STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **15**, January 2026, pp 162–175.
Published online in International Academic Press (www.IAPress.org)

Algorithms for Improving the Designs of Building Universal Kriging Models

Shaymaa M. Younus, Younus Al-Taweel*, Zainab Abdulateef Rasheed

Department of Mathematics, College of Education for Pure Science, University of Mosul, Iraq

Abstract Kriging models (KMs) have been used popularly in the analysis of computer experiments (CEs). Fast-running alternative models for computationally demanding computer codes (CC) are created through Kriging models. To predict the CC response at untested observations using the values at observed points, the design points should be carefully selected. Latin hypercube design (LHD) is a stratified design that has become popular for building KMs. However, in some cases, additional points may need to be added to the LHD to improve its performance. Several algorithms have been proposed for this purpose. In this paper, we investigate and examine the behavior of various LHD algorithms for generating the design points by constructing several KMs to minimize the prediction errors. The KMs that are based on various LHD algorithms are applied to real CCs, and their predictions are compared using some proposed measures.

Keywords Kriging Models, Latin Hypercube Algorithms, Computer Codes, Steel Column function.

AMS 2010 subject classifications 62F15, 62J20.

DOI: 10.19139/soic-2310-5070-2885

1. Introduction

Computer experiments (CEs) have increasingly become popular as alternatives for complex real systems. The fact that real experimentation can be expensive or even impossible in certain situations is a strong argument for employing CEs. Numerical simulation of complicated systems is becoming a more appealing option than conducting costly and time-consuming physical tests due to the widespread availability of extensive computing facilities and recent advancements in software development [1, 2, 3, 4]. The study by [5] is considered the official beginning of CEs, however, [6] introduced model-based approaches, which were considered as a new step.

For many complicated systems, doing CEs can be highly expensive [7]. Developing a Kriging model (KM) is an option for conducting these CEs directly [8]. KMs are frequently used in CEs when analyzing a single grid throughout the entire input space is impractical. The original computer simulation is used to evaluate a limited number of points instead. A KM is then applied to the results of these limited runs. The KMs can provide predictions for new inputs and uncertainty quantification without requiring extensive computer simulation to be rerun at the new points. The predicted value will be close to the true CC value if the KM works properly [9, 10].

Predicting the unobserved outputs depends on the observed design points, also known as training points. Therefore, the training points must be carefully chosen. In CEs, choosing an experimental design is essential to creating a useful and instructive KM. The points should be evenly distributed over the experimental area as a good design concept. Latin Hypercube design (LHD) is a popular space-filling choice for CEs [11]. Several works have been conducted to improve LHD performance. For example, [12] proposed the sliced Latin hypercube design

*Correspondence to: Younus Al-Taweel (Email: younus.altaweel@uomosul.edu.iq). Department of Mathematics, College of Education for Pure Science, University of Mosul, Mosul City, Iraq.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

(SLHD) which is a special LHD, that can be divided into slices of smaller LHDs, each of which attains maximum homogeneity in all one-dimensional projections.

In order to improve upon the SLHD, [13] suggested the Maximin Sliced LHD (MSLHD), which divides the Latin square into slices in order to preserve the statistical characteristics and features of the samples while achieving an ideal distribution of elements within each slice. A traditional LHD is first created as a matrix, with one level for each design factor in each row of the matrix. The matrix is then divided into a predetermined and set number of slices. To guarantee an equitable distribution and balance across all variables, the points distribution inside every slice is optimized using a random distribution. Lastly, by limiting overlap within slices and guaranteeing the ideal distribution of points inside every slice, an optimal solution is obtained using one of the optimization algorithms or numerical techniques.

[14] proposed the Maximum projection designs (MaxPro) to improve the LHD performance. The MaxPro maximizes space-filling properties on projections to all subsets of factors. An optimal design criterion was proposed by [15] that can be used with continuous, nominal, discrete, numeric and ordinal input factors. The suggested design may attain good space-filling properties in the whole design space and in all potential low-dimensional projections, and it is versatile in terms of run size and number of factors. The optimal maximin L_2-distance LHDs was proposed by [16] that is depend on a full factorial design using the rotation method. The optimal maximin L_2-distance LHDs have two advantages which are the do not need a computer search and they are orthogonal. A sequential recursive evolution LHD was suggested by [17], that uses a technique for permutation inheritance to optimize and update the LHD. In this study, we present and investigate several algorithms for improving the LHD behavior through the construction of several KMs. Then, KMs predictions are compared via some suggested measures.

The content of this paper is as follows. In Section 2, we present the methodology for constructing KMs for obtaining CC predictions. In Section 3, we discuss the design and experiments for computer experiments, where we present several LHD algorithms. Section 4 suggests some measures for validating KMs. In Section 5, we apply KMs to a real CC represented by the Steel Column function. Then, the results are compared. The conclusion based on the obtained results is given in Section 6.

2. Universal Kriging Models

The CCs are considered to as unknown functions. It can be represented by a prior distribution, such as the Gaussian process (GP). Therefore, the KM can be considered as a distribution to obtain CC predictions and quantify its uncertainty [18]. Suppose $\mathbf{x} \in \chi \in R^p$ is an input and $y = f(\mathbf{x})$ is the output at $\mathbf{x}$. The output y is considered to have a GP, $Y(\mathbf{x})$, such that

$$E(Y(\mathbf{x})) = \mathbf{f}(\mathbf{x})^T\boldsymbol{\beta} \tag{1}$$

and

$$cov(Y(\mathbf{x}), Y(\mathbf{x}+h)) = \sigma^2 R(h|\boldsymbol{\theta}), \tag{2}$$

where $\mathbf{f}(\mathbf{x}) = (f_0(\mathbf{x}), \dots, f_p(\mathbf{x}))^T$ known functions, $\boldsymbol{\beta} = (\beta_0, \dots, \beta_p)$ are of unknown parameters, σ^2 is the constant variance, and $R(h|\theta)$ is the correlation function with unknown correlation parameters $\boldsymbol{\theta} = (\theta_1, \dots, \theta_p)$ [19, 20]. In this study, we used the Gaussian correlation function

$$R(h|\boldsymbol{\beta}) = \exp\left(-\sum_{i=1}^{k}\left(\frac{h_i}{\theta_i}\right)^2\right). \tag{3}$$

Now, suppose $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ is an input set and $y = (y_1, \dots, y_n)$ are their outputs. When the regression, correlation, and variance parameters of the KM are all known, the Kriging predictor will be known as Simple Kriging. The predictive mean, $Y_{SK}(\mathbf{x}^*)$, and the predictive variance $\sigma^2_{SK}(\mathbf{x}^*)$ at a new input $\mathbf{x}^*$ are provided by

$$Y_{SK}(\mathbf{x}^*) = \mathbf{f}_0(\mathbf{x})^T\boldsymbol{\beta} + \mathbf{r}^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\boldsymbol{\beta}) \tag{4}$$

$$\sigma^2_{SK}(\mathbf{x}^*) = \hat{\sigma}^2\left[1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r}\right] \tag{5}$$

where

$$\mathbf{R} = \mathbf{R}(\mathbf{x}_i - \mathbf{x}_j)$$

$$\mathbf{r} = \{R(\mathbf{x}^*, \mathbf{x}_1), \ldots, R(\mathbf{x}^*, \mathbf{x}_n)\}^T$$

$$\mathbf{F} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n))^T$$

The parameters of the covariance function and the external regression, however, are unknown in practice. They are typically estimated by the maximum likelihood estimation method. In this case, the KM predictor will be called as Universal Kriging. Hence, the Kriging predictor and its predictive variance are adjusted to account for parameter estimation [21, 22].

