Universiteit Antwerpen

UNIVERSITEIT GENT

Surrogate Modelling of Computer Experiments with Sequential Experimental Design

Proefschrift voorgelegd op 1 augustus 2011 met het oog op het behalen van de graad van Doctor in de Wetenschappen (informatica) aan de Universiteit Antwerpen en Doctor in de Ingenieurswetenschappen (computerwetenschappen) aan de Universiteit Gent

PROMOTOREN:
prof. dr. Tom Dhaene
prof. dr. Jan Broeckhove

Karel Crombecq

FONDS VOOR WETENSCHAPPELIJK ONDERZOEK VLAANDEREN

COMPUTATIONAL MODELING AND PROGRAMMING

INTERNET BASED COMMUNICATION NETWORKS AND SERVICES

Summary

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.

The simulation of one single instance of a complex system with multiple inputs and outputs 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 [36]. 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.

The goal of *global* surrogate modelling is to find an approximation function that mimics the behaviour of the original system, but can be evaluated 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 surrogate model that approximates the data and the overall system behaviour quite well. This surrogate model can then be used as a full replacement for the original simulator.

It is clear that the choice of the data points (the experimental design) is of paramount importance to the success of the surrogate modelling task. In traditional (one-shot) experimental design, all data points are selected and simulated beforehand, and a model is trained only after all points are simulated. In sequential design, points are selected with an iterative method, in which intermediate models and outputs from previous simulations are used to make a better, more informed choice on the locations for future simulations.

In this thesis, several new sequential design methods are proposed for black box, deterministic computer simulations. In this context, little or nothing is known up front about the behaviour of the simulator, and the only information available is acquired from previously evaluated samples. If no information is available up front, it can be very difficult to determine the total number of simulations that will be needed, which makes one-shot experimental designs a huge risk. Sequential design methods are much safer, because they avoid the selection of too many samples (oversampling) or too little (undersampling).

First, a number of novel space-filling design methods are proposed. These methods try to spread the data points as evenly over the design space as possible, in order to produce a uniform distribution of samples. This is done without knowing in advance how many samples will be needed. These methods are thoroughly compared against existing methods from different fields of research, and it is demonstrated that the new methods can sequentially produce space-filling designs that rival the best one-shot methods.

Next, a method is proposed that spreads samples according to the local nonlinearity of the system. Regions with high nonlinearity are more densely sampled than linear regions, because it is assumed that these nonlinear regions are more difficult to model than linear (flat) regions. This method, called LOLA-Voronoi, uses an approximation of the local gradient of the system to estimate the nonlinearity. LOLA-Voronoi is compared against a number of other sequential methods, and it is shown that it is a robust algorithm that can greatly reduce the total number of samples in different contexts.

All the methods proposed in this thesis are freely available in the SUMO (SUrrogate MOdelling) and SED (Sequential Experimental Design) toolboxes. SUMO is a Matlab toolbox for adaptive surrogate modelling with sequential design, which tackles the entire modelling process from the first samples through simulation to model training and optimization. SED is a specialized tool for sequential design which can be easily integrated into an existing modelling pipeline. Both toolboxes are open source, and therefore all experiments in this thesis can be reproduced and validated. SUMO and SED can be downloaded from `http://sumo.intec.ugent.be`.

Samenvatting

Titel: Surrogaatmodelleren van Computerexperimenten met Sequentieel Ontwerp

In veel moderne ingenieursproblemen worden nauwkeurige, op fysische eigenschappen gebaseerde computersimulaties gebruikt als alternatief voor reële experimenten, om kosten te drukken of om gevaar te minimaliseren. Deze simulaties worden door de ingenieur gebruikt om inzicht te verwerven in het gedrag van het bestudeerde systeem. Zo kan de ingenieur beter begrijpen hoe de verschillende parameters met elkaar interageren en hoe het gedrag van het systeem geoptimaliseerd kan worden.

Omdat deze computersimulaties dikwijls gebaseerd zijn op nauwkeurige fysische modellen, kan één enkele simulatie met meerdere inputs en meerdere outputs soms zeer lang duren. Zo rapporteerde Ford Motor Company over een car crash simulatie voor een passagiersauto die 36 tot 160 uur duurde [36]. Vanwege deze lange rekentijden, zijn deze computersimulaties nog steeds zeer onpraktisch voor ingenieurs die het systeem willen begrijpen, interpreteren of optimaliseren.

Het doel van globaal surrogaatmodelleren is om een benaderingsmodel te zoeken dat het gedrag van de originele simulator zo goed mogelijk benadert, maar dat veel sneller uitgevoerd kan worden dan de originele simulator. Dit model wordt getraind door een aantal intelligent gekozen simulaties uit te voeren, de resultaten van deze simulaties te analyseren en deze informatie te gebruiken om een model te kiezen dat deze data zo goed mogelijk benadert. Daarna kan dit surrogaatmodel gebruikt worden als een volwaardig alternatief voor de originele simulator dat veel sneller uitgevoerd kan worden en dus praktischer is voor de ingenieurs om te gebruiken.

Het is duidelijk dat de keuze van de datapunten van essentieel belang is voor de kwaliteit van het surrogaatmodel. In traditioneel (one-shot) experimenteel ontwerp worden alle punten op voorhand gekozen, en wordt er pas een model getraind wanneer alle punten geëvalueerd zijn. In sequentieel ontwerp worden punten volgens een iteratieve methode gekozen, waarbij tussentijdse modellen en outputs van voorgaande punten gebruikt worden om een betere, intelligentere keuze te maken voor de locatie van de toekomstige punten.

In deze thesis worden nieuwe methodes voorgesteld voor sequentieel ontwerp van black box, deterministische computersimulaties. Bij deze computersimulaties is er op voorhand weinig of niets geweten over het gedrag van de simulator, en is de

enige informatie die beschikbaar is de resultaten van reeds uitgevoerde simulaties. In deze context kan het zeer moeilijk zijn om op voorhand in te schatten hoeveel datapunten er nodig zijn, waardoor one-shot experimenteel ontwerp zeer risicovol kan zijn. Sequentieel ontwerp is dan een veiligere keuze die kan voorkomen dat er te veel of te weinig datapunten geëvalueerd worden.

Eerst worden een aantal zogenaamde space-filling methodes voorgesteld. Deze methodes proberen de datapunten zo goed mogelijk te verspreiden over de ontwerpruimte, om zo een zo uniform mogelijke dekking van de ontwerpruimte te bekomen. Dit wordt bereikt zonder op voorhand een idee te hebben van hoeveel punten er uiteindelijk nodig zullen zijn. Deze methodes worden uitgebreid vergeleken met bestaande methodes van verschillende onderzoeksdomeinen, en er wordt gedemonstreerd dat ze sequentieel een spreiding kunnen genereren die kan concurreren met de beste one-shot methodes.

Vervolgens wordt een methode voorgesteld en bestudeerd die datapunten spreidt volgens de lokale nonlineariteit van het systeem. Er worden meer datapunten geplaatst op nonlineaire gebieden, omdat er wordt aangenomen dat deze gebieden moeilijker te modelleren zijn dan lineaire (vlakke) gebieden. Deze methode, genaamd LOLA-Voronoi, maakt hiervoor gebruik van een benadering van de gradiënt van het systeem. Deze methode wordt vergeleken met bestaande methodes, en er wordt aangetoond dat het een robuust algoritme is dat werkt voor verschillende problemen in verschillende situaties.

Al de algoritmes voorgesteld in deze thesis zijn geïmplementeerd en beschikbaar in de SUMO (SUrrogate MOdelling) en SED (Sequential Experimental Design) toolboxen. SUMO is een Matlab toolbox voor adaptief surrogaatmodelleren met sequentieel ontwerp, die het hele modelleringsproces beheert, van het kiezen van de eerste punten tot het trainen van het finale model. SED is een gespecialiseerde toolbox voor sequentieel ontwerp die gemakkelijk kan geïntegreerd worden in een bestaand modelleringsproces. Beide toolboxen zijn open source, waardoor alle experimenten in deze thesis gereproduceerd en gecontroleerd kunnen worden. SUMO en SED kunnen gedownload worden op `http://sumo.intec.ugent.be`.

Dankwoord

Why did I follow him...? I don't know. Why do things happen as they do in dreams? All I know is that, when he beckoned... I had to follow him. From that moment, we traveled together, East. Always... into the East.
— Marius

Ik wil via deze weg een aantal mensen bedanken die belangrijk voor me zijn geweest op de 5 jaar lange tocht naar mijn uiteindelijke doctoraatsverdediging. Alleen is maar alleen, en jullie gezelschap heeft me zowel op professioneel als op persoonlijk vlak veel vooruit geholpen.

Eerst en vooral wil ik mijn promotoren Tom Dhaene en Jan Broeckhove bedanken. Tom, bedankt om mij de kans te geven om dit onderzoek te voeren en mijn eigen weg te zoeken in de materie, zodat ik uiteindelijk een niche kon vinden die me echt lag. Je hebt me altijd goed begeleid en gesteund, en je feedback was steeds waardevol. Jan, bedankt om mij logistiek te ondersteunen in het uitwerken van het duodoctoraat, en het in orde brengen van alle zaken achter de schermen. Het heeft mij de kans gegeven om me volledig te concentreren op mijn werk.

Ook wil ik mijn (ex-)collega's Dirk Gorissen, Ivo Couckuyt, Wouter Hendrickx en Wim van Aarle bedanken, voor hun enthousiasme, inzet en medewerking. Het was een bijzonder leerrijke ervaring om met jullie samen te werken aan een groot, complex wetenschappelijk softwarepakket zoals SUMO en alles tot een goed eind te brengen. De kennis opgedaan door samen met jullie SUMO van nul af aan op te bouwen zal me in mijn toekomstige loopbaan zeker nog van pas komen.

Vervolgens wil ik mijn familie bedanken. Papa, voor je nuchtere kijk op de wereld, je eerlijke meningen en de fantastische opvoeding die ik genoten heb. Ik heb van jou de vrijheid gekregen om mijn eigen toekomst uit te stippelen. Ik apprecieer het enorm dat ik van jongsaf aan veel vertrouwen en verantwoordelijkheid heb gekregen, en dat heeft in grote mate bepaald wie ik geworden ben. Ilse en Sanne, voor de leuke feestjes die we samen op jullie appartement gehad hebben, en de leuke gesprekken die daar gevoerd werden. Peter, voor de vele dolle avonden die we beleefd hebben en voor de fantastische brouwerijweek in Wallonië, die me erg de ogen heeft geopend.

In de 9 jaar dat ik op de universiteit heb doorgebracht, heb ik veel nieuwe vrienden gemaakt. Wim van Aarle, Bram Derkinderen, Jan Vlegels, Wesley Jordens en Gert Heiremans: bedankt om mijn studiejaren kleur te geven. Bedankt aan Pieter Belmans om deze tekst zo minutieus na te lezen en te verbeteren. Bedankt aan WINAK om mij te introduceren in het studentenleven. Bedankt ook aan Andie Similon, Evi De Cock, David Staessens, David Van der Cruyssen en al mijn andere vrienden van de Neejberhood: Neejberhood leeft voort!

Verder wil ik nog Dieter Schwerdtfeger bedanken voor de fantastische tijd die we samen gespendeerd hebben, al gamend in het computerkot, en alle goede gesprekken over het leven. Aan Olivier Mees en Dennis Vandaele: ook bedankt voor de duizenden uren die we online samen besteed hebben. En ik mag uiteraard Julian De Backer niet vergeten: je bent mijn oudste vriend, en hoewel we elkaar niet zo veel zien, betekent onze vriendschap zeer veel voor mij.

En tenslotte wil ik Caroline Kussé nog bedanken: ik heb zeer veel van je geleerd over het leven, en ik zal je altijd in mijn hart dragen. Je bent een fantastische, lieve meid. Het ga je goed!

Karel Crombecq

Contents

CHAPTER 1

Introduction

Hello my friend. Stay a while and listen!
— Deckard Cain

1.1 Computer simulation

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.

A computer simulation, also called a computer model, is a program that attempts to simulate a complex system from nature. Given a set of inputs (also called factors or variables), the simulation will produce an output (also called response), which can then be verified with a real-life experiment to make sure the computer program contains no errors. Once the simulation program has been validated, it can be used instead of real-life experiments to further study the subject, with reduced cost and risk.

However, not each computer simulation is created equal, and it is important to understand the different properties of computer simulations, in order to properly analyze the results produced by the simulation. Issues such as noise, domain knowledge and domain constraints can completely change the way the data should be handled. In the following sections, we will shortly discuss the different properties a computer simulation might have.

From now on, we will refer to the space containing all possible input values for each input as the *design space*. A specific combination of input values, together with the output produced by performing the simulation with these input values will be called a *sample*. *Evaluating a sample* means running the simulation on the input values to obtain the appropriate output value.

1.1.1 Time cost

The simulation of one single instance of a complex system with multiple inputs and outputs 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 [36]. 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.

The main focus of this thesis is on expensive computer simulations, and how to minimize the number of simulations that must be performed to understand the problem. If the simulation is cheap (i.e. millions of data points can be evaluated), the complex algorithms proposed in this thesis might not be suitable. It might be cheaper to just generate a large grid of data points than to use a complex (slower) algorithm to determine the ideal locations for the data points. Only when the advantage gained by requiring much less data points outweighs the additional computational cost of the more complex algorithms, do these algorithms become appropriate.

However, even for simulations which take only a few seconds, evaluating a large grid can still be unviable. For example, evaluating a $100 \times 100 \times 100$ grid for a 30-second simulation in 3 dimensions on a supercomputer with 100 cores already takes more than 83 hours to finish, which might or might not be acceptable, depending on the available resources. This clearly demonstrates the need for intelligent algorithms that can substantially reduce the number of simulations.

1.1.2 Output type

Simulations can be broadly categorized in the type of output they produce. A large class of simulation problems produce discrete values as output — these are called classification problems. For classification problems, the goal is to map each set of inputs on the appropriate discrete output value, for example `yes` or `no` in the binary classification problem. This is a long-standing research subject originating from statistics, which was picked up by the machine learning community as supervised learning [93]. Many algorithms are available to tackle problems with discrete output values, such as decision trees, nearest neighbour algorithms, support vector machines, and so on.

Another class of simulations are those with continuous outputs. These can be either real or complex outputs, for which an unlimited number of possible values exist. Because there is an unlimited number of possible output values, algorithms which were designed for discrete output values do not directly apply to these problems. This problem, which is known as regression analysis in statistics and machine learning [30, 80], is the main focus of this research.

1.1.3 Input type

The inputs can also be either discrete or continuous, which again results in different research fields which have their own terminology, methodology and practices. In many contexts, discrete inputs are the natural choice: this is often the case in experiments in which something is either included in the experiment, or excluded from it, such as a chemical component. But even inputs which are naturally continuous are often still treated as discrete inputs, to make it easier to determine the simulations that must be performed. For example, a grid is a discretisation of the input space into evenly spaced points. Many other methods use a subset of the full grid. These methods do not consider the entire range of viable values, but a small subset. This part of the research field is thoroughly discussed in [10].

Other methods treat real-valued inputs as they are, and consider the entire (infinitely large) range of possible values for each simulation [67]. This allows for more optimal decisions in terms of input values, but also makes the problem much harder to tackle, because the search space is infinitely large. Both approaches have their merit, and will be discussed and compared in this thesis.

1.1.4 Dimensionality

The input dimension of the simulation is a deciding factor in how to work with the simulation. The curse of dimensionality dictates that high-dimensional problems quickly become intractable, because data spreads out exponentially with the number of dimensions. Therefore, analyzing high-dimensional data, even when there are millions of data points available, will only be possible to a certain degree.

Several solutions to dealing with high-dimensional data have been proposed over the years. Sometimes, models are limited to linear models (assuming the response is only linearly dependent on each variable). This approach is taken by the linear regression field of statistics [47]. Additionally, the number of inputs can be reduced by variable selection or feature selection methods, which determine the most important inputs of the simulator and reduce the size of the model by discarding irrelevant inputs. Another way to deal with the curse of dimensionality is by assuming the output is discrete, making the modelling process considerably simpler. This is the assumption followed by the classification field of machine learning [60].

Low-dimensional simulators, on the other hand, can be analyzed and modelled in great detail, taking into account nonlinear dependencies, complex interactions between inputs and high accuracy requirements. Many problems can be described as low-dimensional problems, or can be reduced to a low-dimensional problem by means of variable selection. The focus of this thesis is mainly on low-dimensional problems; methods are developed for problems with less than 10 dimensions in mind. However, some of the methods proposed will scale well to higher dimensions.

1.1.5 Noise

Noise is a random and unwanted fluctuation in a signal, or in this case: the outputs of the simulation. Noise can make it considerably more difficult to model a system. In computer simulations, there is a big difference between deterministic noise and stochastic noise. Deterministic noise is noise inherent to the simulation implementation, and performing the same simulation (with the exact same inputs) twice will yield the same noise on the output. This noise comes from inaccuracies in the implementation of the simulation, so that it does not perfectly mimic the behaviour in nature, and introduces some level of noise to the output.

Stochastic noise, on the other hand, is caused by random elements in the implementation. This type of noise will produce (slightly) different outputs when the same simulation is performed twice. This can be the case when, for example, Monte Carlo methods are used in the simulation implementation. It is very important to know if this noise is present, as it needs to be taken into account in every stage of the analysis of the data [98]. When stochastic noise is present, classical statistical methods such as replication become relevant.

In this thesis, the focus will be on deterministic computer simulations, which have either no (relevant) noise at all, or only deterministic noise. Therefore, replication will not be taken into account.

1.1.6 Black or white box

When no information is known about the behaviour of the system, the simulation is called a black box. 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. The only way to acquire information about the behaviour is by performing simulations and analyzing the results.

When the system is a grey or white box, information is available about the inner workings and behaviour of the system. This allows the engineer to make informed decisions about which methods to use to analyze the data and which simulations to perform. For example, when the engineer knows that the different inputs only have a linear effect on the output, linear models can be used to accurately model the system with very little data. When this information is not available, more complex models must be used that can handle nonlinear data, such as Kriging [82]. These models might not be as good at modelling linear data as the linear models.

In this thesis, we focus on black box systems, and limit ourselves to analyzing the output to gain insight into the system behaviour.

1.2 Surrogate modelling

Because simulations are assumed to be expensive, it is impractical to explore the design space thoroughly by evaluating large amounts of samples directly. The goal of *global* surrogate modelling (or metamodelling) is to find an approximation function (also called a surrogate model) that mimics the original system's behaviour, but can be evaluated 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 [8, 43]. This is illustrated in Figure 1.1.

There are a wide variety of model types available, and which one is most suitable depends largely on the system that is to be modelled. Popular choices are polynomial and rational functions [22, 48], Kriging models [9, 87], neural networks [5, 77] and radial basis functions (RBF) models [61]. Once the model is constructed, it can be used to perform optimization and sensitivity analysis and to gain insight in the global structure and behavior of the function [7, 24, 99].

Please note that *global* surrogate modeling differs from *local* surrogate modeling in the way the surrogate models are employed. In *local* surrogate modeling, local models are used to guide the optimization algorithm towards a global optimum [81]. The local models are discarded afterwards. In *global* surrogate modeling, the goal is to create a model that approximates the behavior 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 modeling is to overcome the long computational time of the simulator by providing a fast but accurate approximation, based on a one-time up front modeling effort. In this thesis, we are only concerned with global surrogate modeling.

Mathematically, the simulator can be defined as an unknown function $f : \mathbb{R}^d \to \mathbb{C}$, mapping a vector of d real inputs to a real or complex output. This function can be highly nonlinear and possibly even discontinuous. This unknown function has been sampled at a set of scattered data points $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\} \subset [-1, 1]^d$, for which the function values $\{f(\mathbf{p}_1), f(\mathbf{p}_2), \ldots, f(\mathbf{p}_n)\}$ are known. In order to approximate the function f, a function $\tilde{f} : \mathbb{R}^d \to \mathbb{C}$ is chosen from the (possibly) infinite function set of candidate approximation functions F.

The quality of this approximation depends on both the choice and exploration of the function space F and the samples P. 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 samples P [11, 52].

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

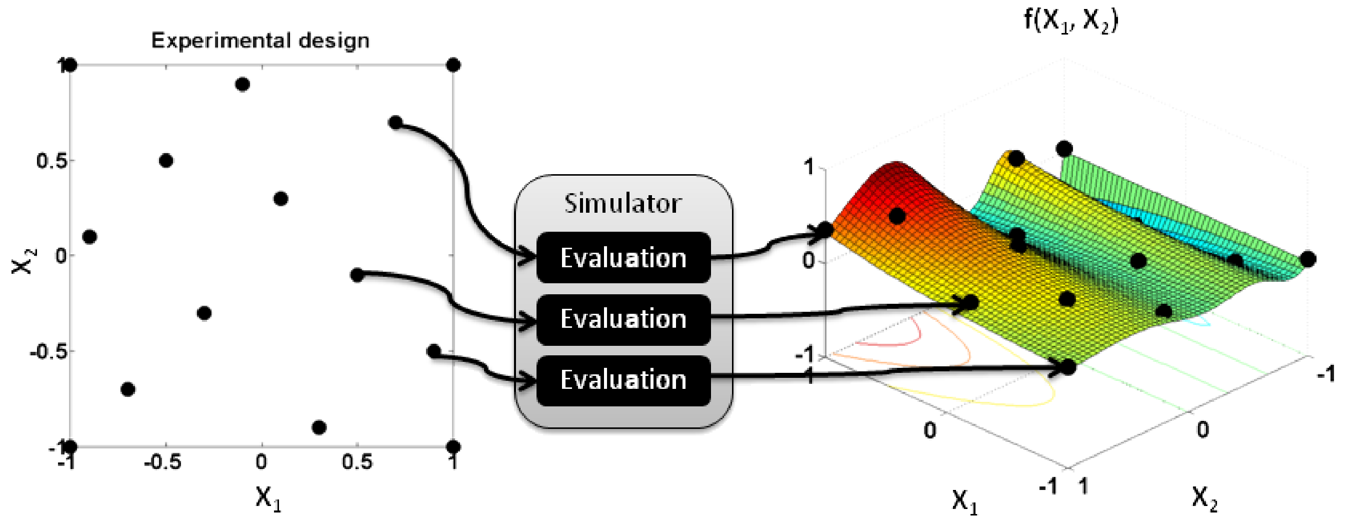


Figure 1.1: A set of data points is evaluated by a black box simulator, which outputs a response for every data point. An approximation model (surrogate model) is fit to the data points, with the goal of minimizing the approximation error on the entire domain.

1.3 Sequential design

In traditional design of experiments (DoE), the experimental design P is chosen based only on information that is available before the first simulation, such as the existence of noise, the relevance of the input variables, the measurement precision and so on. This experimental design is then fed to the simulator, which evaluates all the selected data points. Finally, a surrogate model is built using this data. This is essentially a one-shot approach, as all the data points are chosen at once and the modeling algorithm proceeds from there, without evaluating any additional samples later on. This process is illustrated in Figure 1.2.

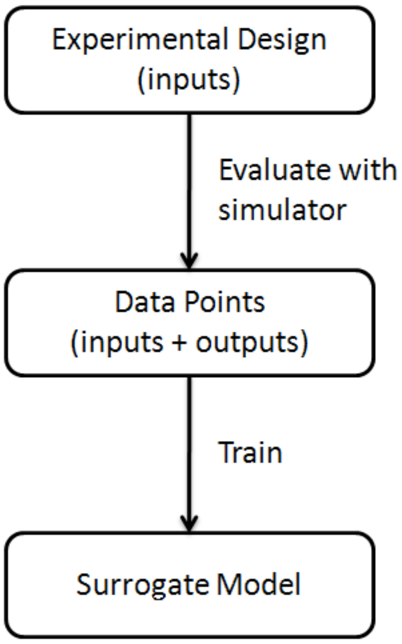


Figure 1.2: A one-shot experimental design flow-chart.

In the deterministic black box setting, where there is no information available up front and statistical methods such as blocking and replication lose their relevance, the only sensible one-shot experimental designs are space-filling designs, which try to cover the design space as evenly as possible. The advantages of classical 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 [88], Latin hypercubes [87] and orthogonal arrays [25].

Sequential design improves on this approach by transforming the one-shot algorithm into an iterative process. Sequential design methods analyze data (samples) and models 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. Sequential design is also known by other names, such as adaptive sampling [64] and active learning [89], depending on the research field. For example, in machine learning, active learning is the prefered term. Throughout this thesis, we will call it sequential design.

1.3.1 Sequential design methods

A typical sequential design method is described in Algorithm 1. First, an initial batch of data points is evaluated using a minimal one-shot experimental design. This design is usually one of the traditional designs from DoE, such as a (sparse) Latin hypercube. The initial design must be large enough to guarantee a minimal coverage of the design space, but should be small enough so that there is room for improvement, allowing the sequential design strategy to do its work.

Algorithm 1 A typical sequential design method.

```
$P \leftarrow$ initial experimental design
Calculate $f(P)$ through simulation
Train model using $P$ and $f(P)$
while accuracy not reached do
    Select new data points $P_{\text{new}}$ using sequential design strategy
    Calculate $f(P_{\text{new}})$ through simulation
    $P \leftarrow P \cup P_{\text{new}}$
    Train model using $P$ and $f(P)$
end while
```

Based on the initial experimental design, a surrogate model is built and the accuracy of this model is estimated using one or more well-known error metrics. Examples of error metrics are cross-validation, an external test-set, error in the data points, and so on. Based on the estimated accuracy of the model, the algorithm may (and probably will, if the initial design was small enough) decide that more samples are needed. This process is shown in Figure 1.3.

The locations of these additional samples are chosen by the adaptive sampling or sequential design strategy. Many different strategies are available, and the optimal strategy may depend on several factors, including the surrogate model type that is used, the number of samples required, the system that is being modelled and so on. Finally, a new surrogate model is built using all the data gathered thus far, and the model accuracy is estimated again. If the desired accuracy is still not reached, the entire sample selection process is started all over again.

Ultimately, the goal of this algorithm is to reduce the overall number of samples, since evaluating the samples (running the simulation) is the dominant cost in the entire surrogate modelling process. If samples are chosen sequentially, more information is available to base the sampling decision on compared to traditional design of experiments. Both the previous data points and the behaviour of the intermediate surrogate model provide important information on where the next

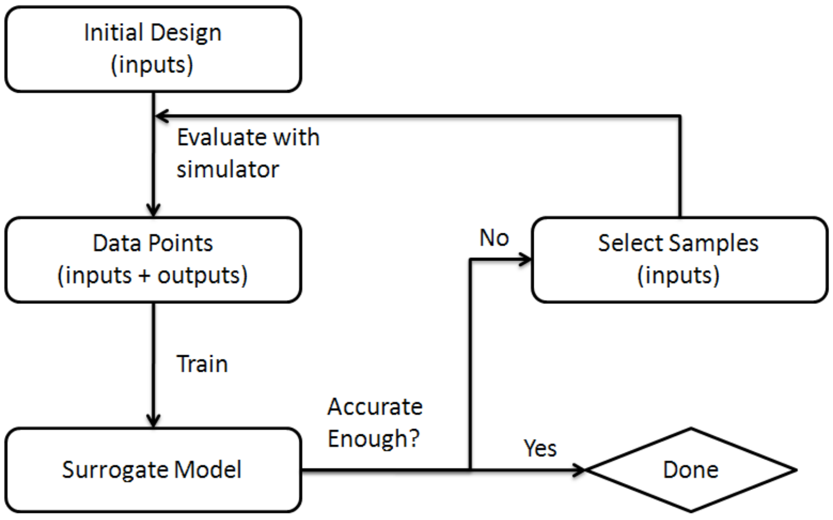


Figure 1.3: A sequential design flow-chart.

sample(s) should be located. If this additional information is used well, the total number of samples can be reduced substantially.

Note that some optimization algorithms use similar iterative schemes, but with a completely different goal. These optimization algorithms may also employ sequential design techniques to minimize the number of samples required to find the global optimum. However, they are not concerned with finding a good global approximation over the entire design space. Because of this, a lot of optimization-oriented sequential design techniques ignore large portions of the design space and focus heavily on the (estimated) optima. In this thesis, we will only consider related work on sequential design in the context of global surrogate modelling. For more information about sequential design in the context of optimization, please refer to [29, 59].

1.3.2 Exploration and exploitation

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 or stable regions, 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 goal is to fill up the input domain evenly. Examples of exploration methods can be found in [33, 73, 78, 79, 92].

The advantage of exploration-based sequential design methods over one-shot experimental designs is that the amount of samples evaluated depends on the feedback from previous iterations of the algorithm (when the model is accurate enough, no more samples are requested). 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 experimental design.

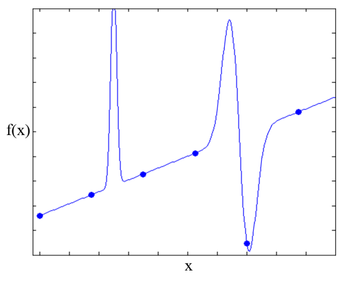
Instead of exploring the input domain, exploitation-based methods select samples in regions which have already been identified as (potentially) interesting. For example, one might want to zoom in on optima, in order to make sure the surrogate model does not overshoot the optimum. Or one might also want 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.

Over the years, many exploitation-based methods have been proposed. Jin et al. [53] described an algorithm that uses a maximin design to select new samples in combination with an importance weighting for the input variables based on previous surrogate models. Farhang-Mehr et al. [28] proposed an algorithm that uses previously built surrogate models to sample regions in which many local optima of the surrogate model are close to each other. Turner et al. [91] use a sequential design technique for a NURBS-based surrogate model that balances multiple sampling criteria using a simulated annealing approach. Lin et al. [67] train a second surrogate model on the prediction errors in order to guide the sampling process to locations in which the surrogate model performs poorly. Kleijnen et al. [58] employ cross-validation and jackknifing to predict the variance at new candidate sample locations in the context of Kriging. Osio et al. [75] developed an adaptive algorithm that weights variables according to their estimated importance, and samples accordingly.

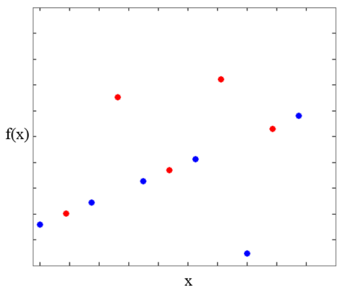
In every sequential design strategy, a trade-off must be made between these two conflicting options. If a sequential design strategy only focuses on exploitation, the initial experimental design must be sufficiently large as to capture all regions of interest right away. Otherwise, large (interesting) areas may be left unsampled, a problem which they share with optimal designs [11]. Because the simulator is often a black box, it is infeasible in practice to predict the required size for the initial design accurately. On the other hand, if a strategy focuses only on exploration, one of the advantages provided by evaluating and selecting the samples sequentially is ignored, because the outputs are not used.

The trade-off between exploration and exploitation is illustrated in Figure 1.4. It is clear that without proper design space exploration, any sequential design strategy is bound to miss important regions in the response surface. Thus, every sequential design strategy must be space-filling to a certain degree. On top of this necessary foundation, exploitation-based methods can then zoom in on interesting regions to improve the generalization error in that region.

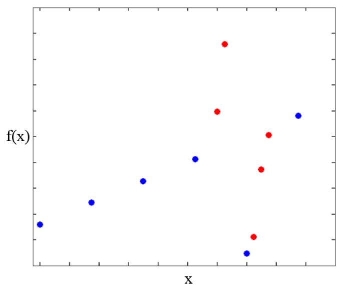
The necessary trade-off between exploration and exploitation can be accounted for in different ways. In sequential design methods [28, 67], the trade-off is buried deep in the formulation of the algorithm. In other strategies [91], the difference between exploration and exploitation is very clear, in that a simulated annealing



(a) Initial set of samples and target function



(b) Exploration



(c) Exploitation

Figure 1.4: This figure demonstrates the trade-off between exploration and exploitation. In Figure 1.4(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 1.4(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 1.4(c).

approach is used to balance between the two and to switch priorities from one to the other during the modelling process.

1.3.3 Optimal and generic sequential design

A large class of sequential design methods assume that the model type is known in advance. This allows the algorithm to exploit the behaviour of this model to guide the sampling process in the optimal direction for this specific model type. This is called optimal design.

Many sequential design methods use some aspects of optimal design to generate new samples. By far the most popular model type for this approach is Kriging (also known as Gaussian process models). For example, Busby et al. [11] split the design space into cells using a domain decomposition strategy, ensuring a certain degree of domain coverage. In each cell, their algorithm applies a local sequential optimal design. Gramacy et al. [44] use a treed Gaussian process model, which is an extension of the standard Gaussian process model, to approximate a black box system. After an initial batch of samples is selected using a Latin hypercube design, active learning methods (Active Learning-McKay and Active Learning-Cohn) are used to select more samples in regions with high levels of uncertainty. Sasena et al. [85] proposed an algorithm called Switch which switches between a Kriging-based exploration and exploitation strategy. Lehmensiek et al. [63] suggest an adaptive sampling strategy which uses the estimated interpolation error calculated from the last two models to choose new sample locations.

All of these methods incorporate to some degree information about the model in the sampling process. These sampling strategies may be highly efficient if the model for which it was developed is suitable for the problem at hand. However, this may not always be the case, as the optimal model type for a specific problem might not be known up front. This motivates the need for a generic algorithm, which makes no assumptions about the model type, the behaviour of the system or the amount of samples needed.

Generic sequential design strategies can only use the outputs from the simulator and previously built models to decide where to sample next. They cannot make any assumptions about how the model will behave, or which type of model is used. In fact, completely different model types may be used at the same time in a heterogeneous modelling environment [41]. This is a major advantage over optimal sequential design strategies, especially in a black box setting where little or nothing is known about the system in advance. In this case, choosing a model type for the problem comes down to guesswork, and if a bad choice is made, the optimal design that will be generated will not be optimal for another model type that might be tried later on. A heterogeneous modelling environment can help solve this problem by automatically looking for model types that match the problem at hand, while generating a sequential design that is not specifically tailored to one model type. Heterogeneous modelling environments, in which many completely different types of models are considered together, have a larger search space of candidate functions F, and can therefore drastically improve the accuracy of the final model. Because of these advantages, all the methods proposed in this thesis will be generic methods.

1.3.4 Generic sequential design overview

Based on the different approaches described in the previous sections, sequential design methods can be divided into four categories, based on how much information they use to determine where to select the next sample. This is illustrated in Table 1.1.

Input-based methods only use the inputs from previously selected samples to determine where to sample next. Because outputs are not used at all, this category contains the space-filling methods. Output-based methods use the outputs generated from previous simulations to determine a more optimal sample distribution, tailored to the simulator. However, these methods are still generic with regards to the surrogate model, because they do not evaluate models from previous iterations to determine the new sample location. Therefore, they are still suitable for heterogeneous modelling environments.

The last two categories use respectively model evaluations and model parameters to determine new sample locations. In the last category, Kriging is very popular because properties such as variance can be directly derived from the model parameters. These two categories optimize the sample distribution for one specific model type, and therefore cannot be used properly in heterogeneous environments.

Table 1.1: The four different categories for sequential design methods.

Input-based (= exploration)	Output-based (= exploitation)	Model output-based	Model-based
Uses only input values from previous samples to determine next sample.	Uses input and output values from previous samples to determine next sample.	Uses previous samples and model evaluations to determine next sample.	Uses previous samples and model properties and parameters to determine next sample.
Examples: • Random sampling • Low-discrepancy sequences [49, 54, 73] • Sequentially nested Latin hypercubes [50, 79, 92] • Voronoi-based sampling [18] • Monte Carlo/Optimization-based sampling [20] • Chapter 2	Examples: • LOLA-Voronoi [18, 19] • Chapter 3	Examples: • Adaptation to irregularities [28] • Slope, local optima and variance criteria [91] • Sequential Exploratory Experimental Design method (SEED) [67] • Model error sampling [48]	Examples: • Kriging-based [11, 44, 53, 58, 63, 85]

1.4 Research goals

The goal of this thesis is to propose new, highly efficient generic sequential design algorithms for deterministic black box computer simulations. The methods proposed in this thesis provide a good alternative for classical experimental design techniques by reducing the total number of simulations required. Because the simulation is assumed to be a black box, it is difficult to determine the best model type up front. Therefore, the focus of this thesis lies on input-based and output-based methods, which do not rely on a particular model type for determining the samples.

In Chapter 2, a set of novel, fast space-filling sequential design methods are proposed. These methods can be used as replacements for pre-optimized Latin hypercubes, which have excellent space-filling properties but are extremely costly to compute and are not available in high dimensions or for a larger of samples. The methods proposed in this chapter scale well to high dimensions and large sample sizes, and can be computed much faster.

In Chapter 3, we propose the LOLA-Voronoi algorithm, a powerful exploitation-based sequential design algorithm which selects new samples in regions of high variability, because these regions are assumed to be more difficult to approximate than smooth, linear regions.

The sequential design methods presented in this thesis are freely available in the SUMO (SUrrogate MOdelling) Toolbox and the SED (Sequential Experimental Design) Toolbox, which are both open source Matlab toolboxes which can be found at `http://sumo.intec.ugent.be`. These toolboxes are discussed in Chapter 4.

CHAPTER 2

Input-based sequential design

You speak of knowledge, Judicator? You speak of experience? I have journeyed through the darkness between the most distant stars. I have beheld the births of negative suns and borne witness to the entropy of entire realities... Unto my experience, Aldaris, all that you've built here on Aiur is but a fleeting dream.
— Zeratul

2.1 Motivation

Space-filling design methods play a prominent role in modeling today. Stemming from traditional experimental design, one-shot methods such as Latin hypercubes have become very popular as the go-to stop for any situation. However, Latin hypercubes are far from perfect, and using them without thought can severely harm the resulting model, as we will try to demonstrate in this chapter.

The one-shot experimental design methods are mainly popular because they are widely available (most computational packages support them out-of-the-box) and easy to use and understand. However, they require an up-front guess of the total number of sample points, which can be very difficult in a black box setting. Sequential space-filling methods, on the other hand, do not need any information on the total number of samples beforehand, and will always produce good designs, no matter after how many points the algorithm is stopped. This is a huge advantage over the one-shot approaches.

One might ask why one would want to develop and use purely input-based (space-filling) methods if output-based methods are available that tailor the design to the problem at hand. There are several reasons why one would prefer input-based methods over the more complex output-based methods.

Firstly, input-based methods are usually much faster than output-based methods, as they do not need to analyse the outputs from previous simulations. In fact, output-based methods tend to scale relatively poor to high dimensions and large numbers of samples, while input-based methods scale much better. When it cannot be afforded to spend large amounts of time on analyzing the data to determine the next sample point, input-based methods can offer real-time sample selection at considerable speeds.

Secondly, every output-based method needs an exploration (input-based) component, as explained in Section 1.3.2. If an output-based method would not perform design space exploration, it would focus purely on the already identified interesting regions, and might overlook large portions of the design space that are equally interesting or more so. Hence, it makes sense to investigate good space-filling methods, even in the context of output-based sequential design.

Thirdly, some of the most popular modeling methods today, such as Kriging, actually prefer space-filling methods over methods that cluster points in hard-to-approximate regions. This is due to the structure of the correlation matrix for Kriging, which tends to become ill-conditioned when two points are placed too close to each other. In order to avoid this problem, data points should be spread out as evenly as possible over the design space, which is exactly what input-based methods do.

In this chapter, we will look into existing one-shot and sequential experimental design methods, and propose several new ones, based on different approaches and ideas such as Voronoi tessellations, Monte Carlo methods and local optimization. We will compare these methods to the existing ones, and show that they are a viable alternative to traditional one-shot designs.

2.2 Important criteria for experimental designs

From now on, we will consider the d-dimensional experimental design $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ containing n samples $\mathbf{p}_i = (p_i^1, p_i^2, \ldots, p_i^d)$ in the (hyper)cube $[-1,1]^d$. In order for this method to be a good space-filling sequential design strategy for computer experiments, it has to maximize the following criteria.

2.2.1 Granularity

The first criterion is the granularity of the strategy. A fine-grained sequential design strategy can select a small number of points (preferably one) during each iteration of the algorithm. It should also generate reasonably space-filling designs, no matter after which iteration the algorithm is stopped. A coarse-grained sequential design strategy, on the other hand, selects new samples in large batches. The reason why a fine-grained method is preferred, is because it completely avoids over- or undersampling. When samples are only selected in large batches, too many samples may be evaluated at once, because only a few samples of the last batch were necessary to arrive at the desired prediction error. It is also possible that the modeller decides to stop sampling before the desired prediction error is reached, because the next batch is too large and there is not enough computational time left to evaluate the entire batch.

Finally, the granularity of an algorithm also refers to the fact that the algorithm does not need to know the total number of samples that will be evaluated. Some methods, such as factorial designs and Latin hypercubes, require that the total number of samples be known in advance. In most real-life situations, this information is unavailable, because the simulator is assumed a black box, and the complexity of the simulator, and the difficulty to model the problem, is not known up front. Therefore a good space-filling sequential design method should not make any assumptions about the maximum number of samples, and should work reasonably well no matter how many samples will be selected in the end.

2.2.2 Space-filling

Secondly, the generated design should be space-filling. Intuitively, a space-filling design is an experimental design P in which the points are spread out evenly over the design space. However, there are several ways to define this property mathematically. Over the course of the years, many different space-filling criteria have been proposed. The goal is to select the design P to maximize the criterion of choice. Depending on the criterion, the optimal design P will look differently. Table 2.1 gives an overview of the most popular ones, along with some references of people using the criterion. Some criteria might be used under different names in different publications; we use the most common name in this table and in further references in this thesis.

Note that most authors are concerned with finding an optimal design when the number of design points n is given and known in advance, and the entire design is generated at once instead of sequentially (i.e. worst possible granularity). In some cases (see [50] and [79]), the authors introduce some granularity in their methods, but they remain too coarse-grained for expensive computer experiments, in which each single sample evaluation may take hours and should be considered carefully.

Of these different criteria, the ϕ_p criterion and the maximin criterion are the most widely used. The ϕ_p criterion is an extension of the maximin criterion, introduced by [72] to differentiate between two designs which have the same maximin value. For large p, the ϕ_p criterion tends to rank designs in the same order as the basic maximin criterion, while for small p, the criterion behaves like the Audze-Eglais criterion. Additionally, if p is large enough, ϕ_p will differentiate between designs which have the same maximin value, but for which the second smallest distance between points is different. In this way, the ϕ_p criterion is a family of criteria that encompasses both the maximin and Audze-Eglais criterion and everything in between.

However, the ϕ_p criterion has several disadvantages. The first problem is that it is numerically unstable in certain circumstances. When two points are very close to each other, and the power p is chosen large enough, ϕ_p will return infinity because of a floating point overflow. It is enough for one intersite distance to be rounded to zero, to result in a value of ϕ_p that is equal to infinity, no matter the quality of the rest of the design. The point at which this happens depends on both the design that is being rated and the number p, and the outcome is therefore difficult to predict in advance. This problem becomes an issue in sequential sampling, where the total number of samples (and therefore the distances that are to be expected) is unknown up front. Figure 2.1 demonstrates this problem.

Another problem with the ϕ_p criterion is that the value returned does not bear any geometrical meaning, and only provides a relative ranking between different designs. It is not intuitive to interpret the number and relate it to the density of the design. This also makes it difficult to combine the ϕ_p criterion with another one (for example: the projected distance criterion discussed in the next section). Additionally, the asymptotic nature of the ϕ_p criterion makes it very difficult to visualize the optimization surface, as the range of values is extremely small in most parts of the design space, and extremely large in small subparts.

