OVERVIEW OF MODERN DESIGN OF EXPERIMENTS METHODS FOR COMPUTATIONAL SIMULATIONS

Anthony A. Giunta*, Steven F. Wojtkiewicz Jr.†, and Michael S. Eldred‡
Sandia National Laboratories§
Albuquerque, NM USA

Abstract

The intent of this paper is to provide an overview of modern design of experiments (DOE) techniques that can be applied in computational engineering design studies. The term *modern* refers to DOE techniques specifically designed for use with deterministic computer simulations. In addition, this term is used to contrast *classical* DOE techniques that were developed for laboratory and field experiments that possess random error sources. Several types of modern DOE methods are described including pseudo-Monte Carlo sampling, quasi-Monte Carlo sampling, Latin hypercube sampling, orthogonal array sampling, and Hammersley sequence sampling.

Keywords: sampling, pseudo-Monte-Carlo, quasi-Monte Carlo, Latin hypercube, orthogonal array, Hammersley sequence, design of experiments.

1. Introduction

1.1 Overview

Both classical and modern design of experiments (DOE) techniques share the common goal of extracting as much information as possible from a limited set of laboratory or computer experiments. This knowledge extraction process is commonly referred to as sensitivity analysis, trend analysis, analysis of variance, or uncertainty quantification.

The fundamental difference between classical and modern DOE stems from the assumption that random error exists in a laboratory experiment, but does not exist in a computer experiment (i.e., a deterministic computer simulation). Classical DOE techniques arose from technical disciplines that assumed some randomness and non-repeatability in field experiments (e.g., agricultural yield studies, experimental chemistry). Hence, classical DOE approaches such as central composite design, Box-Behnken design, and full- and fractional-factorial design generally put sample points at the extremes of the parameter space, since these designs offer more reliable trend extraction in the presence of non-repeatability.

Modern DOE methods are distinguished from classical DOE methods in that the non-repeatability component can be omitted since deterministic computer simulations are involved. In these cases, space filling designs such as quasi-Monte Carlo sampling, orthogonal array sampling, and Latin hypercube sampling are more commonly employed in order to accurately extract trend information.

---

* Senior Member of Technical Staff, Optimization and Uncertainty Estimation Department; Senior Member AIAA
† Senior Member of Technical Staff, Structural Dynamics and Smart Systems Department; Member AIAA
‡ Principal Member of Technical Staff, Optimization and Uncertainty Estimation Department; Senior Member AIAA
§ Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Another feature that distinguishes classical DOE from modern DOE is the choice of probability distribution functions associated with design parameters. That is, classical DOE typically assumes that the possible values of a design parameter are uniformly distributed (i.e., equally likely) between a lower and upper bound. In contrast, modern DOE methods are intended to handle design parameters that have both uniform and non-uniform (e.g., Gaussian, Weibull) probability distributions.

A common attribute shared by both classical DOE and modern DOE is that the sample points are independent, and thus are amenable to concurrent evaluation. In the case of computer experiments, it is possible to exploit parallel computing, either on a multiprocessor computer or over a network, to compress the wall-clock time needed to complete the set of simulations prescribed by a DOE method.

The data produced by a DOE study can be used in other aspects of the engineering design process. For example, a common approach is to reuse the data generated by a DOE study to build surrogate functions, often referred to as response surface approximations, to assist the engineer in optimizing a particular product design.

1.2 Motivation

The motivation for this document is that there presently is no concise description of modern DOE methods. Rather, descriptions of modern DOE techniques are scattered throughout the literature in the statistics, applied mathematics, and engineering communities. As can be expected, there is only limited similarity between the notation and terms used in these publications due to discipline-specific jargon. Unfortunately, this unnecessarily complicates the understanding and application of modern DOE methods to engineering design problems.

In contrast, there are numerous well-written texts on classical DOE techniques that provide easy-to-use tables, and in some cases actual software, for generating sets of samples. Readily available classical DOE methods, coupled with a lack of appreciation of the differences between deterministic computer experiments and nondeterministic laboratory experiments, has resulted in the widespread, and in some cases, inappropriate use of classical DOE methods in computational engineering design studies.

While DOE methods can be misapplied, it is the authors' opinion that the use of any DOE approach is better than the ad hoc "trial-and-error" or "build-and-break" approaches to engineering design. Thus, both modern and classical DOE methods provide utility in performing engineering design studies. The goal of this paper is to educate the computational engineering audience about the advantages and disadvantages of modern DOE techniques, so that the engineer can choose the DOE technique, modern or classical, that best fits a particular design problem.

The remainder of this document is organized with Section 2 providing some background material on the notation and terms used in modern DOE techniques. Section 3 covers pseudo-Monte Carlo sampling methods and related variants. Sections 4-6 address Latin hypercube sampling, orthogonal array sampling, and quasi-Monte Carlo sampling, respectively. Section 7 describes some of the publicly available software for performing modern DOE. Finally, Section 8 provides a summary.

2. Background and Notation

2.1 Definitions and Terms

Prior to discussing classical and modern DOE techniques, it is useful to present some standard definitions and terms.

Design Variables: the parameters or quantities to be varied during the experiment. A synonym in the statistical literature is "factors." In this text, a design variable is represented as an element in an n-dimensional vector, x_i, where $i=1,\ldots,n$. The entire vector of design variables is represented in bold font as $\boldsymbol{x}$.

Design Space: The n-dimensional space defined by the lower and upper bounds of each

design variable. Typically, the design space bounds are scaled to range from –1 to +1 or from 0 to +1. This scaling is a convenience for representing tables of samples, as well as a mathematical necessity for avoiding ill conditioned matrices in some of the linear algebra used in generating DOE samples. An *n*-dimensional design space is indicated in this text using the closed interval notation $[-1,1]^n$ or $[0,1]^n$, as appropriate for the particular DOE method. Both $[-1,1]^n$ and $[0,1]^n$ define *n*-dimensional hypercube design spaces. Note that, in this document, the *n* design variables are considered to be real-valued (i.e., $x \in \Re^n$). Some modern DOE methods are applicable to cases where the x-values are integers or discrete real values, however, this aspect of modern DOE is not addressed here.

Sample: A specific instance of $\boldsymbol{x}$, where all values in the vector $\boldsymbol{x}$ fall within the bounds of the design space. The terms "design point" and "point" are synonymous with "sample." A sample is represented as either a vector of length *n*, or as an ordered *n*-tuple of the form $(x_1, x_2, \ldots, x_n)$.

Design of Experiments: A procedure for choosing a set of samples in the design space, with the general goal of maximizing the amount of information gained from a limited number of samples. The phrase "design and analysis of computer experiments" (DACE) is used in some sources as a synonym for *modern* DOE methods.

Response: A dependent quantity that is measured or evaluated for a specific design point. Mathematically, this is represented as $y(\boldsymbol{x})$. A vector of responses is represented as either $\boldsymbol{y}(\boldsymbol{x})$ or $\boldsymbol{Y}$.

Response Surface: Any function that represents the trends of a response over the range of the design variables. In some engineering fields, the term "response surface" denotes the use of a low-order polynomial function. However, this is not consistent with the statistical community in which "response surface" is the true, unknown response trend, and "response surface approximation" denotes a user-defined function that models the response trend. Synonyms for "response surface approximation" include "model," "metamodel," "surrogate model," "approximation model, and "DACE model."