$$Y_{UK}(\mathbf{x}^*) = \mathbf{f}(\mathbf{x}^*)^T\hat{\boldsymbol{\beta}} + \mathbf{r}^T\mathbf{R}^{-1}(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}) \tag{6}$$

$$\sigma^2_{SK}(\mathbf{x}^*) = \hat{\sigma}^2\left[1 - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{r} + (\mathbf{f}(\mathbf{x}^*)^T - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{F})\left(\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}(\mathbf{f}(\mathbf{x}^*)^T - \mathbf{r}^T\mathbf{R}^{-1}\mathbf{F})^T\right)\right] \tag{7}$$

where

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{F}^T\mathbf{R}^{-1}\mathbf{F}\right)^{-1}\mathbf{F}^T\mathbf{R}^{-1}\mathbf{y} \tag{8}$$

$$\hat{\sigma}^2 = \frac{\left(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}\right)^T\mathbf{R}^{-1}\left(\mathbf{y} - \mathbf{F}\hat{\boldsymbol{\beta}}\right)}{n - p - 1} \tag{9}$$

$$\hat{\boldsymbol{\theta}}^2 = \arg\min\left[\frac{n}{2} + \frac{n}{2}\log(2\pi\hat{\sigma}^2 + \frac{1}{2}\log(\det\mathbf{R}))\right] \tag{10}$$

3. Designs For Computer Experiments

The CEs have deterministic outputs and hence, replicas at specific input sets must be avoided since the points in this case do not reveal any more details for the output. Space-filling designs are good for CEs as they make it easier to create accurate KMs. Latin hypercube design (LHD) is a common Space-filling design in CEs [5]. Several algorithms have been suggested recently to improve the LHD performance. For example, [25] presented the translational propagation algorithm for obtaining optimal LHDs without using formal optimization. This technique uses patterns of point placements for optimal LHDs based on the change of the greatest distance requirement, and it consumes very little computing power and provides results almost in real time. [26] suggested a novel algorithm of maximin LHD using successive local enumeration for generating random m points in $n-$dimensional space. [27] proposed the local search-based genetic algorithm for building an optimal LHD that can achieve good space-filling quality of LHD. This algorithm uses adaptive selection operators, probabilistic mutation, and modified order crossover to increase population diversity and accelerate convergence. To improve the search capability, a local search method is also included in the approach. Several algorithms have been proposed for improving the LHD based on adding extra points to the existing LHDs. We review here the LHD and present several algorithms for adding design points to an existing LHD for improving its performance.

3.1. Latin Hypercube Design (LHD)

The LHD is very common in CEs, where it was suggested by [5]. The LHD splits the variable interval into n equally likely subintervals. Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_n$ where $\mathbf{x}_i = x_{i1}, \ldots, x_{ip}$ need to be generated. Hence, LHD points are obtained by

$$\mathbf{X}_{ij} = \frac{\Lambda_{ik} - U_{jk}}{n} \tag{11}$$

where Λ is an $n \times p$ matrix, its columns are independent random permutations of $\{1, \ldots, n\}$, U_{jk} is a sample in $(0, 1)$ and it is independent of Λ. The LHD is simple to generate, which makes it very popular in CES literature. The LHD projections are uniformly distributed throughout the values of every input variable. However, the entire hypercube may not be well filled in these designs [28].

3.2. *Augmented Latin Hypercube Design (ALHD)*

The ALHD is a method that improves an existing LHD sample by adding points to design with preserving the design's Latin characteristics. To keep the original design points precisely in the same place, the ALHD re-divides the original design into $n + m$ intervals. After that, the empty row-column sets are filled at random [28]. The following algorithm can generate the ALHD points:

- Repartition the design into $(n + m)^2$ cells, and then create a new $(n + m)$ by k matrix to house the candidate points, where n is the number of sample points in the original LHD matrix.
- Locate the missing cells by sweeping at random over each column of the repartitioned design.
- Find an empty row for each column at random, then create a random point that corresponds to that row and enter it in the new matrix. Unless $m = 2n$, the case that the new matrix contains precisely m full rows, the new matrix may include more than m points.
- Lastly, for the new matrix, retain just the first m rows. It is assured that the resulting matrix will have m complete rows. Part of the removed rows are filled. The random search for empty cells results in the random selection of the extra candidate points.

ALHD is a practical method of increasing the design space resolution without having to regenerate the whole design. However, Augmentation is achieved randomly. The augmenting points themselves do not always form an LHD. There can be more than one point in each row in the augmented design, or the original design and augmenting points may form an LHD. A strictly uniform Latin hypercube is ensured if the number of augmenting points equals the number of existing points [23].

3.3. *Optimal Augmented Latin Hypercube Design (OALHD)*

The OALHD enhances an already-existing LHD by adding points to the design with preserving its Latin characteristics. It aims to add points optimally to the design in a technique that maximizes S-optimality. S-optimality searches to distribute the design points as far as feasible by maximizing the mean distance between each point and every other point.

Assume that we wish to increase an existing LHD by m points while maximizing S-optimality. Therefore, to generate OALHD points, the following algorithm can be followed:

- After re-partitioning the design into $(n + m)^2$ cells, create a matrix B to store the candidate points, where n is the number of sample points in the LHD matrix.
- Locate the missing cells by sweeping at random over each column of the repartitioned design.
- Examine every column $(1 - k)$ in the repartitioned design at random to determine the locations of the missing cells.
- Determine a random value that fits in an empty cell for each column, then enter the value in B.
- Construct a matrix P with random integers chosen from B cells and substitute the matching cells from B for the cells from P.
- Calculate the distance between each point in P and the existing design and choose the one that maximizes it.
- Update B by removing the cells that correspond to each of its columns.
- Continue until all m points have been filled.

Adding points to an LHS requires careful consideration of how to maintain the desired design characteristics. In the OALHD, the points are added by a technique that maximizes S-optimality. The S-optimal designs may lead to more accurate KM predictions as the points will be spread as evenly as possible, and this minimizes the average variance. However, this makes the OALHD computationally expensive and more complex than creating a new LHS [28].

3.4. *Optimum Seeded Latin Hypercube Design (OSLHD)*

The OSLHD also enhances an existing LHD by adding points to the design with keeping the Latin characteristics of the design. However, it then optimizes the design using the columnwise pairwise (CP) algorithm for optimizing the design without necessarily holding the original design [24].

