] \tag{40}$$

$$= \frac{1}{\bar{M}} \mathrm{Var}\left[\bar{f}(X)\right] + \frac{1}{\tilde{M}} \mathrm{Var}\left[(f - \bar{f})(X)\right]. \tag{41}$$

Note that we use the independence of the two Monte Carlo averages to go from (40) to (41). According to the central limit theorem [15], the error in our approximation is upper-bounded by $C_\alpha \sqrt{\mathrm{Var}\left[\mathcal{A}^{\mathrm{ML}}\right]}$ with a confidence of $1 - \alpha$. We would like to build a computationally optimal algorithm that minimizes the computational work

of $\mathcal{A}^{\mathrm{ML}}$ to achieve an accuracy of TOL. This can be posed as following optimization problem:

$$\begin{cases} \min\limits_{\{\bar{M},\tilde{M}\}} \mathrm{Work}[\mathcal{A}^{\mathrm{ML}}] = \bar{M}\bar{C} + \tilde{M}\tilde{C} \\ \quad \text{s.t. } C_\alpha \sqrt{\frac{1}{\bar{M}}\mathrm{Var}\left[\bar{f}(X)\right] + \frac{1}{\tilde{M}}\mathrm{Var}\left[(f-\bar{f})(X)\right]} \approx \mathrm{TOL} \end{cases}. \tag{42}$$

Here $\bar{C}$ is the cost of evaluating one sample of the surrogate function $\bar{f}(X)$ and $\tilde{C}$ is the cost of evaluating one sample of the surrogate bias $(f-\bar{f})(X)$. We solve the constrained optimization problem (42) using the Lagrange multiplier method to get,

$$\begin{aligned} \bar{M} &= \left(\frac{C_\alpha}{\mathrm{TOL}}\right)^2 \sqrt{\frac{\mathrm{Var}\left[\bar{f}(X)\right]}{\bar{C}}} \left(\sqrt{\mathrm{Var}\left[\bar{f}(X)\right]\bar{C}} + \sqrt{\mathrm{Var}\left[(f-\bar{f})(X)\right]\tilde{C}}\right), \\ \tilde{M} &= \left(\frac{C_\alpha}{\mathrm{TOL}}\right)^2 \sqrt{\frac{\mathrm{Var}\left[(f-\bar{f})(X)\right]}{\tilde{C}}} \left(\sqrt{\mathrm{Var}\left[\bar{f}(X)\right]\bar{C}} + \sqrt{\mathrm{Var}\left[(f-\bar{f})(X)\right]\tilde{C}}\right). \end{aligned} \tag{43}$$

By plugging in (43) to $\mathrm{Work}[\mathcal{A}^{\mathrm{ML}}]$, we get the following optimal computational cost of $\mathcal{A}^{\mathrm{ML}}$.

$$\mathrm{Work}[\mathcal{A}^{\mathrm{ML}}] = \left(\frac{C_\alpha}{\mathrm{TOL}}\right)^2 \left(\sqrt{\mathrm{Var}\left[\bar{f}(X)\right]\bar{C}} + \sqrt{\mathrm{Var}\left[(f-\bar{f})(X)\right]\tilde{C}}\right)^2. \tag{44}$$

We can perform a similar analysis for a plain Monte Carlo estimator $\mathcal{A}$. Analogous to (42), one can derive the optimal plain Monte Carlo estimator that minimizes the computational cost, while satisfying a given error tolerance TOL. The computational cost of this optimal estimator is given by:

$$\mathrm{Work}[\mathcal{A}] = \left(\frac{C_\alpha}{\mathrm{TOL}}\right)^2 \mathrm{Var}\left[f(X)\right] C, \tag{45}$$

where C is the cost of evaluating one sample of the original integrand $f(X)$. Comparing (45) with (44), we see that our machine learning estimator is better than the plain Monte Carlo estimator only if

$$\sqrt{\mathrm{Var}\left[\bar{f}(X)\right]\bar{C}} + \sqrt{\mathrm{Var}\left[(f-\bar{f})(X)\right]\tilde{C}} < \sqrt{\mathrm{Var}\left[f(X)\right]C}. \tag{46}$$

We make the following remarks from the above analysis:

- It is evident from Equation (43) that a small $\tilde{M}$ would be sufficient to achieve an accuracy of TOL, if the surrogate model $\bar{f}$ is a good approximation of f. This means that we require very few evaluations of f to estimate the surrogate bias in this case.
- $\bar{f} \approx f$ implies $\mathrm{Var}\left[f(X)\right] \approx \mathrm{Var}\left[\bar{f}(X)\right]$. This means that the condition (46) is satisfied only when $\bar{C} \ll C$. This motivates the use of machine learning surrogates, which are generally much cheaper to evaluate than integrands encontered in acoustic applications as discussed in Section 2 (see equations (5) to (7)).

From this, it is clear to see that it is important to have a highly accurate surrogate model to ensure that the machine learning estimator is computationally more efficient than a plain Monte Carlo estimator. Given that we have a machine learning-based surrogate model, we can now develop an algorithm that produces an estimator of g of accuracy TOL with confidence $1-\alpha$ in a computationally optimal manner. Algorithm 2, however, requires cheap and robust estimates of $\mathrm{Var}\left[\bar{f}(X)\right]$, $\mathrm{Var}\left[(f-\bar{f})(X)\right]$, $\bar{C}$, $\tilde{C}$. We look at this is done in more detail in Section 6.3.

