People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University Blida 1

FACULTY OF SCIENCES

DEPARTMENT OF MATHEMATICS

THESIS

In order to obtain a Master's degree

Specialty: Mathematics

Option: Stochastic Modeling and Statistics

TITLE

A New Approach For Generating Designs Of Computer Experiments From Area-Interaction Point Processes

---

Presented by: Boudali Aimen and Lafri Amrane Abderrahmane

Defended before the jury composed of:

Mr. BOUDJMEAA R.	Assistant Professor A	President
Mr. FRIHI R.	Assistant Professor B	Examiner
Mr. ELMOSAOUI H.	Assistant Professor A	Supervisor
Mr. AIT AMEUR A.	PhD student	Co-supervisor

2022/2023

Acknowledgements

We would like to express our gratitude, first and foremost, to God Almighty for granting us the courage, patience, and determination to complete this humble work.

We would like to thank Mr. Elmossaoui Hichem, Assistant Professor at Blida1 University, our thesis advisor, who has guided and advised us wisely throughout this work.

We would like to thank Mr. AIT AMEUR A., our co-supervisor, for his constant support, valuable advice, and expertise throughout this experience.

We extend our deep gratitude and sincere thanks to Mr. O. Tami, the Head of the Mathematics Department at Blida1 University, for all the assistance provided to us.

We take this opportunity to express our profound gratitude and sincere thanks to all the teachers who have contributed to our education.

We would also like to thank the staff members of the Mathematics Department, in particular, Mr. H. Hadj Allah, Ms. N. Djenas, and Ms. S. Takarli.

Lastly, we would like to express our utmost gratitude to the members of the jury for the honor they bestow upon us by agreeing to evaluate this work.

Dedication

To my dearest mother, whose unwavering love and support have guided me every step of the way. To my dear departed father, though you are no longer with us, your memory and legacy live on in our hearts. To my beloved brothers and sisters, who have shared both the joys and challenges of life with me. To my cherished friends, who have stood by my side through thick and thin. Your friendship has been a source of joy, laughter, and comfort in my life. I dedicate my achievements to all of you, for your unconditional love and unwavering support. Forever grateful and forever in my heart.

Aimen Boudali

This thesis is dedicated to my family and friends who have been my rock, my guiding light, and my unwavering support. Your love, belief in me, and encouragement have been instrumental in my journey. Thank you for always being there for me.

Lafri Amrane Abderrahmane

Abstract

Many engineers and technicians rely on experimental designs to improve their products or production processes based on experience. However, traditional strategies for conducting experiments often prove to be expensive, inefficient, and yield limited exploitable results. To address these challenges, the planning of experiments has become essential.

Experimental designs offer a structured approach to conducting tests in scientific research and industrial studies. They find applications in various disciplines and industries when investigating the relationship between a quantity of interest (y) and controllable variables (x_i). The objective is to establish mathematical models that relate these quantities of interest to the variables.

This thesis introduces new digital experimental designs based on the theory of stochastic processes, specifically area interaction point processes, also known as object processes. These designs leverage both the distribution of points within the experimental region and specific characteristics associated with those points. The designs are obtained using a Monte Carlo Markov chain method (MCMC), and a thorough investigation of the Markov chain's convergence has been conducted. Furthermore, a comparative analysis between our approach and other existing computer designs has been performed.

Keywords: Experimental Designs, Numerical Experimental Designs, Point Processes, Area-Interaction Point Processes, Voronoi Tessellation, Markov Chain Monte Carlo (MCMC), Metropolis-Hastings Algorithm.

Contents

List of Figures

List of Tables

Introduction

In recent years, numerical simulation has been used to model increasingly complex phenomena. Such problems, often of very high dimensions, require sophisticated simulation codes that are computationally expensive. The currently favored approach is to define a reduced number of simulations organized according to an experiment computer design. Therefore, it is important to have methods that optimize the selection of these simulations using the methodology of experimental design. This methodology is useful for anyone undertaking scientific research or industrial studies. The use of experimental designs for the empirical study of a response distribution presents specific challenges for statisticians and researchers. While they have limited information about this distribution, they generally have access to only a small sample of observations compared to the number of parameters in the models they consider for their analyses. Before any observation of the response, they must specify not only which models to use but also how to organize the experiments. Indeed, the quality of the statistical analysis depends heavily on the experimental design used to observe the response. Additionally, combinatorial analysis is generally necessary to construct the proposed experimental designs.

Authors such as Fisher [1], Kiefer [2], Box [3], and others have studied experimental designs. Classical designs tend to position points on the boundaries of the domain to account for random variation and provide a more reliable trend in the presence of measurement errors. Most criteria for these designs use experimental error that exists in the context of real experiments, whereas, in computer experiment designs, the error is associated with the model rather than the experimentation itself. Therefore, repeating classical experiments is of great interest for evaluating experimental error, while in the case of computer experiment designs, repeating an experiment under the same conditions is meaningless as it would yield the same response.

The diversity of experimental designs proposed in the literature arises from the fact that there is no perfect design where all optimality criteria are simultaneously satisfied. Each design has advantages for some optimality criteria and disadvantages for others. Therefore, it is necessary to find a compromise based on the specific requirements of each study.

The main objective of our work is to propose a new computer experiment design based on the theory of stochastic processes, particularly area-interaction point processes. By using this type of process instead of simple point processes, we can introduce not only geometric knowledge but also priory knowledge about the n experimental points that constitute the proposed computer experimental design. Thus, the experiments in these designs should best cover the experimental domain in order to obtain information, especially to detect possible irregularities. Therefore, we seek a design in which the points are uniformly distributed within the unit hypercube. The first who proposed a computer experimental design using point processes is Franco in 2008 [4] following its work El Moussaoui et *al.* in 2020 [5, 6, 7] proposed a numerical experimental design based on the utilization of marked Strauss processes [8] and recently in 2023 they proposed a new numerical experimental design method that uses the continuum random cluster point process [9]. To generate such designs, we will use simulation techniques such as Markov Chain Monte Carlo (MCMC), by including Voronoi tessellations [10] inside the Metropolis-Hastings algorithm [11, 12].

The thesis is composed of four chapters organized as follows:

. Chapter One introduces the theory of experimental design methodology and provides an overview of commonly used experimental designs in digital experimentation.

. Chapter Two introduces the theory of point processes. The chapter begins by presenting point processes followed by Markov processes. The simulation of these processes is described in the form of a Markov chain, with the basic concepts and convergence conditions presented. Subsequently, the main algorithms for simulating Markov chain Monte Carlo will be described.

. Chapter 3 provides a comprehensive overview of various techniques and algorithms related to point processes, spatial interactions, simulation methods, experimental design construction, and convergence analysis. These concepts play a crucial role in understanding and applying point processes. By exploring optimality criteria, intrinsic properties, and conducting a comparison between space-filling techniques.

. Chapter 4 provides valuable insights into the assessment and selection of experimental designs. These concepts aid researchers and practitioners in choosing the most appropriate design for their specific objectives and constraints, leading to efficient and reliable experiments.

Finally, a conclusion concludes this work, providing some perspectives for future research.

In the appendix, we present the programs developed in the PYTHON software used to perform the numerical illustrations provided in the third and fourth chapters.

Chapter 1

Generalities of Experimental Designs

In this chapter, the various hypotheses involved in the use of the experimental design method are synthesized and summarized. This method is useful for any experimenter undertaking scientific research or industrial studies. It is applicable to all disciplines as long as one is looking for the relationship that exists between a quantity of interest y and variables x_i that can modify the values of the former. To do this, it is necessary to follow mathematical rules and adopt a rigorous approach.

1.1. History

The methodology of experimental design has a rich historical background, incorporating both ancient and modern influences. As far back as the Middle Ages, Nicolas Oresme (1325-1382) acknowledged the importance of this approach in his writings [13]. Notably, Francis Bacon (1561-1626), an influential figure inspiring Descartes and Leibniz, stands as one of the precursors of the experimental method [14]. However, it was Sir Ronald Fisher who laid the foundation for contemporary experimental design. Fisher's work dates back to 1919 when he conducted agricultural research in a northern London research center, aiming to enhance agricultural yields by experimenting with various factors such as fertilizers, plant varieties, and soil types [1]. Faced with practical constraints that hindered conducting all possible experiments, Fisher proposed rigorous statistical models, including Latin squares, to derive experimental configurations.

Following in Fisher's footsteps, a group of statisticians, including Yates, Youden, Cochran,

Plackett, Burman, and others, have played significant roles in advancing and advocating for the application of experimental planning techniques beyond agronomy. Notably, during the 1950s, Box and his collaborators expanded on the groundwork laid by Yates, developing specific methodologies for constructing fractional factorial designs at two levels [3]. However, it was the pioneering work of Taguchi and Masuyama that truly revolutionized the use of experimental designs, as they introduced tables for constructing orthogonal experimental designs tailored to address the majority of industrial problems [15]. These influential tables were published in 1959 and 1961.

In recent times, a multitude of researchers have dedicated themselves to advancing this specific field of statistics through diverse avenues. Their efforts encompass a range of developments, including adapting experimental designs to address mixture problems [16], incorporating block effects [17], employing nonlinear models [18], exploring models that incorporate neighborhood effects, and devising experimental designs tailored for computer experiments, among other innovative approaches.

1.2. Interest of the experimental design method

The study of a phenomenon can be schematized as follows: the experimenter is interested in a quantity, for example the yield of wheat from a plot of land, the cost of a chemical product, or the wear of an automobile engine part. This quantity depends on a large number of variables. The study of the phenomenon then comes down to measuring the quantity as a function of the different values that can be given to the variables.

The main advantage of the method is that it allows for the variation of all the variable levels at each experiment, but in a programmed and reasoned manner. As surprising as it may seem at first, the fact of varying all the variables at once is not a disadvantage, but on the contrary offers many advantages, including:

- Reduction in the number of trials
- Large number of factors studied
- Detection of interactions between factors
- Better precision on the results
- Modeling of results and obtaining the optimum

Experimental designs allow the study of many factors while maintaining the number of trials

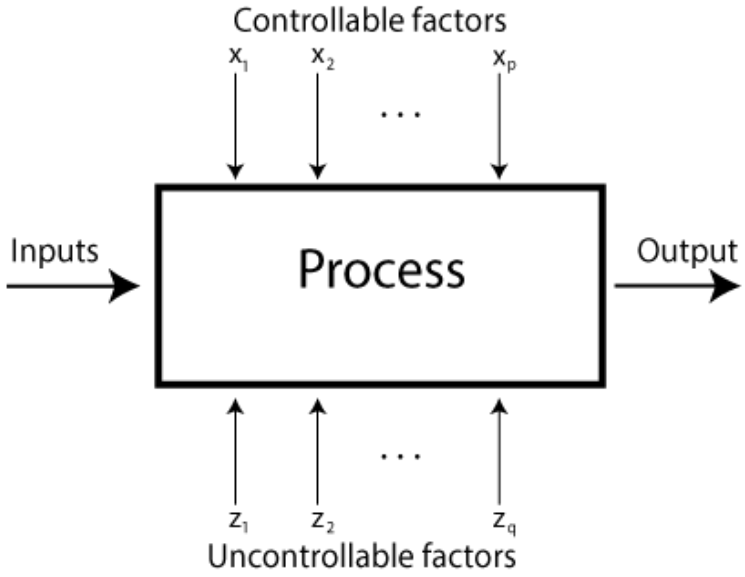


Figure 1.1: The system environment

at reasonable levels. One of their main applications is the search for influential factors. Understanding the method of experimental designs is based on two essential concepts, that of experimental space and that of mathematical modeling of the studied quantities [19].

1.3. Fundamental terminology of experimental designs

The technique of experimental designs has a specific vocabulary in experimental research methodology. Its specific terms are classic but in some statistical domains, they may have slightly different meanings. To ensure that our presentation is well understood, we prefer to recall the meaning of some important terms.

1.3.1. Response

An experimental response of a system is a measurable manifestation that is observed when varying the factors studied to determine their effect on the system. The response can be of a continuous quantitative type, such as yield, mechanical characteristic,...etc., or of a qualitative type. Quantitative responses are generally easier to handle.

1.3.2. Factors and experimental space

Factors are the variables that we want to study and are supposed to have an influence on the system. The value given to a factor to perform a test is called "Level". A factor can be:

• A controllable factor: It is a factor that we can manage, control or modify.

• A non-controllable factor: It is a factor considered as not retained for the study, because it is non-influential and left at its usual value, or an unknown factor that we undergo during the experimentation.

• A quantitative factor: It is translated by a measurable numerical quantity, such as speed, temperature, intensity, etc.

• A qualitative factor: It cannot be directly quantified, we can only identify its different levels, such as a brand, a process, a method, a supplier, etc.

When studying the influence of a factor, we generally limit its variations between two limits (the lower limit is the low level, and the upper limit is the high level).

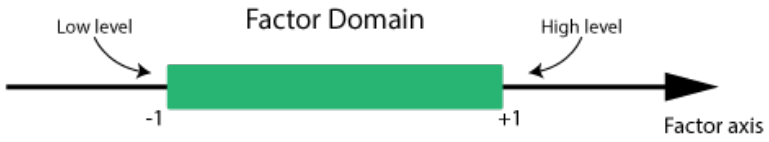


Figure 1.2: Variation domain of the factor.

The effect of a factor is the change in response caused by a change in level in one of the factors. The interaction between two factors characterizes the coupling of the effects of these two factors on the response. If there is a second factor, it is also represented by an axis. We define, as for the first factor, its low level, its high level, and its range of variation. This second axis is arranged orthogonally to the first. We thus obtain a Cartesian coordinate system that defines a Euclidean space with two dimensions. This space is called the experimental space (Figure 1.3).

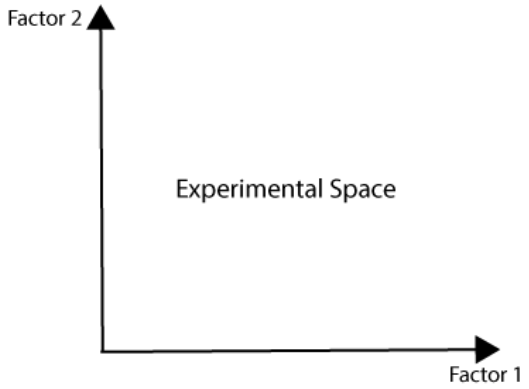


Figure 1.3: Experimental space definition.

The level X_1 of factor 1 and the level X_2 of factor 2 can be considered as the coordinates of a point in the experimental space (Figure 1.4)

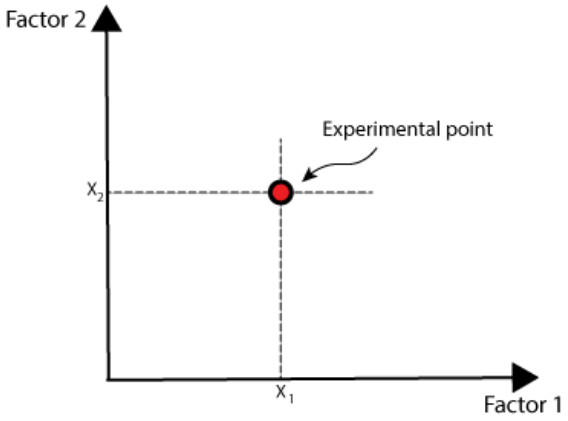


Figure 1.4: Experimental point in experimental space.

A given experiment is then represented by a point in this axis system. An experimental design is represented by a set of experimental points.

1.3.3. Domain of study and Response surface

The grouping of the factor domains defines the "study domain". Given the definition of the k factors and their respective variations, it becomes natural to define a k-dimensional space, in which each point corresponds to a configuration of the k factors. This space is called the study domain or research space. The experimental points can be located either inside or on the

boundaries of the domain (figure 1.5).

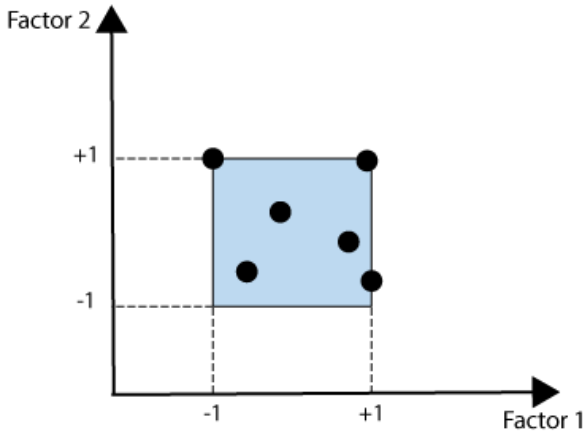


Figure 1.5: Two factors study domain.

Each point in the study domain corresponds to a response. The set of all points in the study domain corresponds to a set of responses that are located on a surface called the response surface. There are two types of response surfaces:

• Actual response surface: The actual response surface of the process is the set of values that the response takes.

• Theoretical response surface: In the case where the variables are continuous, a theoretical response surface can be calculated. In practice, this response surface is constructed from a few experimental points selected by the experimenter. Generally, the fundamental problem of experimental design is to seek to identify a polynomial model that allows for a better approximation of the actual response surface (Figure 1.6).

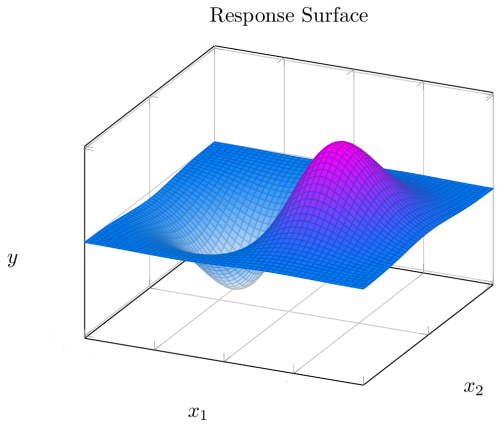


Figure 1.6: Definition of the response surface.

1.3.4. Centered reduced coordinates

When assigning the value of -1 to the low level of a factor and the value of +1 to the high level, two important modifications are made: the origin of the measurements is shifted and the unit of measurement is changed. These two modifications lead to the introduction of new variables called centered and scaled variables, centered to indicate the change in origin and scaled to denote the new unit. The transformation from the original variables Z to the centered and scaled variables X (dimensionless variables), and vice versa, is given by the following formula:

$$x = \frac{z - z_0}{step} \tag{1.1}$$

and $z_0 = \dfrac{highlevel + lowlevel}{2}$, $step = \dfrac{highlevel - lowlevel}{2}$

1.3.5. Experimental Designs

Each point in the study area represents possible operating conditions, and thus an experiment that the operator can perform.

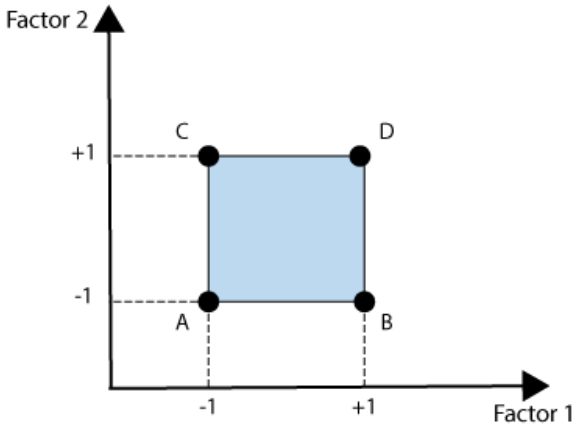


Figure 1.7: Design of experiments theory shows that the best point are those situated in the corners A,B,C and D.

