AIAA JOURNAL
Vol. 51, No. 8, August 2013

Using Supervised Learning to Improve Monte Carlo Integral Estimation

Brendan Tracey*
Stanford University, Stanford, California 94305
David Wolpert†
Los Alamos National Laboratory, Los Alamos, New Mexico 87545
and
Juan J. Alonso‡
Stanford University, Stanford, California 94305

DOI: 10.2514/1.J051655

Monte Carlo techniques are used to estimate the integrals of a function using randomly generated samples. The interest in uncertainty quantification and robust design makes calculating the expected values of such functions (e.g., performance measures) important. Recent developments in scramjets, aircraft technology forecasting, structural reliability, and robust low-boom aircraft designs use Monte Carlo techniques to ensure the appropriate quantification of uncertainties. Because of high variance and slow convergence, Monte Carlo techniques require a large number of function evaluations, limiting the fidelity of the tools that can be used to predict performance. Stacked Monte Carlo is presented, which is a new method for postprocessing an existing set of Monte Carlo samples to improve integral estimation. Stacked Monte Carlo is based on combining fitting functions with cross-validation and should reduce the variance of any type of Monte Carlo integral estimate (importance sampling, quasi-Monte Carlo, etc.) without adding bias. An extensive set of experiments is reported, confirming that the stacked Monte Carlo estimate is more accurate than both the unprocessed Monte Carlo estimate and the estimate from a functional fit. Stacked Monte Carlo is applied to estimate the fuel-burn metrics of future commercial aircraft and sonic boom loudness measures, and the efficiency of Monte Carlo is compared with that of more standard methods. It is shown that for negligible, additional, computational cost, significant increases in accuracy are gained.

Nomenclature

b	=	bias of M
D_x	=	set of samples of $f(x)$
$D_x(i)$	=	ith sample of $f(x)$
$D_x^{i,\text{test}}$	=	subset of samples in the ith testing set
$D_x^{i,\text{train}}$	=	subset of samples in the ith training set
$\mathbb{E}[\cdot]$	=	expected value
$f(D_x(i))$	=	function value at the ith sample
$f(x)$	=	objective function
$\hat{f}_M$	=	estimate of $\hat{f}$ from M
$\hat{f}$	=	true expected value of $f(x)$
$g(x)$	=	fitting algorithm
$g_i(D_x(i))$	=	prediction of fit g_i at $D_x(i)$
$g_i(x)$	=	ith fit to $f(x)$
$\hat{g}$	=	expected value of $g(x)$
k	=	number of folds
L_i	=	likelihood of the expected value of the ith fold
M	=	estimator of $\hat{f}$
m_i	=	number of samples in the ith testing set
N	=	number of samples
N_i	=	number of samples in the ith training set
$p(x)$	=	probability distribution of x
$q(x)$	=	alternate sample distribution
$r(x)$	=	fitting algorithm for $p(x)$
v	=	variance of M
x	=	input parameters
α	=	free parameter of StackMC
β	=	free parameter in the fitting algorithm
ρ	=	correlation
σ	=	standard deviation

I. Introduction

ASSESSING the effects of uncertainties on system performance is fundamental in aerospace design. An ideal system is one that performs well under a wide variety of conditions and that is unlikely to fail. However, a system that is designed for optimal performance under the *nominal* operating conditions can see severe performance degradation at even slightly *off-design* conditions. Aerospace systems rarely operate at precisely the design condition, and a robust design approach dictates trading some performance at the nominal condition for improved performance over a wide range of operating conditions. However, certain input conditions will occur rarely or never, and adding robustness for these conditions will degrade the average system performance. We are often interested in optimizing the *expected* performance of a system rather than the performance at any specific operating point.

Formally, we are interested in an integral of the form

$$\mathbb{E}[f] = \hat{f} = \int f(x)p(x)\,\mathrm{d}x \tag{1}$$

where $f(x)$ is the (potentially multivariate) objective function to be optimized, and $p(x)$ is the probability density function [or probability distribution in which case Eq. (1) is actually a summation] from which x is generated.

Presented as Paper 2011-1843 at the 52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Denver, CO, 4–7 April 2011; received 3 January 2012; revision received 28 February 2013; accepted for publication 4 March 2013; published online 19 June 2013.

*Ph.D. Candidate, Department of Aeronautics and Astronautics, Durand Building 496 Lomita Mall. Member AIAA.

†Research Scientist, Santa Fe Institute, Information Sciences Group, 1399 Hyde Park Road, Mail Stop 460.

‡Associate Professor, Department of Aeronautics and Astronautics, Durand Building 496 Lomita Mall. Associate Fellow AIAA.

Evaluating Eq. (1) is nontrivial, but many problems in aerospace require its computation, such as evaluating failure probabilities [1], performing robust design [2], examining uncertainties in fluid-structure interactions [3], and aerothermal interactions [4]. When the objective function is high-dimensional and/or expensive to evaluate, standard quadrature techniques (such as Simpson's rule) become intractable. Polynomial chaos methods [5–8] are very effective for small numbers of specific classes of uncertain variables (Gaussian, exponential, uniform, etc.). However, polynomial chaos methods are not suitable for all problems, as it is hard to incorporate nonparametric uncertainty, and the computational cost grows exponentially with the number of uncertain parameters. Sparse grid and variations of stochastic collocation techniques have been developed, however, to ameliorate some of these shortcomings [9]. Monte Carlo (MC) methods are powerful techniques that have become the standard for integral estimation due to their ability to incorporate any form of uncertainty and their reasonable scaling properties. However, MC techniques also have their drawbacks; the required computational expense, although less than quadrature techniques, is still prohibitive in many situations. The fundamental question is whether a technique exists to significantly lower the cost of estimating Eq. (1) while not sacrificing the desirable properties of MC. This paper describes such a technique that combines MC with machine learning and statistical techniques that leads to significant computational savings over traditional methods.

We begin this paper with a brief review of integral estimation techniques including MC, fitting algorithms, and stacking. We then introduce a new technique we call stacked MC (StackMC), which uses stacking to reduce the estimation error of any MC method. Finally, we apply StackMC to a number of analytic example problems and two problems from the aerospace literature. We show that StackMC can significantly reduce estimation error, and it never has a higher estimation error than the best of MC and the chosen fitting algorithm.

II. Integral Estimation Techniques

A. Estimation Error

When using any method M returning $\hat{f}_M$ as an estimate of Eq. (1), there are two sources of estimation error: bias (b) and variance (v). Bias is the expected difference between the method and the truth: $b = \mathbb{E}[\hat{f}_M - \hat{f}]$, where the expectation is over all data sets. A method whose average over many runs gives the correct answer has zero bias, whereas a method that estimates high or low on average has nonzero bias. As an example, the (inviscid) Euler equations have significant bias in their estimate of viscous drag. Variance is a measure of how much $\hat{f}_M$ varies between different runs of M: $v = \mathbb{E}[(\hat{f}_M - \mathbb{E}[\hat{f}_M])^2]$. If M is deterministic, it has zero variance, because every run returns the same answer, whereas if M is stochastic, multiple runs have different outputs leading to a nonzero variance. The total expected squared error is the combination of these two factors:

$$\mathbb{E}[|\hat{f}_M - \hat{f}|^2] = b^2 + v \tag{2}$$

Any method seeking to reduce estimation error must keep both sources of error low.

