الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université SAAD DAHLAB Blida 1

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUES

MEMOIRE

En vue de l'obtention du diplôme de Master

Spécialité : Mathématiques

Option : Modélisation Stochastique et Statistique

TITRE

Proposition de Nouveaux Plans d'Expériences

Présenté par : ABDELGHEFFAR Lydia et KHEDIM Abderraouf

Soutenu devant le jury composé de :

Dr TAMI O.	Maître de Conférences B	Président
Dr KERDJOUDJ S.	Maître de Conférences A	Examinateur
Dr. ELMOSAOUI H.	Maître de Conférences A	Superviseur
Mr. AIT AMEUR A.	Doctorant	Co-superviseur

2023/2024

الملخص

معظم المهندسين والفنيين يسعون إلى تحسين منتجاتهم أو عمليات الإنتاج الخاصة بهم من خلال التجارب. للأسف، الاستراتيجيات المستخدمة عادة لإجراء هذه التجارب غالباً ما تكون مكلفة وغير فعالة، مما يؤدي إلى العديد من التجارب التي يصعب استغلالها. للتغلب على هذه التحديات، يتجه العديد من المحترفين إلى تخطيط التجارب.

يسمح تخطيط التجارب بتنظيم الاختبارات التي ترافق البحث العلمي أو الدراسات الصناعية بطريقة مثلى. وهي قابلة للتطبيق في العديد من التخصصات وفي جميع الصناعات، طالما كان الهدف هو إقامة علاقة بين كمية ذات اهتمام y والمتغيرات x_i. الهدف منها هو تحديد نماذج رياضية تربط بين الكميات ذات الاهتمام والمتغيرات القابلة للتحكم.

في هذا البحث، نقترح خطط تجارب رقمية جديدة تعتمد على نظرية العمليات العشوائية، خاصة العمليات النقطية المعلمة بثلاث علامات. تستند هذه الخطط على كل من توزيع النقاط في المجال التجريبي وعلى توصيف معين لعلامات هذه النقاط. يتم الحصول عليها باستخدام طريقة مونتي كارلو بسلسلة ماركوف .(MCMC) تم إجراء دراسة تفصيلية على تقارب سلاسل ماركوف. كما قمنا بمقارنة نهجنا مع خطط رقمية أخرى موجودة، مما يظهر المزايا والعيوب لكل طريقة.

يؤكد هذا العمل على أهمية منهجية تخطيط التجارب لتحسين الاختبارات والمحاكاة الرقمية، مقدماً حلولاً فعالة واقتصادية للمهندسين والفنيين.

الكلمات المفتاحية: تخطيط التجارب، تخطيط التجارب الرقمية، العمليات النقطية، العمليات النقطية المعلمة، مونتي كارلو بسلسلة ماركوف ،(MCMC) خوارزمية متروبوليس-هاستينغ.

Abstract

In this work, we propose new numerical experimental designs based on the theory of stochastic processes, particularly three-mark marked point processes. These designs are based on both the distribution of points within the experimental domain and a specific characterization of the marks of these points. They are obtained using the Monte Carlo Markov Chain method (MCMC). A detailed study on the convergence of Markov chains has been conducted. We have also compared our approach with other existing numerical designs, demonstrating the advantages and disadvantages of each method.

This work highlights the importance of the methodology of experimental designs to optimize tests and numerical simulations, providing effective and economically viable solutions for engineers and technicians.

Keywords : Design of experiments, Computer experiments design, Point processes, Marked point processes, Monte Carlo Markov chain method (MCMC), Metropolis- Hastings algorithm.

Résumé

Dans ce mémoire, nous proposons de nouveaux plans d'expériences numériques fondés sur la théorie des processus stochastiques, particulièrement les processus ponctuels marqués à trois marques. Ces plans sont basés à la fois sur la distribution des points dans le domaine expérimental et sur une certaine caractérisation des marques de ces points. Ils sont obtenus à l'aide de la méthode de Monte Carlo par chaîne de Markov (MCMC). Une étude détaillée sur la convergence des chaînes de Markov a été réalisée. Nous avons également comparé notre approche avec d'autres plans numériques existants, démontrant les avantages et les inconvénients de chaque méthode.

Ce travail souligne l'importance de la méthodologie des plans d'expériences pour optimiser les essais et les simulations numériques, apportant des solutions efficaces et économiquement viables aux ingénieurs et aux techniciens.

Mots clés : Plans d'expériences, Plans d'expériences numériques, Processus Ponctuels, Processus Ponctuels Marqués, Monté Carlo par Chaine de Markov (MCMC), Algorithme de Metropolis-Hasting.

Remerciement

Tout d'abord, Nous tenons à remercier DIEU de nous avoir donné la force de mener à bien ce modeste travail.

Nous tenons à exprimer nos sincères remerciements et notre gratitude au Dr. ELMOSSAOUI Hichem, Maître de conférences-A à l'Université Blida 1, notre directeur de mémoire, qui nous a supervisé dans notre projet de fin d'études et nous a prodigué de précieux conseils et orientations, malgré sa lourde charge de travail. Sa vaste expérience et son soutien continu ont été inestimables.

Nous tenons à remercier Mr. AIT AMEUR Ahmed, notre co-directeur, pour son soutien, ses conseils précieux et son expertise tout au long de cette expérience.

Nous exprimons notre gratitude à Dr TAMI Omar et Dr KERDJOUDJ Samia pour avoir accepté d'examiner notre mémoire de fin d'études. Leurs expertises et conseils seront d'une grande valeur pour nous.

Nous tenons à exprimer notre gratitude à tous les enseignants du département de mathématiques. Particulièrement au chef de département Dr. TAMI Omar.

