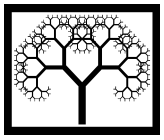
Paper 50



Proceedings of the First International Conference on
Soft Computing Technology in
Civil, Structural and Environmental Engineering,
B.H.V. Topping and Y. Tsompanakis, (Editors),
Civil-Comp Press, Stirlingshire, Scotland

Space-Filling Sequential Design Strategies for Adaptive Surrogate Modelling

K. Crombecq[1], I. Couckuyt[2], D. Gorissen[2] and T. Dhaene[1]
[1] Department of Mathematics and Computer Science
University of Antwerp, Belgium
[2] Interdisciplinary Institute for Broadband Technology (IBBT),
Department of Information Technology (INTEC), Ghent University, Belgium

Abstract

Simulated computer experiments have become a viable cost-effective alternative for controlled real-life experiments. However, the simulation of complex systems with multiple input and output parameters can be a very time-consuming process. Many of these high-fidelity simulators need minutes, hours or even days to perform one simulation. The goal of global surrogate modeling is to create a model that mimics the original simulator, but can be computed much faster. Sequential design strategies are often used to reduce the number of sample evaluations needed to achieve the desired accuracy. In this paper, the authors present a comparison and analysis of different space-filling sequential design methods. The results are compared to traditional one-shot Latin hypercube designs.

Keywords: surrogate modelling, space-filling, sequential design, active learning, Latin hypercube, experimental design, Kriging, Voronoi tessellation, Delaunay triangulation.

1 Introduction

For many modern engineering problems, accurate high fidelity simulations are often used instead of controlled real-life experiments, in order to reduce the overall time, cost and/or risk. These simulations are used by the engineer to understand and interpret the behaviour of the system under study and to identify interesting regions in the design space. They are also used to understand the relationships between the different input parameters and how they affect the outputs.

However, the simulation of one single instance of a complex system with multiple inputs (also called factors or variables) and outputs (also called responses) can be a

very time-consuming process. For example, Ford Motor Company reported on a crash simulation for a full passenger car that takes 36 to 160 hours to compute [1]. Because of this long computational time, using this simulation directly is still impractical for engineers who want to explore, optimize or gain insight into the system.

In our approach, we assume the system under study is a black box, with little or no additional information available about its inner working except for the output it generates. This means that, without running simulations, nothing is known about the behaviour of the function, and no assumptions can be made about continuity or linearity or any other mathematical properties the system might have. A final assumption is that the simulator is deterministic, meaning that the same output is produced if the simulator is run twice with the same input values. This is not the same as saying that there is a complete absence of noise; indeed, noise may be introduced by the way the simulator models and discretises the real world. It only implies that, even if there is noise in the simulation outputs, the noise will be identical for two simulation runs with the same inputs.

The goal of *global* surrogate modelling is to find a function that mimics the original system, but can be computed much faster. This function is constructed by performing multiple simulations (called samples) at key points in the design space, analyzing the results, and selecting a model that approximates the samples and the overall system behavior quite well. This is illustrated in figure 1.

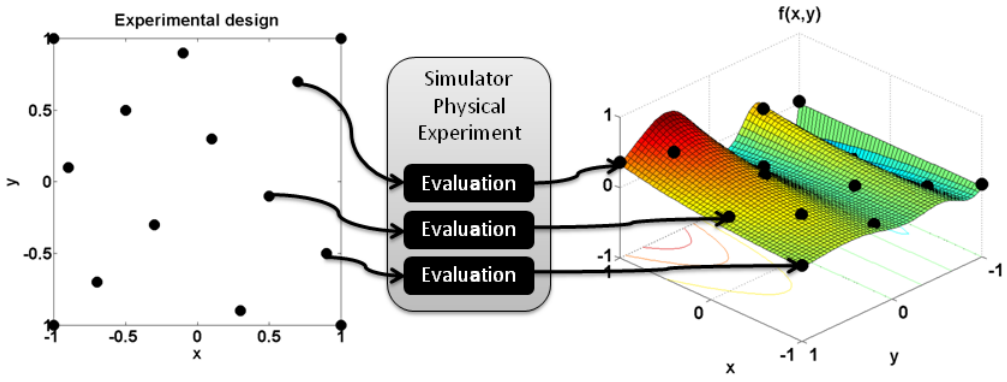


Figure 1: A set of data points is evaluated by a black box simulator, which outputs a response for every data point. An approximation model (surrogate model) is fit to the data points, with the goal of minimizing the approximation error on the entire domain.

Please note that *global* surrogate modelling differs from *local* surrogate modelling in the way the surrogate models are employed. In *local* surrogate modelling, local models are used to guide the optimization algorithm towards a global optimum. The local models are discarded afterwards. In *global* surrogate modelling, the goal is to create a model that approximates the behaviour of the simulator on the entire domain, so that the surrogate model can then be used as a full replacement for the original simulator, or can be used to explore the design space. Thus, the goal of global surrogate modelling is to overcome the long computational time of the simulator by providing a

fast but accurate approximation, based on a one-time upfront modelling effort. In this paper, we are only concerned with global surrogate modelling.

Mathematically, the simulator can be defined as an unknown function $f : \mathbb{R}^d \rightarrow \mathbb{C}$, mapping a vector of real inputs to a real or complex output. This function is assumed to be highly nonlinear and possibly even discontinuous. This unknown function has been sampled at a set of scattered data points $X = x_1, x_2, \ldots, x_n$, for which the function values $f(x_1), f(x_2), \ldots, f(x_n)$ are known. In order to approximate the function f, a function $\tilde{f} : \mathbb{R}^d \rightarrow \mathbb{C}$ is chosen from the (possibly) infinite function set of candidate approximation functions F.

The quality of this approximation depends entirely on the choice and exploration of the function space F and the data points X. Ideally, the function f itself would be in the search space F, in which case it is possible to achieve an exact approximation. However, this is rarely the case, due to the complexity of the underlying system. In practice, the function $\tilde{f}$ is chosen according to a search strategy through the space F, in order to find the function that most closely resembles the original function, based on some error metric for the data points X.

It is clear that the choice of the data points X (called the experimental design) is of paramount importance to the success of the surrogate modelling task. Intuitively, the data points must be spread over the domain $\mathbb{R}^d$ in such a way as to convey a maximum amount of information about the behaviour of f. This is a non-trivial task, since nothing is known about this function in advance.