The choice of the number and location of experimental points is the fundamental problem of experimental designs. We usually refer to sets of experimental points that meet specific properties as experimental designs. These are classical experimental designs, which are well-known and widely published. When the experimental points are arranged differently from classical experimental designs, they are called unconventional designs. Their properties are

often inferior to those of classical designs. However, unconventional designs are encountered because it is not always possible to meet the requirements of classical experimental designs [20].

1.3.6. Experimental Matrix

An experimental matrix is a mathematical object that represents, in coded or normalized form, the set of experiments to be performed. It is a table consisting of n rows, corresponding to the n experiments, and k columns, corresponding to the k variables (factors) being studied. The experimental matrix (Table 1.1) defines the trials represented in Figure 1.6. The element ij of the resulting matrix corresponds to the value of the level that the j-th variable takes on in the i-th experiment. The experimental matrix then defines the trials to be carried out. The term "trial" is equivalent to "experimental point" [21].

Table 1.1: Experimental Matrix

Trials	Factor 1	Factor 2
1(A)	-1	-1
2(B)	+1	-1
3(C)	-1	+1
4(D)	+1	+1

1.4. Mathematical Tools for Experimental Design

This section presents the basic mathematical concepts necessary for a good understanding of the experimental design method.

1.4.1. Concept of Mathematical Modeling

The model is a relationship between the factors x_1,x_2 , ... ,x_k and the response that one wishes to study.

1.4.2. Statistical Model

Consider a random phenomenon dependent on k variables, and suppose we seek to model this phenomenon as accurately as possible. The statistical approach then involves conducting n experiments, judiciously chosen in the case of experimental designs. Each experiment is represented by a point x in R^k (this is possible if the variables studied are quantitative; for the qualitative case, a subset of N^k is used).

Designating by $Y(x)$ the response measured at x, it is conventionally assumed that this response results from the sum of the response law f at x (i.e. the actual response sought) and the residual ϵ at x (i.e. the error made). Therefore:

$$Y(x) = f(x) + \varepsilon(x)$$

The residual can account for many causes such as errors due to the experimenter, a poor postulated model, the omission of certain variables, etc. We generally assume that the residuals are real random variables satisfying the following three hypotheses [22]:

$$\begin{cases} \forall\, x \qquad E(\varepsilon(x)) = 0 \\ \forall\, x \neq x', \qquad Cov(\varepsilon(x), \varepsilon(x')) = 0 \\ \forall\, x, \qquad Var(\varepsilon(x)) = \sigma^2 \end{cases}$$

1.4.3. Linear modeling

The linear model can be specified quite straightforwardly by the equations

$$y_i = f'(x)\beta + \varepsilon_i, \quad i = 1, ..., n \tag{1.2}$$

Where:

- y_i is the ith observation taken at the k explanatory variables specified by the vector $xi = (x_{i1}, x_{i2}, ..., x_{ik})$
- $f(x_i)$ is a $(p+1) \times 1$ vector of functions of those variables.
- $\beta = (\beta_0, \beta_1, ..., \beta_p)$ a conformable vector of unknown parameters.
- and ε_i is an error term which has mean zero and is not necessarily independent of other error terms.

Equation (2) is generally referred to as the observational equation and can be readily assembled in matrix form as

$$y = X\beta + \varepsilon_i$$

Where:

- $y = (y_1, ..., y_n)'$.
- The matrix X is $n \times (p+1)$ with ith row $f'(x_i)$.
- $\varepsilon = (\varepsilon_1, ..., \varepsilon_n)$ is distributed as a random vector with $E(\varepsilon) = 0$ and $Var(\varepsilon) = \sigma^2 I_n$.

Estimation of Parameters

Once the model is established, the problem now consists of determining an estimator $\hat{\beta}$ of β that is as good as possible. The classical approach is to seek $\hat{\beta}$ in such a way that the vector of observed responses Y and the vector of predicted mean responses $\hat{Y} = X\hat{\beta}$ are as close as possible. This leads to the estimator of the least squares of β according to the following definition:

Definition 1.1. We say that $\hat{\beta}$ is a least squares estimator of β if and only if $\hat{\beta}$ minimizes the function:

$$Q(\beta) = \|Y - X\hat{\beta}\|^2$$

where $\|.\|$ is the Euclidean norm in $\mathbb{R}^n$.

The least squares estimator of β provides the minimum of the function Q, and this minimum value is given by:

$$Q\left(\hat{\beta}\right) = \left\|Y - X\hat{\beta}\right\|^2 = \left\|Y - \hat{Y}\right\|^2 = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$$

This demonstrates that this quantity is closely related to the (quadratic) error between the observed responses Y_i and the average responses predicted by the model $\hat{Y}$. Regarding the practical determination of this estimator, it will be shown that:

Proposition 1.1. Let the statistical model be $Y = X\beta + \varepsilon$, where X is a full-rank matrix[1]. The least squares estimator of $\hat{\beta}$ is given by:

$$\hat{\beta} = \left({}^tXX\right)^{-1} {}^tXY$$

Proof. We seek $\hat{\beta}$ that minimizes the quantity $\left\|Y - \hat{Y}\right\|^2 = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$ Since $\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2$ is a scalar product, we can write it as:

[1] A matrix is said to be full-rank if no column is linearly dependent on the other columns, i.e., the rank of the matrix is equal to the number of columns of the matrix.

$\left\langle Y-\hat{Y}, Y-\hat{Y}\right\rangle = {}^t\left(Y-\hat{Y}\right)\left(Y-\hat{Y}\right)$ And since $\hat{Y}=X\hat{\beta}$ we have

$$\begin{aligned}\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2 &= {}^t\left(Y-X\hat{\beta}\right)\left(Y-X\hat{\beta}\ \right)\\ &= \left({}^tY-\ {}^t\hat{\beta}\,{}^tX\right)\left(Y-X\hat{\beta}\ \right)\\ &= {}^tYY-\ {}^t\hat{\beta}\,{}^tXY-\ {}^tYX\hat{\beta}+\ {}^t\hat{\beta}\,{}^tXX\hat{\beta}\end{aligned}$$

We have:

$${}^tYX\hat{\beta} = {}^t\left({}^t\hat{\beta}\,{}^tXY\right) = {}^t\hat{\beta}\,{}^tXY$$

Therefore:

$$\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2 = {}^tYY-2\,{}^t\hat{\beta}\,{}^tXY+{}^t\hat{\beta}\,{}^tXX\hat{\beta}$$

To minimize the value of $\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2$ we calculate its derivative with respect to $\hat{\beta}$

$$\frac{\partial\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2}{\partial\hat{\beta}} = \frac{\partial\,{}^tYY}{\partial\hat{\beta}}-2\frac{\partial\,{}^t\hat{\beta}\,{}^tXY}{\partial\hat{\beta}}+\frac{\partial\,{}^t\hat{\beta}\,{}^tXX\hat{\beta}}{\partial\hat{\beta}}$$

Thus:

- $\frac{\partial\,{}^tYY}{\partial\hat{\beta}}=0$ because ${}^tYY$ is constant with respect to $\hat{\beta}$
- $\frac{\partial\, t\hat{\beta}\,{}^tXY}{\partial\hat{\beta}} = {}^tXY$ because ${}^t\hat{\beta}\,{}^tXY$ is a linear form with respect to $\hat{\beta}$
- $\frac{\partial\,{}^t\hat{\beta}\,{}^tXX\hat{\beta}}{\partial\hat{\beta}} = {}^tXX\hat{\beta}$ because ${}^t\hat{\beta}\,{}^tXX\hat{\beta}$ is a quadratic form with respect to $\hat{\beta}$ which gives

$$\frac{\partial\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2}{\partial\hat{\beta}} = -2\,{}^tXY+2\,{}^tXX\hat{\beta}$$

$\hat{\beta}$ is a local minimum of $\left\|Y-\hat{Y}\right\|$ if

$$\begin{aligned}&\frac{\partial\sum_{i=1}^{n}\left(Y_i-\widehat{Y_i}\right)^2}{\partial\hat{\beta}}=0\Rightarrow -2\,{}^tXY+2\ {}^tXX\hat{\beta}=0\\ &\Rightarrow {}^tXX\hat{\beta} = {}^tXY\Rightarrow\hat{\beta}=\left({}^tXX\right)^{-1}{}^tXY\end{aligned}$$

We now need to verify that this value of $\hat{\beta}$ is a minimum, which is why we will calculate the second derivative of $\left\|Y-\hat{Y}\right\|$, which is equivalent to calculating $\dfrac{\partial^2 \sum_{i=1}^{n}\left(Y_i-\widehat{Y}_i\right)^2}{\partial\hat{\beta}^2}$

$$\begin{aligned}\frac{\partial^2 \sum_{i=1}^{n}\left(Y_i-\widehat{Y}_i\right)^2}{\partial\hat{\beta}^2} &= -2\,\frac{\partial\,{}^tXY}{\partial\hat{\beta}} + 2\,\frac{\partial\,{}^tXX\hat{\beta}}{\partial\hat{\beta}} \\ &= 2\,{}^tXX = 2\left\|X\right\|^2 > 0\end{aligned}$$

therefore it is indeed a minimum

□

Proposition 1.2. If the assumptions on the residuals (errors) are satisfied and if $\hat{\beta}$ is the least squares estimator of β, then:

1. $\hat{\beta}$ is an unbiased estimator of β
2. $\hat{\beta}$ has the variance-covariance matrix: $\mathbb{V}\left(\hat{\beta}\right) = \sigma^2\,({}^tXX)^{-1}$

Proof. 1. We calculate $\mathbb{E}\left(\hat{\beta}\right)$:

$$\mathbb{E}\left(\hat{\beta}\right) = \mathbb{E}\left(\left({}^tXX\right)^{-1}\,{}^tXY\right) = \left({}^tXX\right)^{-1}\,{}^tX\mathbb{E}\left(Y\right) = \left({}^tXX\right)^{-1}\,{}^tXX\beta = \beta$$

2. Similarly, we calculate $\mathbb{V}\left(\hat{\beta}\right)$:

$$\mathbb{V}\left(\hat{\beta}\right) = \mathbb{E}\left[\left(\hat{\beta}-\beta\right)^t\left(\hat{\beta}-\beta\right)\right]$$

□

1.4.4. Different types of experimental designs

In this section, we introduce the main types of experimental designs. These designs can be classified into two categories:

- Factorial designs,

- Response surface designs,

These two categories are related to possible objectives for using the method of experimental designs.

Two-Level Full Factorial Designs

A two-level full factorial design with n factors requires 2^n experimental runs to cover all possible combinations of the input factors. This is illustrated in the factor columns in Table 2.5 for the case of two factors ($n = 2$) at two levels each. If there are five factors at $n = 5$, the total number of runs will be $2^5 = 32$.

Table 1.2: The 2^2 experimental design plan

Run	A	B	Response
1	+1	+1	y_1
2	-1	+1	y_2
3	+1	-1	y_3
4	-1	-1	y_4

Fractional Factorial Design

Fractional factorial designs of rare designs in which no main effects are confounded with any other main effect. However, main effects are confounded with two-factor interactions and two-factor interactions may be confounded with each other.[23].

1.5. Response Surface Designs

In statistical practice, it is common to complement Design of Experiments (DOE) with the application of Response Surface Modelling (RSM). RSM encompasses various techniques used to interpolate or approximate the data obtained from a DOE. These techniques can include linear, nonlinear, polynomial, stochastic, and other methods, each offering distinct approaches within RSM. The main concept behind RSM is to construct a hypersurface in an (n + 1)-dimensional space, where n represents the variables of interest and the additional dimension represents the objective function. The advantage of creating this interpolating or approximating hypersurface is the ability to apply optimization techniques to the response surface, enabling the identification of optimal conditions or settings.[24].

Central Composite

A central composite design is an experimental design technique that combines a 2k full factorial design with additional central and star points. In this design, the star points represent sample points where all parameters, except one, are set at the mean level denoted as "m." The value of the remaining parameter is defined based on its distance from the central point. [24].

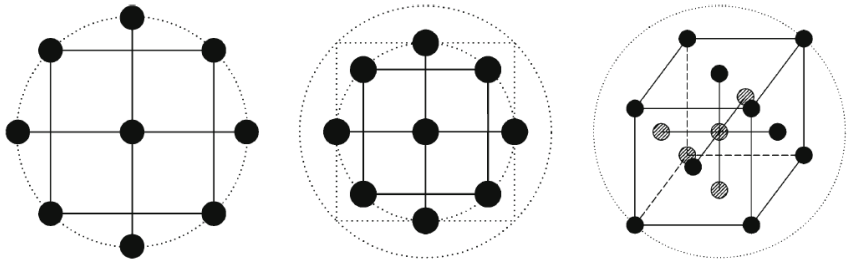


Figure 1.8: Example of central composite experimental designs.

Space Filling DOE Techniques

These methods utilize various techniques to uniformly fill the design space. As a result, they do not rely on the concept of levels, eliminate the need for discretized parameters, and allow the experimenter to independently choose the sample size regardless of the number of parameters in the problem. Space filling techniques are often a suitable choice for constructing response surfaces. This is because, given a certain sample size N, the occurrence of empty areas far from any sample, which could lead to inaccurate interpolation, is unlikely. However, since space filling techniques do not rely on levels, evaluating main effects and interaction effects of the parameters is not as straightforward as in the case of factorial experimental designs.

Latin Hypercube

In Latin hypercube sampling, the design space is divided into an orthogonal grid consisting of N elements of equal length per parameter. Within this multidimensional grid, N sub-volumes are identified such that only one sub-volume is selected along each row and column of the grid.

Halton, Faure, and Sobol Sequences

Several efficient space filling techniques are based on pseudo-random numbers generators. The quality of random numbers is checked by special tests. Pseudo-random numbers generators are mathematical series generating sets of numbers which are able to pass the randomness tests.

A pseudo-random number generator is essentially a function $\phi : [0,1) \longrightarrow [0,1)$ which is applied iteratively in order to find a serie of γ_k values

$$\gamma_k = \phi(\gamma_{k-1}), \qquad for \qquad k = 1, 2, ...$$

starting from a given γ_0. The difficulty is to choose ϕ in order to have a uniform distribution of γ_k. Some of the most popular space filling techniques make use of the quasi-random low-discrepancy mono-dimensional Van der Corput sequence. In the Van der Corput sequence, a base $b \geq 2$ is given and successive integer numbers n are expressed in their b-adic expansion form

$$n = \sum_{j=1}^{T} a_j b^{j-1}$$

where a_j are the coefficients of the expansion. The function

$$\varphi : \mathbb{N}_{\nvDash} \longrightarrow [0,1)$$
$$\varphi(n) = \sum_{j=1}^{T} \frac{a_j}{b^j}$$

gives the numbers of the sequence.
Let us consider $b = 2$ and $n = 4 : 4$ has binary expansion 100, the coefficients of the expansion are $a_1 = 0, a_2 = 0, a_3 = 1$. The fourth number of the sequence is $\phi_2(4) = \frac{0}{2} + \frac{0}{4} + \frac{1}{8} = \frac{1}{8}$. The numbers of the base-two Van der Corput sequence are: $\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}, ...$

The basic idea of the multi-dimensional space filling techniques based on the Van der Corput sequence is to subdivide the design space into sub-volumes and put a sample in each of them before moving on to a finer grid.

The Halton sequence employs a Van der Corput sequence with a base of two for the first dimension, a base of three for the second dimension, a base of five for the third dimension, and so on, using prime numbers as the base for subsequent dimensions. One of the primary challenges is to prevent multi-dimensional clustering. In high-dimensional spaces, the Halton sequence exhibits notable correlations between the dimensions. To address this issue, alternative sequences have been developed in an attempt to mitigate such correlations.

Faure and Sobol sequences use only one base for all dimensions and a different permutation of the vector elements for each dimension.[25]
The base of a Faure sequence is the smallest prime number ≥ 2 that is larger or equal to the number of dimensions of the problem. For reordering the sequence, a recursive equation is applied to the j coefficients. Passing from dimension $d-1$ to dimension d the reordering equation is:

$$a_j^{(d)}(n) = \sum_{j=1}^{T} \frac{(j-1)!}{(i-1)!(j-1)!} a_j^{(d-1)} mod\, b \tag{1.3}$$

Sobol sequence uses base two for all dimensions and the reordering task is much more complex than the one adopted by Faure sequence, and is not reported here. Sobol sequence is more resistant to high-dimensional degradation.

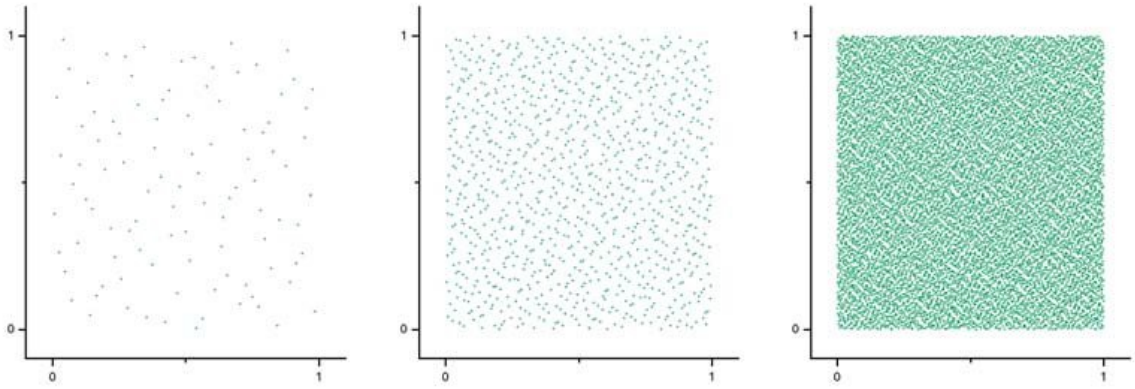


Figure 1.9: Halton sequences.

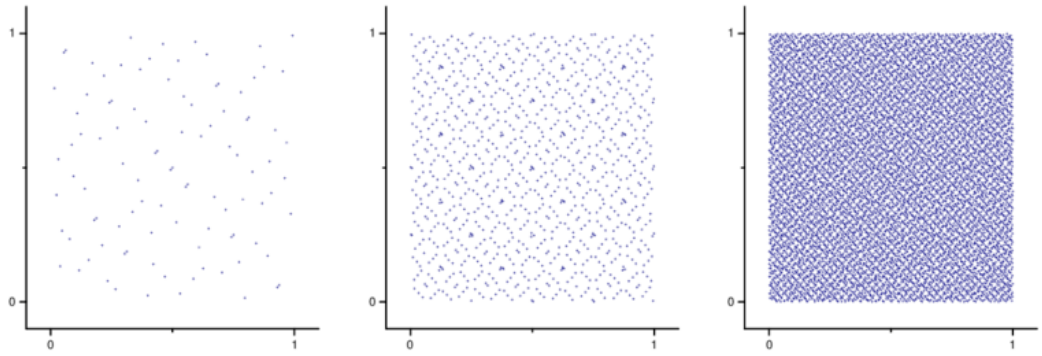


Figure 1.10: Sobol sequences.