Note that response surface approximations are often associated with design of experiments. In some classical DOE methods, there is a strong connection between the form of the response model and the DOE approach (e.g., polynomial models and D-optimal experimental designs[1]). However, for modern DOE methods, there is little or no connection between the form of the response model and the DOE approach. Thus, response approximation methods are not address in this paper. One study that has examined the connection between response modeling and modern DOE approaches is the work of Simpson, et al.[2]

2.2 Discussion – Classical DOE

Classical design of experiments such as central composite design, Box-Behnken design and full-factorial design have a rich history of statistical and mathematical development along with practical application in scientific and engineering studies.[1] An often-overlooked aspect of classical DOE is the assumption that a measured response quantity contains a random error term. This is described mathematically as

$$y_m(\boldsymbol{x}) = y_t(\boldsymbol{x}) + \varepsilon, \tag{1}$$

where y_m is the measured response, y_t is the true response, and ε is a random error term.

In many cases, the ε values are assumed to be independent and identically distributed (i.i.d.). For ease of explanation in this text, consider ε to be a normal (i.e., Gaussian) random variable with a mean value of zero and a variance of unity. Because of the random error term, y_m is non-repeatable even when exactly the same values of $\boldsymbol{x}$ are used in two measurements.

One of the goals of a typical design of experiments study is to estimate and predict the trends in the response data. While there are many forms for the approximation func-

tions, a generic approximation model is used here with the form

$$\hat{y}(\boldsymbol{x}) = f(\boldsymbol{x}_s, \boldsymbol{y}_m(\boldsymbol{x}_s)), \tag{2}$$

where the sample points are denoted as $\boldsymbol{x}_s$, the measured response data are represented as $\boldsymbol{y}_m(\boldsymbol{x}_s)$, f is a user-selected function, and $\hat{y}(\boldsymbol{x})$ is the approximate response value computed at an arbitrary design point, $\boldsymbol{x}$. Typical functions for f include low-order polynomials, splines, kriging, neural networks, and radial basis functions.

The key feature of Equation (1) is the random error term and the accompanying assumption that ε is always present due to sources such as measurement error, inherent fluctuations in the response quantity (e.g., turbulence), or other sources. Another critical assumption is that the experimenter has knowledge of the general trends of the true response, y_t. Based on these assumptions, the goal of most classical DOE methods is to place a fixed number of samples in the design space so as to minimize the influence of the random error term in subsequent computations, e.g., where the approximation model given in Equation (2) is computed.

In classical DOE, the goal of minimizing the effects of random error has the effects of placing the sample sites near or on the boundaries of the design space. Myers and Montgomery[1] provide a detailed description of why this occurs, but the general idea can be observed in Figures 1 and 2.

In Figure 1, there are two sample points x_1 and x_2. The measured response value for each point is indicated by the diamond symbol. The true trend is shown by the solid line, and the estimated trend, $\hat{y}(\boldsymbol{x})$, is shown by the dotted line. In this illustration, the random errors, ε_1 and ε_2, cause the estimated trend to be a poor approximation of the true trend.

Figure 2 illustrates the effect of moving sample sites x_1 and x_2 to the lower and upper bounds of x. In this case, ε_1 and ε_2, remain the same, but the resulting estimated trend is a better approximation to the true trend.

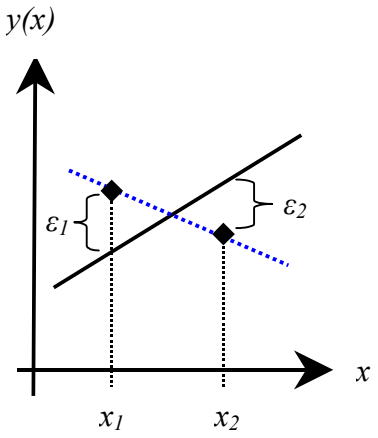

Figure 1. An illustration of the effect of random errors in producing an estimated linear model (dashed) that has a different slope than the true linear model (solid).

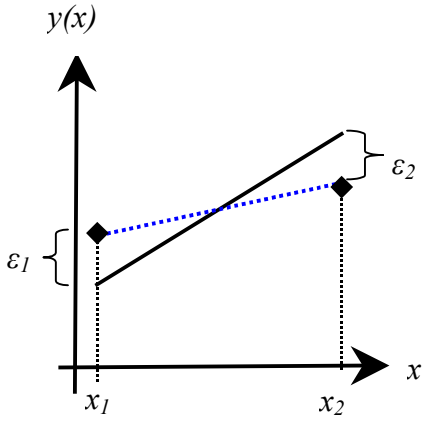

Figure 2. By moving the samples to the boundaries of the design space, the effect of the random error terms is reduced. Now, the estimated linear model (dashed) is a better approximation to the true linear model (solid).

From a statistical perspective, it is easier to discriminate the response trend component from the random error component when the sample sites are spaced as far apart as possible. This principle is followed in classical DOE methods that place samples on the boundaries and/or vertices of the design space, but place very few samples in the interior of the design space.

Another feature of classical DOE that differs from modern DOE is the use of replicated sampling. Since the measured re-

sponse, y_m, contains a random error term, repeated measurements taken for identical design variable values will result in slightly different y_m values. Classical DOE methods typically employ replicated sampling to permit a lack-of-fit statistical analysis and to measure the magnitude of the error term in Equation (1).[1]

2.3 Classical DOE Example

Figure 3 illustrates a commonly used classical DOE technique, central composite design (CCD), for the design space $[0,1]^2$. In CCD, the number of samples grows with the dimension of the design space, n, according to the formula $2^n + 2n + 1$. The 2^n samples correspond to the "corner" points at the vertices of $[0,1]^2$, while the $2n$ samples correspond to the points that lie outside of $[0,1]^2$ (where the distance of these points from the center of the design space changes with respect to n, see Ref. 1). If this CCD were to be used on a laboratory experiment, replicated samples would be taken, at least, at the center of the design space, and at all sample sites if economically possible.

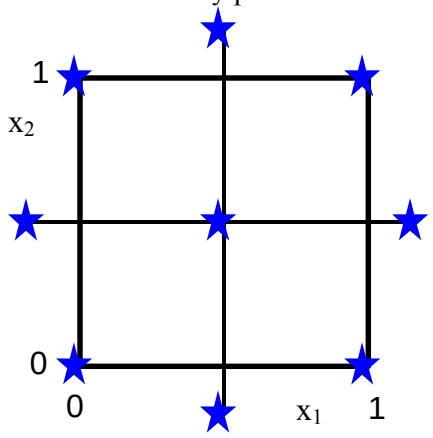

Figure 3. A central composite design from classical DOE for n=2. Note that the number of sample sites (stars) scales as the number of vertices, i.e., as 2^n.