To generate OSLHD points, the following algorithm can be followed:

- Create a random LHD by adding to the seed design.
- Within each of the matrix columns, exchange two numbers.
- Determine the change that minimizes the distances between each point by computing the inverse of the sum of the distances between all of them.
- Continue going till you reach the halting requirement by going from column to column. The interchanges in a particular row either do not benefit S-optimality, the maximum number of sweeps over the matrix is achieved, or the drop in the inverse of the sum of the distances is minimal in comparison to the initial decrease brought on by the interchange in the first column.

The CP algorithm is applied to all of the matrix columns. The CP algorithm is as follows:

- Select randomly an $n \times p$ design, X.
- Every repetition is divided into p stages. The deletion of $\mathbf{x}_i$ occurs at the ith stage. Instead of $\mathbf{x}_i$, the vector $\mathbf{x}^+$ that produces the highest S-criterion value among the first-order adjustment vectors of $\mathbf{x}_i$ is selected. Each stage's best pairwise improvement is added to the design matrix.
- Repeat step 2 if the final design has the greatest S-criterion value. The iteration ends when the design can no longer be improved.

The S-criterion aims to maximize the harmonic mean distance from each design point to all the other points in the design.Steps 1 - 3 result in a good design that complements the specific initial concept. The ability of this algorithm to find equally good designs for any initial design is not guaranteed; hence, stages 1 - 3 are typically repeated multiple times. The optimal design is the best of the good designs that are produced.

The OSLHD can be useful when we want to add more points to an existing LHD to maintain its properties. It essentially provides a method for producing a sample that is evenly dispersed throughout the design space and statistically representative. Although OALHD depends on the computationally expensive CP algorithm, it may produce more accurate KM predictions [28].

4. Kriging Models Validation Measures

In this article, we present some measures for KMs validation to check their assumptions. The first measure that we use here is the Nash-Sutcliffe model efficiency coefficient, which is given by

$$NSE = 1 - \frac{\sum_{i=1}^{m} \left(Y_{UK}(\mathbf{x}^*) - \mathbf{Y}(\mathbf{x}^*)\right)^2}{\sum_{i=1}^{m} \left(\mathbf{Y}(\mathbf{x}^*) - \frac{1}{m}\sum_{i=1}^{m} \mathbf{Y}(\mathbf{x}^*)\right)^2} \tag{12}$$

where m is the number of test points, $Y_{SK}(\mathbf{x}^*)$ is the predictive mean of the KM given by equation (6), and $\mathbf{Y}(\mathbf{x}^*)$ is the CC output [29]. The KM predictions will be accurate as NSE approaches to 1.

The eigen errors are the second measure that we use to check the accuracy of the KMs. If $\mathbf{K}$ is a standard deviation matrix of the predictive covariance matrix such that $\sigma^2_{UK}(\mathbf{x}^*) = \mathbf{K}\mathbf{K}^T$ and $\mathbf{K}$ is the eigen decomposition matrix, then the eigen errors will be given by

$$E = \mathbf{K}^{-1}\left(\mathbf{Y}(\mathbf{x}^*) - Y_{UK}(\mathbf{x}^*)\right) \tag{13}$$

Every eigen error has a standard Student-t distribution under the GP assumption. However, we can consider the eigen errors to be standard normally distributed since the number of training data is typically large enough. In this work, We used 95% credible intervals, and the critical value for a 95% confidence level in the standard normal distribution is approximately 1.96, which represents the point on the curve such that 95% of the data falls within ∓ 1.96 (approximately ∓ 2) standard deviations of the mean. Thus, eigen errors are expected to be between -2 and 2 for valid KM. Therefore, eigen errors outside $[-2, 2]$ will be considered large and may indicate that the KM predictions are not accurate.

We can plot the eigen errors against the index, where index indicates the order of eigen errors with the largest predictive variance. In this plot, it is anticipated that the errors will range around 0 with no discernible patterns and a constant variance. The variance is overestimated when there are too many minor errors and underestimated when there are too many large errors. Additionally, these situations may indicate that the simulator is a nonstationary [30]. Moreover, poor estimated variance or nonstationarity is indicated by either significant or very tiny errors at the plot's beginning. Large (or very tiny) errors at the plot's end suggest that the correlation length parameters have been overestimated or underestimated, or that the correlation structure that was selected is inappropriate.

5. Applications

In this section, two functions are considered as examples of a CC to see how well the presented KMs performed based on different LHD algorithms. The first function is the Dette and Pepelyshev function which has 8 dimensions, the second function is the Steel Column function which has 9 dimensions. For the design points, we used the lhs in R software, where we used the functions randomLHS, augmentLHS, optAugmentLHS, and optSeededLHS for obtaining samples of the LHD, ALHD, OALHD, and OSLHD designs consequently [28]. We used the DiceKriging package in R software for building the KMs where we used the km function for building the KMs and the predict function for obtining predictions [31].

5.1. Dette and Pepelyshev Function

The Dette and Pepelyshev function is an eight-dimensional model and it is presented by [23]. The output is given by the equation below, where the input is in the range $[0, 1]$.

$$f(\mathbf{x}) = 4(x_1 - 2 + 8x_2 + 8x_2^4) + (3 - 4x_2)^2 + 16\sqrt{x_3 + 1} + (3x_3 - 1)^2 \sum_{i=4}^{8} i \ln(1 + \sum_{i=3}^{i} x_j) \qquad (14)$$

where $\mathbf{x}$ is the input vector at which to evaluate and $f(\mathbf{x})$ is function output evaluated at $\mathbf{x}$.

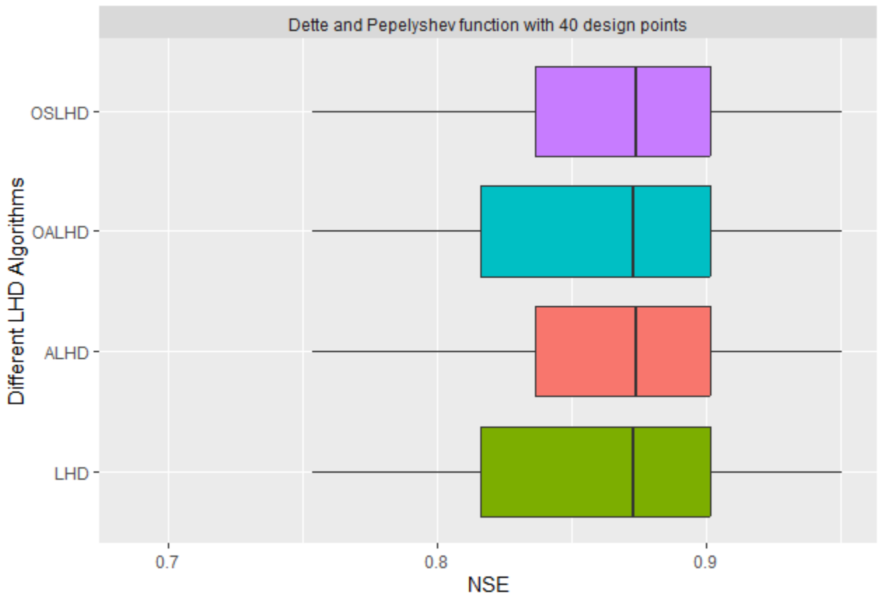

Figure 1. Box plots of NSE values obtained by KMs of the Dette and Pepelyshev function for different LHD algorithms with $n = 40$ training points.