Finally, it is also not easy to determine the ideal choice for the parameter p. If p is taken too small, the designs are not ranked in the same order as the maximin criterion would, which might be undesirable. But if p is taken too large, the instability issues illustrated in Figure 2.1 may occur. Morris et al. [72] perform several runs of a sequential design strategy with different values of p in order to find the value of p that works best for their problem. However, this severely slows down the sequential design method, as several independent runs with different values for p have to be performed.

Because of these issues, the ϕ_p criterion was not used in this study. The novel methods proposed in this thesis combine multiple criteria to find an optimal solution for a multi-objective problem, and because of the lack of geometric meaning and the asymptotic nature of the surface, it is very difficult to combine the ϕ_p criterion with anything else. Instead, the maximin criterion, which does not suffer from any of the aforementioned issues, will be used to both generate and rank the designs. From now on, the maximin space-filling criterion will be referred

Table 2.1: Overview of different space-filling criteria.

Criterion	Interpretation	Formula	References
Manhattan (l_1 norm)	Taxicab distance	$\min_{\mathbf{p_i},\mathbf{p_j}\in P}\sum_{k=1}^{d}\left\lvert p_i^k - p_j^k\right\rvert$	[92, 97]
Maximin (l_2 norm)	Euclidean distance	$\min_{\mathbf{p_i},\mathbf{p_j}\in P}\sqrt{\sum_{k=1}^{d}\left\lvert p_i^k - p_j^k\right\rvert^2}$	[50, 55, 56, 72, 92, 97]
Audze-Eglais	Average distance	$\sum_{\mathbf{p_i},\mathbf{p_j}\in P} 1/\sum_{k=1}^{d}\left\lvert p_i^k - p_j^k\right\rvert^2$	[3]
Centered L_2 discrepancy	Proportion of points in subinterval size of interval	see [27]	[26, 27, 49, 53, 54, 72]
ϕ_p	Generalization of maximin distance	$\left(\sum_{\mathbf{p_i},\mathbf{p_j}\in P}\sqrt{\sum_{k=1}^{d}\left\lvert p_i^k - p_j^k\right\rvert^2}^{-p}\right)^{1/p}$	[45, 53, 54, 72, 94]

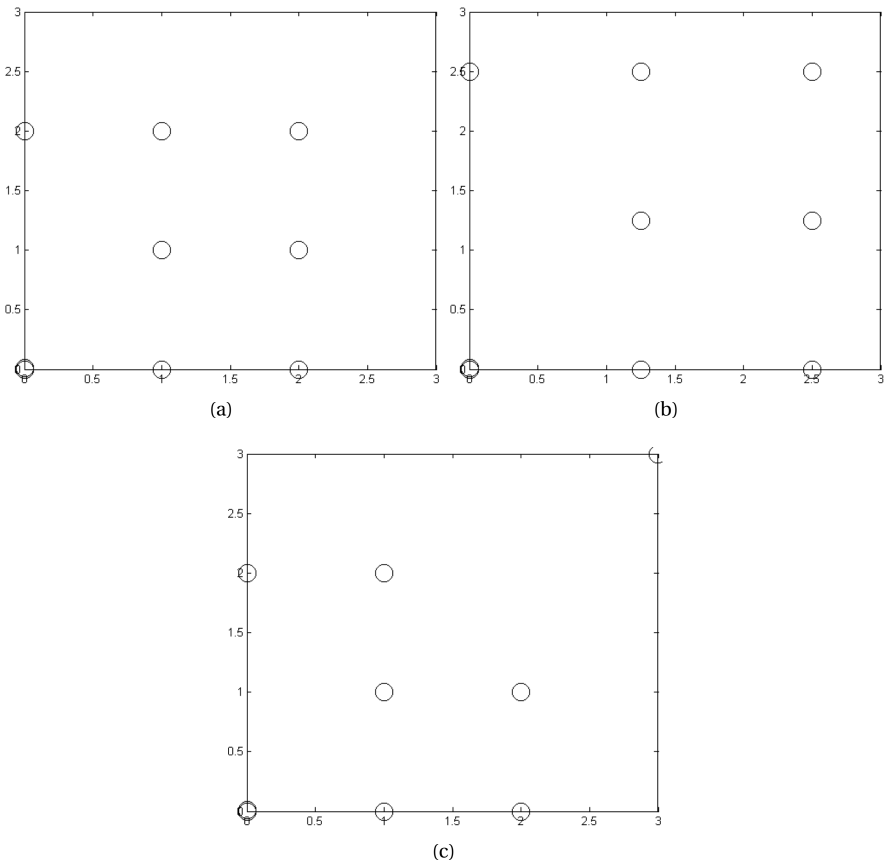


Figure 2.1: Three experimental designs with the same ϕ_p score. Because two points lie close to each other in the bottom left, the difference between these three designs is lost due to roundoff errors. Figure 2.1(b) is clearly a better design than 2.1(a) because the points are more evenly spread out over the design space, but this information is completely lost with the ϕ_p criterion.

to as the *intersite distance*, because it tries to maximize the smallest (Euclidean) distance any two sets of points (sites) in the design. The intersite distance is formally defined as

$$\text{idist}(P) = \min_{\mathbf{p_i},\mathbf{p_j} \in P} \sqrt{\sum_{k=1}^{d} \left\| p_i^k - p_j^k \right\|^2}. \tag{2.1}$$

The problems with ϕ_p are not a major concern when this criterion is used to rank Latin hypercube designs, which already guarantee by construction a minimal distance between points. This explains why authors such as Viana et al. [94] use the ϕ_p criterion without encountering any stability issues. They also do not combine the criterion with other criteria, because Latin hypercubes already guarantee good projective properties.

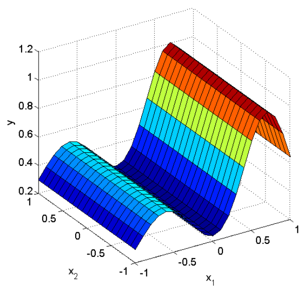
2.2.3 Good projective properties

A good space-filling design should also have good projective properties. This is also called the non-collapsing property by some authors [92]. An experimental design P has good projective properties if, for every point $\mathbf{p}_i$, each value p_i^j is strictly unique and is as different from all other values p_k^j as possible. This property also means that, when the experimental design is projected from the d-dimensional space to a $(d-1)$-dimensional space along one of the axes, no two points are ever projected onto each other.

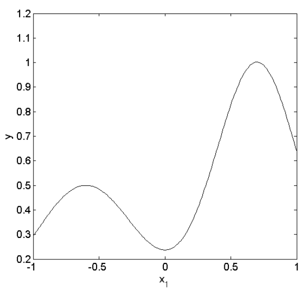
The quality of a design in terms of its projective properties can be defined as the minimum *projected distance* of points from each other:

$$\begin{aligned} \text{pdist}(P) &= \min_{\mathbf{p}_i,\mathbf{p}_j \in P} \min_{1 \le k \le d} \left| p_i^k - p_j^k \right| \\ &= \min_{\mathbf{p}_i,\mathbf{p}_j \in P} \left\| \mathbf{p}_i - \mathbf{p}_j \right\|_{-\infty} \end{aligned} \tag{2.2}$$

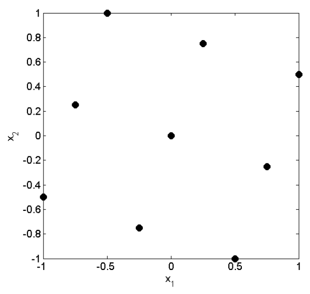
where $\|x\|_{-\infty}$ is the minus infinity norm. This is a useful property if it is unknown up front if there are design parameters included in the experiment which have little or no effect on the response. If this is the case, two samples which differ only in this design parameter can be considered the same point, and evaluating this same point twice is a waste of computational time. Therefore, each sample should preferably have unique values for each design parameter. Ideally, when all the points are projected onto one of the axes the remaining design should be space-filling as well. Preferably, all the projected points should be equidistant. It is expected that an experimental design with optimal (equidistant after projection) non-collapsing points will not suffer a performance hit when one of the design parameters turns out to be irrelevant. The importance of this property is illustrated in Figure 2.2.



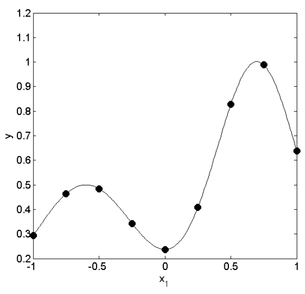
(a) 2D function



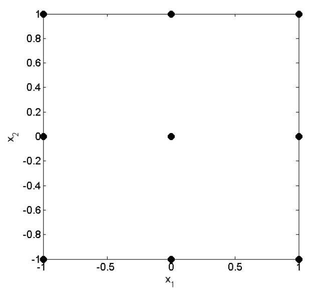
(b) 2D function projected on x_1



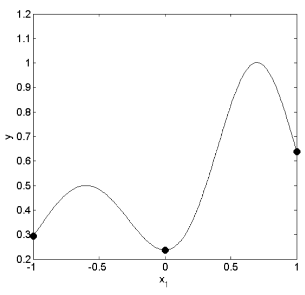
(c) 2D Latin hypercube design



(d) 2D Latin hypercube design projected on x_1



(e) 2D factorial design



(f) 2D factorial design projected on x_1

Figure 2.2: Figure 2.2(a) shows a two-dimensional function, for which only the dimension x_1 is relevant to the output. In other words, the output is constant in dimension x_2. Figure 2.2(b) shows the projection of this function on x_1. Two different designs are shown: a pre-optimized 9-point Latin hypercube with optimal projected distance in Figure 2.2(c) and a 9-point factorial design with the worst possible projected distance in Figure 2.2(e). When the Latin hypercube is projected onto x_1, no data is lost, and all points are still equidistant. When the grid is projected onto x_1, only 3 different points remain, and the remaining 6 are useless. This demonstrates the importance of the projected distance in cases when it is unknown whether each dimension is relevant to the output.

2.2.4 Orthogonality

Orthogonality is another desired property for an experimental design. A design P is called orthogonal with strength r if, for each subset of r inputs, each combination of different input values occurs the same number of times [76, 90]. This ensures that there is no correlation between the inputs in the design. Note that, according to this definition, only a small subset of possible designs can be orthogonal, namely those for which the input values are fixed at particular levels. The only designs included in this study that satisfy this condition are fractional factorial designs and Latin hypercube designs.

Additionally, for a given input dimension d and number of points n, an orthogonal design does not always exist. For the relatively small number of inputs and the (comparatively) large number of design points considered in this study, orthogonality cannot be satisfied. Even though Latin hypercubes can, by construction, never be completely orthogonal, they can be optimized such that subsets of the hypercube are [76, 90]. Because orthogonality is irrelevant to most designs discussed in this study, this criterion will not be considered in this study.

2.3 Optimization surface analysis

From the criteria defined in the previous section, the two most important ones are the intersite and projected distance. Generating an experimental design that optimizes these two criteria is a multi-objective optimization problem. Starting with two initial points (for example, opposing corner points), a new point is selected by finding a location in the design space that maximizes both the intersite and projected distance. Many different methods have been proposed to solve such multi-objective optimization problems efficiently. The simplest approach is to combine the different objectives in a single aggregate objective function. This solution is only acceptable if the scale of both objectives is known, so that they can be combined into a formula that gives each objective equal weight. Fortunately, in the case of the intersite and projected distance, this is indeed the case.

The aggregate intersite and projected distance criterion is defined as follows:

$$\mathrm{dist}(P) = \frac{\sqrt[d]{n+1}-1}{2}\mathrm{idist}(P) + \frac{n+1}{2}\mathrm{pdist}(P). \tag{2.3}$$

However, using this function as the objective is not yet ideal. Consider a design for which two points already have an intersite distance of 0.1. Then all new candidates that lie further away from the other points than 0.1 result in the same objective score, since the minimum intersite distance does not change. However, it is preferable to choose the point farthest away from the existing points. This is illustrated in Figure 2.3.

This might not have a substantial effect on subsequent iterations, but it improves the design at the current iteration. Because we do not know the total number of samples that will be generated up front, it is important to have a design as optimal as possible after each iteration. Therefore, instead of computing the distance of all points from each other, we just compute the distance of the new point from previous points, and optimize this function. The final objective function, which scores a new candidate point **p** when it is added to an existing design P, is defined as:

$$\mathrm{dist}(P,\mathbf{p}) = \frac{\sqrt[d]{n+1}-1}{2}\min_{\mathbf{p}_i\in P}\sqrt{\sum_{k=1}^{d}\left|p_i^k - p^k\right|^2} + \frac{n+1}{2}\min_{\mathbf{p}_i\in P}\left\|\mathbf{p}_i - \mathbf{p}\right\|_{-\infty} \tag{2.4}$$

2.3.1 Experimental setup

The objective function `dist` can now be optimized by any optimization algorithm. In this section, we will compare two approaches to finding the best location for the next point at each iteration. The first one is a Monte Carlo method. In this method a large number of uniformly distributed random points is generated, and for each candidate **p**, the objective function $\mathrm{dist}(P,\mathbf{p})$ is calculated and the best candidate is selected as the new point to be added to P. In the second method, a genetic algorithm will be used to optimize the objective function and find the best candidate. Both methods will be compared for different settings, and the

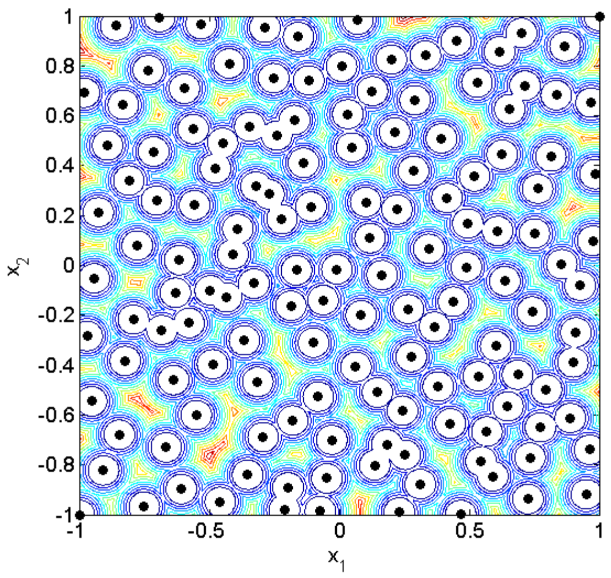


Figure 2.3: This figure shows a 2D space-filling design with a contour plot of the minimum distance of that location from all points in the design. The areas in white are areas that will lower the intersite distance and hence the quality of the design. These are the regions that lie closer to a point than the minimum intersite distance of the design. If the aggregate criterion from Equation 2.1 is used, any point chosen outside of these white areas will have the same score. It is however preferred to select a new point as far away as possible from existing points (the red areas), because this will result in an overall better space-filling design.

final designs be evaluated on the `idist` and `pdist` criteria to compare both approaches.

At each iteration, the Monte Carlo method will generate kn random points, where n is the number of samples evaluated thus far, and k is an algorithm parameter. It is expected that, for larger k, the quality of the design will improve. In this study, the following values for k were considered: $50, 250, 2000, 10000, 50000$.

For the genetic algorithm, the implementation from the Matlab Genetic Algorithm and Direct Search Toolbox (version 3.0) was used. Most of the options were kept at their default values, but some were changed in order to improve the performance. The default mutation function (which offsets each input by a value drawn from a Gaussian distribution) wasn't usable, because it did not respect the boundary constraints (each input must lie in $[-1,1]$). It was changed to a mutation function that changes each input with a probability of 0.01 to a random values in the $[-1,1]$ interval. Preliminary results have shown that playing with the crossover/mutation fraction settings, changing the elite behaviour etc does not affect the outcome much, so these settings were kept at their default values. This experiment was repeated for different numbers of generations:

50, 100, 250, 1000, 2000. Because both the Monte Carlo and genetic algorithms use random numbers, each experiment was repeated 10 times to get a good average of the performance of the methods.

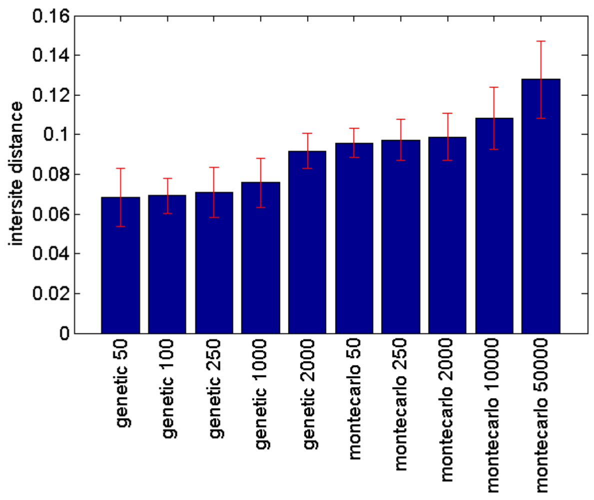
2.3.2 Results

The results for the intersite and projected distance values are shown in Figure 2.4. These plots show the average intersite and projected distance of each method, after generating 144 points in a 2D input space using the algorithms described in the previous section. It is clear that, in both cases, the Monte Carlo method outperforms the genetic algorithm. Figure 2.5 contains the aggregate score defined by Equation 2.3. For the aggregate score, the difference is even more pronounced, due to the fact that the Monte Carlo methods outperform the genetic algorithm in both the intersite and projected distance.

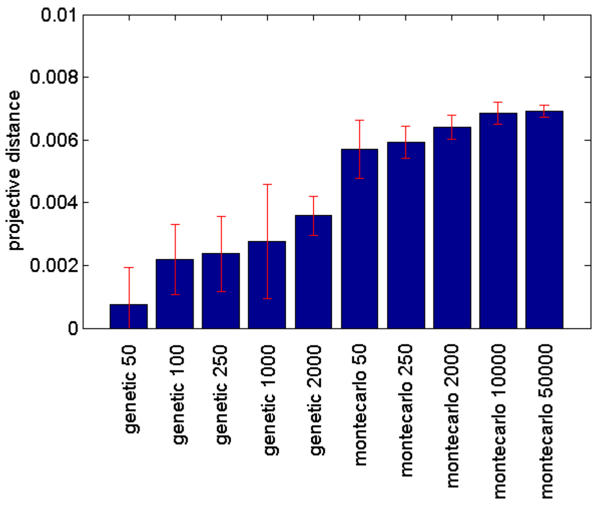
Figure 2.6 shows the time it took to generate the design for all the methods. Even though the final designs generated by the genetic algorithm are considerably worse than the ones generated by the Monte Carlo method, the genetic algorithm requires much more time to generate them. The genetic algorithm with 2000 generations takes a much longer time than the Monte Carlo method with $k = 50$, but still produces worse results. It is also noticeable that the difference between 50 generations and 2000 generations is smaller than the difference between $k = 50$ and $k = 50000$, while the difference in elapsed time is larger for the genetic algorithm. This indicates that the rate at which the genetic algorithm improves is actually lower than the improvement rate for the Monte Carlo method. So no matter how many generations are computed, there will always be a Monte Carlo alternative that requires less time to get the same result.

It is clear that genetic algorithms (and optimization methods in general), which are usually considered a better choice than a naive Monte Carlo approach, perform worse in this test case. In order to understand why this is happening, Figure 2.7 shows the optimization surfaces of the intersite distance, projected distance and the sum of these two (as defined by Equation 2.3), for 20 2D points spread out in a space-filling manner. The intersite distance produces an optimization surface with a considerable number of local optima. But this does not even come close to the number of local optima for the projected distance. In fact, the projected distance surface always has $(n+1)^d$ local optima, and only one of them is the global optimum. This optimization surface is so difficult, that it is practically impossible to optimize in an acceptable timeframe. When these two criteria are combined, the resulting optimization surface is even more erratic. This explains why the genetic algorithm quickly gets stuck in a local optimum, and does not manage to get out of it, no matter how many generations are computed.

Due to the nature of this optimization surface, the new methods proposed in this chapter will either use a Monte Carlo approach, or will avoid optimizing the aggregate surface directly. This can be achieved by only performing local optimizations as a fine-tune step after a Monte Carlo search, or by using the structure of the optimization surface to perform a more directed search for the global optimum.



(a) Intersite distance



(b) Projected distance

Figure 2.4: These figures show the average intersite and projected distance of each method, after generating 144 points in a 2D input space using the algorithms described in Section 2.3.1. Each experiment was repeated 10 times, and the standard deviation is shown as well. It is clear that in both cases, the Monte Carlo method performs considerably better than the genetic algorithm.

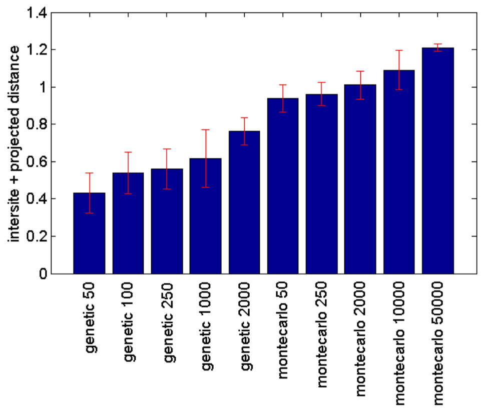


Figure 2.5: This figure show the aggregate intersite and projected distance score of each method as defined in Equation 2.3 after generating 144 points in a 2D input space using the algorithms described in Section 2.3.1. Each experiment was repeated 10 times, and the standard deviation is shown as well. It is clear that the Monte Carlo method performs considerably better than the genetic algorithm.

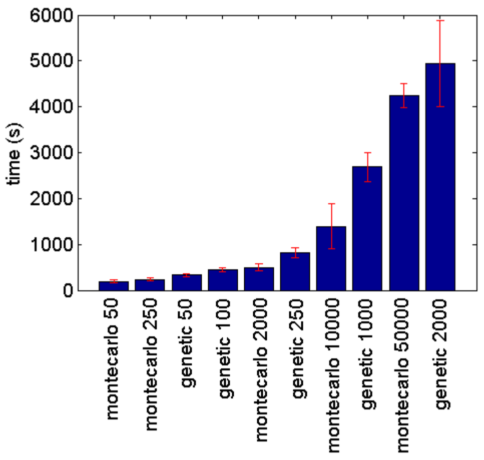
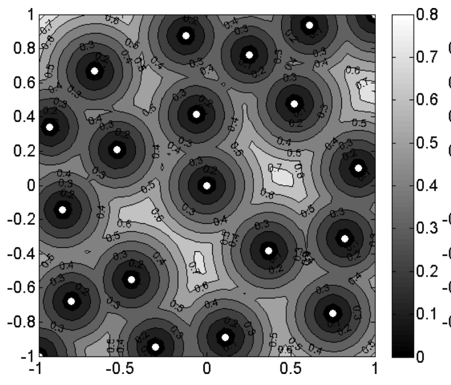
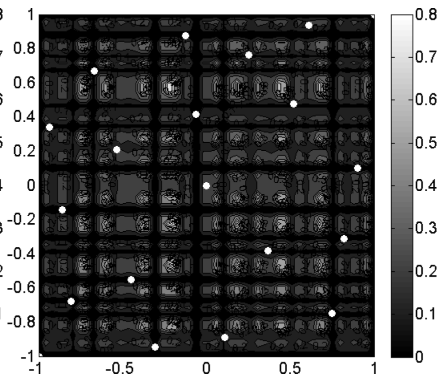


Figure 2.6: This figure shows each experiment, sorted by the average time it took to generate a 144-point experimental design. Note that the genetic algorithm requires much more time, while producing worse designs.


(a) Intersite distance


(b) Projected distance

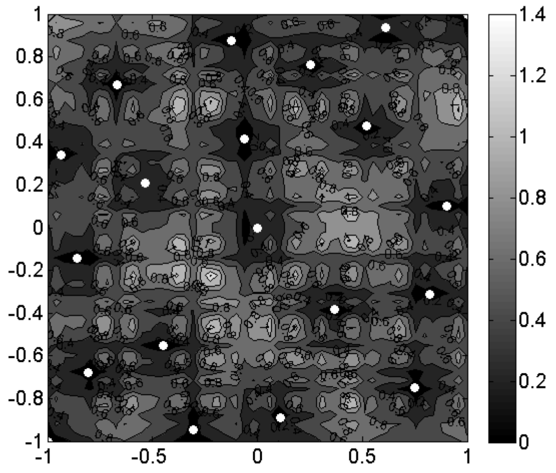

(c) Intersite + projected distance

Figure 2.7: The optimization surfaces for the intersite and projected distance criteria, as well as the sum of both criteria, for 12 points in 2D space. Due to the large number of local optima, optimization methods have a lot of trouble finding a globally optimal solution.

2.4 Existing methods

In this section, we will discuss existing methods that will be compared in this study. Both non-sequential methods, which have favourable properties in one or more of the criteria described in the previous section, as well as sequential methods from different fields of study will be investigated.

Each method will be given a name which will be used later in the discussion to refer to that particular strategy. Note that the design space is a hypercube with range $[-1,1]^d$, as opposed to the unit hypercube $[0,1]^d$, which is sometimes used in other studies. This has no effect on any of the algorithms, but may change some of the formulas.

2.4.1 Factorial designs

Factorial designs are the simplest form of space-filling designs [71]. A full factorial design is a grid of m^d points. The full factorial design is the best possible design in terms of the space-filling criterion; it maximizes the intersite distance for every number of m^d points. It is therefore expected that, if all the design parameters are equally important, a full factorial design will produce the best results when used to train a model.

However, it has several important disadvantages, which limit its practical use. Firstly, it is a very coarse-grained method: the full factorial can only be defined for the dth power of an integer m, which must be determined in advance. The only way to sequentialize a full factorial design is by evaluating the entire factorial design, and refine the grid in subsequent steps, as depicted in Figure 2.8. This increases the size of the design by almost a factor 2^d at each iteration. Secondly, a factorial design has the worst possible projective properties: if one of the design parameters is unimportant, each unique point is evaluated m times. This is an unacceptable risk in a black box setting.

To this end, several methods have been developed based on the factorial design, which tackle some of these issues. Fractional factorial designs remove some of the points from the grid, in order to limit the number of samples, making them feasible in high dimensional problems where a full factorial design would take too much time to evaluate [10]. Latin hypercubes, which are discussed in the next section, can be considered a subclass of fractional factorial designs with additional requirements. In this study, the full factorial design with 12 levels will be considered for the 2-dimensional case (denoted as `factorial`), for a total of 144 points. The full factorial will be left out in higher dimensions because there is no full factorial with 144 points in these dimensions.

2.4.2 Latin hypercube

Latin hypercube designs (commonly denoted as LHDs [94]) are a very popular experimental design technique because of their well-understood mathematical properties, their ease of implementation and use and their speed. A Latin hypercube is constructed by dividing each dimension in the design space in m equally

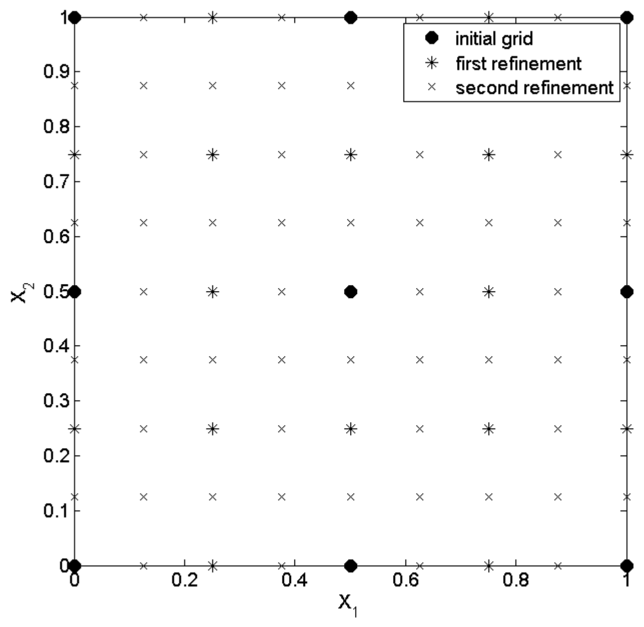


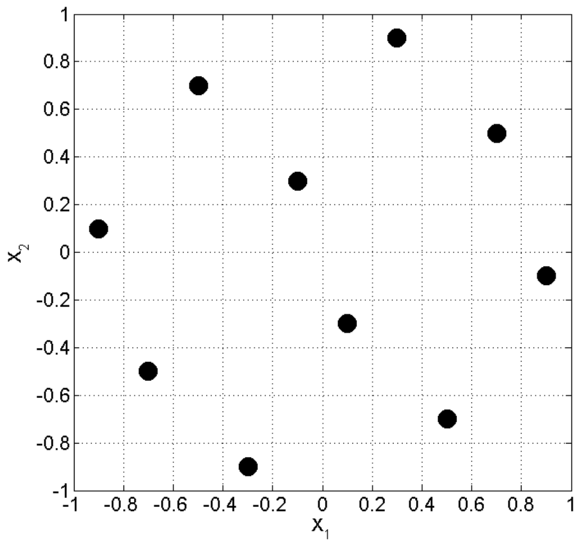
Figure 2.8: A factorial refinement scheme.

sized intervals, and placing exactly one point in each interval for each dimension. This construction method automatically results in an optimally non-collapsing experimental design. In addition, due to the stringent way in which the design space is divided, a Latin hypercube guarantees that each sample is at least $\frac{2}{m}\sqrt{2}$ away from the closest other sample in a $[-1,1]^d$ design space.

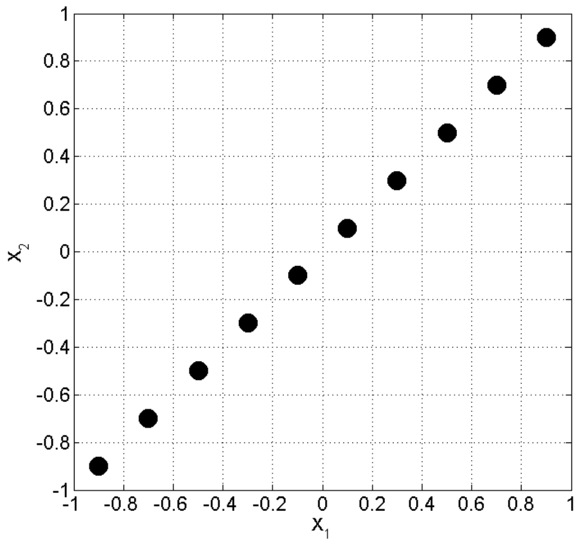
However, not every Latin hypercube has nice space-filling properties; this is illustrated in Figure 2.9. Therefore, Latin hypercubes should not be used blindly and should be optimized according to a space-filling criterion. The optimization of Latin hypercubes is a very active research field, and many methods have been developed to reduce the search space. For a good overview of Latin hypercube optimization techniques, please refer to [94].

The main problem with Latin hypercubes is that it is very difficult to generate a good space-filling Latin hypercube in reasonable time. Even with state-of-the-art optimization algorithms, constructing a good space-filling Latin hypercube can take hours or even days. Husslage et al. [50] report that constructing a 100-point Latin hypercube in 3 dimensions took between 145 and 500 hours on a P3-800MHz processor, depending on the algorithm used. For a larger number of points or higher dimensions, the computational time increases considerably.

To further demonstrate how difficult it is to generate a good Latin hypercube in a short timespan (e.g. 15 minutes), we included two different Latin hypercube



(a) Optimal Latin hypercube



(b) Bad Latin hypercube

Figure 2.9: Two different Latin hypercubes. While 2.9(a) has nice space-filling properties, 2.9(b) has only points in the diagonal and neglects two corners of the design space completely.

generation methods in this study. The first method is an implementation of the optimization algorithm described in [56] (denoted as `lhd-joseph`), which uses simulated annealing to optimize the intersite distance of the Latin hypercube. The second one uses the Matlab function `lhsdesign` from the Mathworks Statistics Toolbox to generate and optimize a Latin hypercube (`lhd-matlab`).

Additionally, we also included the pre-optimized Latin hypercubes from [50, 92], which can be downloaded from `http://www.spacefillingdesigns.nl`. All these Latin hypercubes were optimized for many hours to arrive at a semi-optimal or optimal solution. However, they are not available for every combination of dimensions and points. For our experiment, where we consider 144 points in 2 to 4 dimensions, a pre-optimized Latin hypercube is available on the website. We will refer to this Latin hypercube as `lhd-optimal`.

It is not straightforward to generate Latin hypercubes with a sequential design strategy. Firstly, the total number of samples must be determined in advance, in order to subdivide the design space into equally sized intervals. As mentioned before, this is an undesirable property, since there is little information available up front with which to make an educated guess on the required number of samples.

One way to sequentially generate Latin hypercubes is the idea of nested Latin hypercubes, in which one design is a subset of the other [50, 79]. By using this method, the smallest subset can be evaluated first, after which the decision can be made whether the samples in the superset have to be evaluated as well. This can be extended to a number of so-called layers, in which each layer is a Latin hypercube in itself and is also a subset of the next layer.

This approach is not very suitable for our purpose, because it is not fine-grained enough. The technique proposed by Qian et al. [79] only allows for the size of each superset to be a multiple of the size of its subset, while Husslage et al. [50] only consider two layers of nested designs. Because one simulation is assumed to be expensive, a sequential design algorithm should preferably select samples one by one. Therefore, methods based on nested Latin hypercubes will not be included in this study. However, a similar, but new and more fine-grained method will be included in this study and is described in Section 2.5.3.

A second way to adapt Latin hypercubes to a sequential sampling process is to give up the requirement that each sample is placed in exact intervals, resulting in so-called quasi-Latin hypercubes. Van Dam et al. [92] define a class of quasi-Latin hypercube designs in the design space $[0, n-1]^d$, based on a parameter $\alpha \in [0, 1]$, which defines the minimum distance of samples from each other when projected onto one of the axes. In the case of $\alpha = 0$, this reduces to an unconstrained maximin design, while $\alpha = 1$ results in traditional Latin hypercubes. It was shown that the α value can be relatively close to 1 without reducing the space-filling qualities of the design much. Xiong et al. [96] proposed a sequential design strategy in which the minimum projected distance of points from each other is reduced dynamically as more points are added to the design. A variation on this method will be included in this study, and will be described in Section 2.5.4.

2.4.3 Low-discrepancy sequences

Low-discrepancy sequences are sequences of points with the property that, for each n, the points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ have a low discrepancy. A set of points has a low discrepancy if the number of points from the dataset falling into an arbitrary subset of the design space is close to proportional to a particular measure of size for this subset. Several definitions exist for the discrepancy, based on the shape of the subset, and the measure which is used. For more information on low-discrepancy sequences and different definitions for discrepancy, refer to [49, 54, 73]. Low-discrepancy sequences are also called quasi-random sequences or quasi-Monte Carlo methods, because they can be used as a replacement for random uniform sampling.

Popular low discrepancy sequences have relatively good non-collapsing properties by construction. However, for small numbers of n, their space-filling properties are often subpar. Additionally, for some popular low-discrepancy sequences, such as the Hammersley sequence, the total number of points must be known in advance, because the points that are generated depend on the total number of points. So, for different values of n, completely different point sets are generated. Because these sequences are not suitable as a sequential design method, they will be ommited from this study.

Two of the most popular sequences that do not depend on the total number of samples are the `Halton` sequence and the `Sobol` sequence. Figure 2.10 shows the points generated by the Halton sequence in 2D for respectively 10, 50 and 144 points. The implementation of these sequences that is available in the Matlab Statistics Toolbox will be included in this study.

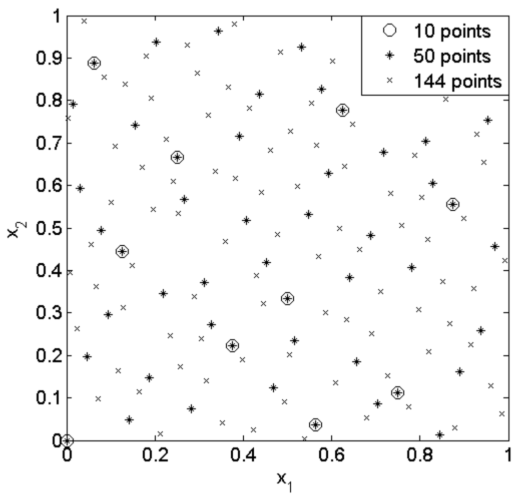


Figure 2.10: The first 144 points of the Halton sequence in 2D.

2.4.4 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. However, for very large sample sets, random sampling is actually a viable method, because it is extremely fast and available everywhere.

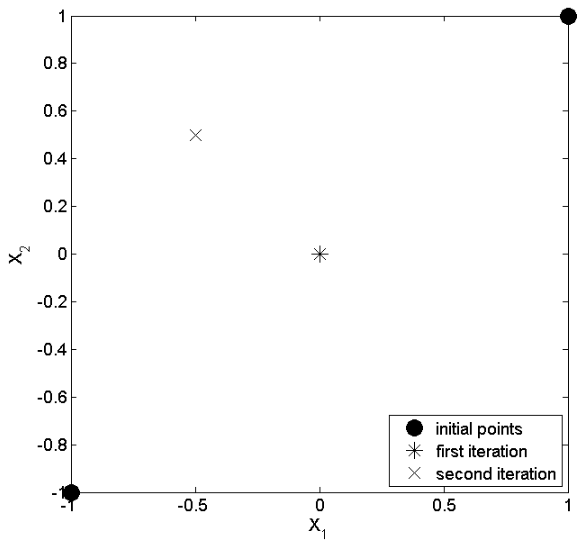
2.5 New space-filling sequential design methods

In this section, we propose a number of new space-filling sequential design methods that attempt to generate a design that scores well on both the intersite and projected distance criterion, while being as fine-grained as possible (each method selects samples one by one). The goal of this study is to develop an algorithm that generates a design sequentially (one by one), with intersite and projected distance as close to the best non-sequential methods as possible. Additionally, this algorithm must run relatively quickly (at most 15 minutes for 144 points in 2D). This study was executed on an Intel quadcore running at 1.86GHz.

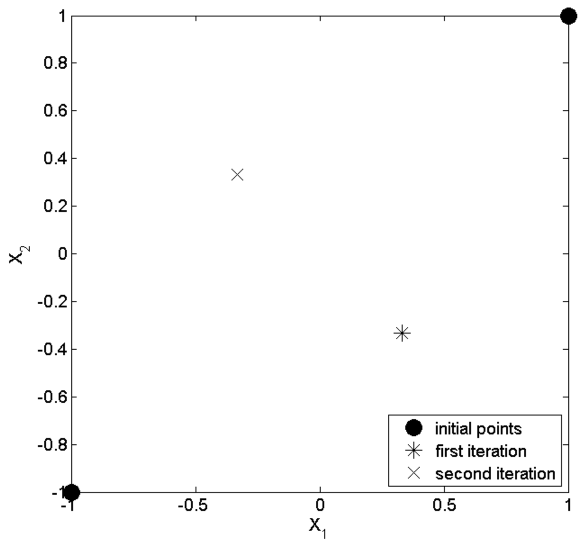
Because the new methods are sequential, they have to make do with a lot less information than their non-sequential counterparts (namely, the total number of samples is unknown in advance). Of course, this comes at a cost, and it is therefore expected that all the sequential methods will perform worse than pre-optimized Latin hypercube designs or factorial designs. However, if the drop in intersite distance and projected distance is small, these methods should be preferred over one-shot methods in a black box environment, because they can avoid over- and undersampling, thus potentially saving lots of computational time. Additionally, some of the proposed methods will also work for very large n, for which optimizing a Latin hypercube is infeasible, and will also work for high dimensional problems.

At each iteration of a sequential design algorithm, the algorithm must determine the optimal location for the new sample point, based on the previously evaluated points. This new point must be located in such a way as to maximize the intersite and projected distance of the resulting design, which is composed of the original points and the new point. However, the new point must also ensure that future iterations of the algorithm can still produce good designs. Even if a point is optimally chosen for intersite and projected distance at one iteration, it might cause the algorithm to get stuck on a local optimum in subsequent iterations. This is illustrated in Figure 2.11. This figure shows two 2D designs which were generated from the same set of two initial points: $(-1,-1)$ and $(1,1)$. The first algorithm places the third point in the origin, while the second algorithm places it at $(-1/3, 1/3)$. After the third point, the first algorithm has produced the best design, both in intersite and projected distance. However, it is now stuck in a local optimum, as the best possible choice for the fourth point results in a design considerably worse than the one for the second algorithm.

This problem is further compounded by the difficult optimization surfaces produced by the intersite and projected distance, as shown in Section 2.3. Due to the extremely complex nature of this optimization surface, all the new methods proposed in this chapter avoid working with this surface directly, by exploiting the structure of the projected distance surface, or by resorting to Monte Carlo methods instead of optimization. In the next sections, the new methods will be discussed in detail.



(a) First algorithm



(b) Second algorithm

Figure 2.11: Two different sequential design algorithms generate a 4-point design starting from the same two initial points. The first algorithm gets stuck in a local optimum after the third point, while the second algorithm does not.

2.5.1 Voronoi-based sequential design

An exploration criterion must accurately and efficiently identify the region of the design space that contains the lowest density of samples. This can be done in many different ways, depending on the density measure used and on the allowed computational complexity of the algorithm.

A Voronoi tessellation (or Voronoi diagram) is an intuitive way to describe sampling density. Assume a discrete and pairwise distinct set of points $P \subset \mathbb{R}^d$ in Euclidean space, which represents the existing data points. For any point $\mathbf{p}_i \in P$, the Voronoi cell $C_i \subset \mathbb{R}^d$ contains all points in $\mathbb{R}^d$ that lie closer to $\mathbf{p}_i$ than any other point in P. The complete set of cells $\{C_1, C_2, \dots, C_n\}$ tessellate the whole space, and is called the Voronoi tessellation corresponding to the set P.

To define a Voronoi tessellation more formally, we adopt the notation from [4]. For two distinct points $\mathbf{p}_i, \mathbf{p}_j \in \mathbb{R}^d$, the *dominance* of $\mathbf{p}_i$ over $\mathbf{p}_j$ is defined as the subset of the plane being at least as close to $\mathbf{p}_i$ as to $\mathbf{p}_j$. Formally,

$$\mathrm{dom}(\mathbf{p}_i, \mathbf{p}_j) = \left\{ \mathbf{p} \in \mathbb{R}^d \middle| \left\| \mathbf{p} - \mathbf{p}_i \right\| \leq \left\| \mathbf{p} - \mathbf{p}_j \right\| \right\}.$$

$\mathrm{dom}(\mathbf{p}_i, \mathbf{p}_j)$ is a closed half plane bounded by the perpendicular bisector of $\mathbf{p}_i$ and $\mathbf{p}_j$. This bisector, which we will call the *separator* of $\mathbf{p}_i$ and $\mathbf{p}_j$, separates all points of the plane closer to $\mathbf{p}_i$ than those closer to $\mathbf{p}_j$. Finally, the Voronoi cell C_i of $\mathbf{p}_i$ is the portion of the design space that lies in all dominances of $\mathbf{p}_i$ over all the other data points in P:

$$C_i = \bigcap_{\mathbf{p_j} \in P \setminus \mathbf{p}_i} \mathrm{dom}(\mathbf{p}_i, \mathbf{p}_j)$$

This is illustrated in Figure 2.12. Note that in this example, the Voronoi cells are noticeably smaller near the center 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 high: their Voronoi cells reach to infinity in their respective directions. This is a basic property of Voronoi tessellations.