Algorithm 2: An optimal machine learning integration algorithm

Input: f,$\bar{f}$,TOL,α
Estimate Var $\left[\bar{f}(X)\right]$, Var $\left[(f-\bar{f})(X)\right]$, $\bar{C}$, $\tilde{C}$;
Compute $\bar{M}$, $\tilde{M}$ using (43);
Estimate $\int_{[0,1]^d} \bar{f}(\mathbf{x})\mathrm{d}\mathbf{x}$ using VEGAS Monte Carlo with $\bar{M}$ samples;
Estimate the surrogate bias using VEGAS Monte Carlo with $\tilde{M}$ samples;
Output: $\mathcal{A}_{\mathrm{ML}}$

6. Numerical results

6.1. **Training Data**

To minimize the prediction error for a given training data set, it is important to collect the training samples at points with high function absolute values (importance sampling; see Section 3.1) and where the function changes rapidly (stratified sampling; see Section 3.2). The VEGAS Monte Carlo algorithm [8] chooses its quadrature points using exactly the same criterion. We, therefore, use the VEGAS algorithm as a generator of our training data points. However, we observed that the training samples produced by VEGAS tend to contain very few points where the function has very low values. This is an important limitation considering acoustics applications since, as pointed out in Section 2, the value of the integrand is zero inside the rigid surface. Some training points must be added inside the rigid surface so that the machine learning model estimate remains valid here. Otherwise, in high dimensions, this leads to poor function approximations in these regions. Hence, we make sure that 10% of our training data points include uniformly distributed sample points in the domain to ensure a good approximation of the functions throughout the domain. We use $N = 10^3$ training points to build all our surrogate functions with 3^{rd}-root scaling and standardization performed on all training data. We use $\bar{N} = 10^3$ uniformly sampled test points to measure the approximation quality of our surrogate models.

6.2. **VEGAS Setup**

In this work, we perform all numerical integration using the VEGAS Monte Carlo algorithm [8]. Specifically, we use Lepage's VEGAS Python library version 5.5 [25]. In this implementation, there are two damping parameters that control how quickly the importance and stratified sampling grids are updated between iterations. This ensures that the stratified grid adapts smoothly to the shape of the integrand without abrupt changes, leading to a more stable algorithm. We use the default values of the two damping parameters $\alpha = 0.5$ and $\beta = 0.75$ for all numerical integration tasks. Another important parameter is the number of iterations for the VEGAS grid adaptation N_{itn}. For all the forthcoming numerical experiments we set $N_{\mathrm{itn}} = 10$. The number of sample evaluations in the VEGAS algorithm is adaptively chosen according to (43) to satisfy an accuracy of TOL. VEGAS integral estimates are obtained by taking a weighted average of the estimates from each VEGAS iteration.

6.3. **Algorithm 2 Setup**

To ensure a fair comparison between all our test integrands, we worked with a relative accuracy $\mathrm{TOL}_{\mathrm{r}}$ instead of an absolute accuracy TOL. They are simply related to one another by $\mathrm{TOL} = \mathrm{TOL}_{\mathrm{r}} g$. For our numerical tests, we approximate g by a reference VEGAS Monte Carlo estimate using $N_{\mathrm{itn}} = 20$ iterations and $M = 10^6$ evaluation points. We set $\alpha = 0.05$, ensuring a confidence level in our estimates of 95%. From our discussion in Section 5, one important facet of Algorithm 2 is the cheap estimation of Var $\left[\bar{f}(X)\right]$, Var $\left[(f-\bar{f})(X)\right]$, $\bar{C}$, $\tilde{C}$. For this, we run a pilot VEGAS run with $N_{\mathrm{itn}} = 5$ and $M_{\mathrm{pilot}} = 10^3$ evaluation points, once for each integrand $\bar{f}$ and $f - \bar{f}$. We then estimate the variances Var $\left[\bar{f}(X)\right]$ and Var $\left[(f-\bar{f})(X)\right]$ using the sample variance of the obtained function evaluations. To estimate $\bar{C}$, $\tilde{C}$, we record the runtime required for the above pilot runs and divide them by the total number of function evaluations to obtain the required quantities.

6.4. **Family of integrand functions**

The 1D case

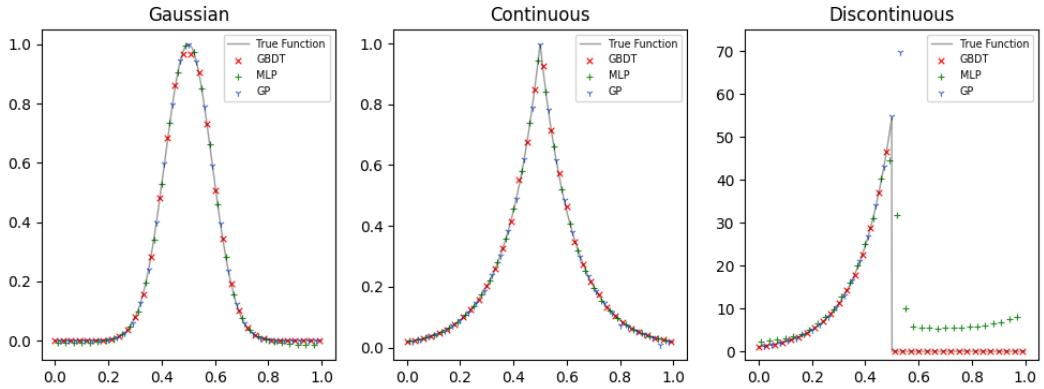


FIGURE 4. Comparison of the approximation

The multidimensional cases

In table 3, the result for the case $D = 6$ and $||c|| = 1$ for function f_3, we used 1^{st}-root data-scaling instead of 3^{rd}-root. This is due to the function values being small by several orders of magnitude. Indeed, the true integral of this function f_3 is of the order 10^{-10}. In this case, a majority of function points are lesser than 1, leading to an increase of the value and not decrease (which is desired) after scaling . The others functions, within the same case, present an integral value of an order of 10^{-1}.