In this paper, we present a comparison and analysis of different space-filling sequential design methods. Different methods are compared against each other on a set of examples, and the results are compared to the Latin hypercube, which is a popular one-shot experimental design technique. The advantages and disadvantages of each method are discussed, and conclusions are drawn as to which method is preferable.

2 Sequential Design

In traditional design of experiments (DOE), the experimental design X is chosen based only on information that is available before the first simulation. Such information can be the existence of noise, the relevance of the input variables, the measurement precision and so on. Then this design is fed to the simulator, which evaluates the selected data points, and a model is built using this data. This is essentially a one-shot approach, as all the data points are chosen at once and the modelling algorithm proceeds from there, without evaluating any additional samples later on.

In the deterministic setting of computer experiments, well-known DOE techniques such as replication, randomization and blocking lose their relevance [2]. This leaves space-filling designs, which try to cover the domain as equally as possible, as the only interesting option. The advantages of the classic space-filling methods are that they can be easily implemented and provide a good (and guaranteed) coverage of the domain. Examples of popular space-filling design are fractional designs [3], Latin

hypercubes [4] and orthogonal arrays [5].

Sequential design (which is also known as adaptive sampling [6] or active learning [7]) further improves on this approach by transforming the one-shot algorithm into an iterative process. Sequential design methods analyze data (models and samples) from previous iterations in order to select new samples in areas that are more difficult to approximate, resulting in a more efficient distribution of samples compared to traditional design of experiments.

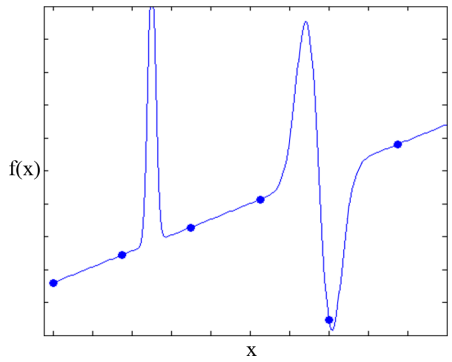
Because the simulator is assumed to be a black box, it is infeasible in practice to predict how large the experimental design must be to achieve a given accuracy. Sequential design strategies solve this problem by selecting samples in an iterative manner, while at the same time updating (retraining) the model and assessing the quality of the model. If the model reaches the desired accuracy, the sampling algorithm is halted, and no more samples are selected.

An essential consideration in sequential design is the trade-off between exploration and exploitation. Exploration is the act of exploring the domain in order to find key regions of the design space, such as discontinuities, steep slopes, optima, stable regions and so on, that have not been identified before. The goal is similar to that of a one-shot experimental design, in that exploration means filling up the domain as evenly as possible. Exploration does not involve the responses of the system, because the design space is defined by the inputs only.

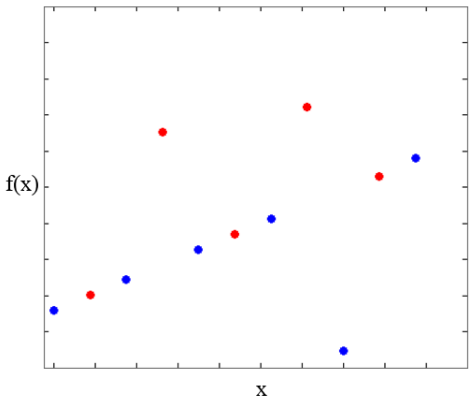
The main advantage of exploration-based sequential designs over one-shot experimental designs is that the amount of samples evaluated depends on the previous iterations of the algorithm. When one large experimental design is used, too many samples may have been evaluated to achieve the desired accuracy (oversampling) or too little samples may have been evaluated (undersampling), in which case one must completely restart the experiment or resolve to sequential methods to improve the initial design. Exploration-based sequential design methods will keep selecting new samples until the desired accuracy is found.

Exploitation is the other option. Instead of exploring the domain, data points are selected in regions which have already been identified as (potentially) interesting. For example, one might want to zoom in on optima, in order to make sure the surrogate model does not overshoot the optimum. Or one might also want to sample near possible discontinuities to verify that they are, in fact, discontinuous, and not just very steep slopes. Exploitation involves using the outputs of the previous function evaluations to guide the sampling process.

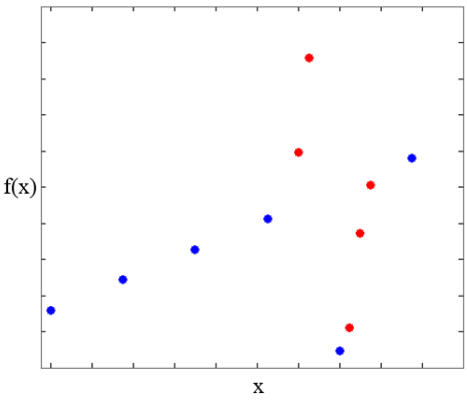
The trade-off between exploration and exploitation is illustrated in Figure 2. In this paper, we are only concerned with comparing purely exploration-based (or space-filling) sequential design strategies. The goal is to see how exploration-based sequential design strategies, which have the advantage of avoiding undersampling and oversampling, hold up against a popular, proven space-filling experimental design called the Latin hypercube, which has interesting mathematical properties.



(a) Initial set of samples



(b) Exploration



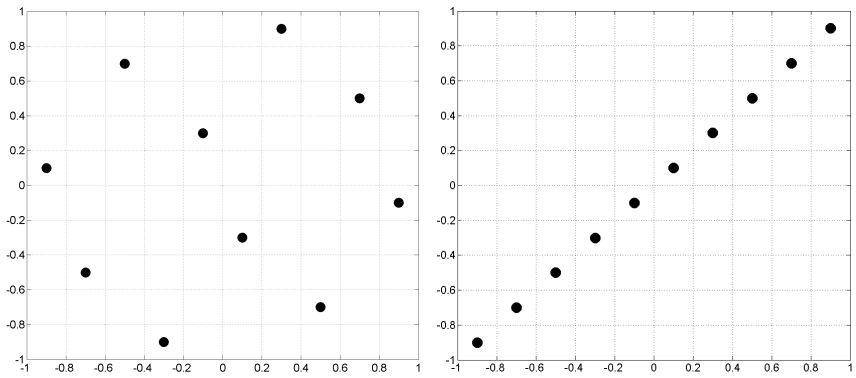
(c) Exploitation

Figure 2: This figure shows the trade-off between exploration and exploitation. In Figure 2(a), a function and an initial set of samples are visualized. The function is unknown, and from looking at the samples, the function seems to behave linearly, except for one sample to the right. As illustrated in Figure 2(b), exploration will explore the entire design space evenly, discovering new nonlinearities on the way. Exploitation, on the other hand, will focus on the area to the right because it seems to be the only nonlinear area, missing the additional nonlinearity to the left. This is depicted in Figure 2(c).