DEDICACE

Je dédie ce mémoire :

"À mes chers parents, que nulle dédicace ne puisse exprimer mes sincères sentiments, pour leur patience illimitée, leur encouragement continu, leur aide, en témoignage de mon profond amour et respect pour leurs grands sacrifices."

"À mon petit frère bien-aimé, à mon petit soutien, je te souhaite prospérite et réussite dans la vie".

"À mes amis, compagnons fidèles pendant mes années de lycée et d'université, pour leur soutien inestimable et les souvenirs précieux."

Que ce travail soit l'accomplissement de vos voeux tant allégués, et le fruit de votre soutien infaillible.

ABDELGHEFFAR LYDIA

DEDICACE

Je dédie ce travail à mes parents, pour leur amour inconditionnel, leur soutien indéfectible et leurs sacrifices qui ont rendu possible la réalisation de ce mémoire. Leur confiance en moi m'a donné la force d'aller de l'avant et de poursuivre mes rêves.

je dédie ce travail à mon cher frère à tous les moments d'enfance passés avec toi mon frère, tu m'as soutenu, réconforté et encouragé. Puissent nos liens fraternels se consolider et se pérenniser encore plus.

je dédie ce travail à ma chère soeur qui n'a pas cessée de me conseiller, encourager et soutenir tout au long de mes études.Que dieu la protège et lui offre la chance et le bonheur

Aux petits enfants WASSIM et YESSMINE

Enfin, je dédie également ce travail à mes amis, pour leur présence constante, leur encouragement et les moments de joie partagés. Leurs encouragements m'ont aidé à surmonter les obstacles et à persévérer.

KHEDIM ABDERRAOUF

Table des matières

Résumé **3**

Remerciement **4**

Table des matières **9**

Listes de figures **11**

Listes des tableaux **12**

Introduction **13**

1 LES PLANS D'EXPÉRIENCES **15**

1.1 Définitions d'un plan d'expérience 15

1.2 Vocabulaire de base des plans d'expériences 16

1.2.1 Réponses 16

1.2.2 Facteur 16

1.2.3 Matrice d'expérience 17

1.2.4 Effet d'un facteur 17

1.2.5 Modèle mathématique 17

1.2.6 Domaine d'étude 17

1.2.7 Plans d'expériences 18

1.3 Différents types de plans d'expériences 19

1.3.1 Plan factoriel 19

1.3.2 Plans pour surface de réponse 21

1.3.3 Plan Latin Hypercube 24

1.3.4 Suites de Halton 24

1.3.5 Les suites de Sobol 25

1.3.6 Les suites de Faure . . . 25
1.4 Estimation des coefficients par la méthode des moindres carrés . . . 26
1.4.1 Estimation des coefficients . . . 26
1.4.2 Espérance mathématique des coefficients . . . 28
1.4.3 Variance des coefficients . . . 29
1.5 Tests statistiques . . . 29
1.5.1 Le coefficient de corrélation multiple R^2 . . . 29
1.5.2 Le F de Fisher . . . 30
1.6 Critères d'optimalités . . . 30
1.6.1 Critère D-optimalité . . . 30
1.6.2 Critère A-optimalité . . . 31
1.6.3 Critère E . . . 31
1.6.4 Critère M . . . 31
1.6.5 Critère d'orthogonalité . . . 31
1.6.6 Critère presque orthogonalité . . . 31
1.6.7 Critère d'isovariance par rotation . . . 32
1.6.8 Critère de recouvrement (cov) . . . 32
1.6.9 Critère de distance (Mindist) . . . 32
1.6.10 Critère de discrépance (Disc) . . . 32

2 LES PROCESSUS PONCTUELS . . . 34
2.1 Quelques définitions et notations . . . 34
2.2 Fidis . . . 35
2.3 Les différents types de processus ponctuels . . . 36
2.3.1 Processus ponctuels de Poisson . . . 36
2.3.2 Processus Binomial . . . 37
2.3.3 Processus de Cox . . . 37
2.3.4 Processus de Strauss . . . 38
2.3.5 Processus de Hawkes . . . 38
2.4 Processus ponctuels marqués . . . 39
2.5 Processus ponctuels de Markov . . . 40
2.5.1 Propriété de Markov au sens de Ripley-Kelly . . . 40

2.5.2 Propriété de Markov pour un processus ponctuel marqué 42
2.6 Chaines de Markov 42
2.6.1 Convergence d'une Chaine de Markov 44
2.7 MCMC - Metropolis Hasting 46
2.7.1 Méthodes de Monte-Carlo par chaînes de Markov 46
2.7.2 Metropolis Hasting général 47
2.7.3 L'échantillonneur Gibbs 48

3 NOUVEAUX PLANS D'EXPÉRIENCES NUMÉRIQUES À PARTIR DE PROCESSUS PONCTUELS MARQUÉS 49
3.1 Plan d'expériences numériques par processus ponctuels markoviens marqués de Strauss à trois marques 49
3.1.1 Choix des marques 50
3.1.2 Simulation des processus ponctuels par la méthode MCMC et l'algorithme de Metropolis-Hasting 51
3.2 L'algorithme de construction du plan proposé 52
3.3 Etude de convergence 55

4 RESULTATS ET COMPARAISONS 60
4.1 Résultats numériques et qualités des plans proposés 60
4.2 Résultats de comparaison pour les suites à faible discrépance 61
4.3 Résultats de comparaison entre les plans stochastiques 61
4.4 Plans avec 20, 50 et 100 points en 5 dimensions. 62
4.5 Plans avec 20, 50 et 100 points en 7 dimensions. 64