6.5. **Algorithm 2 with Gaussian process regression**

We illustrate some numerical results of using Algorithm 2 where $\bar{f}$ is a surrogate obtained from Gaussian process regression. We test this on all five test integrands and compare the results using plain VEGAS Monte Carlo integration. We perform numerical tests on two values of the dimension, i.e. $d = 4, 6$ and two values of $\|c\| = 1, 8$. We use the `scikit` implementation [26] of Gaussian processes. The log-likelihood optimization problem is a smooth, non-convex, and high-dimensional optimization problem. Hence, we use the `L-BFGS-B` optimizer to solve this. We also run the optimization 10 times using 10 different, independent starting points to avoid convergence to local minima in a non-convex landscape. The training runtime for the Gaussian surrogate model scales with the number of training points alone and does not seem to dependent on the regularity or dimension of the function. The smooth functions $f_1 - f_3$ are very well approximated by the corresponding Gaussian process surrogate. In case of f_3, the approximation is almost exact since the integrand itself is Gaussian. Lower the value of $\|c\|$, higher the approximation quality of the Gaussian process surrogate, for a training data set of the same size. The VEGAS integration time of both the surrogate integral and the surrogate bias are more or less constant for both $d = 4$ and $d = 6$ for all functions. Figure 5 shows the exact relative error achieved by both $\mathcal{A}$ using plain VEGAS and $\mathcal{A}^{\text{ML}}$ using Algorithm 2. Both algorithms give good estimates of the integral for integrands $f_1 - f_4$ for all cases of d and $\|c\|$. Plain VEGAS gives unsatisfactory integral estimates for the discontinuous function f_5. This is because of unreliable estimates of $\text{Var}\left[f(X)\right]$. On the other hand, Algorithm 2 gives better integral estimates for f_5 as we get better estimates of $\text{Var}\left[\bar{f}(X)\right]$ for the smoother surrogate model $\bar{f}$. Tables 4-7 compares the average runtimes required to achieve 10% relative accuracy with 95% confidence recorded in the same runs as above for all cases. We see that Algorithm 2 using Gaussian process regression consistently shows very high runtimes for discontinuous function f_5 to achieve the tolerance requirement. We also see that plain VEGAS is computationally much cheaper than Algorithm 2 for

	f_1	f_2	f_3	f_4	f_5
VEGAS	0.0216	0.0258	0.0241	0.0213	0.041
ML + VEGAS	0.2028	0.1851	0.1959	0.2398	8.8491

TABLE 4. Algorithm 2 with Gaussian process regression. Comparing the integration runtime (in seconds) of Algorithm 2 to plain VEGAS Monte Carlo for test integrands to achieve an accuracy of $\mathrm{TOL_r} = 10\%$ with 95% confidence. (Average over 100 runs). Case: $d = 4, \|c\| = 1$

	f_1	f_2	f_3	f_4	f_5
VEGAS	0.1225	0.0244	0.0259	0.021	0.0281
ML + VEGAS	11.5422	0.1573	0.1839	0.3583	2.5091

TABLE 5. Algorithm 2 with Gaussian process regression. Comparing the integration runtime (in seconds) of Algorithm 2 to plain VEGAS Monte Carlo for test integrands to achieve an accuracy of $\mathrm{TOL_r} = 10\%$ with 95% confidence. (Average over 100 runs). Case: $d = 4, \|c\| = 8$

	f_1	f_2	f_3	f_4	f_5
VEGAS	0.021	0.0271	0.0304	0.0208	0.0426
ML + VEGAS	0.2702	0.3904	0.3388	0.3498	44.0773

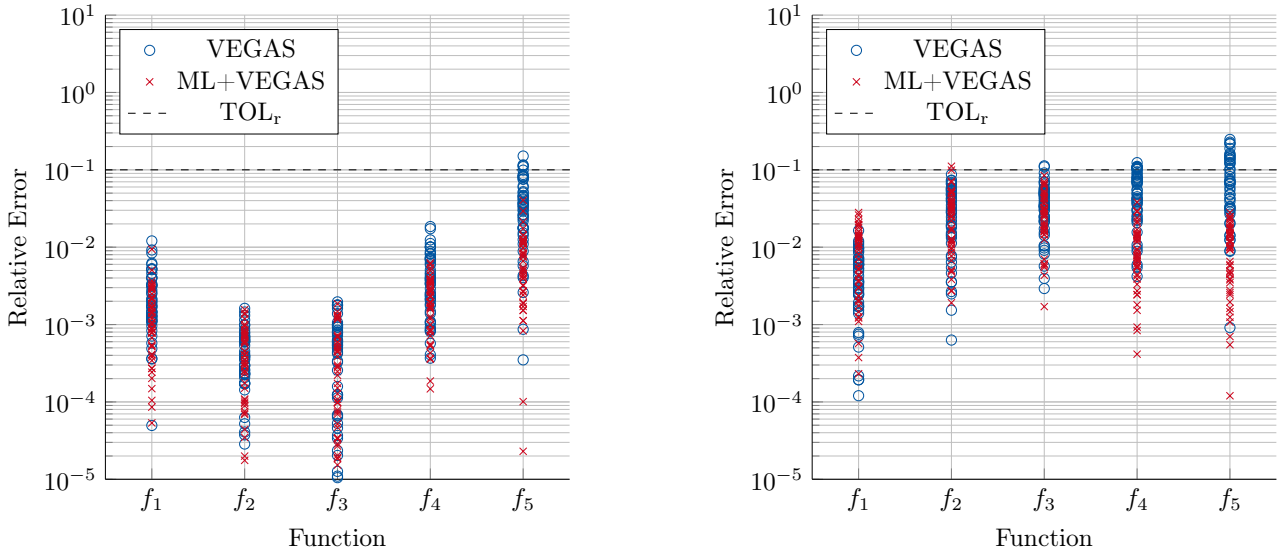
TABLE 6. Algorithm 2 with Gaussian process regression. Comparing the integration runtime (in seconds) of Algorithm 2 to plain VEGAS Monte Carlo for test integrands to achieve an accuracy of $\mathrm{TOL_r} = 10\%$ with 95% confidence. (Average over 100 runs). Case: $d = 6, \|c\| = 1$

	f_1	f_2	f_3	f_4	f_5
VEGAS	0.0745	0.0227	0.0224	0.0216	0.0405
ML + VEGAS	8.2235	0.2107	0.2193	0.2988	23.504