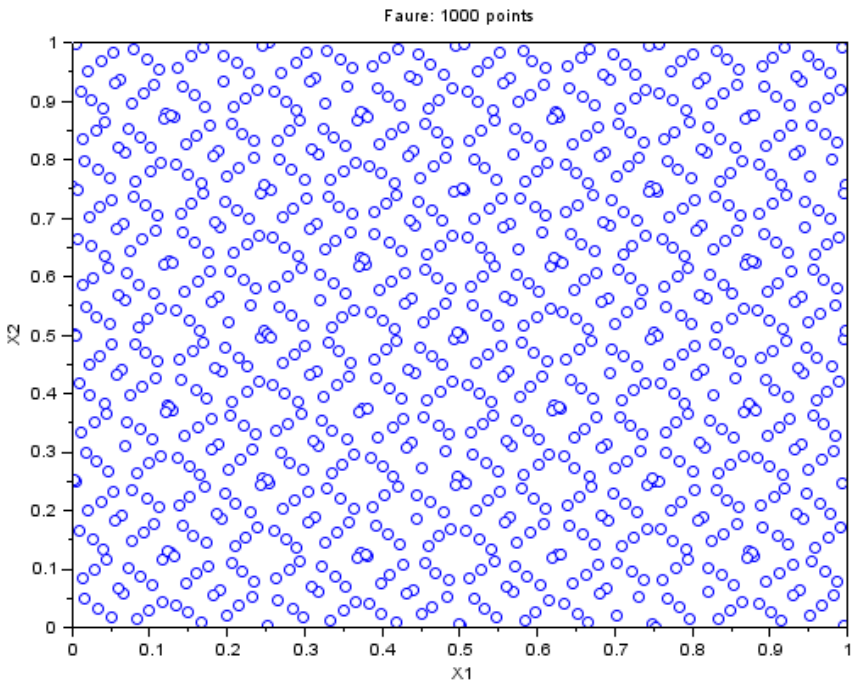


Figure 1.11: Faure sequence.

1.6. Conclusion

In summary, the design of experiments method is a set of complementary techniques that assist users in determining experiments to carry out and in understanding and exploiting the results obtained. The tools used in this method are essentially based on statistical and algebraic foundations. The developments in this chapter have presented the foundations, principles, and possibilities for analysis of the design of experiments method. Its multiple aspects make it a perfectly adapted method of analysis for studying systems.

Chapter 2

Generalities about point processes

Statistics uses its own types of models, which are of a different nature but often not more complicated than the models in classical statistics. These models may be used to formulate scientific hypotheses in terms of model parameters. Statistical approaches may then be used to test whether properties of the pattern derived from these hypotheses are reflected in the pattern, and hence whether the patterns support or disprove the hypotheses. In particular, these models enable the simulation of point patterns, which may be a helpful way to understand the underlying natural processes that have formed the pattern.

2.1. Point Process

point process or point field is a group of mathematical points distributed at random on a mathematical space, such as the real line or Euclidean space, in statistics and probability theory. Point processes can be applied to the analysis of geographical data, which is useful in a variety of fields including forestry, plant ecology, epidemiology, geology, seismology, materials science, astronomy, telecommunications, computational neuroscience, economics, and others. A point process can be interpreted mathematically in a variety of ways, such as a random set or a random counting measure. Although it has been noted that the distinction between point processes and stochastic processes is not entirely obvious, some authors view a point process and a stochastic process as two separate objects, with a point process being a random entity that comes from or is related to a stochastic process. The process is indexed by sets of the underlying space on which it is defined, such as the real line or n-dimensional Euclidean space, in the view of some who consider a point process as a stochastic process. In the theory of point processes, other stochastic processes including renewal and counting processes are examined.

Point processes are also known as random point fields since the term "point process" is not always favored because historically the word "process" signified an evolution of some system through time.

Definition 2.1. Mapped data such as that depicted in Figure can be described mathematically as a finite, unordered set of points

$$x = \{x_1, ..., x_n\}, \quad n = 0, 1, ...$$

in some space $\mathcal{X}$ (e.g. a square in R^2). For brevity, such sets will be referred to as configurations. Thus, we would like to model the stochastic mechanism underlying the data as a random configuration of objects parameterized by points in $\mathcal{X}$.

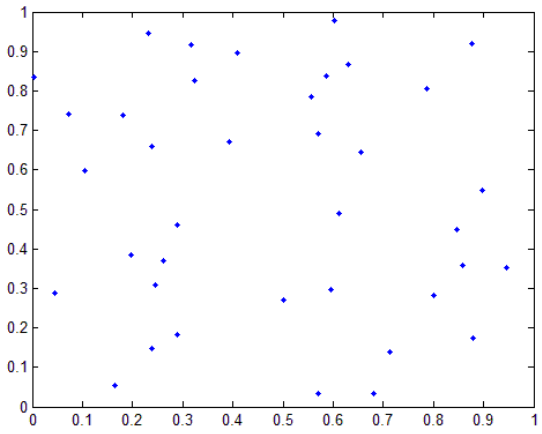


Figure 2.1: An Example Of Point Configuration.

It is assumed that $\mathcal{X}$ is equipped with a metric d such that $(\mathcal{X}, d)$ is complete and separable*. For example, $\mathcal{X}$ could be a compact subset of $\mathbb{R}^d$ equipped with the Euclidean distance. The metric defines a topology and hence a Borel σ-algebra. A configuration $x \subseteq \mathcal{X}$ is said to be locally finite if it places at most a finite number of points in any bounded Borel set $A \subseteq \mathcal{X}$ [26].

Definition 2.2. Let $(\mathcal{X}, d)$ be a complete, separable metric space. A point process on $\mathcal{X}$ is a mapping X from a probability space $(\Omega, \mathcal{A}, \mathcal{P})$ into N^{lf} such that for all bounded Borel sets $A \subseteq \mathcal{X}$, the number $N(A) = N_x(A)$ of points falling in A is a (finite) random variable.

Definition 2.3. A point process X is a random configuration of points such that for each bounded Borel set $A \subseteq \mathcal{X}$, the number of points in A is a random variable. In other words, a

point process is a random variable with values in the measurable space $(N^{lf}, \mathcal{N}^{lf})$, where $\mathcal{N}^{lf}$ is the smallest σ-algebra such that for all bounded Borel sets $A \subseteq \mathcal{X}$ the mapping $X \mapsto N_X(A)$ is measurable. The induced probability measure on $\mathcal{N}^{lf}$ is called the distribution of $\mathcal{X}$.

Definition 2.4. The family of finite-dimensional distributions (or 'fidis') of a point process X on a complete, separable metric space $(\mathcal{X}, d)$ is the collection of joint distributions of $(N(A_1), ..., N(A_m))$ for all finite vectors $(A_1, ..., A_m)$ of bounded Borel sets $A_i \subseteq \mathcal{X}, i = 1, .., m$ of any length $m \in N$.

Definition 2.5. The distribution of the point process is the induced probability measure π on $\mathcal{N}^{lf}$.

2.2. Examples Of Point Processes

In this section we are going to see some examples of point processes.

2.2.1. Binomial Point Process

binomial point process is defined as the union $X = X_1, ..., X_n$ of a fixed number $n \in N$ of independent , uniformly distributed points $X_1, ..., X_n$. Since $P(X_i = X_j) = 0$ for all $i \neq j, X$ is simple. Furthermore, as $P(N(\mathcal{X}) = n) = 1$, the binomial process is finite with

$$p_m = \begin{cases} 0 & \text{if } m \neq n \\ 1 & \text{if } m = n \end{cases}$$

The points Xi are distributed uniformly, hence

$$J_n(x_i, ..., x_n) \equiv \left(\frac{1}{\mu(\mathcal{X})}\right)^n$$

$(x_1, ..., x_m \in \mathcal{X})$. Clearly, $j_n(\cdot, ..., \cdot)$ is permutation invariant.x The binomial process derives its name from the fact that for any Borel set $A \subseteq \mathcal{X}$,

$$N(A) = \sum_{i=n}^{n} \mathbb{1}\{X_i \in A\}$$

follows a binomial distribution with parameters n and $\mu(A)/\mu(\mathcal{X})$.

2.2.2. Poisson Point Process

The Poisson process has a central role in point process statistics. It is fundamental to any successful analysis of point pattern data that the user is familiar with the basic properties of this process.

We shall derive a Poisson process on $\mathbb{R}^d$ in an intuitive way from the binomial process discussed above. The results will serve as the basis for a formal definition.
Let then, for $n \in N, P^n(\dot{)}$ be the distribution of a binomial process of n points in a ball $B_n \subseteq \mathbb{R}^d$ centred at the origin, with radius chosen in such a way that the volume $\mu(B_n) = \frac{n}{\lambda}$ for some $0 < \lambda < \infty$ Furthermore, let A be a bounded Borel set, and k a non-negative integer. Since the sequence of balls is increasing towards $\mathbb{R}^d$, an integer number $n_0 \geq k$ can be found such that for $n \geq n_0$, $A \subseteq B_n$. Hence, for $n \geq n_0$

$$\begin{aligned} P^{(n)}(N(A) = k) &= P^n(N(A) = k; N(B_n \setminus A) = n - k) \\ &= \binom{n}{k} \left(\frac{\mu(A)}{\mu(B_n)}\right)^n \left(\frac{\mu(B_n \setminus A)}{\mu(B_n)}\right)^{n-k} \end{aligned} \tag{2.1}$$

Hence the number of points in A is binomially distributed with parameters n and $\mu(A)/\mu(B)$. Hence

$$\lim_{n\to\infty} P^{(n)}(N(A) = k) = e^{-\lambda\mu(A)} \frac{(\lambda\mu(A))^k}{k!}$$

The properties derived above suggest the following definition. A point process X on $\mathcal{X} = \mathbb{R}^d$ is a homogeneous Poisson process with intensity (or rate) $\lambda > 0$ if

- $N(A)$ is Poisson distributed with mean $\lambda\mu(A)$ for every bounded Borel set $A \subseteq \mathcal{X}$, writing $\mu(A)$ for the volume (Lebesgue measure) of A;
- for any k disjoint bounded Borel sets $A_1, ..., A_k$, the random variables $N(A_1), ..., N(A_k)$ are independent.

2.2.3. Strauss Process

Strauss point process (Strauss, 1975) (Kelly and Ripley, 1976) [27] is an example of the pairwise interaction models in which each single point contributes the same interaction function to the density $G\hat{(}r) = \sum_i e(x_i, r)\mathbb{1}\{t_i \leq r\}$ (Nearest neighbour distance function) irrespective of its position, and each pair of distinct points, which are not more than 'r' apart and are thus defined to be neighbor, contributes a constant interaction = γ. Based on the above definition,

the Strauss process is defined by the density given as:

$$f(x) = \alpha\beta^{n(x)}\gamma^{s(x)}$$

where α is the normalizing constant, β is the intensity of the process, $n(x)$ is the number of points and $s(x)$ is the number of pairs of distinct points in x which are not more than r units apart. The parameter γ controls the strength of interaction between points. If $\gamma = 0$ the model is a hard core process. For $0 < \gamma < 1$, the process exhibits inhibition (repulsion) between points. If $\gamma = 1$ the model reduces to a Poisson process with intensity β. For $\gamma > 1$, the density is not integrable. Originally the Strauss model was proposed as a model for clustering when $\gamma > 1$ but later it turned out to be a model for inhibition and is defined only for $0 < \gamma < 1$ (Turner, 2007).

The figure shows some simulated realizations of a simple Strauss model in a square region of 10 units for different values of intensity, interaction parameter, and interaction radius given as Strauss (β, γ, r) [28].

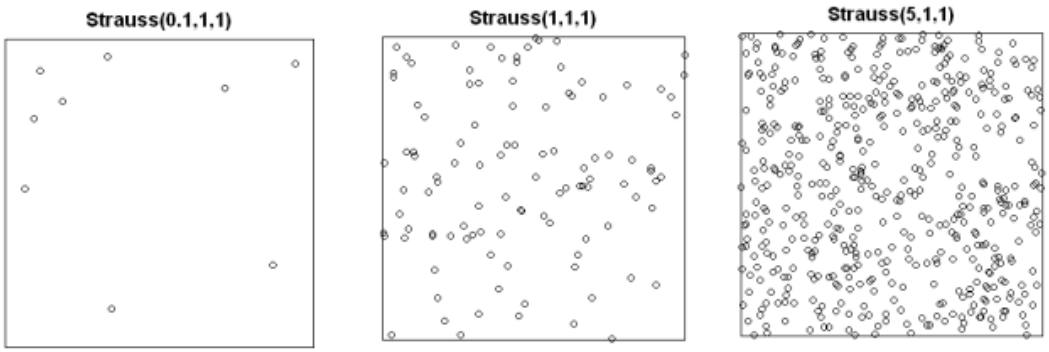


Figure 2.2: Simulation of Strauss model.

2.3. Markov Point Process

This class of models is specially designed to take into account inter-point interactions but includes the Poison process and certain Poisson cluster models as well.

Definition 2.6. The neighbourhood $\delta(A)$ of a set $A \subseteq \mathcal{X}$ is defined as

$$\delta(A) = \{x \subseteq \mathcal{X} : x \sim a \, for \, some \, a \in A\}$$

. In particular, the neighbourhood of a singleton $A = \{a\}$ contains all neighbours of a,

$$\delta(\{a\}) = \{x \subseteq \mathcal{X} : x \sim a\}$$

.

Definition 2.7 (Ripley-Kelly)**.** Let $(\mathcal{X}, d)$ be a complete, separable metric space, $v(.)$ a finite, non-atomic Borel measure, and $\pi_v(.)$ the distribution of a Poison process on $\mathcal{X}$ with intensity measure $v(.)$. Let X be a point process on $\mathcal{X}$ specified by means of a density $p(.)$ with respect to $\pi_{v(.)}$. Then X is a Markov point process with respect to the symmetric, reflexive relation $\sim$ on $\mathcal{X}$ if for all $x \in N^f$ such that $p(x) > 0$,

(a) $p(y) > 0$ for all $y \subseteq x$;

(b) for all $u \subseteq X, \frac{p(x \bigcup \{u\})}{p(x)}$ depends only on u and $\delta((\{u\}) \bigcap x = \{x \subseteq x : u \sim x\}$.

Condition (a) translates that if a configuration can occur, then all the sub-configurations it contains can also occur. The quotient $\frac{p(x \bigcup \{u\})}{p(x)}$ in condition (b), known as the Papangelou conditional intensity [29], is the probability density that a point u occurs given that x is realized elsewhere. This condition expresses a local Markov property: the behavior of a point u with respect to the entire configuration depends only on its close neighbors in that configuration.

2.4. Markov Chains

We call a discrete-time Markov chain any sequence of random variables $(X_n)_{N \in \mathbb{N}}$ taking values in a set A, such that, for any integer $K \in N$ and $A \in A$, the sequence satisfies the following Markov property:

$$P(X_{N+1} \in A | X_0, X_1, ..., X_N) = P(X_{N+1} \in A | X_N), \ \forall A \in A$$

In other words, the value of a random variable in this sequence depends only on the preceding one. We will focus on homogeneous chains, which means that their evolution does not depend on the position in the chain, but only on the current state.

$$P(X_{t_1}, X_{t_2}, ..., X_{t_k} | X_{t_0}) = P(X_{t_1 - t_0}, X_{t_2 - t_0}, ..., X_{t_k - t_0} | X_0)$$

From a computer science perspective, such a chain has the advantage of making it unnec-

essary to recall all previous configurations since it only uses the current state to generate a new configuration. Generating the new configuration requires the definition of a transition kernel. A transition kernel is a function $P : \chi \times A \to [0, 1]$, such that:

- For any $A \in A, P(\cdot, A)$ is measurable.
- For any $X \in \chi$, the function $P(X, \cdot)$ is a probability measure.

Thus, a homogeneous Markov chain is completely defined by the value or distribution of X_0 and its transition kernel P. We can approximate a density π by using a Markov

chain whenever we can construct a transition P such that, for any initial distribution $X_0 = \nu, \nu P^k \to \pi$. Before presenting the various theoretical results and simulation algorithms, let us review some preliminary definitions about Markov chains.

- **Invariance:** A distribution π is invariant for the Markov chain if:

$$\pi = \pi P$$

This condition is necessary to achieve the convergence of the chain towards π

- **Reversibility:** The chain is reversible for π if the transition kernel P satisfies:

$$\forall A, B \in A : \int P(x, A)\pi(dx) = \int P(x, B)\pi(dx)$$

This condition implies invariance for π and means that under the stationary distribution π, the probability of transitioning from A to B is the same as transitioning from B to A. Most simulation algorithms are actually designed to produce reversible Markov chains.

- **Irreducible:** he chain is said to be π-irreducible if:

$$\forall x \in \Omega \; and \; \forall A \in A, \; such \; that \; \pi(A) > 0, \; \exists t / \; p^t(x, A) > 0$$

This means that the chain has a non-zero probability of reaching any π-probable set in finite time. This condition is clearly necessary for the chain to converge in distribution to π with any initial condition. In the context of this project, we will only consider irreducible chains, which means chains that have only one class of states.

- **A Periodicity:** Periodicity ensures that the transitions between states are not too constrained. Formally, the chain is said to be aperiodic if there does not exist a disjoint partition

$A = \bigcup_{i=0}^{k} A_i$ for $r \geq 2$ such that:

$$P(x, A_i) = 1, \forall x \in A_i$$

There is a connection between irreducibility and aperiodicity, in the sense that if a Markov chain is irreducible and there exists a state X_k such that the density $P(X_k, X_k) > 0$, the Markov chain is strongly aperiodic (this is particularly relevant in the context of Metropolis-Hastings type algorithms that we will study later)

- A π irreducible and π invariant chain is positive recurrent if $\forall A \in A \; such \; that \; \pi(A) > 0$, it satisfies:

$$\forall x, P_x\{X \in \; infinitely \; often \; \} > 0 \; and \; P_x\{X \in \; infinitely \; often \; \} = 1$$

- In a discrete space, the transition kernel P is primitive (or regular) if $\exists \; k \geq 1$ such that P^k has all its terms strictly positive.

2.4.1. Convergence of a Markov chain

Before presenting the methods for simulating MCMC chains, we introduce the necessary conditions for a chain to converge and reach the desired distribution π. If P is π-irreducible, *pi*-invariant, and recurrent, then the invariant measure is unique, and the chain is said to be positive recurrent if the total mass of this measure is finite. This holds true if π is a probability. The ergodicity results are obtained [30]:

Proposition 2.1. If P is π-irreducible and P is π-invariant, then P is positive recurrent and π is the unique invariant distribution of P. If P is aperiodic, then for $x \in \chi$, we have:

$$\|P^m(x, .) - \pi\| \to 0 \text{ when } m \to \infty$$

Controlling the convergence of $\|P^m(x, .) - \pi\| \to 0$ is a central and very challenging question. This control ensures that νP^m for sufficiently large m provides an acceptable simulation of π. There exist numerous theoretical results, some of which utilize lower bounds of P on a small set E. The contraction coefficient on a finite state space allows for the control of this convergence [31].

- **Geometric ergodicity, uniform ergodicity:** Geometric ergodicity is characterized

by:

$$\|P^m(x,.) - \pi\| \leq M(x)l^m$$

Where $M(x)$ is π-integrable and $l < 1$. Uniform ergodicity is achieved if we can choose a finite constant for M.