B. MC Techniques

In "Simple Sampling" MC, a set of N samples, D_x, is generated randomly according to $p(x)$. The estimate of the expected value of the function based on D_x is the average of the function values of the samples:

$$\hat{f} \approx \hat{f}_{\text{MC}} = \frac{1}{N}\sum_{i=1}^{N} f(D_x(i))$$

where $D_x(i)$ refers to the ith generated sample. Simple MC has two extremely useful properties. First, MC is guaranteed to be unbiased, and, thus, on average, will return the correct answer. Second, MC is guaranteed to converge to the correct answer at a rate of $\mathcal{O}(n^{1/2})$, which is a rate *independent with respect to the number of dimensions*. That is, for *any* problem, increasing the number of samples by a factor of 100 decreases the expected error by a factor of 10. Because simple MC has zero bias, the estimation error is solely due to the variance of the MC algorithm. If $f(x)$ has large fluctuations, the MC estimate will have large fluctuations as well: the larger the variance of $f(x)$, the larger the variance, and, thus, the error of Monte Carlo. Similarly, the expected error of MC is smaller for functions with lower variance.

Many different sample generation techniques exist to help reduce the variance of Monte Carlo. Importance sampling [10] generates samples from an alternate distribution $q(x)$ [instead of the true distribution $p(x)$] and estimates integral (1) as a weighted average of sample values:

$$\hat{f} = \int f(x)p(x)\,\mathrm{d}x = \int \frac{f(x)p(x)}{q(x)} q(x)\,\mathrm{d}x \tag{3}$$

$$\hat{f}_{\text{is}} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(D_x(i))p(D_x(i))}{q(D_x(i))} \tag{4}$$

Importance sampling is often used when the tails of a distribution have a measurable effect on $\hat{f}$ but occur very infrequently. When using simple MC, several million samples are needed to accurately capture the effects of a once-in-a-million event, whereas importance sampling causes unlikely outcomes to occur more frequently. This reduces the total number of samples needed to accurately capture the effects of tail events, allowing fewer total samples for the same level of accuracy.

Quasi-MC (QMC) techniques reduce variance by choosing sample locations more carefully. Sample points generated from simple MC will inevitably cluster in some locations and leave other locations void of samples. QMC methods are usually deterministic and reduce variance by spreading points evenly throughout the sample space. Two such methods are the scrambled Halton sequence [11] and the Sobol sequence [12]. Although often effective, due to deterministic sample generation, they are not guaranteed to be unbiased. It is also difficult to generate points from an arbitrary $p(x)$; most QMC algorithms generate sample points from a uniform distribution over the unit hypercube.

C. Supervised Learning and Fitting

A second class of techniques for estimating integrals from data seeks to use the data samples more efficiently through the use supervised learning techniques. A specific supervised learning method, a "fitting algorithm," takes a set of data samples and creates a "fit," that is, an approximation to the functional form of $f(x)$. This fit is then integrated (analytically, numerically, or by Monte Carlo) and used as an approximation to the true integral. Examples of fitting algorithms include splines, high-order polynomials, and Fourier expansions. More complicated methods are also possible, such as piecewise linear curves [13] or piecewise quadratic polynomials, as is done in Simpson's rule. Fits incorporate the spatial distribution of sample points and, thus, often give more accurate estimates of $\hat{f}$ rather than MC. However, using a fitting algorithm can induce bias and may or may not exhibit convergence to the correct answer as the number of points increases. Additionally, when the number of data points is "too small," many fitting algorithms exhibit higher variance than MC, and, as a result, using a fitting algorithm can be worse than not using one, especially with low numbers of sample points and in high-dimensional spaces. It is usually impossible to know a priori how many points is "too small," making it difficult to know when to use a fitting algorithm.

Additionally, it is difficult to know if a fit to the data samples is an accurate representation of $f(x)$ at the values of x which are not in the data set. Many fitting algorithms exhibit a property known as "overfitting," in which a fit to the data is very accurate at x locations that are among the data samples but is very inaccurate at x locations not among the data samples (this is true of higher order polynomial fits, in which the oscillatory nature of the fit results in large inaccuracies away from the sample points). As a result, one cannot use the data samples themselves to make a fit and evaluate its accuracy. The standard technique for addressing this issue is known as cross-validation, in which a certain percentage of the data are used to make the fit, and the rest of the data are used to evaluate its performance. This process is repeated several times, each time using a different subset of the data to make the fit. The fit which performs best on the remaining data is used as the output of the fitting algorithm, and the rest of the fits are ignored (*winner takes all*).

Stacking is a technique first introduced by Wolpert [14] that improves upon cross-validation. Instead of ignoring the losing fits, as in winner takes all, stacking combines all of the fits together. Because each fit was created with a different set of data, they all contain unique information about $f(x)$. The combination of all of this information should be a better approximation to $f(x)$ than any one fit alone. There are many different ways to combine the fits together, but a simple way is to give each fit a weight proportional to how well it predicted the data on which it was not trained. Interested readers can see stacking applied to improving regression [15], probability density estimation [16], confidence interval estimation [17], and optimization [18]. For a comparison of stacking to other methods, see [19].

III. Stacked Monte Carlo

The main idea of StackMC is to incorporate the variance reduction potential of a fitting algorithm while avoiding the introduction of bias and overconfidence in poor fits to the data. It makes no assumptions about the function $f(x)$ nor the sample generation method; therefore, StackMC can be used to augment nearly any MC application.

Let us assume that we have a function $g(x)$ that is a reasonable (though not necessarily perfect) fit to $f(x)$. Equation (1) can be rewritten as

$$\begin{aligned}\hat{f} &= \int \alpha g(x)p(x)\,\mathrm{d}x + \int (f(x) - \alpha g(x))p(x)\,\mathrm{d}x \\ &= \alpha\hat{g} + \int (f(x) - \alpha g(x))p(x)\,\mathrm{d}x \end{aligned} \tag{5}$$

where α is a constant, and $\hat{g} = \int g(x)p(x)\,\mathrm{d}x$. Instead of using MC samples to estimate Eq. (1) directly, we can use the MC samples to estimate the integral term in Eq. (5), that is

$$\hat{f} \approx \alpha\hat{g} + \frac{1}{N}\sum_{i=1}^{N} f(D_x(i)) - \alpha g(D_x(i)) \tag{6}$$

Because $g(x)$ is assumed to be a reasonable fit, for a properly chosen α, $f(x) - \alpha g(x)$ has lower variance than $f(x)$ alone (see Fig. 1), and so an MC estimate of Eq. (5) has a lower expected error than an estimate of Eq. (1). In fact, it can be shown [20] that the optimal value of α to minimize expected error is

$$\alpha = \rho\frac{\sigma_f}{\sigma_g} \tag{7}$$

where σ_f and σ_g are the standard deviations of $f(x)$ and $g(x)$, respectively, and ρ is the correlation between f and g. Intuitively, if g is a good fit to f, ρ (and correspondingly α) will be high, and $\hat{g}$ will be trusted as a good estimate for $\hat{f}$. If g is a poor fit to f, ρ and α will both be low, and $\hat{g}$ will be mostly ignored. From a different perspective, Eq. (5) takes the expected value estimated from $g(x)$ and corrects it with a term estimating the bias of $g(x)$. Either way, by using Eq. (5) the error should be lower than using either MC or the fitting function alone. Because the expected value of the fit is both added and subtracted, Eq. (6) remains an unbiased estimate of Eq. (1). Thus Eq. (6) incorporates information from a fit while remaining unbiased, and α allows us to emphasize good fits while deemphasizing poor ones.