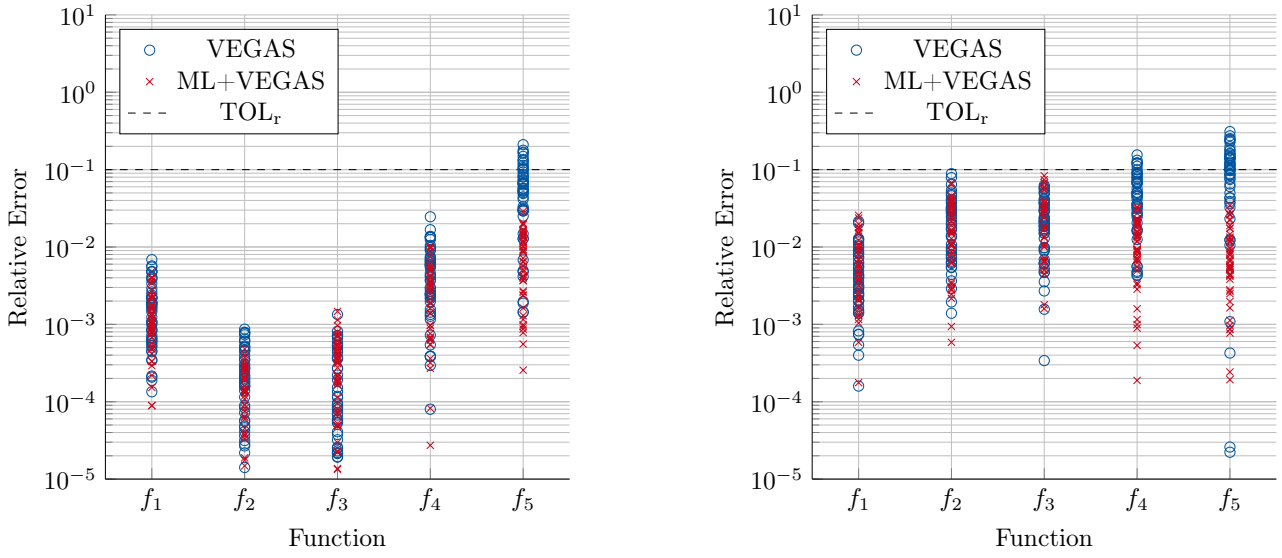
TABLE 7. Algorithm 2 with Gaussian process regression. Comparing the integration runtime (in seconds) of Algorithm 2 to plain VEGAS Monte Carlo for test integrands to achieve an accuracy of $\mathrm{TOL_r} = 10\%$ with 95% confidence. (Average over 100 runs). Case: $d = 6, \|c\| = 8$.

test integrands $f_1 - f_5$ mainly because the cost of evaluating the test integrand is much lower compared to the cost of evaluating the corresponding surrogate model. Since the surrogate evaluation cost remains of the same order regardless of the regularity or dimension of the problem, it can be expected that Algorithm 2 gives an advantage when the evaluation cost of f increases, as is prevalent in acoustic noise predictions.

We also compare Algorithm 2 with plain VEGAS over a range of tolerances. For this, we test both algorithms on the 6-dimensional Gaussian test integrand (f_3) with $\|c\| = 8$. We estimate g using both $\mathcal{A}$ and $\mathcal{A}^{\mathrm{ML}}$ over a range of relative accuracies (1-10%). We run both algorithms 100 times independently and compare their exact relative errors and average runtimes. Figure 6a shows that both Algorithm 2 and plain VEGAS achieves the relative error accuracy $\mathrm{TOL_r}$ with 95% confidence. However, it seems that the algorithm is over-estimating the required number of samples required to achieve $\mathrm{TOL_r}$. We claim that this is because of the mismatch in the sampling points used in estimating the variances and the points used to perform Monte Carlo integration. Figure 6b illustrates that Algorithm 2 takes more time on average than plain VEGAS for the test integrand f_3. Note that we exclude the runtime of training the surrogate model here. Hence, it is evident that the cost of sampling the original integrand is much cheaper than the cost of sampling the surrogate model. The sampling cost of the original integrand in more complex cases, like in acoustic noise predictions, would be much higher than the sampling cost of the surrogate model. This suggests that Algorithm 2 could perform better in more complex cases.

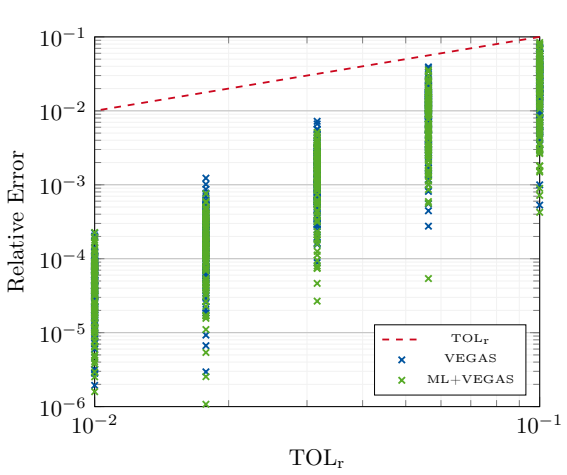


(A) $d = 4, \|c\| = 1$ (B) $d = 4, \|c\| = 8$

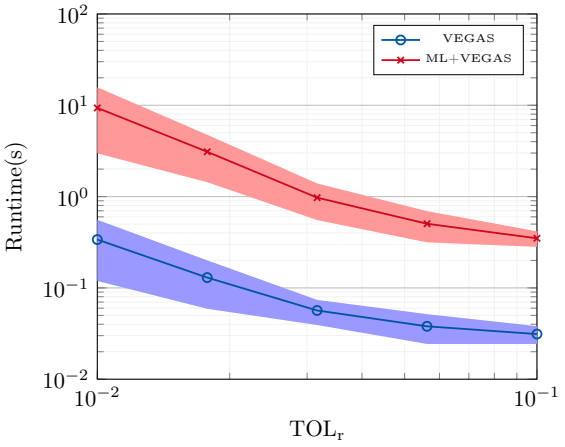


(C) $d = 6, \|c\| = 1$ (D) $d = 6, \|c\| = 8$

FIGURE 5. Algorithm 2 with Gaussian process regression. We run Algorithm 2 100 times with Gaussian process surrogate model for all five test integrands in various cases to achieve a relative accuracy of 10% with 95% confidence. We compare it with running plain VEGAS Monte Carlo 100 times for each test integrand with the same setting. Each marker in the plot is the exact relative error of the estimator, which is estimated using a reference approximation. In these plots, ML+VEGAS denotes runs of Algorithm 2, whereas VEGAS indicates runs of plain VEGAS Monte Carlo.



(A) Exact relative error versus input tolerance



(B) Average runtime (in seconds) versus input tolerance

FIGURE 6. Algorithm 1 with Gaussian process regression. We approximate g for the Gaussian test integrand f_3 over a range of relative accuracies. We run Algorithm 2 100 times to achieve these accuracies with 95% confidence. We compare it with running plain VEGAS Monte Carlo 100 times with the same setting. Each marker in the plot is the exact relative error of the estimator, which is estimated using a reference approximation. In these plots, ML+VEGAS denotes runs of Algorithm 2, whereas VEGAS indicates runs of plain VEGAS Monte Carlo.