3 The Latin Hypercube

In this paper, we will compare the Latin hypercube against a set of space-filling sequential design strategies. Latin hypercubes are a very popular experimental design technique because of their well-understood mathematical properties, their ease of implementation and use and their speed. An experimental design is a Latin hypercube if, for every point in the initial design, there is at most one sample in every axis-aligned hyperplane. This property also means that, when the Latin hypercube is projected from d-dimensional space to $(d-1)$-dimensional space along one of the axes, no two points are ever projected onto the same location.

The Latin hypercube implementation used in this experiment is an optimized version which addresses well-known issues that a naive Latin hypercube implementation has. More specifically, not every Latin hypercube has nice space-filling properties; this is illustrated in Figure 3. The optimization algorithm used in the implementation is based on the one described in [8]. It optimizes using a space-filling criterion, resulting in a robust Latin hypercube implementation that considerably improves the space-filling properties of the Latin hypercube.



(a) Good Latin hypercube (b) Bad Latin hypercube

Figure 3: Two different Latin hypercubes. While 3(a) has nice space-filling properties, 3(b) has only points in the diagonal and neglects two corners of the design space completely.

4 Space-filling Sequential Design Strategies

A space-filling sampling strategy must accurately and efficiently identify the region of the design space that contains the lowest density of samples. This can be done in a many different ways, depending on the density measure used and on the allowed computational complexity of the algorithm. In this section, the different exploration-based sequential design strategies included in this study will be described and analyzed.

4.1 Random Sampling

As a base case, random sampling will be considered. A random sampling scheme just randomly selects samples in the design space, with no regards for previously evaluated samples. If enough samples are taken, random sampling will approximate a good space-filling design, while at the same time being the simplest and cheapest sampling scheme available. This makes random sampling actually a good choice if the number of allowed data points is sufficiently large. However, for a small sample set, large deviation from space-filling is to be expected, and the behaviour of this sampling scheme can be very erratic. Because sample evaluations are considered to be very expensive, we mostly operate on small sample sizes, which makes random sampling a very unreliable method.

4.2 Voronoi-based Sequential Design

4.2.1 Definition

A Voronoi tessellation (or Voronoi diagram) is an intuitive way to describe sampling density. Assume a discrete and pairwise distinct set of points $X \subset \mathbb{R}^d$ in Euclidian space (existing data points). For any point $x_i \in X$, the Voronoi cell $C_i \subset \mathbb{R}^d$ contains all the points that lie closer to x_i than any other point in X. The complete set of these polytopes $C_1, C_2, \ldots, C_n$ tesselate the whole space, and is called the Voronoi tessellation corresponding to the set X.

To define a Voronoi tessellation more formally, we adopt the notation from [9]. For two distinct points $x_i, x_j \in \mathbb{R}^d$, the *dominance* of x_i over x_j is defined as the subset of the plane being at least as close to x_i as to x_j. Formally,

$$dom(x_i, x_j) = \{x \in \mathbb{R}^d \,|\, \|x - p\| \leq \|x - q\|\}$$

.

$dom(x_i, x_j)$ is a closed half plane bounded by the perpendicular bisector of x_i and x_j. This bisector, which we will call the *separator* of x_i and x_j, separates all points of the plane closer to x_i than to x_j. Finally, the Voronoi cell C_i of x_i is the portion of the design space that lies in all dominances of x_i over the other data points in X:

$$C_i = \bigcap_{x_j \in X \setminus x_i} dom(x_i, x_j)$$

A Voronoi tessellation is shown in Figure 4. Note that near the center, the Voronoi cells are noticeably smaller than near the borders; a space-filling sequential design algorithm should select new data points in the larger (darker) Voronoi cells in order to achieve a more equal distribution of data. However, the points that lie closest to the border are colored black, because their volume is infinitely large: their Voronoi cells reach to infinity in their respective directions. This is a basic property of Voronoi tessellations.

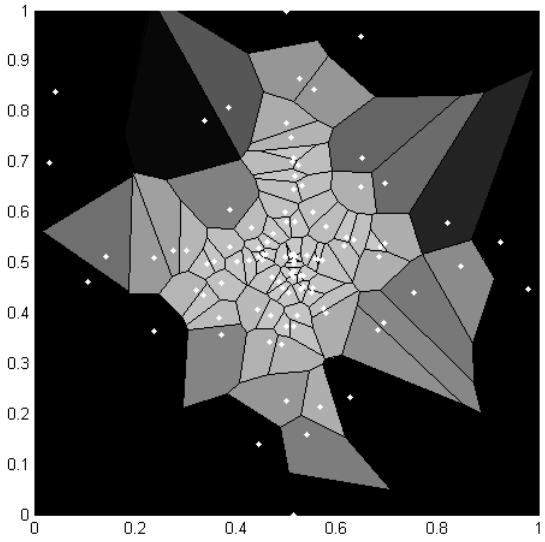


Figure 4: A set of data points and their Voronoi cells. Larger Voronoi cells have a darker colour. The data points are drawn as white dots. Unbounded Voronoi cells are coloured black.

4.2.2 Implementation

Computing a Voronoi tessellation is non-trivial, and is usually done by calculating the dual Delaunay triangulation, from which the Voronoi tessellation can be computed in O(n) time [9]. However, this still does not give us the volume of each Voronoi cell, which requires another computationally intensive step. In order to calculate the volume of each Voronoi cell, the unbounded Voronoi cells near the border of the domain must first be bounded. After this, the volume of each cell can be computed and used as a measure of density.