Figure 3 clearly illustrates some of the drawbacks to classical DOE. That is, at best, the number of samples in CCD scales as 2^n; a rate that can be unacceptable if n is large and/or if experiments are expensive. Furthermore, CCD and other classical DOE methods tend to place samples on or near the boundary of the design space, leaving the interior of the design space largely unexplored. For example, in the two-dimensional CCD shown in Figure 3, eight of the nine samples are on or outside the boundary of the design space, and only one sample, the center point, lies in the interior of the design space. Similar trends are exhibited by Box-Behnken design and many other classical DOE methods.

2.4 Discussion – Modern DOE

In modern DOE applied to deterministic computer experiments, there is no notion of random error. That is, if a computer simulation is run twice with exactly the same input data, then the output data produced from both simulations will, in general, be exactly the same. In addition to the assumption that there is no random error in a computer experiment, an additional assumption made in modern DOE is that the true response trend is unknown. For this reason, modern DOE methods tend to place samples on the interior of the design space in what is often termed a "space-filling" set of samples. Sampling in the interior of the design space is performed in an effort to minimize *bias error*. Bias errors arise when there is a difference between the functional form of the true response trend, and the functional form of the assumed or estimated trend. For example, if the true trend is a cubic polynomial and the assumed trend is a quadratic polynomial, then bias error reflects the inability of a quadratic function to model the trends in the cubic function, irrespective of how many data samples are used.

Myers and Montgomery[1] provide an example that demonstrates how sampling in the interior of the design space can reduce bias error in an approximation model. In this example, the true function trend is a quadratic function of a single variable, $\boldsymbol{x}$, and the approximation model is a linear function. Myers and Montgomery show that the bias error in the approximation model is reduced by placing two samples inside the interval $[\boldsymbol{x}_L, \boldsymbol{x}_U]$, rather than at the endpoints of the interval. While the sampling approach de-

scribed by Myers and Montgomery does not correspond to any of the modern DOE methods described below, conceptually there are many similarities.

The remaining sections of this report describe various modern DOE approaches. This is not intended to be an all-inclusive list, but rather it is intended to serve as an overview of some of the more commonly used techniques in modern DOE.

3. Pseudo-Monte Carlo Sampling

3.1 Basic Method

Pseudo-random sampling, also known as pseudo-Monte Carlo sampling, was first applied to computer simulations by Metropolis and Ulam[3] in 1949. The prefix *pseudo-* refers to the use of a pseudo-random number generation algorithm that is intended to mimic a truly random natural process. In many cases, the pseudo- prefix is dropped and this class of DOE methods is known simply as Monte Carlo methods. However, it is important to note that pseudo-Monte Carlo methods differ from quasi-Monte Carlo methods which do not use random number generation algorithms. More information on quasi-Monte Carlo sampling is provided in Section 6.

Given an interval $[\boldsymbol{x}_L, \boldsymbol{x}_U]$, pseudo-Monte Carlo (MC) sampling selects a random number that lies in the interval. For a 1-dimensional design space this random number is the sample site. Of course, this sampling approach is readily extended to an n-dimensional design space $[\boldsymbol{x}_L, \boldsymbol{x}_U]^n$ in which the sample site is an ordered n-tuple. Figure 4 shows MC samples in a two dimensional design space on the interval $[0,1]^2$.

For design spaces that are convex but not rectangular, it is relatively straightforward to adapt MC sampling to the design space, either by enforcing simple geometric properties (e.g., to produce MC samples in a circular region, inscribe the circle in a square, sample over the square and discard samples that fall outside of the circle), or by applying some type of transformation that maps the boundary of the rectangular design space to the boundary of the convex design space. For nonconvex design spaces, MC sampling can be more difficult to employ, especially in high-dimensional design spaces, depending on how the nonconvex region is defined.

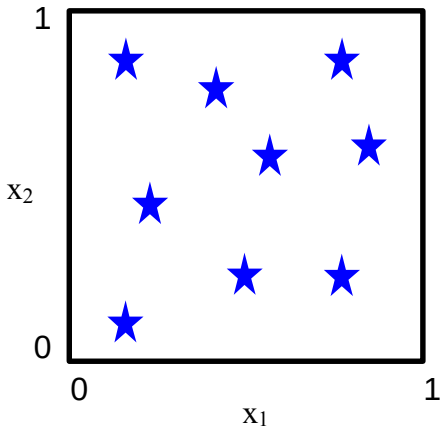

Figure 4. An example of pseudo-Monte Carlo sampling in a two-dimensional design space. The sample sites (stars) are randomly placed in the interval $[0,1]^2$.

A nontrivial aspect of MC sampling is the selection of a reliable algorithm to generate random numbers. This topic is covered in numerous texts on numerical methods and statistics and is not addressed here.

While MC sampling is simple to implement, a set of MC samples will often leave large regions of the design space unexplored. This occurrence stems from the random and independent nature of the sample sites produced by a random number generator. Several modern DOE methods have been developed to address this deficiency of basic MC sampling. Some of these are variants are described below.

3.2 Stratified Monte Carlo Sampling

The stratified Monte Carlo sampling method[4] was developed in an effort to provide a more uniform sampling of the design space as compared to the basic MC sampling approach. In stratified Monte Carlo sampling each of the n intervals of $[\boldsymbol{x}_L, \boldsymbol{x}_U]^n$ is divided into subintervals, or "bins," of equal probability. In the case where all of the design variables have uniform probabil-

ity distributions, the bins are of equal size. Once the bins are defined, a sample site then is randomly selected within each bin.

An example of stratified MC sampling is shown in Figure 4, where there are two design variables, x_1 and x_2, both of which have uniform probability distributions. The interval along x_1 is subdivided into four bins and the interval along x_2 is subdivided into three bins. This yields 12 equally sized bins.

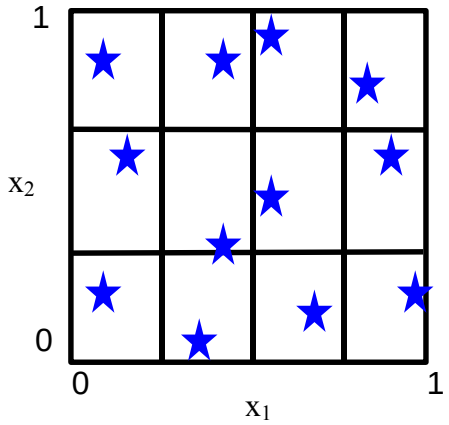

Figure 5. Stratified Monte Carlo sampling where the bins are sized to have equal probability, and a sample is randomly placed in each bin.

Stratified MC sampling provides better overall coverage of the design space than does basic MC sampling. In addition, the user has flexibility in choosing the number of subintervals created in each interval in $[\boldsymbol{x}_L, \boldsymbol{x}_U]^n$. This allows the user to control the number of bins in the design space to best match the available computational budget. Some of the other modern DOE methods do not permit the user to specify a different number of bins for each interval.

A drawback to stratified MC sampling is that the number of samples scales at best as 2^n, i.e., two bins for each design variable interval. In cases where n is large, and/or where computer simulation runs are expensive, it may not be possible to evaluate $O(2^n)$ samples.

3.3 Monte Carlo Sampling Algorithm

As stated above, an integral aspect of pseudo-Monte Carlo sampling is the selection of a reliable method for generating random real-valued numbers. In addition to the numerous textbooks on numerical methods that cover random number generation, many computer languages have random number generation functions/subroutines as standard features. Modern DOE software packages that contain variants of pseudo-MC sampling are described in Section 7.