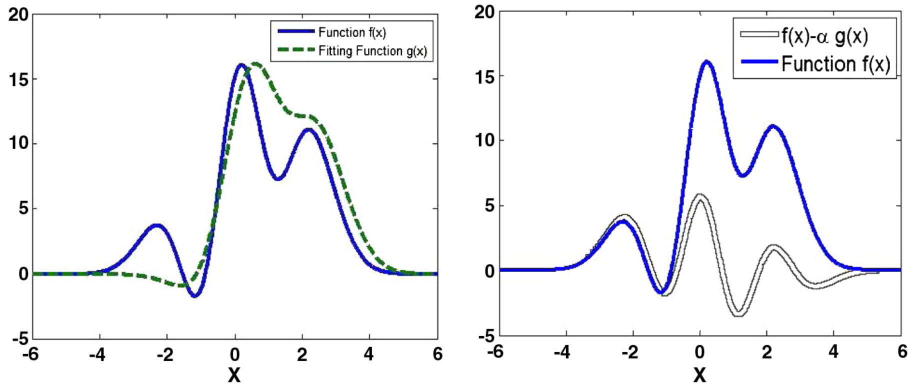
The obvious question is how to obtain $g(x)$ and find α from data samples. The first step is to pick a functional form for $g(x)$ which can be nearly anything, as long as it can make predictions at new x values. For example, $g(x)$ could be a polynomial with unknown coefficients. For now, we also require that g is analytically integrable over the volume, that is, we can calculate

$$\hat{g} = \int g(x)p(x)\,\mathrm{d}x \tag{8}$$

analytically (see further discussion later in the paper). By comparing the output of a fit $g(x)$ to the true $f(x)$ values, an estimate for the "goodness" of the fit (and by extension α) could be obtained, and Eq. (6) could be applied. However, doing this directly would cause overfitting and would lead to an inaccurate estimate of α.

Overfitting can be mitigated by using a technique known as k-fold cross-validation [21]. The N data samples are randomly partitioned into k testing sets, $D_x^{i,\mathrm{test}}$, $i = 1, \ldots, k$, which are mutually exclusive and contain the same number of samples (differing slightly if N/k is not an integer). Training sets, $D_x^{i,\mathrm{train}}$, $i = 1, \ldots, k$, contain all of the data samples not in $D_x^{i,\mathrm{test}}$. We call N_i the number of samples in $D_x^{i,\mathrm{test}}$ and m_i the number of samples in $D_x^{i,\mathrm{train}}$ (so that $N_i + m_i = N$). One fit per fold, $g_i(x)$, is created using only the N_i samples in $D_x^{i,\mathrm{train}}$ for a total of k fitters [k different $g(x)$ functions]. The samples in a training set are "held-in" points, because they are used to generate the fit, whereas points in the corresponding testing set are "held-out" points, because the fit is generated without using these samples. Each data sample is in a held-out data set one time and in the held-in data set $k - 1$ times. Using $g_i(x)$, a prediction of the function values for the points in $D_x^{i,\mathrm{test}}$ is made [$g_i(D_x^{i,\mathrm{train}})$]. Standard cross-validation evaluates the accuracy of each of the fits and chooses the best fit to use as a single $g(x)$.

Instead of ignoring all but the best fit, we adopt the approach of stacking and use Eq. (5) to get an estimate of $\hat{f}$ from each of the fitters. We use the held-out data points as an estimate of the integral term:



a) Example $f(x)$ and $g(x)$ b) $f(x)$ and $f(x) - \alpha g(x)$

Fig. 1 Comparison of $f(x)$ and $f(x) - \alpha g(x)$ for a value of $\alpha = 0.85$. The variance of $f - \alpha g$ is lower than that of f alone, so that a MC estimate of $f - \alpha g$ will have less error than an estimate of f.

$$\hat{f}_{\text{StackMC}}(i) = \alpha \hat{g}_i + \int (f(x) - \alpha g_i(x)) p(x)\,\mathrm{d}x \tag{9}$$

$$\hat{f}_{\text{StackMC}}(i) = \alpha \hat{g}_i + \frac{1}{m_i}\sum_{j=1}^{m_i} f(D_x^{i,\text{test}}(j)) - \alpha g_i(D_x^{i,\text{test}}(j)) \tag{10}$$

and finally the mean of these estimates is taken as the final prediction:

$$\hat{f} \approx \hat{f}_{\text{StackMC}} = \frac{1}{k}\sum_{i=1}^{k} \hat{f}_{\text{StackMC}}(i) \tag{11}$$

We can also use the predictions at the held-out points to estimate α; σ_f is estimated from the variance of the true sample function values, σ_g from the variance of the predictions from the $g_i(x)$, and ρ from the correlation between the predictions and the truth:

$$\mu_f = \frac{1}{N}\sum_{j=1}^{N} f(D_x(j)) \tag{12}$$

$$\mu_g = \frac{1}{N}\sum_{i=1}^{k}\sum_{j=1}^{N_i} g_i(D_x^{i,\text{train}}(j)) \tag{13}$$

$$\sigma_f = \sqrt{\frac{1}{N-1}\sum_{j=1}^{N}(f(D_x(j)) - \mu_f)^2} \tag{14}$$

$$\sigma_g = \sqrt{\frac{1}{N-1}\sum_{i=1}^{k}\sum_{j=1}^{N_i}(g_i(D_x^{i,\text{train}}(j)) - \mu_g)^2} \tag{15}$$

$$\operatorname{cov}(f,g) = \frac{1}{N-1}\sum_{i=1}^{k}\sum_{j=1}^{N_i}(f(D_x^{i,\text{train}}(j)) - \mu_f)(g(D_x^{i,\text{train}}(j)) - \mu_g) \tag{16}$$

$$\rho = \frac{\operatorname{cov}(f,g)}{\sigma_f \sigma_g} \tag{17}$$

Finally, we plug into Eq. (11) to obtain the StackMC estimate:

$$\begin{aligned}\hat{f}_{\text{StackMC}} &= \frac{1}{k}\sum_{i=1}^{k}\hat{f}_{\text{StackMC}}(i)\\ &= \frac{1}{k}\sum_{i=1}^{k}\left[\alpha\hat{g}_i + \frac{1}{m_i}\sum_{j=1}^{m_i} f(D_x^{i,\text{test}}(j)) - \alpha g_i(D_x^{i,\text{test}}(j))\right]\end{aligned} \tag{18}$$

One final correction is needed to complete StackMC. It is possible that some or all of the $\hat{g}_i$ differ greatly from $\hat{f}$, but the calculated value of α is high because of good predictions at the held-out data points. If left alone, this would cause StackMC to return a low-quality prediction on some data sets.