Conclusion 66

Annex A 67

Bibliography 108

Table des figures

1.1 Schéma de la démarche associée à un plan d'expériences 16

1.2 Illustration de l'effet d'un facteur 17

1.3 Définition du domaine d'étude pour deux facteurs 18

1.4 Plan d'expérience 19

1.5 Plan factoriel complet à deux niveaux pour trois facteurs. 21

1.6 Plan composite pour deux facteurs. 22

1.7 Plan de Box-Behnken pour trois facteurs 22

1.8 Plan de Doehlert pour l'étude de deux facteurs 23

1.9 5 points issus d'un échantillonnage par hyper cube latin en dimension 2. 24

1.10 les 100, 1000, 10000 premiers points de la suite de halton en base 2 et 3. 25

1.11 Echantillonnage via les suites de Sobol 25

2.1 Processus homogène (gauche) et processus inhomogène (droite) 37

2.2 Processus ponctuel Binomial 37

2.3 Résultat d'une simulation à partir d'un processus de Strauss à échelle locale sur [-1, 1]. 38

2.4 Illustration du processus de branchement 39

2.5 Exemple d'un processus ponctuel marqué 40

2.6 Les familles d'algorithme pour la simulation des chaînes MCMC 46

3.1 A gauche, une configuration initiale aléatoire de 20 points et à droite, une configuration finale pour $\beta_1 = 0.5, \beta_2 = 0.5, \beta_3 = 0.5, \gamma_{11} = 0.1, \gamma_{12} = 0.02, \gamma_{13} = 0.05, \gamma_{23} = 0.2, \gamma_{22} = 0.02, \gamma_{33} = 0.08, R_1 = 0.05, R_2 = 0.08, R_3 = 0.05$ *et* $r = 0.1$ 54

3.2 A gauche, une configuration initiale aléatoire de 50 points et à droite, une configuration finale pour $\beta_1 = 0.5, \beta_2 = 0.5, \beta_3 = 0.5, \gamma_{11} = 0.1, \gamma_{12} = 0.02, \gamma_{13} = 0.05, \gamma_{23} = 0.2, \gamma_{22} = 0.02, \gamma_{33} = 0.08, R_1 = 0.07, R_2 = 0.09, R_3 = 0.04$ *et* $r = 0.1$ 54

4.1 Box plots des critères de qualité calculés sur les 100 plans à 20 points en dimension 5 62

4.2 Box plots des critères de qualité calculés sur les 100 plans à 50 points en dimension 5 63

4.3 Box plots des critères de qualité calculés sur les 100 plans à 100 points en dimension 5 63
4.4 Box plots des critères de qualité calculés sur les 100 plans à 20 points en dimension 7 64
4.5 Box plots des critères de qualité calculés sur les 100 plans à 50 points en dimension 7 64
4.6 Box plots des critères de qualité calculés sur les 100 plans à 100 points en dimension 7 65

Liste des tableaux

1.1 Matrice d'expériences 17
1.2 Matrice d'expériences d'un plan 2^2 20
1.3 Matrice d'expériences d'un plan 2^3 20
1.4 Matrice d'expérience plan de Box-Behnken pour 3 facteurs. 23
1.5 Matrice d'expériences de Doehlert pour deux facteurs. 24

4.1 La valeur de la discrépance pour les plans proposés (THMD), les suites de Halton, les suites de Sobol et les suites de Faure pour quatre, sept et dix dimensions. 61

Introduction

Ces dernières années, la simulation numérique est devenue un outil incontournable pour modéliser des phénomènes de plus en plus complexes. Toutefois, la résolution de problèmes de grande dimension implique l'utilisation de codes de simulation sophistiqués, entraînant des coûts élevés en termes de temps de calcul. Dans ce contexte, l'approche privilégiée consiste à organiser un nombre restreint de simulations selon un plan d'expériences numériques, afin d'optimiser l'efficacité des analyses.

Le présent mémoire s'inscrit dans cette perspective, en explorant la méthodologie des plans d'expériences et en proposant des méthodes pour optimiser le choix de ces simulations. Les plans d'expériences sont des outils fondamentaux pour les chercheurs et les ingénieurs qui entreprennent des études scientifiques ou industrielles. Ils permettent de définir un cadre rigoureux pour l'observation des phénomènes et d'optimiser les ressources disponibles.

Les travaux existants sur les plans d'expériences, initiés par des chercheurs renommés tels que Fisher [Van Loggerenberg-Hattingh, 2003], Kiefer [Kiefer, 1959] et Box [Box and Behnken, 1960] , ont jeté les bases de cette méthodologie. Cependant, l'application des plans d'expériences à des simulations numériques pose des défis spécifiques. Contrairement aux expériences réelles, où l'erreur expérimentale peut être évaluée par la répétition des expériences, dans le contexte numérique, l'erreur est liée au modèle et non à l'expérimentation elle-même.

La diversité des plans d'expériences proposés dans la littérature reflète la complexité des problèmes rencontrés. Chaque plan présente des avantages et des inconvénients selon les critères d'optimalité considérés, nécessitant souvent un compromis entre différentes exigences. L'objectif principal de ce travail est de proposer de nouveaux plans d'expériences numériques fondés sur la théorie des

processus ponctuels marqués [Daley et al., 2003]. Cette approche permet d'introduire des connaissances géométriques et des informations a priori sur les points expérimentaux, améliorant ainsi la qualité des expériences réalisées. Nous nous appuierons notamment sur les processus ponctuels marqués de Strauss [Franco, 2008], en utilisant des techniques de simulation telles que la méthode MCMC avec l'algorithme de Metropolis-Hastings [Chib and Greenberg, 1995a, Hastings, 1970c].