- **Coefficient of Contraction:** In a finite state space, the coefficient of contraction for a transition kernel P is given by [32]:

$$C(P) = \frac{1}{2} \max_{x,y \in E} \|P(x,\cdot) - P(y,\cdot)\|$$

Lemma 2.1 (Winkler)**.** [33] Let ν and μ be two distributions, P and Q be two transition kernels. Then,

$$\|\mu P - \nu P\| \leq \|\mu - \nu\| C(P) \quad \text{and} \quad C(PQ) \leq C(P)C(Q)$$

In particular,

$$\|\mu P - \nu P\| \leq \|\mu - \nu\| \quad \text{and} \quad \|\mu P - \nu P\| \leq 2C(P)$$

And if P is primitive, then $C(P) \leq 1$.

According to this result, if we take $\mu = \pi$, the invariant distribution of P, we can deduce:

$$\|\nu P^m - \pi P^m\| = \|\nu P^m - \pi\| \leq 2C(P^m) \leq 2C(P)^m$$

In this case, as $m \to \infty$, the chain is uniformly ergodic if P is primitive.

2.5. MCMC - Metropolis Hasting

2.5.1. Markov Chain Monte Carlo

In recent decades, Markov Chain Monte Carlo (MCMC) methods have received a great deal of attention due to advances in understanding and increased computing power. These algorithms primarily allow for the approximation of expectations (probability integrals) through numerical simulations. As their name suggests, these methods combine both the Monte Carlo principle

- approximating an integral by the mean of a sample - and the concept of a Markov chain - a temporal sequence of independent random variables. The Markov chain produces the Monte Carlo sample used to approximate the desired expectation.

Monte Carlo Approach

In statistics, several situations lead to the calculation of expectations of the form

$$\pi(f) = \int_{\mathcal{X}} f(x)\pi(dx)$$

where π is a probability measure on a state space $\mathcal{X}$ and $f : \mathcal{X} \to \mathbb{R}$ is a π-measurable function. However, it may happen that the analytical calculation of this expectation is not possible, and alternative methods must be used to obtain an approximation of $\pi(f)$. A common approach is the use of numerical integration methods, which involve dividing the space $\mathcal{X}$ into rectangles and approximating the function f with simpler functions. However, this method can pose problems when the dimension of $\mathcal{X}$ is high or the integration domain is unbounded. Monte Carlo methods, which are specifically tailored to expectations, are generally less affected by these issues. [34]

Monte Carlo Method

Monte Carlo methods are based on the fundamental principle of the law of large numbers. In fact, the law of large numbers is a mathematical result that states that, under certain conditions, the average of a large number of independent realizations of the same random variable converges to its expected value.

$$\frac{1}{N}\sum_{n=1}^{N} f(x_n) \longrightarrow \pi(f), \qquad n \to \infty,$$

In other words, the average of the function f taken over a sample $\{x_n\}_{n=1}^{n}$ converges, under certain conditions, to the expectation of this function taken under the distribution π, for a mode of convergence $c \in \{Almost\, everywhere\}$. This sample average can, in turn, be seen as the following expectation:

$$\frac{1}{N}\sum_{n=1}^{N} f(x_n) = \int_{\mathcal{X}} f(x)\hat{\pi}_N(dx) = \hat{\pi}_N(f),$$

Where $\hat{\pi}_N$ is the empirical mass function of the Monte Carlo sample $\{x_n\}_{n=1}^{n}$, given by

$$\hat{\pi}_N = \frac{1}{N}\sum_{n=1}^{N} \delta_{x_n}(x)$$

The delta mass function at y is denoted by $\delta_y(.)$. Thus, the empirical measure $\hat{\pi}_N$ is used instead of π in the expectation. In order for the Monte Carlo estimator $\hat{\pi}_N(f)$ to satisfy the law of large numbers, the properties of the sample $\{x_n\}_{n=1}^n$ must be such that the empirical measure $\hat{\pi}_N$ provides a good approximation of the distribution π. Therefore, different Monte Carlo methods generally differ only in the way the sample is generated. A second property often sought for an estimator is a central limit theorem. This type of result shows that the asymptotic distribution of the estimator is a Gaussian distribution:

$$N^{-1/2}\hat{\pi}_N(f) \xrightarrow{\mathcal{D}} \mathcal{N}(\pi(f), \sigma_f^2), \qquad n \to \infty$$

The notation $\xrightarrow{\mathcal{D}}$ is used to denote convergence in distribution, and σ_f^2 is the asymptotic variance of the estimator. When a central limit theorem is satisfied, it is possible to add a Monte Carlo standard error to the point estimate $\hat{\pi}_N(f)$, given by $\hat{\sigma}_f^2/\sqrt{N}$, where $\hat{\sigma}_f^2$ is an estimate of σ such that the sampling variance:

$$\hat{\sigma}_f^2 = \frac{1}{N}\sum_{n=1}^{N}(f(x_n) - \hat{\pi}_N(f))^2$$

It is therefore possible to provide an assessment of the quality of the estimation made by the Monte Carlo estimator when such a result is verified.

Note that most of the terms used in Monte Carlo methods are also used more generally in statistics (estimator, standard error, sample, etc.). To distinguish the two concepts, the mention "Monte Carlo" is often added after these terms. This distinction is particularly relevant when the expectation $\pi(f)$ itself depends on a real sample from an experiment; the Monte Carlo sample is a collection of points in $\mathcal{X}$ and not the set of units in the experiment. Similarly, the Monte Carlo standard error does not correspond to the standard error of the sample mean.

Monte Carlo Algorithm

The simplest versions of the law of large numbers require that each element of the sample be a realization of independent and identically distributed random variables according to the target distribution π, denoted by $X_n \xrightarrow{i.i.d} \pi$. The empirical distribution will therefore be representative of the target distribution since the sample is generated from π itself. This type of sampling, called i.i.d. sampling and described in the algorithm below constitutes the standard Monte Carlo method.

Algorithm 1 Monte Carlo Algorithm i.i.d

Data: Distribution target π and size of the Monte Carlo sample N.
Procedure: for $n = 1, ..., N$ sample $X_n \xrightarrow{i.i.d} \pi$.
Output: The sample $x_{1:N}$ and Monte Carlo estimator.

MCMC Algorithm

Markov chain Monte Carlo (MCMC) methods generate the sample Monte Carlo sample sequentially using a Markov chain which contains sequential dependence. The algorithm below details the general procedure.

Algorithm 2 General MCMC algorithm

Data: Target distribution π, Markov transition P and sample size Monte Carlo N.
Procedure:
1 initialization. Initial value of the chain, x_0.
2 For $n = 0, ..., N-1$,
Sampling. Generate the new state of the string:

$$X_{n+1}|X_n = x_n \sim P(.|_n)$$

.
Output: The sample $x_{0:N}$.

2.5.2. Metropolis Hasting

One of the most widely used MCMC algorithms whose properties are best known is the Metropolis-Hastings (MH) algorithm. It is an algorithm based on the principle of acceptance/rejection of candidates using a Markov chain. As for any Markov chain, a new state of the chain is generated from the current state. To do so, a candidate generated conditionally to the current state is proposed as the new state of the chain; then, the new state of the chain is chosen according to a certain probability between this proposal and the current state. The exact procedure is described in Algorithm 2. For a certain choice of acceptance probability and for certain conditions on the instrumental distribution generating the candidates, the produced Markov chain will be ergodic to the target distribution.

Definition 2.8. Let Π be a target distribution that admits a density π with respect to a σ-finite measure μ and let Q be a Markov transition kernel admitting a density q with respect to μ, called the instrumental density, i.e,

$$Q(dy|x) = q(y|x)\mu(dy)$$

The Metropolis-Hastings acceptance probability is defined as

$$\alpha(y|x) = min\left\{1, \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)}\right\}$$

A Metropolis-Hastings kernal takes the following form

$$P(B|x) = \int_B \alpha(y|x)Q(dy|x) + r(x)\mathbb{1}(x \in B)$$

and has the following (pseudo) density

$$p(y|x) = \alpha(y|x)q(y|x) + r(x)\delta_x(y)$$

Where $\delta_x(.)$ is the Dirac delta mass function at x and $r(x)$ is the probability of the chain remain in x, given by

$$r(x) = 1 - \int_{\mathcal{X}} \alpha(y|x)Q(dy|x)$$

Algorithm 3 Metropolis-Hastings algorithm

Data: The target density π, instrumental density q, and sample size Monte Carlo N.
Procedure:
1 initialization. Initial value of the chain, x_0.
2 For $n = 0, ..., N-1$,
(a) Proposal. Generate proposal

$$Y|X_n = x_n \sim q(.|x_n)$$

(b) Acceptance. With probability

$$\alpha(y|x) = min\left\{1, \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)}\right\}$$

accept the proposition($x_{n+1} = y$); otherwise reject the proposition ($x_{n+1} = x_n$)
Output: The sample $x_{0:N}$.

Chapter 3

New Configuration of Computer Experiment Designs

Area interaction processes are a type of point process model used to analyze the spatial distribution of points in a given area. These models can be analyzed using Markov Chain Monte Carlo (MCMC) algorithms, which are a class of computational methods used to simulate complex probability distributions. MCMC algorithms are particularly useful for analyzing area interaction processes because they allow for the estimation of posterior distributions of model parameters, which can be used to make inferences about the underlying spatial processes. In this way, MCMC algorithms provide a powerful tool for analyzing complex spatial data and can be used to address a wide range of research questions in fields such as ecology, epidemiology, and spatial statistics.

3.1. Area Interaction Process

In this section, We introduce a family of Markov point processes that can produce patterns that are both moderately clustered and moderately ordered. It is possible to say that they interact in infinitely complex ways. In the most basic case the probability density of a point pattern $x = \{x_1, ..., x_n\}$ $(n \geq 0)$ in a window $A \subseteq \mathbb{R}^2$ is defined to be [35]

$$\pi(x) = \alpha\beta^n\gamma^{-u(x)}$$

where $u(x)$ is the area of the plane set formed by taking the union of discs of radius r centred at the points x_i. Here $\beta.\gamma.r > 0$ are parameters and α is the normalising constant.

Definition 3.1. The area-interaction process in a compact region $A \subseteq \mathbb{R}^d$ is the process with density

$$\pi(x) = \alpha\beta^{n(x)}\gamma^{-m(U_r(x))} \tag{3.1}$$

Where m in this case is Lebesgue measure, and

$$U_r(x) = \bigcup_{i=1}^{n} B(x_i, r)$$

where $B(x_i, r)$ is a disc of radius r centered at each data point x_i such that

$$B(x_i, r) = \{a \in \mathbb{R}^d : \|a - x_i\| \leq r\}$$

And $U_r(x)$ is the union of spheres or discs of radius r centred at the points of the realisation. The area of the union of discs may be expressed as the decomposition of the union of grains, $U_r(x)$, in an inclusion-exclusion style This is expressed concisely as:

$$m\left(U_r(x)\right) = \sum_{i=1}^{n(x)} m(B(x_i, r)) - \sum_{i<j} m(B(x_i, r) \cap B(x_j, r)) + ... + (-1)^{n(x)+1} m\left(\bigcap_{i=1}^{n(x)} B(x_i, r)\right)$$

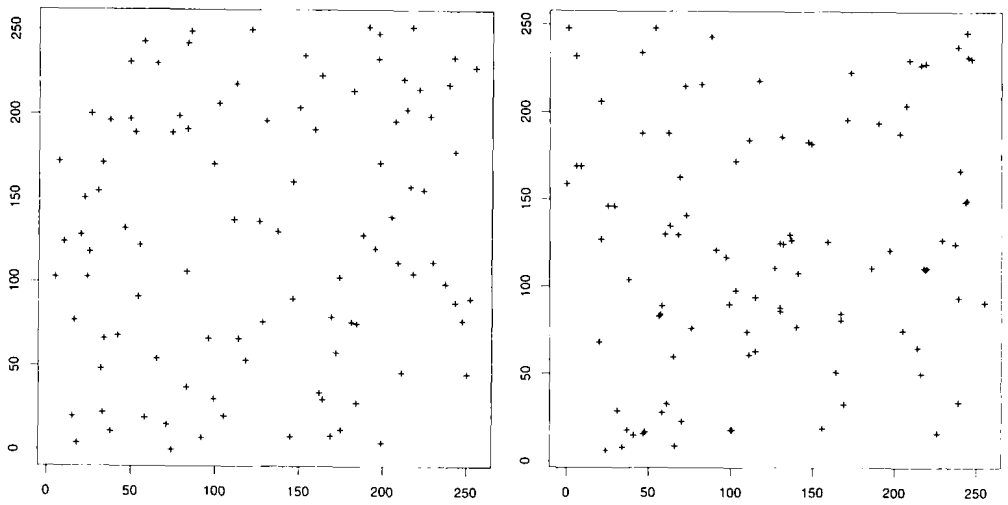


Figure 3.1: Simulated realizations of an area-interaction process conditional on $n = 100$ points, with $r = 5$ in a window of size 256 x 256. Left: ordered pattern, $\gamma = 0.9711$, $\gamma^{25\pi} = 10$; Right: clustered pattern, $\gamma = 1.02975$, $\gamma^{25\pi} = 0.1$.

3.2. Voronoi Tessellation

The Voronoi diagram, also known as a Dirichlet tessellation, is a fundamental geometric structure with numerous applications in various domains such as form modeling, motion planning, scientific visualization, geography, chemistry, and biology. This mathematical construct, named after German mathematician Peter Gustav Lejeune Dirichlet, is used to divide a space into regions based on the proximity of points. In a Dirichlet tessellation, each point in a set of distinct points defines a region of space that is closer to it than any other point, forming tiles that partition the space. The tessellation can be extended to any number of dimensions, with the planar case being the first nontrivial one. Green and Sibson developed an efficient algorithm for computing the tessellation in the planar case in 1978. These tessellations have been applied in various fields, including statistics and data analysis, where they help in understanding spatial relationships and patterns [36].

A point seed x_i defines a Voronoi cell, V the subset of the domain that is closer to that seed than any other seed. The cell equation is related to the maximal sampling condition

$$V_i = \{p\} \in \mathcal{D} : \forall j, \|p - x_i\| \leq \|p - x_j\|$$

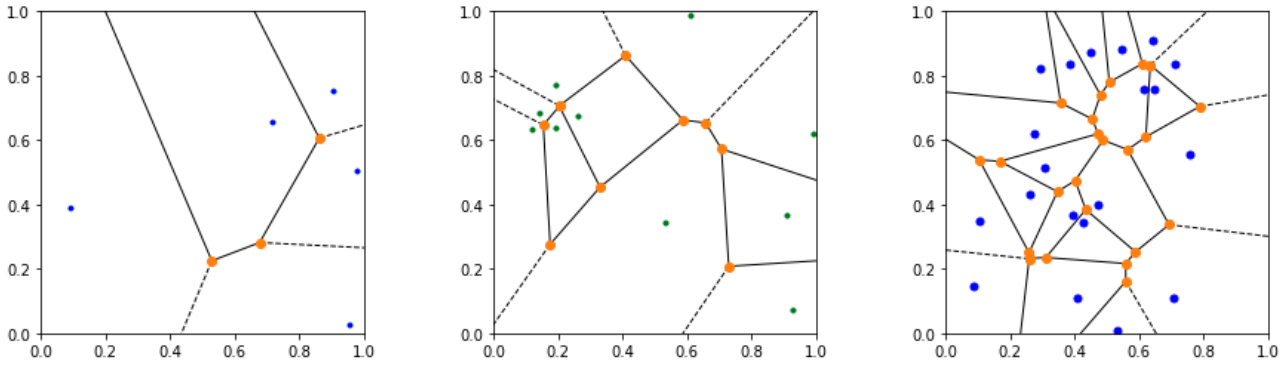


Figure 3.2: Examples of Voronoi tessellation generating 5, 10 and 20 points on the unit square.

3.3. Simulation of point processes using MCMC method and Metropolis-Hasting algorithm.

The method involves constructing a chain $\{X_0, X_1, \cdots, X_N\}$ that converges to the desired distribution π. In fact, the Metropolis-Hastings algorithm can construct this chain using a

π-reversible transition kernel. The algorithm proceeds in two steps:

- A proposal for a state change is made: from x to y, according to a probability distribution $Q(x,.)$ with density $q(x,y)$ called instrumental density.
- y is accepted with probability $a(x,y)$, otherwise the state remains at x (where $a : \Omega \times \Omega \rightarrow [0,1]$). The transition kernel is given by [Chib, 1995]:

$P_{MH}(x,y) = a(x,y)\, q(x,y) + \left[\int_{\Omega} 1 - a(x,z)\, q(x,z)\, dz\right] \delta_x(y)$ Where, $\delta x(.)$ is the Dirac measure at x, and for simplicity of the calculations, we use the mass of the point at x. ($\delta x(y) = 1$ if $x = y$, and 0 otherwise). The choice of (Q,a) will ensure the π-reversibility of P_{MH} if the following equilibrium equation is satisfied:

$$\forall x,\ y \in \Omega : \pi(x) \times q(x,\ y) \times a(x,\ y) = \pi(y) \times q(y,\ x) \times a(y,\ x)$$

The choice of the acceptance probability a is more limited and essentially dictated by the goal of simulating a given probability distribution π asymptotically. The usual choice is:

$$a(x,\ y) = \frac{\pi(y) \times q(y,\ x)}{\pi(x) \times q(x,y)}$$

There are a few noteworthy aspects to consider. Firstly, the calculation of $a(x,y)$ does not depend on the normalization constant of (3.1). Secondly, in this study, we focus on scenarios where the configurations x and y exhibit variations at multiple points, termed as homogeneous birth and death dynamics of point sets. Consequently, the selected density function q is typically symmetric, enabling a more manageable calculation process: $q(x,y) = q(y,x)$. As a result, the acceptance probability is reduced to:

$$a(x,y) = \frac{\pi(y)}{\pi(x)} = \frac{\beta^{n(y)}\gamma^{-m(U_r(y))}}{\beta^{n(x)}\gamma^{-m(U_r(x))}} = \frac{\gamma^{-m(U_r(y))}}{\gamma^{-m(U_r(x))}}$$

3.4. Algorithm for constructing experimental designs using Markov point process with area interaction

The numerical experimental designs proposed in this work are generated using the following algorithm, which is actually a version of the Metropolis-Hastings algorithm.

Algorithm 4 Constructing Experimental Designs Using Markov Point Process with area interaction

Initialization step: Choose an initial configuration (a design of experiments) $X_0 = x$ according to a given probability distribution, for example, the uniform distribution.

Iteration step:

for $i = 1, 2, \ldots, N_{MCMC}$ **do**

for each configuration x **do**

Subdivide the configuration x into neighborhoods $\vartheta(x_k)$ for each point x_k, $k \in \{1, 2, \ldots, n\}$ using the Voronoi-Dirichlet tessellations given by:

$$\vartheta(x_k) = \{s \in [0,1]^p : \forall q \in x, \|s - x_k\| \leq \|s - q\|\}$$

For each neighborhood $\vartheta(x_k)$, simulate an experiment y_k according to the proposal distribution $q \sim U_{\vartheta(x_k)}$.

Take $y = x \cup \{y_1, y_2, \ldots, y_n\}$.