However, the error in the mean (EIM) of the MC samples can be used as a second metric to evaluate the goodness of the fit. EIM is defined as

$$\bar{\sigma} = \frac{\sigma}{\sqrt{N}} \tag{19}$$

and represents the uncertainty in the mean of a set of MC samples. Specifically, it gives us a likelihood bound on $\hat{f}$ and based on $\hat{f}_{\text{MC}}$ and σ_f. We can find a "likelihood" for each fold by calculating

$$\tilde{L}_i = \frac{|\hat{g}_i - \hat{f}_{\text{MC}}|}{\bar{\sigma}} \tag{20}$$

The higher L_i, the less likely it is that $\hat{g}_i = \hat{f}$, and the more likely it is that the fitter is bad and should be ignored. A heuristic is set if that $\max(L_i) > C$, $\hat{f}_{\text{StackMC}} = \hat{f}_{\text{MC}}$, that is, all of the fits are ignored entirely, and the MC average is used. If C is set too low, then too many reasonable fits are ignored, and if C is set too high, then too many unreasonable fits are kept. A value of $C = 5$ was chosen based on experimentation; it was clear that setting C as low as 3 or as high as 7 were inferior.

Finally, here is a note about the bias of StackMC. Because the expected value of the fit is being both added and subtracted, it is true that Eq. (5) [and by extension Eqs. (9) and (11)] is also unbiased for a fixed α. However, using the held-out data to both set α and estimating the integral can introduce bias. In practice, we have found that this is only a problem for very small numbers of sample points, in which the result of the fitting algorithm changes significantly depending on the held-in samples.

A. Generalized Stacked Monte Carlo

The discussion above was for the case in which the samples are generated according to a known $p(x)$, but this is only a special case of a class of estimation scenarios. Although the overall methodology remains approximately the same for these other scenarios, some specifics (such as the calculation of α) change with the choice for $g(x)$ and the sample generation method.

1. *Importance Sampling*

If importance sampling methods are used, samples are generated from $q(x)$ instead of $p(x)$. As a result, Eq. (1) is expanded as

$$\hat{f} = \int \alpha g(x) q(x)\,\mathrm{d}x + \int\left(\frac{f(x)p(x)}{q(x)} - \alpha g(x)\right) q(x)\,\mathrm{d}x \tag{21}$$

and so $g(x)$ should be a fitting algorithm for $f(x)p(x)/q(x)$ instead of just $f(x)$. As a result, a few modifications are needed, including the fact that

$$\hat{g}_i = \int g(x) q(x)\,\mathrm{d}x \tag{22}$$

and that in the calculation of α and Eq. (18), fp/q is used instead of just f.

2. *Inability of Integrating $g(x)$ over $p(x)$*

If the samples are generated from $p(x)$, but the choice for $g(x)$ cannot be integrated over $p(x)$, there are two options. The first option is to use another function $r(x)$ as a fitting algorithm for $p(x)$ over which $g(x)$ is integrable. We expand Eq. (1) as

$$\begin{aligned}\hat{f} &= \int \alpha g(x) r(x)\,\mathrm{d}x + \int f(x)p(x) - \alpha g(x) r(x)\,\mathrm{d}x\\ &= \int \alpha g(x) r(x)\,\mathrm{d}x + \int\left(f(x) - \alpha\frac{g(x)r(x)}{p(x)}\right) p(x)\,\mathrm{d}x\end{aligned} \tag{23}$$

Similar to the above case, when calculating α and Eq. (18), gr/p replaces g.

Alternately, when $g(x)$ is significantly less computationally expensive than $f(x)$, $\hat{g}$ itself can be estimated by MC techniques. When estimating $\hat{g}$ from the N_g additional samples, it should be the case that $N_g \gg N$, so that errors in the estimate of $\hat{g}$ do not cause significant errors in the estimate of $\hat{f}$.

B. Simple Example

We attempt to find the expected value of $3x^6 + 3.6x^5 - 91.29x^4 - 19.41x^3 + 57.15x^2 - 14.43x + 0.9$, where x is generated according to a uniform distribution between −3 and 3. Thus, the integral of concern is

$$\hat{f} = \frac{1}{2}\int_{-3}^{3}(x^6 + 1.2x^5 - 30.43x^4 - 6.47x^3 + 19.05x^2 - 4.81x + 0.3)\,\mathrm{d}x \tag{24}$$

The actual value of $\hat{f}$ can be calculated to be 0.7069. There are 20 samples generated and divided into five folds, with each fold's testing set containing four points, and each corresponding training set containing the other 16 points. Although the function of interest is a sixth-order polynomial, $g(x)$ was chosen to be a third-order polynomial: $\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$. The fit to the first fold, $g_1(x)$ is found by choosing the βs which minimize the squared error of the data in $D_x^{1,\text{train}}$ (using the standard pseudoinverse). The fits to the rest of the folds $g_2(x)$–$g_5(x)$ are set similarly. Next, $g_i(D_x^{i,\text{test}}(j))$ is evaluated for each test point in $D_x^{i,\text{test}}$.

The following parameters are calculated to find α:

$$\mu_f = 1.1337 \quad \mu_g = 1.9258 \quad \sigma_f = 5.6835 \quad \sigma_g = 5.2678$$
$$\mathrm{cov}(f,g) = 24.0999 \quad \rho = \frac{\mathrm{cov}(f,g)}{\sigma_f\sigma_g} = 0.8049$$
$$\alpha = \rho\frac{\sigma_f}{\sigma_g} = 0.8685 \tag{25}$$

Finally, Eq. (18) is applied to calculate $\hat{f}_{\text{StackMC}}$.

Using the MC estimate alone gives $\hat{f}_{\text{mc}} = 1.1337$, and using a fit to all of the data points alone gives $\hat{f}_{\text{fit}} = 1.3412$. By using StackMC, we find $\hat{f}_{\text{StackMC}} = 0.8081$, which is much closer to the true value than either of the individual estimates. Details of the calculations appear in Tables 1 and 2, and the fit from the first fold appears in Fig. 2.

IV. Results

We test the performance of StackMC on a number of different problems. In the first set of example cases, the problems have known analytic results, and an exact analysis of the performance of StackMC is examined. In the second set, StackMC is applied to two aerospace problems in the literature. Although an exact solution to the aerospace problems is not available, an approximate answer is obtained from a MC estimate using 100,000 samples. For each example problem, a range of dataset sizes was tested, and for each dataset size, 2000 simulations were run using ten-fold cross validation. Unless otherwise noted, the plots in this section have three lines, which show the mean squared error versus the number of sample points. The lines represent the average squared error from using just the prediction of MC (green), the fit to all the samples (red), and StackMC (blue).