La structure de ce mémoire est articulée autour de quatre chapitres, chacun apportant une contribution spécifique à la méthodologie des plans d'expériences numériques :

Le premier chapitre expose la théorie et la méthodologie des plans d'expériences, offrant un état de l'art des pratiques courantes dans le domaine de l'expérimentation numérique. Des critères pertinents pour évaluer la qualité de ces plans sont également discutés.

Dans **le deuxième chapitre**, nous introduisons la théorie des processus ponctuels et des processus objets. Après une présentation des fondements des processus ponctuels, qui servent de base aux processus objets, nous abordons les processus de Markov, mettant en lumière leur capacité à modéliser les interactions entre objets. La simulation de ces processus est décrite à travers le prisme des chaînes de Markov, avec une discussion sur les concepts fondamentaux et les conditions de convergence.

Le troisième chapitre se concentre sur l'idée centrale de la construction de nouvelles matrices d'expériences numériques, en utilisant des processus ponctuels marqués à trois marques via la méthode MCMC avec l'algorithme de Metropolis-Hastings. Nous abordons également la question de la convergence de cette construction.

Dans **le quatrième** chapitre, nous effectuerons une comparaison avec d'autres plans d'expérience décrits dans la littérature, en tenant compte des besoins spécifiques de chaque étude et de chaque utilisateur.

Enfin, une conclusion termine ce travail, donnant quelques perspectives pour des recherches futures. En annexe, nous présentons les programmes développés dans le logiciel PYTHON permettant de réaliser les illustrations numériques fournis dans le troisième et le quatrième chapitre.

Chapitre 1

LES PLANS D'EXPÉRIENCES

Dans de nombreuses disciplines, il est courant d'étudier des phénomènes dépendant de plusieurs paramètres. L'approche intuitive, qui consiste à fixer les niveaux de toutes les variables sauf une et à mesurer la réponse pour différentes valeurs de cette variable, n'est pas toujours la plus efficace. En effet, si plusieurs paramètres doivent être étudiés, cette méthode nécessite de répéter ces étapes pour chaque paramètre, ce qui peut devenir un travail long et souvent irréalisable et difficile. C'est là qu'interviennent les plans d'expériences, également appelés plans d'expérimentation ou plans d'échantillonnage, qui sont des outils statistiques utilisés pour concevoir, organiser et analyser des expériences scientifiques de manière efficace. Contrairement à l'approche intuitive, les plans d'expériences font varier simultanément tous les niveaux à chaque expérience. Cette méthode permet d'établir un plan expérimental comportant le minimum d'essais tout en fournissant des résultats précis et fiables. Un plan d'expérience est souvent utilisé dans plusieurs contextes tels que :

- Mise au point d'un nouveau produit,
- Étude technique,
- Optimisation de processus,
- Amélioration de la qualité d'un produit.

1.1 Définitions d'un plan d'expérience

Un plan d'expériences est une stratégie optimale permettant de prédire avec le maximum de précision une réponse à partir d'un nombre minimal d'essais. Un plan d'expériences n'est pas une série d'essais au hasard ni sélectionnés par la seule intuition. D'une manière plus générale, est une méthode systématique pour organiser une expérience afin d'obtenir des informations précises et fiables

sur un processus ou un système. Il est largement utilisé dans la recherche scientifique, l'ingénierie, l'industrie, et d'autres domaines pour étudier l'effet de variables sur un résultat donné.

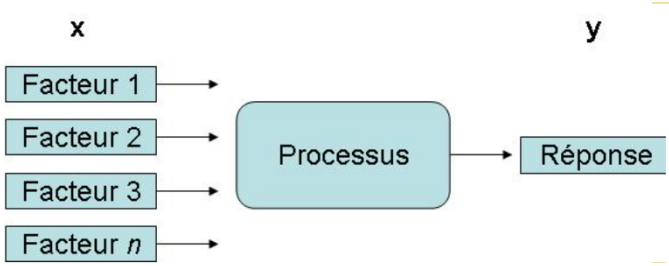

Figure 1.1 – Schéma de la démarche associée à un plan d'expériences

Les avantages des plans d'expériences sont nombreux :

- La diminution du nombre d'essais,
- La possibilité d'étudier un grand nombre de facteurs,
- La connaissance des effets des paramètres et la détection des interactions entre facteurs,
- La modélisation mathématique des réponses étudiées,
- Précision des résultats,
- La prédiction et l'optimisation.

1.2 Vocabulaire de base des plans d'expériences

La compréhension de la méthode des plans d'expériences s'appuie sur plusieurs notions essentielles que nous allons expliquer dans les paragraphes suivants.

1.2.1 Réponses

Une réponse c'est une grandeur qu'on mesure à chaque essai. Il est possible qu'un phénomène puisse être décrit par plusieurs réponses expérimentales.

1.2.2 Facteur

Les facteurs sont les variables que l'on souhaite examiner. Ils peuvent être de nature quantitative ou qualitative, continue ou discrète. En général, nous nous concentrons sur les facteurs caractérisés par une seule variable naturelle, qualitative ou quantitative, que l'on peut contrôler. Chaque facteur est délimité par deux niveaux, un niveau bas (-1) et un niveau haut (+1).

1.2.3 Matrice d'expérience

La matrice d'expérience est un tableau constitué de n lignes correspondant aux n expériences à réaliser et de K colonnes correspondant aux K facteurs étudiées. L'élément ij de la matrice ainsi formé correspond à la valeur des niveaux que prend la $j^{\text{ème}}$ variable à la $i^{\text{ème}}$ expérience.