2.5.1.1 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 $\mathrm{O}(n)$ time [4]. 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 2.13 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 [1], which is based on the Quickhull algorithm described in [6]. It is clear that the calculation of a Voronoi diagram scales terribly

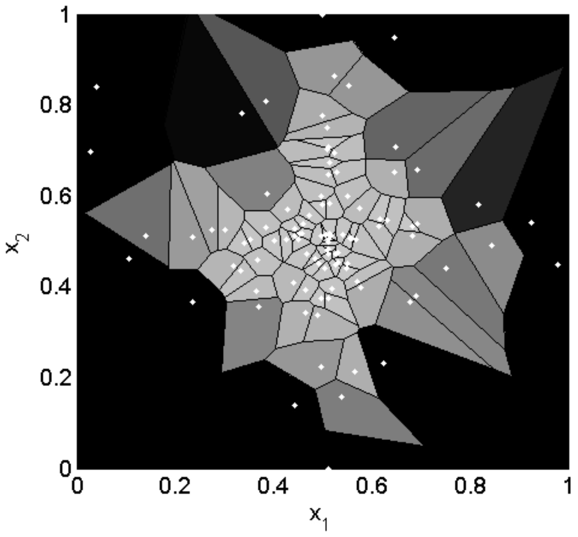


Figure 2.12: A set of data points and their Voronoi cells in a 2D design space. Larger Voronoi cells have a darker colour. The data points are drawn as white dots. Unbounded Voronoi cells are coloured black.

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 do not 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 is used. In order to estimate the volume of each Voronoi cell, a large number of random, uniformly distributed test points are generated in the domain. For each test point, the minimum distance from all the samples is computed, and the test point is assigned to the sample which is the closest. If enough test points are generated like this, the algorithm produces an estimation of the (relative) size of each Voronoi cell. This is described more formally in Algorithm 2.

2.5.1.2 Sampling strategy

Now that the Voronoi cell size is estimated to an acceptable accuracy, this information can be used to select new samples in undersampled regions of the design space. First, the Voronoi cell size is estimated for all the samples $\mathbf{p} \in P$. The sample with the largest Voronoi cell size is identified, and a number of random candidates are generated in the neighbourhood of this sample. These candidates are ranked, based on the maximin criterion, and the best candidate is selected as the new sample. At the next iteration of the sampling algorithm, the Voronoi cell size is esti-

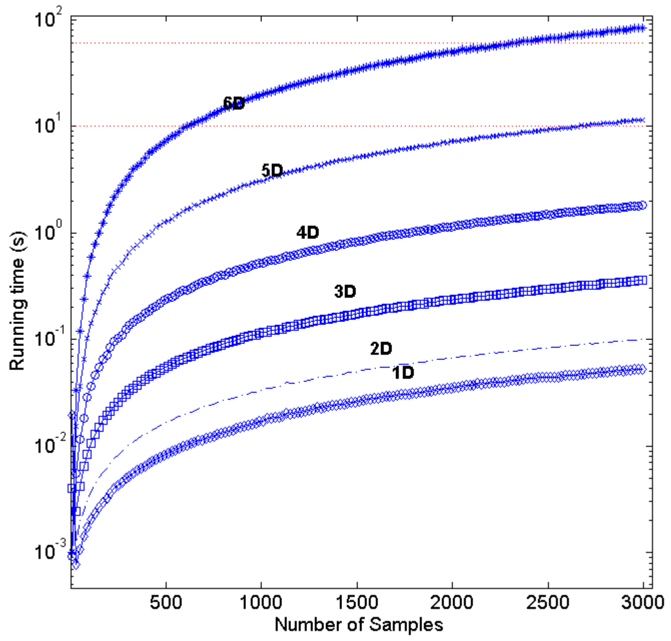


Figure 2.13: A performance plot of the Qhull package as a function of the number of samples and the dimensionality of the problem. It is clear that the algorithm scales relatively well with the number of samples, but poorly with the dimensionality of the problem. In practice, the algorithm is infeasible for problems with more than six dimensions.

mated again, and the process starts all over again. Of course, all the random points used in the Monte Carlo estimation of the Voronoi cell sizes are also considered as candidates, because their distance was already calculated anyway.

Basically, the Voronoi tessellation is used to reduce the design space to a small subset in which additional random candidates will be generated, by zooming in on those regions which have few samples. Instead of blindly generating random candidates in the entire design space, they are only generated in the largest Voronoi cell, which leads to a better design. This is an example of a *search-space reducing Monte Carlo method*. Random candidates are not generated in the entire design space, but in specifically, intelligently chosen regions of the design space instead. This increases the chances of finding the best possible location for the new sample.

2.5.1.3 Performance 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 optimal number, we have performed a large set of tests with different numbers, ranging

Algorithm 2 Estimating the Voronoi cell size. P is the set of samples that have to be ranked according to their respective Voronoi cell size.

```
$S \leftarrow 100|P|$ random points in the domain
$V \leftarrow [0,0,\dots,0]$
for all $\mathbf{s} \in S$ do
    $d \leftarrow \infty$
    for all $\mathbf{p} \in P$ do
        if $\|\mathbf{p}-\mathbf{s}\| < d$ then
            $\mathbf{p}_{\text{closest}} \leftarrow \mathbf{p}$
            $d \leftarrow \|\mathbf{p}-\mathbf{s}\|$
        end if
    end for
    $V[\mathbf{p}_{\text{closest}}] \leftarrow V[\mathbf{p}_{\text{closest}}] + (1/|S|)$
end for
```

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 (Bayesian Estimation Error Quotient) metric proposed in [65]. This is a relative error metric is defined as:

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

where V_i is the real volume of the ith Voronoi cell, $\tilde{V}_i$ is the estimated volume of the ith 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 2.14.

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 only 10 000 random samples, or 100 samples per data point, are required 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

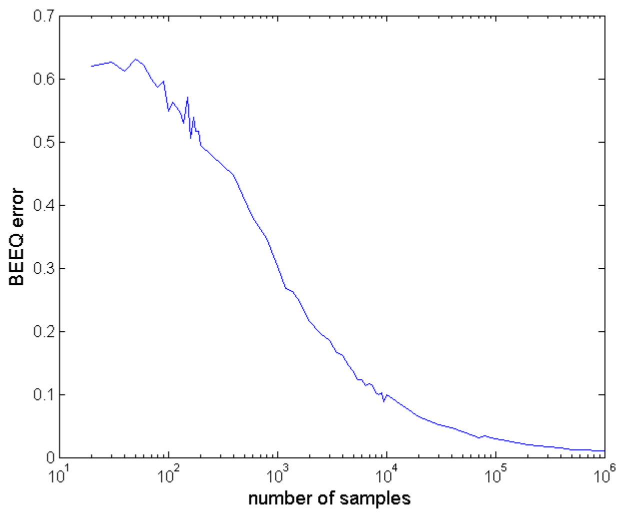


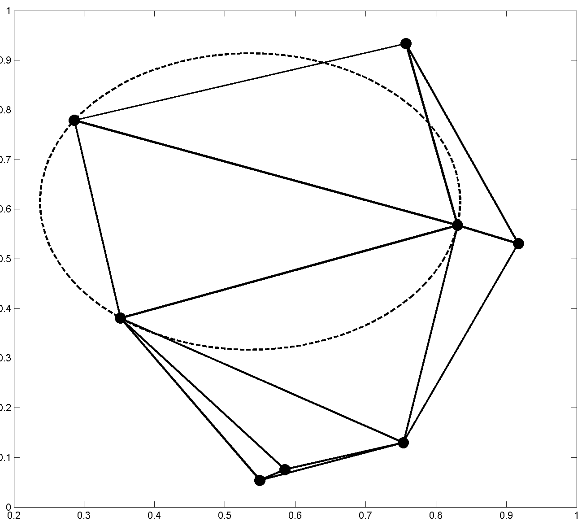
Figure 2.14: A benchmark of the accuracy of a Monte Carlo Voronoi approximation.

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.

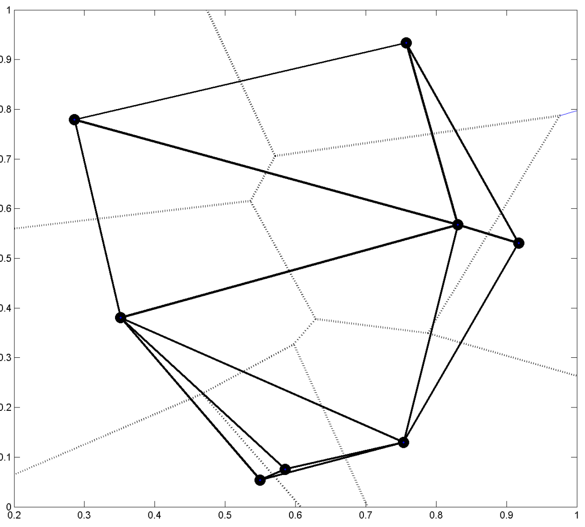
2.5.2 Delaunay-based sequential design

A Delaunay triangulation is a triangulation of a set of points P such that no point in P lies inside the circum-hypersphere of any n-simplex in the triangulation [21]. In two dimensions, this reduces to the circumcircle of each triangle. The Delaunay triangulation is the dual of the Voronoi diagram described in Section 2.5.1. One can be computed exactly from the other. The Voronoi tessellation for a given Delaunay triangulation can be computed by connecting the edges of the centers of the circum-hyperspheres of all the simplices. The Delaunay triangulation of a given Voronoi tessellation can be computed by adding an edge for every two points in P for which the Voronoi cells share an edge. This is illustrated in Figure 2.15.

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, the performance analysis performed in Section 2.5.1.1 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



(a) A Delaunay triangulation



(b) A Delaunay triangulation with its dual Voronoi tessellation

Figure 2.15: A Delaunay triangulation of a set of 2D points. In Figure 2.15(a), the triangulation is displayed along with one circumcircle. Note that no points lie within the circumcircle of any triangle of the triangulation. On Figure 2.15(b), the Delaunay triangulation is plotted along with its dual Voronoi tessellation.

Qhull implementation and not an approximation as is the case for Voronoi-based sequential design.

The idea behind Delaunay-based sampling is to find the centers of gravity of the largest simplices of the triangulation, and take these as new samples for the next iterations. 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:

$$V = \frac{1}{d!} \left| \begin{array}{cccc} v_2 - v_1 & v_3 - v_1 & \ldots & v_{d+1} - v_1 \end{array} \right| \quad \text{where} \quad v_i = \begin{bmatrix} v_i^1 \\ v_i^2 \\ \vdots \\ v_i^d \end{bmatrix} \tag{2.5}$$

The simplices are ranked according to their volume. If n_{new} samples need to be selected, n_{new} simplices with the largest volume are selected. For each of these 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. This method will be called `delaunay`.

2.5.3 Sequential nested Latin hypercubes

A more fine-grained variant of the nested Latin hypercube method described in Section 2.4.2 was also included in this study. In order to sequentially generate Latin hypercube-like designs, one could refine the grid on which the points are chosen, similar to the idea of the factorial refinement scheme proposed in Figure 2.8. Figure 2.16 shows a refinement scheme for Latin hypercubes. Starting from an initial grid of m^d candidate points, m new samples are iteratively chosen on this grid. When a new sample is selected, all the other candidate points on the grid that have the same value for one of the design parameters are removed from the grid and will not be selected later. When m points have been selected, a new grid is created at the midpoints between the samples, and the process is repeated, thus (asymptotically) doubling the grid size at each iteration.

To determine which candidate point will be chosen next, the distance of each candidate point on the grid from all the previously selected points is computed. The candidate point that lies the farthest away is selected as the next sample, and all the other candidates that share one of the design parameter values with this sample are removed from the grid. Because the search space only contains the points on the grid instead of the entire design space, it is feasible to compute the distance for all the candidate points, without having to resort to optimization or Monte Carlo methods. This method is called `lhd-nested`.

This method results in an exact Latin hypercube when exactly $m+(m-1)(2^p-1)$ samples have been selected, for $p > 0$. At these iterations, which depend solely on the initial grid size m, the $\|P\|_{-\infty}$ score is maximal. In the worst case, which is when $1+m+(m-1)(2^p-1)$ samples have been selected, the $\|P\|_{-\infty}$ score is almost half of the optimal score. When the total number of samples is known in advance, the number m can be tweaked such that the total number of samples is close to

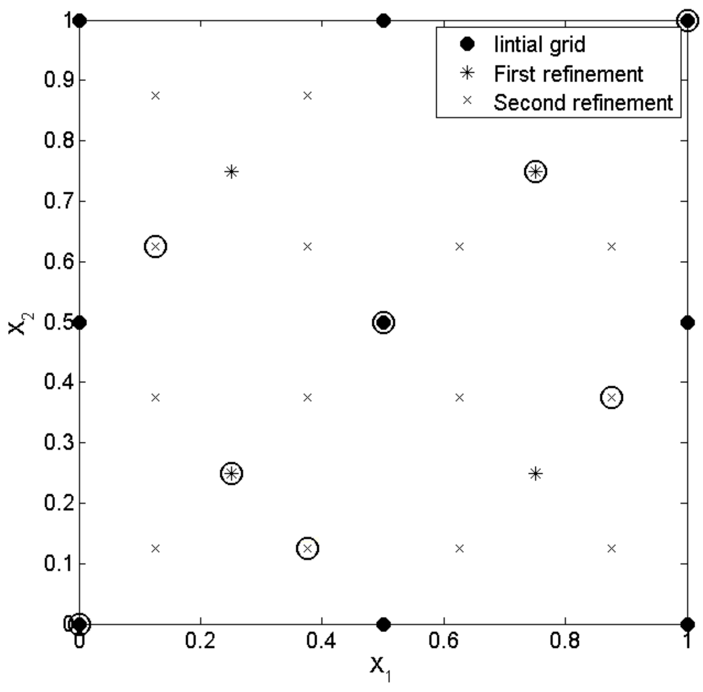


Figure 2.16: A Latin hypercube refinement scheme, starting with $m = 3$. The points highlighted with a circle are the ones that were chosen by the sampling algorithm. Note that the design, composed of the encircled points, forms a Latin hypercube, and therefore has optimal projected distance.

but not larger than $m + (m-1)(2^p - 1)$. However, in this study, this information is considered unknown, so m is fixed at 2.

2.5.4 Global Monte Carlo methods

A Monte Carlo method is a method that relies on repeated random sampling to compute the results. In the context of sequential design, Monte Carlo methods generate a large number of random candidate points in the design space, compute a criterion for all of these points, and select the point with the best (highest) score on the criterion as the next sample to be evaluated. This is repeated at each iteration.

2.5.4.1 Intersite-projected distance criterion

Two different criteria to rank the random points were considered in this study. The first criterion that was used is the aggregate of the intersite and projected distance `dist`, defined in Equation 2.4, which ranks a point according to its (weighted) intersite and projected distance from all other points. For convenience, the definition of this criterion is repeated:

$$\text{dist}(P,\mathbf{p}) = \frac{(n+1)^{\frac{1}{d}-1}}{2}\min_{\mathbf{p}_i\in P}\sqrt{\sum_{k=1}^{d}\left|p_i^k - p^k\right|^2} + \frac{n+1}{2}\min_{\mathbf{p}_i\in P}\left\|\mathbf{p}_i-\mathbf{p}\right\|_{-\infty}$$

At each iteration, the point which maximizes this criterion will be picked as the next point. This method will be referred to as `mc-intersite-proj`.

2.5.4.2 Search space reduction

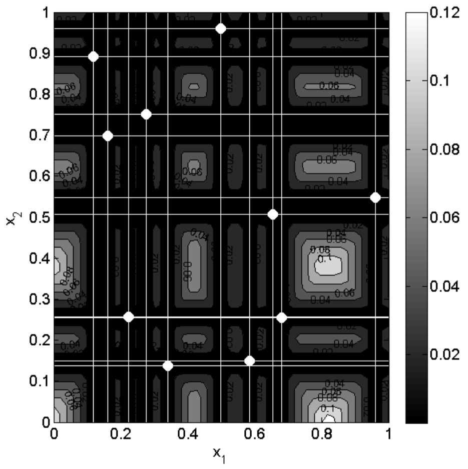
Note that, in the previous section, points are still ranked based on the complex surface shown in Figure 2.7(c). An alternative is to consider the projected distance as a threshold function. The idea is to discard points that lie too close to other points in terms of projected distance. All the remaining points are then ranked solely on intersite distance. This is similar to the idea of quasi-Latin hypercube designs proposed in [92]. The difference between the standard projected distance criterion and the threshold projected distance criterion is shown in Figure 2.17. In the case of the threshold criterion, only random points in the white areas are considered, and the best candidate in these areas based on the intersite distance is selected as the next sample.

The threshold, or minimum allowed projected distance, is defined as:

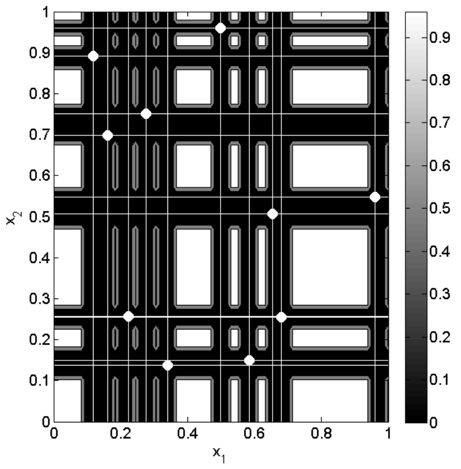
$$d_{\min} = \frac{2\alpha}{n} \tag{2.6}$$

where α is a tolerance parameter, which defines the importance of the projected distance. The objective function for the threshold version of Equation 2.4 is defined as follows:

$$\begin{aligned}&\texttt{intersite-proj-th}(P,\mathbf{p})\\ &= \begin{cases}0 & \text{if } \min_{\mathbf{p}_i\in P}\left\|\mathbf{p}_i-\mathbf{p}\right\|_{-\infty} < d_{\min}\\ \min_{\mathbf{p}_i\in P}\left\|\mathbf{p}_i-\mathbf{p}\right\|_2 & \text{if } \min_{\mathbf{p}_i\in P}\left\|\mathbf{p}_i-\mathbf{p}\right\|_{-\infty} >= d_{\min}.\end{cases}\end{aligned} \tag{2.7}$$



(a) Projected distance



(b) Threshold projected distance

Figure 2.17: The optimization surfaces for projected distance and threshold projected distance criteria.

If $\alpha = 0$, there are no constraints, and the projected distance is not taken into account at all. If $\alpha = 1$, only points that lie exactly on an optimal configuration are considered. In practice, this means that all points are rejected, because the candidates are generated randomly. The trade-off between intersite and projected distance is illustrated in Figure 2.18. For this experiment, $\alpha = 0.5$ was chosen because it results in a good trade-off between intersite and projected distance. The method using this objective function for ranking the candidate points will be referred to as `mc-intersite-proj-th`.

This method can be further fine-tuned by adapting the algorithm to only generate random points in areas that fall outside of the threshold region, instead of eliminating the points after generation. This further improves the efficiency of this method. This is another example of a *search-space reducing Monte Carlo method*, such as the Voronoi-based sequential design strategy. Instead of finding the best Monte Carlo sample in the entire design space, the design space is reduced to certain regions which already have favourable properties. In this case, these are the white areas in Figure 2.17(b).

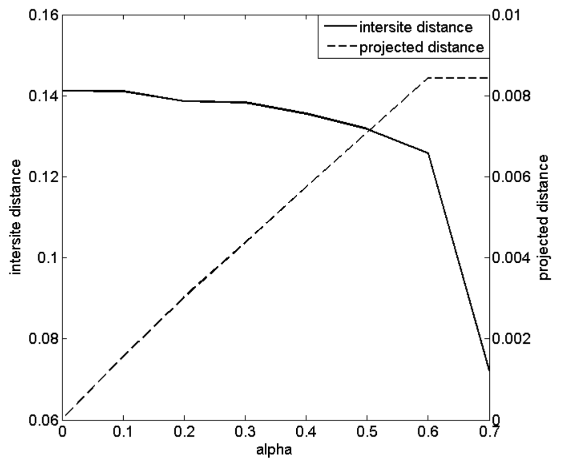


Figure 2.18: The effect of the α parameter from the `mc-intersite-proj-th` algorithm on the intersite and projected distance. Lower values of α favour intersite distance, while higher values of α favour projected distance. For $\alpha = 0.5$, a good trade-off is found between intersite and projected distance.

2.5.4.3 Performance analysis

An essential parameter to any Monte Carlo method is the number of random points to generate. This number does not have to be same between subsequent iterations. Indeed, it makes sense to scale up the total number of random points as

P increases in size. Intuitively, more random candidates should be needed when the total number of previously evaluated samples is larger.

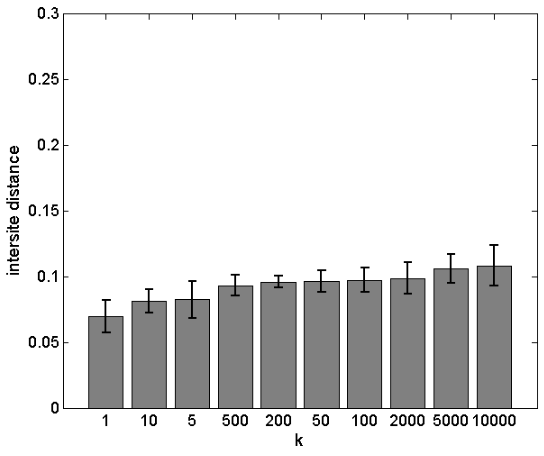
As discussed in Section 2.3, the Monte Carlo method will generate kn random points at each iteration, where n is the number of samples evaluated thus far, and k is an algorithm parameter. Figure 2.4 already showed that increasing k will improve the final result. In order to determine the optimal choice for k, the experiment was repeated for additional values of k, up to a total of 144 samples (so $143k$ random points will be generated for the last iteration). For each value of k, 10 experiments were run to get a good average of the performance. Additionally, the experiments were also conducted in 3D, in order to see how the dimensionality affects the choice of k. The results for the 2D case are summarized in Figure 2.19 for the `mc-intersite-proj` method and in Figure 2.20 for the `mc-intersite-proj-th` method.

Surprisingly, the parameter k has very little effect on the quality of the final design. For `mc-intersite-proj`, increasing k will increase the average quality of the design, but the difference between $k = 50$ and $k = 2000$ is rather small, while the runtime is considerably longer. If time is a concern, a relatively low value of k around 100 should be more than sufficient for acceptable intersite distance. For the projected distance, the differences are larger, but once again $k = 50$ does not perform much worse than $k = 2000$. In this use case, we therefore suggest $k = 100$ as a good choice for stable, fast designs, without wasting too much computational time. Increasing k by a factor 50 or 100 will only yield a marginal advantage. The reason why a low value for k is acceptable, is because even for $k = 5$, a 2D design space in a $[-1,1]^2$ square will already be covered very well once n goes up. This is shown in Figure 2.21.

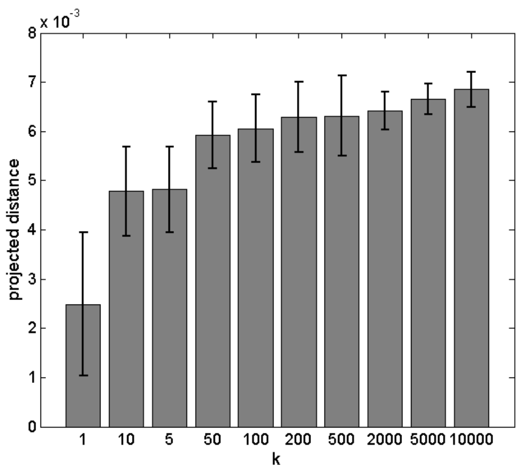
For `mc-intersite-proj-th`, the results are even more striking. Beyond $k = 50$, the difference in quality is almost negligible. Increasing k beyond this point will yield no advantage whatsoever. Additionally, even for $k = 1$, the design is considerably better than `mc-intersite-proj` with $k = 10000$. This can be explained by the threshold property of `mc-intersite-proj-th`. Points that lie in the regions that do not violate the threshold already have, by definition, a minimum distance of $\sqrt{2}d_{\min}$ from all the samples so far. By restricting the randomly generated points to these areas, every candidate point already lies an acceptable distance from existing samples. Picking any single random point from these viable areas will already yield a good design! Because of the threshold requirement, the k parameter does not affect the projected distance at all. Any value for k will result in the same projected distance score; in order to change this, the α parameter should be changed.

In the 3D case, the results are somewhat different. The results for the 3D case are summarized in Figure 2.22 for the `mc-intersite-proj` method and in Figure 2.23 for the `mc-intersite-proj-th` method. In 3D, the parameter k has a much larger effect than in 2D. This can be explained by the fact that the design space is much larger; more random points are needed to cover it adequately.

For `mc-intersite-proj`, a noticeable increase in both the intersite and especially the projected distance can be noted as k is increased. This seems to cap again somewhat at $k = 2000$, indicating that at this point, the 3D design space is

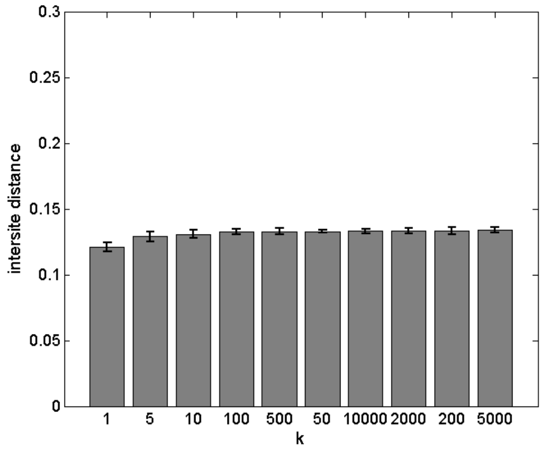


(a) Intersite distance for `mc-intersite-proj` in 2D

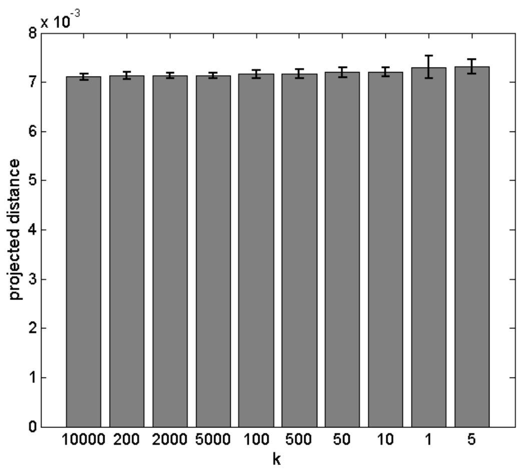


(b) Projected distance for `mc-intersite-proj` in 2D

Figure 2.19: Respectively the intersite and projected distance as a function of the parameter k, which represents how many random candidates are generated each iteration of the Monte Carlo method `mc-intersite-proj`. This graph shows how the performance of the `mc-intersite-proj` method scales in 2D with the parameter k.



(a) Intersite distance for `mc-intersite-proj-th` in 2D



(b) Projected distance for `mc-intersite-proj-th` in 2D

Figure 2.20: Respectively the intersite and projected distance as a function of the parameter k, which represents how many random candidates are generated each iteration of the Monte Carlo method `mc-intersite-proj-th`. This graph shows how the performance of the `mc-intersite-proj-th` method scales in 2D with the parameter k.

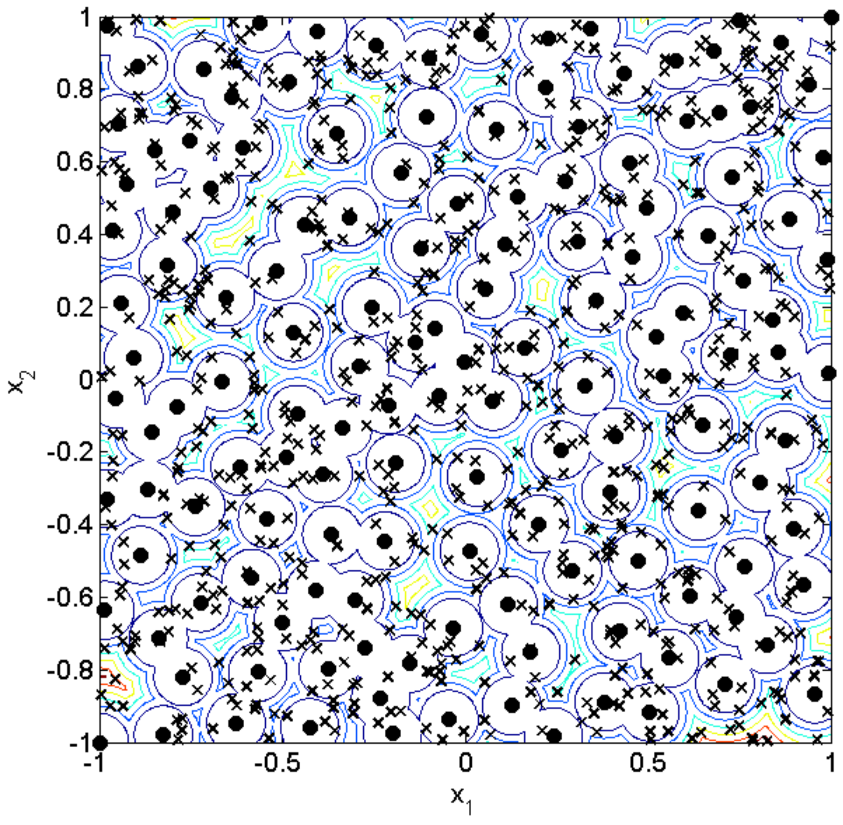
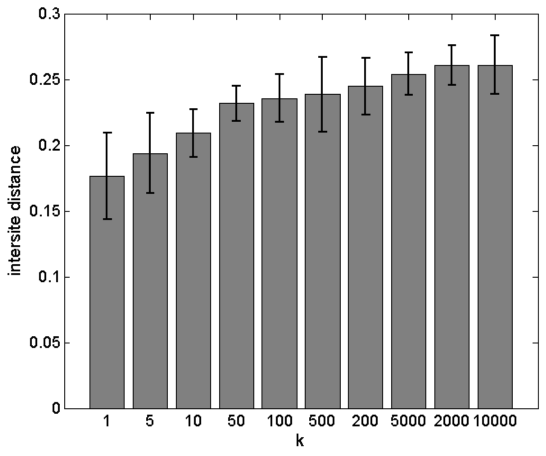


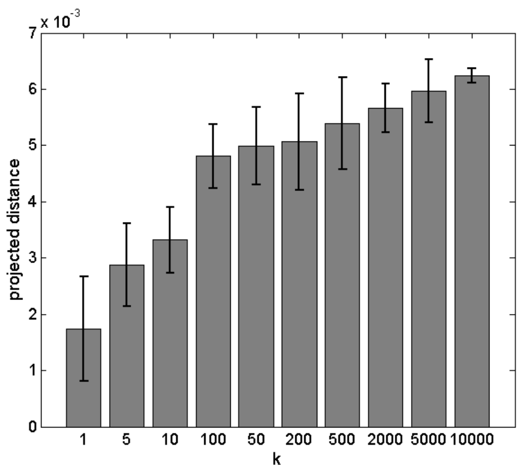
Figure 2.21: A random sample for $n = 143$ and $k = 5$. The circles are the existing 143 samples, the crosses are the kn randomly generated candidates. In the background, a contour plot of the resulting maximin criterion after adding the 144th point is shown. Note that plenty of candidate points are found in the red areas, which are the best choice in terms of intersite distance.

sufficiently covered. It is expected that, for $k > 10\,000$, the quality of the design will eventually stabilize and not improve any further. Again, $k = 50$ seems like a good choice, as this is the point where the clear lack of coverage seen in lower k values seems to disappear. However, due to the large variance in the results, it might be better to pick a larger k for the 3D case. For excellent performance, $k = 2\,000$ seems like a very good choice. If more speed is required, $k = 200$ is a good trade-off between speed and stability.

While `mc-intersite-proj` does seem to suffer quite a bit from the additional dimension, `mc-intersite-proj-th` remains surprisingly stable. While k does have a larger effect on the intersite distance than in the 2D case, the effect

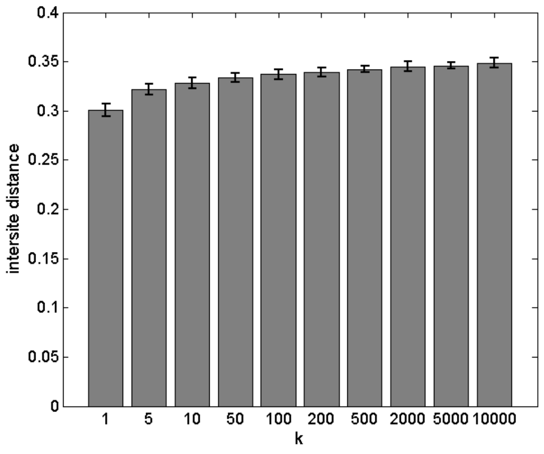


(a) Intersite distance for `mc-intersite-proj` in 3D

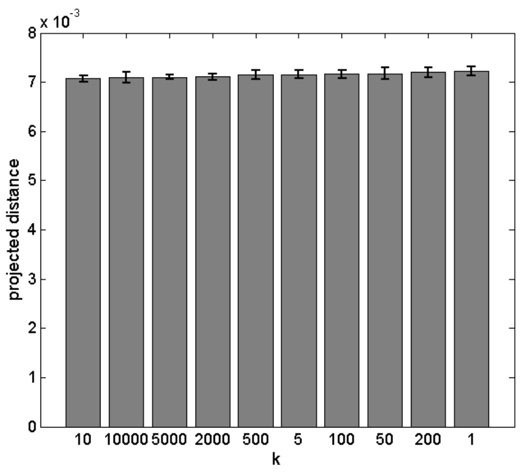


(b) Projected distance for `mc-intersite-proj` in 3D

Figure 2.22: Respectively the intersite and projected distance as a function of the parameter *k*, which represents how many random candidates are generated each iteration of the Monte Carlo method `mc-intersite-proj`. This graph shows how the performance of the `mc-intersite-proj-th` method scales in 3D with the parameter *k*.



(a) Intersite distance for `mc-intersite-proj-th` in 3D



(b) Projected distance for `mc-intersite-proj-th` in 3D

Figure 2.23: Respectively the intersite and projected distance as a function of the parameter k, which represents how many random candidates are generated each iteration of the Monte Carlo method `mc-intersite-proj-th`. This graph shows how the performance of the `mc-intersite-proj-th` method scales in 3D with the parameter k.

is still very small. The difference between $k = 50$ and $k = 10\,000$ is negligible, and once again even `mc-intersite-proj-th` with $k = 1$ performs better than `mc-intersite-proj` with $k = 10\,000$. Again, the projected distance is the same for all k values, and is also the same as in the 2D case.

It is very clear from these results that `mc-intersite-proj-th` is by far the superior method, demonstrating that it is highly stable, requires very little random points for optimal results and scales very well to high dimensions. Both methods, however, can produce very good designs for relatively small values of k.

2.5.5 Optimization-based methods

Even though global optimization methods do not seem to work well for this problem, local optimization methods might still be able to deliver a considerable improvement when used after a Monte Carlo method. With this in mind, we propose two additional algorithms that perform a fast, constrained, local optimization after generating a large number of points, either based on a Monte Carlo method or based on the structure of the projected distance surface. We opted for the pattern search function from the Genetic Algorithm and Direct Search Toolbox of Matlab as the optimizer of choice, since it is a relatively fast but good optimizer that can get out of local optima quite easily.

2.5.5.1 Optimize projected distance locally

The first algorithm uses Monte Carlo to find the best points for the intersite distance, and then locally optimizes the best candidates for the projected distance, effectively giving up some intersite distance in exchange for better projected distance results. This method will be called `optimizer-proj`. Pseudocode for this method can be found in Algorithm 3.

First, the algorithm selects a large amount of random points, and computes the intersite distance for all of these points. The 30 highest scoring points are selected as potential candidates, and the minimum distance from all the previously evaluated points is computed for these candidates. This distance is multiplied by a factor β which determines how much the optimizer may deviate from the selected candidate locations to improve the projective properties of the candidate. This is illustrated in Figure 2.24.

If β is set to 0, the algorithm selects points based solely on the intersite distance. If β is set to 1, the algorithm completely abandons the intersite distance and optimizes completely towards the projected distance. This trade-off is illustrated in Figure 2.25. The β parameter effectively specifies how much space-fillingness the user is willing to give up for better projective properties. For this experiment, $\beta = 0.3$ was chosen because it provides a good trade-off between the two criteria.

Algorithm 3 The `optimizer-proj` algorithm.

```
$P_{\text{candidates}} \leftarrow 100n$ random points
$P_{\text{new}} \leftarrow 30$ best points using intersite distance
for all $\mathbf{P}_{\text{new}} \in P_{\text{new}}$ do
    $m(\mathbf{P}_{\text{new}}) \leftarrow min_{\mathbf{p} \in P} \left\| \mathbf{P}_{\text{new}} - \mathbf{p} \right\|_2$
    $d_{\max} \leftarrow \frac{\beta m(\mathbf{P}_{\text{new}})}{2}$
    Optimize $\mathbf{P}_{\text{new}}$ towards $\| P \cup \mathbf{P}_{\text{new}} \|_{-\infty}$ on $[\mathbf{P}_{\text{new}} - d_{\max}, \mathbf{P}_{\text{new}} + d_{\max}]$
end for
Choose best $\mathbf{P}_{\text{new}}$ based on $\| P \cup \mathbf{P}_{\text{new}} \|_{-\infty}$
```

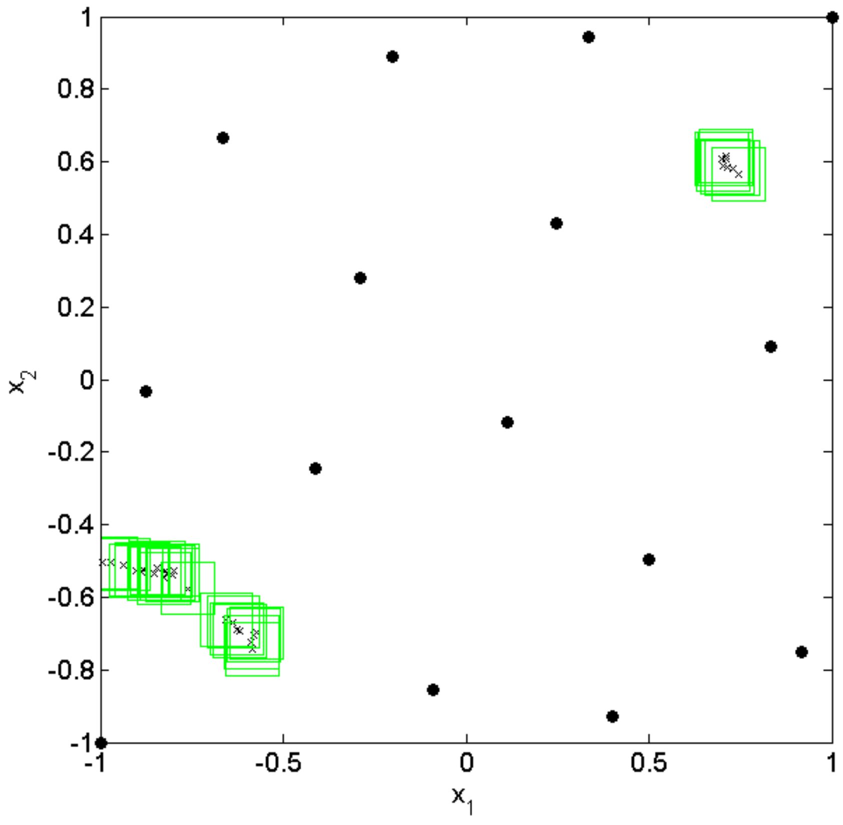


Figure 2.24: In this figure, 15 points (displayed by circles) are already generated in a space-filling manner by the algorithm. In order to determine the optimal location for the next point, a large set of random candidates is generated, and the 30 best are selected based on the intersite distance (displayed as crosses). For each of these 30 candidates, an optimization area is determined (defined by the rectangles) in which the optimizer may deviate in order to improve the projected distance. Each candidate is optimized, and the best candidate in terms of projected distance is picked as the next sample.

2.5.5.2 Optimize intersite distance locally

Even though the optimization surface of the $\|P\|_{-\infty}$ criterion is highly multimodal, it is also very structured, and the optima can easily be derived from the samples without having to use an optimization algorithm. This structure was already used by `mc-intersite-proj-th` to only generate Monte Carlo points in those regions which satisfy a threshold parameter α. Another way to use this structure is by

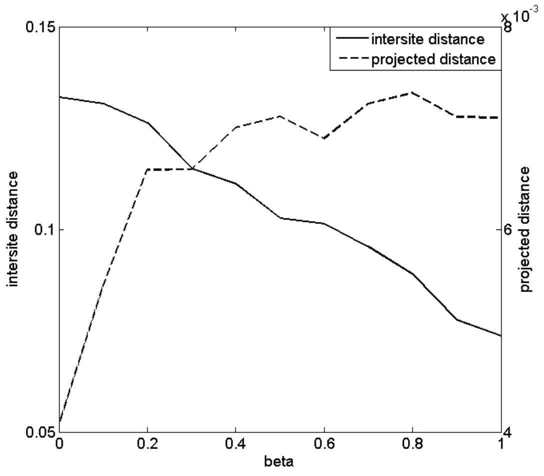


Figure 2.25: The effect of the β parameter from the `optimizer-proj` algorithm on the intersite and projected distance. Lower values of β favour intersite distance, while higher values of β favour projected distance. For $\beta = 0.3$, a good trade-off is found between intersite and projected distance.

calculating the centers of the hypercubes that satisfy this parameter, and optimize in these legal regions using a local optimizer. Instead of optimizing the reduced, restricted surface shown in Figure 2.17(b) with Monte Carlo, an optimizer is used to achieve the same goal. This method will be called `optimizer-intersite`.

This method is described in detail in Algorithm 4. The value of d_{min} is computed as in Equation 2.6. First, all intervals between consecutive values are computed for each dimension. This is done by sorting all the different values in P and substracting subsequent values from each other. The center of every interval is also computed. Then, all intervals are removed that violate the α threshold. Finally, all the hypercubes that do not violate the threshold in any dimension are produced. In the example shown in Figure 2.17(b), this will result in all the white squares. For each of these hypercubes, the intersite distance of its center from all points in P is calculated. The $n_{\text{hypercubes}}$ best scoring candidates are selected, and they are further optimized in this hypercube using a pattern search algorithm.