Choose a pair of points $\{u, v\}$ from the configuration y uniformly at random, then choose u or v with probability 0.5 and remove it from y (i.e., $y = \begin{cases} y \setminus \{u\} & \text{if } u \text{ is chosen} \\ y \setminus \{v\} & \text{if } v \text{ is chosen} \end{cases}$).

Repeat the fourth step n times.

The new configuration is then taken to be y.

end for

Calculation of the acceptance probability $a(x, y) = \min\left(1, \gamma^{m(U_r(x)) - m(U_r(y))}\right)$.

Accept the proposal $x = y$ with probability $a(x, y)$.

end for

Result: Take $X_N = x$.

For $N = 3000$, Figure 3.3,Figure 3.4 and Figure 3.5 demonstrate the convergence of a configuration that features the fulfillment of the area interaction process starting from an initial configuration of 20 points.

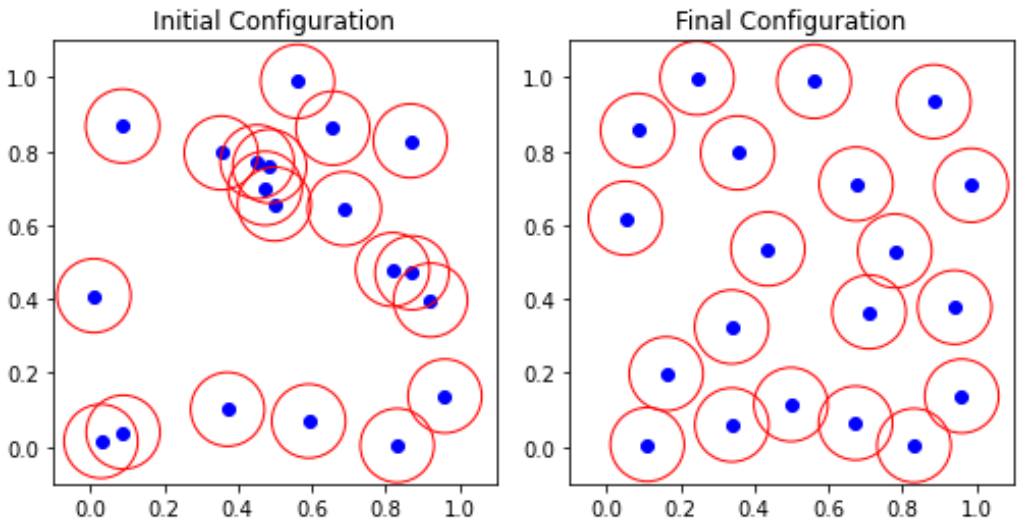


Figure 3.3: Initial configuration with area = 0.49 & final configuration with area = 0.61 ($\gamma = 3$, $r = 0.1$).

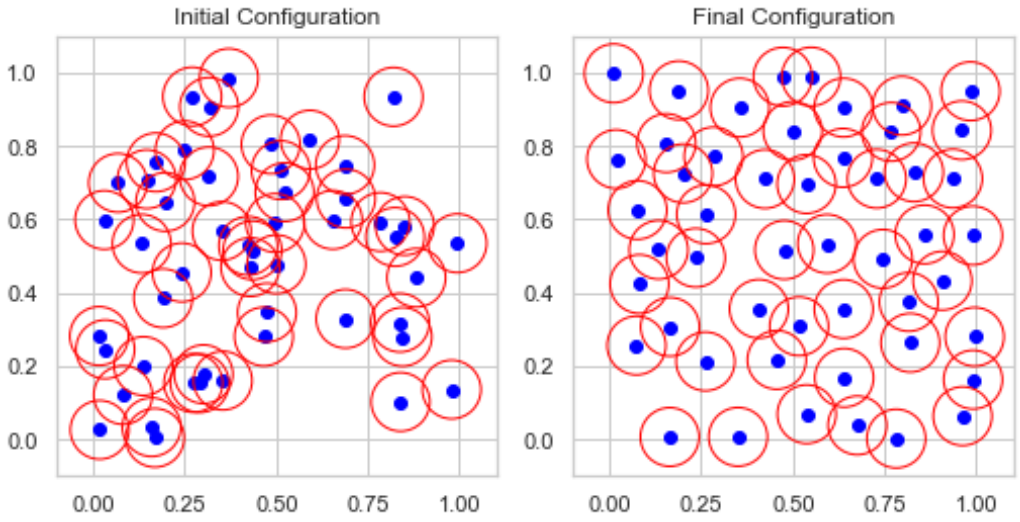


Figure 3.4: Initial configuration of 50 points with area = 0.6631 & final configuration with area = 0.87 ($\gamma = 3$, $r = 0.08$).

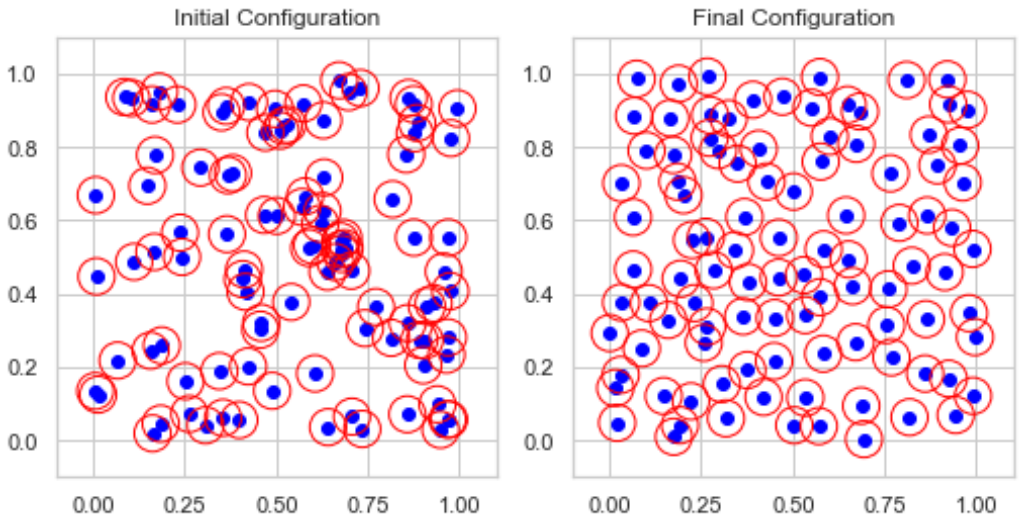


Figure 3.5: Initial configuration of 100 points with area = 0.52 & final configuration with area = 0.69 ($\gamma = 3$, $r = 0.05$).

For 3 dimensions and $N = 1000$ Figure 3.6, Figure 3.7 and Figure 3.8 show the configuration of 25, 50, and 100 points.

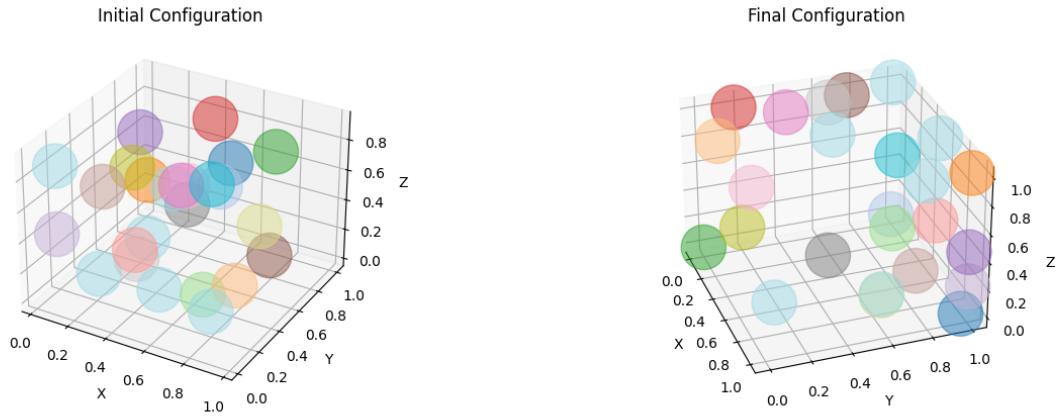


Figure 3.6: Initial configuration of 25 points & final configuration of 3 factors ($\gamma = 2$, $r = 0.1$).

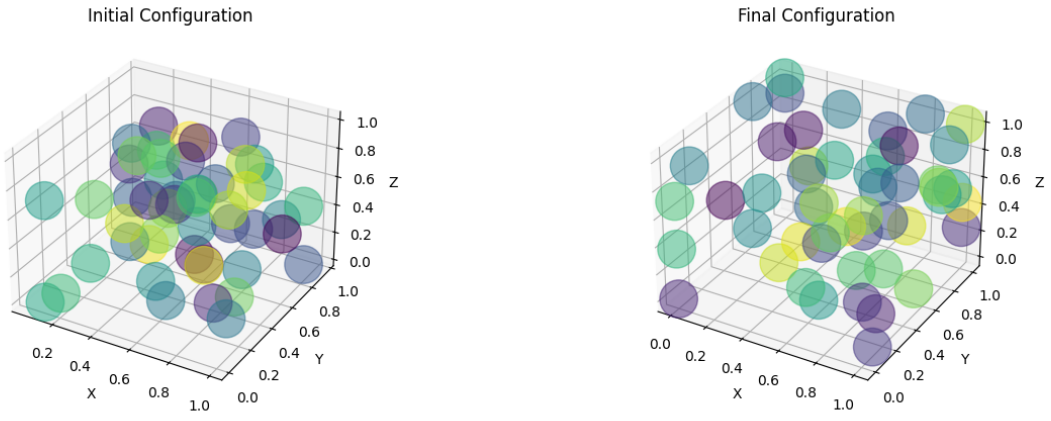


Figure 3.7: Initial configuration of 50 points & final configuration of 3 factors ($\gamma = 2$, $r = 0.07$).

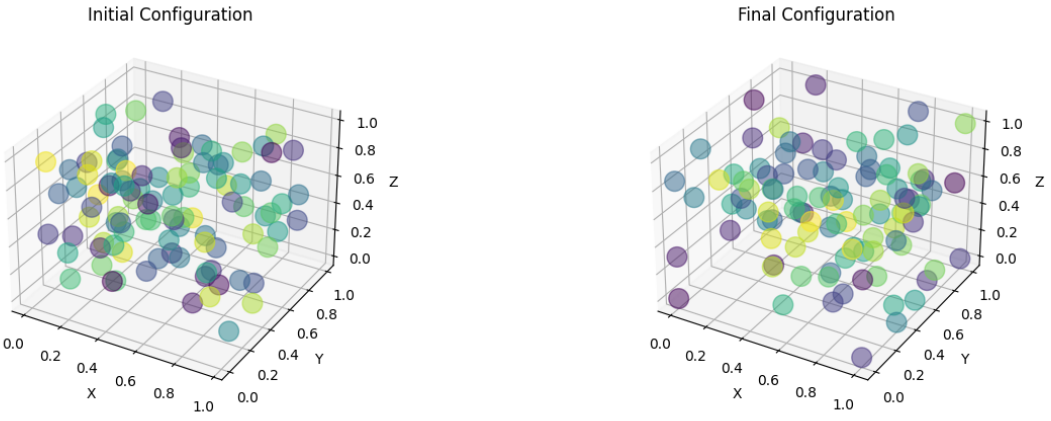


Figure 3.8: Initial configuration of 25 points & final configuration of 3 factors ($\gamma = 2$, $r = 0.02$).

3.4.1. Influence of parameters

The figure above shows the influence of parameter r on the final distribution. The choice of the radius proves to be important. A radius that is too small generates a distribution without interaction. However, a radius that is too large leads to a distribution with clusters.

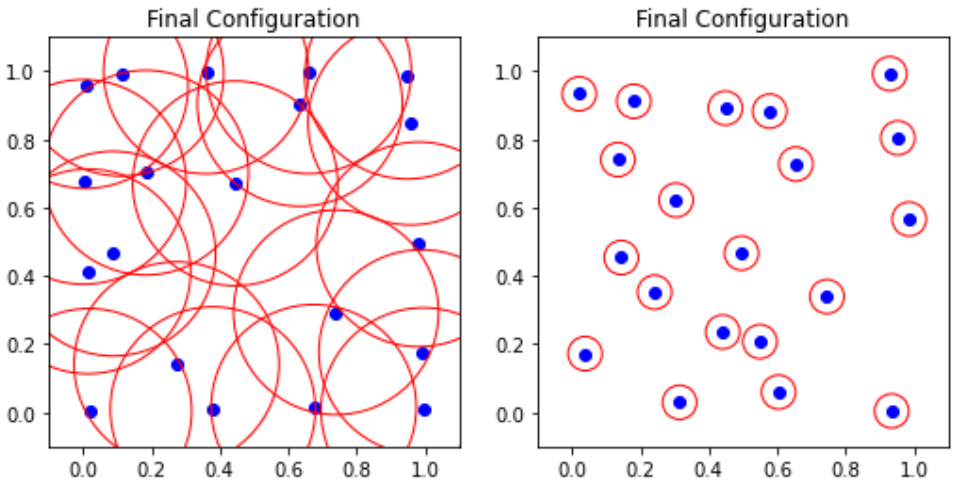


Figure 3.9: Left: $(r = 0.3, \gamma = 3, Area = 2.33)$. Right: $(r = 0.05, \gamma = 3, Area = 0.15)$.

3.5. Convergence study

For each iteration N of the algorithm described above, the chain of experimental designs $(X)_{N\geq 0}$ generated is the realization of a Markov chain with a transition kernel: $P(x, y) = P_{MH}(x, y)$ The essential question is whether the chain converges to the distribution $\pi(x)$ defined in (3.1).

At this level, the fundamental question that arises is whether the chain converges to the distribution $\pi(x)$ defined in (3.1). The chain converges to the invariant distribution π if: $P^t(x, A) \underset{t\to\infty}{\longrightarrow} \pi(A)$. Here, A is a Borel set in $\mathcal{B}$, and $K^t(x, A) = P(X_t = A|X_0 = x)$ is a transition kernel with a step size of t. Let's state the main result of interest here:

Theorem 3.1. If the transition kernel $P = P_{MH}^n$ on a finite space has a unique invariant distribution π, and P is positive recurrent, then the Markov chain $(X_N)_{N\geq 0}$ obtained from the construction algorithm is uniformly ergodic, and this kernel realizes the simulation of a marked Strauss process with density $\pi(x) = \alpha\beta^{n(x)}\gamma^{-m(U_r(x))}$ (i.e., vP^m converges to π as m tends to infinity, where v is the initial distribution).

Proof. Firstly, we show three important properties for the kernel P_{MH}: π-reversibility, π-stationarity, and π-irreducibility.

- **π-reversibility**

  By definition, the transition PMH is π-reversible if:

$$\forall x,\ y\ \in\ \Omega\ :\ \pi(x)\, P_{MH}(x,\ y)\ \ =\ \pi(y)\, P_{MH}(y,\ x)$$

we have:

$$\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) P_{MH}\left(x,y\right) dx = \int_{\Omega} 1_{B(x,y)}\pi\left(x\right) a\left(x,y\right) q\left(x,y\right) dx +$$

$$\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) \left[1 - \int_{\Omega} a\left(x,z\right) q\left(x,z\right) dz\right] \delta_x\left(y\right) dx$$

$$= \int_{\Omega} 1_{B(x,y)}\pi\left(x\right) a\left(x,y\right) q\left(x,y\right) dx + \int_{\Omega} 1_{B(x,x)}\pi\left(x\right) \left[1 - \int_{\Omega} a\left(x,z\right) q\left(x,z\right) dz\right] dx$$

And as :

$$\begin{aligned}
&\pi\left(x\right) a(x,y) q(x,y) = \alpha\beta^{n(x)}\gamma^{-m(U_r(x))} \min\left(1, \beta^{n(y)-n(x_i)}\gamma^{m(U_r(x_i))-m(U_r(y))}\right) q(x,y) \\
&= \alpha \min\left(\beta^{n(x)}\gamma^{-m(U_r(x))}, \beta^{n(y)}\gamma^{-m(U_r(y))}\right) q(x,y) \\
&= \alpha\beta^{n(y)}\gamma^{-m(U_r(y))} \min\left(\gamma^{m(U_r(y))-m(U_r(x))}, 1\right) q(x,y) \\
&= \alpha\beta^{n(y)}\gamma^{-m(U_r(y))} \min\left(1, \gamma^{m(U_r(y))-m(U_r(x))}\right) q(x,y)
\end{aligned}$$

and since $q\left(x,y\right) = q\left(y,x\right)$, we have $\pi\left(x\right) a\left(x,\ y\right) q\left(x,y\right) =\ \pi\left(y\right) a\left(y,x\right) q\left(y,x\right)$,
which gives $\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) P_{MH}\left(x,y\right) dx = \int_{\Omega} 1_{B(x,y)}\pi\left(y\right) P_{MH}\left(y,x\right) dy$
Thus, $\pi\left(x\right) P_{MH}\left(x,\ y\right)\ =\ \pi\left(y\right) P_{MH}\left(y,\ x\right)$,
so the chain is π-reversible.

- **π-stationarity**

  The transition P_{MH} is π-stationary if $\pi P_{MH} = \pi$.
  Let $x \in \Omega$ et $B \in A$. We have:
  $\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) P_{MH}\left(x,y\right) dx = \int_{\Omega} 1_{B(x,y)}\pi\left(x\right) \left[\int_{\Omega} a\left(x,y\right) q\left(x,y\right) dy\right] dx +$
  $\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) \left[\int_{\Omega} 1 - a\left(x,z\right) q\left(x,z\right) dz\right] \delta_x\left(y\right) dx = \int_{\Omega} 1_{B(x,y)}\pi\left(x\right) \left[\int_{\Omega} a\left(x,y\right) q\left(x,y\right) dy\right] dx$
  $+ \int_{\Omega} 1_{B(x,x)}\pi\left(x\right) \left[\int_{\Omega} 1 - a\left(x,z\right) q\left(x,z\right) dz\right] dx = \int_{\Omega}\int_{\Omega} 1_{B(x,y)}\pi\left(x\right) a\left(x,y\right) q\left(x,y\right) dydx +$
  $\int_{\Omega} 1_{B(x,x)}\pi\left(x\right) dx - \int_{\Omega}\int_{\Omega} \pi\left(x\right) a\left(x,z\right) q\left(x,z\right) dzdx = \int_{\Omega} 1_{B(x,x)}\pi\left(x\right) dx$
  Therefore, the chain admits π as a stationary distribution.