We built KMs based on different LHD algorithms using $n = 5p = 40$ training points where $p = 8$ is the dimension of the Dette and Pepelyshev function. Thus, we have $(\mathbf{x}_1, \ldots, \mathbf{x}_{40})$ and the Dette and Pepelyshev function was evaluated at these inputs to obtain the Dette outputs $\mathbf{y} = (y_1 = f(x_1), \ldots, y_{40} = f(x_{40}))$. The KMs parameters were estimated using the maximum likelihood estimation method (MLE). To validate KMs, we used $m = 2p = 16$ to obtain test points $(x_1^*, \ldots, x_{16}^*)$. Then, we calculated NSE, equation (12), and E, equation (13). We replicated these processes 1000 times for different training and test points.

Figure 1 presents the box plots of the NSE values of the KMs using different LHD algorithms. We can see from Figure 1 that the NSE values seem to be good for all KMs that are built for different LHD algorithms. However, we can see that the NSE values for all the KMs are quite similar. Therefore, to investigate the best performance of the different LHD algorithms, we calculated the mean of NSE values for each KM. Table 1 presents the mean of NSE values for the KMs built using different LHD algorithms with the central processing unit (CPU) time of each algorithm. We can notice that the OSLHD has the best NSE mean value. However, the higher efficiency of the OSLHD goes with longer CPU time.

Table 1. The NSE mean value obtained by KMs using different LHD algorithms with $n = 40$ training points.

Design	LHD	ALHD	OALHD	OSLHD
NSE	0.8738	0.8411	0.8534	0.8830
CPU time (mins)	3.85	5.02	9.63	164

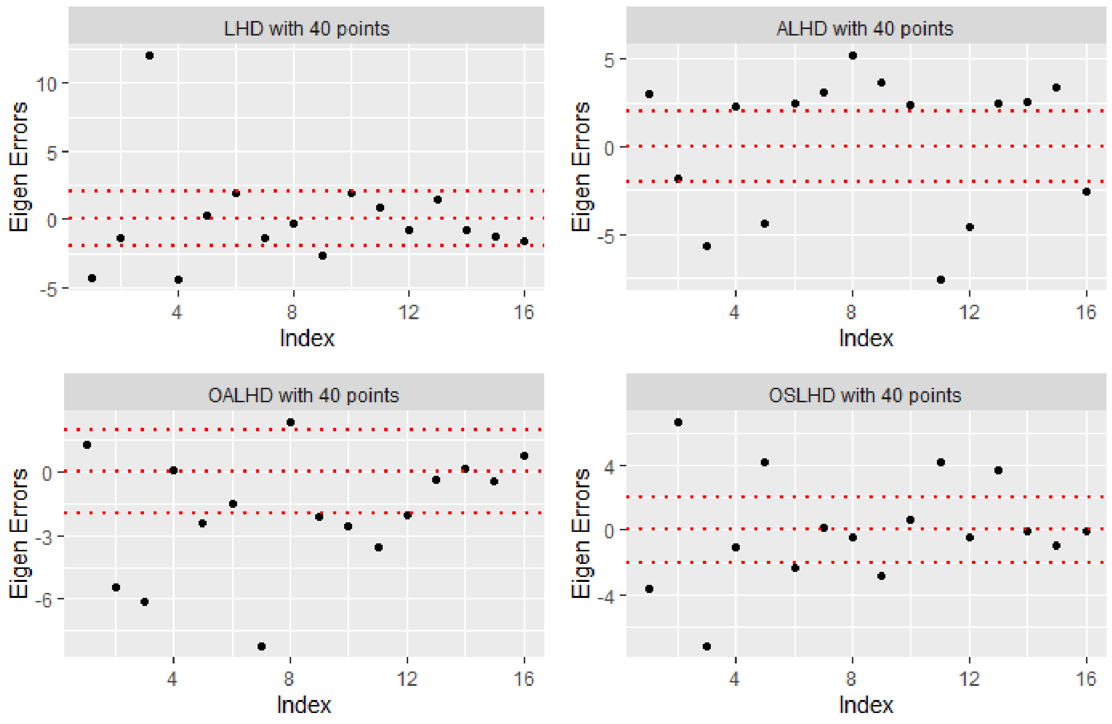

Figure 2. The E plots for the KMs of the Dette and Pepelyshev function based on different LHD algorithms with 40 training points.

We applied the second measure that not only depends on the KM predictions and outputs of the Dette and Pepelyshev function, but also takes into account the prediction uncertainty of KMs. We calculated the eigen errors

(E), equation (13). Then, we plotted eigen errors against the index to compare the KMs performance based on different LHD algorithms.

Figure 2 reviews the E values against the index obtained by KMs using different LHD algorithms with 40 training points and 16 test points. In Figure 2, the top left panel presents the E values obtained by KM using LHD. We see that most of the E values are small with 25% of the eigen errors fell outside the error interval [-2, 2]. The top right panel presents the E obtained by KM using ALHD. It can be shown that about 75% of the eigen errors are seen outside the error interval.

The bottom left panel presents the E obtained by KM using OALHD. It can be seen that 50% of the eigen errors are seen in outside the [-2, 2] interval. Therefore, the E plots of the KMs based on ALHD and OALHD indicate a conflict between the KM prediction and the Dette and Pepelyshev outputs. This may indicate poor estimation of predictive variance. The bottom right panel presents the E values for the KM that is built using OSLHD. It can be shown that about 43% of the E values are seen outside the error interval.

We increased the training points to be $n = 10p = 80$ and the processes are repeated. In Figure 3, we show the box plots of NSE values for KMs using different LHD algorithms with 80 training points and 16 test points. We notice that NSE values seem to be improved for all KMs that are built using different LHD algorithms, as they are now very close to 1. This indicates that the KMs predictions that are built for different LHD algorithms are good approximations for the Dette and Pepelyshev outputs. We also calculated the mean of NSE values for each KM.