Again, $\alpha = 0.5$ was picked because the results from Section 2.5.4.2 showed that this value gives a good trade-off between intersite and projected distance. Pattern search was chosen because it is a fast, efficient optimization technique that can get out of local minima quite easily. For this study, the pattern search implementation from Matlab's Global Optimization Toolbox was used. Because pattern search is still quite slow, a selection must be made as to how many hypercubes will be optimized (the algorithm parameter $n_{\text{hypercubes}}$) and how long the optimizer is allowed to run for each hypercube. Preliminary experiments have shown that, no matter how long the optimizer is allowed to run, it will always converge to a final

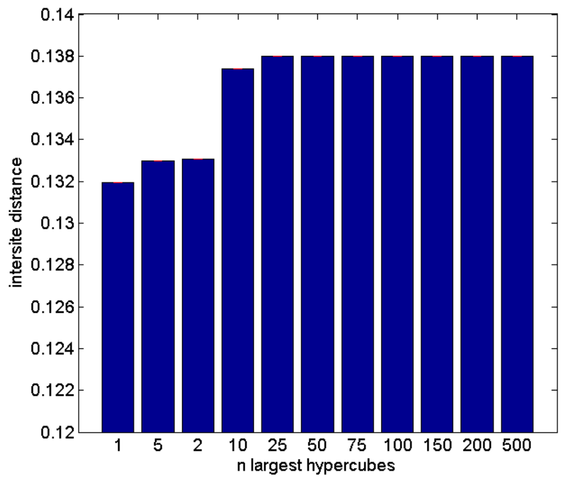
Algorithm 4 The `optimizer-intersite` algorithm.

```
for $i = 1$ to $d$ do
    $P^i \leftarrow$ values of $P$ in dimension $i$
    Sort $P^i$
    $I^i = P^i_{2 \text{ to } n} - P^i_{1 \text{ to } n-1}$
    $C^i = \frac{P^i_{1 \text{ to } n-1} + P^i_{2 \text{ to } n}}{2}$
    $I^i = I^i - 2d_{\text{min}}$
end for
for all $\mathbf{v}$ where $I^i_{\mathbf{v}_i} > 0$ do
    $\mathbf{p}_{\text{new}_i} = C^i_{\mathbf{v}_i}$
    $P_{\text{new}} = P_{\text{new}} \cup \mathbf{P}_{\text{new}}$
end for
Take $n_{\text{hypercubes}}$ points in $P_{\text{new}}$ with largest idist
for all $\mathbf{p}_{\text{new}} \in P_{\text{new}}$ do
    Optimize $\mathbf{p}_{\text{new}}$ towards idist in $[\mathbf{p}_{\text{new}} - \frac{I_{\mathbf{p}_{\text{new}}}}{2}, \mathbf{p}_{\text{new}} + \frac{I_{\mathbf{p}_{\text{new}}}}{2}]$
end for
Choose best $\mathbf{p}_{\text{new}}$ after optimization
```

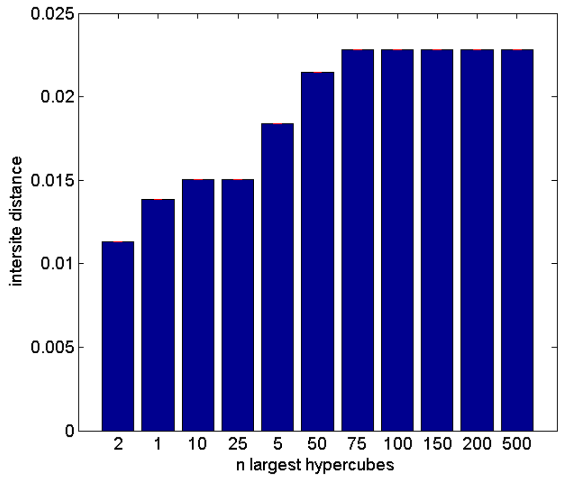
optimum after about 100 evaluations, so no absolute maximum was set for the optimizer.

The other algorithm parameter, $n_{\text{hypercubes}}$, does have a huge influence on the quality of the final design. It is clear that, if $n_{\text{hypercubes}}$ is low, only a few hypercubes will be considered (purely based on the projected distance criterion) and these might not be the best choice for the intersite distance criterion. However, it was already shown in Section 2.5.4.3 that a sample lying in a hypercube that satisfied the α threshold already has by definition acceptable space-filling properties. Therefore, it is expected that a relatively small number of $n_{\text{hypercubes}}$ will already be sufficient for a good design.

A comparison of different $n_{\text{hypercubes}}$ values for the intersite distance in 2D and 3D is shown in Figure 2.26. In 2D, no noticeable improvement can be noticed when $n_{\text{hypercubes}}$ is increased beyond 50, while in 3D, 75 hypercubes are enough. This indicates that the center of the intervals is already a very good guess in terms of the intersite distance, which intuitively makes sense. However, considering that the total number of valid hypercubes after 144 samples can be larger than 10 000, this is still an interesting result. The projected distance is not shown as, similar to `mc-intersite-proj-th`, only the α parameter has an influence on this value.



(a) 2D



(b) 3D

Figure 2.26: The intersite distance after generating 144 points using different values for the algorithm parameter $n_{\text{hypercubes}}$ in 2D and 3D.

2.6 Experiments

In this section, we present several experiments that were conducted to study the validity of the existing and newly suggested methods. Each experiment took a different approach to comparing the methods. In total, the results will give a clear view of the quality and performance of the different methods presented in this thesis.

The first approach is to generate a fixed number experimental design using the different methods, and measuring the criteria described in Section 2.2 on the final design. This results in a clear view of the efficiency of the different methods, as there are no outside influences to affect the outcome.

The second approach is to do a full surrogate modeling run, using different sequential design methods to generate the samples. Once a desired accuracy on the model was reached, the process is halted, and the total number of required samples is compared. This will clearly illustrate the importance of good sampling for the quality and accuracy of the final model, but the results will also be blurred by the compatibility between the model type and the sampling method.

It is expected that the results of both approaches will be similar; if a design has very good intersite and projected distance, it is expected to be a good design for modeling as well, and the surrogate modeling process should greatly benefit from it.

In order to compare the different sequential design methods, all strategies were implemented in the SUrrogate MOdelling (SUMO) research platform [40]. This Matlab toolbox, designed for adaptive surrogate modelling and sampling, has excellent extensibility, making it possible for the user to add, customize and replace any component of the modelling 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 these experiments.

In the following sections, the methodology and results for the two approaches will be presented.

2.6.1 Criterion-based comparison

In this experiment, the goal is to measure for each sequential method the quality of the generated design purely based on the criteria defined in Section 2.2. This results in a neutral comparison of each method, in which the results are not affected by other parts of the adaptive surrogate modelling process, such as the model type.

Each of the strategies mentioned in the previous sections will be used to generate 144 points in 2D, 3D and 4D. Each method will be allowed to run at most 15 minutes to generate a design of 144 points on an Intel quadcore running at 1.86GHz[1]. This is acceptable, considering the fact that simulations are assumed to be expensive, and can take hours or days for one evaluation. In this context, 15

[1]No parallelization was explicitly programmed into the algorithms, but Matlab may use different cores to execute built-in commands faster.

minutes to generate a good space-filling design is a good time investment. For each method in each dimension, the experiment will be run 30 times in order to get an estimate of the standard deviation on each method.

All the methods were compared on three criteria discussed in Section 2.2: granularity, intersite and projected distance. The granularity of the methods is summarized in Table 2.2. Each new method proposed in this chapter, except for `lhd-nested`, has the best possible granularity: the total number of samples does not have to be known in advance, they produce good designs whenever they are aborted, and they select samples one by one.

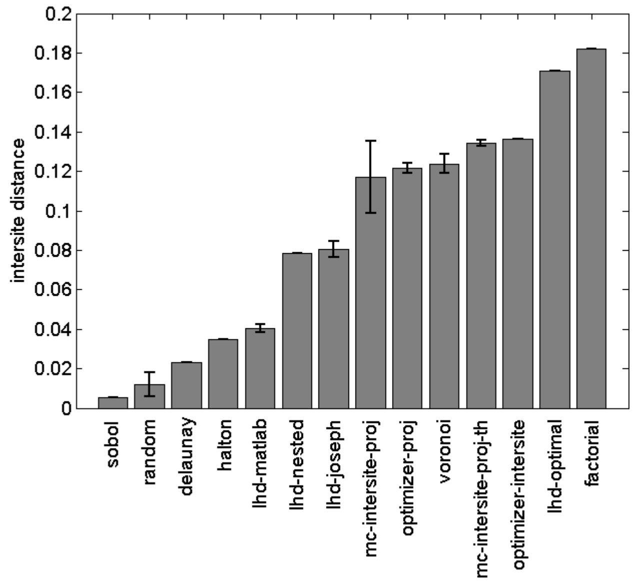
Table 2.2: The different space-filling design methods in terms of their granularity. It shows, for each method, whether the method must know the total number of samples in advance, whether the method is available for all number of samples and how many samples it selects at each iteration.

Method	# samples known	n restricted	Step size
`factorial`	yes	no	∞
`lhd-optimal`	yes	yes	∞
`lhd-nested`	no	yes	2^k
`voronoi`	no	yes	1
`delaunay`	no	yes	1
`random`	no	yes	1
`halton`, `sobol`	no	yes	1
`mc-intersite-proj`	no	yes	1
`mc-intersite-proj-th`	no	yes	1
`optimizer-intersite`	no	yes	1
`optimizer-proj`	no	yes	1

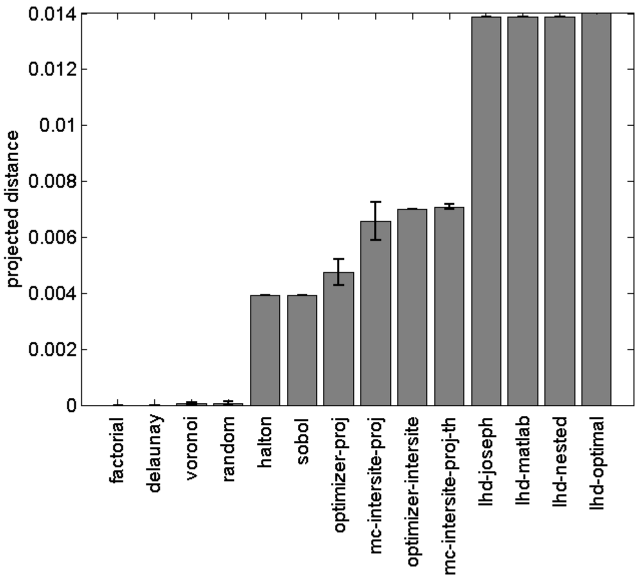
Figure 2.27(a) contains the results for the intersite distance in 2D, after 144 points were generated. `factorial` is, of course, the best space-filling design. However, it is closely followed by `lhd-optimal`, which demonstrates that, if the total number of points is known in advance, it is possible to generate a design practically as space-filling as a factorial, but with optimal projected distance as well.

The next best methods are five new methods proposed in this chapter. The best method turns out to be `optimizer-intersite`, which only performs 20% worse than the pre-optimized Latin hypercube, yet produced the design in a much smaller timespan, and with no knowledge at all of the total number of samples that were going to be needed. After these five methods, the quality of the design goes down significantly.

Note the big difference between the two sequential Monte Carlo strategies `mc-intersite-proj` and `mc-intersite-proj-th`. By replacing the projected distance by a threshold function, the quality of the design in terms of intersite distance improves considerably. The variance is also reduced, making the method much more stable. Also interesting to note is the rather poor performance of the Matlab Latin hypercube implementation, which was allowed to optimize for 15



(a) Intersite distance in 2D



(b) Projected distance in 2D

Figure 2.27: The average intersite and projected distance score for each design method discussed in this chapter, after generating a 144-point design in 2D.

minutes to allow for a fair comparison. This method fails to generate a good space-filling design, and should be avoided. The same can be said for the low-discrepancy sequences, which, even though they generate good space-filling designs for large numbers of points, perform bad for small sample sizes. Also noticeable is the bad performance of `lhd-nested`. This can be explained by the fact that, by selecting the optimal point from the Latin hypercube grid at one iteration, future iterations may get stuck in a local optimum, as described in Section 2.5. In this case, the last point selected before the grid is refined will be a very bad choice, resulting in a dramatic drop in quality of the design.

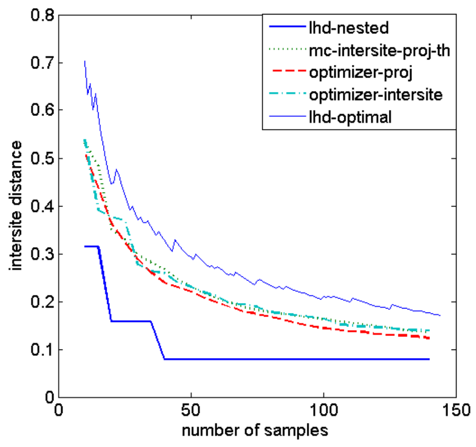
Figure 2.27(b) shows the projected distance for the same designs. Obviously, the `factorial` design has the worst projected distance, while the Latin hypercubes have the best score, followed by the five methods proposed in this chapter, which have a projected distance about 50% worse than the Latin hypercube. This is still very good, considering that the Latin hypercube has the best possible projected distance by construction. The projected distance of many of these methods can be further improved by tweaking the algorithm parameters (such as the α threshold parameter), at the expense of intersite distance. Since the intersite distance is deemed the more important criterion of the two, more priority was given to achieving a high intersite distance in these experiments.

Figure 2.28 shows the evolution over time of the intersite and projected distance for the algorithms proposed in this chapter, compared to the distance scores for each `lhd-optimal` for that number of points. Note that the curve drops smoothly for all of the algorithms, except the nested Latin hypercube method. This demonstrates again the tendency of this method to get stuck in local optima, where at one point, the algorithm is forced to pick a very bad sample. The other methods suffer much less from this problem, because the points are not selected on a fixed candidate grid.

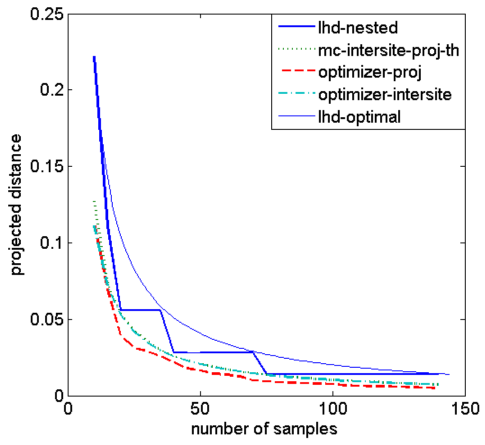
In 3D and 4D, some of the methods that were available in 2D will not work anymore. More particularly, there is no 144-point factorial design available in 3D and 4D. Also, the grid in `lhd-nested` becomes too large to evaluate completely within 15 minutes, so this method was also left out. Finally, computing a Delaunay triangulation becomes considerably more expensive in higher dimensions (see [17] for an analysis), so due to the strict time limitation, this method was left out as well.

Figure 2.29 shows the intersite and projected distance scores for 3D, while Figure 2.30 shows the intersite and projected distance for 4D. Note that the `optimizer-intersite` method performs 21% worse than `lhd-optimal` in 2D, but only 16% worse in 3D and 8% worse in 4D. This is an extremely good result, considering that this method only ran for 15 minutes, while the 4D 144-point Latin hypercube was optimized for 6 hours. The projected distance is in all dimensions about 50% worse than `lhd-optimal`.

Even though a limit of 15 minutes was imposed on all the methods, not all methods are equally demanding in terms of computing power. As shown in the different performance analysis sections of this chapter, increasing some method parameters does not always give a noticeable improvement in the final design. Several methods did not threaten to violate the 15-minute limit when they were con-

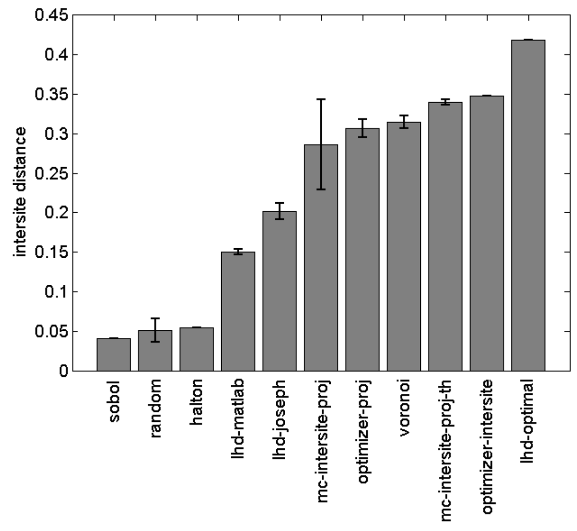


(a) Intersite distance in 2D

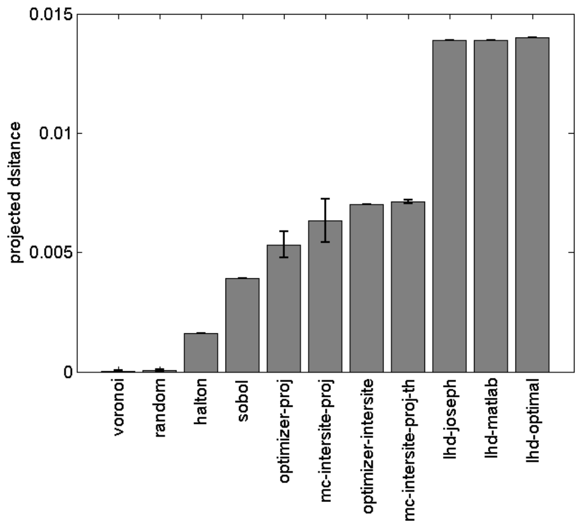


(b) Projected distance in 2D

Figure 2.28: Respectively the intersite and projected distance as a function of the number of points selected so far. This graph shows the evolution over time as the algorithm selects more points, up to a maximum of 144 in 2D. For comparison, the intersite and projected distance of each pre-optimized Latin hypercube is also shown, even though it is not a sequential algorithm.

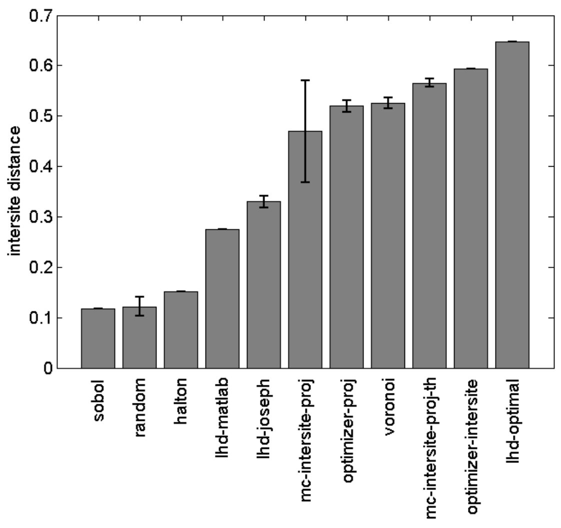


(a) Intersite distance in 3D

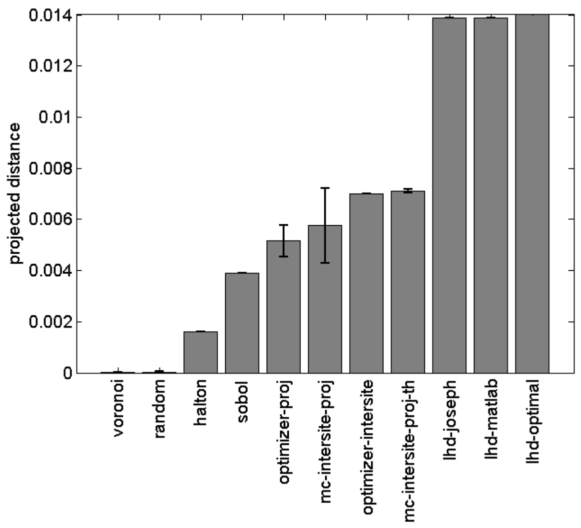


(b) Projected distance in 3D

Figure 2.29: The average intersite and projected distance score for each design method discussed in this chapter, after generating a 144-point design in 3D.



(a) Intersite distance in 4D



(b) Projected distance in 4D

Figure 2.30: The average intersite and projected distance score for each design method discussed in this chapter, after generating a 144-point design in 4D.

figured with the parameters that came out of the different performance analyses in this chapter. In fact, every sequential method proposed in this study requires less than 5 minutes, except for `optimizer-intersite` and `optimizer-proj`, which require on average 10 minutes. Especially the Monte Carlo methods prove to be very time-efficient: both Monte Carlo designs were generated in under 5 minutes, since increasing the number of random points does not improve the quality of the design much, as demonstrated in Section 2.5.4.3. These methods also don't increase much in terms of computing time when the dimension is increased. This is opposed to the optimization-based methods, which require considerably more time in higher dimensions. This may cause the `optimizer-intersite` method to become impractical in higher dimensions. The Monte Carlo method, on the other hand, should remain fast and viable in dimensions higher than 4.

2.6.2 Model-based comparison

In order to compare the different methods using a full surrogate modeling run, the SUMO Toolbox was used. The work flow of SUMO for a typical surrogate modelling task with sequential design is illustrated in Figure 2.31. 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 hyperparameters which can be modified, such as the order of numerator and denominator for rational models, number and size of hidden layers in neural networks, smoothness parameters for RBF models, and so on. These parameters are adjusted using an 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 selects a set of new sample locations to be evaluated, and the algorithm starts all over again. For more information on the SUMO Toolbox, please refer to Section 4.1.

All problems in this chapter 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 Sacks et al. [82]. In Kriging, the surrogate model is of the form:

$$\tilde{f}(x) = \sum_{i=1}^{n} \beta_i h_i(\mathbf{x}) + Z(\mathbf{x}) \tag{2.8}$$

The first part of the equation is a linear regression model, while the second part $Z(\mathbf{x})$ is a Gaussian random process with zero mean and non-zero covariance. Kriging works very well with space-filling methods, because it has stability problems with points that are placed too close to each other. This is due to the structure of the correlation matrix for Kriging, which tends to become ill-conditioned when two points close to each other. For more information on Kriging and the implementation used in the SUMO Toolbox, please refer to [13, 68].

The accuracy of the model is measured by comparing it against a very dense, pre-evaluated test set. Thus, the error is a very accurate estimate of the true

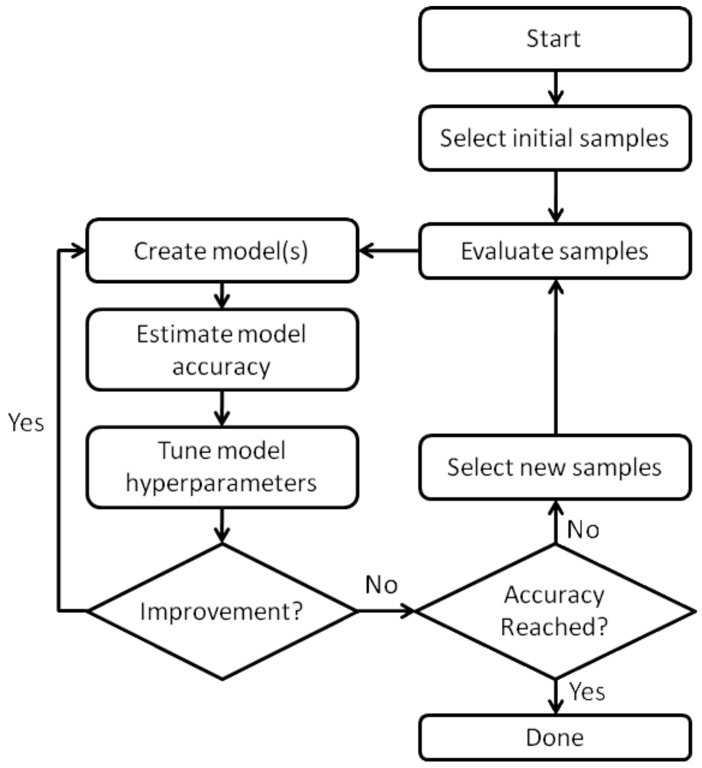


Figure 2.31: Flow chart of the SUMO Toolbox.

prediction error of the model. Each run will be terminated when the average euclidean error between the model outputs and the test set reaches the desired threshold. The average Euclidean error (AEE) is defined as:

$$\text{AEE} = \frac{1}{n}\sum_{i}^{n}\sqrt{(f(x_i) - \tilde{f}(x_i))^2} \tag{2.9}$$

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 addition to sequential design methods, which fit neatly into the flow of a typical SUMO surrogate modeling run, Latin hypercubes will also be included in this study. In order to compare the sequential design strategies against the Latin hypercube, which is inherently non-sequential, we generated Latin hypercubes of size 50, 60, 70 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.

The following methods were included in this study: `lhd-optimal`, `lhd-joseph`, `voronoi`, `delaunay`, `mc-intersite-proj-th`, `optimizer-intersite` and finally `random`. Please note that all the algorithms used in this experiment are available in the open software distribution of the SUMO Toolbox (which can be found at `http://sumo.intec.ugent.be`) and therefore, the results of this experiment can easily be reproduced.

To properly investigate the overall quality of each design method, three different use cases were investigated.

2.6.2.1 Ackley's Path

The first test case is Ackley's Path [2], which is defined as:

$$y = -a\exp\left(-b\sqrt{\frac{(2x_1)^2+(2x_2)^2}{2}}\right) - \exp\left(\frac{cos(2cx_1)+cos(2cx_2)}{2}\right) + a\exp(1) \tag{2.10}$$

where $a = 20$, $b = 0.2$ and $c = 2\pi$. The behaviour of Ackley's Path on the domain $[-1,1]^2$ is shown in Figure 2.32. 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's Path 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.

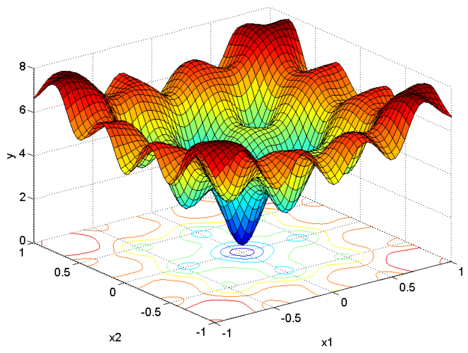


Figure 2.32: Ackley's Path function.

2.6.2.2 Results

The results of the experiment are depicted in Table 2.3. The best results are achieved using the pre-optimized Latin hypercube `lhd-optimal`, followed very closely by `mc-intersite-proj-th`, which only requires 4 more points on average than `lhd-optimal`, but has a larger variance. Considering that `lhd-optimal` took many hours to optimize and `mc-intersite-proj-th` was calculated in real-time in a couple of minutes, this result is quite impressive. The next best methods are `optimizer-intersite`, `voronoi` and `delaunay`.

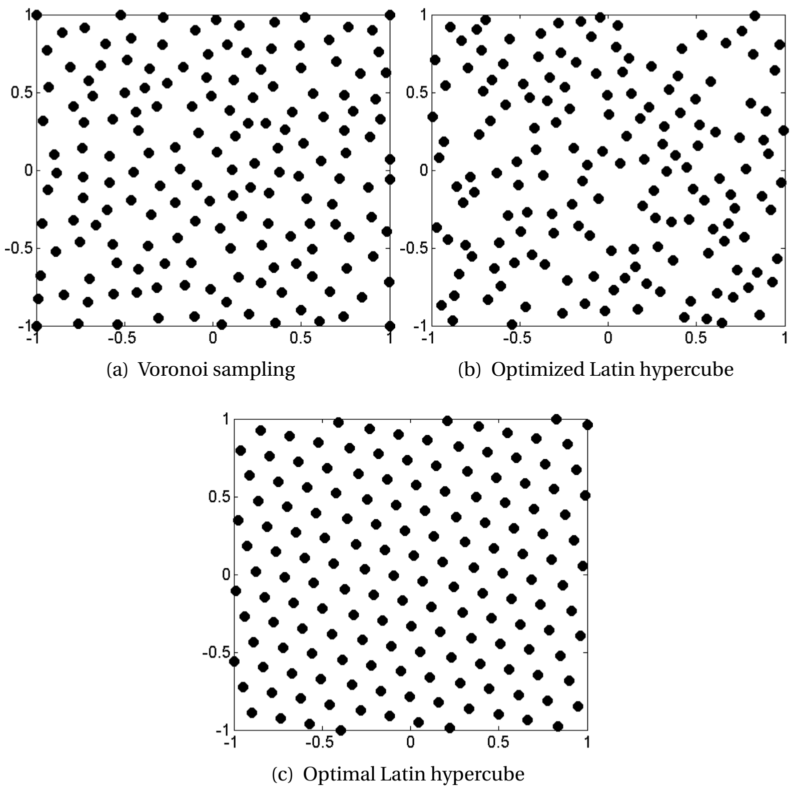
Table 2.3: Summary of the test results of modelling Ackley's Path with different sampling strategies. The average number of samples required to reach an average Euclidean error of 0.1 are shown for each sampling strategy. For the Latin hypercube, the smallest Latin hypercube size that on average over 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
`lhd-optimal`	130	0
`lhd-joseph`	180	0
`mc-intersite-proj-th`	134	11
`optimizer-intersite`	147	0
`voronoi`	158	7.4
`delaunay`	155	7.9
`random`	226	12

Optimized Latin hypercubes, which are widely used and popular, perform much worse than all other methods except for `random`, 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 2.33 compares the sample distribution of a run with `voronoi` against a 160-point optimized Latin hypercube and a 160-point optimal Latin hypercube. It is visually 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. The optimal Latin hypercube, on the other hand, is still much more uniformly space-filling than the Voronoi-based design.

2.6.2.3 Electrical low-noise amplifier (LNA)

The second test case is a real world problem from electronics: a narrowband Low Noise Amplifier (LNA), which is a simple RF circuit [62]. 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.



(a) Voronoi sampling (b) Optimized Latin hypercube

(c) Optimal Latin hypercube

Figure 2.33: A comparison of the sample distribution of Voronoi-based sampling against optimized and optimal Latin hypercube sampling for 160 points.

The performance figures of an LNA (gain, input impedance, noise figure and power consumption) can be determined by means of computer simulations where the underlying 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. Initial manual tests indicate that results obtained from this simplified model are also applicable to the physics-based simulator.

The two input parameters used in this experiment are the inductances L_{sn} and the MOSFET width W_n. The output modelled in this experiment is the input-noise current $\sqrt{\overline{i_{in}^2}}$, which is defined by Equations 2.11 to 2.18. This output was chosen because it is the most challenging performance figure of the LNA [38] to model. The behaviour of the input-noise current is shown in Figure 2.34. Note that the response is very linear in most of the design space, except for one tall ridge near $W = 0$.

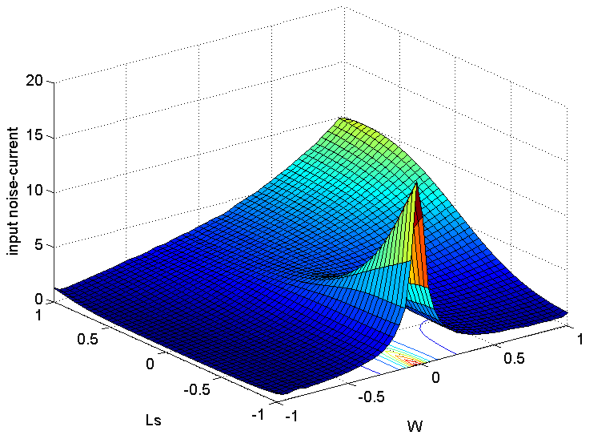


Figure 2.34: The input noise-current of a low-noise amplifier as a function of the inductance L_s and the MOSFET width W.

$$\omega = 2\pi f \tag{2.11}$$

$$g_m = 1 \cdot 10^{-4} \frac{W}{L} V_{GT} \text{ AV}^{-1} \tag{2.12}$$

$$C_{GS} = 0.01 \cdot W L \text{ F} \tag{2.13}$$

$$f_{gs,in} = \frac{1 + j\omega L_s g_m}{1 - \omega^2 C_{gs}(L_s + L_m) + j\omega L_s g_m} \tag{2.14}$$

$$f_{ds,in} = \frac{\omega^2 C_{gs}(L_s + L_m)}{1 - \omega^2 C_{gs}(L_s + L_m) + j\omega L_s g_m} \tag{2.15}$$

$$\overline{i_{gs}^2} = 2 \cdot 10^{-3} \frac{W}{L} \text{ pA}^2\text{Hz}^{-1} \tag{2.16}$$

$$\overline{i_{ds}^2} = 0.5 \frac{W}{L} \text{ pA}^2\text{Hz}^{-1} \tag{2.17}$$

$$\sqrt{\overline{i_{in}^2}} = \sqrt{|f_{gs,in}|^2 \cdot \overline{i_{gs}^2} + |f_{ds,in}|^2 \cdot \overline{i_{ds}^2} - 2 \cdot \text{Im}(0.4 f_{gs,in} f_{ds,in}^*) \sqrt{\overline{i_{gs}^2} \cdot \overline{i_{ds}^2}}} \tag{2.18}$$

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:

$$L_m = 10^{-9} \text{ H},$$

$$f = 13 \cdot 10^9 \text{ Hz},$$

$$L = 82.5 \cdot 10^{-9} \text{ m},$$

$$V_{GT} = 0.325 \text{ V}.$$

2.6.2.4 Results

The results are summarized in Table 2.4. Several interesting observations can be made.

Table 2.4: 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
`lhd-optimal`	180	0
`lhd-joseph`	225	0
`mc-intersite-proj-th`	197	6
`optimizer-intersite`	204	0
`voronoi`	176	30
`delaunay`	199	32
`random`	256	75

Firstly, `voronoi` is now the best method, surpassing `lhd-optimal`, even though the optimal Latin hypercube is much more uniformly space-filling than `voronoi`. This can be explained by the fact that the optimal Latin hypercube fails to capture the ridge, resulting in subpar accuracy. This is illustrated in Figure 2.35, which shows the optimal Latin hypercube of 170 and 180 points. Due to unfortunate placement of the samples, the 170-point design almost completely misses the ridge, resulting in an average Euclidean error of 0.1. The 180-point design, while similar in shape, has several samples right on the ridge, resulting in a dramatic increase of accuracy to 0.03. This highlights the importance of sequential sampling: perhaps only a couple of extra samples could have improved the 170-point design to an acceptable level. But because of the one-shot nature of Latin hypercubes, the only way to do this is to resort to sequential sampling.

Secondly, even though `voronoi` has the best performance, it also has a huge variance compared to the much more stable methods `mc-intersite-proj-th` and `optimizer-intersite`. These methods take more measures to limit the influence of the Monte Carlo aspect of the sampling strategy, resulting in less variance. `mc-intersite-proj-th` limits the region in which Monte Carlo samples are taken to those already satisfying the projected distance constraint, while `optimizer-intersite` performs local optimization to reduce the variance. These methods will always produce very similar designs. For this test problem, this might be a disadvantage, because if this one design misses the ridge for a while, the results will on average always be worse than a method such as `voronoi` which will sometimes produce much less suitable and sometimes much more suitable designs.

Thirdly, except for `random`, `lhd-joseph` is once again the worst choice. These results are in line with the results obtained from the previous experiment. Again, it appears that Latin hypercubes, even those optimized for intersite distance, leave relatively large gaps in the design space. If such a gap happens to lie on or near the ridge, the accuracy suffers significantly. 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.

2.6.2.5 Truss structure

The third and final problem is the design of a two-dimensional truss, constructed by 42 Euler-Bernoulli beams. The goal is to study the effect of node displacement on the passive vibration isolation. The truss structure is shown in Figure 2.36 and is a simplification of a truss type typically used in satellites.

The beams consist each of two finite elements and are subject to a unit force excitation at node 1 across a 100 – 200 Hz frequency range. The two leftmost nodes are fixed (cantilevered nodes) and all the other nodes are free to move around. There are 2 input parameters defining the position of node 9 in the structure and 1 output parameter, which describes the stress that the outermost node (the tip) receives. The geometry of the node is changed by moving node 9 around inside the 0.9×0.9 square while the remaining nodes are fixed at their positions. The objective is to model the band-averaged vibration attenuation at the tip compared to the baseline structure. For an in-depth discussion of the problem, please refer to [57].

The vibration attenuation as a function of the geometrical position of the 9th node is shown in Figure 2.37. Note that the surface is quite erratic, with several local optima. Without a good space-filling design, several important features of the response may be missed.

2.6.2.6 Results

The results are summarized in Table 2.5. In this case, `optimizer-intersite` performs the best, outperforming `lhd-optimal` by a small margin. The other methods follow closely with similar results. Note the large difference between the sequential methods and random sampling; this demonstrates once again that intelligently chosen sample locations can have a huge effect on the accuracy of the model, especially with small sample sizes.

This third experiment once again demonstrates that sequential design strategies can perform equally well as or even outperform pre-optimized Latin hypercubes in realistic modelling situations. While pre-optimized Latin hypercubes might have better space-filling properties than the sequential design methods, this does not necessarily translate into better models. Because of the everything-or-nothing nature of one-shot designs, there is no room for correction when a design completely misses an essential feature of the response. Sequential methods, on the other hand, will keep selecting samples one by one until the design covers this feature.

Table 2.5: Summary of the test results of modelling a two-dimensional truss structure as a function of the position of the 9th node. 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
`lhd-optimal`	70	0
`lhd-joseph`	160	0
`mc-intersite-proj-th`	72	5
`optimizer-intersite`	66	0
`voronoi`	74	5
`delaunay`	99	7
`random`	171	52

2.7 Conclusions

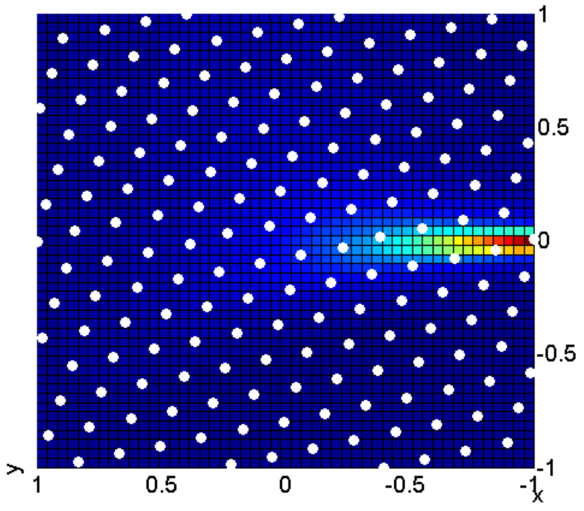
In this chapter, several new methods for sequentially generating space-filling designs for simulation-based experiments were proposed. These methods were thoroughly compared against proven and popular techniques (such as Latin hypercubes and low-discrepancy sequences) on three criteria: granularity, intersite (or maximin) distance and projected (or noncollapsing) distance. It was demonstrated that the new methods manage to generate good designs, close to the quality of a pre-optimized Latin hypercube. They also manage to generate these designs orders of magnitude faster than it takes optimizing a Latin hypercube of the same size. It was shown that in higher dimensions, the methods come even closer to the pre-optimized Latin hypercube: in 4D, the best new method produced a space-filling design only 8% worse than the pre-optimized Latin hypercube.

Of the new methods proposed in this chapter, `optimizer-intersite` and `mc-intersite-proj-th` produce the best space-filling designs overall. Of these, the second method is considerably faster than the first one: where the first one requires approximately 3 minutes to generate a design, `optimizer-intersite` takes about 10 minutes in 2D and up to 15 in higher dimensions.

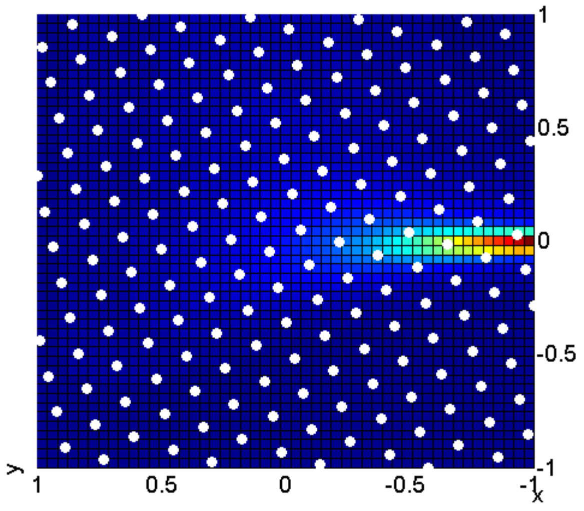
Furthermore, the most promising methods were also compared against each other in a real scenario, by using the SUMO Toolbox to perform a full-fledged adaptive surrogate modelling experiment. In order to determine which design produces the most accurate models, Kriging models were trained on three different test cases from different problem domains, and the number of samples needed to achieve a particular accuracy was measured. It was demonstrated that in this context, the most promising sequential methods such as `optimizer-intersite` and `mc-intersite-proj-th` can sometimes generate models with the same accuracy with less samples than the pre-optimized Latin hypercube `lhd-optimal`. Because these methods do not need to know the total number of samples in advance and therefore do not take the risk of over- or undersampling, it is strongly advised to use the most performant sequential methods instead of the pre-optimized Latin hypercube. The small advantage in intersite- and projected distance of `lhd-optimal` apparently does not translate into better models, and the advantages of sequential methods strongly outweigh the minor improvement in accuracy.

As a rule of thumb, we suggest to use a pre-optimized Latin hypercube only if the total number of samples that will or can be evaluated is known in advance. It is strongly discouraged to use the built-in Latin hypercube method from Matlab, as well as optimizing a Latin hypercube on the fly, as it may take many hours to generate a design that is as good or better than the algorithms proposed in this chapter. If the total number of samples is not known in advance, or no pre-optimized Latin hypercube is available for a particular number of samples with the right number of dimensions, the first choice should be the threshold Monte Carlo method `mc-intersite-proj-th`, which is easy to implement, extremely fast and performs very well in all dimensions. If a little more time can be spent on generating the design, the `optimizer-intersite` is a very good choice as well, since it produces slightly better designs on average. In higher dimensions, for which optimizing a Latin hypercube can be unviable, these methods may be the only choice for producing a good space-filling design with good projective properties.

All of these methods are available in the SED (Sequential Experimental Design) Toolbox, an open source Matlab which is very easy to use and embed in your workflow. For more information on the SED Toolbox, please refer to Section 4.2.



(a) 170-point optimal Latin hypercube



(b) 180-point optimal Latin hypercube

Figure 2.35: Two optimal Latin hypercubes. Note that the 170-point design almost completely misses the tall ridge, while the 180-point design has more samples in this important area.

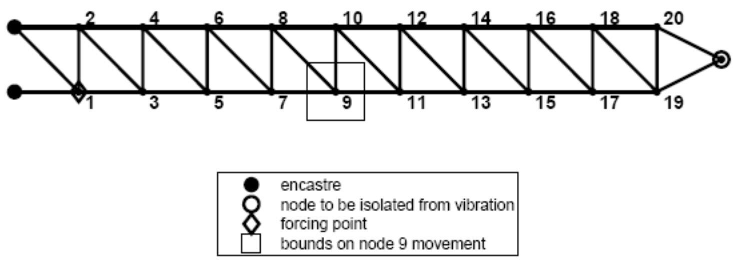


Figure 2.36: Two-dimensional truss structure. The location of node 9 is changed within the bounds indicated by a square. The goal is to study the effect of the position of node 9 on the vibration attenuation at the tip of the structure.

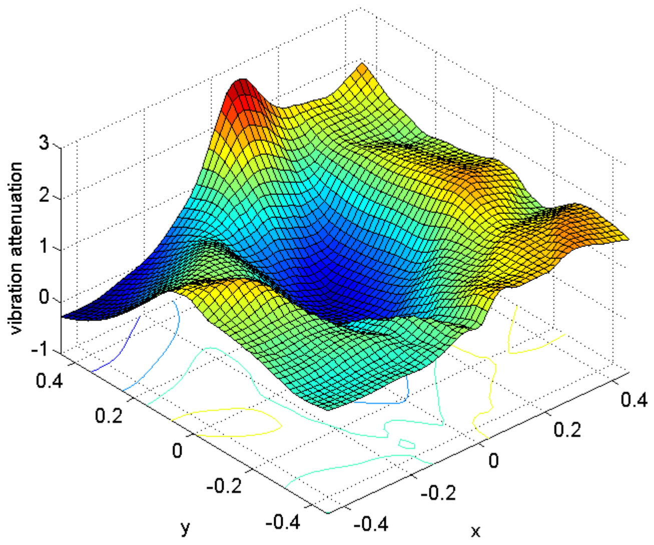


Figure 2.37: The vibration attenuation at the tip of the structure as a function of displacement of node 9 within a 0.9×0.9 square.

CHAPTER 3

Output-based sequential design

When life gives you lemons, don't make lemonade. Make life take the lemons back! Get mad! I don't want your damn lemons, what the hell am I supposed to do with these? Demand to see life's manager! Make life rue the day it thought it could give Cave Johnson lemons! Do you know who I am? I'm the man who's gonna burn your house down! With the lemons! I'm gonna get my engineers to invent a combustible lemon that burns your house down!
— Cave Johnson

3.1 Introduction

In the previous chapter, we proposed a number of input-based sequential methods or space-filling methods that only use the inputs from previous samples to determine where to sample next. These methods can be highly efficient at generating very good space-filling designs, even in high dimensions. However, they do not use the outputs from previous simulations to tailor the design to the problem at hand. In some cases, for example those in which some areas of the design space are much more difficult to approximate than others, it might be very benificial to the accuracy of the model to not distribute the samples evenly.