Exemple 1.1 : Une matrice d'expérience à 4 essais et 2 facteurs

Table 1.1 – Matrice d'expériences

Essai N	Facteur 1	Facteur 2
1	-1	-1
2	+1	-1
3	-1	+1
4	+1	+1

1.2.4 Effet d'un facteur

L'effet d'un facteur X correspond à la variation de la réponse Y lorsque X passe d'une valeur au niveau (-1) à une autre valeur au niveau (+1), comme indiqué sur la Figure 1.2. L'ensemble de toutes les valeurs que peut prendre un facteur entre ces deux bornes est le domaine de variation [Kai-Tai Fang., 2006].

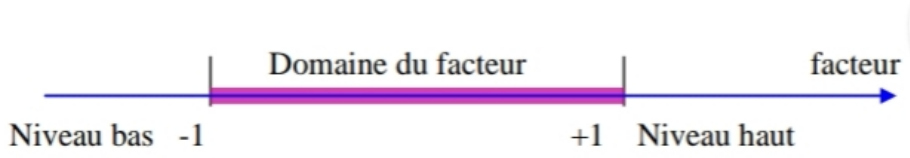

Figure 1.2 – Illustration de l'effet d'un facteur

1.2.5 Modèle mathématique

Il s'agit d'une relation mathématique qui illustre comment une réponse change en fonction de la variation d'un ou de plusieurs facteurs. L'objectif de modéliser la réponse par un modèle mathématique est de pouvoir prédire toutes les réponses possibles du domaine d'étude sans avoir à réaliser toutes les expériences. Ce modèle est appelé modèle postulé ou a priori.

1.2.6 Domaine d'étude

Le regroupement des domaines des facteurs définit le domaine d'étude. Étant donné la définition des k facteurs et de leurs variations respectives, il devient naturel de définir un espace k-dimensionnel

dans lequel chaque point correspond à une configuration des k facteurs. Cet espace est appelé domaine d'étude ou encore espace de recherche. Les points d'expérience peuvent se situer soit à l'intérieur, soit sur les frontières du domaine (voir Figure 1.3).

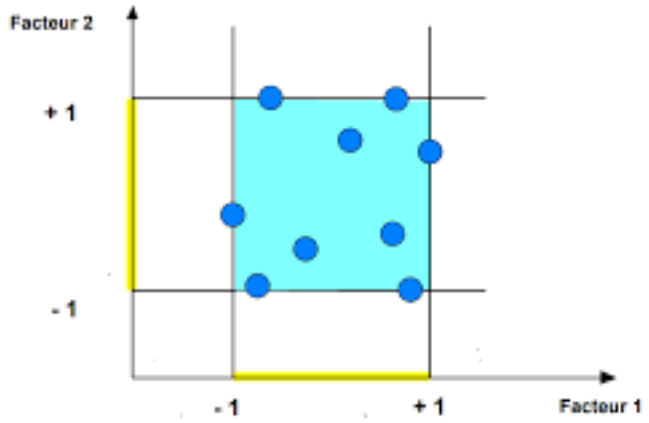

Figure 1.3 – Définition du domaine d'étude pour deux facteurs

1.2.7 Plans d'expériences

Chaque point du domaine d'étude représente des conditions opératoires possibles, donc une expérience que l'opérateur peut réaliser [Goupy, 1999]. Le choix du nombre et de l'emplacement des points d'expérience est le problème fondamental des plans d'expériences. Nous avons l'habitude d'appeler plans d'expériences des ensembles de points expérimentaux répondant à des propriétés bien précises. Ce sont les plans d'expériences classiques. Ils sont connus et largement publiés. Lorsque les points expérimentaux sont disposés autrement que dans les plans d'expériences classiques, nous parlons de plans non conventionnels. Leurs propriétés sont le plus souvent moins bonnes que celles des plans classiques. Mais ce sont des plans que l'on rencontre, car il n'est pas toujours possible de respecter les impératifs des plans d'expériences classiques.

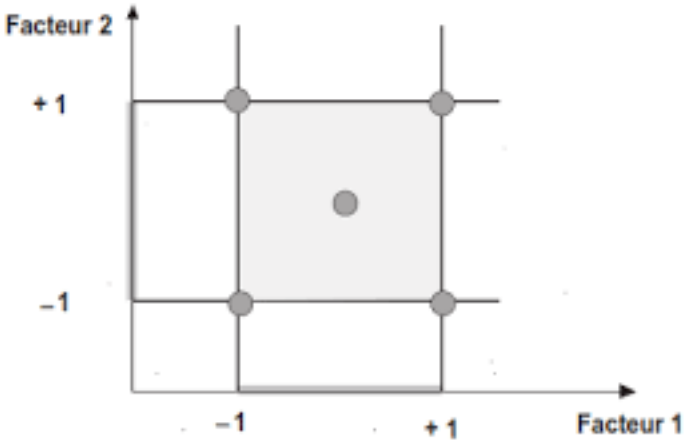

Figure 1.4 – Plan d'expérience

1.3 Différents types de plans d'expériences

Il existe plusieurs types de plans d'expériences, chacun adapté à des situations expérimentales spécifiques. Ces types de plans peuvent être classés en fonction de différents critères tels que le nombre de facteurs étudiés, le nombre de niveaux de chaque facteur et la répartition des essais. Nous introduisons dans cette partie quelques types de plans d'expériences classiques et de plans d'expériences numériques :

1.3.1 Plan factoriel

Ces plans permettent d'identifier les facteurs les plus influents sur une réponse donnée, sans chercher à établir une relation précise entre les variations des facteurs et celles de la réponse. Ils sont conçus comme des plans discrets orthogonaux, où les facteurs sont discrétisés pour être testés à un nombre fini de niveaux. Il y a deux types principaux de plans factoriels : les plans factoriels complets et les plans fractionnaires.