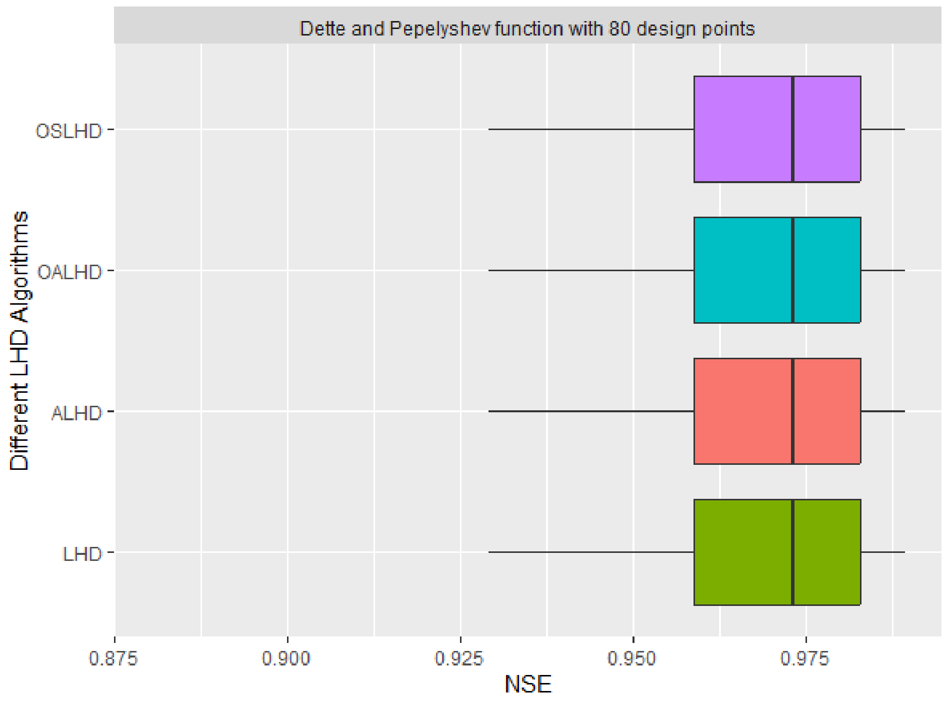

Figure 3. Box plots of NSE values for KMs of Dette and Pepelyshev function using different LHD algorithms with $n = 80$ training points.

Table 2 presents the mean of the 1000 NSE for the KMs that were built using different LHD algorithms with 80 training points and 16 test points. We can also notice that the algorithm of OSLHD produces the best NSE value but takes the longest CPU time.

Table 2. The NSE mean values of KMs of Dette and Pepelyshev function using different LHD algorithms with $n = 80$ training points.

	LHD	ALHD	OALHD	OSLHD
NSE	0.9668	0.9479	0.9701	0.9810
CPU time (mins)	6.32	9.86	16.08	540

We also calculated the eigen errors (E), equation (13) of KMs, using different LHD algorithms with 80 training points and 18 test points. Figure 4 presents the E values against the index for KMs using different LHD algorithms with 80 training points and 16 test points.

The top left panel presents the E values for the KM built using LHD. It can be seen that the values of E have been improved. However, about 18% of the E values are outside the error interval. The top right panel presents the E for the KM built using ALHD. It can be shown that the E values have also been improved, but there are 25% of the E values errors seen outside the error interval. The E plots of the KMs based on LHD and ALHD may indicate a conflict between the KM prediction and the Dette and Pepelyshev outputs.

The bottom left panel presents the E values for the KM built using OALHD. We notice that about 12% of the E values are outside the error interval. The bottom right panel of Figure 4 presents the E values for the KM built using OSLHD. It can be shown that only about 6% of the E values are outside the error interval. Therefore, the predictions of the KMs based on OSLHD are reasonable approximations for the Dette and Pepelyshev outputs.

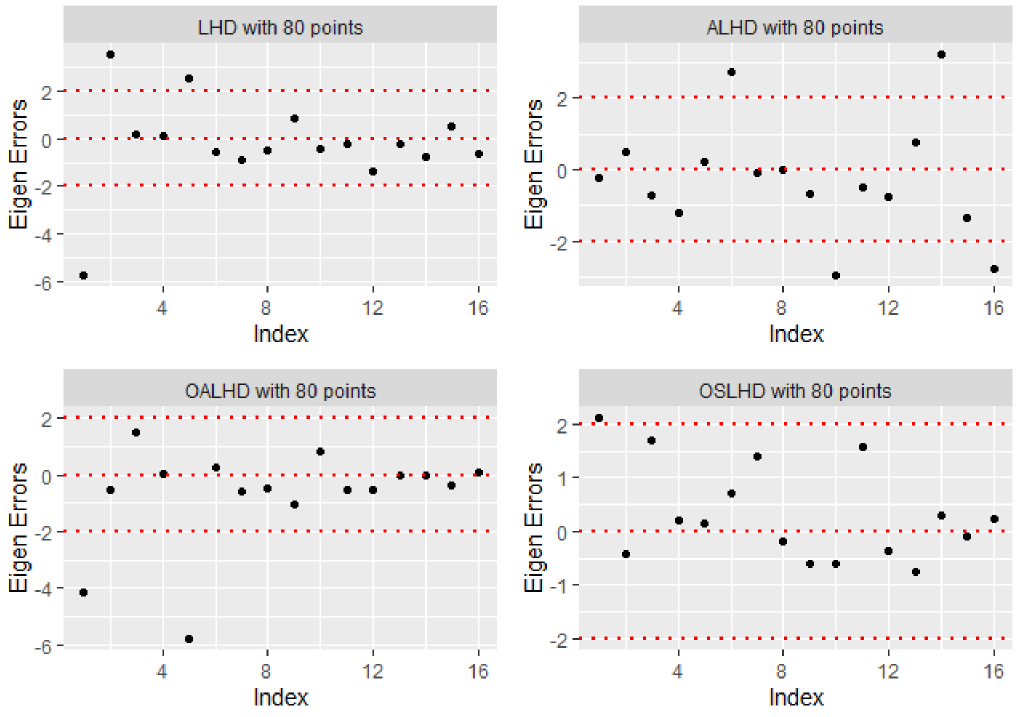

Figure 4. The plots of the E values for the KMs of the Dette and Pepelyshev function using different LHD algorithms with $n = 80$ training points.

5.2. *Steel Column Function*

The Steel Column (SC) function simulates the trade-off between cost and reliability. The steel column cost is: $Cost = bt + 5h$, where t (mm) represents mean flange thickness (mm), b represents the mean flange breadth, and h represents the mean profile height (mm). The column length L is 7500 mm. The values $b = 300$,$d = 20$ and $h = 300$ are used by [32]. The SC function is given by

$$f(\mathbf{x}) = F_s - P\left[\frac{1}{2DB} + \frac{F_0 E_B}{DBH(E_b - P)}\right] \tag{15}$$

where $P = P_1 + P_2 + P_3$ and $\frac{E_b = (\pi^2 EDBH^2)}{(2L^2)}$. The input variables, $\mathbf{x} = (F_s, P_1, P_2, P_3, B, D, H, F_0, E)$, of the SC are: $F_s \in [330, 470]$ is yield stress (MPa), $P_1 \in [400000, 600000]$ is dead weight load (N), $P_2 \in [420000, 780000]$ is variable load (N), $P_3 \in [420000, 780000]$ is variable load (N), $B \in [200, 400]$ is flange breadth (mm), $D \in [10, 30]$ is flange thickness (mm), $H \in [100, 500]$ is profile height (mm), $F_0 \in [10, 50]$ is initial deflection (mm), and $E \in [12600, 29400]$ is Young's modulus (MPa) [32].