In output-based methods, the trade-off between exploration and exploitation becomes a central issue. As explained in Section 1.3.2, a balance must be found between the two in order to properly explore the design space on one hand, and focus on interesting areas on the other hand. In order to tackle the issue of exploration vs exploitation, we propose a novel, generic, disjunct approach, in which two different criteria are defined: one for exploration and one for exploitation. For the exploration criterion, the Voronoi-based space-filling method `voronoi` explained in Section 2.5.1 was used. For the exploitation criterion, we have developed an algorithm that selects additional samples near locations that deviate significantly from a local linear approximation of the system (based on the gradient of the function).

The Voronoi-based method is essentially a space-filling algorithm that was designed to work well in conjunction with the LOLA portion of LOLA-Voronoi. It can be used independently as a space-filling algorithm, but some of the more specialized methods proposed in Chapter 2 produce considerably better results. Moreover, Voronoi does not directly take the projective distance into account; this would only make it more difficult for LOLA to sample nonlinear regions. If used separately from LOLA, this is a major drawback.

LOLA-Voronoi works by ranking the neighbourhood of all existing data points. This ranking is based on the two aforementioned criteria. If a neighbourhood is ranked highly, it is either undersampled or very non-linear. In either case, an additional sample will be selected in this neighbourhood.

In the next sections, we will discuss the two components of the new hybrid sequential design algorithm in great detail. First, the Monte Carlo Voronoi method will be briefly discussed in the context of LOLA-Voronoi. Next, the Local Linear Approximation (LOLA) algorithm will be discussed and analyzed. Finally, the hybrid algorithm that combines these two components will be presented and tested on a number of test cases.

3.2 Exploration using a Voronoi approximation

From now on, what was known as `voronoi` in the previous Chapter will now simply be called Voronoi because there are no other space-filling methods to compete with. As explained in Section 2.5.1, Voronoi ranks each sample according to how large its (estimated) Voronoi cell is, based on a Monte Carlo approximation.

This results in a ranking for the least densely populated regions of the design space. This is shown in Algorithm 5.

Algorithm 5 Estimating the Voronoi cell size. P is the set of samples that have to be ranked according to their respective Voronoi cell size.

```
$S \leftarrow 100|P|$ random points in the domain
$V \leftarrow [0, 0, \ldots, 0]$
for all $\mathbf{s} \in S$ do
    $d \leftarrow \infty$
    for all $\mathbf{p} \in P$ do
        if $\|\mathbf{p} - \mathbf{s}\| < d$ then
            $\mathbf{p}_{\text{closest}} \leftarrow \mathbf{p}$
            $d \leftarrow \|\mathbf{p} - \mathbf{s}\|$
        end if
    end for
    $V[\mathbf{p}_{\text{closest}}] \leftarrow V[\mathbf{p}_{\text{closest}}] + (1/|S|)$
end for
```

The difference lies with how this information is consequently used. In the previous chapter, the ranking contained in V is used to select the largest Voronoi cells, and generate new samples in these cells. In this chapter, V will be used as the exploration criterion of LOLA-Voronoi, and will be combined with an exploitation criterion in order to arrive at a final ranking for each region, which will then be used to determine new sample locations. In the next section, the exploitation criterion will be discussed in great detail.

3.3 Exploitation using local linear approximations

The goal of the exploitation part of a hybrid sequential design algorithm is to use the responses from previous samples to guide the sampling process to interesting regions in the design space. Which regions are deemed interesting depends mainly on the purpose of the model. In optimization, interesting regions are regions which may or do contain (local) optima. In global surrogate modelling, the goal is to find a model that accurately approximates the system over the entire domain.

However, some regions of the domain may be more difficult to approximate than others. This may be due to discontinuities, many (local) optima close together, and so on. It is therefore intuitively a good idea to sample more densely at these difficult regions. More generally, samples should be distributed according to the local nonlinearity of the function. This is illustrated in Figure 3.1.

In order to be able to sample according to the local nonlinearity of the function, one needs a measure of this nonlinearity. To this end, we use the gradient of the system response. The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d} \right) \tag{3.1}$$

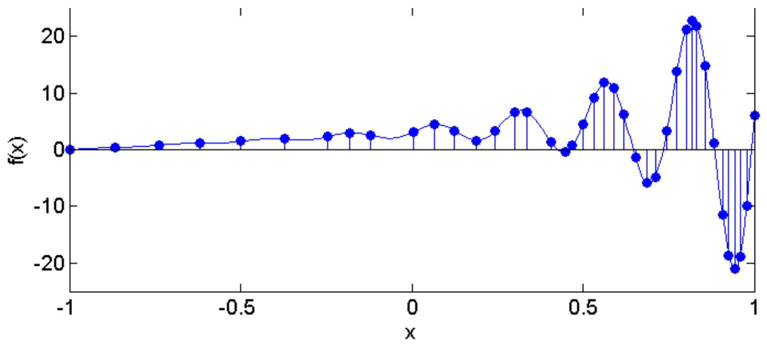


Figure 3.1: This plot shows a one-dimensional function and a set of samples for this function. The function is very simple and easy to approximate on the left hand side, but very nonlinear on the right hand side. Intuitively, more samples should be selected on the right to compensate for this nonlinearity. The samples in this plot were selected using the hybrid algorithm proposed in this chapter. As expected, more samples are selected at the right hand side to better capture the highly nonlinear behaviour.

The gradient of a function at a given point $\mathbf{p}_0 \in \mathbb{R}^d$ in the design space has the property that it represents the best local linear approximation for f around $\mathbf{p}_0$:

$$f(\mathbf{p}) = f(\mathbf{p}_0) + \nabla f_{\mathbf{p}_0}(\mathbf{p} - \mathbf{p}_0) \tag{3.2}$$

Therefore, the gradient can be used to estimate and quantify the nonlinearity in the region around $\mathbf{p}_0$ [46]. However, the gradient of the black-box function f is rarely known, so it cannot be used directly. The idea behind LOLA is to estimate the gradient at the data points, in order to measure the nonlinearity around these data points. Each *region* or *neighbourhood* is ranked according to its estimated nonlinearity, and new samples are selected in neighbourhoods which are highly ranked. From now on, the term neighbourhood will only be used to identify a set of samples which are chosen to represent the region around a particular data point.

A high level pseudocode overview of the algorithm can be found in Algorithm 6. When new samples have been evaluated by the simulator (for example, from a previous iteration of the sequential design method or from an initial experimental design), these samples have to be pre-processed by the LOLA algorithm. The algorithm considers a new sample $\mathbf{p}_{\text{new}}$ as a candidate neighbour sample for each previously processed sample $\mathbf{p}$. The neighbourhood $N(\mathbf{p})$ of a sample will be used to estimate the gradient at $\mathbf{p}$ later on. At the same time, $\mathbf{p}$ is also considered for the neighbourhood $N(\mathbf{p}_{\text{new}})$ of $\mathbf{p}_{\text{new}}$. After this initial preprocessing step, the gradient at each data point is estimated using the newly updated neighbourhoods. Finally, the local nonlinearity of the neighbourhoods is estimated by comparing the samples in the neighbourhood to the gradient estimation. This results in a ranking of each neighbourhood from linear to highly nonlinear. Finally, this ranking is used to select new samples in the highest ranking regions.

Algorithm 6 A high level overview of the LOLA algorithm. P are all the samples that have been processed by LOLA before. P_{new} are the data points that have been selected by the sequential design algorithm in the previous iteration, but have not been processed yet by the sampling algorithm. n_{new} is the number of new samples requested from the algorithm.

```
for all $\mathbf{p}_{\text{new}} \in P_{\text{new}}$ do
    for all $\mathbf{p} \in P$ do
        Try to add $\mathbf{p}_{\text{new}}$ to neighbourhood $N(\mathbf{p})$ of $\mathbf{p}$
        Try to add $\mathbf{p}$ to neighbourhood $N(\mathbf{p}_{\text{new}})$ of $\mathbf{p}_{\text{new}}$
        Update gradient estimations for $\mathbf{p}$ and $\mathbf{p}_{\text{new}}$
    end for
    $P \leftarrow P \cup \mathbf{p}_{\text{new}}$
end for
for all $\mathbf{p} \in P$ do
    Calculate error on gradient estimation in $N(\mathbf{p})$
end for
Pick $n_{\text{new}}$ highest ranked neighbourhoods
Select new samples in these neighbourhoods
```

Each component of the LOLA algorithm will be discussed in great detail in the following sections of this chapter. First, the mathematical background behind the neighbourhood selection algorithm will be explained. Next, it is shown how the neighbourhoods can be used to estimate the gradient at the data point, and how the gradient estimation is subsequently used to estimate the local nonlinearity of the function.

3.3.1 Estimating the gradient

A lot of research has been done on gradient-estimating methods [31]. These methods try to estimate the gradient at one point in the design space by evaluating samples near this point. This is often done in the context of optimization, following the assumption that the gradient is a good indication of the location of the optimum. Well-known optimization techniques such as hill climbing use this knowledge to guide the optimizer to the optimum.

These gradient estimation methods are further divided into indirect and direct estimation methods [95]. The main difference is that indirect methods assume a black box simulator, while direct methods use internal knowledge of the simulator or its behaviour. Examples of indirect gradient estimation techniques are finite differences, simultaneous perturbation, response surface methods and frequency domain methods [32]. An overview of indirect methods can be found in [31]. Examples of direct techniques are infinitesimal perturbation analysis and likelihood ratios.

All of these methods are, however, useless for the purpose of this algorithm, as they assume that additional new samples can be evaluated in order to achieve a proper estimate for the gradient. In the context of surrogate modelling of an expensive black box simulator, it is usually not acceptable to evaluate additional data points just to obtain the gradient. The objective is fundamentally different: in sequential design, estimating the gradient is only a small subgoal of a much larger problem, and samples are chosen as to maximize the accuracy of the surrogate model, not the accuracy of the gradient. This renders most traditional gradient estimation methods useless for the LOLA algorithm.

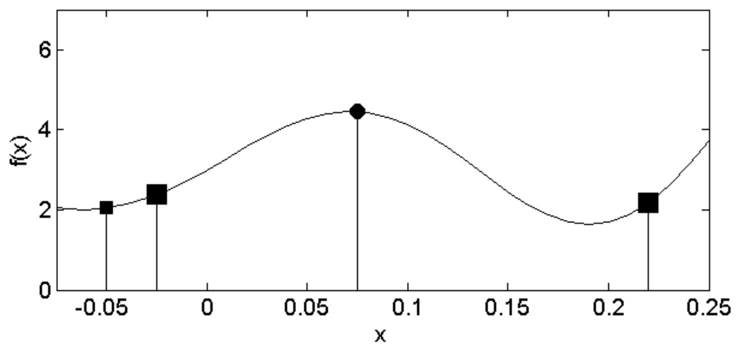
The LOLA algorithm requires a method that estimates the gradient as accurately as possible using only the data that is available. No assumptions can be made about the distribution of the data over the domain, because the algorithm has no complete control over the choice of all the data points: LOLA can be used in conjunction with other (space filling) sampling strategies, and the initial experimental design can take any form. Thus, gradient estimation methods that assume that data is available on a grid or any other pattern are unusable.

In the technique we propose, estimating the gradient in a sample location $\mathbf{p}_i$ comes down to choosing a set of neighbouring samples $N(\mathbf{p}_i) = \{\mathbf{p}_{i1}, \mathbf{p}_{i2}, \ldots, \mathbf{p}_{im}\}$ that lie close to the sample $\mathbf{p}_i$ and provide as much information as possible about the region around $\mathbf{p}_i$. This is illustrated in Figure 3.2 for the one-dimensional case. The sample, for which we want to find a neighbourhood, is drawn as a circle. Three candidate neighbours are drawn as squares. The problem of choosing two neighbours out of these three candidates will now be considered. In Figure 3.2(a), two neighbours are chosen on opposite sides (drawn as larger squares), while in Figure 3.2(b), two neighbours are chosen on the left side.

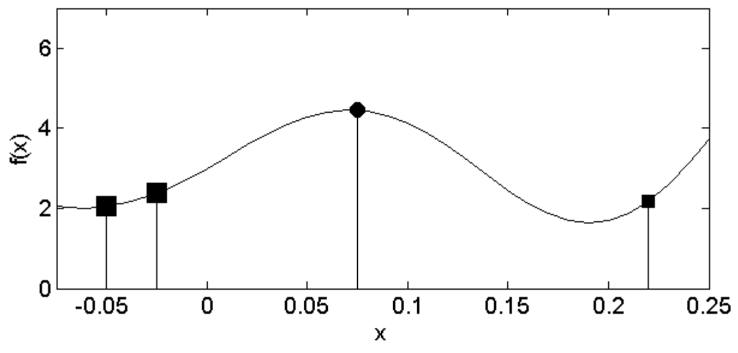
If distance from the sample is chosen as the metric to determine the best candidates for the neighbourhood, the neighbourhood displayed in Figure 3.2(b) will be chosen over the one displayed in Figure 3.2(a), since both candidates on the left lie closer to the middle than the one on the right. However, it is obvious that the other neighbourhood conveys much more information about the behaviour of the function in this region. The information gain from adding a second neighbour close to another one is much smaller than the information gain from adding one on the other side. Thus the need arises for a metric that scores neighbourhoods according to how informative they are.

3.3.2 Constructing the neighbourhoods

When new samples are available (either from a previous iteration of the algorithm, or from the initial experimental design), they have to be processed, and neighbourhoods have to be assigned to each sample. This will be done in a sequential manner. Each sample will be considered as a candidate neighbour for each previously processed sample, and at the same time each previously processed sample will be considered as a candidate neighbour for the new sample. Thus, for each new sample, one needs to revisit all previous samples and their neighbourhoods. We will now first consider the simplified case of finding the optimal neighbourhood for one particular sample, given a set of other samples.



(a) Good neighbourhood



(b) Bad neighbourhood

Figure 3.2: Two different neighbourhoods with 2 samples are shown in a 1D design space. The sample for which the neighbourhood has been chosen is drawn as a circle in the middle. Three candidate neighbours are drawn. The two candidates which have been selected as neighbours are drawn as large squares. The third candidate, which is not in the neighbourhood set, is drawn as a smaller square.

3.3.2.1 The ideal neighbourhood

For clarity, we will now refer to the sample for which we want to find a neighbourhood as the *reference sample* $\mathbf{p}_r \in P$, and to all the other available samples $P_r = P \setminus \mathbf{p}_r$ as *candidate neighbours.* Without loss of generality, we assume that $\mathbf{p}_r$ lies in the origin. This will allow us to omit the translation in the following formulas. The goal is to find a subset $N(\mathbf{p}_r) = \{\mathbf{p}_{r1}, \mathbf{p}_{r2}, \ldots, \mathbf{p}_{rm}\} \subset P_r$ that best represents the region around $\mathbf{p}_r$.

The ideal neighbourhood is a good representation of the region around $\mathbf{p}_r$, covering each direction equally, thus providing the highest amount of information on the behaviour of the function around $\mathbf{p}_r$ as possible. The samples of such a neighbourhood must lie relatively close to the reference sample to be meaningful. They must also lie far away from each other, in order to cover each direction as equally as possible. This results in two fundamental properties for the ideal neighbourhood:

1. **Cohesion:** neighbours lie as close to the reference sample as possible.
2. **Adhesion:** neighbours lie as far away from each other as possible.

These two properties necessarily conflict with each other. The optimally cohesive neighbourhood consists of all samples that lie as close to the reference sample as possible, while the optimally adhesive neighbourhood consists of all samples spread out over the far reaches of the design space. Therefore, a compromise will have to be made, giving preference over adhesive neighbourhoods that still lie relatively close to the reference sample.

First, the concepts of cohesion and adhesion will be defined mathematically. There are multiple sensible formulas, but the following definitions are chosen for reasons later explained. Cohesion is defined as the average distance of all neighbours from the origin (i.e. from $\mathbf{p}_r$):

$$C(N(\mathbf{p}_r)) = \frac{1}{m}\sum_{i=1}^{m} \left\|\mathbf{p}_{ri}\right\|. \tag{3.3}$$

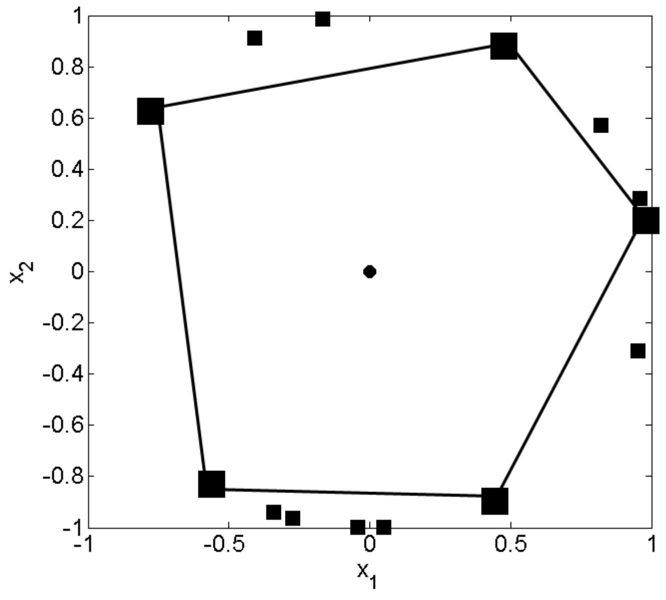
Furthermore, adhesion is defined as the average minimum distance of neighbours from each other:

$$A(N(\mathbf{p}_r)) = \frac{1}{m}\sum_{i=1}^{m} min\left\{\left\|\mathbf{p}_{ri} - \mathbf{p}_{rj}\right\| \mid j \neq i\right\}. \tag{3.4}$$

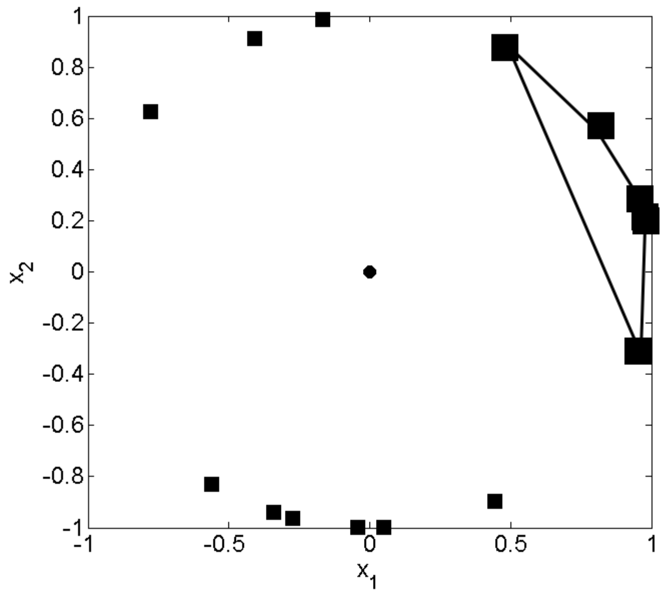
Initially, consider the simplified case where all candidate neighbours have the same contribution to the cohesion value, which means they all lie on the same distance from the origin. This is illustrated for the two-dimensional case in Figure 3.3, where all candidate neighbours lie randomly distributed on a circle. The point in the middle is the reference sample. The goal is to find a set of m samples in the circle that maximizes the adhesion value.

If all points on the circle are available as candidate neighbours, the best set of m neighbours are those that form a m-sided regular polygon. Of course, because of the non-uniform distribution of the samples, an ideal neighbourhood can rarely be formed; however, amongst neighbourhoods with equal cohesion values, some are clearly superior to others. There is a strict hierarchy amongst candidate neighbourhoods, defined by their adhesion, as long as cohesion is identical for all candidate neighbours. The neighbourhood with the highest adhesion value for the given candidate neighbours is illustrated in Figure 3.3(a) for $m = 5$. An example of a bad neighbourhood can be found in Figure 3.3(b). This neighbourhood provides no information at all on the behaviour of the function on the left side of the reference sample.

In the 2D case, the ideal configuration for m neighbours when all candidates have the same cohesion contribution is, as previously mentioned, the m-sided regular polygon. In higher dimensions, this extends to the problem of placing m points in an ideal configuration on a (hyper)sphere so that the adhesion value



(a) Good neighbourhood



(b) Bad neighbourhood

Figure 3.3: Examples of good and bad neighbourhoods chosen from the same set of candidate neighbours in a 2D design space. Samples that were selected for a neighbourhood are drawn as large squares, rejected neighbours are drawn as small squares.

as defined by Equation 3.4 is maximized, which is a well-known open problem in mathematics for which there is no known general solution in all dimensions. In fact, this is considered as one of the great mathematical challenges of the 21st century [15]. This is a major problem because the LOLA sampling algorithm should function independent of the dimensionality of the design space.

Because there is no known optimal solution for the problem of placing m points on a d-dimensional hypersphere [83], we have focused on a subproblem for which there *is* a solution known for all dimensions: the special case where $m = 2d$, or the size of the neighbourhood m is twice the dimensionality d of the design space. It can be intuitively seen that the optimal configuration in 1D has one neighbour on each side of the reference point, while the optimal configuration in 2D is a square. This generalizes to the d-cross-polytope [12] in the d-dimensional case.

The d-dimensional cross-polytope contains all the points obtained by permuting the coordinates $(\pm 1, 0, 0, \ldots, 0)$. It has been proven that the cross-polytope configuration maximizes Equation 3.4 in all dimensions [12]. This means that the ideal neighbourhood resembles the cross-polytope as closely as possible. When the set of candidate neighbours consists of points that lie equally far from the origin, the best choice of neighbourhood will always be a cross-polytope shape.

3.3.2.2 The cross-polytope ratio

In reality, the problem is more complex. Candidate points do not lie on a hypersphere; they differ in distance from the reference point. This results in a multi-objective optimization problem, in which the goal is to minimize the cohesion function defined in Equation 3.3 while at the same time maximizing the adhesion function from Equation 3.4. Many different methods have been proposed to solve such multi-objective optimization problems efficiently. The simplest approach is to combine the different objectives in a single aggregate objective function. This solution is only acceptable if the scale of both objectives is known, so that they can be combined into a formula that gives each objective equal weight. Fortunately, in the case of the cohesion and adhesion objectives, these weights are known.

For points lying on a sphere with a given radius, the cross-polytope is the optimal configuration, maximizing the adhesion value. This means that any given neighbourhood with cohesion value $C(N(\mathbf{p}_r))$ must always have a lower adhesion value than the cross-polytope with radius $C(N(\mathbf{p}_r))$. A cross-polytope with radius $C(N(\mathbf{p}_r))$ has an adhesion value of $\sqrt{2}C(N(\mathbf{p}_r))$, because, in a cross-polytope configuration, the distance between points (the adhesion) is $\sqrt{2}$ times larger than the distance from the origin (the cohesion) for any dimension higher than 1. Hence, $\sqrt{2}C(N(\mathbf{p}_r))$ is the absolute upper bound for the adhesion value of any neighbourhood with cohesion $C(N(\mathbf{p}_r))$. Based on this property, we can now describe how much a given neighbourhood resembles a cross-polytope by the following measure:

$$R(N(\mathbf{p}_r)) = \frac{A(N(\mathbf{p}_r))}{\sqrt{2}C(N(\mathbf{p}_r))} \qquad d > 1 \tag{3.5}$$

If $R(N(\mathbf{p}_r)) = 1$ for a neighbourhood, the neighbourhood must form a perfect cross-polytope configuration. If the score is 0, all points of the neighbourhood lie in the exact same spot, reducing the adhesion value to zero. This measure is called the *cross-polytope ratio*, and indicates how much a neighbourhood resembles a cross-polytope.

The 1D case forms a unique exception on this rule, as the distance of the two points from each other is twice the distance from the origin, instead of $\sqrt{2}$. Additionally, in the 1D case, there exists an infinite number of configurations which maximize the adhesion value for any given cohesion value c: any two points $x, -2c + x$ with $0 \leq x \leq c$ will result in a maximized adhesion value, since $|(-2c + x) - x| = 2c$. So we propose an alternative measure, which exhibits similar behaviour in the one-dimensional case as Equation 3.5 does in general the d-dimensional case ($d > 1$):

$$R(N(\mathbf{p}_r)) = 1 - \frac{|\mathbf{p}_{r1}+\mathbf{p}_{r2}|}{|\mathbf{p}_{r1}|+|\mathbf{p}_{r2}|+|\mathbf{p}_{r1}-\mathbf{p}_{r2}|} \qquad d = 1. \tag{3.6}$$

To illustrate the behaviour of the cross-polytope ratio, we now consider the case of finding the optimal neighbourhood for a sample $\mathbf{p}_r = 0$ in a 1D design space. Assume that one sample $\mathbf{p}_{r1} = 1$ is already added to the neighbourhood of $\mathbf{p}_r$. The cross-polytope ratio is illustrated in Figure 3.4(a). This plot shows the cross-polytope ratio when the second neighbour is moved over the domain while the first neighbour is kept fixed at 1. As expected, the function is maximized at location -1, because this will result in a perfect cross-polytope neighbourhood. For any positive value x, the score at $-x$ is always better than the one at x, which illustrates that samples that lie on the opposite side of the fixed neighbour are preferred, because they add more information.

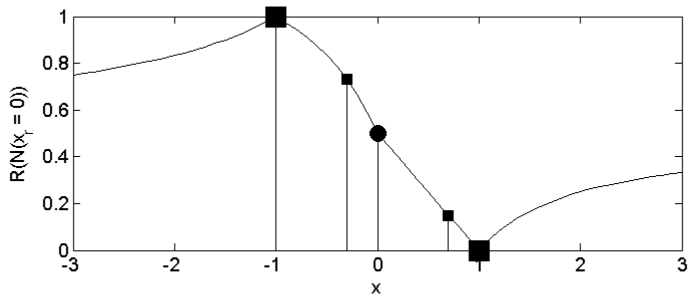
The cross-polytope ratio has some useful and desirable properties: sampling on the "unsampled" side is highly encouraged and samples near a cross-polytope configuration are prefered over samples far away. However, this metric has a serious drawback, as it completely ignores distance from the reference point when scoring neighbourhoods.

This is also illustrated in Figure 3.4(a), where 4 candidate neighbours are visualized as squares. The two big squares are the ones that are selected as neighbours, because they form a perfect cross-polytope. However, it is clear that the two points that lie closer to the origin form the best neighbourhood and convey much more information about the environment of the reference sample $\mathbf{p}_r = 0$ than the two that were selected. Because of this, the cross-polytope ratio cannot be used directly as a measure for selecting a suitable neighbourhood. In order to solve this issue, the distance of the candidate neighbours from the origin must be taken into account.

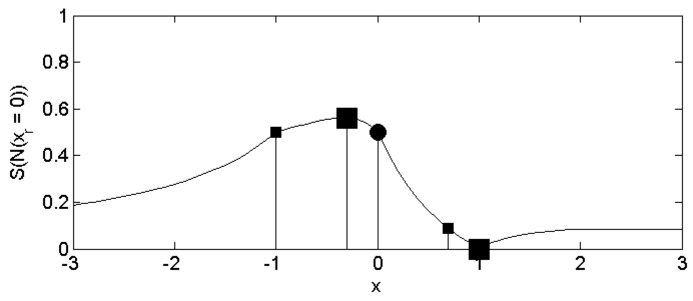
3.3.2.3 The neighbourhood score

We define the *neighbourhood score* as follows:

$$S(N(\mathbf{p}_r)) = \frac{R(N(\mathbf{p}_r))}{C(N(\mathbf{p}_r))}. \tag{3.7}$$



(a) Cross-polytope ratio



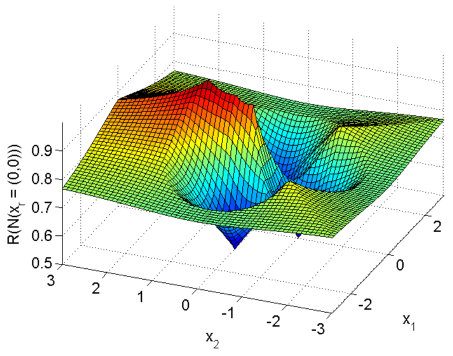
(b) Neighbourhood score

Figure 3.4: Cross-polytope ratio and neighbourhood score for reference sample 0, with one neighbour fixed at 1 in a 1D design space. Three other candidate neighbours are drawn as squares. The candidates with respectively the highest cross-polytope ratio and neighbourhood score are drawn as larger squares.

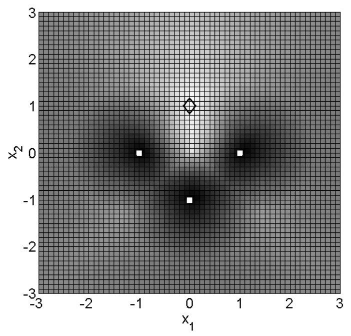
By dividing the cross-polytope ratio by the cohesion value $C(N(\mathbf{p}_r))$ of the neighbourhood, a measure is acquired that prefers neighbourhoods that resemble a cross-polytope as well as neighbourhoods that lie closer to the reference sample $\mathbf{p}_r$.

The neighbourhood score is shown in Figure 3.4(b) for the one-dimensional case. By using the neighbourhood score (instead of the cross-polytope ratio), samples lying closer to the origin are given preference over samples that lie further away. However, the key properties of the cross-polytope ratio are maintained: samples on the unsampled side are still preferred over samples near other neighbours. In Figure 3.4(b), the sample closer to 0 is now chosen as the second neighbour instead of the sample at -1 as in Figure 3.4(a).

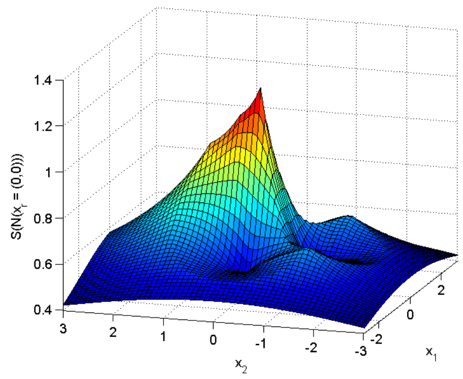
In Figure 3.5, the cross-polytope ratio and the neighbourhood score are shown for the two-dimensional case. In case of the cross-polytope ratio (Figure 3.5(a)), the surface is maximized at $(0,1)$. In case of the neighbourhood score function (Figure 3.5(c)), the surface is maximized at $(0,0)$. In both cases, the surface reaches a local minimum at the three fixed neighbours, which makes sense, since adding another point at the same location will not provide any additional information. Furthermore, both functions strongly prefer new samples around $(0,1)$. It is clear that the neighbourhood score shows similar behaviour in the two-dimensional case as in the one-dimensional case. Because the fundamental properties of the cross-polytope do not depend on its dimensionality, the neighbourhood score behaves the same in higher dimensions.



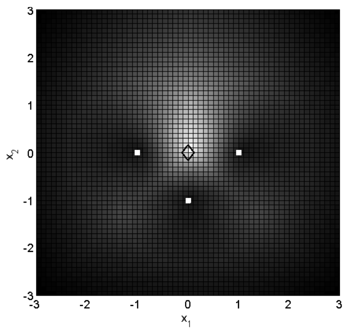
(a) Cross-polytope ratio.



(b) Cross-polytope ratio surface plot.



(c) Neighbourhood score.



(d) Neighbourhood score surface plot.

Figure 3.5: The cross-polytope ratio and the neighbourhood score in a 2D design space with three neighbours fixed at $(-1,0),(1,0),(0,-1)$. The fourth neighbour is moved over the domain and the resulting cross-polytope ratio and neighbourhood score is displayed in Figures 3.5(a) and 3.5(c), respectively. The global maximum is indicated by a diamond on the surface plots, while the fixed neighbours are drawn as squares.

To further illustrate how the neighbourhood score function behaves in different circumstances, Figure 3.6 and Figure 3.7 contain 4 different situations, in which 3 neighbours are fixed while the fourth is moved over the domain, and the neighbourhood score is computed for the 4 points. Note that, in all cases, points near the origin are strongly prefered. However, it is also clear that preference is given to points that lie in the direction the farthest away from existing points, and preferably in the direction that most closely resembles a cross-polytope.

3.3.3 Gradient estimation

Based on the neighbourhood score (as defined in the previous section), we can select a good set of neighbours for each reference sample $\mathbf{p}_r$, so that the neighbourhood provides a proper coverage of the design space in each direction. These neighbours may not be the samples closest to $\mathbf{p}_r$, but they provide more information about the behaviour of the system around the reference sample than other neighbourhoods with potentially lower cohesion values.

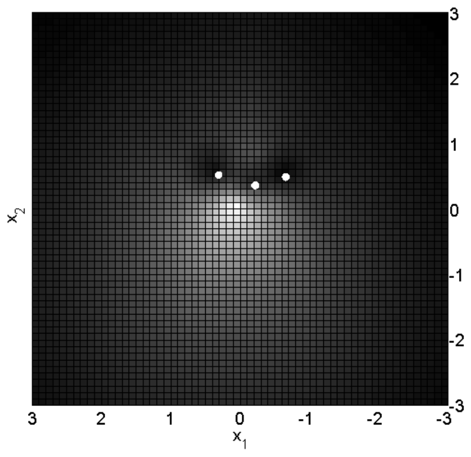
Once a set of suitable candidate neighbours has been chosen, estimating the gradient becomes straight-forward. We define the neighbourhood for reference sample $\mathbf{p}_r$ as $N(\mathbf{p}_r) = \{\mathbf{p}_{r1}, \mathbf{p}_{r2}, \ldots, \mathbf{p}_{rm}\}$, with $m = 2d$, as explained earlier. The gradient at $\mathbf{p}_r$ is estimated by fitting a hyperplane through $\mathbf{p}_r$ and its neighbours. To ensure that the hyperplane goes exactly through $\mathbf{p}_r$, the following system is solved using least squares:

$$\begin{pmatrix} p_{r1}^{(1)} - p_r^{(1)} & p_{r1}^{(2)} - p_r^{(2)} & \ldots & p_{r1}^{(d)} - p_r^{(d)} \\ p_{r2}^{(1)} - p_r^{(1)} & p_{r2}^{(2)} - p_r^{(2)} & \ldots & p_{r2}^{(d)} - p_r^{(d)} \\ \vdots & \vdots & & \vdots \\ p_{rm}^{(1)} - p_r^{(1)} & p_{rm}^{(2)} - p_r^{(2)} & \ldots & p_{rm}^{(d)} - p_r^{(d)} \end{pmatrix} \begin{pmatrix} g_r^{(1)} \\ g_r^{(2)} \\ \vdots \\ g_r^{(d)} \end{pmatrix} = \begin{pmatrix} f(p_{r1}) \\ f(p_{r2}) \\ \vdots \\ f(p_{rm}) \end{pmatrix} \tag{3.8}$$

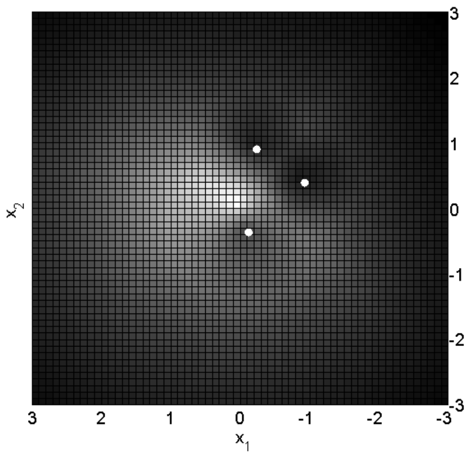
where $\mathbf{p}_{ri} = (p_{ri}^{(1)}, p_{ri}^{(2)}, \ldots, p_{ri}^{(d)})$ is the i-th neighbour for $\mathbf{p}_r$ with evaluated value $f(\mathbf{p}_{ri})$ and $\mathbf{g} = (g_r^{(1)}, g_r^{(2)}, \ldots, g_r^{(d)})$ is the gradient that is being calculated. This system can be inverted, and will result in the hyperplane which minimizes the distance from the neighbours in a least squares sense. This results in the best linear approximation for the neighbours, which will eventually converge to the best local linear approximation at the reference sample as the neighbours lie closer to the reference sample.

Because of the way the neighbours $\mathbf{p}_{ri}$ are chosen, the matrix from Equation 3.8 is always well-conditioned. This can be seen from the fact that, in a perfect cross-polytope configuration, all the vectors $\mathbf{p}_{ri} - \mathbf{p}_r$ are orthogonal with respect to each other, and therefore the matrix composed of these vectors is well-conditioned. Because the neighbourhood selection algorithm produces a neighbourhood that resembles a cross-polytope as closely as possible, the resulting matrix is also well-conditioned.

It is very important to have a good neighbourhood in order to get a good estimation of the gradient. If all neighbours lie in the same direction, it becomes impossible to make a good estimation of the gradient, since the hyperplane will be completely biased towards the behaviour of the system near the neighbours. This is why a lot of attention is paid to proper neighbourhood selection in the LOLA algorithm.

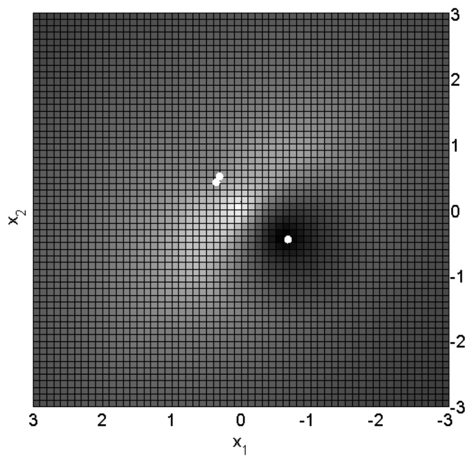


(a) The 3 fixed points are located at the same side of the origin. A very strong preference is given to points on the other side; however, the exact location does not matter very much.

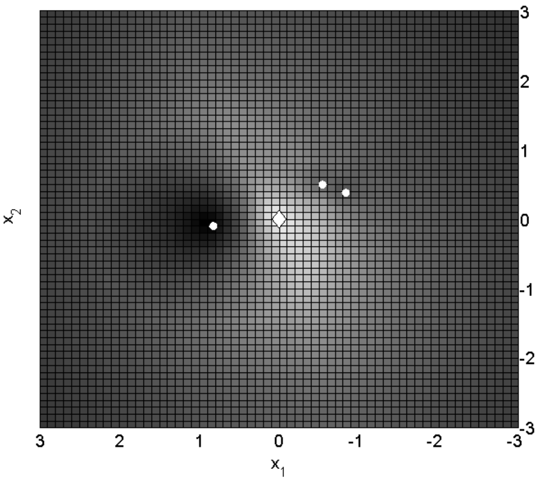


(b) The 3 fixed points are placed in a rather nice cross-polytope configuration. The location of the fourth point is prefered where it completes the cross-polytope.

Figure 3.6: The neighbourhood score in a 2D design space with three neighbours fixed at given locations. The fourth neighbour is moved over the domain and the neighbourhood score is displayed shown.



(a) The 3 fixed points are placed all on the same diagonal. Points which lie on the other diagonal are strongly prefered to those in other locations, because they will contribute the most new information.



(b) The 3 fixed points are all placed quite close to the horizontal line going through the origin. Points on the vertical line are prefered; however, points below the origin are prefered to those above, because two points are placed above the origin and only one is placed below.

Figure 3.7: The neighbourhood score in a 2D design space with three neighbours fixed at given locations. The fourth neighbour is moved over the domain and the neighbourhood score is displayed shown.

3.3.4 Nonlinearity measure

Once the gradient estimation available, it is possible to estimate the (non-)linearity of the system behaviour around the reference sample. The local nonlinearity of the system can be estimated from the normal of the hyperplane using the following formula:

$$E(\mathbf{p}_r) = \sum_{i=1}^{m} \left| f(\mathbf{p}_{ri}) - (f(\mathbf{p}_r) + \mathbf{g} \cdot (\mathbf{p}_{ri} - \mathbf{p}_r)) \right| \tag{3.9}$$

This formula computes how much the response at the neighbours differs from the local linear approximation that was computed earlier. The nonlinearity measure $E(\mathbf{p}_r)$ can now be used to get an idea of how nonlinear the function behaves in the area around the reference sample $\mathbf{p}_r$, using solely previously evaluated samples to compute this estimation. Furthermore, this approach works both for real outputs and for complex outputs, making the LOLA algorithm suitable in both cases without requiring a change in the algorithm.

Also note that the gradient estimation and calculation of the nonlinearity measure are the only places in the algorithm where the actual simulator output is used; the rest of the algorithm works on input values only and, if desired, some ratios and scores can be precalculated, independent of the actual system behaviour.

3.4 Hybrid sequential design using Voronoi and LOLA

In the previous two sections, we have basically developed two different methods for ranking previously evaluated samples. The Monte Carlo Voronoi approximation method explained in Section 3.2 ranks samples according to the size of their Voronoi cells, while the LOLA algorithm discussed in Section 3.3 ranks samples according to local nonlinearity. The first method is an exploration strategy, while the second is an exploitation strategy. By combining these two metrics, we can counteract the disadvantages of both approaches, and deliver a solid, robust and flexible sampling strategy.

In order to properly combine the two measures, they first have to be normalized. The Voronoi cell size is already in the range $[0,1]$, because it represents which portion of the design space is contained within each Voronoi cell. The nonlinearity measure, however, is initially not scaled to $[0,1]$. Therefore, the hybrid score for a sample $\mathbf{p}_i \in P$ is computed using the following formula:

$$H(\mathbf{p}_i) = V(\mathbf{p}_i) + \frac{E(\mathbf{p}_i)}{\sum_{j=1}^{n} E(\mathbf{p}_j)}. \tag{3.10}$$

The LOLA-Voronoi sequential design strategy is described in pseudocode in Algorithm 7. First, the nonlinearity measure $E(\mathbf{p})$ is calculated using the LOLA algorithm. Then, the Voronoi cell size $V(\mathbf{p})$ is computed using the Voronoi approximation, as defined in Algorithm 2. These two measures are then combined into a single value, and this value is used to rank all the samples according to how undersampled their environment is. Finally, n_{new} new samples are selected around the samples which are ranked the highest. This is done by generating a number of random points in the Voronoi cell of $\mathbf{p}_i$ and picking the one farthest away from $\mathbf{p}_i$ and its neighbours. This process can be sped up by reusing the points that were generated for the Voronoi approximation. Combining an exploration strategy with an exploitation strategy guarantees that the design space is filled up everywhere and that no large areas are left unsampled. However, nonlinear regions will be sampled much more densely, which will in turn allow the surrogate model to capture complex behaviour more easily.