Table 2 Details of fold combination calculations

Fold	$\hat{g}_i$	$\sum_{j=1}^{m_i} f(D_x^{i,\text{test}}(j)) - \alpha g_i(D_x^{i,\text{test}}(j))$	Corrected $\hat{g}_i$
1	1.4281	−0.3553	0.8795
2	1.4622	−0.6428	0.6271
3	1.9032	−0.6122	1.0407
4	1.7141	−1.3569	0.1318
5	1.2529	−0.2732	1.3613
		$\hat{f}_{\text{StackMC}}$	0.8081

A. Analytic Test Cases

1. 10-Dimensional Rosenbrock Function Under a Uniform $p(x)$, Polynomial Fitter

The D-dimensional Rosenbrock function is given by

$$f(x) = \sum_{i=1}^{D-1}[(1-x_i)^2 + 100(x_{i+1} - x_i^2)^2] \tag{26}$$

and x is drawn from a uniform distribution over the $[-3, 3]$ hypercube.

The fitting algorithm is chosen as a third-order polynomial in each dimension whose form is

$$g(x) = \beta_0 + \sum_{i=1}^{D}\beta_{1,i}x_i + \beta_{2,i}x_i^2 + \beta_{3,i}x_i^3 \tag{27}$$

where β_i are free parameters that are set from the data samples.

A comparison of the error of StackMC can be seen in Fig. 3 for the 10-dimensional version of the Rosenbrock function. At low numbers of sample points, MC is more accurate, on average, than the polynomial fit to all the data points, but the polynomial outperforms MC at higher numbers of points. Throughout the entire range of number of samples, StackMC performs at least as well as the best of the two. Additionally, as displayed in Fig. 4, the polynomial fitter is actually a biased fitter for this example problem. StackMC is able to use cross-validation to remove the bias of the fitter while keeping the accuracy of its estimate.

Table 1 Details of simple example calculations

Leftout fold	x	$f(x)$	β_0	β_1	β_2	β_3	$g_i(x)$	$\hat{g}_i$
1	0.4087	0.2438	2.7385	−4.3737	−3.9500	−9.4712	−0.3549	1.4281
	−0.6950	7.8350					7.0498	
	−0.0943	0.9259					3.1237	
	0.1152	−0.0166					2.1675	
2	0.4117	0.2420	2.4683	−3.3829	−3.0183	−11.3595	−0.2284	1.4622
	0.2745	0.1108					1.0774	
	0.1823	−0.0163					1.6825	
	0.2882	0.1342					0.9708	
3	−0.6318	7.6689	0.7900	0.3755	3.3397	−22.0257	7.4398	1.9032
	−0.3923	4.7811					2.4864	
	−0.8345	6.4358					15.6002	
	0.7716	−5.2874					−7.0489	
4	0.5711	−0.5683	1.8054	−6.7386	−0.2738	−1.3468	−2.3831	1.7141
	−0.5988	7.4412					6.0312	
	0.9607	−16.6302					−6.1154	
	−0.6411	7.7172					6.3677	
5	0.7124	−3.2834	2.3965	−3.8653	−3.4306	−10.9995	−6.0740	1.2529
	0.1206	−0.0208					1.8609	
	−0.3960	4.8377					4.0722	
	0.2816	0.1230					0.7901	

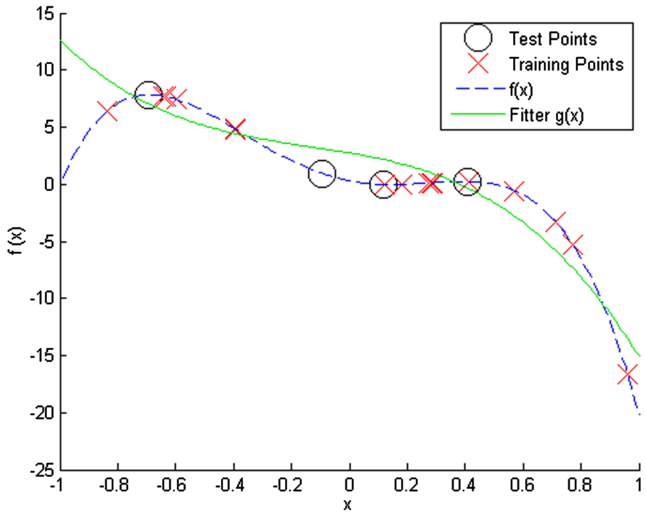


Fig. 2 Example of the first fold.

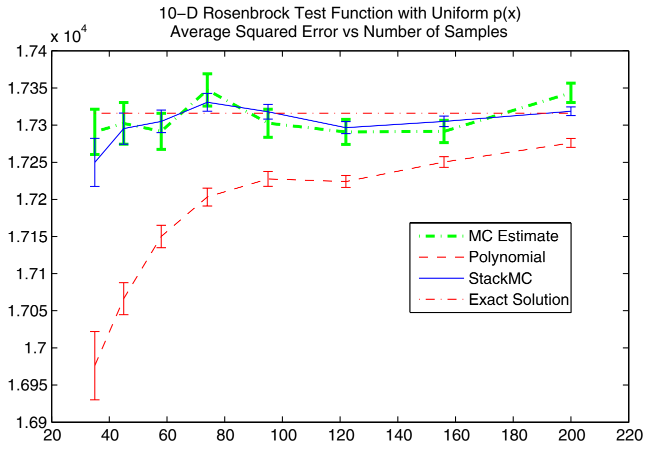


Fig. 4 Expected output of MC, fitter, and StackMC with EIM for the 10-dimensional Rosenbrock function. Note that the fitting function is biased whereas StackMC and MC are not.

2. 10-Dimensional Rosenbrock Function Under a Gaussian $p(x)$

This is the same setup as in the preceding section, except that x is generated according to a multivariate Gaussian distribution, in which each dimension is independent with $\mu = 0$ and $\sigma = 2$.

Similar to the preceding case, Fig. 5 shows that at low numbers of sample points MC outperforms the polynomial fit, whereas at high sample points the polynomial does better than MC alone. However, for all numbers of sample points, StackMC outperforms the other two algorithms.

3. 20-Dimensional BTButterfly, Uniform $p(x)$, Fourier Fitter

In the preceding examples, the Rosenbrock function is a fourth-order polynomial, and the fitting algorithm is a third-order polynomial, meaning StackMC had reasonable fits to use to improve upon Monte Carlo. To challenge StackMC, a function we call the BTButterfly was created so that it would be very difficult to fit accurately.

Like the Rosenbrock function, its general form is given by

$$f(x) = \sum_{i=1}^{D-1} h(x_i, x_{i+1}) \tag{28}$$

with the contour plot of h shown in Fig. 6. The uncertainty in x is uniform over the $[-3, 3]$ hypercube. The regions that appear as boxes are introduced discontinuities.