Figure 5 illustrates the calculation time for a Voronoi diagram as a function of the number of samples and the dimension of the input space. This plot was made using the Qhull library [10], which is based on the Quickhull algorithm described in [11]. It is clear that the calculation of a Voronoi diagram scales terribly with the dimensionality of the problem. For high-dimensional problems (in practice, any problem with more than 6 dimensions), the direct computation of the Voronoi tessellation is not an option.

Fortunately, we don't need the complete computation of the Voronoi cells for our purposes: an estimation of the volume of the cells is enough. Thus, instead of solving the problem exactly by calculating the Voronoi tessellation and the volume, a Monte Carlo approach was used. In order to estimate the volume of each Voronoi cell, a large number of random points are generated in the domain. For each random point, the minimum distance from all the data points is computed, and the random point is assigned to the data point which is the closest. If enough random points are generated like this, the algorithm produces an estimation of the (relative) size of each Voronoi

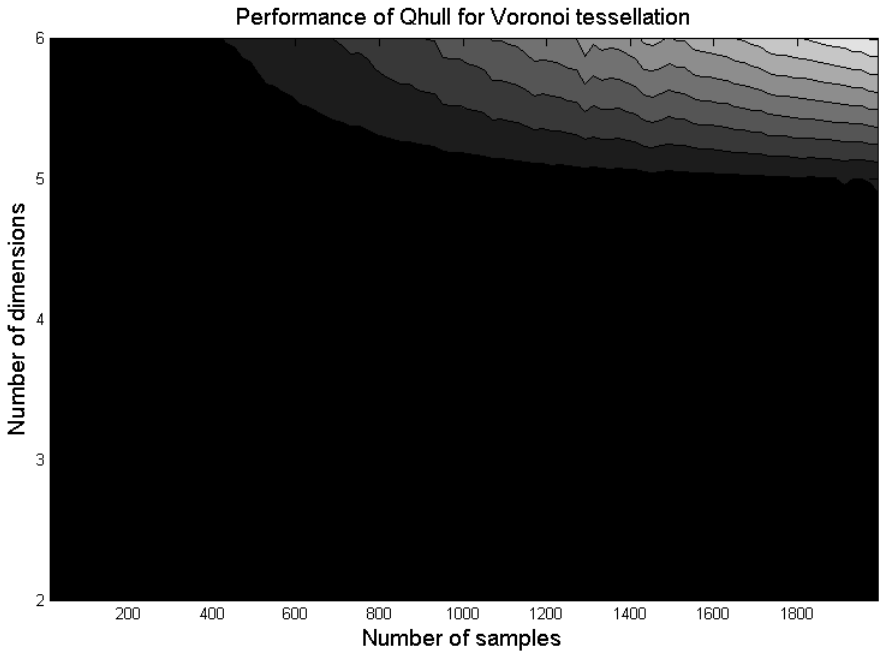


Figure 5: A performance plot of the Qhull package as a function of the number of samples and the dimensionality of the problem. Brighter colours mean a higher computation time. It is clear that the algorithm scales well with the number of samples, but poorly with the dimensionality of the problem. In practice, the algorithm is infeasible for problems with more than six dimensions.

cell. This is described more formally in Algorithm 1. If m new samples have to be selected by the sequential design strategy, the m cells with the largest estimated volume are picked, and a new sample is chosen at a location in the cell as far away as possible from any existing samples.

4.2.3 Error analysis

The number of randomly generated samples mainly determines the accuracy of the estimation, as a higher number of samples clearly results in a better volume estimation, at the cost of a higher computational time. In order to find the ideal number, we have performed a large set of tests with different numbers, ranging from 10 random samples to 500 000 random samples. For each of these numbers, the (bounded) Voronoi volume estimation was calculated for a dataset of 100 random points in three-dimensional space.

To calculate the error on the volume estimation, we used the BEEQ metric proposed in [12]. This is a relative error metric is defined as:

$$BEEQ = exp\left(\frac{1}{n}\sum_{i=1}^{n} ln\left(\frac{\left\|V_i - \tilde{V}_i\right\|}{\left\|V_i - \bar{V}_i\right\|}\right)\right)$$

Algorithm 1 Estimating the Voronoi cell size. X is the set of data points that have to be ranked according to their respective Voronoi cell size.

$S \leftarrow |X| \times 100$ random points in the domain
$V \leftarrow [0, 0, \ldots, 0]$
for all $s \in S$ **do**
  $d \leftarrow \infty$
  for all $x \in X$ **do**
    if $\|x - s\| < d$ **then**
      $x_{closest} \leftarrow x$
      $d \leftarrow \|x - s\|$
    end if
  end for
  $V[x_{closest}] \leftarrow V[x_{closest}] + (1/|S|)$
end for

where V_i is the real volume of the i-th Voronoi cell, $\tilde{V}_i$ is the estimated volume of the i-th Voronoi cell and $\bar{V}_i$ is the average real Voronoi cell size. The BEEQ error was computed for each volume estimation and the exact volume. Each case was repeated 10 times, and the average error was taken over all experiments. The result is shown in Figure 6.

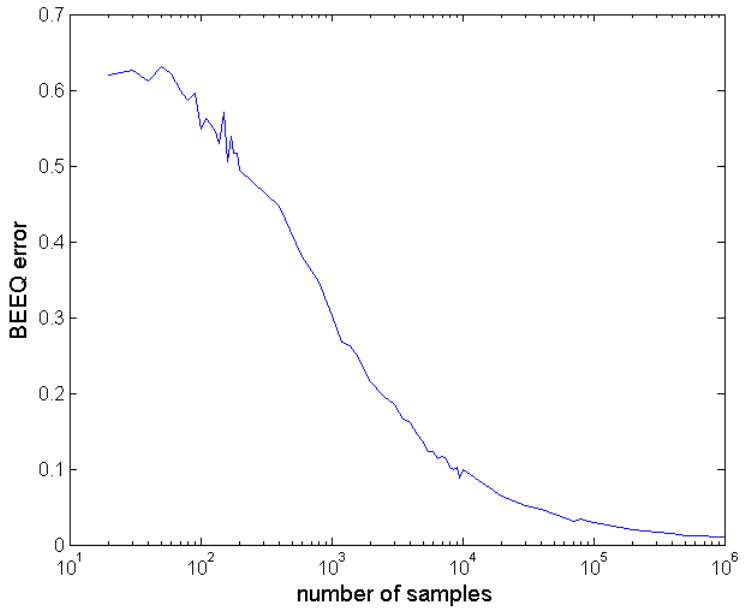


Figure 6: A benchmark of the accuracy of a Monte Carlo Voronoi approximation.