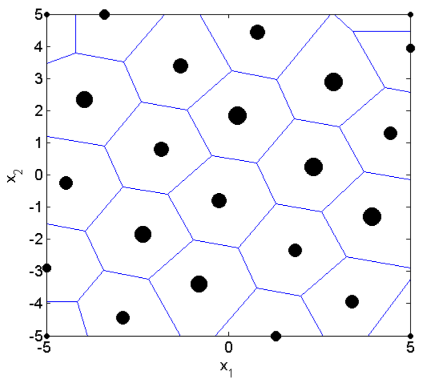
Algorithm 7 Hybrid sequential design using Voronoi approximations and LOLA. n_{new} is the number of samples requested by the user of the algorithm.

```
for all $\mathbf{p} \in P$ do
    Calculate $E(\mathbf{p})$
    Calculate $V(\mathbf{p})$
    Compute final ranking $H(\mathbf{p})$ using $E(\mathbf{p})$ and $V(\mathbf{p})$
end for
Sort $P$ by $H$
for $i = 1$ to $n_{\text{new}}$ do
    $\mathbf{p}_{new} \leftarrow$ location near $\mathbf{p}_i$ farthest from other samples
    $P_{\text{new}} \leftarrow P_{\text{new}} \cup \mathbf{p}_{new}$
end for
```

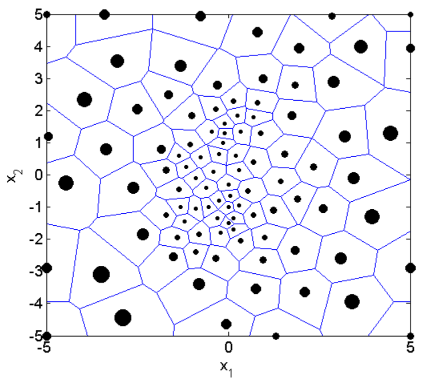
Note that the choice of n_{new} affects the quality of the design that is being generated: if n_{new} is small, more information is available to determine the location of the next sample as optimally as possible. Preferably, n_{new} should be set to 1, but higher values will also produce good designs, because at most one sample is chosen in each Voronoi cell during each iteration of the sequential algorithm, thereby ensuring a proper coverage of the design space, even for $n_{\text{new}} > 1$.

To demonstrate how effective LOLA and Voronoi are at respectively identifying nonlinear and undersampled regions, an example run was performed of LOLA-Voronoi on the 2D `Peaks` function with range $[-5,5]^2$ shown in Figure 3.20(b), which is part of a case study which will be investigated in more detail later. The relative values for each Voronoi cell for the Voronoi component V after 24 and 100 samples are shown in Figure 3.8, while the relative values for each Voronoi cell for the LOLA component E are shown in Figure 3.9. It is clear that for both low and higher number of samples, both LOLA and Voronoi can easily identify respectively nonlinear and undersampled regions.

The efficiency of the LOLA-Voronoi sequential design strategy comes at the cost of additional computing time for sample selection, which is mainly caused by the neighbourhood selection algorithm, which considers each sample as a candidate neighbour for each other sample. Every time a new sample has been evaluated, it has to be considered as a potential candidate for every other sample. This is done by replacing each current neighbour by the new candidate, and calculating the neighbourhood score. Finally, the neighbourhood is picked with the best score. This means that, eventually, every sample will have to be checked against every other sample, so the algorithm will run in $O(n^2)$ time. Whether this

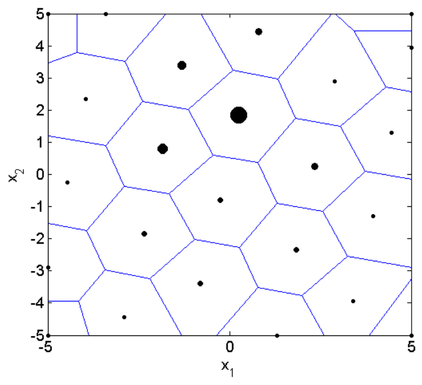


(a) 24 points

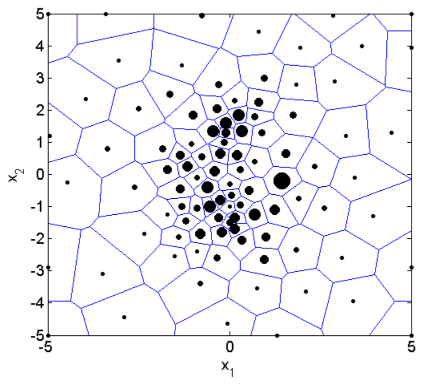


(b) 100 points

Figure 3.8: The sample distribution at the start of a LOLA-Voronoi run of the `Peaks` function in the $[-5,5]^2$ domain and after 100 points were selected by LOLA-Voronoi. Each sample is indicated by a circle, and a larger circle means a larger score on the Voronoi component of the LOLA-Voronoi algorithm. Samples with large circles lie in undersampled regions of the design space. After 100 samples, large portions of the design space are highly undersampled, because the LOLA component pushes sample selection towards the nonlinear regions. However, note that even in the nonlinear region in the center, samples are still quite evenly distributed relative to each other, thanks to the Voronoi component.



(a) 24 points



(b) 100 points

Figure 3.9: The sample distribution at the start of a LOLA-Voronoi run of the `Peaks` function in the $[-5,5]^2$ domain and after 100 points were selected by LOLA-Voronoi. Each sample is indicated by a circle, and a larger circle means a larger score on the LOLA component of the LOLA-Voronoi algorithm. Samples with large circles lie in nonlinear regions of the design space. Note that, even with very small sample sizes and the little information they provide, LOLA already efficiently identified the nonlinear region.

is an issue or not depends largely on the problem at hand. If evaluating samples is very expensive (minutes, hours or even days), the additional computing time for the sequential sampling process may be negligible compared to the sample evaluation time. However, if acquiring new data is relatively cheap and the number of data points is large, the neighbourhood selection algorithm can severely slow down the overall modelling process.

3.5 Algorithm optimization

Even though a straight-forward implementation of the LOLA-Voronoi algorithm will be quite slow, several optimizations can be introduced that significantly speed up the entire algorithm. Because the Voronoi component was already heavily optimized by using a Monte Carlo approach, the focus of this section will be on optimizing the LOLA component, and more specifically, the neighbourhood calculation.

3.5.1 Pre-processing of neighbourhood score function

When the neighbourhood update function is programmed in a straight-forward manner, we get a time complexity of $O(2^{2d} n_{\text{new}} n)$ for the pre-processing step where 2^d is the neighbourhood size, n the amount of evaluated samples and n_{new} is the amount of newly evaluated samples that need to be considered for all neighbourhoods. This can be seen when the adhesion calculation part of the algorithm is described as follows:

Algorithm 8 Algorithm for neighbourhood score calculation.

```
for all $\mathbf{p}_{\text{new}} \in P_{\text{new}}$ do
    for all $\mathbf{p} \in P$ do
        for all $\mathbf{p}_{\text{replace}} \in N(\mathbf{p})$ do
            Replace $\mathbf{p}_{\text{replace}}$ by $\mathbf{p}_{\text{new}}$ in $N(\mathbf{p})$
            for all $p_{\text{neighbour}} \in N(P)$ do
                Calculate distance of $\mathbf{p}_{\text{neighbour}}$ from $\mathbf{p}_{\text{new}}$
            end for
        end for
    end for
end for
```

Note that this is not the entire algorithm, but just the most computationally intensive part to illustrate where the computational complexity comes from. The three outermost loops are unfortunately impossible to optimize; to find the optimal neighbourhood, all combinations of samples have to be considered. However, with clever use of built-in Matlab functions, the most inner loop can be flattened and reduced to a few operations. Even though this does not really change the complexity of the algorithm, but merely hides part of it behind the scenes, in practice, the complutational time is indeed reduced by a factor 2^d due to the extremely efficient execution of the built-in Matlab functions compared to manual code.

The calculation that will be optimized is the following:

$$A(N(\mathbf{p}_r)) = \frac{1}{m}\sum_{i=1}^{m} min\{\|\mathbf{p}_{ri} - \mathbf{p}_{rj}\| \,|\, j \neq i\}.$$

where $m = 2^d$ as explained in Section 3.3.2. When this equation is implemented directly as a loop, all the subtractions $x_i - x_j$ are performed twice. This can be eliminated by doing all the subtractions once outside of the loop. At initialization time, we generate the following matrix and store it for further use:

$$\begin{array}{cc}
1 & 2 \\
1 & 3 \\
\vdots & \\
1 & n \\
2 & 3 \\
2 & 4 \\
\vdots & \\
n-1 & n
\end{array}$$

We then generate, for each i, an array of indices for rows that contain i. By pre-calculating these arrays, we can simply perform all subtractions at once, and then reference, for each i, to the appropriate values in the array. This can be done in only 3 lines of Matlab code with the following instructions:

```
distanceArray = neighbourhood(s.neighbourhoodSubLeftSide,:) - ...
    neighbourhood(s.neighbourhoodSubRightSide,:);
distanceArray = dot(distanceArray, distanceArray, 2);
distances = sqrt(min(distanceArray(s.neighbourhoodSubIndexArray), ...
    [], 2));
```

3.5.2 Adding a "too far" heuristic

The vast majority of the computation time of the LOLA-Voronoi algorithm is spent on neighbourhood selection. It is therefore obvious to focus optimizations on this part of the code. The following extremely simple optimization will increase the overall performance of the algorithm by over 50% by not computing the cohesion and adhesion for a lot of potential neighbourhoods, based on a heuristic.

When constructing the algorithm, we have assumed that we want neighbourhoods that resemble a cross-polytope shape as closely as possible, yet give preference to smaller neighbourhoods. In other words, the cohesion must be slightly more important than the adhesion. This also means that, when one candidate neighbour lies a certain distance from the sample, it will become impossible for a neighbourhood containing this candidate to have a good score. The fact that cohesion is given more weight than adhesion, rules our very large neighbourhoods immediately.

Yet, in the original algorithm, all possible combinations of samples are considered as candidate neighbourhood sets, independent of their distance from the reference sample. This is why we introduce an additional heuristic which will immediately discard any samples that lie too far away before evaluating the neighbourhood score function for any candidate neighbourhoods containing this sample. This is shown in Algorithm 9.

Algorithm 9 Adding a "too far" heuristic to the LOLA algorithm to speed up computation.

```
for all $\mathbf{p}_{\text{new}} \in P_{\text{new}}$ do
  for all $\mathbf{p}_r \in P$ do
    if $\|\mathbf{p}_r - \mathbf{p}_{\text{new}}\| < 3.7985 \max_i(\|\mathbf{p}_r - \mathbf{p}_{ri}\|)$ then
      Consider $\mathbf{p}_{\text{new}}$ as new candidate neigbour for $\mathbf{p}_r$
    end if
  end for
end for
```

If the distance of new candidate neighbour $\mathbf{p}_{\text{new}}$ is more than 3.7985 the maximum distance of the current best neighbourhood, we immediately discard $\mathbf{p}_{\text{new}}$ and do not proceed with the neighbourhood score evaluation. The number 3.7985 is not a magic number, but is derived from the formula for the neighbourhood score function as follows.

Consider a neighbourhood $N(\mathbf{p}_r)$ of a reference sample $\mathbf{p}_r$, with $\mathbf{p}_{r1}$ lying in the worst possible location relatively close to the origin, which is right on top of another sample in the neighbourhood. We assume $\mathbf{p}_{r1} = \mathbf{p}_{r2}$. Now a (hyper)sphere can be defined that contains all the possible locations on the design space that can result in a better neighbourhood score when $\mathbf{p}_{r1}$ is replaced by a sample in this location.

This would mean that, if a new candidate would emerge in the worst-case scenario of two identical neighbours, and if this new candidate lies outside of this (hyper)sphere, it will never be able to achieve a better neighbourhood score. It is therefore futile to even calculate the score function, as it is already known in advance that any neighbourhood containing the new candidate will be inferior.

In order to determine the radius of the hypersphere (and hence the value for the distance check), it must be known at which distance it is completely impossible to get a better score than a worst-case scenario. Assume a fixed cohesion value $C(N(\mathbf{p}_r))$. This cohesion value is, by definition, lower than the maximum distance of any point in the neighbourhood from the origin:

$$MC(N(\mathbf{p}_r)) = \max_i \|\mathbf{p}_r - \mathbf{p}_{ri}\| \tag{3.11}$$

Since the adhesion value is maximized at a cross-polytope configuration, the cross-polytope ratio is so as well for a fixed cohesion value. At a cross-polytope configuration, it is also clear that $C(N(\mathbf{p}_r)) = MC(N(\mathbf{p}_r))$, since all points lie the same distance from $\mathbf{p}_r$, as this is a requirement for a cross-polytope configuration. Since any configuration for which $C(N(\mathbf{p}_r)) < MC(N(\mathbf{p}_r))$ is not a cross-polytope configuration, it is also impossible for such a configuration to have a better adhesion value. This allows us to conclude that, for any possible neighbour configuration with cohesion value $C(N(\mathbf{p}_r))$, the absolute highest possible cross-polytope ratio is achieved when the fourth point completes a cross-polytope configuration.

If all current neighbours are fixed in a given non-cross-polytope configuration, the cross-polytope ratio for any candidate neighbour that does not lie further away from $\mathbf{p}_r$ than the maximum distance of the other points, must be lower than the cross-polytope ratio of a cross-polytope with the same distance. Additionally, for a cross-polytope configuration, the maximum score when the candidate neighbour lies a given distance from the origin always lies on the line going through the origin and the point that completes the cross-polytope. This means that the cross-polytope ratio achieved by adding a candidate neighbour at a given distance is always lower than the cross-polytope ratio achieved by adding a candidate neighbour to a configuration which is already in a cross-polytope.

Hence, in order to determine the distance at which it becomes impossible to achieve a better score, no matter how good or bad the initial configuration is, an optimization can be performed on the line going through the candidate neighbour of a cross-polytope configuration, until the neighbourhood score drops below the local minimum at one of the existing neighbours. Since the neighbourhood score drops monotonely on this line, this optimization is easy to perform and very accurate. It turns out that the value at which this happens is 3.7985, so this value was picked for the heuristic.

This is illustrated in Figure 3.10. In this figure, the neighbourhood score function is plotted as in Figure 3.5(d), but zoomed out so that the hypersphere can be displayed (white circle). It can easily be seen that the neighbourhood score function outside of this circle is lower than in any of the white dots, which are the local minima around the origin. For any possible neighbourhood configuration with $MC(N(\mathbf{p}_r))$ smaller than the size of this cross-polytope, this will always be the case. If a candidate neighbour emerges which lies outside of this circle, it will therefore never be considered a viable candidate, as it can never improve the neighbourhood score. It is discared immediately, and an expensive neighbourhood score function evaluation is saved.

To demonstrate the efficiency of adding this heuristic, Figure 3.11 shows the total number of neighbourhood score calculations with and without the heuristic in place. For less than 100 samples, only a very small number of neighbourhood score calculations are saved. However, as the number of samples increases, the advantage of the heuristic becomes larger. At 1000 samples, the heuristic will have prevented 60% of the total number of neighbourhood score calculations. This percentage will further increase as the number of samples grows.

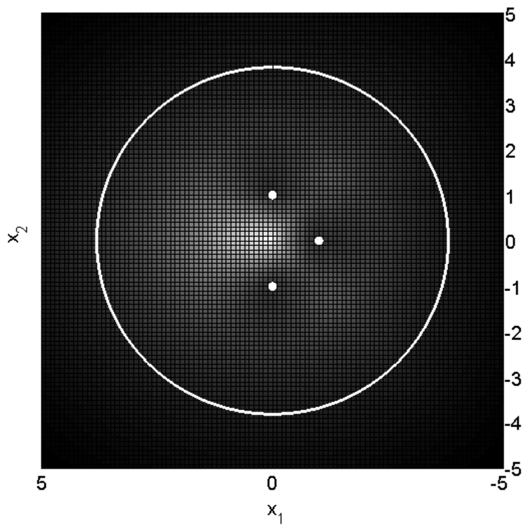


Figure 3.10: A plot of the neighbourhood score function, illustrating that, outside of the circle with radius 3.7985, the neighbourhood score function will never score higher than in the local minima at (0, 1), (0, −1) and (1, 0).

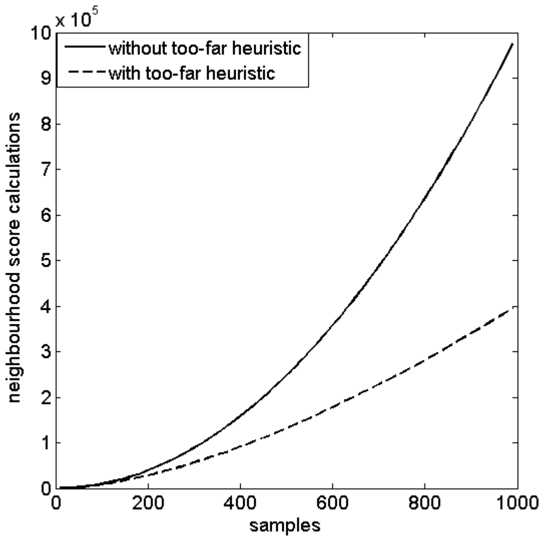


Figure 3.11: A plot of the total number of neighbourhood score calculations as a function of the total number of samples selected thus far. Note that, as the total number of samples increases, the heuristic becomes more effective.

3.6 Multiple outputs and frequency-domain parameters

Thus far, we have only considered one real or complex output associated with a set of inputs. However, in many real-life problems, multiple outputs are often associated and computed at the same time during one simulation. In this case, the engineer might want to perform one sequential design run, and model all the outputs at the same time. It is possible to do multiple sequential design runs for each output separately, thereby optimizing the sample distribution for that one particular output. But this might take too much time, as new samples will have to be simulated for each output separately.

Instead, ideally, the sequential design method looks for some kind of compromise between the different outputs, by selecting points in either undersampled regions or regions in which one or more outputs behave nonlinearly. Fortunately, because LOLA-Voronoi only uses the output of the simulator in one specific location of the algorithm (described in Section 3.3.3 and Section 3.3.4), LOLA-Voronoi can easily be adapted to deal with multiple outputs.

Consider a set of outputs $f_1(\mathbf{p}_i), f_2(\mathbf{p}_i), \dots, f_k(\mathbf{p}_i)$ associated with a samples $\mathbf{p}_i$. For each of these outputs, the gradient can be computed as described in Equation 3.8, resulting in a set of different gradient vectors $g_1, g_2, \dots, g_k$. To determine the most nonlinear location based on a number of different outputs, the nonlinearity measure from Equation 3.9 is adapted to:

$$E(\mathbf{p}_r) = \max_j \sum_{i=1}^{m} \left| f_j(\mathbf{p}_{ri}) - (f_j(\mathbf{p}_r) + \mathbf{g}_j \cdot (\mathbf{p}_{ri} - \mathbf{p}_r)) \right|. \tag{3.12}$$

The maximum was taken as opposed to the average, because when the average is taken, LOLA does not tend to have an large effect on sampling at all. This is because, if there are a lot of outputs, every region might be nonlinear for one particular output and linear for most of the others. When averaged, every region will have about the same score. By taking the maximum, the most nonlinear regions for all the outputs are taken into account. If there are only a few outputs, taking the average can be a good alternative.

3.6.1 Frequency-domain parametrs

In electrical engineering, there is often a special-case input parameter called the frequency parameter. In many commercial software tools for this research field, the outputs for a large number of frequencies can be computed at little or no additional cost (for example, using AFS algorithms [23]). This basically means that, if one of the parameters is a frequency parameter, one simulation with given values for the other parameters will yield a whole range of outputs, for different frequency values.

There are two ways to deal with the frequency parameter. One way is to treat it like any other dimension. In this case, it will also be sampled by the sequential design algorithm like any other dimension, and the advantage of the free frequency sweep will be lost. Instead, only the requested frequency will be simulated. It is clear that this is less than ideal in the context of expensive simulations. The second way is to take this special dimension into account in the sampling process. It is important to note that the frequency dimension tends to be extremely densely sampled, since computing the outputs for different frequencies is practically free. It is not uncommon to have hundreds of different outputs for different frequency values after one simulation. This results in extremely large datasets which are impossible to work with for the methods proposed in this thesis.

In order to deal with this issue and reduce the sample size to manageable numbers, LOLA-Voronoi eliminates the frequency dimension from the sampling process. Instead, each output associated with a given frequency is considered a separate output of this reduced design space, and LOLA-Voronoi will select samples according to Equation 3.12.

3.6.2 Example

To demonstrate this approach, a real-life example from electronics will be considered. This example deals with the parametric macromodeling of the reflection coefficient of a scalable H-shaped microwave antenna. Figure 3.12 shows a 3-D view of the antenna, which consists of three layers: a top layer with the H-shaped antenna, a bottom layer with the feed line, and a middle slot layer with a rectangular aperture that realizes the coupling between the feed and the antenna.

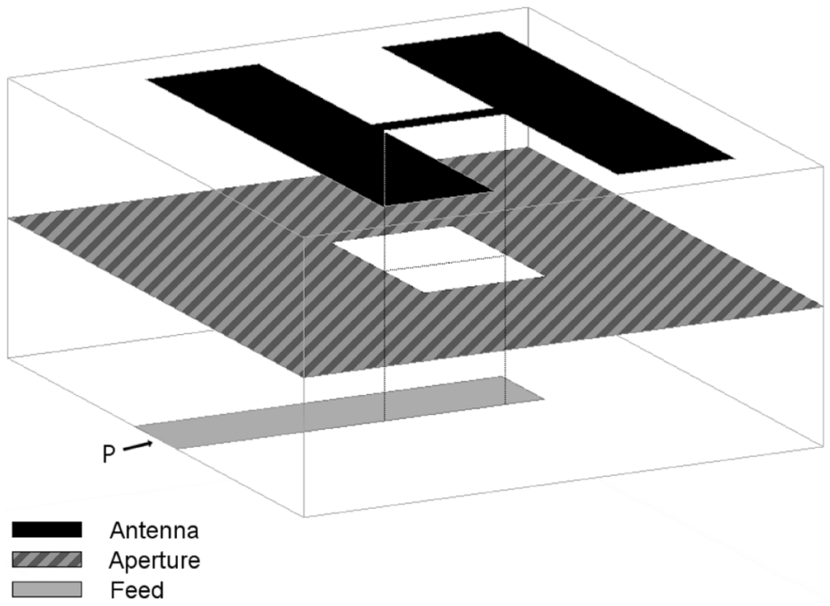


Figure 3.12: 3D view of a microwave H-antenna.

The input parameters of the model are the length L of the antenna, the width W of the aperture and the frequency f. The frequency range of interest varies between [4.5 – 5.5 GHz]. All data samples are simulated with the full-wave planar EM simulator ADS Momentum [51]. This simulator will return a very dense frequency sample at no additional cost. Therefore, the frequency parameter f is not considered a normal dimension, and the problem will be reduced to a 2D sampling problem, in which the 3D samples are grouped by their L and W values, and each frequency value f will correspond with a different output for this 2D problem.

LOLA-Voronoi was used on this simulator up to a total of 2000 points in the 2D space. The resulting sample distributions are shown in Figure 3.13. It is clear that a region near the center is much more densely sampled than other regions; this should indicate that much more dynamic, nonlinear behaviour occurs in this region than in the rest of the design space.

To validate the effectiveness of the sample distribution, the parametrized frequency response is simulated for a constant value of $W = 2.406$ mm and a varying length L. In terms of the design space, this corresponds to the horizontal solid line that is shown in Figure 3.13(d). It is seen from this figure that data points are distributed more densely if L has a value in between approximately 5 and 8 mm, as marked by the vertical dashed lines. The reason becomes clear when Figure 3.14 is considered. If L is varied in between these values, the frequency response contains a sharp resonance that moves toward the lower frequencies as the length increases. For other values of L, this resonance is located outside the frequency range of interest, leading to a smoother frequency response.

As an additional test, the frequency response is simulated for a constant value of $L = 9$ mm and a varying width W. This corresponds to the vertical solid line shown in Figure 3.13(d). Here, it is also found that the data points are distributed more densely if W has a value in between approximately 0.7 and 1.9 mm, as marked by the horizontal dashed lines. In between these values, the frequency response contains a sharp resonance that moves towards the lower frequencies as the width increases, as shown in Figure 3.15. For other values of W, this resonance is located outside the frequency range.

It is clear that, even for very large numbers of outputs (in this case caused by a frequency parameter), LOLA efficiently identifies and focuses on nonlinear regions, without neglecting the rest of the design space. Because the main cost of LOLA-Voronoi lies in calculating the neighbourhoods, additional outputs (even thousands) can be incorporated without noticeable computational cost, because the neighbourhood depends on inputs only and needs to be computed only once. This makes LOLA-Voronoi a very useful technique for modelling data with a high number of outputs.

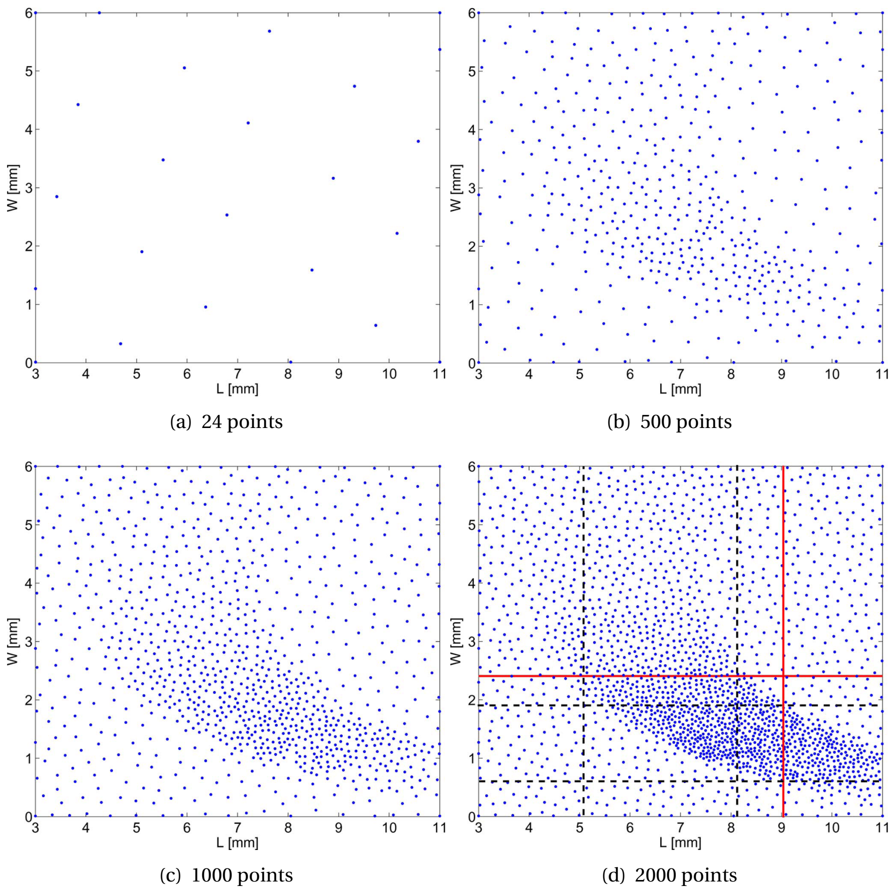


Figure 3.13: Plots of the sample distributions for respectively 24, 500, 1000 and 2000 points were selecte by LOLA-Voronoi for the H-antenna problem.

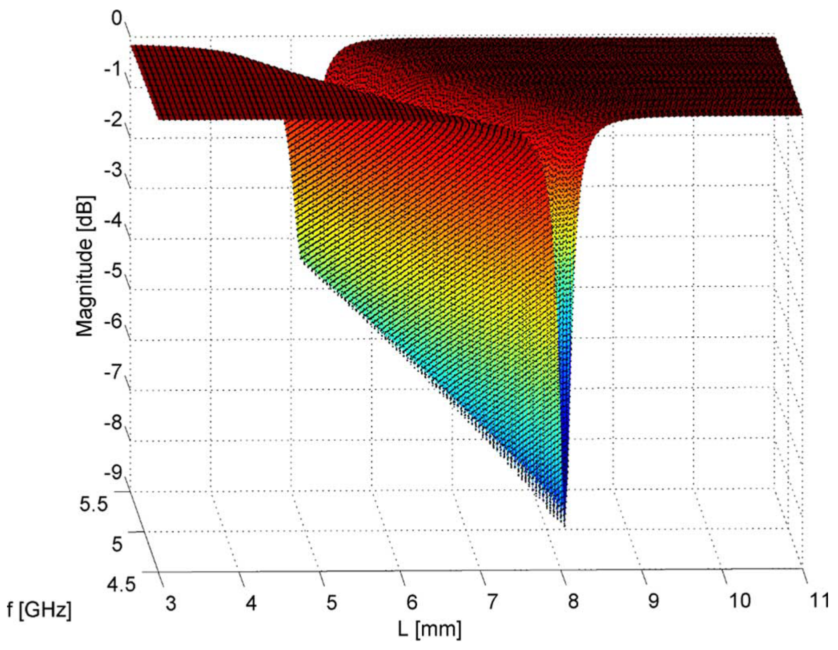


Figure 3.14: Magnitude parameterized S-parameter response for $W = 2.406$ mm.

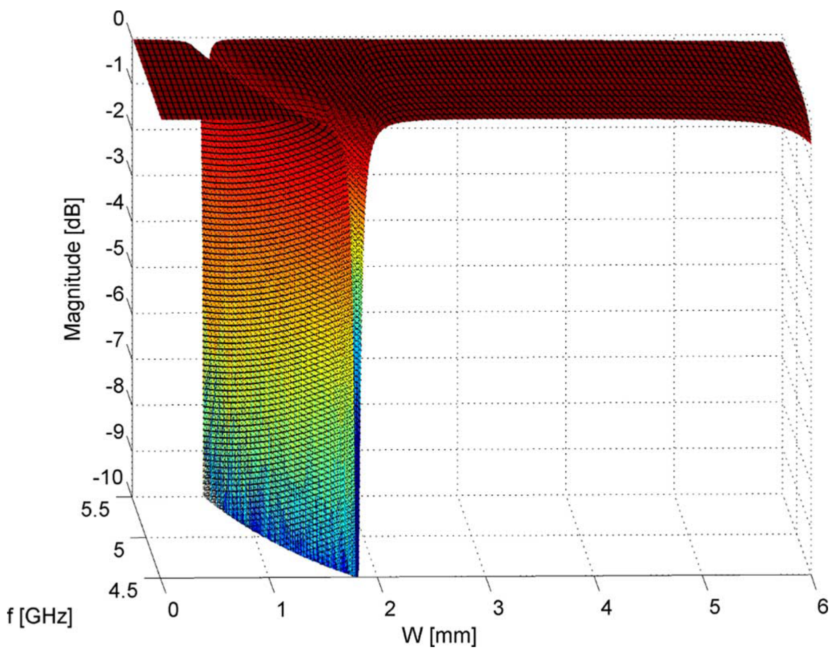


Figure 3.15: Magnitude parameterized S-parameter response for $L = 9$ mm.

3.7 Experiments

Two studies were conducted in order to investigate the efficiency of LOLA-Voronoi in different circumstances. The first study does an in-depth investigation of LOLA-Voronoi and other sampling methods in the context of a real-life problem from electronics. Because of computational limits, the total number of samples was fixed at a given number, and the accuracy of the models are measured when the total number of samples is reached.

The second study does not fix the total number of samples, but keeps selecting samples until a desired accuracy is met. This is done for three examples, each demonstrating the flexibility and robustness of LOLA-Voronoi under different conditions.

3.7.1 Other sampling methods

LOLA-Voronoi will be compared against a number of methods that have proven their merit in other studies: a pure exploitation method using the model error, an exploration method using a Voronoi tessellation and a random sampling scheme (as a base case).

The exploitation-based method first constructs a very dense, randomly perturbated grid over the entire design space (typically 2500 points, even though the number may be higher for high-dimensional problems). It then evaluates and compares the best models from the previous iterations on this grid. This is done by subtracting the outputs from these models pairwise from each other, and finding the locations on the grid where the difference is greatest, as described by Hendrickx et al. [48]. Places where the models disagree indicate locations of high uncertainty, and will be sampled next. Because surrogate models can be evaluated fairly quickly, evaluating them over such a dense grid is usually not a problem. For very high-dimensional problems, however, the number of evaluations required to get a sufficiently dense grid may be too large, which makes this sampling strategy a poor choice for high-dimensional problems. This sampling strategy is very efficient at locating areas in the design space that are difficult to approximate, such as asymptotes or discontinuous regions, enabling the modeller to quickly increase accuracy in those regions. It, however, tends to undersample large regions of the design space and generate large clusters of samples, which might result in inaccurate global models. We will from now on refer to this strategy as Model Error sampling, because it estimates the approximation error by substracting the model outputs from each other.

The exploration-based method is the Voronoi component of LOLA-Voronoi. It uses a Voronoi tessellation of the design space to find regions with large Voronoi cells. New samples are chosen in the largest Voronoi cells, as far from any other sample as possible. This method will distribute the samples evenly, which makes it very robust: if enough samples are taken, each portion of the design space will be sampled equally dense.

Finally, 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 method available. For a small sample set, however, large deviation from space-filling is to be expected, and the behaviour of this sampling scheme can be very erratic.

3.7.2 SUMO research platform

In order to compare the different sampling strategies, each method was implemented in the SUMO research platform [40, 43]. This Matlab toolbox, designed for adaptive surrogate modelling and sampling, has excellent extensibility, making it possible for the user to add, customize and replace any component of the sampling and modelling 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 a good choice for conducting this experiment. The work flow of the SUMO Toolbox was already described in Section 2.6.2 and is further discussed in Section 4.1.

3.7.3 In-depth analysis of LOLA-Voronoi with fixed sample size

LOLA-Voronoi will be tested and compared against the other methods using a test-case from electronics: a narrowband Low Noise Amplifier (LNA), which is a simple RF circuit [62]. The goal is to compare the robustness of these sampling techniques in different modeling environments for a difficult problem.

3.7.3.1 Problem description

The LNA problem was already described in detail in Section 2.6.2.3. However, this experiment was conducted earlier, and uses slightly different formulas, since they were updated later to give more accurate, useful results. Additionally, three inputs were considered in this experiment: the inductances L_s, L_m, and the MOSFET width W, as compared to the two inputs L_s and W for the previous experiment. The input noise-current $\sqrt{i_{in}^2}$ was again chosen as the output of choice because it is the most difficult to approximate. This implementation of the input noise-current is defined by Equations 3.13 to 3.17.

The remaining parameters have been set to the following:

$$C'_{gs} = 1 \cdot 10^{-9}\mathrm{Fm}^{-1},$$

$$g'_m = 100\mathrm{AV}^{-1}\mathrm{m}^{-1},$$

$$\omega = 2\pi \cdot 5 \cdot 10^{9}\mathrm{Hz},$$

$$\overline{i_{gs}^2}' = 2 \cdot 10^{4}\mathrm{pA}^2\mathrm{Hz}^{-1}\mathrm{m}^{-1},$$

$$\overline{i_{ds}^2}' = 5 \cdot 10^{6}\mathrm{pA}^2\mathrm{Hz}^{-1}\mathrm{m}^{-1}.$$

$$f_{gs,in} = \frac{1 + j\omega L_s g'_m W}{1 - \omega^2 C'_{gs} W (L_s + L_m) + j\omega L_s g'_m W} \tag{3.13}$$

$$f_{ds,in} = \frac{\omega^2 C'_{gs} W (L_s + L_m)}{1 - \omega^2 C'_{gs} W (L_s + L_m) + j\omega L_s g'_m W} \tag{3.14}$$

$$\overline{i^2_{gs}} = W \cdot \overline{i^2_{gs}} \tag{3.15}$$

$$\overline{i^2_{ds}} = W \cdot \overline{i^2_{ds}} \tag{3.16}$$

$$\sqrt{\overline{i^2_{in}}} = \sqrt{|f_{gs,in}|^2 \cdot \overline{i^2_{gs}} + |f_{ds,in}|^2 \cdot \overline{i^2_{ds}} - 2 \cdot \mathrm{Im}(0.4 f_{gs,in} f^*_{ds,in}) \sqrt{\overline{i^2_{gs}} \cdot \overline{i^2_{ds}}}} \tag{3.17}$$

A plot of the input noise-current $\sqrt{\overline{i^2_{in}}}$ for 3 input parameters is depicted in Figure 3.16. Note that the overall shape of the surface hasn't changed by adding a third parameter or by adjusting the formulas; it was only rotated and displaced a bit.

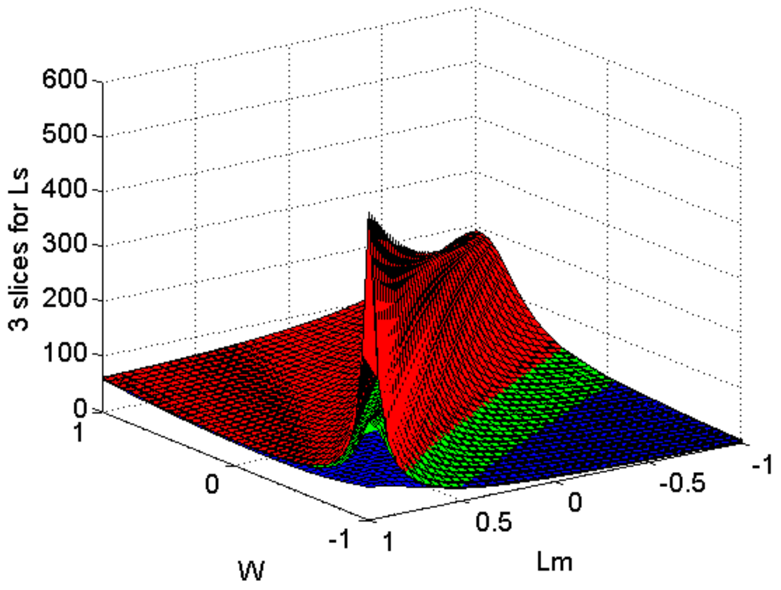


Figure 3.16: A plot of the input noise-current ($\sqrt{\overline{i^2_{in}}}$) of an LNA for inputs L_s, L_m and W. The plot shows that the function is very flat, except for a tall ridge on the diagonal.

3.7.3.2 Model types

All of these methods will be compared against each other using 4 different model types: Kriging, artificial neural networks (ANN), radial basis functions (RBF) and (least squares) support vector machines (LS-SVM). For neural networks, two implementations will be used: the default Matlab implementation (M-ANN), which is rather slow, and the much faster Neural Network Based System Identification Toolbox (NNSYSID toolbox or N-ANN) developed by Norgaard et al. [74]. The parameters of all these models are changed on the fly by SUMO to improve their accuracy.

3.7.3.3 SUMO configuration

The SUMO toolbox is configured so that each run starts with an initial design of 50 points in a Latin hypercube configuration. 25 new samples are taken after each modeling iteration, up to a total of 800 samples, after which the toolbox is halted. After each modeling iteration, the best model up to that point, along with its accuracy, is recorded on disk. In order to measure the accuracy of a model, a grid of 8000 samples (20^3) was evaluated in advance. This grid was used to calculate the root relative square error (RRSE) of the model on the grid. This means that models are evaluated by their true error, and not by an estimation of the error, to avoid bias in the results. The formula for the root relative square error is:

$$\text{RRSE} = \sqrt{\frac{\sum_{i=0}^{n}\left(x_i - \overline{x}_i\right)^2}{\sum_{i=0}^{n}\left(x_i - \tilde{x}\right)^2}},$$

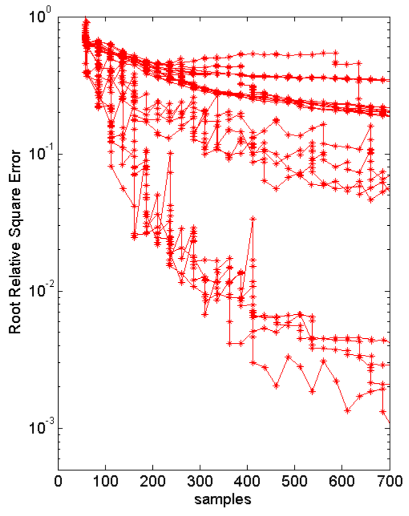
where x_i is the true value at a sample location, $\overline{x}_i$ is the estimated value and $\tilde{x}$ is the average of the true values.

3.7.3.4 Results

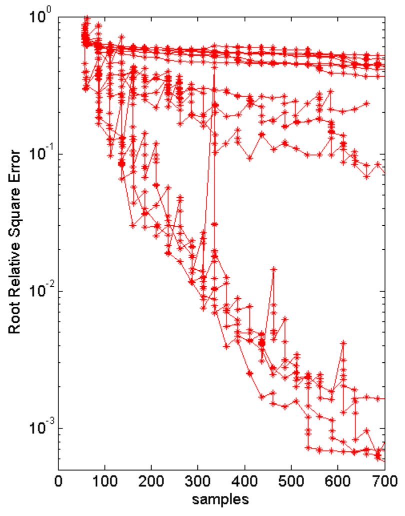
A summary of the results of the test runs can be found in Table 3.1. A plot of the root relative square error of all the runs grouped by sampling method can be found in Figure 3.17. Each plot contains a line for each test run that was performed using that sampling method. The lines indicate that the accuracy increases (smaller error) as the number of samples increases; however, the rate of improvement differs between sampling methods, and also depends on the model that was used for that run.

The overall best model was produced using the Model Error sampling scheme in combination with neural networks, resulting in a root relative square error of 4.343×10^{-4}. The neural networks toolbox of Matlab is the only model type that managed to achieve an accuracy greater than 10^{-2} or 1%. The other model types failed to achieve that level of accuracy in this difficult use-case.

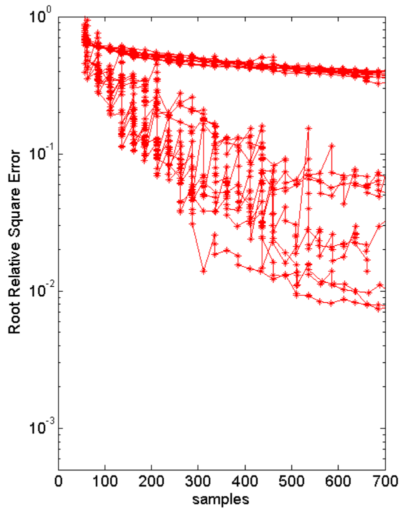
The Matlab toolbox produces very smooth models, and hence does not suffer a lot from undersampling in very flat regions. An error-based measure is therefore free to focus completely on the most dynamic area, which is in this case the ridge along the diagonal axis, without giving up accuracy in the rest of the (relatively flat)



(a) LOLA-Voronoi



(b) Model Error



(c) Voronoi

Figure 3.17: Plots of the root relative square error of all test runs as a function of time, grouped by sampling method. The Model Error method has the overall best model, but also the worst ones. The worst model produced by LOLA-Voronoi is still better than 11 Voronoi models and 8 Model Error models. This means that *half* of the total amount of Voronoi and Model Error-based runs perform worse than the worst LOLA-Voronoi run. Overall, models built using LOLA-Voronoi were considerably more accurate than models of the same type built with the other two sampling methods.

Table 3.1: Summary of test results for the 3D LNA-simulator. The best results for each model type are printed in bold. The worst results are printed in italic. M-ANN is the Matlab ANN toolbox, N-ANN is the NNSYSID toolbox. The average error is the average root relative square error of the 4 runs.

	Average Error			
	LOLA-Voronoi	Model Error	Voronoi	Random
M-ANN	0.002	**0.0008**	*0.013*	0.007
N-ANN	0.060	*0.141*	**0.049**	0.080
Kriging	**0.193**	*0.490*	0.376	0.360
LS-SVM	**0.341**	0.412	0.388	*0.429*
RBF	**0.206**	*0.502*	0.377	0.386
Total	**0.160**	*0.309*	0.240	0.253
	Standard Deviation			
	LOLA-Voronoi	Model Error	Voronoi	Random
M-ANN	0.001	0.0005	0.007	0.001
N-ANN	0.010	0.087	0.003	0.033
Kriging	0.007	0.000009	0.044	0.006
LS-SVM	0.005	0.032	0.016	0.015
RBF	0.015	0.037	0.010	0.006
Total	0.007	0.031	0.016	0.012

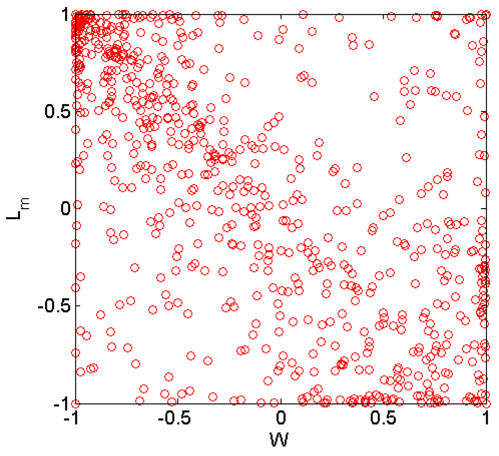
design space. This results in very accurate models. LOLA-Voronoi also focuses on the difficult areas, but still samples the flat areas as well, ensuring a minimal coverage of the entire design space. This resulted in the only case in which LOLA-Voronoi is significantly outperformed by another sampling scheme.