We built KMs based on different LHD algorithms using $n = 5p = 45$ training points where p=9 is the dimension of the SC function. Thus, we have $(x_1, \ldots, x_{45})$ and the SC function was evaluated at these inputs to obtain the SC outputs $\mathbf{y} = (y_1 = f(x_1), \ldots, y_{45} = f(x_{45}))$. Before fitting KMs, we transformed each input variable for the SC function to be in the interval [-1,1]. The KMs parameters were estimated using the MLE. To validate KMs, we used $m = 2p = 18$ to obtain test points $(x_1^*, \ldots, x_{18}^*)$. Then, we calculated NSE, equation (12), and E, equation (13). We replicated these processes 1000 times for different training and test points. Figure 5 presents the box plots of the NSE values of the KMs using different LHD algorithms. We can see from Figure 5 that the NSE values seem to be good for all KMs that are built for different LHD algorithms, as they are very close to 1. This indicates that the KMs predictions that are built for different LHD algorithms are good approximations for the outputs of the SC function. However, we can see that the NSE values for all the KMs are quite similar. Therefore, to investigate the best performance of the different LHD algorithms, we calculated the mean of NSE values for each KM.

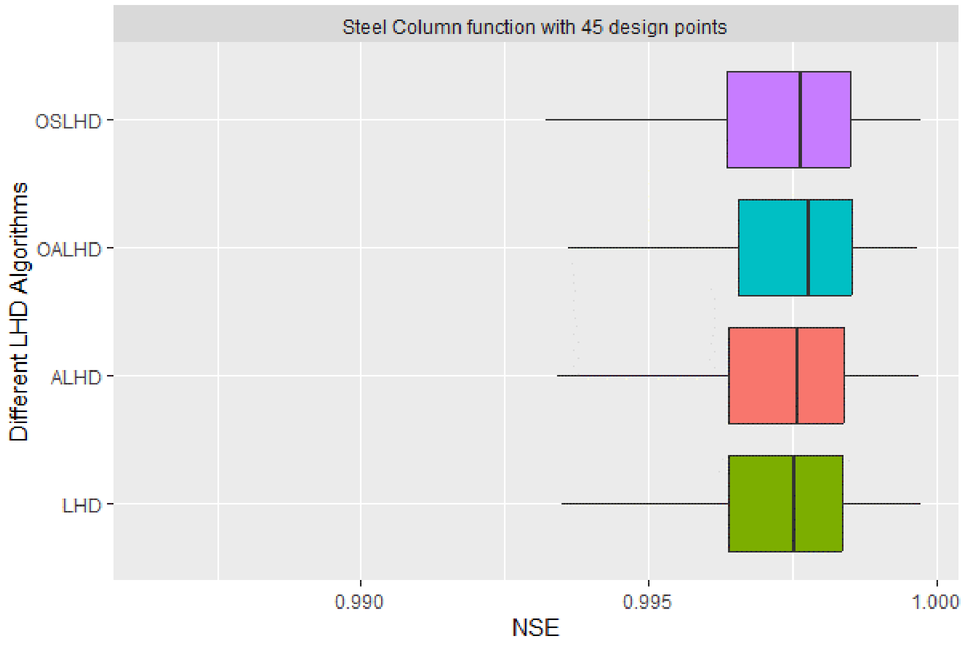

Figure 5. Box plots of the NSE values obtained by KMs of the SC function for different LHD algorithms with $n = 45$ training points.

Table 3 presents the mean of NSE values for KMs built using different LHD algorithms. We can notice that the OSLHD has the best NSE mean value but takes the longest CPU time.

Table 3. The NSE mean value obtained by KMs of the SC function using different LHD algorithms with $n = 45$ training points.

	LHD	ALHD	OALHD	OSLHD
NSE	0.9668	0.9479	0.9701	0.9810
CPU time (mins)	2.4	3.55	7.61	153

We applied the second measure that not only depends on the KM predictions and outputs of the SC function, but also takes into account the prediction uncertainty of KMs. We calculated the eigen errors (E), equation (14). Then, we plotted eigen errors against the index to compare the KMs performance based on different LHD algorithms. Figure 6 reviews the E values against the index obtained by KMs using different LHD algorithms with 45 training points and 18 test points. In Figure 6, the top left panel presents the E values obtained by KM using LHD. We see that most of the E values are small, with about 28% of the errors are seen outside the error interval, [-2, 2]. The top right panel presents the E obtained by KM using ALHD. It can be shown that about 39% large E values are seen in the plot. Therefore, the E plots of the KMs based on LHD and ALHD indicate a conflict between the KM prediction and the SC outputs.

The bottom left panel presents the E obtained by KM using OALHD. It can be seen that only about 5% error is seen outside the error interval, and the rest 95% of E values are inside the error interval. The bottom right panel presents the E values for the KM that is built using OSLHD. It can be shown that all of the E values are seen inside the error interval. This indicates that the predictions of the KMs based on OALHD and OSLHD are reasonable approximations for the SC outputs.

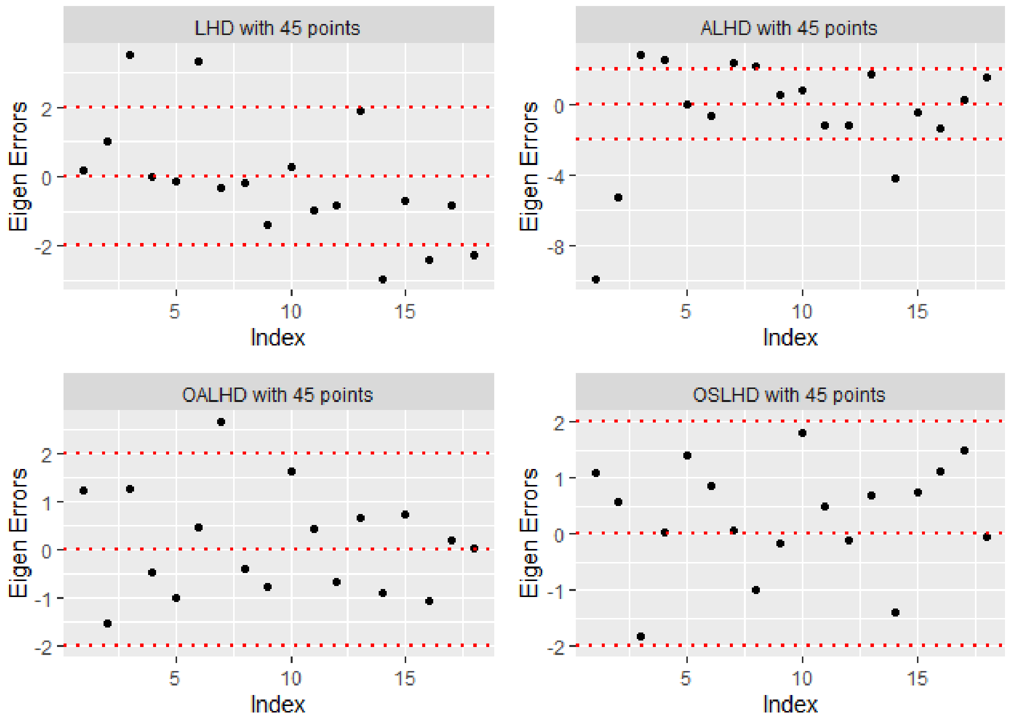

Figure 6. The E plots for the KMs of the SC function based on different LHD algorithms with with $n = 45$ training points.