It is clear that the error drops considerably as the number of random samples grows. However, the rate at which the estimation improves slows down as the number of samples is increased. It appears you only need 10 000 random samples, or 100 samples

per data point, to achieve an error below 0.1. This is an acceptable error, since we only need a rough estimation to make a ranking between Voronoi cells based on their volume. By increasing the number of random samples to 300 per data point, an error of 0.05 can be reached. However, by further increasing the total number of samples to 1 000 000 (10 000 per data point), the error is only lowered to 0.01. Thus, it is not worth selecting more than about 300 samples, because the gain in accuracy is negligible. Furthermore, additional experiments have shown that the results are similar for higher dimensions.

We conclude that, in order to get an acceptable Voronoi volume estimation, we need at least 100 random samples per data point, but not much more. Because acquiring data points (running a simulation) is computationally expensive, we typically work with relatively small datasets, and thus 100 random samples per data point is a reasonable number. An additional advantage of the Monte Carlo approach is that it works for high-dimensional problems, while computing the Voronoi tessellation and volume exactly turns out to be infeasible for more than 6 dimensions.

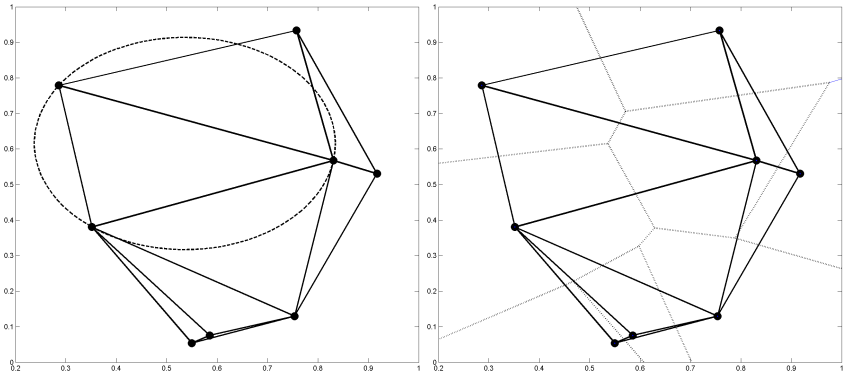
4.3 Delaunay-based Sequential Design

A Delaunay triangulation is a triangulation of a set of points X such that no point in X lies inside the circum-hypersphere of any n-simplex in the triangulation [13]. In two dimensions, this reduces to the circumcircle of each triangle. The Delaunay triangulation is the dual of the Voronoi diagram described in Section 4.2. One can be computed exactly from the other. The dual Voronoi tessellation of a given Delaunay triangulation can be computed by connecting the edges of the centers of the circum-hyperspheres of all the simplexes. The Delaunay triangulation of a given Voronoi tessellation can be computed by adding an edge for every two points in X for which the Voronoi cells share an edge. This is illustrated in Figure 7.

Since calculating the Voronoi tessellation is performed in Matlab by first calculating the Delaunay triangulation and then taking its dual, both share similar problems of efficiency in high dimensional spaces. Therefore, performance analysis performed in Section 4.2.2 can just as well be applied to a Delaunay triangulation, with the difference that it is much more difficult to approximate a Delaunay triangulation as it is to approximate a Voronoi tessellation. Consequently, this sampling strategy is only feasible for problems with less than 6 dimensions, or problems with a small amount of samples, because it uses the Qhull implementation and not an approximation as is the case for Voronoi-based sequential design.

Once the delaunay triangulation is computed, the volume is calculated for each simplex $(v_1, v_2, \ldots, v_{d+1})$ using the following formula:

$$V = \frac{1}{d!} \left| \begin{array}{cccc} v_2 - v_1 & v_3 - v_1 & \ldots & v_{d+1} - v_1 \end{array} \right| \quad \text{where} \quad v_i = \begin{bmatrix} v_i^1 \\ v_i^2 \\ \vdots \\ v_i^d \end{bmatrix} \tag{1}$$



(a) A Delaunay triangunation.

(b) A Delaunay triangulation with its dual Voronoi tessellation.

Figure 7: A Delaunay triangulation of a set of two-dimensional points. On the left side, the triangulation is displayed along with one circumcircle. Note that no points lie within the circumcircle of any triangle of the triangulation. On the right side, the Delaunay triangulation is plotted along with its dual Voronoi tessellation.

The simplices are ranked according to their volume. If m samples need to be selected, m simplices with the largest volume are selected. For each of their simplices, the algorithm calculates the centers of gravity (not to be confused with the center of the circum-hypersphere). One new sample is submitted at each center of gravity.

5 Experimental Setup

In the following sections, we will compare the different sequential design strategies to Latin hypercubes on two different test cases. First, all the methods will be compared on the Ackley's Path function, which is a well-known multimodal test function. Secondly, the methods will be compared on a problem from electronics.

In order to compare the different sequential design methods, all strategies were implemented in the SUrrogate MOdeling (*SUMO*) research platform [14]. This Matlab toolbox, designed for adaptive surrogate modeling and sampling, has excellent extensibility, making it possible for the user to add, customize and replace any component of the modeling process. It also has a wide variety of built-in test functions and test cases, as well as support for many different model types. Because of this, SUMO was the ideal choice for conducting this experiment.

The work-flow of SUMO for a typical surrogate modelling task with sequential design is illustrated in Figure 8. First, a set of initial samples are generated and evaluated. Then a set of models is built, and the accuracy of these models is estimated. Each type of model has several parameters which can be modified, such as degrees of freedom for rational models, number and size of hidden layers in neural networks, smoothness for RBF models, and so on. These parameters are adjusted using a optimization method, and more models are generated until no further improvement can be made

by changing the model parameters. If the desired accuracy has not yet been reached, a call is made to the sequential design strategy, which generates a set of new sample locations to be evaluated, and the algorithm starts all over again. For more information on the different components of the SUMO Toolbox, please refer to [14, 15].