- **π-irreducibility**

  The transition P_{MH} is said to be π-irreducible if: $\mathcal{A}$ Let A be a Borel set in Ω, and for $x, y \in \Omega$, we obtain:

$$\int_{\Omega} 1_{B(x,A)} P_{MH}(x,A)\,dx = \int_{\Omega} 1_{B(x,A)} a(x,A)\,q(x,A)\,dx$$
$$+ \int_{\Omega} 1_{B(x,A)} \left[1 - \int_{\Omega} a(x,z)\,q(x,z)\,dz\right] \delta_x(A)\,dx$$
$$\int_{\Omega} 1_{B(x,A)} a(x,A)\,q(x,A)\,dx + \int_{\Omega} 1_{B(x,x)} \left[1 - \int_{\Omega} a(x,z)\,q(x,z)\,dz\right] dx$$
$$\int_{\Omega} 1_{B(x,A)} a(x,A)\,q(x,A)\,dx + 1 - \int_{\Omega}\int_{\Omega} a(x,z)\,q(x,z)\,dzdx$$

Since $a(x,A) = \min\left(1, \gamma^{m(Ur(x))-m(Ur(A))}\right)$ and $a(x,z) = \min\left(1, \gamma^{m(Ur(x))-m(Ur(z))}\right)$, we can distinguish four possible cases:

- If $a(x,A) = 1$ and $a(x,z) = \gamma^{m(U_r(x))-m(U_r(z))}$ so:
$$\int_{\Omega} 1_{B(x,A)} P_{MH}(x,A)\,dx = \int_{\Omega} 1_{B(x,A)} q(x,A)\,dx + 1 - \int_{\Omega}\int_{\Omega} \gamma^{m(U_r(x))-m(U_r(z))} q(x,z)\,dzdx$$
$$= \int_{\Omega} 1_{B(x,A)} q(x,A)\,dx + 1 - \gamma^{m(U_r(x))-m(U_r(z))} > 0$$
- if $a(x,A) = \gamma^{m(U_r(x))-m(U_r(A))}$ and $a(x,z) = 1$ so:
$$\int_{\Omega} 1_{B(x,A)} P_{MH}(x,A)\,dx = \int_{\Omega} 1_{B(x,A)} \gamma^{m(U_r(x))-m(U_r(A))} q(x,A)\,dx + 1 - \int_{\Omega}\int_{\Omega} q(x,z)\,dzdx$$
$$= \gamma^{m(U_r(x))-m(U_r(A))} \int_{\Omega} 1_{B(x,A)} q(x,A)\,dx > 0$$
- if $a(x,A) = \gamma^{m(U_r(x))-m(U_r(A))}$and $a(x,z) = \gamma^{m(U_r(x))-m(U_r(z))}$ so :
$$\int_{\Omega} 1_{B(x,A)} P_{MH}(x,A)\,dx = \int_{\Omega} 1_{B(x,A)} \gamma^{m(U_r(x))-m(U_r(A))} q(x,A)\,dx + 1$$
$$- \int_{\Omega}\int_{\Omega} \gamma^{m(U_r(x))-m(U_r(z))} q(x,z)\,dzdx$$
$$= \gamma^{m(U_r(x))-m(U_r(A))} \int_{\Omega} 1_{B(x,A)} q(x,A)\,dx + 1 - \gamma^{m(U_r(x))-m(U_r(z))} \int_{\Omega}\int_{\Omega} q(x,z)\,dzdx$$
$$= \gamma^{m(U_r(x))-m(U_r(A))} \int_{\Omega} 1_{B(x,A)} q(x,A)\,dx + 1 - \gamma^{m(U_r(x))-m(U_r(z))} > 0$$

Therefore, P_{MH} is π-irreducible.

Therefore, the chain admits π as a stationary distribution. Since π is the invariant measure of P_{MH}, it is also invariant for $P = P^n{}_{MH}$. Indeed, $\pi P_{MH} = \pi$, and by induction on the integer n,

$$\pi P_{MH} = \pi {P_{MH}}^2 = \pi {P_{MH}}^3 = \ldots = \pi {P_{MH}}^n = \pi$$

. Thus, $\pi P = \pi$. By the construction of $P = P^n_{MH}$, the π-irreducibility of P_{MH} implies the

π-irreducibility of P. If P is π-irreducible and has a π-invariant distribution, then P is recurrent positive and π is the unique invariant distribution of P [33]. Let v be an initial distribution, for any integer m and $\forall\ x \in\ N^{lf}$, we have:

$$\|vP^n\,(x,.) - \pi\| = \|vP^n - \pi P^n\| \leq 2C\,(P^n) \leq 2(C\,(P))^n$$

Here, $0 \leq C\,(P) < 1$ (lemma 2.1) is the contraction coefficient therefore, the chain is uniformly ergodic, and $\|vP^m - \pi\|$ tends to zero as m tends to infinity. Thus, the chain converges to the area interaction point processes.

□

Chapter 4

Optimality Criteria And Digital Results

The optimality of a design depends on the statistical model and is assessed with respect to a statistical criterion, which is related to the variance-matrix of the estimator. Specifying an appropriate model and specifying a suitable criterion function both require an understanding of statistical theory and practical knowledge of designs of experiments. In the DOE for estimating statistical models, optimal designs allow parameters to be estimated without bias and with minimum variance. A non-optimal design requires a greater number of experimental runs to estimate the parameters with the same precision as an optimal design. In practical terms, optimal experiments can reduce the costs of experimentation . One of the most important aspects of DOE is the selection of an appropriate optimality criterion. Uniform experimental designs are one kind of space-filling designs that can be used for computer experiments and also for industrial experiments when the uniform distribution is desired

. The uniform experimental design is one important kind of space-filling design and has been applied in computer experiments, laboratory experiments under model uncertainty, and industrial experiments

4.1. Optimality criteria

The evaluation of the quality of a set of points obtained from a database or an experimental design requires the use of quantitative criteria. There are many criteria that allow us to evaluate the quality of an experimental design. For space-filling designs, we seek the uniformity of a point

distribution (such as the discrepancy criterion, distance criterion).

- **Distance criterion (Mindist)**: represents the smallest distance between a pair among a design of n points. This criterion is defined by

$$\text{Mindist} = \min_{1\leq i\leq n} \min_{j\neq i} d\left(x_i, x_j\right)$$

Where $d(x_i, x_j)$ is the Euclidean distance between point x_i and x_j. A higher value of Mindist should correspond to a more regular dispersion of the points in the design.

- **Coverage criterion**: allows us to measure the difference between the points of the design and those of a regular grid. This criterion is zero for a regular grid. The objective is therefore to minimize it to approach a regular grid and thus ensure space-filling, without reaching it to respect a uniform distribution, particularly in projection onto the factorial axes:

$$cov = \frac{1}{\bar{\delta}}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\delta_i - \bar{\delta})^2}$$

Where $\delta_i = \min_{i\neq j} d\left(x_i, x_j\right)$ and $\bar{\delta} = \frac{1}{n}\sum_{i=1}^{n}\delta_i$. For a regular grid, $\delta_1 = \delta_2 = ... = \delta_n$, then $cov = 0$.

In the same context, one can use the ratio R, defined by:

$$R = \frac{\max_{1..n}\delta_i}{\min_{1..n}\delta_i}$$

For a regular grid $R = 1$, thus, the closer R is to 1, the closer the points are to those of a regular grid.

- **Discrepancy criterion (Disc)**: The discrepancy measures the difference between the empirical distribution function of the points on the design and that of the uniform distribution. Unlike the previous two criteria, the discrepancy is not based on the distance between points. There are different measures of discrepancy. We retain the L2-norm discrepancy.

$$Disc = \left(\frac{1}{3}\right)^p - \frac{2^{1-p}}{n}\sum_{i=1}^{n}\prod_{j=1}^{p}\left(1-(x_j^i)^2\right) + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{k=1}^{n}\prod_{j=1}^{p}\left(1-\max(x_i^j, x_k^j)\right)$$

In order to evaluate the quality of our computer experimental design, it is crucial to employ optimality criteria that ensure adequate space coverage and uniform distribution. The purpose

of this section is to compute the values of these criteria for the proposed design To achieve this, we utilize three types of criteria:

- Discrepancy criterion (Disc)
- Distance criterion (Mindist)
- coverage criterion (coverage)

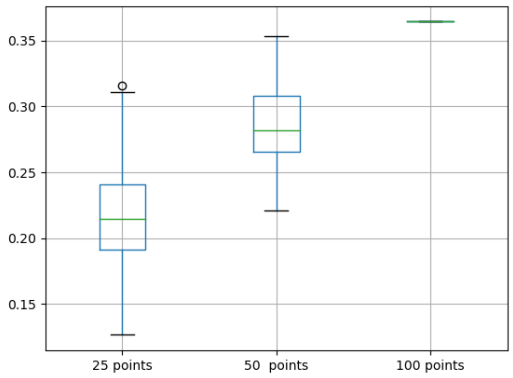
4.2. Intrinsic study of designs using distance, discrepancy, and coverage criteria.

The objective of this section is to evaluate the intrinsic criteria presented in this chapter for the proposed designs. These analyses will allow us to confirm or refute the relationship between the intrinsic properties of these designs and their predictive capacity in applications. We examine three types of criteria: discrepancy, distance, and coverage.

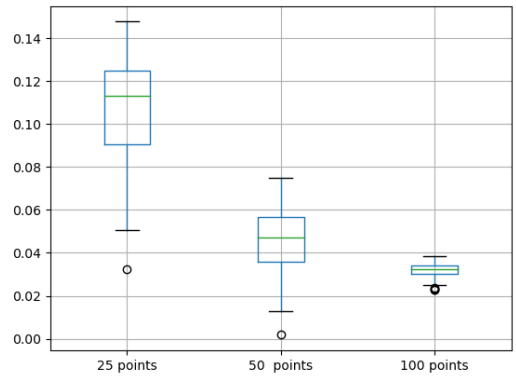
In order to give meaning to the results, the criteria were calculated on a sample of 100 two-dimensional designs.

4.2.1. Designs with 25, 50, and 100 points in 2 dimensions.

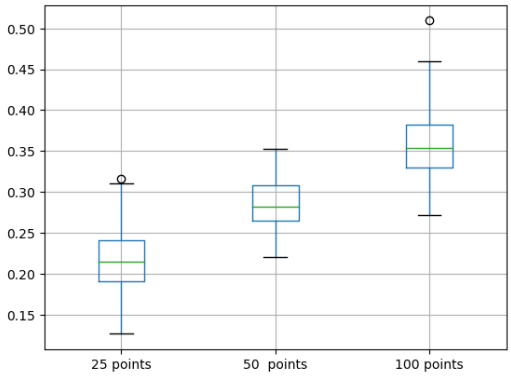
The figures below visually present the most significant criteria that have been calculated. These graphical representations help to better understand and interpret the results, highlighting the distribution and observed variations for each criterion.



(a) Disc



(b) Cov



(c) Mindist

Figure 4.1: Boxplots of the calculated quality criteria for 100 designs in 2 dimensions with 25, 50, and 100 points.

The following table presents a comparison of the mean values of optimality criteria among the proposed designs, calculated for 100 designs in this study, for various point configurations in 2 dimensions:

Table 4.1: Comparison of the means of optimality criteria for different points in 2D

	Disc	Cov	Mindist
25 points	0,21666137	0,107201618	0,217134022
50 points	0,220877652	0,046055877	0,284731164
100 points	0,271505188	0,032026729	0,359933737

4.2.2. Comparison between space-filling techniques and our proposed computer design for 3 factors

The Table presents a comparison based on the discrepancy criterion between our proposed design in this work (referred to as Area interaction) and low-discrepancy sequences (Halton

sequence, Sobol sequence, and Latin hypercube).

Table 4.2: The values of discrepancy for the proposed designs Area Interaction, Halton sequences, MSD, Sobol sequences and Latin hypercube for three dimensions.

Number of points	Area Interaction	MSD	Halton sequence	Sobol sequence	Latin hypercube
25	0,007704809	0,002398653	0,002158069	0,001118936	0,001345872
50	0,005517269	0,00098329	0,00081556	0,000419937	0,000683383
100	0,005714501	0,000589917	0,000178113	0,000112592	0,000330035

The table shows the discrepancy values for each design and the number of points considered. The discrepancy values indicate the uniformity and coverage of the points within the design space. Lower discrepancy values generally suggest better point distributions. By comparing the discrepancy values across the different designs, we can observe differences in their performance. For example, in all cases (25, 50, and 100 points), the Area Interaction design has the highest discrepancy values, indicating potentially less uniform point distribution compared to other designs. On the other hand, Sobol sequences and Latin hypercube designs consistently have the lowest discrepancy values, suggesting better point distributions.

Table 4.3: The values of distance criterion for the proposed designs Area Interaction, Halton sequences, MSD, Sobol sequences and Latin hypercube for three dimensions.

Number of points	Area Interaction	MSD	Halton sequence	Sobol sequence	Latin hypercube
25	0,158064285	0,207993197	0,171961272	0,103644525	0,09797959
50	0,069540641	0,200097292	0,154433996	0,103644525	0,067816526
100	0,028080964	0,033162775	0,054089568	0,040594941	0,040826521

The table displays the values of the distance criterion for each design and the number of points considered. The distance criterion measures the average distance between points in the design. Lower distance criterion values indicate closer proximity between points. By comparing the discrepancy values across the different designs, we can observe differences in their performance. For example, in all cases (25, 50, and 100 points), the Area Interaction design has the highest discrepancy values, indicating potentially less uniform point distribution compared to other designs. On the other hand, Sobol sequences and Latin hypercube designs consistently have the lowest discrepancy values, suggesting better point distributions.

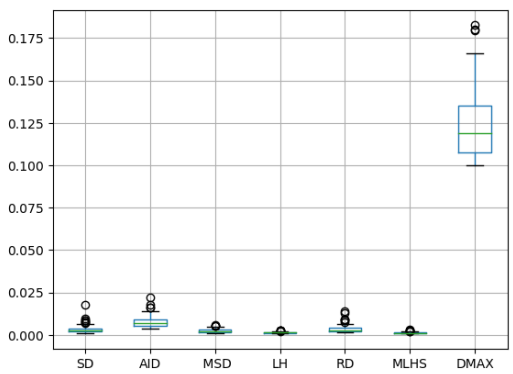
Table 4.4: The value of coverage criterion for the proposed designs Area Interaction, Halton sequences, MSD, Sobol sequences and Latin hypercube for three dimensions.

Number of points	Area Interaction	MSD	Halton sequence	Sobol sequence	Latin hypercube
25	0,347394892	0,199980509	0,275645621	0,410169256	0,43659704
50	0,389000221	0,12933013	0,16954721	0,323542636	0,360887895
100	0,468159086	0,332237677	0,271089016	0,287382553	0,370043181

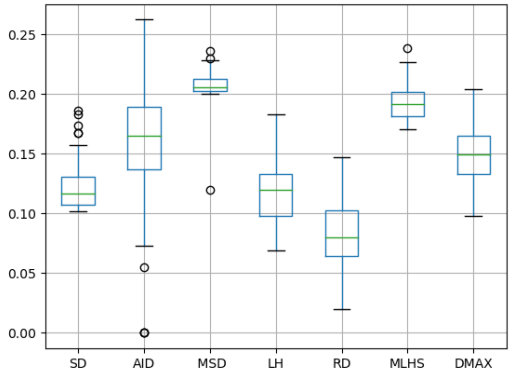
The table displays the values of the coverage criterion for each design and the number of points considered. The coverage criterion measures the extent to which the points cover the design space. Higher coverage criterion values indicate better coverage and distribution of points. As the number of points increases (25, 50 and 100), we can see a trend of increasing coverage criterion values for most designs. This indicates that increasing the number of points generally leads to better coverage of the design space.

The stochastic designs used in the study, were employed in the study of 100 designs to evaluate their performance based on the calculated criteria.

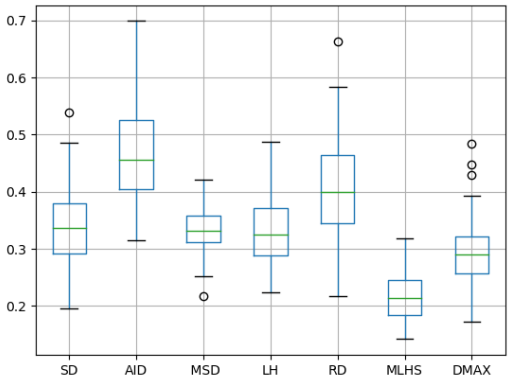
- Random designs (RD)
- Latin Hypercube Sampling (LHS)[25]
- Maximin Latin Hypercube Sampling [37]
- Maximum Entropy Designs (Dmax) [38]
- Strauss designs (SD) [4]
- Marked Strauss designs (MSD) [5]



(a) Disc



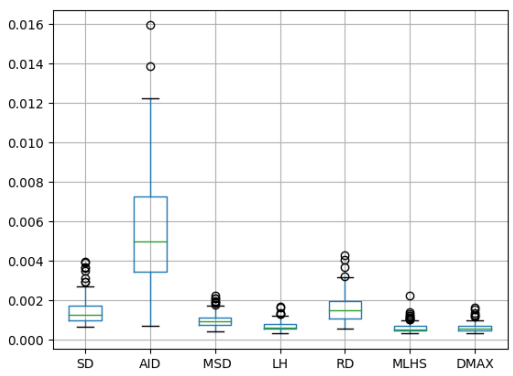
(b) Cov



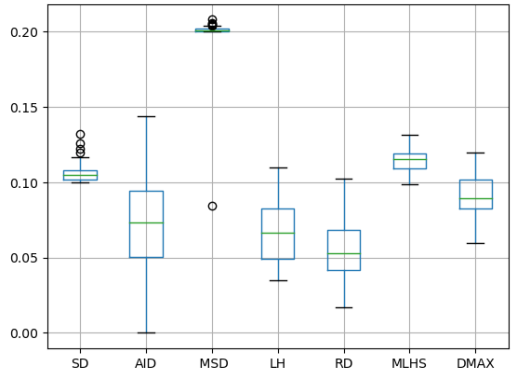
(c) Mindist

Figure 4.2: Boxplots of the calculated quality criteria for 100 designs in 3 dimensions with 25 points.

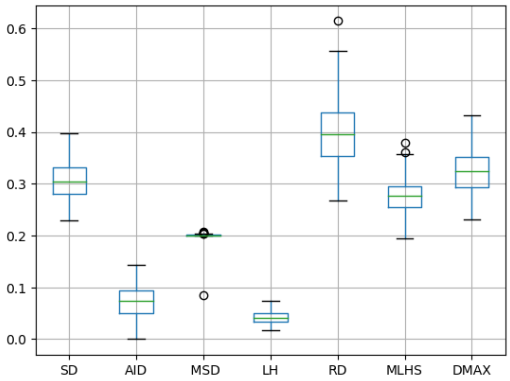
The boxplots provide a visual comparison of our proposed computer experimental design (Area Interaction) with MSD, Latin hypercube, RD, MLHS, DMAX, and SD based on three criteria: discrepancy, distance, and coverage. The median, positioned towards the lower end of the box, suggests a slightly left-skewed distribution. The wide box indicates significant variability and a broad range of values in the data. Additionally, the presence of outliers on the upper end of the plot indicates the existence of extreme values in the dataset. Notably, for 25 points, Area interaction performed very well on the discrepancy criterion.



(a) Disc



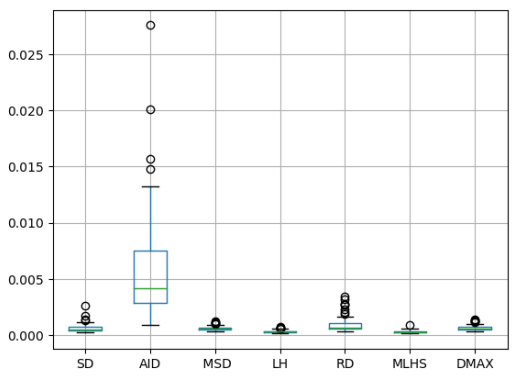
(b) Cov



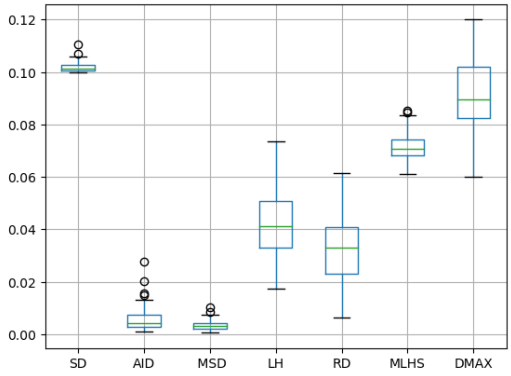
(c) Mindist

Figure 4.3: Boxplots of the calculated quality criteria for 100 designs in 3 dimensions with 50 points.