7. Application to Acoustic Function

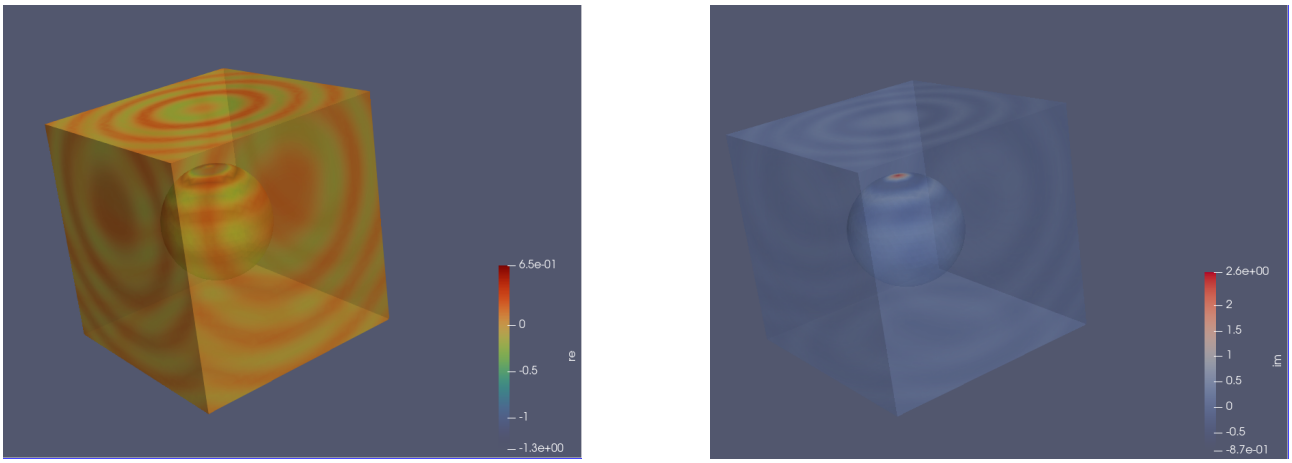
7.1. Function Description

We consider a function F_6 defined in a six-dimensional space, represented in a 3D domain. The function F_6 can be expressed as the product of a function F_3 which is the fundamental solution to the 3D Helmholtz equation for acoustics and an exponential function, defined as:

$$\begin{cases} F_3(x_1,y_1,z_1) = \dfrac{e^{ik_0\sqrt{(x_1-x_0)^2+(y_1-y_0)^2+(z_1-z_0)^2}}}{4\pi\sqrt{(x_1-x_0)^2+(y_1-y_0)^2+(z_1-z_0)^2}} \\ F_6(x_1,y_1,z_1,x_2,y_2,z_2) = F_3(x_1,y_1,z_1)e^{-(x_1-x_2)^2}e^{-(y_1-y_2)^2}e^{-(z_1-z_2)^2} \end{cases} \tag{47}$$

The function F_3 describes the response of the medium in each point (x_1,y_1,z_1) to an impulsive point source located in (x_0,y_0,z_0). k_0 is the acoustic wave-number and can be related to the frequency or to the acoustic wavelength. The Gaussian functions introduced in the three directions in F_6 represents the decay in the cross-correlation function for the turbulent velocity fluctuations inside a turbulent flow. This decay typically arises from the turbulence velocity fluctuations spectrum ϕ_{kl} in (5). The domain of integration is defined as the exterior of a sphere with a radius of $\frac{1}{4}$, which is inscribed within a $1\times1\times1$ cube centered at the origin. To provide an intuitive understanding of the function's behavior , Figure 7 shows the real part and imaginary part of F_3 in each point of the integration volume for $k_0 = 50\,\mathrm{m}^{-1}$ and $(x_0,y_0,z_0) = (0.22\,\mathrm{m},0,0)$. Since it is convenient to consider an (hyper-)cubic integration domain in many quadrature methods, we decide to consider a zero-integrand inside the boundary surface (here, a sphere of radius $\frac{1}{4}$).

Evaluating the function F_6 poses substantial computational challenges due to its high dimension and complex nature, which arise from the Helmholtz equation's solution for acoustics. The intricate spatial variations, as seen in the heat-map visualizations of both its real and imaginary parts, indicate regions of rapid changes which



(A) Real part of F_3 (B) Imaginary part of F_3

FIGURE 7. Heatmap representations of the real and imaginary parts of the function F_3 for $(x_0, y_0, z_0) = (0.22\,\mathrm{m}, 0, 0)$ and $k_0 = 50\,\mathrm{m}^{-1}$.

can lead to integration difficulties. For functions like F_6, direct application of VEGAS leads to a substantial number of evaluations, greatly increasing computational costs. Using models such as MLP, GBDT, and GP to approximate F_6, we can provide VEGAS with a more tractable version of the function, thus enhancing its performance. From Section 6, we observed that the MLP surrogate had the best generalization potential to efficiently approximate functions with varying regularity. Section 6.5 illustrated the limitations of using GP surrogates to approximate non-differentiable or continuous-function.From Table 3, it is evident that MLP surrogates performed better than the GBDT surrogates in terms of relative errors. Hence, we use the MLP surrogate approximator to capture the essential features of F_6 while reducing the computational cost. This makes the integration process not only faster but also potentially more accurate, as the approximators can smooth out erratic behavior that might otherwise hinder the VEGAS algorithm.