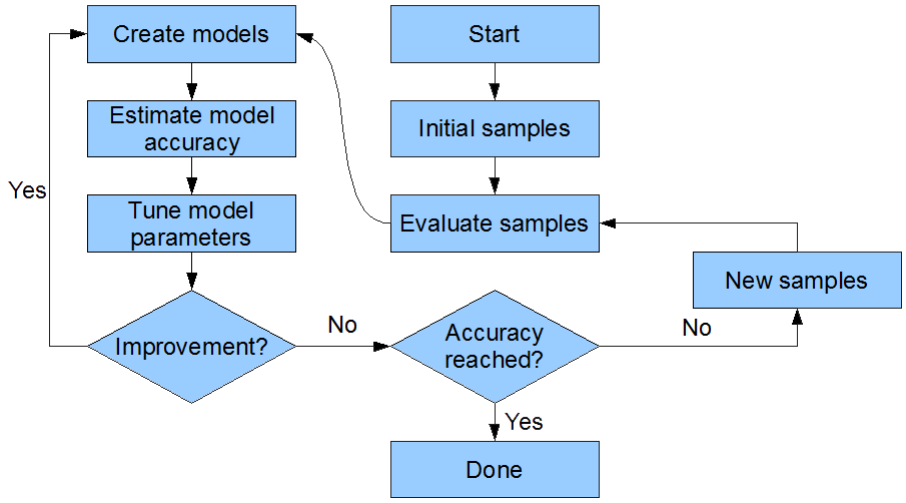


Figure 8: Flow-chart of the SUMO toolbox.

All of the problems in this paper were modelled using Kriging, which is a popular and powerful surrogate modelling technique which originates from geostatistics, and was introduced as part of a modelling technique called Design And Analysis of Computer Experiments (DACE) by [2]. For more information on Kriging and the implementation used in the SUMO Toolbox, please refer to [16].

The accuracy of the model is measured by comparing the model against a very dense, pre-evaluated test set. Thus, the error is a very accurate estimate of the true prediction error of the model. Each run will be terminated when the average euclidean error reaches the desired level. The average euclidean error is defined as:

$$AEE = \frac{1}{n}\sum_{i}^{n}\sqrt{(f(x_i) - \tilde{f}(x_i))^2} \tag{2}$$

where x_i a sample in the dense test set, $f(x_i)$ is the true value at point x_i and $\tilde{f}(x_i)$ the model estimation. At the end of each run, the number of samples required to reach this accuracy will be recorded. To eliminate noise caused by random factors in the SUMO toolbox (such as randomization in the model parameter optimization algorithm), the configuration for each sampling strategy will be run 10 times, and the average will be used for the results.

In order to compare the sequential design strategies against the Latin hypercube, which is inherently non-sequential, we generated Latin hypercubes of size 100, 110, 120 and so on, up to 300. Models were trained for each of these Latin hypercubes, and

the error was calculated as described in the previous paragraphs. The smallest Latin hypercube which, averaged over 10 runs, manages to reach the desired accuracy, is considered the smallest number of samples needed to achieve the desired accuracy using Latin hypercubes. This method allows us to compare the evolution of the accuracy of the Latin hypercube to that of the sequential design strategies. Please note that all the algorithms used in this paper are available in the distribution of the SUMO Toolbox (found at `http://sumo.intec.ugent.be`) and therefore, the results of this experiment can be reproduced.

6 Ackley's Path

6.1 Problem Description

The first test case is Ackley's Path, which is defined as:

$$y = -ae^{-b\sqrt{\frac{(2x_1)^2+(2x_2)^2}{2}}} - e^{\frac{cos(2cx_1)+cos(2cx_2)}{2}} + ae \tag{3}$$

where $a = 20$, $b = 0.2$ and $c = 2\pi$. The behaviour of Ackley's Path on the domain $[-1,1]$ is shown in Figure 9. There are many local optima scattered systematically throughout the design space, but there is only one global optimum at the origin. Because of this property, Ackley is a popular test function for optimization problems. In this case, however, we are not concerned with finding the optima; the goal is to model the Ackley function accurately on the entire domain.

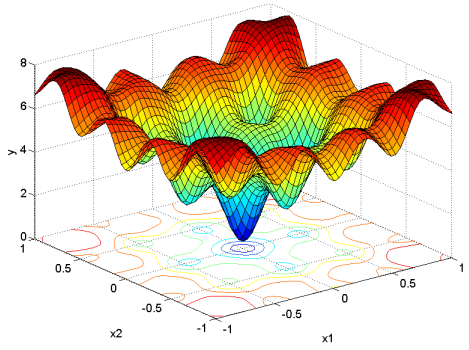


Figure 9: Ackley's Path function.

6.2 Results

The results of the experiment are depicted in Table 1. The best results are achieved using the Delaunay space-filling sequential design strategy, closely followed by Voronoi. The fact that Delaunay performs slightly better can be explained by the fact that Delaunay uses the exact Delaunay triangulation, while Voronoi uses an approximation of the

Voronoi cell size, which may result in suboptimal sample selection. However, Voronoi is usable in dimensions higher than 6 and for large sample sizes, and is considerably faster than Delaunay all across the board. Since the difference between Delaunay and Voronoi is so small, it might be advisable to choose Voronoi over Delaunay.

Table 1: Summary of the test results of modelling the Ackley function with different sampling strategies. The average number of samples required to reach an average euclidean error of 0.1 are shown for each sampling strategy. For the Latin hypercube, the smallest Latin hypercube size that on average over 10 runs reaches the target is shown. Since these Latin hypercubes all have the same number of samples, the variance is zero.

Sampling strategy	Average	Variance
Latin hypercube	180	0
Voronoi	158	7.4
Delaunay	**155**	7.9
Random	226	12

Surprisingly, Latin hypercubes, which are widely used, perform much worse than both Delaunay and Voronoi, managing only to achieve an average error of 0.1 for a Latin hypercube of size 180. However, this is still considerably better than random sampling, which needs 226 samples for the same accuracy. In order to better understand the problems with Latin hypercubes, Figure 10 compares the sample distribution of a run with Voronoi sampling against a 160-point Latin hypercube. It is clear that Voronoi provides a much better uniform sampling of the design space than the Latin hypercube, which leaves some noticeable gaps, for example near the top right.