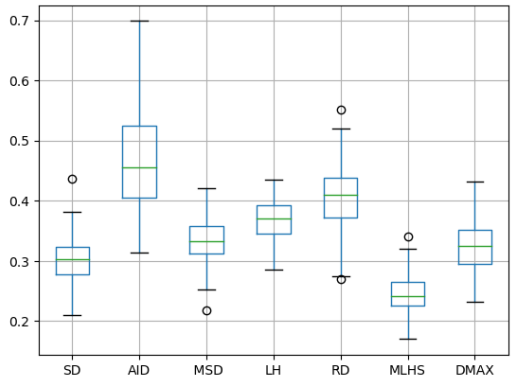
The boxplots provide a comparative analysis of our proposed computer experimental design (Area Interaction) with the other stochastic designs. The wide box in the boxplots indicates a substantial variability and a broad range of values in the data, while also exhibiting some outliers. In terms of coverage criterion, our proposed design (Area Interaction) outperformed nearly all of the stochastic designs except the Latin hypercube for a configuration of 50 points. However, when evaluating the coverage criterion, our design ranked second, trailing behind another approach. Nevertheless, our design closely matched the other designs in terms of discrepancy criterion.



(a) Disc



(b) Cov



(c) MinDist

Figure 4.4: Boxplots of the calculated quality criteria for 100 designs in 3 dimensions with 100 points.

For 100 points our proposed design (Area Interaction) highlights its strengths in terms of coverage criterion while acknowledging its relatively lower performance in the distance criterion and discrepancy. The presence of outliers further underscores the variability in the data.

Conclusion

The methodology of experimental design, combined with the use of area interaction point processes, provides researchers and practitioners with powerful tools for designing optimal experiments, selecting relevant experimental points, and effectively modeling complex phenomena through computer simulations. This approach draws on a rich historical heritage and influences both ancient and modern, demonstrating its relevance and validity in scientific research and industrial studies. Using these methods, it becomes possible to generate patterns with moderate aggregation and organization, which is a considerable advantage for the design of computer experiments. The point process proposal we have presented has proven its effectiveness in a variety of optimality criteria, enabling the selection of the most relevant experimental points for obtaining reliable and accurate results. In addition, computer experimental designs play a crucial role in modeling complex phenomena using computer simulations. They considerably reduce the number of simulations required to obtain reliable results, resulting in significant time and cost savings in the simulation process. They are particularly effective when it comes to solving complex simulation code problems. However, it is important to consider the specific requirements of each situation and choose an appropriate experimental design accordingly, as there is no one-size-fits-all solution. By exploiting these approaches and carefully considering the specific requirements of each situation, researchers can improve the effectiveness and efficiency of their research efforts. Ultimately, experimental design methodology and the use of area interaction point processes offer promising prospects for the future of scientific research and industrial studies, opening up new possibilities for knowledge discovery and innovation.

The use of Area Interaction designs using Voronoi tessellations appears to be promising in computational experimentation. Additionally, we believe that it could be interesting to make improvements by finding ways to simulate designs for 4 Dimensions and higher. Also identifying issues related to points inside the infinite regions of the Voronoi tessellation within a closed hypercube.

Appendix

General functions of the 2D Algorithm

Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
import random
from matplotlib.patches import Polygon
import math
from matplotlib.path import Path
from matplotlib.patches import Circle
from shapely.geometry import Point
from shapely.ops import cascaded_union, unary_union
from matplotlib.path import Path
```

Voronoi Functions

```
def voronoi_finite_polygons_2d(vor, radius=None):
    # Function implementation goes here
    if vor.points.shape[1] != 2:
        raise ValueError("Requires 2D input")

    new_regions = []
    new_vertices = vor.vertices.tolist()

    center = vor.points.mean(axis=0)
    if radius is None:
        radius = vor.points.ptp().max() * 2

    # Construct a map containing all ridges for a given point
```

```
    all_ridges = {}
    for (p1, p2), (v1, v2) in zip(vor.ridge_points, vor.ridge_vertices):
        all_ridges.setdefault(p1, []).append((p2, v1, v2))
        all_ridges.setdefault(p2, []).append((p1, v1, v2))

    # Reconstruct infinite regions
    for p1, region in enumerate(vor.point_region):
        vertices = vor.regions[region]

        if all(v >= 0 for v in vertices):
            # finite region
            new_regions.append(vertices)
            continue

        # reconstruct a non-finite region
        ridges = all_ridges[p1]
        new_region = [v for v in vertices if v >= 0]

        for p2, v1, v2 in ridges:
            if v2 < 0:
                v1, v2 = v2, v1
            if v1 >= 0:
                # finite ridge: already in the region
                continue

            # Compute the missing endpoint of an infinite ridge
            t = vor.points[p2] - vor.points[p1]  # tangent
            t /= np.linalg.norm(t)
            n = np.array([-t[1], t[0]])  # normal

            midpoint = vor.points[[p1, p2]].mean(axis=0)
            direction = np.sign(np.dot(midpoint - center, n)) * n
            far_point = vor.vertices[v2] + direction * radius

            new_region.append(len(new_vertices))
            new_vertices.append(far_point.tolist())

        # sort region counterclockwise
        vs = np.asarray([new_vertices[v] for v in new_region])
        c = vs.mean(axis=0)
        angles = np.arctan2(vs[:, 1] - c[1], vs[:, 0] - c[0])
        new_region = np.array(new_region)[np.argsort(angles)]
```

```
56
57          # finish
58          new_regions.append(new_region.tolist())
59
60      return new_regions, np.asarray(new_vertices)
61
62  def voronoi_function(X):
63      # Create Voronoi tessellation and plot it
64      vor = Voronoi(X)
65
66      # Apply the voronoi_finite_polygons_2d function
67      new_regions, new_vertices = voronoi_finite_polygons_2d(vor)
68      # Iterate over each region and add a point
69      new_points = []
70      for region in new_regions:
71          polygon = [new_vertices[i] for i in region]
72          path = Path(polygon)
73          while True:
74              random_point = np.random.uniform(0, 1, 2)
75              if path.contains_point(random_point):
76                  new_points.append(random_point.tolist())
77                  break
78      # Convert the new points to a NumPy array
79      new_points = np.array(new_points)
80      # Add the new points to the original set of points
81      X = np.vstack((X, new_points))
82
83      # Randomly select and remove one of two given points with 50% probability
84      while len(X) > n:
85          indices = np.arange(len(X))
86          chosen_points = set()
87          i = np.random.choice(indices)
88          if i in chosen_points:
89              continue
90          chosen_points.add(i)
91          if i >= len(X):
92              continue
93          distances = np.linalg.norm(X - X[i], axis=1)
94          distances[i] = np.inf
95          min_distance_index = np.argmin(distances)
96          indices = np.delete(indices, min_distance_index)
97          X = np.delete(X, min_distance_index, axis=0)
```

```
98
99      return X
```

Area Interaction Functions

```
def calculate_overlap(points , radius):

    circles = [Point(point).buffer(radius) for point in points]
    union = unary_union(circles)
    total_area = union.area
    return total_area
```

Metropolis-Hasting Algorithm and plot of the Initial and Final configurations

```
n = 25          #Number of points
N = 10000       #Number of iterations of MH
r = 0.095       #Radius
gamma =1.5

X = np.random.uniform(size=(n, 2))
X_intial = X.copy()
X_final = X.copy()


    # Plot the initial configuration
fig1, ax1 = plt.subplots()
ax1.scatter(X[:, 0], X[:, 1], color='blue', label='Initial Configuration')

    # Draw circles on the initial points
for point in X:
    circle = Circle((point[0], point[1]), r, color='red', fill=False)
    ax1.add_patch(circle)

ax1.set_aspect('equal')
ax1.set_title('Initial Configuration')

# Set x and y limits
ax1.set_xlim([-0.01, 1.01])
ax1.set_ylim([-0.01, 1.01])

```

```
# Calculate and print the initial area
area = calculate_overlap(X, r)

print("The initial area is:", area)
for iteration in range(N):
    X_intial = X
    area = calculate_overlap(X, r)
     #print("Iteration:", iteration + 1)
    # Perform the voronoi_function
    X = voronoi_function(X)

    # Calculate and print the final area
    area1 = calculate_overlap(X, r)

    if gamma >1:
       # Acceptance probability
        a = min(1, gamma ** (area1 - area))
      # print("The probability of acceptance is:", a)

    else:
        a = min(1, gamma ** (area - area1))

    if a == 1:
        #print("We accept the final Configuration")
        X_final = X
       # print("the accepeted area is:",area1)
    else:
        #print("Playing Daft Punk: ONE MORE TIME")
        X_final = X_intial
       # print("the accepeted area is :",area)

    #print('a1=',area1,'a0=',area)
    X = X_final
    area_final = calculate_overlap(X,r)


print("The final area is:", area_final)
# Plot the final configuration
fig2, ax2 = plt.subplots()
ax2.scatter(X[:, 0], X[:, 1], color='blue', label='Final Configuration')

# Draw circles on the final points
```

```
for point in X:
    circle = Circle((point[0], point[1]), r, color='red', fill=False)
    ax2.add_patch(circle)

ax2.set_aspect('equal')
ax2.set_title('Final Configuration')

# Set x and y limits
ax2.set_xlim([-0.01, 1.01])
ax2.set_ylim([-0.01, 1.01])

plt.show()
```

Distance Criterion

```

def mindist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    y = np.min(np.min(M))
    return y
```

Discrepancy Criterion

```

def dsc(x):
    n = x.shape[0]  # Nombre de points dans x
```

```
    w = x.shape[1]  # Dimension de x
    s1 = 0

    for i in range(n):
        p1 = 1
        for j in range(w):
            p1 *= (1 - x[i, j]) * (1 + x[i, j])
        s1 += p1

    s2 = 0
    for i in range(n):
        for j in range(n):
            p2 = 1
            for k in range(w):
                m = max(x[i, k], x[j, k])
                p2 *= (1 - m)
            s2 += p2

    y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n * 2 ** (w - 1))) * s1
    return y
```

Coverage Criterion

```

def mdist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    t = np.zeros(n)
```

```
    for i in range(n):
        t[i] = np.min(M[i, :])

    q = np.sum(t)
    q1 = q / n
    lamda = 0
    for i in range(n):
        lamda += (t[i] - q1) ** 2

    y = (1 / q1) * ((1 / n) * lamda) ** 0.5
    return y
```

Exporting distributions to Excel files

```

n = 100
N = 100
r = 0.02
gamma =2

# Execute the code N times
results_a = []
results_b = []
results_c= []
for _ in range(100):
    #Generation of an initial configuration of random points in a unit
       hypercube.
    dim = 2  #Dimensions of the hypercube.
    X = np.random.rand(n, dim)
    area = calculate_overlap(X, r)
    print("The initial area is:", area)
    # Simulation of NMC steps
    for iteration in range(N):
        X_intial = X
        area = calculate_overlap(X, r)
        X = voronoi_function(X)

    # Calculate and print the final area
        area1 = calculate_overlap(X, r)

        if gamma >1:
```

```
        # Acceptance probability
            a = min(1, gamma ** (area1 - area))

        else:
            a = min(1, gamma ** (area - area1))

        if a == 1:
        )
            X_final = X

        else:

            X_final = X_intial

        X = X_final
        area_final = calculate_overlap(X,r)


    print("The final area is:", area_final)

   # Calcul de la Value de a et b
    a = mdist(X)
    b = mindist(X)
    c = dsc(X)
    results_a.append(a)
    results_b.append(b)
    results_c.append(c)

# Creat a DataFrame for each result
df_a = pd.DataFrame({'Value': results_a})
df_b = pd.DataFrame({'Value': results_b})
df_c = pd.DataFrame({'Value': results_c})
# Export the DataFrames to separate Excel files.
file_name_a = 'mdist7dim_35p.xlsx'
file_name_b = 'mindist7dim_35p.xlsx'
file_name_c = 'disc7dim_35p .xlsx'
df_a.to_excel(file_name_a, index=False)
df_b.to_excel(file_name_b, index=False)
df_b.to_excel(file_name_c, index=False)

print(f'Result of "a" exported to {file_name_a}')
print(f'Result of  "b" exported to {file_name_b}')
```

```
print(f'Result of  "c" exported to {file_name_c}')
```

Box plot of discrepancy

```

import pandas as pd
import matplotlib.pyplot as plt

# List of paths for the Excel files to import
file_paths = [r'C:\Users\StarTech\Desktop\disc2dim_25p .xlsx',
              r'C:\Users\StarTech\Desktop\disc2dim_50p .xlsx',
              r'C:\Users\StarTech\Desktop\disc2dim_100p .xlsx']

# Create a list to store the data from the Excel files
data = []

# Import the Excel data into the 'data' list
for path in file_paths:
    data.append(pd.read_excel(path))

# Create a figure and an axis for the box plot
fig, ax = plt.subplots()

# Plot a box plot for each dataset
for i, dataset in enumerate(data):
    ax.boxplot(dataset.values, positions=[i+1])

# Set the x-axis labels
ax.set_xticks(range(1, len(file_paths)+1))
ax.set_xticklabels(['25 points', '50 points', '100 points'], rotation=45)

# Add a title to the graph
plt.title('Disc')

# Display the graph
plt.show()
```

Box plot of mdist

```

```

```
import pandas as pd
import matplotlib.pyplot as plt

# List of paths for the Excel files to import
file_paths = [r'C:\Users\StarTech\Desktop\mdist2dim_25p.xlsx',
              r'C:\Users\StarTech\Desktop\mdist2dim_50p.xlsx',
              r'C:\Users\StarTech\Desktop\mdist2dim_100p.xlsx']

# Create a list to store the data from the Excel files
data = []

# Import the Excel data into the 'data' list
for path in file_paths:
    data.append(pd.read_excel(path))

# Create a figure and an axis for the box plot
fig, ax = plt.subplots()

# Plot a box plot for each dataset
for i, dataset in enumerate(data):
    ax.boxplot(dataset.values, positions=[i+1])

# Set the x-axis labels
ax.set_xticks(range(1, len(file_paths)+1))
ax.set_xticklabels(['25 points', '50 points', '100 points'], rotation=45)

# Add a title to the graph
plt.title('Mdist')

# Display the graph
plt.show()
```

Mindist box plot

```

import pandas as pd
import matplotlib.pyplot as plt

# List of paths for the Excel files to import
file_paths = [r'C:\Users\StarTech\Desktop\mindist2dim_25p.xlsx',
              r'C:\Users\StarTech\Desktop\mindist2dim_50p.xlsx',
```

```
                r'C:\Users\StarTech\Desktop\mindist2dim_100p.xlsx']

# Create a list to store the data from the Excel files
data = []

# Import the Excel data into the 'data' list
for path in file_paths:
    data.append(pd.read_excel(path))

# Create a figure and an axis for the box plot
fig, ax = plt.subplots()

# Plot a box plot for each dataset
for i, dataset in enumerate(data):
    ax.boxplot(dataset.values, positions=[i+1])

# Set the x-axis labels
ax.set_xticks(range(1, len(file_paths)+1))
ax.set_xticklabels(['25 points', '50 points', '100 points'], rotation=45)

# Add a title to the graph
plt.title('Mindist')

# Display the graph
plt.show()
```

General functions of the 3D Algorithm

Libraries

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi
import random
import pandas as pd
```

Voronoi Function

```
def voronoi_function(points):
    vor = Voronoi(points)
```

```
    infinite_regions = [region for region in vor.regions if -1 in region]
    bounded_regions = []

    for region in infinite_regions:
        region_points = vor.vertices[region]
        min_bound = np.min(region_points, axis=0)
        max_bound = np.max(region_points, axis=0)
        bounded_regions.append((min_bound, max_bound))

    new_points = []
    for bounds in bounded_regions:
        min_bound, max_bound = bounds
        random_point = np.random.uniform(low=min_bound, high=max_bound)
        new_points.append(random_point)
        '''
        while True:
            random_point = np.random.uniform(low=min_bound, high=max_bound)
            if np.all(random_point >= 0) and np.all(random_point <= 1):
                break
        new_points.append(random_point)
        '''
    finite_regions = [region for region in vor.regions if region and -1 not in
       region]

    for region in finite_regions:
        region_points = vor.vertices[region]
        min_bound = np.min(region_points, axis=0)
        max_bound = np.max(region_points, axis=0)
        random_point = np.random.uniform(low=min_bound, high=max_bound)
        new_points.append(random_point)
        '''
        random_point = None

        while True:
            random_point = np.random.uniform(low=min_bound, high=max_bound)

            if np.all(random_point >= 0) and np.all(random_point <= 1):
                break
        new_points.append(random_point)
        '''
    all_points = np.vstack((points, new_points))
```

```
44
45      X = np.clip(all_points, 0, 1)
46
47      while len(X) > n:
48          indices = np.arange(len(X))
49          chosen_points = set()
50          i = np.random.choice(indices)
51          if i in chosen_points:
52              continue
53          chosen_points.add(i)
54          if i >= len(X):
55              continue
56          distances = np.linalg.norm(X - X[i], axis=1)
57          distances[i] = np.inf
58          min_distance_index = np.argmin(distances)
59          indices = np.delete(indices, min_distance_index)
60          X = np.delete(X, min_distance_index, axis=0)
61
62      return X
```

Calculating Volume Interaction

```
1  def calculate_union_volume(points, radius, num_samples=1000000):
2      min_point = np.min(points, axis=0) - radius
3      max_point = np.max(points, axis=0) + radius
4
5      random_points = np.random.uniform(min_point, max_point, size=(num_samples,
           3))
6      distances = np.linalg.norm(random_points[:, np.newaxis] - points, axis=-1)
7      total_volume = np.sum(np.any(distances <= radius, axis=-1))
8
9      bounding_box_volume = np.prod(max_point - min_point)
10     volume_fraction = total_volume / num_samples
11     total_volume = volume_fraction * bounding_box_volume
12
13     return total_volume
```

Coverage Criterion Function

```
1
```

```
def mdist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    y = np.min(np.min(M))
    return y
```

Discrepancy Criterion Function

```
def dsc(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    s1 = 0

    for i in range(n):
        p1 = 1
        for j in range(w):
            p1 *= (1 - x[i, j]) * (1 + x[i, j])
        s1 += p1

    s2 = 0
    for i in range(n):
        for j in range(n):
            p2 = 1
            for k in range(w):
                m = max(x[i, k], x[j, k])
                p2 *= (1 - m)
            s2 += p2

```

```
    y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n * 2 ** (w - 1))) * s1
    return y
```