7.2. Integration results

Verification case - Computing the volume of the integration domain

Before dealing with functions F_3 or F_6, we decide to focus on one last verification and to check if the procedure introduced in this paper is able to compute the volume of the integration domain. To do that, we consider the same integration domain as defined in figure 7 and an integrand equal to 1. These computations were performed using the MLP model, as detailed in 4.1. The architecture used for this calculation is defined as follows:

- **First Hidden Layer**: $\mathbf{h}_1 = \tanh(\mathbf{W}_1\mathbf{h_0} + \mathbf{b}_1)$ where $\mathbf{W}_1 \in \mathbb{R}^{n\times 3}$ and $\mathbf{b}_1 \in \mathbb{R}^n$.
- **Hidden Layers**: $\mathbf{h}_i = \tanh(\mathbf{W}_i\mathbf{h}_{i-1} + \mathbf{b}_i)$ where $\mathbf{W}_i \in \mathbb{R}^{n\times n}, \mathbf{b}_i \in \mathbb{R}^n$ with $i \in \{1 \dots l\}$.
- **Output Layer**: $y = \mathbf{W}_3\mathbf{h}_2 + b_3$ where $\mathbf{W}_3 \in \mathbb{R}^{1\times n}$ and $b_3 \in \mathbb{R}$.

with a mean-squared error loss used for all simulation. The methodology is outlined as follows:

(1) Integrate the function using the VEGAS algorithm, with a relative tolerance of 5%.

(2) Build a first grid using the $N_{\text{eval}} = 2500$ points distributed according to the VEGAS algorithm.

(3) Add $N_{\text{add}} = 2500$ points, chosen randomly in the entire integration domain (including the sphere)

	Integral estimate	Deviation
Exact	0.9346	-
VEGAS	0.9347	0.08%
ML + VEGAS	0.9363	0.57%

TABLE 8. Comparison between the volume computed using the VEGAS algorithm and the methodology introduced in section 7.2. (Average over 10 runs).

(4) Train the MLP model ($n = 128, l = 3$) on the standardized values of the characteristic function at these points up to a loss function of 0.001. To do so we consider 16 batches of the data that are randomly shuffled on the points.

(5) Integrate the approximation of the MLP model, using a relative tolerance of 5%.

(6) Integrate the error function $\chi(x_1, x_2, x_3) - y(x_1, x_2, x_3)$, using a relative tolerance of 5%.

(7) Sum the two integrals from the previous steps.

It is important to note that the absolute tolerances derived from the VEGAS algorithm for the model and for the error function do not collectively constitute a reliable error bound for the integral estimation. Table 8 shows a comparison between the volume computed using the VEGAS algorithm and the ML integration. It appears that both approaches yield very good accuracy for a very few integration points.

3D Case - Integrating function F_3

For the function F_3 as defined in equation (47), a modification in the choice of activation function was necessary. Instead of employing tanh, the $\mathrm{ReLU}(x) = \max(0, x)$ activation function is used. Figure 8 shows a comparison between the function that we want to integrate and the ML approximation using the MLP model along the diagonal of the domain. The model for the real part and for the imaginary part are trained using the same set of points. The function is evaluated over the diagonal of the domain (i.e $F_3(x, x, x) \quad \forall x \in [-0.5, 0.5]$) and is taken equal to zero inside the sphere. Two strategies are compared. In the first one, we only use the N_{eval}-points grid computed by the VEGAS algorithm as training points for the MLP model. In the second strategy, we add N_{add} points randomly distributed over the integration domain, including the interior of the sphere. As discussed in section 4, it is expected that adding these training points will improve the model approximation inside the sphere where there are a very few points in the grid computed by VEGAS algorithm. It appears that a great improvement in the accuracy of the approximation can be obtained using the second strategy. However, this accuracy is mostly improved far from the sphere which is not expected. Further investigations are required to explain this observation.

Integrating the function F_3 poses consequent numerical challenges. For $k_0 = 10\,\mathrm{m}^{-1}$ or $k_0 = 50\,\mathrm{m}^{-1}$, the integral of the error function $(F_3 - \tilde{F}_3)$ appears to be of the same order of magnitude as the integral of the function itself. We conclude that we did not manage to properly train the model. In order to continue our investigation, we decide to reduce the value of k_0, to a point where the function F_3 does not oscillate in the domain. Using $k_0 = 1\,\mathrm{m}^{-1}$, the wavelength is equal to 7.56 m which is larger than the integration domain. Table 9 shows a comparison of the integral estimated by VEGAS or by the method based on the MLP model. The methodology is outlined as follows:

(1) Integrate the function using the VEGAS algorithm, with a relative tolerance of 5%.

(2) Build a first grid using the $N_{\mathrm{eval}} = 10^5$ points distributed according to the VEGAS algorithm.

(3) Add an additional $N_{\mathrm{add}} = 10^5$ points, chosen randomly in the entire integration domain (including the sphere)