In the case of the NNSYSID toolbox, the results were comparable in quality for all the sampling methods, except for the Model Error sampling scheme, which did noticeably worse. Some of the Model Error runs actually failed to produce any useful models, because the sampling algorithm initially completely missed the ridge and failed to locate it. This happened because the initial experimental design (a Latin hypercube) did not generate any samples on the ridge, causing the Model Error method to focus on (relatively) uninteresting regions, instead of exploring the design space to locate the ridge. This is a good illustration of the lack of robustness of the Model Error method.

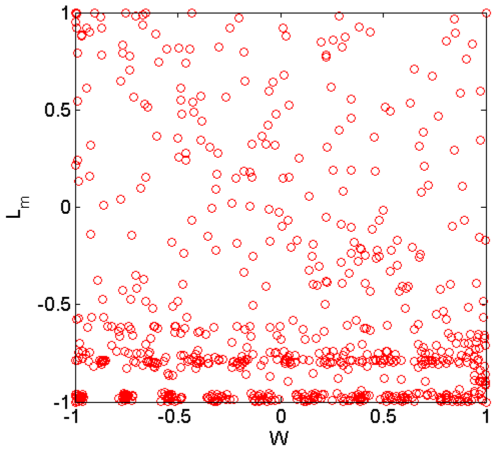
The difference between a M-ANN and an N-ANN run is illustrated in Figure 3.18. The same Latin hypercube is used to start the experiment, so in both cases, initially the ridge is missed. However, since the M-ANN model behaves very nicely in flat regions, Model Error sampling will perform a relatively uniform sampling until the ridge is identified, after which M-ANN will have great difficulty modelling this region. At this point, Model Error sampling will place lots of new samples there until the models stabilize again. For the N-ANN case, the initial models already differ much more, and because of this, Model Error will not perform uniform sampling but will focus on those regions in which the initial N-ANN models disagree the most, resulting in a much worse design.

With Kriging, LS-SVM and RBF, the best results were achieved using LOLA-Voronoi. The similarities in the results can be explained by the fact that all three methods internally use basis functions that depend on the Euclidean distance of points from each other (LS-SVM uses an RBF kernel). Therefore, a good coverage of the design space is essential to success. This explains why a purely exploitation-based sampling scheme such as Model Error performs poorly with these model types, even compared to random sampling. Exploitation-based sampling tends to concentrate on specific areas which have been identified as interesting, leaving wide gaps in the design space, which makes it difficult to produce meaningful approximations in these undersampled areas. This observation is also confirmed by Gorissen et al. [39] in a related study.

When the average is taken over all the model types, LOLA-Voronoi clearly performs the best across the board, producing models that are on average 33% better than the second best choice (Voronoi sampling). This can be seen on Figure 3.19, which groups the runs by sampling method and model type. Even though Model Error and Voronoi perform better with respectively M-ANN and N-ANN, LOLA-Voronoi follows close behind. Voronoi-based sampling is the second most robust method, performing decently in all test runs, but not excelling in any. Model Error sampling performs the worst overall, mainly due to its very poor performance with RBF and Kriging models.



(a) M-ANN



(b) N-ANN

Figure 3.18: Plots of the sample distribution of a run with Model Error sampling, for the M-ANN and N-ANN model types. The plot shows the sample distribution projected onto the W and L_m plane, clearly showing that for the M-ANN case, the ridge is sampled intensively, while it is completely ignored in the N-ANN case, where Model Error focuses on the uninteresting bottom part of the design space.

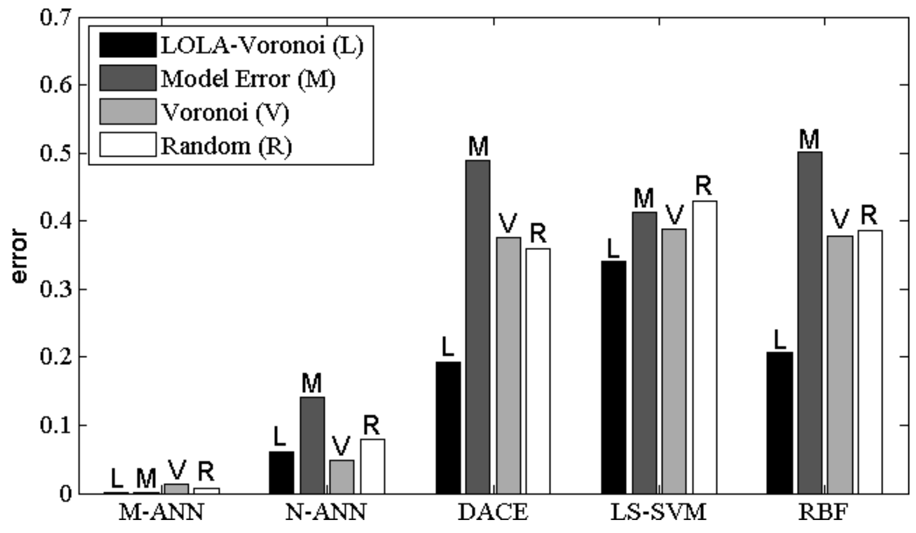


Figure 3.19: A bar chart of the root relative square error of all the runs performed in this experiment. The bars represent (from left to right) the runs performed using LOLA-Voronoi, Model Error, Voronoi and Random sampling schemes. It is clear from the chart that LOLA-Voronoi performs the best in the last three cases, and is only marginally outperformed by another sampling method in the first two. On average, it is by far the most reliable method.

3.7.4 Broad analysis of LOLA-Voronoi with fixed accuracy

Three test cases will be examined in this study, each demonstrating the quality and robustness of LOLA-Voronoi in a different context.

For each test case, we started the SUMO Toolbox run with a very sparse Latin hypercube (10 samples) augmented with a 2-level fractional design. Because we want to measure the efficiency of a sequential design strategy, the initial design is kept very small, so that the majority of the samples is chosen adaptively.

The quality 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. The root mean square error (RMSE) is defined as:

$$\mathrm{RMSE}(f,\tilde{f}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left|f(\mathbf{q}_i)-\tilde{f}(\mathbf{q}_i)\right|^2} \tag{3.18}$$

where f is the target function, $\tilde{f}$ is the surrogate model and $\mathbf{q}_i$ are the samples in the dense pre-evaluated test set. At the end of each run, the number of samples required to reach this accuracy will be recorded. To take into account 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.

3.7.4.1 Case 1: Peaks function

The first test problem is a two-dimensional function called `Peaks`, which is available in Matlab as a built-in command. The `Peaks` function is obtained by translating and scaling Gaussian distributions. It is interesting to note that the function is almost zero on the entire domain except for the region close to the origin, where it has several local optima in close proximity. In order to demonstrate the ability of LOLA-Voronoi to zoom in on nonlinear regions, the problem will be modelled on three different domains: $[-3,3]^2$, $[-5,5]^2$ and $[-8,8]^2$. We expect that the advantage of LOLA-Voronoi over the other methods will substantially increase as the domain grows, because the larger domains will contain proportionally more flat space. The `Peaks` function for all three domains is illustrated in Figure 3.20.

Because the `Peaks` function is a combination of Gaussian distributions, Kriging is a natural choice for modelling this function. Kriging was already described in Section 2.6.2. Because the correlation function for the random process $Z(\mathbf{x})$ is Gaussian, the `Peaks` function can be modelled accurately. However, the convergence rate of the modelling process greatly depends on the sampling strategy. It is expected that a sampling strategy which focuses extensively on the highly dynamic area near the origin will converge faster than a strategy which samples the design space uniformly, since many samples will be wasted on the flat regions near the edges of the design space. After each sequential step, the hyperparameters of the Kriging model are optimized dynamically.

3.7.4.2 Case 2: low-noise amplifier

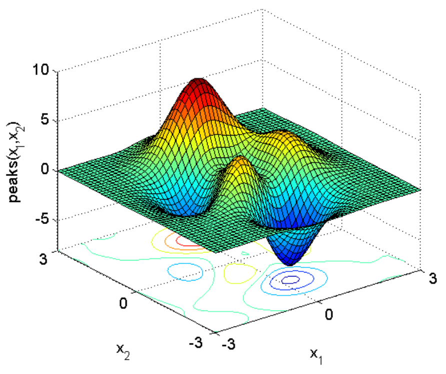
The second test problem is again the input noise-current of the LNA problem described in Section 2.6.2.3. This time, the 2D version of the problem is considered again. The 2D version of the LNA problem modelled in this experiment is shown in Figure 3.21.

To demonstrate the efficiency of LOLA-Voronoi in suboptimal conditions, the LNA problem will be modelled with three different model types. The first model type is artificial neural networks (ANN), because they have proven to be the best choice for this test case in related studies [40]. The ANN models are based on the Matlab Neural Network Toolbox, and are trained with Levenberg Marquard backpropagation with Bayesian regularization [69] (300 epochs). The topology and initial weights are optimized by a genetic algorithm. Furthermore, the LNA problem will also be modelled with radial basis function models (RBF) and rational models. In preliminary experiments, both RBF models and rational models have more difficulty modelling the LNA problem than ANN. The goal is to demonstrate that, even with a suboptimal pairing of model and problem, LOLA-Voronoi should produce better results than uniform sampling.

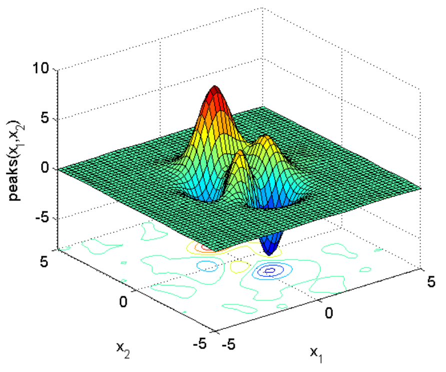
3.7.4.3 Case 3: shekel function

The third and final test case is the `Shekel` function, which is a well-known multidimensional test function from optimization [86]. We use the four-dimensional version on a $[2,6]^4$ domain, with a global optimum at $(4,4,4,4)$. In order to demonstrate the scalability of LOLA-Voronoi to higher dimensions, this problem was modelled in 2D, 3D and 4D. For the 2D case, the last two inputs were fixed at 4, while in 3D only the last input is fixed at 4. The 2D and 3D versions of the function are shown in 3.22. Note that only one nonlinear region exists around $(4,4,4,4)$. In the 3D case, for other values of z besides 4, the function remains mostly zero.

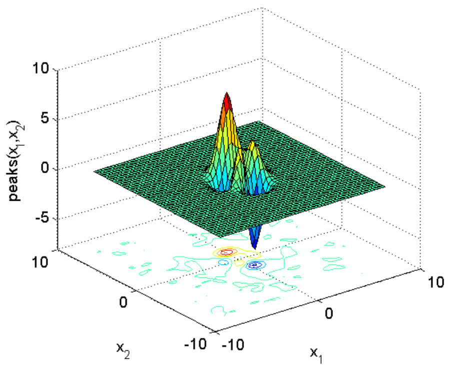
This function was modelled in all dimensions with artificial neural networks, because of their good overall performance and robustness. Because of the simple nature of the surface, it is expected that, even for higher dimensions, it should be relatively easy to model this problem using ANN.



(a) $[-3,3]$



(b) $[-5,5]$



(c) $[-8,8]$

Figure 3.20: The `Peaks` problem on the $[-3,3]^2$, $[-5,5]^2$ and $[-8,8]^2$ domains.

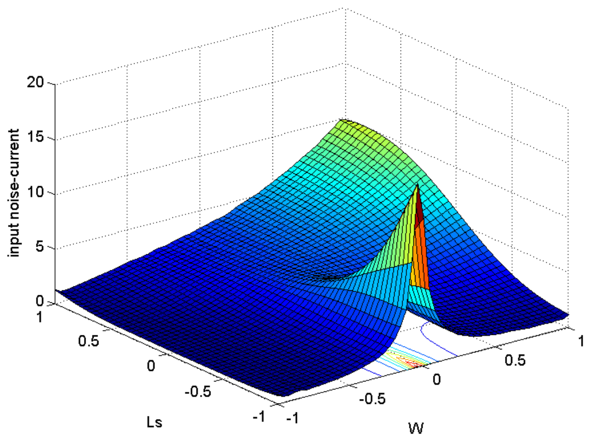
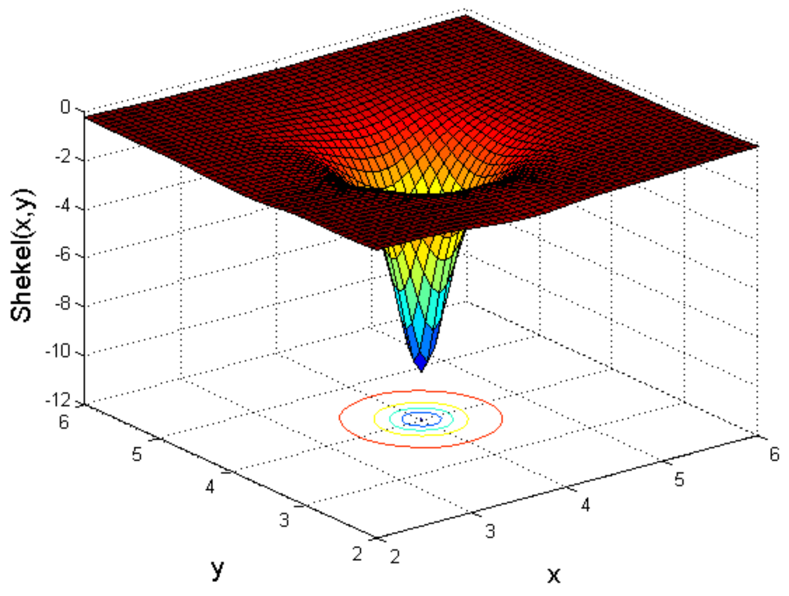
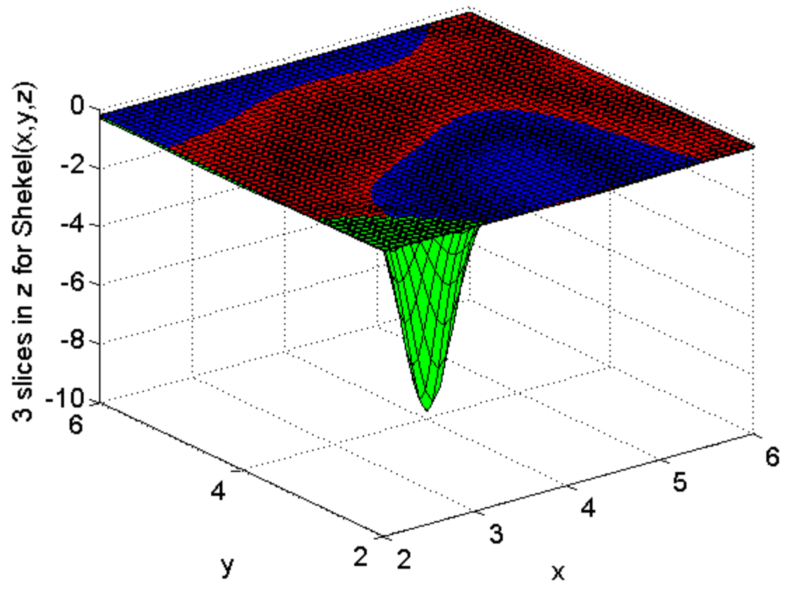


Figure 3.21: A plot of the input noise-current ($\sqrt{\overline{i_{in}^2}}$) of an LNA for inputs L_s and W. The plot shows that the function is very flat, except for a tall ridge where $W = 0$.



(a) `Shekel` 2D



(b) `Shekel` 3D

Figure 3.22: The 2D and 3D version of the `Shekel` function modelled in this experiment.

3.7.5 Results

The results of the experiments are depicted in Table 3.2.

Table 3.2: Summary of the test results of modelling `Peaks`, LNA and `Shekel` with different sampling strategies. The average number of samples required to reach a RMSE of 0.05 and the standard deviation (over 10 runs) are shown for each sampling strategy.

Sampling strategy	Peaks $[-3,3]$	Peaks $[-5,5]$	Peaks $[-8,8]$
LOLA-Voronoi	**90 ± 0**	**114 ± 4**	**135 ± 16**
Voronoi	106 ± 6	247 ± 7	516 ± 26
Model error	126 ± 9	275 ± 31	648 ± 33
Random	141 ± 14	355 ± 92	720 ± 190
	LNA ANN	LNA RBF	LNA rational
LOLA-Voronoi	**95 ± 2**	**183 ± 19**	**131 ± 26**
Voronoi	173 ± 8	> 1500 *	1112 ± 714
Model error	435 ± 74	> 1500 *	1048 ± 465
Random	263 ± 140	> 1500 *	1567 ± 756
	Shekel-2D	Shekel-3D	Shekel-4D
LOLA-Voronoi	81 ± 12	**195 ± 74**	**204 ± 69**
Voronoi	122 ± 16	448 ± 107	543 ± 103
Model error	**72 ± 6**	199 ± 51	212 ± 63
Random	134 ± 26	511 ± 158	557 ± 274

* The RBF implementation in the SUMO Toolbox can only generate models up to 1500 data points, due to memory limitations in Matlab. Since the accuracy was not reached at this point, the run was halted.

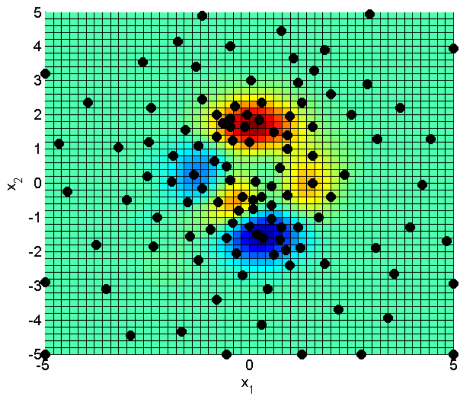
For the `Peaks` function, LOLA-Voronoi produces the best results in every test case, performing 15% better than Voronoi-based sampling, 29% better than Model Error sampling and 36% better than random sampling on the $[-3,3]$ domain. Voronoi-based uniform sampling performs much better than random sampling because the sample size is relatively small, resulting in large unsampled regions for random sampling, while Voronoi-based sampling properly covers up the design space quite uniformly. The Model Error method, even though it is an exploitation-based method, performs worse than space-filling sampling using Voronoi, but still better than random sampling. On the $[-5,5]$ and $[-8,8]$ domain, LOLA-Voronoi only requires a small number of additional samples to fill up the flat regions, while Voronoi-based, Model Error and random sampling both waste large amounts of samples on these regions, resulting in a huge difference in the total number of samples compared to LOLA-Voronoi. On the $[-8,8]$ domain, Voronoi and Model Error sampling require respectively about 3 and 5 times the number of samples that LOLA-Voronoi needs to achieve the same accuracy.

For the LNA test case, a considerable improvement can be noted as well. Because there is only one small nonlinear region, focusing heavily on this region greatly improves the accuracy. LOLA-Voronoi quickly identifies this nonlinear region and selects additional samples nearby. This allows LOLA-Voronoi to reach the same accuracy as Voronoi with only half the number of samples in the ANN case. In this test case, Model Error sampling needs considerably more samples than random sampling. This again highlights an important issue with the Model Error method: its inherent instability. Model Error sampling tends to cluster points in locations that are difficult to approximate by the model type used, leaving large portions of the design space undersampled and unexplored. If the initial experimental design does not have any samples located on the ridge, the Model Error method will focus on improving (relatively) uninteresting regions, instead of exploring the design space to locate this ridge. This can severely reduce the performance of the method.

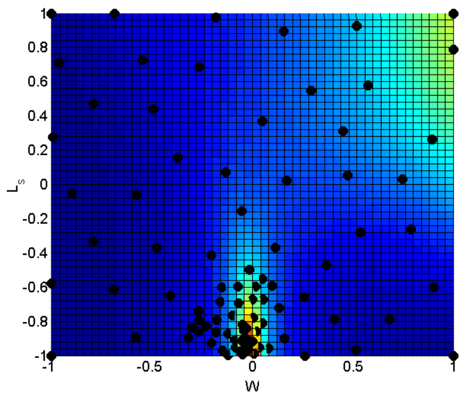
For the other model types, the difference is even more dramatic: for RBF models, LOLA-Voronoi only needs 183 samples, while the other methods fail to achieve an accuracy of 0.05 at all before reaching 1500 samples, at which point the SUMO Toolbox was aborted due to memory limitations. It is expected that, eventually, an accuracy of 0.05 can be reached, but this might take thousands of samples. For rational models, LOLA-Voronoi only needs 131 samples, while the other methods require an order of magnitude more. This example highlights the importance of focusing on difficult regions of the design space. Please note again that these results were obtained by running the toolbox 10 times for each configuration, thereby ruling out potential lucky runs.

Finally, the `Shekel` function demonstrates that LOLA-Voronoi works just as well in higher dimensions. For this test case, Model Error sampling shows that under the right circumstances, it can produce very good results, obtaining an average number of samples marginally lower than LOLA-Voronoi for the 2D case, and marginally higher for 3D and 4D. These two methods perform considerably better in 2D, 3D and 4D than Voronoi-based and random sampling. In the 4D case, LOLA-Voronoi needs less than half of the samples that Voronoi needs, and little more than one third of the samples random sampling needs. The `Shekel` function has only one nonlinear area near in the middle of the design space. In higher dimensions, the part of the design space that is completely linear is much larger than in lower dimensions. Therefore, the gap between exploitation-based and exploration-based methods is much larger in higher dimensions. Even though Model Error sampling performs very well for this last test case, the previous two cases show that it is a very unstable method, which can do very well and very poorly, and its performance is largely dependant on the problem at hand and the model type used. LOLA-Voronoi, on the other hand, works very well for all the test cases, for different model types and for multiple dimensions, due to its robust implementation of exploration and exploitation.

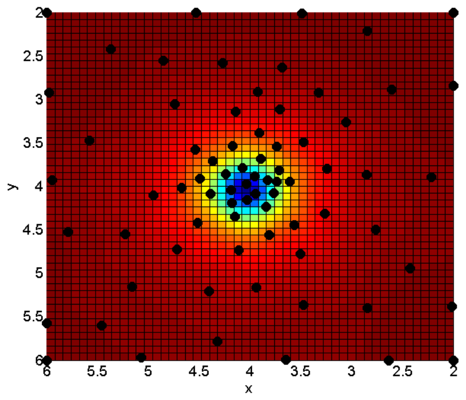
To illustrate how LOLA-Voronoi identifies nonlinear regions while still maintaining proper domain coverage, one run with the LOLA-Voronoi algorithm for each test problem is shown in Figure 3.23. Figure 3.23(a) contains all the points that were selected by LOLA-Voronoi during one of the runs for the `Peaks` problem on the $[-5,5]$ domain, while Figure 3.23(b) contains the samples selected during one run of the LNA problem. Finally, Figure 3.23(c) shows one run for the 2D version of the `Shekel` problem. It is clear that the hybrid strategy efficiently located the nonlinear regions and focused sampling heavily on those regions, without completely neglecting the other parts of the design space. In Figure 3.23(a), the flat region near the edges is sampled sparsely, but the samples are distributed quite evenly over the entire flat region. The steep slopes in the middle are sampled much more densely than gentler slopes, which are still sampled more densely than the flat regions. In Figure 3.23(b), the tall ridge is sampled much more densely than the rest of the design space. Due to this intelligent sampling approach, the average number of samples required is reduced drastically, potentially saving lots of resources and time.



(a) The Peaks function.



(b) Input noise-current $\sqrt{\overline{i_{in}^2}}$ of an LNA



(c) Shekel 2D

Figure 3.23: The sample distribution of one run of the first two test cases using the LOLA-Voronoi hybrid sampling strategy.

3.8 Conclusions

In this chapter, we proposed a novel hybrid sequential design technique that combines an exploration metric based on Voronoi tessellations with an exploitation metric using local linear approximations. We showed that LOLA-Voronoi performs better than Model Error-based, Voronoi-based and random sampling in a number of different test cases, thus demonstrating the usefulness of hybrid sequential design methods. It was shown that LOLA-Voronoi outperforms the other methods for different model types, problems and in multiple dimensions.

LOLA-Voronoi was also succesfully applied to multiple real-world test cases from different problem domains by users of the SUMO Toolbox. Fellow researchers have succesfully used LOLA-Voronoi for problems from electrical engineering [18, 37, 40, 41], aerospace engineering [37] and hydrology [14]. Because of its efficiency, it has since become the default sequential design strategy for the SUMO Toolbox.

The first study showed that LOLA-Voronoi is especially efficient for local approximation models such as RBF and Kriging, which assume a correlation between simulator output and geometrical distance in the design space. This property is magnified for problems in which the function is relatively flat in large parts of the design space and is highly dynamic in small regions. LOLA-Voronoi will quickly locate these unstable areas and focus on them, without neglecting the rest of the design space.

The Model Error method, while also very effective, is not as reliable as LOLA-Voronoi, as it may fail to locate important features of the system, resulting in useless models. Voronoi-based sampling is very reliable, but completely ignores system behaviour, and therefore tends to produce models that perform worse than LOLA-Voronoi. When using large amounts of samples, its performance is often only slightly better than random sampling. Its simplicity and reliability make it a viable choice if no better alternatives are available.

LOLA-Voronoi was designed as a very robust, reliable and widely applicable sequential design method, able to produce good results with any model type, regardless of the problem at hand. To achieve this, the only information used to guide the sampling process consists of previously evaluated samples and their output values. The robustness of the sequential design method comes at a cost, however. It is rather slow compared to other sequential design methods, mainly due to the expensive pre-processing required to estimate the gradient ($O(n^2)$ in the number of samples). However, this additional cost becomes negligible in a real-life environment in which sample evaluations may take hours or even days, and many heuristics are available to substantially reduce the overhead of LOLA-Voronoi.

CHAPTER 4

Software

Stop! The beast contained herein shall not be set free... Not even by you!
— Tyrael

Motivation

In order to properly implement and present the sequential design methods proposed in this thesis, two software packages were developed. The SUMO Toolbox is a Matlab toolbox for adaptive surrogate modelling with sequential design, while the SED Toolbox is a Matlab toolbox for generating sequential designs. Both toolboxes contain all the methods proposed in this thesis, but present them to the user in a different way. Where the SUMO Toolbox is a all-round tool that deals with the entire surrogate modelling process, the SED Toolbox is a specialized tool for generating sequential designs, easily integrated in the modelling pipeline of the user.

4.1 SUMO Toolbox

SUMO is a Matlab toolbox that automatically builds accurate surrogate models of a given data source (simulation code, data set, script, ...) within the accuracy and time constraints set by the user. In doing so the toolbox minimizes the number of data points (which it chooses automatically) since they are usually expensive. The toolbox tries to be as adaptive and autonomous as possible, requiring no user input besides an initial configuration.

However, since there is no such thing as a one-size-fits-all, the toolbox has been designed to be fully pluggable and extensible using standard object oriented design patterns. Implementations of the different components (model types, sampling strategies, model selection criteria, hyperparameter optimization algorithms,...) can be plugged in, compared, or replaced by custom implementations. In this way the SUMO Toolbox provides a common platform to easily test and benchmark different sampling and approximation strategies, while still being easy to integrate in the engineering design process.

The work-flow of SUMO is illustrated in Figure 4.1. First, an initial design is generated and evaluated. Then, a set of surrogate models is built, and the accuracy of these models is estimated using a set of measures. Each model type has several hyperparameters which can be modified, such as the order of numerator and denominator for rational models, number and size of hidden layers in neural networks, smoothness parameters for RBF models, and so on. These parameters are adjusted using a hyperparameter optimization technique, and more models are built until no further improvement can be made by changing the hyperparameters. If the overall desired accuracy has not yet been reached, a call is made to the sequential design routine, which selects a new sample to be evaluated, and the algorithm starts all over again. The algorithm is halted when the stopping condition (total number of samples or required accuracy) is reached.

In order to deal with the different problems and issues encountered during the surrogate modelling process, SUMO splits the surrogate modelling problem up into smaller subproblems, which are solved by different components. The following subproblems can be identified:

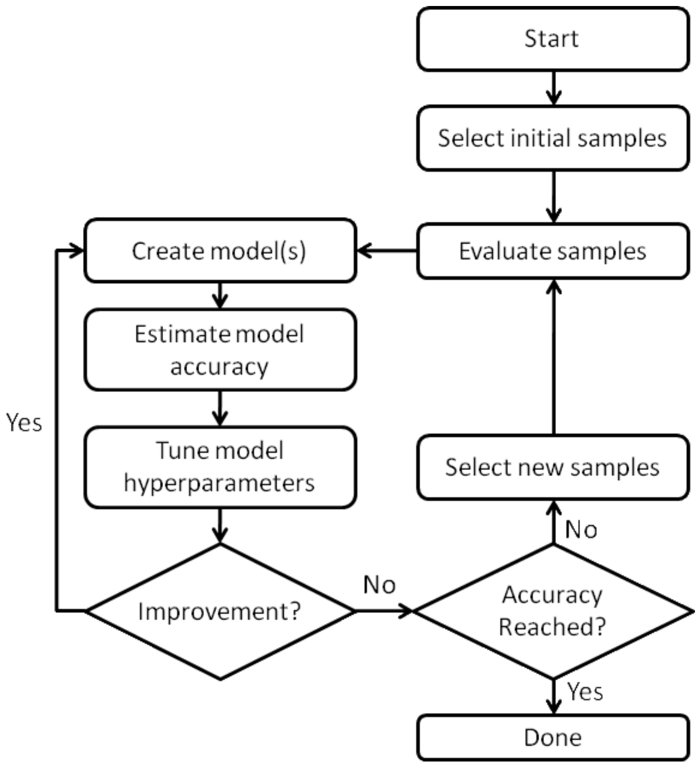


Figure 4.1: Flow-chart of the SUMO toolbox.

- Everything starts with the **initial design**. Because the simulator is assumed to be a black box, the initial design must be a space-filling design, because no information is available to base the design on besides the dimensionality of the problem. The initial design must be sufficiently large as to allow for the sequential design strategy to get a good start. For input-based sequential design methods, the initial design should be kept at a bare minimum (for example, the two corner points), while for output-based sequential design methods such as LOLA-Voronoi, the initial design should be a little bit larger (for example, 10 or 20 points), because otherwise the sequential method has no output information to base its decisions on.

- The **model type** is the next important component. Because little or no information is available about the simulator up front, choosing the right model can be a very difficult or even impossible task. SUMO features a whole set of surrogate models: Kriging, rational, polynomial, radial basis function (RBF) models, splines, artificial neural networks (ANN) and least squares support vector machines (LS-SVM). Additionally, for all of these model types, different **hyperparameter optimization algorithms** are available. To make matters worse, the right model does not only depend on the simulator; some sequential design methods work better with certain model types, so this should be taken into account as well.

- Once a model has been generated, it must be validated through some means of **model selection**. In order to determine the quality of the model, and see if it is a better model than previously generated models, many different measures are available in the SUMO Toolbox: cross-validation, external validation set, leave-one-out, model difference, etc.

- When the best model is not good enough yet, additional samples must be selected using a **sequential design method**. These methods were already thoroughly discussed in Chapters 2 and 3. Multiple methods can also be combined, by selecting part of a batch of new samples with one method and the other part with another. This allows methods to cancel out each other's issues.

SUMO is configured by means of a two configuration files:

- The **simulator config** defines the properties of the simulator. It determines the number of inputs and outputs, the type of the outputs (real, complex, ...), the executables (Matlab, native, java, ...) and datasets associated with the simulator and the constraints defined on the inputs. This config file should be constructed once to serve as an interface between the simulator files and the SUMO Toolbox.

- The **toolbox config** defines which components will be used to model a given simulator. It determines the global SUMO settings such as stopping criteria (time limit, sample limit, ...), output directory and so on. It also allows the user to select the right component for the task, and fine-tune this component if so desired.

The modular structure of the SUMO Toolbox allows the user to experiment with different component combinations to find the best match for the problem at hand. A default configuration containing all of these components with pre-configured options is distributed with the SUMO Toolbox, ready to be used in an experiment. If the user has enough knowledge of the inner workings of a given component, the user can change its settings and options through the toolbox config to fine-tune the modelling process even further. Alternatively, the user can implement (in Matlab) his own components, and add them to the toolbox config as well. All component types available in the Toolbox are exposed through simple interfaces that can be implemented to add new self-made components.

For documentation on how to download, install, configure and use the SUMO Toolbox, please refer to the website: `http://sumo.intec.ugent.be`. A detailed overview of the different subsystems and code structure of the SUMO Toolbox can be found in [35]. In the next sections, we will briefly discuss the choice of components in the context of sequential design. These sections can be used as a guide for determining the right configuration of the SUMO Toolbox for the right job.

4.1.1 Initial design

Several initial designs are available in the SUMO Toolbox. Of these, the following are the most interesting:

- `FactorialDesign` is a simple factorial. It can also be used to add the corner points of the design space to another design; this is always a good idea, because some surrogate models are only stable in the bounds defined by the outermost samples. Adding the corner points to the initial design ensures that at least the model will behave nicely inside the design space.

- `LatinHypercubeDesign` is the standard go-to initial design in most cases. Latin hypercubes have optimal projected distance properties, and, if optimized well, extremely good space-filling properties as well. The LatinHypercubeDesign implementation in the SUMO Toolbox uses a waterfall model. It automatically tries to find the best possible Latin hypercube for the number of samples and the input dimension requested. First, it checks if a pre-optimized design is available for download from the website `http://www.spacefillingdesigns.nl/`. These designs are highly optimized and have very good space-filling properties. If such a design is not currently available, or if the internet connection fails, a local cache is searched for designs previously downloaded from the website. If no design is found in the cache, a Latin hypercube is generated and optimized on the fly, using the algorithm described in [56]. While these designs are far from optimal, as demonstrated in Chapter 2, they still serve as an adequate starting point for the SUMO Toolbox.

- `DatasetDesign` allows the user to input previously evaluated samples in a new run of the SUMO Toolbox. This is useful if a previous run was aborted, and the resulting model did not suffice. By re-adding the evaluated samples as the initial design, no previously evaluated samples go to waste.

Overall, the `lhdWithCornerPoints` component defined in the default config should be the initial design of choice for most problems. By default, a Latin hypercube of 20 points augmented with the corner points is generated. Depending in how computationally intensive a simulation is, the number of points in the Latin hypercube can be lowered or raised.

4.1.2 Model type

The first concern in choosing the model type should obviously be any knowledge about the simulator. But care should be taken that the right sequential method is chosen with the right model type. Not every model type works equally well with each sequential method. We will now discuss the most popular model types available in the SUMO Toolbox, and suggest the right sequential method to go with them.

4.1.2.1 Rational models

Rational models can be highly efficient at modelling simulators that are internally based on rational functions. It is possible to achieve very accurate or even perfect models for black box simulators, but rational models also tend to produce very bad models with asymptotes in the design space [14]. Rational models work very well with most sequential design methods. However, due to the unstable nature of the model type, a Model Error sampling component should definitely be included. Model Error sampling will identify asymptotes because of the large output difference with other models, and will select samples in these regions to reduce the chance of an asymptote re-appearing there later. However, LOLA-Voronoi, due to its robust nature, is also a good choice. The `default` sequential method in SUMO is a combination of LOLA-Voronoi (70% of the samples) and model error (30%) of the samples, which is a good option for rational models.

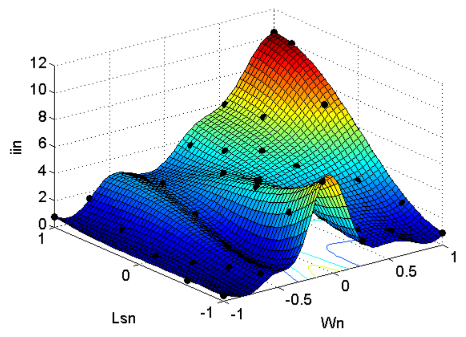
4.1.2.2 Kriging models

Kriging models have proven to be a very good choice in most circumstances, because they are very stable and interpolate between the samples, guaranteeing a 100% accuracy in the sampled locations. Kriging models internally use basis functions that depend on the euclidean distance of points from each other (LS-SVM uses an RBF kernel), so the distribution of samples is very important for the accuracy of this model type.

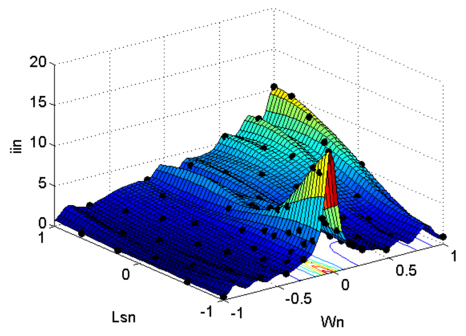
LOLA-Voronoi seems like a very good fit, but some issues arise when a lot of samples are clustered in one (nonlinear) region. When points are placed too close to each other, the correlation matrix in Kriging becomes ill-conditioned, and the model will collapse [39, 70]. Therefore, LOLA-Voronoi should be used with care for Kriging models. As long as points are not placed too close to each other, an increase in accuracy can be achieved, but once a certain threshold is passed, the models can actually become worse.

In order to demonstrate this, a test run of the SUMO Toolbox was performed for the 2D LNA problem described in Section 3.7.4.2. The results as the number of samples increases is shown in Figure 4.2. Note that after 34 samples, the model starts to resemble the correct surface already. After 84 samples, the overall shape is captured correctly, and the run seems to be converging to the correct surface. However, after 134, due to many samples placed too close to each other on the ridge, the correlation matrix has become ill-conditioned, and the model collapses. As more samples are selected, the accuracy actually drops further instead of improving.

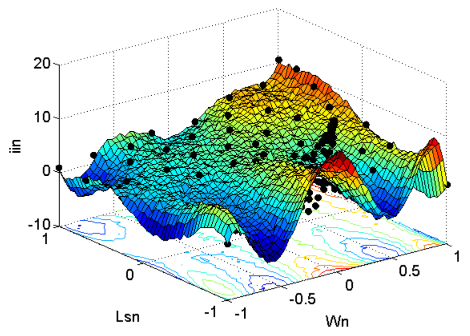
This problem does not always occur. The `Peaks` function from Section 3.7.4.1 was succesfully modelled with a combination of Kriging and LOLA-Voronoi, without any instability issues. Whether this problem occurs depends on the surface to be modelled and how much LOLA-Voronoi focuses on one area in the



(a) 34 samples



(b) 84 samples



(c) 134 samples

Figure 4.2: Plots of the Kriging surface generated after respectively 34, 84 and 134 samples were selected with the LOLA-Voronoi method.

design space. If this risk cannot be afforded, space-filling methods such as `mc-intersite-proj-th` should be considered instead. By spreading samples out over the design space, these methods will make sure the Kriging model remains stable. `mc-intersite-proj-th` is available in the default config under the name `density`.

4.1.2.3 Artificial neural networks

Artificial neural networks (ANN) are very flexible and powerful models capable of modelling extremely complex, highly nonlinear system behaviour. Training and optimizing the hyperparameters of an ANN can be extremely slow compared to Kriging or rational models. However, experiments have shown that they often do produce the most accurate models [39, 40, 41, 42], so if simulations are truly expensive, the additional computational cost of training the ANN models may still be negligible.

Several implementations of ANN models are available in the SUMO Toolbox: an implementation based on the Matlab Neural Network Toolbox, an implementation based on the Network Based System Identification Toolbox (NNSYSID toolbox) developed by Norgaard et al. [74] and an implementation based on the Fast Artificial Neural Network Library (FANN), which is available for download on `http://leenissen.dk/fann/wp/`. Of these, the Matlab Neural Network Toolbox is both the slowest and most accurate overall.

Because neural networks do not suffer from the instability issues from Kriging, LOLA-Voronoi is a very good match with them. Additionally, if the time investment to go for neural networks is already made, the additional computational cost of LOLA-Voronoi becomes irrelevant, since training neural network models is still considerably slower than selecting additional samples with LOLA-Voronoi.

4.1.2.4 Heterogeneous model builders

The idea behind heterogeneous model builders is that, since the simulator is assumed to be a black box, it is difficult in advance to predict which model type will perform well for a particular problem. One way to deal with this issue is by performing multiple runs of the SUMO Toolbox with different model types, and take the best final model. However, if a simulation is expensive, this is often infeasible, because each run will select new samples from scratch.

Instead a heterogeneous model selection approach can be adopted. A heterogeneous model builder starts with a set of model types, and will automatically look for the model type that best matches the problem at hand, by using a genetic algorithm to determine the fittest and least fit models. After a number of modelling (and sampling) iterations, certain model types will come out on top as most succesful, and these will be focused on in consequent iterations. This approach is integrated in the SUMO Toolbox, so that it can be executed automatically in one run, without having to perform multiple runs with new sample selections for each model type.

Additionally, the hetereogeneous model builder can also produce so-called ensemble models. Ensemble models are models that combine the (weighted) outputs of other (succesful) models to produce an averaged output that better matches the problem. Ensemble models reduce the impact of erroneous predictions of one of the models by averaging over a number of models, and can perform better than individual models [34, 66, 84].

Because the heterogeneous model builder will train completely different types of models during each iteration, the sampling strategy should be independent of the model type. The sampling strategy should be based solely on the inputs and outputs from the simulator, and should make no assumptions about the model type that is being used. This makes the input-based and output-based methods presented in Chapter 2 and 3 especially interesting for this model builder. These methods will produce designs that will work well with any model type currently used by the heterogeneous model builder, since it is not fine-tuned towards one particular model.

The heterogeneous model builder is very slow, because it needs to train a multitude of different models and compare them against each other. However, in a black box setting, this may very well be worth it, because the alternative is either generating a design up front and using it with many different model types, or picking one model type and going with it. Whether this time investment is worth it is up to the user.

For an in-depth analysis of the heterogeneous model builder and experimental results on its performance, please refer to [41]. The heterogeneous model builder is available in the default config as `heterogenetic`.

4.1.3 Sequential design method

The advantages and disadvantages of the different sequential design methods were already thoroughly discussed in the previous chapters. In this section, we will first briefly summarize the results from the previous chapters, and then discuss the implementation strategy used in the SUMO Toolbox for all these methods.

4.1.3.1 General guidelines

If the sequential design method needs to be fast, or if LOLA-Voronoi poses problems with the given model type, either `mc-intersite-proj-th` (available in SUMO as `density`) or `optimizer-intersite` (available as `density-optimizer`) should be used, depending on the computational time available and the dimensionality of the problem.

If the problem is highly nonlinear in certain regions and linear in others, LOLA-Voronoi (available as `lola-voronoi`) is an excellent choice. LOLA-Voronoi can, if desired, be augmented with the Model Error sampling method, so that the model itself is also taken into account during the sampling process. This hybrid approach (available as `default`) will select samples in undersampled regions, nonlinear regions or regions where the models have high uncertainty.

4.1.3.2 Implementation

All of these components are implemented as part of the sequential design framework of SUMO. This framework was designed to be extremely flexible, allowing the programmer or user to swap, combine and implement small pieces of each method. This allowed the authors to easily construct many different sequential design methods and variations of these methods, and to compare them to each other efficiently.