Distance Criterion Function

```
def mindist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    t = np.zeros(n)
    for i in range(n):
        t[i] = np.min(M[i, :])

    q = np.sum(t)
    q1 = q / n
    lamda = 0
    for i in range(n):
        lamda += (t[i] - q1) ** 2

    y = (1 / q1) * ((1 / n) * lamda) ** 0.5
    return y
```

Metropolis-Hasting Algorithm and exporting the Distributions

```
n = 25
N = 1000
r = 0.1
gamma =2
```

```
# Create empty DataFrames to store values
df_a = pd.DataFrame(columns=['Value'])
df_b = pd.DataFrame(columns=['Value'])
df_c = pd.DataFrame(columns=['Value'])
# Execute the code 20 times
'''results_a = []
results_b = []
results_c= []'''
for _ in range(100):
    #Generating initial configuration in the unit cube
    X = np.random.uniform(size=(n, 3))
    X_initial = X.copy()
    X_final = X.copy()

    area_intial = calculate_union_volume(X, r)
    print("The initial area is:", area_intial)
    # Simulation of NMC steps
    for iteration in range(N):
        X_intial = X
        area = calculate_union_volume(X, r)
        print("Iteration:", iteration + 1)
        # Perform the voronoi_function
        X = voronoi_function(X)

    # Calculate and print the final area
        area1 = calculate_union_volume(X, r)

        if gamma >1:
        # Acceptance probability
            a = min(1, gamma ** (area1 - area))
        # print("The probability of acceptance is:", a)
# ================================================================================
        else:
            a = min(1, gamma ** (area - area1))
#
# ================================================================================
        if a == 1:
        #print("We accept the final Configuration")
            X_final = X
            area = area1
       # print("the accepeted area is:",area1)
        else:
```

```
        #print("Playing Daft Punk: ONE MORE TIME")
            X_final = X_intial
       # print("the accepeted area is :",area)

    #print('a1=',area1,'a0=',area)
        X = X_final


    area_final = calculate_union_volume(X,r)
    print("The final area is:", area_final)

   # Calculate the values of a,b and c
    a = mdist(X)
    b = mindist(X)
    c = dsc(X)
    df_a = pd.concat([df_a, pd.DataFrame({'Value': [a]})], ignore_index=True)
    df_b = pd.concat([df_b, pd.DataFrame({'Value': [b]})], ignore_index=True)
    df_c = pd.concat([df_c, pd.DataFrame({'Value': [c]})], ignore_index=True)

# Export the DataFrames to the same Excel file
    file_name_a = 'mdist3dim_25p.xlsx'
    file_name_b = 'mindist3dim_25p.xlsx'
    file_name_c = 'disc3dim_25p.xlsx'
    df_a.to_excel(file_name_a, index=False)
    df_b.to_excel(file_name_b, index=False)
    df_c.to_excel(file_name_c, index=False)

    print(f'Result of  "a" Exported to {file_name_a}')
    print(f'Result of  "b" Exported to {file_name_b}')
    print(f'Result of  "c" Exported to {file_name_c}')

    results_a.append(a)
    results_b.append(b)
    results_c.append(c)

#Creat a DataFrame for each result
df_a = pd.DataFrame({'Value': results_a})
df_b = pd.DataFrame({'Value': results_b})
df_c = pd.DataFrame({'Value': results_c})
# Export DataFrames to separated Excel files
file_name_a = 'mdist7dim_35p.xlsx'
file_name_b = 'mindist7dim_35p.xlsx'
```

```
file_name_c = 'disc7dim_35p .xlsx'
df_a.to_excel(file_name_a, index=False)
df_b.to_excel(file_name_b, index=False)
df_b.to_excel(file_name_c, index=False)

print(f'Result of  "a" Exported to {file_name_a}')
print(f'Result of  "b" Exported to {file_name_b}')
print(f'Result of  "b" Exported to {file_name_c}')
```

Data of the Three Optimality Criteria for Used Designs

RD, MLHS, LATIN HYPERCUBE, D-MAX, SD

```
import numpy as np
import random
import pandas as pd
from scipy.stats import qmc
from pyDOE import lhs
from scipy.stats import special_ortho_group

def mdist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    y = np.min(np.min(M))
    return y

def dsc(x):
```

```
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    s1 = 0

    for i in range(n):
        p1 = 1
        for j in range(w):
            p1 *= (1 - x[i, j]) * (1 + x[i, j])
        s1 += p1

    s2 = 0
    for i in range(n):
        for j in range(n):
            p2 = 1
            for k in range(w):
                m = max(x[i, k], x[j, k])
                p2 *= (1 - m)
            s2 += p2

    y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n * 2 ** (w - 1))) * s1
    return y

def mindist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    t = np.zeros(n)
    for i in range(n):
        t[i] = np.min(M[i, :])

```

```
    q = np.sum(t)
    q1 = q / n
    lamda = 0
    for i in range(n):
        lamda += (t[i] - q1) ** 2

    y = (1 / q1) * ((1 / n) * lamda) ** 0.5
    return y
def generate_maximal_entropy_matrix(n, d):
    # Generate random orthogonal matrix
    matrix = lhs(d,samples=n,criterion='centermaximin')
    return matrix.tolist()

    # Generate uniform random numbers
  #  uniform_matrix = np.random.rand(n, d)

    # Transform uniform matrix using orthogonal matrix
  #  matrix = np.matmul(uniform_matrix, ortho_matrix)

    return matrix.tolist()

def generate_random_matrix(n, d):
    matrix = [[random.random() for _ in range(d)] for _ in range(n)]
    return matrix

def generate_maximin_lhs_matrix(n, d):
    matrix = lhs(d, samples=n, criterion='maximin', iterations=1000)
    return matrix.tolist()
def generate_strauss_matrix(n, d, r):
    matrix = np.zeros((n, d))
    matrix[0] = np.random.rand(d)

    for i in range(1, n):
        candidate = np.random.rand(d)
        distances = np.linalg.norm(matrix[:i] - candidate, axis=1)
        while np.any(distances < r):
            candidate = np.random.rand(d)
            distances = np.linalg.norm(matrix[:i] - candidate, axis=1)
        matrix[i] = candidate

    return matrix.tolist()

```

```
112
113
114
115
116 n = 100
117 d = 3
118
119
120 # Create empty DataFrames to store values
121 df = pd.DataFrame(columns=[])
122
123 for _ in range(100):
124     #RD
125     rd_matrix = np.array(generate_random_matrix(n, d))
126
127     # LatinHyperCube
128     latiner = qmc.LatinHypercube(d, scramble=False)
129     latin = latiner.random(n)
130     #mLHC
131     mlhs_matrix = np.array(generate_maximin_lhs_matrix(n,d))
132     #Dmax
133     dmax_matrix =np.array(generate_maximal_entropy_matrix(n, d))
134     #SD
135     sd_matrix = np.array(generate_strauss_matrix(n, d, r=0.1))
136
137     # Calcul de la valeur de a et b
138     a = mdist(rd_matrix)
139     b = mindist(rd_matrix)
140     c = dsc(rd_matrix)
141
142     a1 = mdist(mlhs_matrix)
143     b1 = mindist(mlhs_matrix)
144     c1 = dsc(mlhs_matrix)
145
146     a2 = mdist(latin)
147     b2 = mindist(latin)
148     c2 = dsc(latin)
149
150     a3 = mdist(dmax_matrix)
151     b3 = mindist(dmax_matrix)
152     c3 = dsc(dmax_matrix)
153
```

```
    a4 = mdist(sd_matrix)
    b4 = mindist(sd_matrix)
    c4 = dsc(sd_matrix)

    # Ajouter les valeurs à la DataFrame
    new_row = {'mdist(RD)': a, 'mindist(RD)': b, 'dsc(RD)': c, 'mdist(
        mlhs_matrix)': a1, 'mindist(mlhs_matrix)': b1, 'dsc(mlhs_matrix)': c1,
        'mdist(latin)': a2, 'mindist(latin)': b2, 'dsc(latin)': c2,'mdist(
        dmax_matrix)': a3, 'mindist(dmax_matrix)': b3, 'dsc(dmax_matrix)': c3,'
        mdist(sd_matrix)': a4, 'mindist(sd_matrix)': b4, 'dsc(sd_matrix)': c4}
    df = pd.concat([df, pd.DataFrame(new_row, index=[0])], ignore_index=True)
# Export the DataFrame to an Excel file
nom_fichier = 'resultats100p.xlsx'
df.to_excel(nom_fichier, index=False)
print(f"Les résultats ont été exportés vers {nom_fichier}")
```

MSD

```
import numpy as np

import pandas as pd


def mdist(x):
    n = x.shape[0]  # Nombre de points dans x
    w = x.shape[1]  # Dimension de x
    M = np.zeros((n, n))

    for i in range(n - 1):
        for k in range(i + 1, n):
            s = 0
            for j in range(w):
                s += (x[i, j] - x[k, j]) ** 2
            M[i, k] = np.sqrt(s)
            M[k, i] = np.sqrt(s)

    for i in range(n):
        M[i, i] = np.inf

    y = np.min(np.min(M))
    return y
```

```
24
25 def dsc(x):
26     n = x.shape[0]  # Nombre de points dans x
27     w = x.shape[1]  # Dimension de x
28     s1 = 0
29
30     for i in range(n):
31         p1 = 1
32         for j in range(w):
33             p1 *= (1 - x[i, j]) * (1 + x[i, j])
34         s1 += p1
35
36     s2 = 0
37     for i in range(n):
38         for j in range(n):
39             p2 = 1
40             for k in range(w):
41                 m = max(x[i, k], x[j, k])
42                 p2 *= (1 - m)
43             s2 += p2
44
45     y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n * 2 ** (w - 1))) * s1
46     return y
47
48 def mindist(x):
49     n = x.shape[0]  # Nombre de points dans x
50     w = x.shape[1]  # Dimension de x
51     M = np.zeros((n, n))
52
53     for i in range(n - 1):
54         for k in range(i + 1, n):
55             s = 0
56             for j in range(w):
57                 s += (x[i, j] - x[k, j]) ** 2
58             M[i, k] = np.sqrt(s)
59             M[k, i] = np.sqrt(s)
60
61     for i in range(n):
62         M[i, i] = np.inf
63
64     t = np.zeros(n)
65     for i in range(n):
```

```
        t[i] = np.min(M[i, :])

    q = np.sum(t)
    q1 = q / n
    lamda = 0
    for i in range(n):
        lamda += (t[i] - q1) ** 2

    y = (1 / q1) * ((1 / n) * lamda) ** 0.5
    return y



n = 100
R = 0.2
t = 0.1
b = 2
w = 3
eps = 0.04
NMC = 1000

# Create empty DataFrames to store values
df_a = pd.DataFrame(columns=['Valeur'])
df_b = pd.DataFrame(columns=['Valeur'])
df_c = pd.DataFrame(columns=['Valeur'])
# Execute the code 20 times
'''results_a = []
results_b = []
results_c= []'''
for _ in range(100):
    # initialisation de la configuration
    X = np.random.rand(n, w)
    print("Iteration:", _ + 1)
    for N in range(NMC):
        # choix d'un point au hasard
        k = np.random.randint(n)
        # création d'une nouvelle configuration Y
        y = np.random.rand(1, w)
        Y = np.copy(X)
        Y[k] = y

        # calcul du nombre d'interactions pour X et Y
```

```
        Sx, Sy = 0, 0
        for i in range(n):
            if i != k:
                dx = np.linalg.norm(X[i] - X[k])
                dy = np.linalg.norm(Y[i] - y)
                if dx <= R:
                    Sx += 1
                if dy <= R:
                    Sy += 1

        # calcul des marques pour X et Y
        somme = 0
        for i in range(n):
            m1 = np.sum(X[i] * np.linalg.inv(np.matmul(X.T, X)) * X[i])
            if m1 <= eps:
                somme += 1




        # calcul de la probabilité d'acceptation
        f1 = b ** somme
        by = t ** Sy * f1
        bx = t ** Sx * b ** somme
        a = min(1, by / bx)

        # mise à jour de la configuration
        if a == 1:
            X[k] = y

   # Calcul de la valeur de a et b
    a = mdist(X)
    b = mindist(X)
    c = dsc(X)
    df_a = pd.concat([df_a, pd.DataFrame({'Valeur': [a]})], ignore_index=True)
    df_b = pd.concat([df_b, pd.DataFrame({'Valeur': [b]})], ignore_index=True)
    df_c = pd.concat([df_c, pd.DataFrame({'Valeur': [c]})], ignore_index=True)

# Export the DataFrames to the same Excel file
    nom_fichier_a = 'mdist3dim_100p.xlsx'
    nom_fichier_b = 'mindist3dim_100p.xlsx'
    nom_fichier_c = 'disc3dim_100p.xlsx'
    df_a.to_excel(nom_fichier_a, index=False)
```

```
    df_b.to_excel(nom_fichier_b, index=False)
    df_c.to_excel(nom_fichier_c, index=False)

    print(f'Résultats de "a" exportés vers {nom_fichier_a}')
    print(f'Résultats de "b" exportés vers {nom_fichier_b}')
    print(f'Résultats de "c" exportés vers {nom_fichier_c}')

    '''results_a.append(a)
    results_b.append(b)
    results_c.append(c)

# Créer un DataFrame pour chaque résultat
df_a = pd.DataFrame({'Valeur': results_a})
df_b = pd.DataFrame({'Valeur': results_b})
df_c = pd.DataFrame({'Valeur': results_c})
# Exporter les DataFrames vers des fichiers Excel séparés
nom_fichier_a = 'mdist7dim_35p.xlsx'
nom_fichier_b = 'mindist7dim_35p.xlsx'
nom_fichier_c = 'disc7dim_35p .xlsx'
df_a.to_excel(nom_fichier_a, index=False)
df_b.to_excel(nom_fichier_b, index=False)
df_b.to_excel(nom_fichier_c, index=False)

print(f'Résultats de "a" exportés vers {nom_fichier_a}')
print(f'Résultats de "b" exportés vers {nom_fichier_b}')
print(f'Résultats de "b" exportés vers {nom_fichier_c}')'''
```

Bibliography

[1] J. F. Box, "Ra fisher and the design of experiments, 1922–1926," *The American Statistician*, vol. 34, no. 1, pp. 1–7, 1980.

[2] J. Kiefer, "Optimum experimental designs," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 21, no. 2, pp. 272–304, 1959.

[3] G. E. Box and D. W. Behnken, "Some new three level designs for the study of quantitative variables," *Technometrics*, vol. 2, no. 4, pp. 455–475, 1960.

[4] J. Franco, *Planification d'expériences numériques en phase exploratoire pour la simulation des phénomènes complexes*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2008.

[5] H. Elmossaoui, N. Oukid, and F. Hannane, "Construction of computer experiment designs using marked point processes," *Afrika Matematika*, vol. 31, pp. 917–928, 2020.

[6] H. Elmossaoui, *Contribution à la méthodologie de la Recherche expérimentale*. PhD thesis, Université Blida 1, 2020.

[7] H. Elmossaoui, N. Oukid, "Construction de plans d'expériences numérique à partir des processus ponctuels," *Algerian Journal of Engineering, Architecture and Urbanism*, pp. 273–282, 2021.

[8] D. J. Strauss, "A model for clustering," *Biometrika*, vol. 62, no. 2, pp. 467–475, 1975.

[9] H. Elmossaoui and N. Oukid, "New computer experiment designs using continuum random cluster point process," *International Journal of Analysis and Applications*, vol. 21, pp. 51–51, 2023.

[10] R. Sibson, "The dirichlet tessellation as an aid in data analysis," *Scandinavian Journal of Statistics*, pp. 14–20, 1980.

[11] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[12] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

[13] P. Schimmerling, J. C. Sisson, and A. Zaïdi, *Pratique des plans d'expériences*. TEC & DOC, 1998.

[14] W. Tinsson, *Plans d'expérience: constructions et analyses statistiques*, vol. 67. Springer Science & Business Media, 2010.

[15] G. Taguchi and Y. Wu, "Introduction to off-line quality control (nagoya, central japan quality control association)," *CRITICAL CORE SAND MIX PARAMETERS*, 1980.

[16] J. Goupy, *plans d'expérience: Les mélanges*. Dunod, 2000.

[17] S. I. Loeza-Serrano, *Optimal statistical design for variance components in multistage variability models*. The University of Manchester (United Kingdom), 2014.

[18] J.-P. Gauchi, "Plans d'expériences optimaux pour modèles non linéaires," 1997.

[19] J. Goupy, "Modélisation par les plans d'expériences," *Techniques de l'ingénieur. Mesures et contrôle*, no. R275, pp. R275–1, 2000.

[20] J. Goupy, *Plans d'expériences pour surfaces de réponse*. Dunod, 1999.

[21] D. Mathieu and R. Phan-Tan-Luu, "Lprai université d'aix-marseille," *NEMROD®: New Efficient Methodology for Research using Optimal Design*, 1998.

[22] G. Saporta, *Probabilités, analyse des données et statistique*. Editions technip, 2006.

[23] D. J. Finney, "The fractional replication of factorial arrangements," *Annals of Eugenics*, vol. 12, no. 1, pp. 291–301, 1943.

[24] M. Cavazzuti, *Optimization methods: from theory to design scientific and technological aspects in mechanics*. Springer Science & Business Media, 2012.

[25] H. Faure, "Discrépance quadratique de la suite de van der corput et de sa symétrique," *Acta Arithmetica*, vol. 55, no. 4, pp. 333–350, 1990.

[26] M. Van Lieshout, *Markov point processes and their applications*. World Scientific, 2000.

[27] F. P. Kelly and B. D. Ripley, "A note on strauss's model for clustering," *Biometrika*, pp. 357–360, 1976.

[28] S. Anwar in *Implementation of Strauss point process model to earthquake data*, ITC, 2009.

[29] X. Descombes, *Méthodes stochastiques en analyse d'image: des champs de Markov aux processus ponctuels marqués*. PhD thesis, Université Nice Sophia Antipolis, 2004.

[30] S. Chib and E. Greenberg, "Markov chain monte carlo simulation methods in econometrics," *Econometric theory*, vol. 12, no. 3, pp. 409–431, 1996.

[31] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*. Springer Science & Business Media, 2012.

[32] R. L. Dobrushin, "Central limit theorem for nonstationary markov chains. i," *Theory of Probability & Its Applications*, vol. 1, no. 1, pp. 65–80, 1956.

[33] G. Winkler, *Image analysis, random fields and Markov chain Monte Carlo methods: a mathematical introduction*, vol. 27. Springer Science & Business Media, 2003.

[34] S. Fontaine, "Mcmc adaptatifs à essais multiples," 2019.

[35] A. J. Baddeley and M. Van Lieshout, "Area-interaction point processes," *Annals of the Institute of Statistical Mathematics*, vol. 47, pp. 601–619, 1995.

[36] D.-M. Yan, W. Wang, B. Lévy, and Y. Liu, "Efficient computation of clipped voronoi diagram for mesh generation," *Computer-Aided Design*, vol. 45, no. 4, pp. 843–852, 2013.

[37] M. D. Morris and T. J. Mitchell, "Exploratory designs for computational experiments," *Journal of statistical planning and inference*, vol. 43, no. 3, pp. 381–402, 1995.

[38] M. C. Shewry and H. P. Wynn, "Maximum entropy sampling," *Journal of applied statistics*, vol. 14, no. 2, pp. 165–170, 1987.