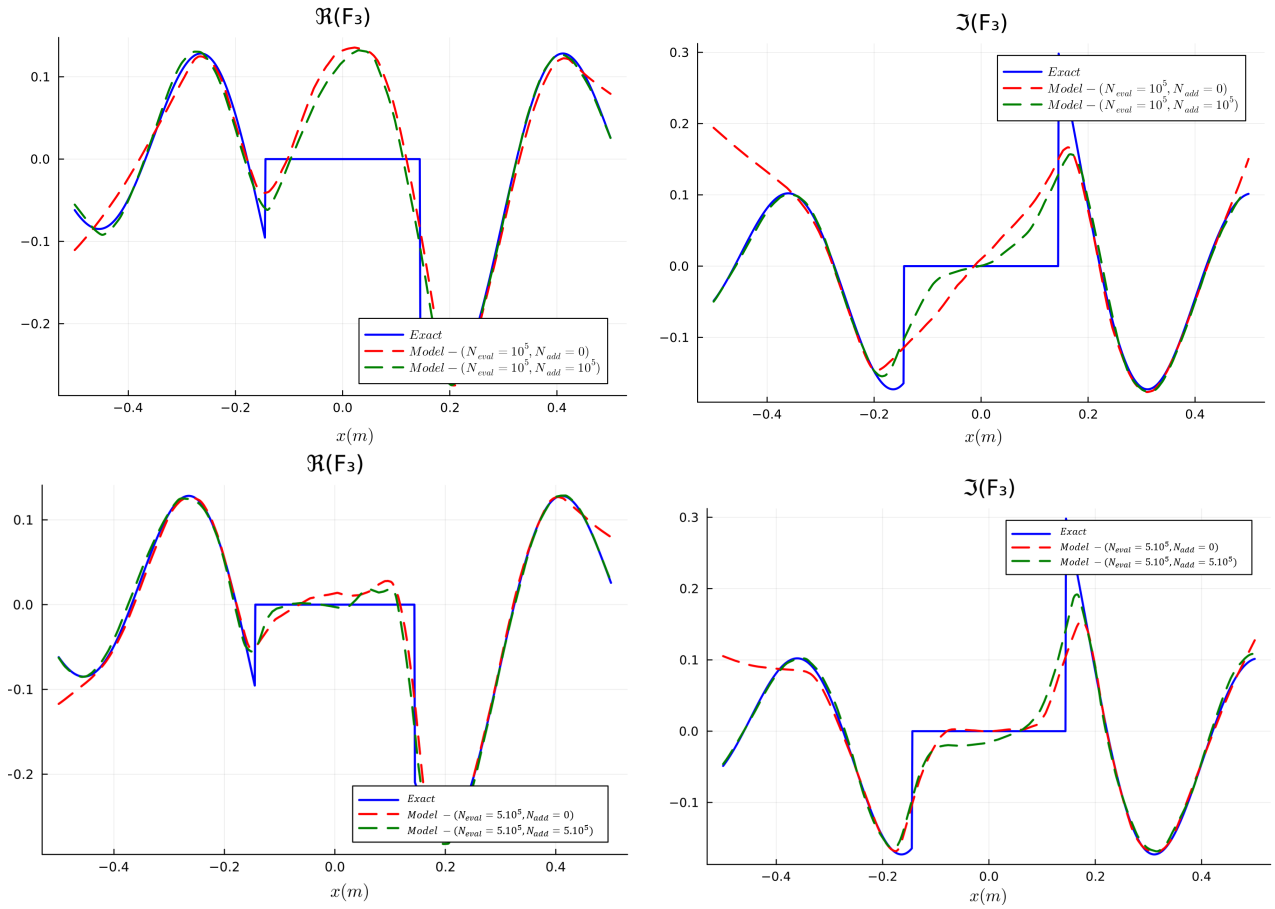


FIGURE 8. Comparison between the function F_3 and the ML approximation using the MLP model along the diagonal of the domain (i.e $F_3(x, x, x), \quad \forall x \in [-0.5, 0.5]$). Impact of the number of training points N_{eval} and N_{add} on the ML approximation for $k_0 = 10\,\mathrm{m}^{-1}$.

(4) Train the MLP model ($n = 128, l = 3$) on the standardized values of the characteristic function at these points up to a loss function of 0.001. To do so, be consider 16 batch of the data that are randomly shuffle on the points.

(5) Integrate the approximation of the MLP model, using a relative tolerance of 5%.

(6) Integrate the error function $\chi(x_1, x_2, x_3) - y(x_1, x_2, x_3)$, using a relative tolerance of 5%.

(7) Sum the two integrals from the previous steps.

It appears that we achieve a good accuracy on the result using a very small number of integration points. The time required to evaluate the function F_3 being of same order of magnitude than the time required to evaluate the MLP model.

6D Case - Integrating function F_6

In the 6D scenario of (47), the results are akin to those observed in lower dimensions. Table 10 shows a comparison of the integral estimated by VEGAS or by the method based on the MLP model using $N_{eval} = N_{add} = 10^5$ integration points. We use the same methodology as presented in section 7.2. Again, it appears

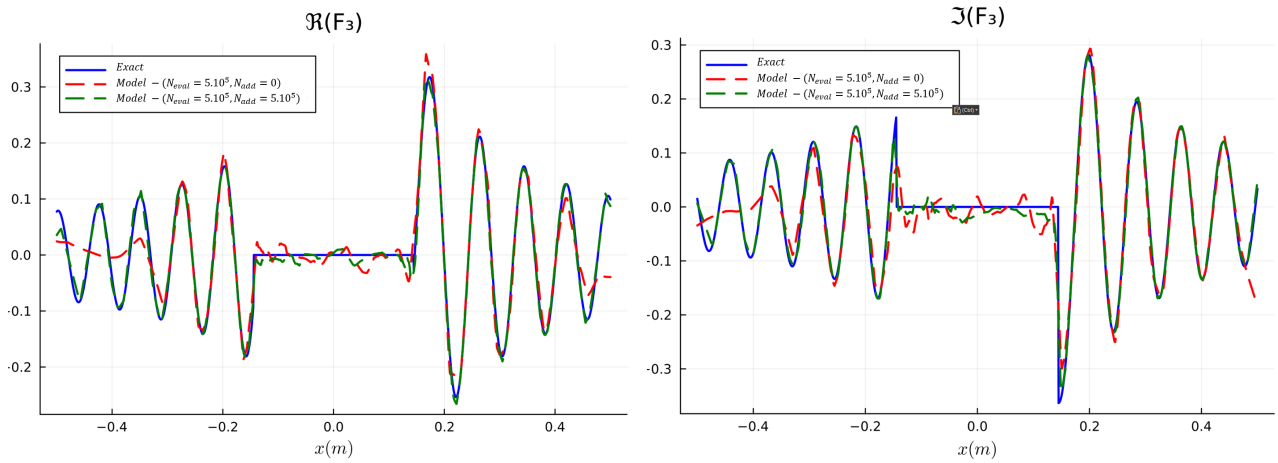


FIGURE 9. Comparison between the function F_3 and the ML approximation using the MLP model along the diagonal of the domain (i.e $F_3(x, x, x), \quad \forall x \in [-0.5, 0.5]$) for $k_0 = 50\,\mathrm{m}^{-1}$.

	Integral estimate	Deviation	Computation time
VEGAS	0.1388 + 0.0706i	-	0.08 s
ML + VEGAS	0.1443 + 0.0707i	3.6 %	11.6 s

TABLE 9. Comparison between the integral computed using the VEGAS algorithm directly on F_3 and on the ML based methodology for $k_0 = 1\,\mathrm{m}^{-1}$ and using $N_{\mathrm{eval}} = 10^5$ and $N_{\mathrm{add}} = 10^5$. (Average over 10 runs).