A Fourier expansion for $g(x)$ was chosen, whose form is given by

$$g(x) = \beta_0 + \sum_{i=1}^{D} \beta_{1,i} \cos(x_i) + \beta_{2,i} \cos(2x_i) + \beta_{3,i} \cos(3x_i) + \beta_{4,i} \sin(x)_i + \beta_{5,i} \sin(2x_i) + \beta_{3,i} \sin(3x_i) \tag{29}$$

Results for this function are shown in Fig. 7 and exhibit the same trends discussed in the preceding text; StackMC has as low of an error as the lowest of each method.

B. Aerospace Applications

1. Future Aircraft Uncertainty Quantification

In [22], the authors use the Program for Aircraft Synthesis Studies [23], a conceptual aircraft design tool, to predict the fuel burn of future aircraft given certain assumptions about technology advancement in the 2020 and 2030 time frames. In their predictions for single-aisle aircraft in 2020, the authors model eight probabilistic variables representing different effects of improvements in aircraft technology (propulsion, structures, and aerodynamics). Each variable is represented by a unique beta distribution. The authors generated 15,000 samples (each representing one optimized aircraft) from which they measured the expected fuel-burn metric and the standard deviation of the expected fuel-burn metric.

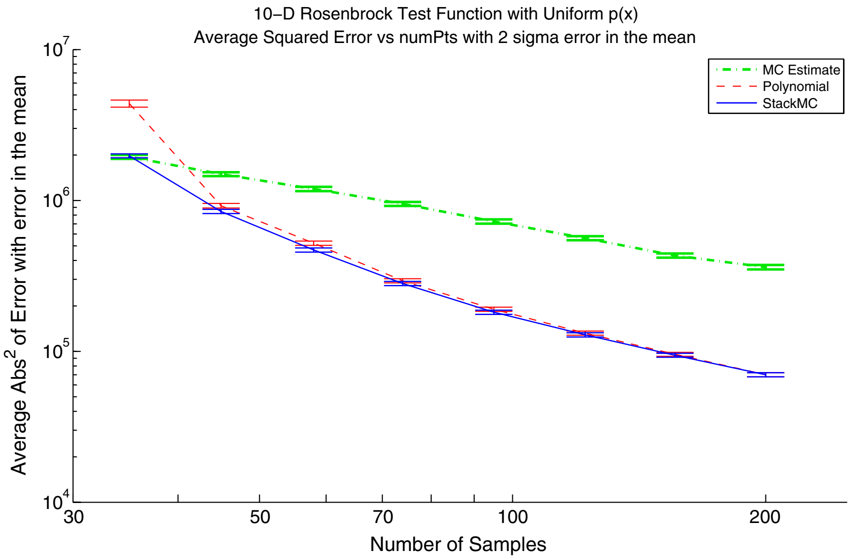


Fig. 3 Comparison of MC, fitter, and StackMC for the Rosenbrock function with uniform uncertainty.

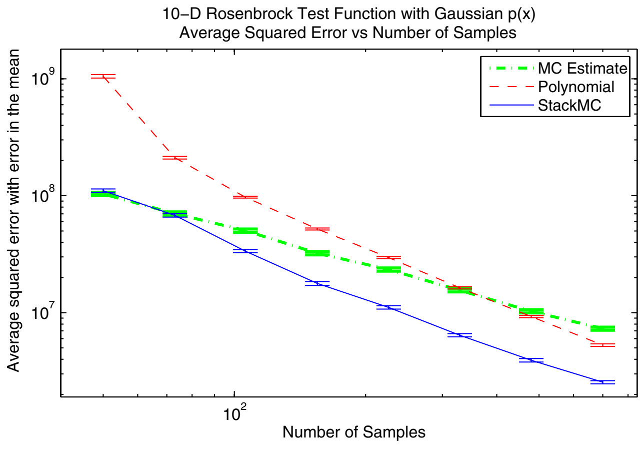


Fig. 5 Comparison of MC, fitter, and StackMC for the 10-dimensional Rosenbrock function with Gaussian uncertainty.

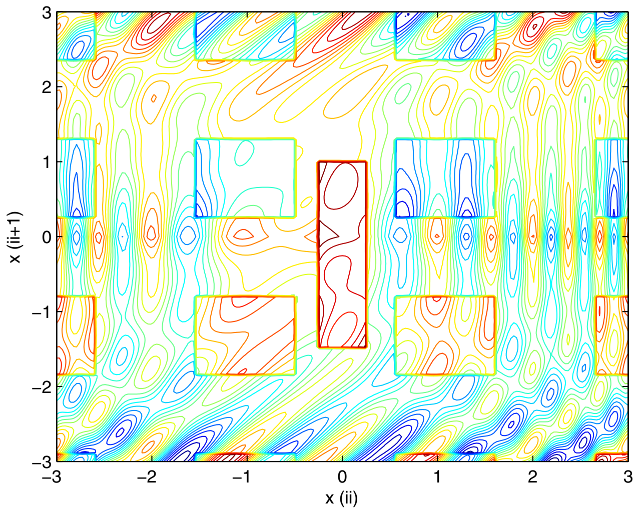


Fig. 6 Contour plot of successive dimensions of the BTButterfly function.

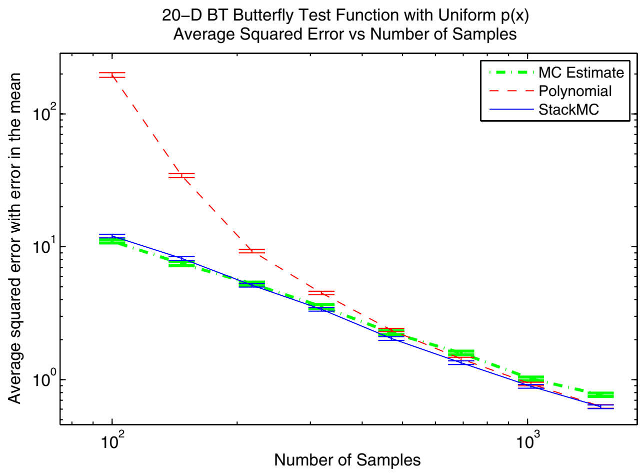


Fig. 7 Comparison of MC, fitter, and StackMC for the BTButterfly function with uniform uncertainty.

To apply StackMC, we choose a third-order polynomial as the fitting algorithm. A polynomial fit is convenient for this case, because the $\mathbb{E}[x^n]$ over a beta distribution is easily found analytically as follows:

$$\mathbb{E}[x^n] = \frac{\prod_{i=1}^{n} \alpha + i - 1}{\prod_{i=1}^{n} \alpha + \beta + i - 1} \tag{30}$$

where α and β are the two parameters of the beta distribution, $B(\alpha, \beta)$. In Eq. (30), α is not to be confused with the StackMC parameter α.

A set of 100,000 samples were generated via MC sampling, and the mean and standard deviation of their function values were taken as the “true” expected value and standard deviation. The standard deviation is defined as $\sqrt{\mathbb{E}[x^2] - \mathbb{E}[x]^2}$; therefore, to find the standard deviation, we can run StackMC twice: one time to find $\mathbb{E}[x]$ and a second time to find $\mathbb{E}[x^2]$. The results of applying StackMC to find the expected value and standard deviation are shown in Figs. 8 and 9, respectively.