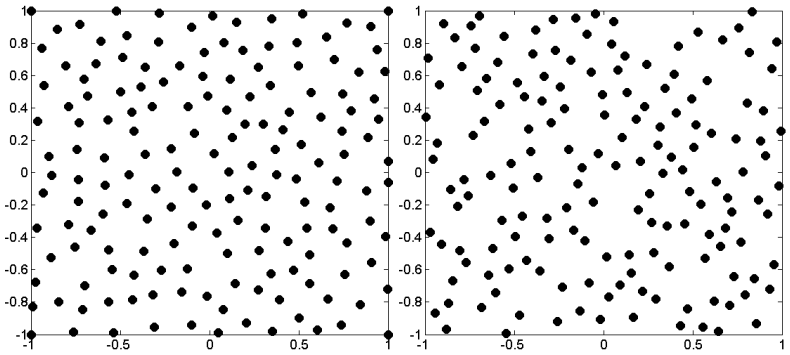
This can be explained by the fact that generating optimally space-filling Latin hypercubes is a very time-intensive task, and that suboptimal designs often leave large gaps in the design space. This is demonstrated in [17], where generating an optimal 4-dimensional 26-point Latin hypercube took over 3 hours.

7 Electrical Low-Noise Amplifier

7.1 Problem Description

The second test case is a real world problem from electronics: a narrowband Low Noise Amplifier (LNA), which is a simple RF circuit [18]. An LNA is the typical first stage of a receiver, having the main function of providing the gain needed to suppress the noise of subsequent stages, such as a mixer. In addition it has to give negligible distortion to the signal while adding as little noise as possible itself.

The performance figures of an LNA (gain, input impedance, noise figure and power consumption) can be determined by means of computer simulations where the under-



(a) Voronoi sampling (b) 160-point Latin hypercube

Figure 10: A comparison of the sample distribution of Voronoi sampling against Latin hypercube sampling.

lying physical behavior is accurately taken into account. Each simulation typically requires a couple of minutes, too long for a circuit designer to work with efficiently. Instead a designer could use a very accurate surrogate model (based on circuit simulations) to quickly explore how the performance figures of the LNA scale with key circuit-design parameters, such as the dimensions of transistors, passive components, signal properties and bias conditions.

In this experiment, in order to keep the computation times manageable, the expensive simulations will be replaced by a first-order analytic model, which is a direct implementation of the formulas described below. Initial manual tests indicate that results obtained from this simplified model are also applicable to the physics-based simulator. The output parameter

The two input parameters used in this experiment are the inductances L_{sn} and the MOSFET width W_n. The output is the approximate input noise-current ($\sqrt{\overline{i_{in}^2}}$), which is defined by equations (4) to (11). The behaviour of the input noise-current is shown in Figure 11. Note that the response is very linear in most of the design space, except for one very tall ridge where $W = 0$.

The parameters W_n and L_{sn} are used as follows:

$$W = 100 \cdot 10^{-6} \cdot 10^{W_n} \text{ m},$$
$$L_s = 0.1 \cdot 10^{-9} \cdot 10^{L_{sn}} \text{ H}.$$

Finally, the remaining parameters are set to fixed, typical values:

$$L_m = 10^{-9} \text{ H},$$
$$f = 13 \cdot 10^{9} \text{ Hz},$$
$$L = 82.5 \cdot 10^{-9} \text{ m},$$
$$V_{GT} = 0.325 \text{ V}.$$

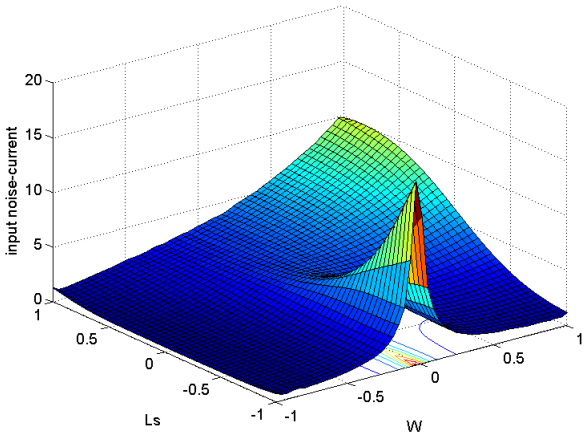


Figure 11: The input noise-current of a low-noise amplifier as a function of the inductance L_s and the MOSFET width W.

7.2 Results

The results are summarized in Table 2. Several interesting observations can be made.

Firstly, Voronoi-based sampling now performs better than Delaunay, which still remains the second best strategy. Even though Delaunay-based sampling uses the exact Delaunay triangulation instead of an approximation, it suffers from some inherent problems of Delaunay triangulations. Delaunay triangulations tend to create very long, narrow triangles near the edges of the design space, which results in new samples being selected close to other samples, because the centers of gravity of long triangles lie close to two of the three corner points. Hence, Delaunay-based sampling does not have good space-filling properties near the edges, and in this case, this is where most of the interesting behaviour is located. Voronoi-based sampling, on the other hand, does not suffer from this problem, and thus produces better results in this case, where it is of critical importance for the accuracy of the model that the ridge at $W = 0$ is sampled sufficiently.

Secondly, Latin hypercubes perform considerably worse than the sequential design strategies, but still better than random sampling. These results are in line with the results obtained from the previous experiment. Again, it appears that Latin hypercubes, though optimized towards space filling, leave relatively large gaps in the design space. If this gap happens to lie on or near the ridge, the accuracy suffers terribly. In the case of a sequential design strategy, more samples would be taken until the accuracy is improved, but with one-shot designs such as the Latin hypercube, it is all or nothing.