We increased the training points to be $n = 10p = 90$ and the processes are repeated. In Figure 7, we show the box plots of NSE values for KMs using different LHD algorithms with 90 training points and 18 test points. We notice from Figure 7 that NSE values seem to be improved for all KMs that are built using different LHD algorithms, as they are now very close to 1. This confirms that the KMs predictions that are built for different LHD algorithms are good approximations for the SC outputs. However, NSE values of the KM that is built using OSLHD are the best. We also calculated the mean of NSE values for each KM.

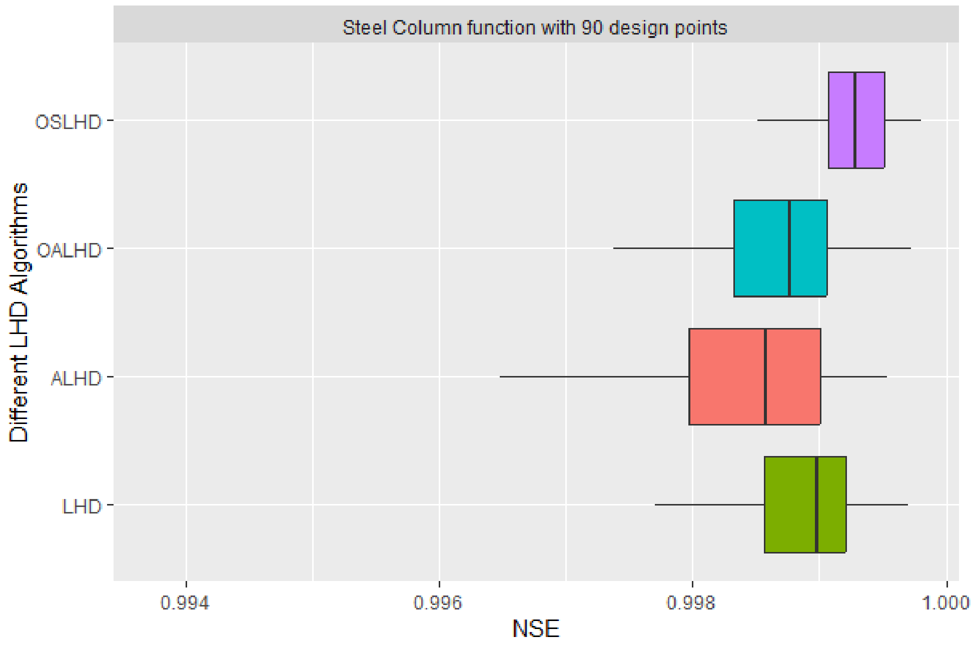

Figure 7. Box plots of NSE values for KMs of SC function using different LHD algorithms with $n = 90$ training points.

Table 4 presents the mean of the 1000 NSE for the KMs that were built using different LHD algorithms with 90 training points and 18 test points. We can also notice that the OSLHD also has the best NSE mean value but with the longest CPU time.

Table 4. The NSE mean values of KMs of the SC function using different LHD algorithms with $n = 90$ training points.

	LHD	ALHD	OALHD	OSLHD
NSE	0.9988	0.9983	0.99861	0.9992
CPU time (mins)	5.54	6.18	12.78	496

We also calculated the eigen errors (E), equation (13) of KMs, using different LHD algorithms with 90 training points and 18 test points. Figure 8 presents the E values against the index for KMs using different LHD algorithms with 90 training points and 18 test points. The top left panel presents the E values for the KM built using LHD. It can be seen that the values of E have been improved, and all of them are now small. The top right panel presents the E for the KM built using ALHD. It can be shown that the E values have also been improved, with only about 11% of the E errors seen outside the error interval.

The bottom left panel presents the E values for the KM built using OALHD. We notice that all of the E values are inside the error interval, and only about 5% of the errors are seen outside the error interval. The bottom right panel of Figure 8 presents the E values for the KM built using OSLHD. It can be shown that all of the E values are seen inside the error interval. Therefore, the E plots of the KMs based on ALHD and OALHD indicate a conflict between the KM prediction and the SC outputs. In contrast, the predictions of the KMs based on OSLHD are reasonable approximations for the SC outputs.

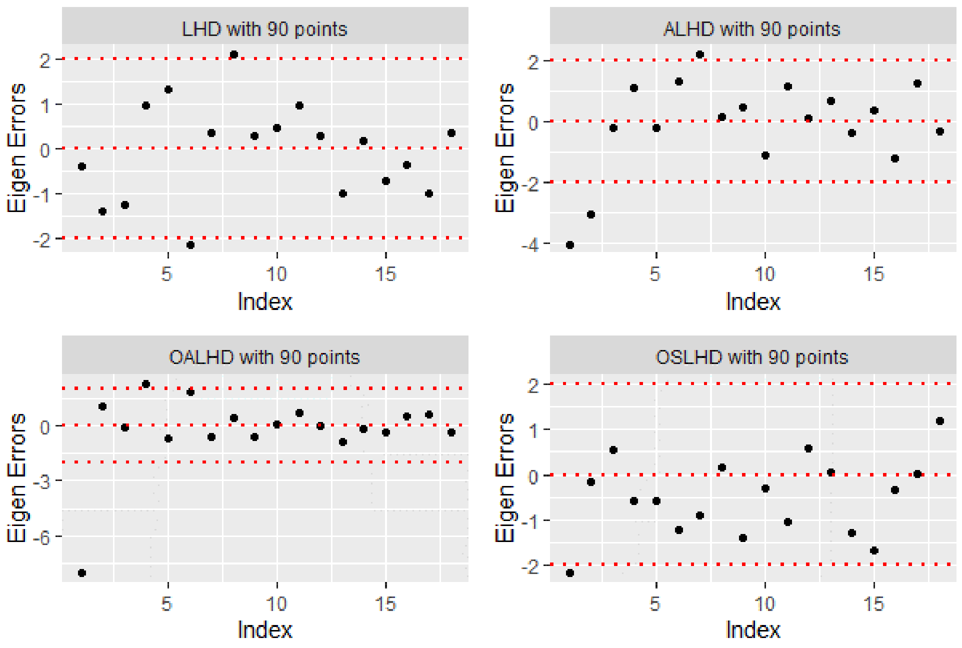

Figure 8. The plots of the E values for the KMs of the SC function using different LHD algorithms with $n = 90$ training points.

Based on the result of the suggested measures, the NSE and the eigen errors, we can conclude that the OSLHD has the best performance among all the other designs, as it has the smallest eigen errors and the best NSE values among all the other LHD algorithms.