Plan factoriel complet à deux niveaux

Un plan factoriel complet implique d'étudier toutes les combinaisons possibles des niveaux des facteurs étudiés dans l'expérience [George E. P. Box, 2007]. Cela signifie que tous les effets principaux et toutes les interactions entre les facteurs peuvent être estimés à partir d'un plan factoriel complet [Ray-Bing Chen., 2008]. Le nombre d'expériences à réaliser se calcule par : 2^k où k représente le nombre de facteurs.

À mesure que le nombre de facteurs dans un plan factoriel à deux niveaux augmente, le nombre d'essais nécessaires pour effectuer un plan factoriel complet augmente rapidement. Par exemple, un plan factoriel complet à 2 niveaux avec 6 facteurs nécessite 2^6 essais (64 essais) ; tandis qu'un plan avec 9 facteurs nécessite 2^9 essais (512 essais).

Table 1.2 – Matrice d'expériences d'un plan 2^2

Essai Nř	Facteur 1	Facteur 2
1	-1	-1
2	+1	-1
3	-1	+1
4	+1	+1

La matrice des essais comporte k colonnes et 2^k lignes. Elle se construit simplement :

- La colonne du premier facteur : une alternance de -1 et +1,
- La colonne du deuxième facteur : une alternance de -1 et +1 de 2 en 2,
- La colonne du troisième facteur : une alternance de -1 et +1 de 4 en 4,
- La colonne du quatrième facteur : une alternance de -1 et +1 de 8 en 8, ... etc.

Pour trois facteurs, la matrice d'expériences d'un plan factoriel complet à deux niveaux 2^3 est représentée par le tableau 1.3.

Table 1.3 – Matrice d'expériences d'un plan 2^3

Essai Nř	Facteur 1	Facteur 2	Facteur 3
1	-1	-1	-1
2	+1	-1	-1
3	-1	+1	-1
4	+1	+1	-1
5	-1	-1	+1
6	+1	-1	+1
7	-1	+1	+1
8	+1	+1	+1

Les points expérimentaux d'un plan 2^k sont représentés par les sommets d'un hypercube à k dimensions. La figure ci-dessus montre le domaine d'étude pour le plan factoriel 2^3.

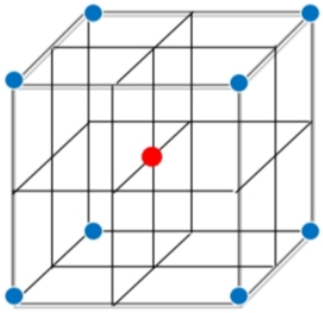

Figure 1.5 – Plan factoriel complet à deux niveaux pour trois facteurs.

Plans factoriels fractionnaires à deux niveaux

Les plans fractionnaires sont désignés sous le nom de plans 2^{k-p}. Le chiffre 2 indique que chaque facteur possède 2 niveaux, le k indique qu'il y a k facteurs étudiés, et le p signifie qu'il y a p facteurs supplémentaires par rapport au nombre de facteurs du plan complet. L'avantage des plans fractionnaires est évident : la charge expérimentale est réduite par facteur de 2^p. Car $2^{k-p} = \frac{2^k}{2^p}$.

1.3.2 Plans pour surface de réponse

Plus que de classer les effets des différents facteurs, cette catégorie de plans vise à décrire de manière précise le comportement de la réponse en fonction des variations des facteurs. L'objectif de cette forme d'étude consiste donc à obtenir une représentation graphique du phénomène étudié en utilisant des expériences. Ces plans offrent la possibilité d'établir les valeurs des facteurs d'entrée d'un dispositif afin d'obtenir une ou plusieurs réponses désirées, en utilisant des modèles polynomiaux de second degré.

Plans composite

Un plan composite permet d'étudier un modèle quadratique, adapté à de nombreux phénomènes non linéaires. Les plans composites se composent de trois parties distinctes, ce qui permet une approche séquentielle [Lin., 2008] :

- **Partie 1** : Un plan factoriel : il s'agit d'un plan factoriel complet ou fractionnaire à deux niveaux par facteur, les points expérimentaux étant placés aux sommets du domaine d'étude.
- **Partie 2** : Un plan en étoile : les points du plan en étoile se trouvent sur les axes et sont généralement tous situés à la même distance du centre du domaine d'étude (les points F, G, H, et I de la Figure 1.6).

- **Partie 3** : Des points au centre du domaine d'étude : des points expérimentaux sont toujours placés au centre du domaine d'étude.

La Figure 1.6 montre les points expérimentaux pour un plan composite à deux facteurs.

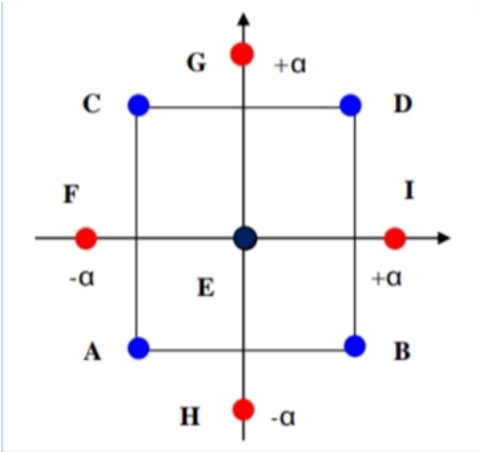

Figure 1.6 – Plan composite pour deux facteurs.

Plans de Box-Behnken

Box et Behnken (1960) [Box and Behnken, 1960] ont introduit un type de plans d'expériences différent pour les modèles du deuxième ordre. Les plans de Box-Behnken comportent moins de points que les plans composites pour le même nombre de facteurs, ce qui les rend moins coûteux. Un plan de Box-Behnken est un type de plan de surface de réponse qui ne comprend pas de plan factoriel fractionnaire. Les plans de Box-Behnken pour 3 facteurs impliquent trois blocs, dans chacun desquels un plan factoriel 2^2 est représenté par chaque paire de traitements, tandis que le troisième facteur reste fixé à 0. Ainsi, le plan de Box-Behnken pour 3 facteurs comporte 15 essais (12 essais sur les arêtes et 3 au centre, Figure 1.7).