The exact same trend is seen here as in all of the analytic problems. At low numbers of samples, in which the fit to all the data points performs poorly, StackMC does as well as Monte Carlo. At high numbers of samples, in which the fit greatly outperforms Monte Carlo, StackMC does as well as the fitter. In between these two extremes, StackMC outperforms both algorithms.

In this example problem, each sample takes about 6 s to generate, and so it takes 150 min to generate 1500 samples. Figure 8 shows that by using StackMC, the same level of accuracy is achieved with only 150 samples (taking only 15 min to generate). StackMC takes less than 0.5 s to form the 10 fits, calculate α, and calculate Eq. (18), which is negligible compared to the cost of forming the data set.

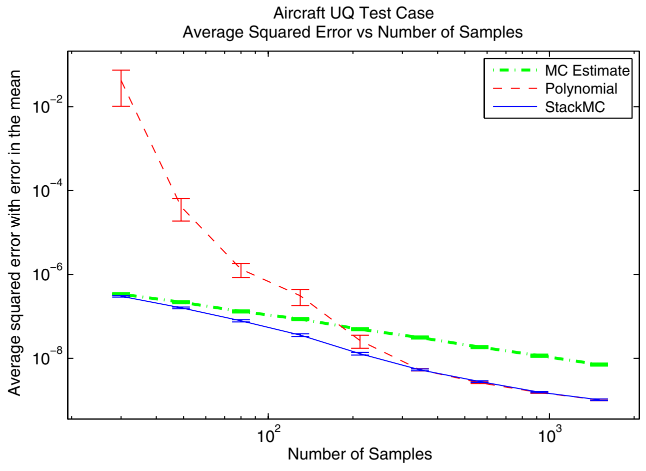


Fig. 8 Comparison of MC, fitter, and StackMC for the aircraft UQ test case finding $\mathbb{E}[x]$.

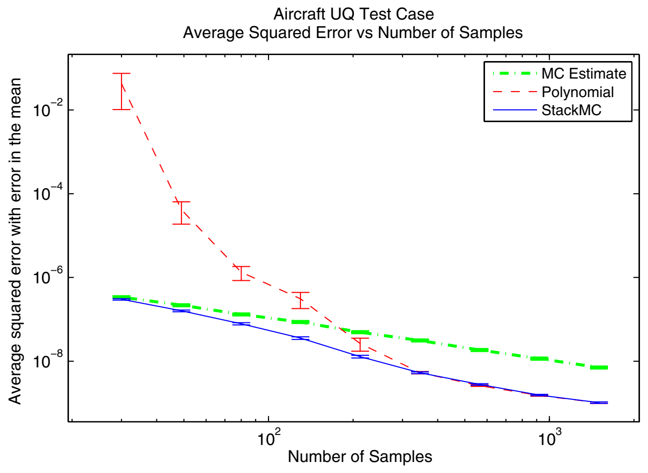


Fig. 9 Comparison of MC, fitter, and StackMC for the aircraft UQ test case finding $\mathbb{E}[x^2]$.

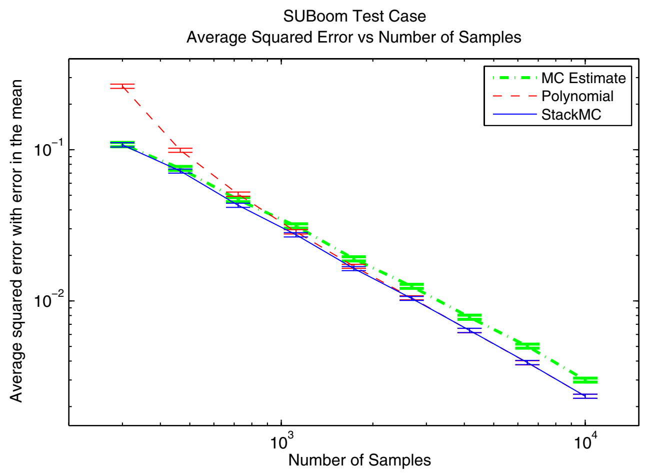


Fig. 10 Comparison of MC, fitter, and StackMC for the SUBoom UQ test case.

2. Sonic Boom Uncertainty Quantification

The uncertainty quantification (UQ) of sonic boom noise provides the second application case. Unlike the aircraft design test case, the response for the sonic boom noise signature is not smooth; the output can vary significantly with slight adjustments to the input parameters. Colonno and Alonso [24] recently created a new sonic boom propagation tool, SUBoom, and used it to analyze the robustness of several aircraft pressure signatures optimized for minimal boom noise. Specifically, in their high-fidelity near-field case, they have 62 uncertain variables: Four variables represent aircraft parameters, such as cruise Mach number and roll angle, and 58 represented uncertainties in the near-field pressure signal. Similar to the aircraft UQ case, 100,000 MC samples were generated, and their mean was used as the true mean. A third-order polynomial was used as $g(x)$. The results can be seen in Fig. 10.

Even with the discontinuities in the function space, the same performance for StackMC is seen. StackMC does as well as the best of MC and the fitting algorithm. The reduction in error is not as great here as in the aircraft UQ case, because a polynomial is not a good fit for $f(x)$, although using the domain knowledge to improve the accuracy of the fitting algorithm would further increase the performance of StackMC. Despite the relatively poor performance of the fitting algorithm, it is not worse to use StackMC regardless of the number of sample points used. In some cases, this test case demonstrates one of the strengths of StackMC: with virtually no additional computation cost, the user gets the accuracy of the most accurate method and doesn't need to concern himself with the relative accuracies of the fit and Monte Carlo.

V. Limitations and Future Work

The results shown in the preceding section are promising, but even larger gains could be achieved as the StackMC algorithm implemented in this paper is relatively simple. At the moment, $\hat{f}_{\mathrm{StackMC}}$ is computed as the average of the expected values from each of the fits. Instead, one could take a weighted average of the fits, in which more accurate fits have higher weight. StackMC currently only partitions the data into folds one time, but we could imagine repartitioning the same samples several times to have more fits to combine. We use MC to estimate the integral term in Eq. (9), but using a fitter, or even using StackMC recursively, could improve the estimates of $\hat{f}_{\mathrm{smc}}$. Furthermore, α was set as a constant, but in general α could vary between the folds or could even be a function of x, so that the confidence in the fit varies spatially.

In a future paper, we will present the results of the application of StackMC to different input regimes and different sample generation methods. We will examine higher dimensional spaces, explore the application of StackMC to multifidelity methods, and extend StackMC to incorporate multiple fitting functions.

VI. Conclusions

In this paper, a new technique has been introduced, stacked Monte Carlo (StackMC), which reduces the error of Monte Carlo (MC) sampling by blending several different fitters of $f(x)$. StackMC is unbiased, thus retaining a major advantage of MC and uses cross-validation to determine the accuracy of the fitting function.

StackMC is an extremely generic postprocessing technique. It is applied after the samples are generated, and it makes no assumptions about the generation method. Therefore, StackMC can be used not only with simple MC, but also with other sample generation techniques, such as importance sampling, quasi-MC, and Markov–Chain MC. Furthermore, StackMC makes no assumptions about $f(x)$; it not only applies to smooth functions but also to discontinuous functions and even functions with discrete variables.