6. Conclusion

In this work, the behavior of Latin hypercube design (LHD) and three improvement algorithms of LHD were investigated. This investigation was achieved by comparing the KMs predictions that are built using these different LHD algorithms. This comparison is conducted by several proposed measures, the NSE and the eigen errors, that examine the KM behavior. The results of the NSE measure that applied to KMs for the two computer codes show that the performances of the different LHD algorithms are quite similar. The results of the eigen errors and the NSE indicate that the Optimum Seeded Latin Hypercube Design performs the best. For the NSE value, we have observed that the KM based on the Optimum Seeded Latin Hypercube Design has the closest value to one among all KMs based on other LHD algorithms. Moreover, results of the eigen errors also confirm the superiority of the Optimum Seeded Latin Hypercube Design, where almost all eigen errors were in the interval [-2, 2] in contrast to KMs based on the rest of the LHD algorithms, where we have observed that several eigen errors were outside the interval [-2, 2]. A stationary covariance function has been used in building Universal Kriging Models, which is suitable for our examples. However, functions with irregular landscapes require advanced Kriging Models with maybe non-stationary covariance function, which deserves further investigation.

Acknowledgement

The authors are very grateful to the University of Mosul/ College of Education for Pure Science for its provided facilities, which helped to improve the quality of this work.

REFERENCES

1. M. J. Sasena, *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*, University of Michigan, 2002.
2. T. J. Santner, B. J. Williams, and W. I. Notz, *The design and analysis of computer experiments*, Springer, vol. 1, 2003.
3. K. T. Fang, R. Li, and A. Sudjianto, *Design and modeling for computer experiments*, Chapman and Hall/CRC, 2005.
4. B. Al-Khayyat, and S. Al-Dabbagh, *Using Fuzzy Logic to Construction of Expert Computer Model for Forecast of Compressive Strength of the Portland Cement*, Al-Rafidain Journal of Computer Sciences and Mathematics, vol. 2, no. 2, pp. 29–49, 2009.
5. M. D.McKay, R. J. Beckman, and W. J. Conover, *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*, Technometrics, vol. 42, no. 1, pp. 55–61, 2000.
6. J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, *Design and analysis of computer experiments*, Statistical science, vol. 4, no. 4, pp. 409–423, 1989.
7. D. Williams, I. S. Heng, J. Gair, J. A. Clark, B. and Khamesra, *Precessing numerical relativity waveform surrogate model for binary black holes: A Gaussian process regression approach*, Physical Review D, vol. 101, no. 6, 2020.
8. R. B. Gramacy, and H. K. H. Lee, *Bayesian treed Gaussian process models with an application to computer modeling*, Journal of the American Statistical Association, vol. 103 no. 483, pp. 1119–30, 2008.
9. Y. H. Al-Taweel, and N. Sadeek, *A comparison of different methods for building Bayesian kriging models*, Pakistan Journal of Statistics and Operation Research, pp. 73–82, 2020.
10. R. W. Al-Naser, and Y. Al-Taweel, *Using Kriging models for approximating computer models and quantifying their uncertainty*, In AIP Conference Proceedings, vol. 2845, no. 1, 2023.
11. VR. Joseph, *Space-filling designs for computer experiments: A review*, Quality Engineering, vol. 18, no. 1, pp. 28–35, 2016.
12. P. Z. Qian, *Sliced Latin hypercube designs*, Journal of the American Statistical Association, vol. 107, no. 497, pp. 393–399, 2012.
13. S. Ba, W. R. Myers, and W. A. Brenneman, *Optimal sliced Latin hypercube designs*, Technometrics, vol. 57, no. 4, pp. 479–487, 2015.
14. VR. Joseph, E. Gul, and S. Ba, *Maximum projection designs for computer experiments*, Biometrika, vol. 102, no. 2, pp. 371–380, 2015.
15. VR. Joseph, E. Gul, and S. Ba, *Designing computer experiments with multiple types of factors: The MaxPro approach*, Journal of Quality Technology, vol. 52, no. 4, pp. 343–354, 2020.
16. W. Zhou, J. F. Yang, and M. Q. Liu, *Optimal maximin L2-distance Latin hypercube designs*, Engineering Optimization, vol. 56, no. 2, pp. 179–198, 2020.
17. G. Li, J. Yang, Z. Wu, W. Zhang, P. Okolo, and D. Zhang, *A sequential optimal Latin hypercube design method using an efficient recursive permutation evolution algorithm*, Engineering Optimization, vol. 56, no. 2, pp. 179–198, 2024.
18. Y. Al-Taweel, *Uncertainty quantification of multivariate Gaussian process regression for approximating multivariate computer codes*, WMS Journal of Applied and Engineering Mathematics, 2024.
19. H. M. Saeid, and Y. H. Al-Taweel, *Bayesian Approach for Analyzing Computer Models using Gaussian Process Models*, Journal of Education and Science, vol. 30, no. 2, pp. 148–164, 2021.
20. S. A. Salih, and E. E. Aboudi, *Bayesian Approach for Analyzing Computer Models using Gaussian Process Models*, Iraqi Journal of Statistical Sciences, vol. 18, no. 2, pp. 51–64, 2021.
21. C. Helbert, D. Dupuy, and L. Carraro, *Assessment of uncertainty in computer experiments from Universal to Bayesian Kriging*, Applied Stochastic Models in Business and Industry, vol. 25, no. 2, pp. 99–113, 2009.
22. Y. H. Al-Taweel, and Z. Y. Algamal, *Almost Unbiased Ridge Estimator in the Inverse Gaussian Regression Model*, Electronic Journal of Applied Statistical Analysis, vol. 15, no. 3, 2022.
23. H. Dette, and A. Pepelyshev, *Generalized Latin hypercube design for computer experiments*, Technometrics, vol. 52, no. 4, pp. 421–429, 2010.
24. M. Stein, *Large sample properties of simulations using Latin hypercube sampling*, Technometrics, vol. 29, no. 2, pp. 143–151, 1987.
25. F. A. Viana, G. Venter, and V. Balabanov, *An algorithm for fast optimal Latin hypercube design of experiments*, International journal for numerical methods in engineering, vol. 82, no. 2, pp. 135–156, 2010.
26. H. Zhu, L. Liu, T. Long, L. and Peng, *A novel algorithm of maximin Latin hypercube design using successive local enumeration*, Engineering Optimization, vol. 44, no. 5, pp. 551–564, 2012.
27. X. Shang, T. Chao, P. Ma, and M. Yang, *An efficient local search-based genetic algorithm for constructing optimal Latin hypercube design*, Engineering Optimization, vol. 44, no. 5, pp. 551–564, 2012.
28. R. Carnell, *Package 'lhs'*, R package version 0.10, 2022.
29. A. M. Overstall, and D. C. Woods, *Multivariate emulation of computer simulators: model selection and diagnostics with application to a humanitarian relief model*, Journal of the Royal Statistical Society Series C: Applied Statistics, vol. 65, no. 4, pp. 483–505, 2016.
30. L. S. Bastos, and A. O'hagan, *Diagnostics for Gaussian process emulators*, Technometrics, vol. 51, no. 4, pp. 425–438, 2009.
31. O. Roustant, D. Ginsbourger, and Y. Deville, *DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization*, Journal of statistical softwar, vol. 51, pp. 1–55, 2012.
32. M. Eldred, C. Webster, and P. Constantine, *Evaluation of non-intrusive approaches for wiener-askey generalized polynomial chaos, AIAA-paper 2008-1892*, 10th AIAA Non-Deterministic Approaches Forum, Schaumburg, IL, 2008.