In order for a sequential design method to be able to interact with the SUMO Toolbox, it must implement the `SampleSelector` interface. This interface contains one simple function:

```
[this, newSamples, priorities] = selectSamples(this, state);
```

It takes the state of the SUMO Toolbox, and returns a set of new samples to evaluate, and priority values for each sample, that can be used by the sample evaluator to determine the order of evaluation. `state` is a struct containing an array of previously evaluated samples, the samples that failed to evaluate (due to an error in the sample evaluator or because the sample just can't be evaluated due to physical limitations), a list of previously built models and the number of new samples requested this iteration.

Several built-in sample selectors implement this interface directly: most notably, LOLA-Voronoi and random. The others, however, use some of the frameworks that make designing new methods much easier. Two such frameworks are currently available in the SUMO Toolbox: the `PipelineSampleSelector` and the `OptimizeCriterion`.

A flow chart of the `PipelineSampleSelector` is shown in Figure 4.3. The pipeline sample selector consists of three components, which are called in order: the candidate generator (`CandidateGenerator` interface) one or more candidate rankers (`CandidateRanker` interface) and a merge criterion (`MergeCriterion` interface). These components are implemented as separate classes, allowing the user to switch and combine each part of the pipeline sample selector at will.

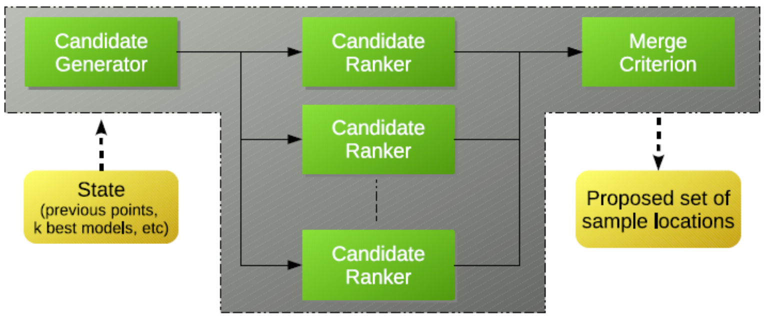


Figure 4.3: A flow chart of the `PipelineSampleSelector`.

The candidate generator must generate a number of candidates from which eventually the new samples will be selected. If constraints are specified in the simulator config, they must be respected by the candidate generator. Examples of candidate generators are a random candidate generator, a Delaunay-based candidate generator that generates the centers of the Delaunay triangles as candidates, a projected distance random candidate generator that only generates candidates in regions that respect the projected distance threshold, and so on.

The candidates generated by the candidate generator are then ranked by one or more candidate rankers. These must score and order the candidates according to some criterion. Examples of criteria are intersite and projected distance, Manhattan distance, ϕ_p, and so on.

Once the scores for each candidate are calculated, they must be combined into a single score that will serve as the final ranking between candidates. This is done by the merge criterion. The simplest merge criterion is `WeightedAverage`. This criterion will simply take the weighted average of all the scores and take this as the final score for the candidate. It will take the n_{new} best-scoring samples, and use their weighted average as the priority score for the sample evaluator.

`WeightedAverage` is a good choice if samples are selected one by one; in other words, if $n_{\text{new}} = 1$. But if $n_{\text{new}} > 1$, a problem may occur that can seriously jeopardize the quality of the design. This is illustrated in Figure 4.4. In this example, the SUMO run starts with an initial design of 50 points, and 10 additional points are requested from the sample selector. The sample selector uses a random candidate generator, and ranks the points according to the intersite and projected distance, as in the `mc-intersite-proj` method. However, because the highest ranking points tend to be clustered together near the optima, the newly selected points are not distributed properly over the design space, but clustered in 3 groups. This results in very bad space-filling designs. This problem can easily be resolved by selecting the points one by one; however, this would also mean more modelling iterations and thus a slower run of the SUMO Toolbox. The sample selector will also be slower, because a new set of candidates will be generated for each new sample.

This problem is solved by the `ClosenessThreshold` merge criterion. This criterion will make sure that no samples are selected too close to each other at each iteration, by defining a threshold. The highest ranking point is always selected, but then the criterion will go down the list until it encounters a sample that does not violate the threshold, thereby guaranteeing a minimum distance between the newly selected points and a good spread over the design space. While the quality of the final design will never be on the same level as when the samples are selected one by one, this method will avoid the clusters generated by the weighted average criterion. This is illustrated in Figure 4.5, which shows the same configuration as in Figure 4.4, but with the closeness threshold criterion instead of the weighted average criterion. Note that samples are now spread nicely over the design space.

The pipeline sample selector framework is used by several of the methods proposed in this thesis, including `mc-intersite-proj`, `mc-intersite-proj-th` and `delaunay`. Without the flexible pipeline framework, it would have been much

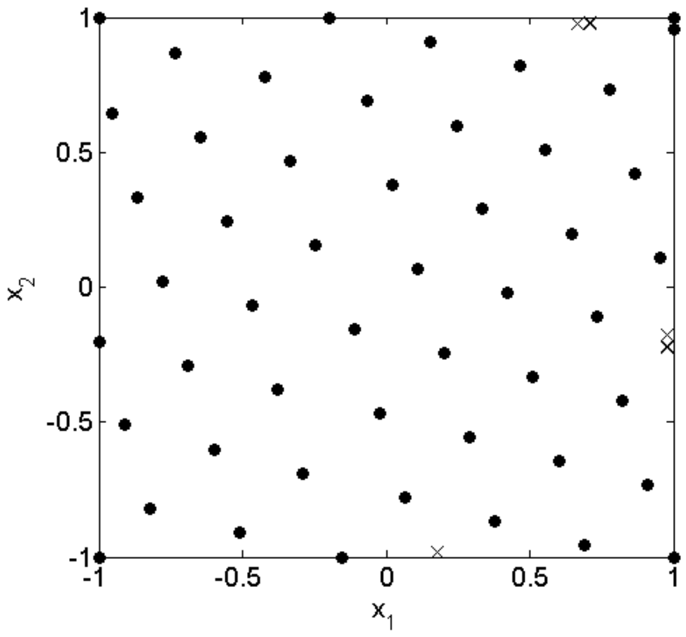


Figure 4.4: The weighted average merge criterion combined with the random candidate generator, while selecting 10 samples at once. The circles are the points from the initial design, the crosses are the 10 newly selected samples.

more difficult to design, implement and compare these specialized methods.

The second framework available in SUMO is the optimize criterion. A flow chart of this framework is shown in Figure 4.6. This method uses the same candidate rankers as the pipeline framework, but sees a candidate ranker as an objective function and uses an optimizer to optimize this function. Optionally, a candidate generator can be used to set the initial population of the optimizer. Several fallback candidate rankers can be defined as well; if the optimizer fails to find (one or more) new samples using the first ranker, the second ranker is used, and so on. Not every optimizer uses this feature, as some optimizers always return new samples.

Many different optimizers are available, and some were written specifically for a given sequential design method. General-purpose optimizers include a hill climber, a pattern search optimizer, a genetic algorithm, a particle swarm optimizer, and so on. But for complicated methods such as `optimizer-intersite` and `optimizer-projected`, specialized optimizers were necessary that respect the boundary constraints. However, implementing them was still considerably easier thanks to the optimizer framework that was in place. Without the flexibility of these frameworks, developing and comparing all these methods would have been much more difficult.

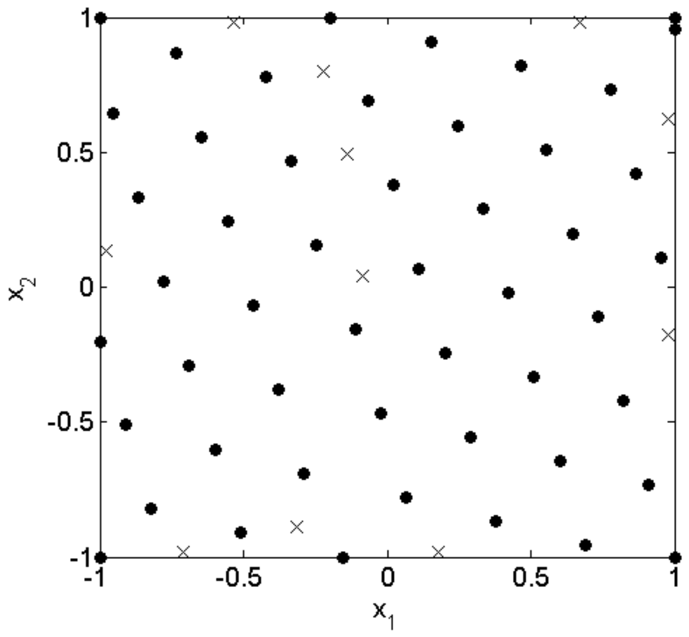


Figure 4.5: The closeness threshold merge criterion combined with the random candidate generator, while selecting 10 samples at once. The circles are the points from the initial design, the crosses are the 10 newly selected samples.

4.2 SED Toolbox

The Sequential Experimental Design (SED) Toolbox is a subset of the SUMO Toolbox that focuses on the sequential design features available in SUMO [16]. It packages them into an easy-to-use, standalone package that does not depend on external libraries or Java binaries. SED was designed to be integrated easily in the work-flow of any user, allowing through both command-line and XML configuration to quickly generate high quality designs on the fly. The key features of the SED Toolbox are:

- Includes several highly efficient space-filling sequential experimental design algorithms, which generate designs competitive with state-of-the-art one-shot experimental design techniques such as Latin hypercubes.
- Optimized for speed: several algorithms are included, ranging from extremely fast, even in high-dimensional problems, to slightly slower (yet still relatively fast).
- All methods support constraints of any kind (linear and nonlinear). These constraints can be specified with a Matlab function.

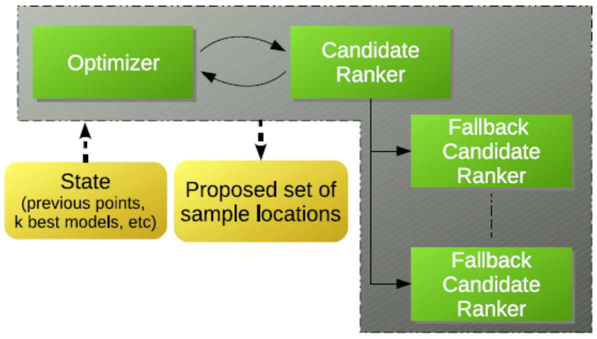

Optimizer
Candidate Ranker
Fallback Candidate Ranker
Fallback Candidate Ranker
State (previous points, k best models, etc)
Proposed set of sample locations

Figure 4.6: A flow chart of the `OptimizeCriterion`.

- Includes the powerful LOLA-Voronoi algorithm, which distributes design points sequentially according to the nonlinearity of the problem, following the assumption that nonlinear regions are more difficult to understand than linear ones.
- Support for rectangular input spaces: not every input is necessarily equally important, so weights can be given to each input and the algorithms will take this into account when selecting design points.
- Easy to use and configure, with examples amptly available in the documentation and in the distribution.
- Proper object-oriented design.

The SED Toolbox has two modi it can operate in. If the toolbox is configured through XML files, a simplified layout similar to the one used by the SUMO Toolbox is used. This modus allows for the most flexibility, and makes it easy to fine-tune the options of each method. The command-line based method, however, is easier to use, and can be learned in seconds by anyone familiar with Matlab. It uses Matlab structs to define the input and output dimension of the problem and other settings.

The SED Toolbox uses the same frameworks and code used by the sample selector part of the SUMO Toolbox, but wraps them in a smaller, easier to use package. Hence, it also uses the flexible implementation described in 4.1.3.2, and has all the advantages of this framework.

For documentation on how to download, install, configure and use the SED Toolbox, please refer to the website: `http://sumo.intec.ugent.be`.

4.2.1 Quick start guide

In this section, we briefly explain how to install, configure and use the SED Toolbox. This section can act as a manual for new users of the toolbox.

Before the toolbox can be used, you have to set it up for use, by browsing to the directory in which the toolbox was unpacked and running the startup command:

```
startup;
```

Now the toolbox is ready to be used. The SED Toolbox can be used in several ways, based on how much freedom you want in configuring and fine-tuning the parameters of the algorithms. We will now describe three ways the toolbox can be used, in order of complexity, based on your requirements. If you prefer to learn by example, you can check out the examples directory in the distribution, which contains several applications and example problems for the toolbox.

4.2.1.1 You want an ND design of X points

In order to quickly generate a good ND design in X points, you can use the following code:

```
startup; % configure the toolbox
config.inputs.nInputs = N; % set the number of inputs in the ...
    config struct
generator = SequentialDesign(config); % set up the sequential design
generator = generator.generateTotalPoints(X); % generate a total ...
    of X points
points = generator.getAllPoints(); % return the entire design

% optional:
generator.plot(); % plot the design
generator.getMetrics(); % get some metrics about the quality of ...
    the design
```

4.2.1.2 You want to use the more advanced features of the SED Toolbox

If you want to use some of the more advanced features of the SED Toolbox, such as input ranges and weights and constraints, you have two options. The first one is to use Matlab structs as in the previous example. The second one is to use simple XML files to configure the toolbox. Note that constraints will only work with XML configuration. You can open the `problem.xml` file in the SED directory to get an idea of how a problem configuration looks like. You can edit this file to suit your needs and use it to configure the toolbox using the following command:

```
% generate a sequential design for the problem defined in ...
    problem.xml:
generator = SequentialDesign('problem.xml');

% generate a sequential design using the specified method for the ...
    problem defined in problem.xml:
generator = SequentialDesign('problem.xml', ...
    'methods/mc-intersite-projected-threshold.xml');
```

If you instead prefer to use Matlab structs, you can use the following code to configure the toolbox:

```
config.inputs.nInputs = 2; % this is a 2D example
config.inputs.minima = [-1 -1]; % define the minimum of each input
config.inputs.maxima = [3 1]; % define the maximum of each input
config.inputs.weights = [2 1]; % the first input is twice as ...
    important as the second one
generator = SequentialDesign(config); % set up the sequential design
```

4.2.1.3 You want full control over all the method parameters

If you want full control over all the parameters of both the problem specification and the sequential design method, XML files are the only option. By editing the method XML files, you can tweak each method to your own preferences. Even though the options are documented, it might be difficult to understand their effect on the sampling process. Note that the default settings have been chosen based on extensive studies and comparisons, and are in most cases the best choice. If you have any questions or suggestions, please contact the authors at `Karel.Crombecq@ua.ac.be`.

In addition to the methods provided by the XML files packaged with the SED Toolbox, SED also contains a huge library of components (such as candidate generators, optimizers, metrics) from which the user can compose his own sequential design methods. This feature is undocumented and unsupported, but users are free to experiment with them.

4.2.2 Function reference

This section will contain a list of all the functionality available in the SED Toolbox.

seq = SequentialDesign(problemStruct)
Create a sequentual design object for the specified problem, as described in problemStruct. Uses the default algorithm (mc-intersite-projected-threshold) to generate the design.

seq = SequentialDesign('problem.xml')
Create a sequentual design object for the specified problem, as described in the problem.xml XML file. Uses the default algorithm (mc-intersite-projected-threshold) to generate the design.

seq = SequentialDesign(problemStruct, 'methods/method.xml')
Create a sequentual design object for the specified problem, as described in problemStruct. Uses the algorithm described in methods/method.xml to generate the design.

seq = SequentialDesign('problem.xml', 'methods/method.xml')
Create a sequentual design object for the specified problem, as described

in the problem.xml XML file. Uses the algorithm described in methods/method.xml to generate the design.

[seq, newPoints] = seq.generatePoints(10)
Use the sequential design algorithm to generate an additional 10 points on top of the already generated points. Will return the new points as the second return parameter.

[seq, newPoints] = seq.generateTotalPoints(10)
Use the sequential design algorithm to generate a total of 10 points. If, for example, 6 points were previously generated, 4 additional points will be selected to get the total up to 10. Will return the new points as the second return parameter.

seq.getInitialDesign()
The initial design is a set of samples which is generated in advance, to get the sequential design algorithm started. In the SED Toolbox, these are kept as small as possible (most methods need at least 2 points to get going, so the initial design will typically be 2 points). Note that the initial design might not respect the constraints. You can manually remove the initial points from the design, or you can request all the points excluding the initial design using the getAllPointsWithoutInitialDesign() function.

seq.getAllPoints()
Get all points generated thus far, including the initial design.

seq.getAllPointsWithoutInitialDesign()
Get all points generated thus far, excluding the initial design.

seq.plot()
Generate a plot of the design generated thus far. Will only work for 1-3D.

seq.getMetrics()
Will calculate some metrics about the quality of the design. Two metrics are calculated and plotted: the intersite distance (minimum distance between points) and the projected distance (minimum distance between points after being projected onto one of the axes). These can be used as a basis of comparison between designs.

metrics = seq.getMetrics()
Will return the same metrics as described above in a struct. Will not plot or print any data.

metrics = seq.getMetrics(points)
Calculate the same metrics as above, but then for the points provided as an argument instead of the design generated by the object. This function can be used to compare a design from another source against the design generated by the SED Toolbox.

seq = seq.updatePoints(newPoints, newValues)

When using LOLA-Voronoi, you need to provide the outputs produced through simulation after every call of generatePoints. This is required because LOLA-Voronoi uses the outputs to determine the optimal distribution of points. When using the other methods, you do not need to call updatePoints.

seq = seq.plotLive(true)

This will enable live plotting of sample generation for 2D designs. If this is enabled, after each point that is generated, a plot will be built that shows the current design. This nicely demonstrates how points are selected and distributed over the design space.

seq.save('file.txt')

Store the entire design in a text file called file.txt. Can later be loaded again by calling data = load('file.txt').

4.2.3 Rules of thumb

The default sequential design method for the SED Toolbox is the Monte Carlo method `mc-intersite-proj-th`, found in mc-intersite-projected-threshold.xml. This method is very fast and can be applied to highly dimensional problems and for large designs. It also works well with constraints and input weights. However, there are some cases in which one of the other methods might be a better choice. This section contains some rules of thumb for picking the right method for the right job.

4.2.3.1 Constraints

the default method `mc-intersite-proj` can run into problems when you are using very strict constraints. Because the Monte Carlo points are filtered by the projected distance threshold, it might be possible that no candidates remain that satisfy the constraints. In that case, `mc-intersite-proj` (available as mc-intersite-projected.xml) can be a good alternative. It produces slightly worse designs but is much more robust in terms of constraints. Additionally, `mc-intersite-proj-th` and all other methods available in the SED Toolbox besides `mc-intersite-proj` need the corner points $[-1,\dots,-1]$ and $[1,\dots,1]$ to start, and if they violate the constraints they will still be selected because the other methods need them to even start. You can later request the design without these corner points using the getAllPointsWithoutInitialDesign() function, so this might not be an issue, but keep it in mind.

4.2.3.2 Quality vs speed

The slowest method available in SED is `optimizer-intersite`, but this method also generates the best designs (slightly better than `mc-intersite-proj-th`). If you have the time, consider using this method instead. It also supports constraints, but might also run into problems with very tight constraints.

If time is of no concern, you can also consider increasing some of the method parameters to further improve the design. For `mc-intersite-proj-th`, the candidatesPerSample option can be increased to improve the quality at the cost of speed. In lower dimensions, this will not return much better results as shown in Section 2.5.4.3, but it might help in higher dimensions. For `optimizer-intersite`, as shown in Section 2.5.5.2, the $n_{\text{hypercubes}}$ option has the biggest influence on the quality of the design. This option is called nPop in the XML file, and for 4D designs and higher, increasing it can improve the deisgn.

4.2.3.3 Dimensionality

The Monte Carlo methods scale very well with the number of dimensions and points and should work for high-dimensional problems. However, the optimizer methods suffer more from the curse of dimensionality. optimizer-intersite should work up to 10D, but will run into memory problems for higher dimensions.

CHAPTER 5

Conclusions

This kingdom shall fall... And from the ashes shall arise a new order, that will shake the very foundations of the world.
— Arthas

In this thesis, we proposed several new input- and output-based sequential design strategies in the context of adaptive surrogate modelling of deterministic, black box computer experiments.

In Chapter 2, several state-of-the-art methods were proposed based on Monte Carlo and local optimization techniques, and these new methods were compared against popular and proven techniques from different research domains. It was shown that the new methods offer many advantages over the classical methods. Existing one-shot experimental designs can, given enough time, generate designs of equal or better quality than the new methods, but do not have the advantages of being a sequential method, and are prone to problems such as over- and undersampling.

In Chapter 3, we proposed a new output-based method called LOLA-Voronoi, which was designed to distribute samples according to the local nonlinearity of the system that is to be modelled, following the assumption that nonlinear regions are more difficult to approximate than linear regions. It was shown that LOLA-Voronoi is an extremely efficient and stable sequential design method, producing good results in a large number of different test cases. LOLA-Voronoi does not make any assumptions about the model type being used, and is therefore ideal for a heterogeneous modelling environment, where multiple models are considered.

All these methods are freely available in both the SUMO and SED Toolboxes. SUMO integrates these sequential design methods in a fully featured adaptive surrogate modelling environment, with many different model types, hyperparameter optimization techniques, simulator configurations and so on. SED, on the other hand, offers these methods in a tight, easy to install and use package that can be integrated in the modelling pipeline of the user. Both toolboxes are free for academic use and open source. Installation instructions and documentation can be found on `http://sumo.intec.ugent.be`.

The methods presented in this thesis are competitive and mature enough for use by engineers. However, there are still some open issues that can be adressed, in order to build upon the foundations laid by this work, or to improve the existing methods. Firstly, LOLA-Voronoi now only works with the Voronoi-based space-filling method, even though it is not the best space-filling method, as shown in Chapter 2. Voronoi was designed to produce a ranking of previously evaluated points. This works nicely with the LOLA component, which does the same. The other space-filling methods produce a ranking of candidate points, and therefore can't be used directly with LOLA. It might be worth investigating how these methods can be adapted to work with each other, so that Voronoi can be replaced by a more competitive space-filling method.

Secondly, it was briefly touched in Section 2.5 that methods that produce an optimal design at one iteration might get stuck in a local optimum on the subsequent iterations. It should be interesting to investigate where that behaviour comes from, and how it affects the quality of the resulting design. Most methods in this thesis avoid this issue by using some random element in the algorithm, to avoid getting stuck in the same optimum every time. Looking for other ways to avoid this pitfall could lead to other good space-filling methods.

Thirdly, most methods proposed in this thesis support (linear and nonlinear) constraints and rectangular input spaces (where each dimension is not equally important, but some are considered more important than others, and hence should be sampled more densely). However, these features were not thoroughly tested, and no problems are presented in this thesis that use these features. It should be interesting to perform a study with such problems.

Ultimately, this thesis takes several important steps towards highly efficient sequential design strategies that can be used in a variety of circumstances. By designing these methods and analyzing them on a wide range of problems, many new insights were obtained in how good sequential designs can be generated. Hopefully, these methods will aid engineers in more efficiently analyzing, optimizing and understanding the problems they encounter every day in a world where computer simulation is everywhere, from the design of small microchips in cell phones to entire airplanes.

APPENDIX A

Publications

A.1 Journal papers

- **The SED Toolbox: a Sequential Experimental Design Toolbox for Regression**
  K. Crombecq and T. Dhaene
  Journal of Machine Learning Research, submitted

- **A Novel Hybrid Sequential Design Strategy for Global Surrogate Modelling of Computer Experiments**
  K. Crombecq, D. Gorissen, D. Deschrijver and T. Dhaene
  SIAM Journal of Scientific Computing, accepted, 2011

- **Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling**
  K. Crombecq, E. Laermans and T. Dhaene
  European Journal of Operational Research, Vol. 214, No. 3, pp. 683-696, 2011

- **Adaptive Sampling Algorithm for Macromodeling of Parameterized S-Parameter Responses**
  D. Deschrijver, K. Crombecq, H. M. Nguyen and T. Dhaene
  IEEE Transactions on Microwave Theory and Techniques, Vol. 59, No. 1, pp. 39-45, 2011

- **Surrogate based sensitivity analysis of process equipment**
  D. W. Stephens, D. Gorissen, K. Crombecq and T. Dhaene
  Journal of Applied Mathematical Modelling, Vol. 35, No. 4., pp. 1676-1687, 2011

- **A Surrogate Modeling and Adaptive Sampling Toolbox for Computer Based Design**

D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene and P. Demeester
Journal of Machine Learning Research, Vol. 11, pp. 2051-2055, 2010

- **Elastic characterization of membranes with a complex shape using point indentation measurements and inverse modelling**
  J. Aernouts, I. Couckuyt, K. Crombecq and J.J.J. Dirckx
  International Journal of Engineering Science, 48, pp. 599-611, 2010

- **Sequential Modeling of a Low Noise Amplifier with Neural Networks and Active Learning**
  D. Gorissen, L. De Tommasi, K. Crombecq and T. Dhaene
  Neural Computation & Applications, Vol. 18, Nr. 5, pp. 485-494, 2009

A.2 Conference papers

- **Efficient parameter estimation for discrete tomography using adaptive modeling**
  W. van Aarle, K. Crombecq, I. Couckuyt, K. J. Batenburg, J. Sijbers
  Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pp. 229-232, 2011

- **Generating Sequential Space-filling Designs Using Genetic Algorithms and Monte Carlo Methods**
  K. Crombecq and T. Dhaene
  Simulated Evolution And Learning (SEAL-2010), Kanpur, India, pp. 80-84, 2010

- **Automated Response Surface Model Generation with Sequential Design**
  I. Couckuyt, K. Crombecq, D. Gorissen and T. Dhaene
  Soft Computing Technology in Civil, Structural and Environmental Engineering, 2009

- **Pareto-based multi-output model type selection**
  D. Gorissen, I. Couckuyt, K. Crombecq and T. Dhaene
  Proceedings of the 4th International Conference on Hybrid Artificial Intelligence (HAIS 2009), Salamanca, Spain Springer - Lecture Notes in Artificial Intelligence, Vol. LNCS 5572, pp. 442-449, 2009

- **Space-filling Sequential Design Strategies for Adaptive Surrogate Modelling**
  K. Crombecq, I. Couckuyt, D. Gorissen and T. Dhaene
  Soft Computing Technology in Civil, Structural and Environmental Engineering, 2009

- **A Novel Hybrid Active Learning Strategy for Nonlinear Regression**
  K. Crombecq, I. Couckuyt, E. Laermans and T. Dhaene
  The 18th Annual Belgian-Dutch Conference on Machine Learning (Benelearn 09), pp. 109-110, 2009

- **A Novel Sequential Design Strategy for Global Surrogate Modeling**
  K. Crombecq, D. Gorissen, L. De Tommasi and T. Dhaene
  Proceedings of the 41th Conference on Winter Simulation, Austin, Texas, December 2009, pp. 731-742, 2009

- **A comparison of sequential design methods for RF circuit block modeling**
  K. Crombecq, L. De Tommasi, D. Gorissen and T. Dhaene
  Proceedings of the 40th Conference on Winter Simulation, pp. 2942-2942, Miami, Florida, 2008

- **Adaptive Distributed Metamodeling**
  D. Gorissen, K. Crombecq, W. Hendrickx and T. Dhaene
  7th International Meeting on High Performance Computing for Computational Science (VECPAR 2006), Rio de Janeiro (Brazil), Springer - Lecture Notes in Computer Science, Vol. LNCS 4395, pp.579-588, 2007

- **Adaptive Global Surrogate Modeling**
  D. Gorissen, W. Hendrickx, K. Crombecq, W. van Aarle and T. Dhaene
  SIAM Conference on Computational Science and Engineering (CSE07), Costa Mesa (CA), pp. 160, February 2007. Poster Session

- **Integrating Gridcomputing and Metamodeling**
  D. Gorissen, W. Hendrickx, K. Crombecq and T. Dhaene
  6th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2006), Singapore (Singapore), pp. 185-192, 2006

A.3 Book chapters

- **Automatic Approximation of Expensive Functions with Active Learning**
  D. Gorissen, K. Crombecq, I. Couckuyt and T. Dhaene
  Foundations of Computational Intelligence Volume 1: Learning and Approximation: Theoretical Foundations and Applications, Part I: Function ApproximationŤ, Edited by A-E. Hassanien, A. Abraham, A.V. Vasilakos, and W. Pedrycz, ISBN: 978-3-642-01081-1, pp 35-62, 2009

Bibliography

[1] Qhull. `http://www.qhull.org`.

[2] D. H. Ackley. A connectionist machine for genetic hillclimbing. *Kluwer International Series In Engineering And Computer Science*, 28, 1987.

[3] P. Audze and V. Eglajs. New approach for planning out of experiments. *Problems of Dynamics and Strengths*, 35:104–107, 1977.

[4] F. Aurenhammer. Voronoi diagrams–a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[5] L. Balewski and M. Mrozowski. Creating neural models using an adaptive algorithm for optimal size of neural network and training set. *15th International Conference on Microwaves, Radar and Wireless Communications*, 2: 543–546, 2004.

[6] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathemathical Software*, 22(4):469–483, 1996.

[7] R. R. Barton. Simulation metamodels. In D. J. Medeiros and Edward F. Watson, editors, *Proceedings of the 30th Winter Simulation Conference*, pages 167–174, 1998.

[8] I. Batmaz and S. Tunali. Small response surface designs for metamodel estimation. *European Journal of Operational Research*, 145(2):455–470, 2003.

[9] W. C. M. Van Beers. Kriging metamodeling in discrete-event simulation: an overview. In N. Steiger and M. E. Kuhl, editors, *Proceedings of the 37th Winter Simulation Conference*, pages 202–208, 2005.

[10] G. E. P. Box, J. S. Hunter, and W. G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley-Interscience, 2005.

[11] D. Busby, C. L. Farmer, and A. Iske. Hierarchical nonlinear approximation for experimental design and statistical data fitting. *SIAM Journal on Scientific Computing*, 29(1):49–69, 2007.

[12] H. Cohn and A. Kumar. Universally optimal distribution of points on spheres. *Journal of the American Mathematical Society*, 20(1):99–148, 2007.

[13] I. Couckuyt, K. Crombecq, D. Gorissen, and T. Dhaene. Automated response surface model generation with sequential design. In *First International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering (CSC), Funchal, Portugal*, 2009.

[14] I. Couckuyt, D. Gorissen, H. Rouhani, E. Laermans, and T. Dhaene. Evolutionary regression modeling with active learning: An application to rainfall runoff modeling. In *Proceedings of the International Conference on Adaptive and Natural Computing Algorithms*, pages 548–558, 2009.

[15] H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry*. Springer, 1994.

[16] K. Crombecq and T. Dhaene. The sed toolbox: a sequential experimental design toolbox for regression. *Journal of Machine Learning Research*, 2011, submitted.

[17] K. Crombecq, I. Couckuyt, D. Gorissen, and T. Dhaene. Space-filling sequential design strategies for adaptive surrogate modelling. In *The First International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering, 20 pages*, 2009.

[18] K. Crombecq, D. Gorissen, L. De Tommasi, and T. Dhaene. A novel sequential design strategy for global surrogate modeling. In *Proceedings of the 41st Winter Simulation Conference*, pages 731–742, 2009.

[19] K. Crombecq, D. Gorissen, D. Deschrijver, and T. Dhaene. A novel hybrid sequential design strategy for global surrogate modelling of computer experiments. *SIAM Journal of Scientific Computing*, 33(4), 2011.

[20] K. Crombecq, E. Laermans, and T. Dhaene. Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. *European Journal of Operational Research*, 214(3):683–696, 2011.

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

[22] D. Deschrijver, T. Dhaene, and J. Broeckhove. Adaptive model based parameter estimation, based on sparse data and frequency derivatives. *International Conference on Computational Science (LNCS 3037)*, pages 443–450, 2004.

[23] T. Dhaene, J. Ureel, N. Faché, and D. De Zutter. Adaptive frequency sampling algorithm for fast and accurate s-parameter modeling of general planar structures. In *IEEE International Microwave Symposium*, volume 3, pages 1427–1430, 1995.

[24] H. Eres, G. Pound, Z. Jiao, J. Wason, F. Xu, A. Keane, and Simon Cox. Implementation of a grid-enabled problem solving environment in matlab. *International Conference on Computational Science (LNCS 2660)*, pages 420–429, 2003.

[25] K. T. Fang. Experimental design by uniform distribution. *Acta Mathematice Applicatae Sinica*, 3:363–372, 1980.

[26] K. T. Fang and D. K. J. Lin. Uniform experimental designs and their applications in industry. *Handbook of Statistics*, 22:131–170, 2003.

[27] K. T. Fang, C. X. Ma, and P. Winker. Centered l2-discrepancy of random sampling and latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, 71:275–296, 2002.

[28] A. Farhang-Mehr and S. Azarm. Bayesian meta-modelling of engineering design simulations: a sequential approach with adaptation to irregularities in the response behaviour. *International Journal for Numerical Methods in Engineering*, 62(15):2104–2126, 2005.

[29] A. Forrester, A. Sobester, and A. Keane. *Engineering Design Via Surrogate Modelling: A Practical Guide*. Wiley, 2008.

[30] D. A. Freedman. *Statistical Models: Theory and Practice*. Cambridge University Press, 2005.

[31] M. Fu. Stochastic gradient estimation, pre-print version of chapter 19. In S. G. Henderson and B. L. Nelson, editors, *Handbook on Operations Research and Management Science: Simulation*. Elsevier, 2005.

[32] M. C. Fu and S. D. Hill. Optimization of discrete event systems via simultaneous perturbation stochastic approximation. *IEEE Transactions*, 29(3): 233–243, 1997.

[33] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Y. Loh. Boat - optimistic decision tree construction. *SIGMOD Record*, 28(2):169–180, 1999.

[34] T. Goel, R. Haftka, W. Shyy, and N. Queipo. Ensemble of surrogates. *Structural and Multidisciplinary Optimization*, 33:199–216, 2007.

[35] D. Gorissen. *Grid-enabled Adaptive Surrogate Modeling for Computer Aided Engineering*. PhD thesis, Ghent University, 2010.

[36] D. Gorissen, K. Crombecq, W. Hendrickx, and T. Dhaene. Adaptive distributed metamodeling. *High Performance Computing for Computational Science - VECPAR 2006*, 4395:579–588, 2007.

[37] D. Gorissen, K. Crombecq, I. Couckuyt, and T. Dhaene. Automatic approximation of expensive functions with active learning. In *Foundation on Computational Intelligence, Learning and Approximation*. Springer Verlag, 2008.

[38] D. Gorissen, L. De Tommasi, J. Croon, and T. Dhaene. Automatic model type selection with heterogeneous evolution: An application to rf circuit block modeling. In *IEEE World Congress on Computational Intelligence (WCCI 2008)*, pages 989–996, 2008.

[39] D. Gorissen, L. De Tommasi, W. Hendrickx, J. Croon, and T. Dhaene. Rf circuit block modeling via kriging surrogates. *17th International Conference on Microwaves, Radar and Wireless Communications*, 2008.

[40] D. Gorissen, L. De Tommasi, K. Crombecq, and T. Dhaene. Sequential modeling of a low noise amplifier with neural networks and active learning. *Neural Computation & Applications*, 18(5):485–494, 2009.

[41] D. Gorissen, F. De Turck, and T. Dhaene. Evolutionary model type selection for global surrogate modeling. *Journal of Machine Learning Research*, 10(1): 2039–2078, 2009.

[42] D. Gorissen, I. Couckuyt, E. Laermans, and T. Dhaene. Multiobjective surrogate modeling, dealing with the 5-percent problem. *Engineering with Computers*, 26(1):81–98, 2010.

[43] D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene, and P. Demeester. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11:2051–2055, 2010.

[44] R. Gramacy and H. K. H. Lee. Adaptive design of supercomputer experiments. Technical report, Dept of Applied Math & Statistics, University of California, 2006.

[45] A. Grosso, A. Jamali, and M. Locatelli. Finding maximin latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, 197(2):541–547, 2009.

[46] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Wann Jensen. Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Transactions on Graphics*, 27(3):1–10, 2008.

[47] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[48] W. Hendrickx and T. Dhaene. Sequential design and rational metamodelling. In N. Steiger and M. E. Kuhl, editors, *Proceedings of the 37th Winter Simulation Conference*, pages 290–298, 2005.

[49] F. J. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of Computation*, 67:299–322, 1998.

[50] B. Husslage. *Maximin Designs for Computer Experiments*. PhD thesis, Tilburg University, Center of Economic Research, 2006.

[51] Agilent Technologies Inc. *ADS Momentum Software*. 2009.

[52] A. A. Jamshidi and M. J. Kirby. Towards a black box algorithm for nonlinear function approximation over high-dimensional domains. *SIAM Journal on Scientific Computing*, 29(3):941–963, 2007.

[53] R. Jin, W. Chen, and A. Sudjianto. On sequential sampling for global metamodeling in engineering design. In *Proceedings of DETCŠ02 ASME 2002 Design Engineering Technical Conferences And Computers and Information in Engineering Conference*, pages 539–548, 2002.

[54] R. Jin, W. Chen, and A. Sudjianto. An effcient algorithm for constructing optimal design of computer experiments. *Journal of Statistical Planning and Inference*, 134(1):268–287, 2005.

[55] M.E. Johnson, L.M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, 26:131–148, 1990.

[56] V. Roshan Joseph and Y. Hung. Orthogonal-maximin latin hypercube designs. *Statistica Sinica*, 18:171–186, 2008.

[57] A. J. Keane and A. P. Bright. Passive vibration control via unusual geometries: experiments on model aerospace structures. *Journal of Sound and Vibration*, 190(4):713–719, 1996.

[58] J. P. C. Kleijnen and W. C. M. van Beer. Application-driven sequential designs for simulation experiments: Kriging metamodelling. *Journal of the Operational Research Society*, 55(8):876–883, 2004.

[59] J. Knowles and H. Nakayama. Meta-modeling in multiobjective optimization. In *Multiobjective Optimization: Interactive and Evolutionary Approaches*, pages 245–284. Springer-Verlag, 2008.

[60] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

[61] A. Lamecki, P. Kozakowski, and M. Mrozowski. Cad-model construction based on adaptive radial basis functions interpolation technique. *15th International Conference on Microwaves, Radar and Wireless Communications*, 2:799–802, 2004.

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

[63] R. Lehmensiek and P. Meyer. Creating accurate multivariate rational interpolation models of microwave circuits by using efficient adaptive sampling to minimize the number of computational electromagnetic analyses. *IEEE Transactions Microwave Theory and Technology*, 49(8):1419–1430, 2001.

[64] R. Lehmensiek, P. Meyer, and 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.

[65] X. Rong Li and Zhanlue Zhao. Evaluation of estimation algorithms part i: incomprehensive measures of performance. *IEEE Transactions on Aerospace and Electronic Systems*, 42(4):1340–1358, 2006.

[66] D. Lim, Y-S. Ong, Y. Jin, and B. Sendhoff. A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1288–1295, 2007.

[67] Y. Lin. *An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design.* PhD thesis, Georgia Institute of Technology, 2004.

[68] S. N. Lophaven, H. B. Nielsen, and J. Søndergaard. Dace: A matlab kriging toolbox. Technical report, Technical University of Denmark, 2002.

[69] D. J. C. MacKay. Bayesian model comparison and backprop nets. In *Advances in Neural Information Processing Systems 4*, pages 839–846. Morgan Kaufmann, 1992.

[70] J. D. Martin and T. W. Simpson. Use of kriging models to approximate deterministic computer models. *AIAA Journal*, 43(4):853–863, 2005.

[71] D. C. Montgomery. *Design and Analysis of Experiments.* 2001.

[72] M. D. Morris and T. J. Mitchell. Exploratory designs for computer experiments. *Journal of Statistical Planning and Inference*, 43:381–402, 1995.

[73] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods.* Society for Industrial and Applied Mathematics, 1992.

[74] M. Norgaard, O. Ravn, L. Hansen, and N. Poulsen. The nnsysid toolbox - a matlab toolbox for system identification with neural networks. *Computer-Aided Control System Design*, pages 374–379, 1996.

[75] I. G. Osio and C. H. Amon. An engineering design methodology with multistage bayesian surrogates and optimal sampling. *Research in Engineering Design*, 8(4):189–206, 1996.

[76] A. B. Owen. Orthogonal arrays for computer experiments, integration and visualization. *Statistica Sinica*, 2:439–452, 1992.

[77] C.G. Panayiotou, C.G. Cassandras, and Wei-Bo Gong. Model abstraction for discrete event systems using neural networks and sensitivity information. pages 335–341, 2000.

[78] F. J. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. *Knowledge Discovery and Data Mining*, pages 23–32, 1999.

[79] Peter Z. G. Qian. Nested latin hypercube designs. *Biometrika*, 96(4):957–970.

[80] C. E. Rasmussen. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[81] R. G. Regis. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. *Computers & Operations Research*, 38(5):837–853, 2011.

[82] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.

[83] E. B. Saff and A. B. J. Kuijlaars. Distributing many points on a sphere. *Mathematical Intelligencer*, 19(1):5–11, 1997.

[84] E. Sanchez, S. Pintos, and N.V. Queipo. Toward an optimal ensemble of kernel-based approximations with engineering applications. In *Proceedings of the International Joint Conference on Neural Networks*, pages 2152–2158, 2006.

[85] M. J. Sasena. *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, 2002.

[86] J. Shekel. Test functions for multimodal search techniques. In *Fifth Annual Princeton Conference on Information Science and Systems*, pages 354–359, 1971.

[87] T. W. Simpson, D. K. J. Lin, and W. Chen. Sampling strategies for computer experiments: Design and analysis. *International Journal of Reliability and Applications*, 2(3):209–240, 2001.

[88] T. W. Simpson, J. Peplinski, P. N. Koch, and J. K. Allen. Metamodels for computer-based engineering design: Survey and recommendations. *Engineering with Computers*, 17(2):129–150, 2001.

[89] M. Sugiyama. Active learning in approximately linear regression based on conditional expectation of generalization error. *Journal of Machine Learning Research*, 7:141–166, 2006.

[90] B. Tang. Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, 88(424):1392–1397, 1993.

[91] C. J. Turner, R. H. Crawford, and M. I. Campbell. Multidimensional sequential sampling for nurbs-based metamodel development. *Engineering with Computers*, 23(3):155–174, 2007.

[92] E. R. van Dam, B. Husslage, D. den Hertog, and H. Melissen. Maximin latin hypercube design in two dimensions. *Operations Research*, 55(1):158–169, 2007.

[93] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1999.

[94] F. A. C. Viana, G. Venter, and V. Balabanov. An algorithm for fast optimal latin hypercube design of experiments. *International Journal for Numerical Methods in Engineering*, 82(2):135–156, 2009.

[95] J. R. Wieland and B. W. Schmeiser. Stochastic gradient estimation using a single design point. In *Proceedings of the 38th Winter Simulation Conference*, pages 390–397, 2006.

[96] F. Xiong, Y. Xiong, W. Chen, and S. Yang. Optimizing latin hypercube design for sequential sampling of computer experiments. *Engineering Optimization*, 41(8):793–810, 2009.

[97] K. Q. Ye, W. Li, and A. Sidjianto. Algorithmic construction of optimal symmetric latin hypercube designs. *Journal of Statistical Planning and Inference*, 90 (1):145–159, 2000.

[98] J. Yin, S. H. Ng, and K. M. Ng. A study on the effects of parameter estimation on kriging model's prediction error in stochastic simulations. In *Proceedings of the 41st Winter Simulation Conference*, pages 674–685, 2009.

[99] H. Zhao and D. Knight. Data driven design optimization methodology development and application. *International Conference on Computational Science (LNCS 3038)*, pages 748–755, 2004.