The field of Monte Carlo sampling is much more broad than presented here. For example, there are entire texts devoted to variants of Monte Carlo sampling in situations where the design variables have non-uniform probability distributions, or where correlations exist among the design variables. For additional information on Monte Carlo sampling and its many variants, see the texts by Sobol[5], Niederreiter[6], and Evans and Swartz[7].

4. Latin Hypercube Sampling

4.1 Overview

Latin hypercube sampling (LHS) is one popular modern DOE method that has found wide application in computational applications. McKay, et al, originally developed the LHS method.[8] as an alternative to pseudo-Monte Carlo sampling. Under certain assumptions associated with the function to be sampled, Latin hypercube sampling provides a more accurate estimate of the mean value of the function than does Monte Carlo sampling. That is, given an equal number of samples, the LHS estimate of the mean will have less error (i.e., smaller confidence bounds) than the mean value obtained through Monte Carlo sampling.

Another attractive aspect of LHS is that it allows the user to tailor the number of samples to the available computational budget. That is, a LHS experimental design can be configured with any number of samples and is not restricted to sample sizes that are specific multiples or powers of n. For example, with an expensive computer simulation with n design parameters, one may only be able to afford $O(n)$ samples. In contrast, for many classical DOE methods, and some modern DOE methods, the number of

samples scales as $O(2^n)$, e.g., central composite design and stratified-MC sampling.

Figure 6 demonstrates Latin hypercube sampling on a two-variable parameter space. Here, the range of both parameters, x_1 and x_2, is [0,1]. Also, for this example both x_1 and x_2 have uniform probability distributions. For p Latin hypercube samples, the range of each parameter is divided into p bins of equal probability. For n design parameters, this partitioning yields a total of p^n bins in the parameter space. Next, p samples are randomly selected in the parameter space, with the following restrictions: (a) each sample is randomly placed inside a bin, and (b) for all one-dimensional projections of the p samples and bins, there will be one and only one sample in each bin.

In a two-dimensional example such as that shown in Figure 6, the sample placement rules for LHS guarantee that only one bin can be selected in each row and column. For $p=4$, there are four partitions in both x_1 and x_2. This gives a total of 16 bins, of which four will be chosen according to criteria (a) and (b) described above. The stars in Figure 6 represent the four sample sites in this example, where each sample is randomly located in its bin.

An inspection of Figure 6 shows how LHS criterion (b) is satisfied. That is, if all four of the samples are projected vertically down to the x_1 axis, there would be one sample in each bin along x_1. Similarly, if the four samples were projected horizontally over to the x_2 axis, there would be one sample in each of the x_2 bins.

This insight is useful in visualizing LHS sampling in three dimensions. For example, for n=3, add a third axis, x_3, (out of the page) to Figure 6. This would produce 64 equally sized bins in the design space. Next, re-position the four samples in the vertical direction so that one sample is located in each interval along x_3. Again, all one-dimensional projections of the four samples would yield one sample per bin along each axis.

One drawback to Latin hypercube sampling is that there is more than one possible arrangement of bins and samples that meets the LHS criteria. For example, the four sample sites in Figure 6 could have been placed in the four bins along either diagonal. Not only is this a poor arrangement of samples with respect to coverage of the design space, but also the four samples sites would be nearly co-linear (in statistical jargon, this is known as high spatial correlation). This can lead to ill conditioned systems of linear equations when the sample site data are used in other calculations (e.g., linear regression).

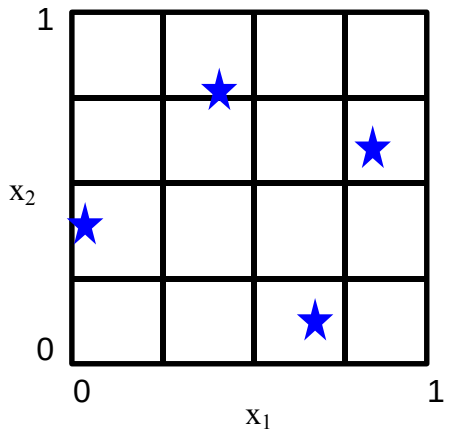

Figure 6. Latin hypercube sampling with four bins in each of the parameters x_1 and x_2. The stars are sample sites randomly selected inside each bin.

Fortunately, there are extensions to the basic LHS technique that minimize the amount of correlation among the sample sites, or, at the user's discretion, can enforce a user-prescribed correlation among the samples. These techniques are available in some, but not all software packages that perform Latin hypercube sampling. One software package that does employ the correlation minimizing approach is the "LHS" code originally developed by Iman and Shortencarrier.[9] See Section 7 for more information on the LHS code.

4.2 LHS Algorithm

The algorithm that generates Latin hypercube sample sites is described in detail by Koehler and Owen.[4] This algorithm is presented below using a slightly different notation to maintain consistency within this

document. The algorithm for generating LHS points is:

$$x_j^{(i)} = \frac{\pi_j^{(i)} + U_j^{(i)}}{k}, \qquad (3)$$

for $1 \le j \le n$ and $1 \le i \le k$,

where k is the number of samples, n is the number of design variables, U is a uniform random value on [0,1], and π is an independent random permutation of the sequence of integers 0,1,…,k-1. The subscript j denotes the dimension index and the superscript *(i)* denotes the sample number. Note that there are $p!$ permutations of the sequence of integers in π, all of which are equally probable.

For example, the Latin hypercube samples shown in Figure 6 were constructed using the sequence $\pi^{(1)}_1,\ldots,\pi^{(4)}_1 = 0,1,2,3$ and the sequence $\pi^{(1)}_2,\ldots,\pi^{(4)}_2 = 1,3,0,2$. These sequences for π are then arranged as consecutive columns in a matrix. This produces the (row, column) bin location of each sample, e.g., (0,1), (1,3), (2,0), and (3,2), where bin (0,0) is in the lower left corner of Figure 6. The approximate values of the random value sequences are $U^{(1)}_1,\ldots,U^{(4)}_1 = 0.09, 0.63, 0.71, 0.34$ and $U^{(1)}_2,\ldots,U^{(4)}_2 = 0.60, 0.08, 0.54, 0.41$. These random values determine the location of each sample within its respective bin.

A simple modification to the Latin hypercube sampling algorithm produces what is known as *lattice sampling*. In lattice sampling the $U_i^{(j)}$ sequence is replaced with a fixed value of 0.5. The result is that each sample is placed at the center of its respective bin, rather than randomly within the bin.

As with stratified Monte Carlo sampling, it is possible to perform Latin hypercube sampling and lattice sampling for design variables that have non-normal probability distributions as well as correlations among the variables. See the text by Helton and Davis[10] for more information on variants of Latin hypercube sampling that are applicable in these cases.

5. Orthogonal Array Sampling

5.1 Overview

Conceptually, orthogonal array (OA) sampling shares many similarities with Latin hypercube sampling, and in fact, the OA algorithm can be used to produce Latin hypercube samples.[4,11] The key feature of OA sampling is that it produces a set of samples that yield uniform sampling in any t-dimensional projection of an n-dimensional design space (where $t<n$). In Latin hypercube sampling, $t=1$, and hence LHS can be considered to be a special case of orthogonal array sampling. The value of t is known as the *strength* of the OA.