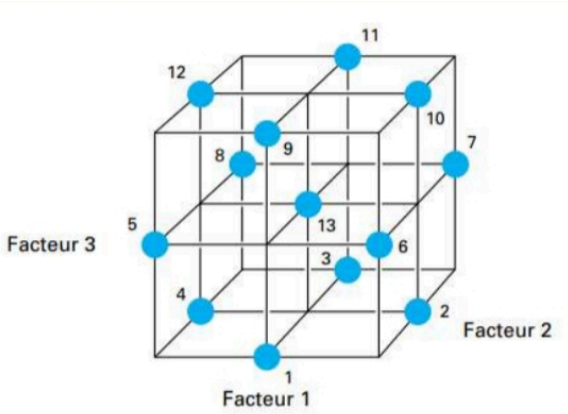

Figure 1.7 – Plan de Box-Behnken pour trois facteurs

La matrice d'expériences pour trois facteurs est représentée par le table 1.4 suivant :

Table 1.4 – Matrice d'expérience plan de Box-Behnken pour 3 facteurs.

Essai Nř	Facteur 1	Facteur 2	Facteur 3
1	-1	-1	0
2	+1	-1	0
3	-1	+1	0
4	+1	+1	0
5	0	-1	-1
6	0	-1	+1
7	0	+1	-1
8	0	+1	+1
9	-1	0	-1
10	-1	0	+1
11	+1	0	-1
12	+1	0	+1
13	0	0	0
14	0	0	0
15	0	0	0

Plans de Doehlert

Les plans expérimentaux proposés par David H. Doehlert en 1970 [Jacques, 2000] sont caractérisés par une répartition uniforme des points dans l'espace expérimental. Pour deux facteurs, les points sont disposés aux sommets d'un hexagone régulier, avec un point central. Avec sept points expérimentaux, ce plan permet de déterminer au moins sept inconnues, correspondant à sept coefficients. Grâce à la répartition régulière des points, il est aisé d'étendre le plan dans n'importe quelle direction de l'espace en ajoutant des points régulièrement espacés.

Ces plans offrent également une facilité d'introduction de nouveaux facteurs. Les nouvelles expériences compléteront les premières, sans aucune perte. Il est simplement nécessaire de maintenir les facteurs non étudiés à une valeur constante (niveau 0) pendant l'étude des facteurs actifs.

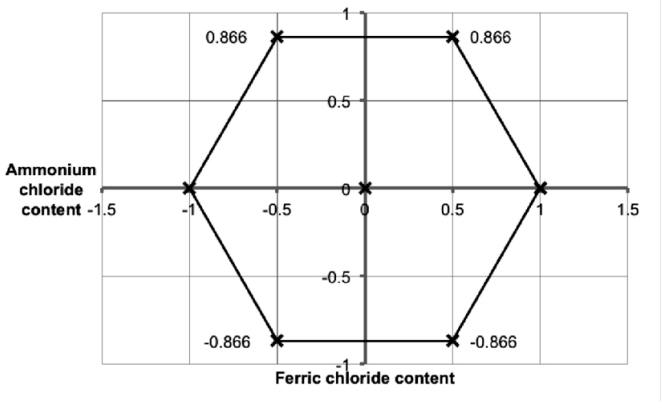

Figure 1.8 – Plan de Doehlert pour l'étude de deux facteurs

Le tableau 1.5 représente, sous forme de matrice d'expériences, la disposition des points selon la

Figure 1.8. Cette disposition conduit à cinq niveaux pour le facteur 1 et trois niveaux pour le facteur 2.

Table 1.5 – Matrice d'expériences de Doehlert pour deux facteurs.

Essai nř	Facteur 1	Facteur 2
1	0	0
2	1	0
3	0.5	0.866
4	-0.5	0.866
5	-1	0
6	-0.5	-0.866
7	0.5	-0.866

1.3.3 Plan Latin Hypercube

Un plan Latin Hypercube (PLH) en n essais est un plan d'expériences où chaque facteur possède le même nombre de niveaux n et chaque facteur prend chaque niveau une fois et une seule. Les niveaux sont répartis de manière équilibrée. Ainsi, chaque colonne du plan d'expériences est un tirage aléatoire sans remise parmi 1,2,...,n. Pour illustrer, un échantillonnage par hypercube latin en dimension 2, nous divisons l'intervalle de chaque variable en $n = 5$ sous-intervalles de même taille. Pour chaque variable et dans chaque sous-intervalle, un point est généré selon une loi uniforme, comme illustré dans la Figure 1.9.

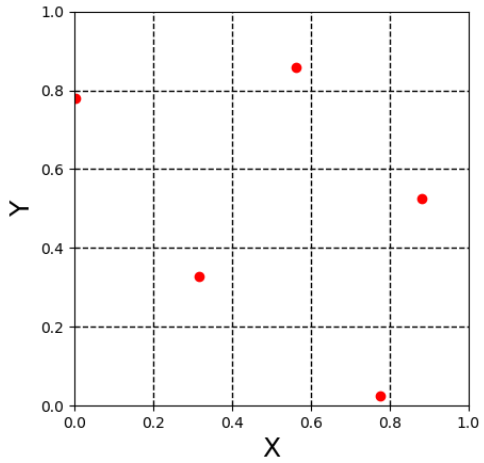

Figure 1.9 – 5 points issus d'un échantillonnage par hyper cube latin en dimension 2.