	Integral estimate	Deviation	Computation time
VEGAS	0.09031 + 0.04463i	-	0.11 s
ML + VEGAS	0.1049 + 0.04469i	14.5 %	18.2 s

TABLE 10. Comparison between the integral computed using the VEGAS algorithm directly on F_6 and on the ML based methodology for $k_0 = 1\,\mathrm{m}^{-1}$ and using $N_{\mathrm{eval}} = 10^5$, and $N_{\mathrm{add}} = 10^5$. (Average over 10 runs).

that a consequently larger amount of time is required to perform the integration using the MLP model than using the function F_6 directly. Increasing k_0 yields the same conclusion as before and we did not manage to properly integrate the function F_6 using the MLP model. Also, increasing the number of integration points did not improve the results and further investigations are required.

CONCLUSION

In this study, we performed a comprehensive comparison between traditional and machine learning approaches for complex integration in acoustic modelling. Our machine learning approach was built around replacing the true integrand with a machine learning surrogate model and then performing numerical integration using the VEGAS algorithm [27]. We first compared three machine learning models: multi-layer perceptrons, gradient-boosting decision trees, and Gaussian processes; to reconstruct 6 test integrands of varying regularity, smoothness, and oscillatory behaviour. Our findings revealed that integrating these machine learning models present a viable alternative to direct integration using VEGAS, especially when dealing with time-intensive function evaluations. We also observed that MLP surrogate models demonstrate better approximation quality,

as compared to the GP and GBDT models, for the canonical functions. For fair comparison, we also developed a methodology in this work to ascertain the optimal number of sampling points for the VEGAS algorithm by minimizing the computational work required to achieve a certain accuracy in our estimation. Subsequently, we tested our numerical integration algorithm using the MLP model on 3- and 6-dimensional applications of acoustic wave propagation. The results in the 3-dimensional test case, characterized by low oscillation, were promising, indicating the potential of MLP to handle moderately complex functions. Our tests suggested, however, that MLP models struggled with highly oscillatory functions, suggesting the need for further refinement in our surrogate building technique to adequately capture high-frequency behaviour. Our experiments also suggested that handling more intricate 6-dimensional functions was also feasible, given advanced computational resources. In conclusion, this study underscored the potential of machine learning methods for multidimensional integration in acoustic modelling. VEGAS remains the benchmark for the best performance in direct numerical integration. Our results demonstrated that machine learning methods achieve acceptable performance, proving the feasibility of this approach. However, further work is required to determine whether machine learning methods can surpass VEGAS in terms of performance. The limitation of machine learning models to capture high-frequency characteristics, requires further refinement of these models to ensure widespread applicability and enhanced precision, leading to more accurate and computationally cheaper acoustic predictions.

References

[1] J. A. Hildebrand, "Anthropogenic and natural sources of ambient noise in the ocean," *Marine Ecology Progress Series*, vol. 395, pp. 5–20, 2009.

[2] D. Wittekind and M. Schuster, "Propeller cavitation noise and background noise in the sea," *Ocean Engineering*, vol. 120, pp. 116–121, 2016.

[3] T. Joung, D. Hwang, and J. Shin, "An introduction of the international maritime policy trend on underwater radiated noise from shipping," *Journal of International Maritime Safety, Environmental Affairs, and Shipping*, vol. 6, no. 4, pp. 216–223, 2022.

[4] C. Bogey, C. Bailly, and D. Juvé, "Computation of flow noise using source terms in linearized euler's equations," *AIAA Journal*, vol. 40, pp. 235–243, 2002.

[5] S. Chaillat, B. Cotté, J.-F. Mercier, G. Serre, and N. Trafny, "Efficient evaluation of three-dimensional helmholtz green's functions tailored to arbitrary rigid geometries for flow noise simulations," *Journal of Computational Physics*, vol. 452, p. 110915, 2022.

[6] M. Kastner, "Monte carlo methods in statistical physics: Mathematical foundations and strategies," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, pp. 1589–1602, jun 2010.

[7] W. H. Press, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[8] G. P. Lepage, "A new algorithm for adaptive multidimensional integration," *Journal of Computational Physics*, vol. 27, no. 2, pp. 192–203, 1978.

[9] B. Yoon, "A machine learning approach for efficient multi-dimensional integration," *Scientific Reports*, vol. 11, sep 2021.

[10] R. Gupta, T. Bhattacharya, and B. Yoon, "Ai and theoretical particle physics," 2022.

[11] C. Gao, J. Isaacson, and C. Krause, "i- flow: High-dimensional integration and sampling with normalizing flows," *Machine Learning: Science and Technology*, vol. 1, p. 045023, oct 2020.

[12] J. Lighthill, "On sound generated aerodynamically. I. General theory," *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 211, pp. 564–587, 1952.

[13] S. Glegg and W. Devenport, *Aeroacoustics of low Mach number flows: Fundamentals, analysis, and measurement*. Academic Press, 2017.

[14] N. Trafny, G. Serre, B. Cotté, and J.-F. Mercier, "A stochastic volume approach based on tailored green's functions for airfoil noise prediction at low mach number," *Journal of Sound and Vibration*, vol. 551, p. 117603, 2023.

[15] M. Lefebvre, *Applied probability and statistics*. Springer Science & Business Media, 2006.

[16] W. H. Press and G. R. Farrar, "Recursive stratified sampling for multidimensional monte carlo integration," *Computers in Physics*, vol. 4, no. 2, pp. 190–195, 1990.

[17] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[18] A. Rana, A. Singh Rawat, A. Bijalwan, and H. Bahuguna, "Application of multi layer (perceptron) artificial neural network in the diagnosis system: A systematic review," in *2018 International Conference on Research in Intelligent and Computing in Engineering (RICE)*, pp. 1–6, 2018.

[19] P. Soni, Y. Tewari, and D. Krishnan, "Machine learning approaches in stock price prediction: A systematic review," *Journal of Physics: Conference Series*, vol. 2161, p. 012065, jan 2022.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[22] C. Williams and C. Rasmussen, "Gaussian processes for regression," *Advances in neural information processing systems*, vol. 8, 1995.

[23] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*. Taylor & Francis, 1984.

[24] J. H. Friedman, "Greedy function approximation: A gradient boosting machine.," *Annals of Statistics*, vol. 29, pp. 1189–1232, 2001.

[25] P. Lepage, "gplepage/vegas: vegas version 5.5," Oct. 2023.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] G. P. Lepage, "Adaptive multidimensional integration: vegas enhanced," *Journal of Computational Physics*, vol. 439, p. 110386, aug 2021.