An orthogonal array is characterized by five integer values with the notation OA(k,n,p,t) and the expression $k=\lambda p^t$. In this notation, k is the total number of samples, n is the dimension of the design space, p is the number of bins in each variable (p is the same for all variables), t is the strength of the array, and λ is the *index* of the array. The index term refers to the number of samples that occur in each bin following a t-dimensional projection of the samples.

Note that the term *orthogonal* in OA is not related to the definition of orthogonal as used in linear algebra (i.e., where two n-dimensional vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$, are orthogonal if $\boldsymbol{u}^T\boldsymbol{v}=0$). Rather, in the context of OAs, orthogonality is a special property of the matrix of integers (i.e., the *array*) used to generate sample sites. This $k \times n$ matrix, $\boldsymbol{A}$, is said to be orthogonal if for any t columns of the matrix ($t<n$), each ordered t-tuple appears exactly λ times.

Figure 7 illustrates a simple orthogonal array having four samples in a three-dimensional design space. The three design variables have uniform probability distributions, and each interval is subdivided into two bins. Each shaded bin contains one randomly placed sample site. Thus, for all two-dimensional projections of the design space, there is one sample in each bin. Using the OA notation described above, this set of samples is represented as OA(4,3,2,2) with λ=1. The array, $\boldsymbol{A}$, that produced the shaded bins is:

$$A = \begin{bmatrix} 0\,0\,0 \\ 1\,0\,1 \\ 1\,1\,0 \\ 0\,1\,1 \end{bmatrix}.$$

Note that for any t=2 columns of $\boldsymbol{A}$, each of the four bins in the two-dimensional projection (i.e., (0,0), (1,0), (0,1), (1,1)), appears one time. Hence, λ=1. Also note that the integers in $\boldsymbol{A}$ correspond to the p bins in each interval, where the bins are numbered 0, 1, …, p-1.

To produce an OA with λ=2, consider the case where Figure 7 is changed so that there is one sample point in each of the eight bins. In such a case, $\boldsymbol{A}$ becomes:

$$A = \begin{bmatrix} 0\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \\ 1\,0\,0 \\ 0\,1\,1 \\ 1\,1\,0 \\ 1\,0\,1 \\ 1\,1\,1 \end{bmatrix},$$

and all two-column projections of $\boldsymbol{A}$ yield two samples in each bin.

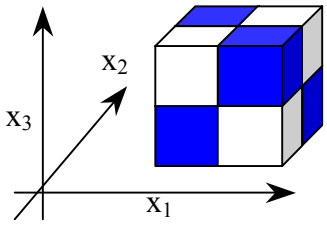

Figure 7. A four sample, strength=2 orthogonal array in a three-dimensional design space. There is one sample in each of the shaded bins.

A disadvantage of OA sampling is that the user does not have the ability to specify an arbitrary number of samples if strength ≥ 2 is desired. That is, there are only certain values of k, n, p, t, and λ that generate an $\boldsymbol{A}$ array that satisfies orthogonality. As an example, OA(36,4,6,2) does not exist.[12]

Since OA generation can be nontrivial, and since there can be several valid permutations of an OA, various often used arrays have been published in tabular form (cf. Reference 11). With the bin locations specified by a given array, it is then left to the user to place the sample site at either the center of the bin or randomly within the bin.

In addition to tables of OAs, some public-use software products are available for generating OA samples. See Section 7 for more information.

5.2 Orthogonal Array Algorithm

The algorithm for generating orthogonal array samples is:

$$x_j^{(i)} = \frac{\pi_j(A_j^{(i)}) + U_j^{(i)}}{p}, \qquad (4)$$

for $1 \le j \le n$ and $1 \le i \le k$.

As with the LHS algorithm, k is the number of samples, n is the number of design variables, U is a uniform random value on [0,1], the subscript j denotes the dimension index and the superscript (i) denotes the sample number. The term $\pi_j(A_j^{(i)})$ denotes the $(i, j)^{\text{th}}$ element in array $\boldsymbol{A}$, where the columns of $\boldsymbol{A}$ satisfy the OA definition of orthogonality.

As with the pseudo-Monte Carlo and Latin hypercube sampling methods, it is possible to employ OA sampling with design variables having non-uniform distributions. Some of the available OA software packages (see Section 7) are able to accommodate non-uniform design variables.

6. Quasi-Monte Carlo Sampling

6.1 Background

The final class of sampling methods addressed in this paper is known as quasi-Monte Carlo methods, or, alternatively, as low-discrepancy sampling.[6] As mentioned in Section 3, there is a difference between quasi-Monte Carlo sampling and the classical pseudo-Monte Carlo approach. However, both quasi-Monte Carlo sampling and pseudo-Monte Carlo sampling have a common heritage in that both methods were developed for multidimensional integration.

The *quasi-* prefix refers to a sampling approach that employs a deterministic algorithm to generate sample sites in an n-dimensional space, so that the points are as

close as possible to a uniform sampling. The term *discrepancy* refers to a quantitative measure of how much the distribution of samples deviates from an ideal uniform distribution. Hence, low-discrepancy is a desired feature of this class of sampling methods. Stated another way, quasi-MC sampling seeks to distribute the sample sites evenly throughout the design space, but does not employ a regular grid or a Cartesian lattice of sample sites. Conceptually, think of the quasi-MC sample sites as a cloud of electrons that, due to electrostatic repulsion, move around inside an *n*-dimensional space $[0,1]^n$ until some sort of minimum energy state is reached. In the final state, the electrons would not lie on a regular grid, but instead would be nearly equally spaced, but unstructured, throughout $[0,1]^n$.

6.2 Error Bounds for Numerical Integration Methods

The text by Niederreiter[6] provides a discussion of the error bounds in multidimensional integration when using classical integration methods (e.g., trapezoidal rule), pseudo-Monte Carlo sampling, and a particular variant of quasi-Monte Carlo sampling that uses the Hammersley sequence.[13] In summary, these error bounds are listed in Table 1 where N is the number of samples and n is the dimension of the design space.

For this comparison, all of the pseudo-MC and related sampling methods (basic-MC, stratified-MC, Latin hypercube, orthogonal arrays) have the same order-of-magnitude error bounds. The differences between the various pseudo-MC methods are in the constant terms rather than in the exponent on the number of samples.[14]

Note that the pseudo-MC error bound is a *probabilistic bound* (due to random sampling) whereas the classical integration and quasi-MC error bounds are absolute (due to deterministic sampling). Figure 8, Figure 9, and Figure 10 plot these error bound trends versus increasing N for *n*=2, *n*=3, and *n*=5, respectively.

Table 1. Error bounds for numerical integration methods (obtained from Ref. 6).

Method	Error Bound
Classical Integration	$O(N^{-2/n})$
Pseudo-Monte Carlo	$O(N^{-1/2})$
Quasi-Monte Carlo	$O(N^{-1}(\log_{10}N)^{n-1}))$

In Figure 8 both classical integration and the Hammersley sequence (HS) variant of quasi-MC sampling have lower error bounds than pseudo-MC sampling. Stated another way, for a fixed number of samples, numerical integration in a two-dimensional design space using either classical methods or quasi-MC samples will be more accurate than integration using pseudo-MC samples.