The computation time of StackMC is dominated by forming the fits $g_i(x)$. As a result, StackMC is only affected by the curse of dimensionality to the extent of the fitting algorithm. In both of the aerospace applications, the additional cost of the entire StackMC algorithm was negligible; there are significant gains in accuracy for less computation time than the cost of one additional function evaluation. The only assumptions made about $g(x)$ are that it can predict the value at new sample locations, and that $\hat{g}$ can be determined accurately, either analytically or through some other method. As shown in the BTButterfly and sonic boom example cases, the fit does not need to be particularly good to see improvement, so the choice for fitting algorithm can be modified as computational effort and domain knowledge allows. StackMC does not eliminate the need for finding better fitting methods; a better fitter will always lead to an improved result. With the exception of C for the EIM test (whose value was not changed for any of the example cases), StackMC requires no user-set parameters that need to be heuristically tuned.

Despite a simplistic implementation, StackMC performs at least as well as the better of MC or the fitting function, and it outperforms both for a range of samples. Normally, there is a transition number of samples at which using a fit to all of the data samples has a smaller average error than MC, but it is hard to know a priori if it is better to use the fit. StackMC eliminates the need to decide whether or not to use a fit; it will never be harmful to do so. Although it is not true that for any set of data samples $\hat{f}_{\mathrm{StackMC}}$ is closer to $\hat{f}$ than $\hat{f}_{\mathrm{MC}}$, on average, StackMC reduces the expected error. StackMC is a very generic method for the postprocessing of data samples; it can be used by anyone trying to estimate an integral or the expected value of a function where $p(x)$ is known.

References

[1] Mahadevan, S., and Liu, X., "Probabilistic Analysis of Composite Structure Ultimate Strength," *AIAA Journal*, Vol. 40, No. 7, 2002, pp. 1408–1414.
doi:10.2514/2.1802

[2] Mavris, D. N., Bandte, O., and DeLaurentis, D. A., "Robust Design Simulation: A Probabilistic Approach to Multidisciplinary Design," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 298–307.
doi:10.2514/2.2437

[3] Verhoosel, C., "Uncertainty and Reliability Analysis of Fluid-Structure Stability Boundaries," *AIAA Journal*, Vol. 47, No. 1, 1968, pp. 91–104.
doi:10.2514/1.35770

[4] Wright, M. J., Bose, D., and Chen, Y.-K., "Probabilistic Modeling of Aerothermal and Thermal Protection Material Response Uncertainties," *AIAA Journal*, Vol. 45, No. 2, 2007, pp. 399–410.
doi:10.2514/1.26018

[5] Prabhakar, A., Fisher, J., and Bhattacharya, R., "Polynomial Chaos-Based Analysis of Probabilistic Uncertainty in Hypersonic Flight Dynamics," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 1, 2010, pp. 222–324.
doi:10.2514/1.41551

[6] Xiu, D., and Karniadakis, G. E., "The Wiener-Askey Polynomial Chaos for Stochastic Differential Equations," *SIAM Journal of Scientific Computing*, Vol. 24, No. 2, 2002, pp. 619–644.
doi:10.1137/S1064827501387826

[7] Xiu, D., *Numerical Methods for Stocahstic Computations: A Spectral Method Approach*, Princeton Univ. Press, Princeton, NJ, 2010, pp. 1–8, Chap. 1.

[8] Eldred, M. S., and Constantine, P., “Evaluation of Non-Intrusive Approaches for Wiener-Askey Generalized Polynomial Chaos,” *49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, AIAA Paper 2008-1892, Schaumburg, IL, 2008.

[9] Witteveen, J., Loeven, A., and Bijl, H., “An Adaptive Stochastic Finite Elements Approach Based on Newton–Cotes Quadrature in Simplex Elements,” *Computers and Fluids*, Vol. 38, No. 6, 2009, pp. 1270–1288. doi:10.1016/j.compfluid.2008.12.002

[10] Haugh, M., “Variance Reduction Methods II,” *Monte Carlo Simulation: IEOR E4703*, Columbia Univ., 2004, pp. 1–20.

[11] Kocis, L., and Whiten, W. J., “Computational Investigations of Low-Discrepancy Sequences,” *ACM Transactions on Mathematical Software*, Vol. 23, No. 2, 1997, pp. 266–294. doi:10.1145/264029.264064

[12] Sobol, I., “Uniformly Distributed Sequences with Additional Uniformity Properties,” *USSR Computational Mathematics and Mathematical Physics*, Vol. 16, No. 5, 1976, pp. 236–242. doi:10.1016/0041-5553(76)90154-3

[13] Africano, R., “A Modified Monte Carlo Procedure,” *AIAA Journal*, Vol. 6, No. 6, 1968, pp. 1111–1117. doi:10.2514/3.4681

[14] Wolpert, D., “Stacked Generalizations,” *Neural Networks*, Vol. 5, No. 2, 1992, pp. 241–260. doi:10.1016/S0893-6080(05)80023-1

[15] Breiman, L., “Stacked Regressions,” *Machine Learning*, Vol. 24, Kluwer Academic Publishers, Boston, 1996, pp. 49–64.

[16] Smyth, P., and Wolpert, D., “Stacked Density Estimation,” *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10, NIPS ’97*, MIT Press, Cambridge, MA, 1998, pp. 668–674.

[17] Kim, K., and Bartlett, E. B., “Error Estimation by Series Association for Neural Network Systems,” *Neural Computation*, Vol. 7, No. 4, 1995, pp. 799–808. doi:10.1162/neco.1995.7.4.799

[18] Rajnarayan, D., and Wolpert, D., “Exploiting Parametric Learning to Improve Black-Box Optimization,” 2007.

[19] Clarke, B., “Comparing Bayes Modeling Averaging and Stacking when Model Approximation Error Cannot Be Ignored,” *Journal of Machine Learning Research*, Vol. 4, 2003, pp. 683–712.

[20] Haugh, M., “Variance Reduction Methods I,” *Monte Carlo Simulation: IEOR E4703*, Columbia Univ., 2004, pp. 1–18.

[21] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*, 2nd ed., Wiley, New York, NY, 2001, pp. 483–485.

[22] Economon, T., Copeland, S., Alonso, J., Zeinali, M., and Rutherford, D., “Design and Optimization of Future Aircraft for Assessing the Fuel Burn Trends of Commercial Aviation,” *49th AIAA Aerospace Sciences Meeting*, AIAA Paper 2011-267, Orlando, FL, Jan. 2011.

[23] Kroo, I., “An Interactive System for Aircraft Design and Optimization,” *Aerospace Design Conference*, AIAA Paper 1992-1190, Irvine, CA, 1992.

[24] Colonno, M., and Alonso, J., “Sonic Boom Minimization Revisited: The Robustness of Optimal Low-Boom Designs,” *13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, AIAA Paper 2010-9364, Fort Worth, TX, Sept. 2010.

R. Ghanem
Associate Editor