8 Conclusions and Future Work

In this paper, we have compared several sequential design strategies to the highly popular Latin hypercube sampling on two completely different test cases, and concluded

$$\omega = 2\pi f \tag{4}$$

$$g_m = 1 \cdot 10^{-4} \frac{W}{L} V_{GT} \text{ AV}^{-1} \tag{5}$$

$$C_{GS} = 0.01 \cdot WL \text{ F} \tag{6}$$

$$f_{gs,in} = \frac{1 + j\omega L_s g_m}{1 - \omega^2 C_{gs}(L_s + L_m) + j\omega L_s g_m} \tag{7}$$

$$f_{ds,in} = \frac{\omega^2 C_{gs}(L_s + L_m)}{1 - \omega^2 C_{gs}(L_s + L_m) + j\omega L_s g_m} \tag{8}$$

$$\overline{i_{gs}^2} = 2 \cdot 10^{-3} \frac{W}{L} \text{ pA}^2\text{Hz}^{-1} \tag{9}$$

$$\overline{i_{ds}^2} = 0.5 \frac{W}{L} \text{ pA}^2\text{Hz}^{-1} \tag{10}$$

$$\sqrt{\overline{i_{in}^2}} = \sqrt{|f_{gs,in}|^2 \cdot \overline{i_{gs}^2} + |f_{ds,in}|^2 \cdot \overline{i_{ds}^2} - 2 \cdot \text{Im}(0.4 f_{gs,in} f_{ds,in}^*) \sqrt{\overline{i_{gs}^2} \cdot \overline{i_{ds}^2}}} \tag{11}$$

that sequential schemes have several big advantages over classical Latin hypercubes. Firstly, they are more flexible, avoiding undersampling and oversampling by selecting new samples one by one and stopping when the desired accuracy is met. When using a one-shot experimental design, the engineer has to guess the required number of samples in advance, and if the desired accuracy is not met, must resort to sequential design strategies to generate more samples.

Secondly, creating an optimally space-filling Latin hypercube is a very difficult task, and has a large computational cost because of the high dimensionality of the search space. Suboptimal Latin hypercubes often leave large gaps in the design space, which substantially reduces the accuracy of the model. Both Delaunay-based and

Table 2: Summary of the test results of modelling the input noise-current of an LNA with different sampling strategies. The average number of samples required to reach an average euclidean error of 0.05 are shown for each sampling strategy. For the Latin hypercube, the smallest Latin hypercube size that on average over 10 runs reaches the target is shown.

Sampling strategy	Average	Variance
Latin hypercube	225	0
Voronoi	**176**	30
Delaunay	199	32
Random	256	75

Voronoi-based sequential design strategies produce much better space-filling designs and require less time to compute than an optimized Latin hypercube.

In order to determine which space-filling sequential design strategy is the best choice, the two sequential design methods in this paper were compared against each other. Voronoi-based sampling uses an approximation of the Voronoi tessellation, making it a much faster alternative than Delaunay-based sampling, which requires the computation of the Delaunay triangulation. One additional disadvantage of Delaunay-based sampling is that it does not have good space-filling properties near the edges of the design space, due to the nature of the triangulation. This may have a negative impact on the accuracy of the model. Thus, Voronoi-based sampling should in most cases be prefered over Delaunay-based sampling and self-generated Latin hypercubes.

In future work, sequential design methods will be compared against pre-calculated Latin hypercube designs, which have been optimized extensively. Large databases of good space-filling Latin hypercube designs exist, and instead of generating and optimizing a Latin hypercube as part of the SUMO Toolbox, the design could be retrieved from one of these databases. A considerable improvement is expected when using these extremely optimized Latin hypercubes, and they are expected to perform on the same level as Voronoi or Delaunay-based sequential design methods. However, they still have the disadvantage that the engineer has to guess the number of required samples in advance, which may lead to oversampling or undersampling.

Acknowledgments

This research was supported by FWO (Flemish Fund for Scientific Research). The authors would like to thank Jeroen Croon from the NXP-TSMC Research Centre, Eindhoven, Netherlands for making the LNA code available.

References

[1] D. Gorissen, K. Crombecq, W. Hendrickx, T. Dhaene, "Adaptive Distributed Metamodeling", *High Performance Computing for Computational Science - VECPAR 2006*, 4395: 579–588, 2007.

[2] J. Sacks, W.J. Welch, T.J. Mitchell, H.P. Wynn, "Design and analysis of computer experiments", *Statist. Sci.*, 1989.

[3] T.W. Simpson, J. Peplinski, P.N. Koch, J.K. Allen, "Metamodels for Computer-Based Engineering Design: Survey and Recommendations", *Engineering with Computers*, 2001.

[4] T.W. Simpson, D.K.J. Lin, W. Chen, "Sampling Strategies for Computer Experiments: Design and Analysis", *International Journal of Reliability and Applications*, 2(3): 209–240, 2001.

[5] K.T. Fang, "Experimental Design By Uniform Distribution", *Acta Mathematice Applicatae Sinica*, 1980.

[6] R. Lehmensiek, P. Meyer, M. Müller, "Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits", *International Journal of RF and Microwave Computer-Aided Engineering*, 12(4): 332–340, 2002.

[7] M. Sugiyama, "Active Learning in Approximately Linear Regression Based on Conditional Expectation of Generalization Error", *Journal of Machine Learning Research*, 7: 141–166, 2006.

[8] V.R. Joseph, Y. Hung, "Orthogonal-Maximin Latin Hypercube Designs", *Statistica Sinica*, 18: 171–186, 2008.

[9] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure", *ACM Comput. Surv.*, 23(3): 345–405, 1991.

[10] "Qhull", `http://www.qhull.org`.

[11] C.B. Barber, D.P. Dobkin, H. Huhdanpaa, "The quickhull algorithm for convex hulls", *ACM Transactions on Mathemathical Software*, 22(4): 469–483, 1996.

[12] X.R. Li, Z. Zhao, "Evaluation of estimation algorithms part I: incomprehensive measures of performance", *IEEE Transactions on Aerospace and Electronic Systems*, 42(4): 1340–1358, 2006.

[13] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, 2008.

[14] D. Gorissen, L.D. Tommasi, K. Crombecq, T. Dhaene, "Sequential Modeling of a Low Noise Amplifier with Neural Networks and Active Learning", *Neural Computation & Applications, accepted*, 2008.

[15] D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene, "Automatic Approximation of Expensive Functions with Active Learning", in *Foundation on Computational Intelligence, Learning and Approximation*. Springer Verlag, 2008.

[16] S.N. Lophaven, H.B. Nielsen, J. Søndergaard, "DACE: A MATLAB Kriging Toolbox", Technical report, Technical University of Denmark, 2002.

[17] K.Q. Ye, W. Li, A. Sudjianto, "Algorithmic Construction of Optimal Symmetric Latin Hypercube Designs", *Journal of Statistical Planning and Inferences*, 90: 145–159, 2000.

[18] T.H. Lee, *The Design of CMOS Radio-Frequency Integrated Circuits 2nd ed.*, Cambridge University Press, 2004.