However, the error trends shown in Figure 8 are not maintained when considering higher dimensional design spaces. Figure 9 shows that quasi-MC sampling leads to lower integration errors than the other two methods for *n*=3. The same is true for *n*=4 (plot not shown).

For $n \geq 5$ the error bound trends shift again. Figure 10 (for *n*=5) shows that for $100 \leq N \leq 1.0\times10^6$, pseudo-MC sampling yields lower errors, whereas for $N \leq 100$ and $N \geq 1.0\times10^6$ quasi-MC sampling yields lower errors. These general trends hold in higher dimensional spaces. That is, pseudo-MC sampling yields lower integration errors for most reasonable values of N that one might use in an actual numerical integration task.

In summary, for n<5, quasi-MC sampling leads to lower integration errors than pseudo-MC sampling. For $n \geq 5$ the *average* error of pseudo-MC sampling is lower than the *exact* error of quasi-MC sampling over most reasonable values of the number of samples. However, since the error bound on pseudo-MC sampling is a probabilistic quantity, there is no guarantee that any particular set of pseudo-MC samples attains this error bound. For this reason, many researchers in the numerical integration community prefer using quasi-MC sampling for numerical integration when $n \geq 5$, since the error bound is exactly known. Interestingly, there is current research in the statistics commu-

nity that explores the combination of pseudo-MC and quasi-MC methods.[14]

6.3 Quasi-Monte Carlo Sampling as a Modern DOE Method

The text above addresses the error bounds for numerical integration given a set of N points in an *n*-dimensional design space, but it does not specifically address the use of quasi-MC sampling as a modern design of experiments method. It is reasonable to argue that the main attribute of quasi-MC sampling that leads to well-characterized error bounds, i.e., uniform sampling of an *n*-dimensional space, is also a desirable feature of a modern DOE method. That is, the goal of modern DOE approaches is to gather information on the trends of the response function(s) over the entire design space. Thus, a quasi-MC sampling approach that uniformly disperses sample sites throughout the design space should be an attractive DOE method.

One example of the use of the Hammersley sequence variant of quasi-Monte Carlo sampling as a DOE method is described by Kalagnanam and Diwekar.[15] This source also provides an algorithm for generating Hammersley sequence points. This algorithm is provided below, with minor modifications to preserve consistency with the notation used earlier in this document.

6.4 Hammersley Sampling Algorithm

The algorithm that generates a set of N Hammersley points makes use of the radix-*R* notation of an integer. That is, a specific integer, *p*, in radix-*R* notation can be represented as

$$p = p_m p_{m-1} \cdots p_2 p_1 p_0$$
$$p = p_0 + p_1 R + p_2 R^2 + \ldots + p_m R^m$$

where $m = [\log_R p] = [(\ln p)/(\ln R)]$, and the square brackets, [ ], denote the integer portion of the number inside the brackets. For example, in the familiar base-10 (i.e., radix-10) number system, the integer 756 has $p_0 = 6$, $p_1 = 5$, and $p_2 = 7$, with R=10 and m=2.

The inverse radix number function constructs a unique number on the interval [0,1] by reversing the order of the digits of *p* around the decimal point. The inverse radix number function is:

$$\phi_R(p) = .p_0 p_1 p_2 \cdots p_m$$
$$\phi_R(p) = p_0 R^{-1} + p_1 R^{-2} + \ldots + p_m R^{-m-1}$$

Finally, the Hammersley sequence of *n*-dimensional points is generated as

$$x_n(p) = \left(\frac{p}{N}, \varphi_{R_1}(p), \varphi_{R_2}(p), \ldots, \varphi_{R_{n-1}}(p) \right)$$

where p = 0,1,2,…,N-1; and the values for $R_1, R_2, \ldots, R_{n-1}$ are the first *n-1* prime numbers (2,3,5,7,11,13,17,…). This approach generates a set of N points in the *n*-dimensional design space $[0,1]^n$.

Table 2. A Maple™ computer program to generate N Hammersley sequence points in an *n*-dimensional design space $[0,1]^n$. This program is available online from Reference 16.

```
# user supplies values for
# n and N
n:=5:N:=100:

Hammersley:=proc(n,N,I)
local R,x,y,j,k,T;
  T:=[0$k=1..n];
  T[1]:=i/N;
  for k from 2 to n do
    R:=ithprime(k-1);
    for j to i do
      x:=1-T[k];
      y:=1/R;
      while x<=y do
        y:=y/R
      od;
      T[k]:= T[k]+(R+1)*y-1
      od
    od;
  T
end:

plot([‘eval(Hammersley(2,N,i))
’$’i’=0..N-1],x=0..1,y=0..1,
style=POINT,
symbol=CIRCLE,
scaling=CONSTRAINED,
axes=BOXED,
title=`Hammersley Sequence`);
```

Aszkenazy[16] has written a computer program (Table 2) for the Maple™ mathematics software package to generate Hammersley sequence points for an *n*-dimensional space. This algorithm is similar to the algorithm outlined in Kalagnanam and Diwekar, but there are minor differences (e.g., Aszkenazy uses the sequence p=0,1,…,N-1, whereas Kalagnanam and Diwekar use p=1,2,…,N). Using the Hammersley sampling algorithm given in Table 2 with n=5 and N=8 produces the data points listed in Table 3.

Table 3. Eight data points in $[0,1]^5$ produced by the Hammersley sequence algorithm.

Pt. Num.	x_1	x_2	x_3	x_4	x_5
1	0	0	0	0	0
2	1/8	1/2	1/3	1/5	1/7
3	1/4	1/4	2/3	2/5	2/7
4	3/8	3/4	1/9	3/5	3/7
5	1/2	1/8	4/9	4/5	4/7
6	5/8	5/8	7/9	1/25	5/7
7	3/4	3/8	2/9	6/25	6/7
8	7/8	7/8	5/9	11/25	1/49

7. Modern DOE Software

There are numerous sources of modern DOE software, both public domain and commercial. The text given below is intended to cover some of the publicly available, or soon-to-be publicly available, modern DOE software packages. In particular, software developed by staff at Sandia National Laboratories is emphasized, since it is these software packages with which the authors are most familiar.

7.1 Pseudo-Monte Carlo Sampling

If all of the design variables have uniform distributions, then there are many options for software to generate pseudo-MC samples. As stated in Section 3, there are numerous textbooks that describe the generation of pseudo-random number sequences. Also many computer languages provide functions or subroutines that provide pseudo-random number sequences. Thus, it is relatively easy to write a computer program to generate Monte Carlo samples.

One source of publicly available software for pseudo-Monte Carlo sampling is the GNU Scientific Library (GSL).[17] The GSL provides C-language functions in many different areas of mathematics and statistics. For example, the GSL contains functions for random number generation (uniform distribution and many non-uniform distributions), as well functions for several types of linear and nonlinear curve fitting.

In some cases it may be easier to use existing software rather than to write a specialized computer program. The LHS and DDACE software packages, described below, provide various options for pseudo-MC sampling for variables with non-normal distributions.

7.2 Latin Hypercube Sampling

LHS Software Package

The LHS software package developed by Sandia National Laboratories, provides pseudo-Monte Carlo sampling and Latin hypercube sampling, both of which can be used with design variables having various random distributions including Gaussian (normal), lognormal, uniform, loguniform, Weibull, and user-supplied histograms.

In addition, the user can supply a correlation matrix for the design variables and the LHS code will attempt to produce a set of sample sites that best matches the user's correlation matrix. This approach is used to generate uncorrelated samples where the desired correlation matrix is the identity matrix.

LHS is in the public domain as a FORTRAN77 software package. Current efforts are underway at Sandia to develop a FORTRAN90 version of LHS. The current F77 version of LHS is available in the DAKOTA Toolkit (see below), and plans are to incorporate the F90 version of LHS into DA-

KOTA once the newest LHS code becomes available and approved for public release.

DDACE Software Package

The DDACE (Distributed Design and Analysis of Computer Experiments) software package[18] also has been developed by Sandia National Laboratories. DDACE contains both modern and classical design of experiments methods. The modern DOE methods are pseudo-Monte Carlo sampling, Latin hypercube sampling, and orthogonal array sampling (where the OA software library is due to Prof. A. B. Owen, see below). These three modern DOE methods support design variables that have either normal or uniform distributions. The classical DOE methods include central composite design sampling and Box-Behnken design and are only applicable to variables that have uniform distributions.

DDACE is currently in review for release under the GNU general public license. The timeframe for the public release of DDACE is early-to-mid 2003.

7.3 Orthogonal Array Sampling

Prof. A. B. Owen of Stanford has developed a library of C-language functions for generating OA samples. The software package is named "oa.c" and it is available on the StatLib online software repository (see: http://lib.stat.cmu.edu/designs/).[19] As noted above, Owen's orthogonal array sampling package has been incorporated in the DDACE package.

7.4 Quasi-Monte Carlo Sampling

The GNU Scientific Library contains functions for various types of quasi-Monte Carlo sampling and low-discrepancy sequence generation. Another software library of quasi-Monte Carlo methods is "libseq" which is available as a beta-release from the Caltech Multi-Res Modeling Group.[20]

Note that the Hammersley sampling sequence is not available in either of these software packages. The algorithm given by Aszkenazy is the only publicly available Hammersley sampling software program known to the authors.

7.5 DAKOTA Toolkit

The DAKOTA (Design Analysis Kit for Optimization and Terascale Applications) Toolkit[21] is an open-source software framework for systems analysis and design. DAKOTA includes methods for optimization, parameter estimation, sensitivity analysis, uncertainty quantification, design of experiments, and statistical sampling. It also provides parallel computing services and various simulation code interface methods

Both LHS and DDACE have been incorporated into the DAKOTA toolkit, although until DDACE is publicly released it is only available to Sandia users and to other users affiliated with the U.S. Government.

The open-source status of DAKOTA is intended to promote software sharing and co-development through a community of users. Near-term plans for DAKOTA include the implementation of quasi-Monte Carlo sampling methods. Those interested in pursuing a software development collaboration via DAKOTA are encouraged to contact the authors.

8. Summary

This paper has provided an overview of modern design of experiments techniques including pseudo-Monte Carlo sampling (and variants such as stratified-Monte Carlo sampling, Latin hypercube sampling, and orthogonal array sampling) and quasi-Monte Carlo sampling. Modern DOE techniques are preferable to classical DOE techniques when using deterministic computer experiments, since assumptions in classical DOE related to experimental error and non-repeatability are not valid when using deterministic computer simulations.

When pseudo-Monte Carlo and quasi-Monte Carlo methods are used in numerical integration, theoretical predictions of error bounds indicate that quasi-Monte Carlo techniques are preferable to pseudo-Monte Carlo techniques for distributing sample points in the n-dimensional interval $[0,1]^n$

when $n < 5$. For cases where $n \geq 5$, error bound predictions favor pseudo-Monte Carlo sampling over quasi-Monte Carlo sampling for most reasonable sample sizes that would be used in a computational study (although quasi-Monte Carlo is better for extremely large sample sizes, e.g., $\geq 10^7$ samples for n=5). However, there is no absolute method to determine when pseudo-Monte Carlo techniques are preferable to quasi-Monte Carlo techniques since the pseudo-Monte Carlo bounds are probabilistic quantities and the quasi-Monte Carlo error bounds are absolute quantities. Current research in the statistical community is focused on DOE techniques that are combinations of pseudo-Monte Carlo and quasi-Monte Carlo sampling. Thus, the research and development of modern DOE methods is an active field of study.

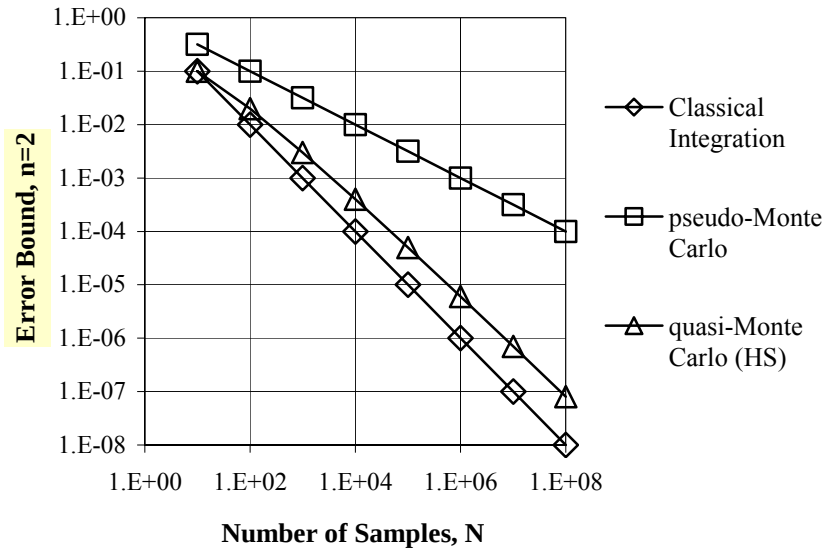

Figure 8. Numerical integration error bounds versus the number of samples for a two-dimensional (n=2) design space.

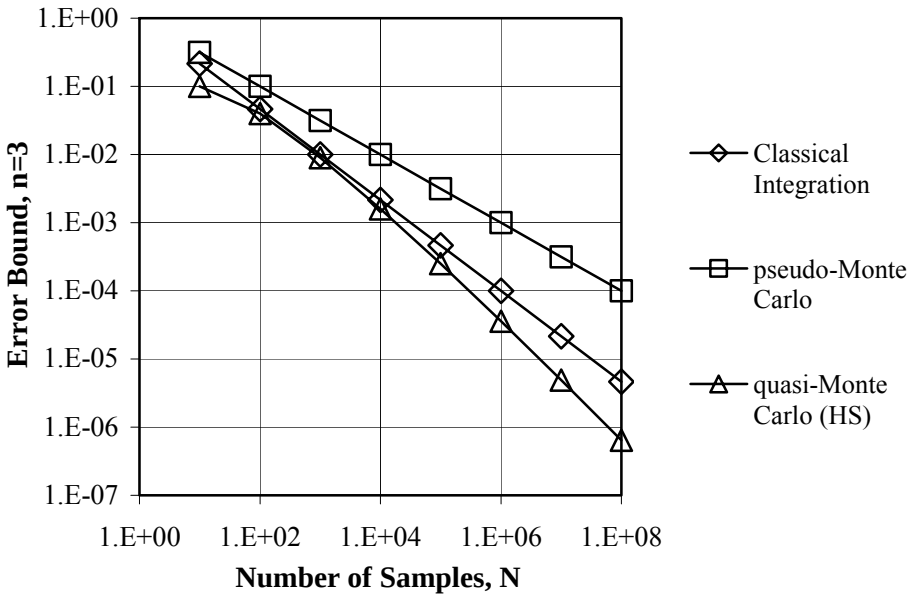

Figure 9. Numerical integration error bounds versus the number of samples for a three-dimensional (n=3) design space.

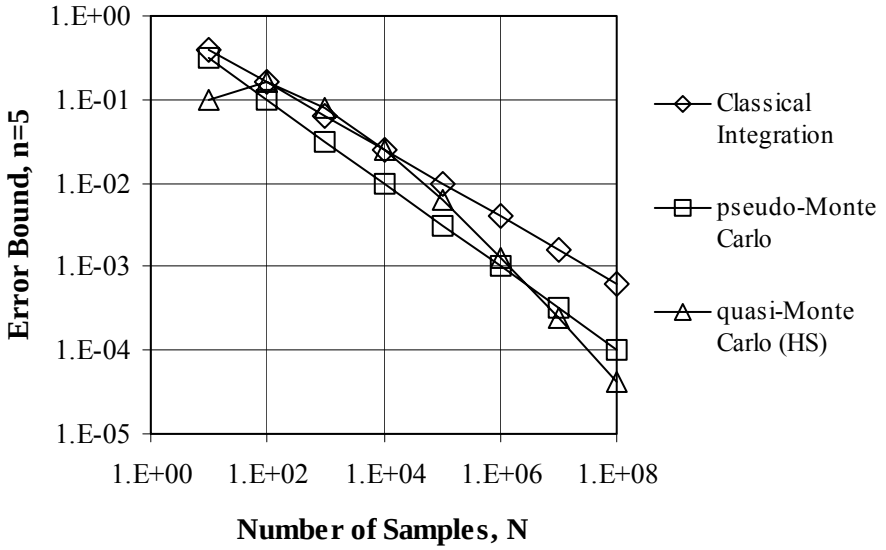

Figure 10. Numerical integration error bounds versus the number of samples for a five-dimensional (n=5) design space.

References

[1] Myers, R. H., and Montgomery, D. C., *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, John Wiley & Sons, Inc., New York, NY, 1995.
[2] Simpson, T. W., Lin, D. K. J., and Chen, W., "Sampling Strategies for Computer Experiments: Design and Analysis," *Intl. J. of Reliability and Applications*, Vol. 2, No. 3, 2001, pp. 209-240.
[3] Metropolis, N., and Ulam, S., "The Monte Carlo Method," *Journal of the American Statistical Association*, Vol. 44, No. 247, 1949, pp. 335-341.
[4] Koehler, J. R., and Owen, A. B., "Computer Experiments," in *Handbook of Statistics* (eds. S. Ghosh and C. R. Rao), Vol. 13, Elsevier-Science, 1996, pp. 261-308.
[5] Sobol', I. M., *A Primer for the Monte Carlo Method*, CRC Press, New York, NY, 1994.
[6] Niederreiter, H., *Random Number Generation and Quasi-Monte Carlo Methods*, SIAM Press, Philadelphia, PA, 1992.
[7] Evans, M., and Swartz, T., *Approximating Integrals via Monte Carlo and Deterministic Methods*, Oxford University Press, Oxford, UK, 2000.
[8] McKay, M. D., Beckman, R. J., and Conover, W. J., "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, Vol. 21, No. 2, 1979, pp. 239-245.
[9] Iman, R. L. and Shortencarier, M. J., ''A Fortran 77 Program and User's Guide for the Generation of Latin Hypercube Samples for Use with Computer Models,'' NUREG/CR-3624, Technical Report SAND83-2365, Sandia National Laboratories, Albuquerque, NM, 1984.
[10] Helton, J. C., and Davis, F. J., "Sampling-Based Methods for Uncertainty and Sensitivity Analysis," Sandia Report SAND99-2240, Sandia National Laboratories, Albuquerque, NM, 2000.
[11] Hedayat, A. S., Sloane, N. J. A., and Stufken, J., *Orthogonal Arrays: Theory and Applications*, Springer, New York, NY, 1999.
[12] Owen, A. B., "Orthogonal Arrays for Computer Experiments, Integration and Visualization," *Statistica Sinca*, Vol. 2, 1992, pp. 439-452.
[13] Hammersley, J.M., "Monte Carlo Methods for Solving Multivariable Problems," *Annals of the New York Academy of Sciences*, Vol. 86, Art. 3, 1960, pp. 844-874.
[14] Owen, A.B., "Monte Carlo Extension of Quasi-Monte Carlo," Winter Simulation Conference Proceedings. (D. J. Medieiros, E.F. Watson, M. Manivannan, and J. Carson, Eds.), 1998, pp. 571—577. (see: http://www-stat.stanford.edu/~owen/reports/)
[15] Kalagnanam, J. R. and Diwekar, U. M., "An Efficient Sampling Technique for Off-line Quality Control," *Technometrics*, Vol. 39, No. 3, 1997.
[16] Aszkenazy, W. O., "MUG: Hammersley/Halton Sequence Generation," online document: http://www-math.math.rwth-aachen.de/MapleAnswers/456.html, Oct. 2002.
[17] Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., and Rossi, F., *GNU Scientific Library: Reference Manual Edition 1.3*, Dec. 2002. (see http://www.gnu.org/software/gsl)
[18] Tong, C. H., and Meza, J. C., ''DDACE: A Distributed Object-Oriented Software with Multiple Samplings for the Design and Analysis of Computer Experiments,'' Sandia Report (in preparation), Sandia National Laboratories, Livermore, CA. (see http://csmr.ca.sandia.gov)
[19] Owen, A. B., Software Package "oa.c", StatLib online repository of statistics software. (see: http://lib.stat.cmu.edu/ and http://lib.stat.cmu.edu/designs/)
[20] Friedel, I., and Keller, A., "Fast Generation of Low-Discrepancy Point Sets," online document, Caltech Multi-Res Modeling Group, 2002. (see: http://www.multires.caltech.edu/software/libseq/)
[21] Eldred, M. S., Giunta, A. A., van Bloemen Waanders, B. G., Wojtkiewicz, S. F., Jr., Hart, W. E. and Alleva, M. P., *DAKOTA Users Manual: Version 3.0*, Sandia Technical Report SAND2001-3796, Sandia National Laboratories, Albuquerque, NM, 2001. (see: http://endo.sandia.gov/DAKOTA/software.html)