1.3.4 Suites de Halton

Les suites de Halton étendent les suites de Van Der Corput en dimension $d > 1$. Les suites de Van Der Corput sont leur version unidimensionnelle. Pour générer les suites de Halton, nous utilisons une

base distincte pour chaque dimension.

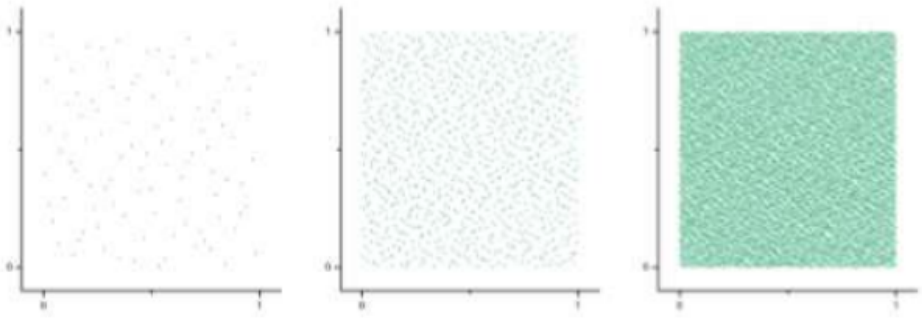

Figure 1.10 – les 100, 1000, 10000 premiers points de la suite de halton en base 2 et 3.

1.3.5 Les suites de Sobol

Ces suites permettent de répartir les points dans l'espace de manière à minimiser la distance entre chaque observation. Leur construction est complexe et repose sur des récurrences linéaires à partir de polynômes primitifs sur le corps $Z_2 = \{0, 1\}$. Elles sont qualifiées de quasi-aléatoires car il est toujours possible de déterminer les coordonnées du deuxième point à partir du premier, et ainsi de suite.

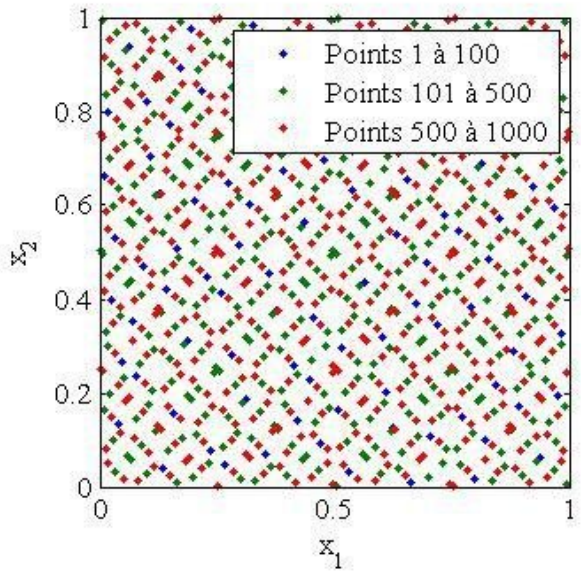

Figure 1.11 – Echantillonnage via les suites de Sobol

1.3.6 Les suites de Faure

Ces suites sont considérées comme meilleures que celles de Halton et de Sobol. Dans une séquence de Faure, la base est définie comme le plus petit nombre premier > 2 qui est supérieur ou égal

au nombre de dimensions du problème. Pour réorganiser la séquence, nous utilisons une équation récursive appliquée aux coefficients j.

1.4 Estimation des coefficients par la méthode des moindres carrés

La méthode des moindres carrés est une méthode utilisée pour estimer les coefficients d'un modèle de régression. Elle vise à minimiser la somme des carrés des résidus, c'est-à-dire des différences entre les valeurs observées et les valeurs prédites par le modèle.

1.4.1 Estimation des coefficients

Nous avons des inconnues $a_0, a_1, \ldots, a_p$ et nous voulons les estimé par $\hat{a}_0, \hat{a}_1, \ldots, \hat{a}_p$, nous avons aussi e_i sont des estimateurs des ϵ_i et nous calculons la réponse au point i par :

$$\hat{y}_i = \hat{a}_0 + \hat{a}_1 x_{i1} + \hat{a}_2 x_{i2} + \cdots + \hat{a}_i x_{ij} + \cdots + \hat{a}_p x_{ip}$$

Et nous avons :

$$y_i = \hat{y}_i + e_i$$

Alors quelque soit i nous obtenons ce système :

$$\begin{cases} y_1 = \hat{a}_0 + \hat{a}_1 x_{11} + \hat{a}_2 x_{12} + \cdots + \hat{a}_j x_{1j} + \cdots + \hat{a}_p x_{1p} + e_1 \\ y_2 = \hat{a}_0 + \hat{a}_1 x_{21} + \hat{a}_2 x_{22} + \cdots + \hat{a}_j x_{2j} + \cdots + \hat{a}_p x_{2p} + e_2 \\ \vdots \\ y_i = \hat{a}_0 + \hat{a}_1 x_{i1} + \hat{a}_2 x_{i2} + \cdots + \hat{a}_j x_{ij} + \cdots + \hat{a}_p x_{ip} + e_i \\ \vdots \\ y_n = \hat{a}_0 + \hat{a}_1 x_{n1} + \hat{a}_2 x_{n2} + \cdots + \hat{a}_j x_{nj} + \cdots + \hat{a}_p x_{np} + e_n \end{cases}$$

Il faut que le nombre des équations n soit supérieure au nombre de coefficients. Nous cherchons les valeurs $\hat{a}_j$ qui minimisent $\sum e_i^2$. Pour cela nous définissons :

1) La matrice des réponses :

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix}$$

2) La matrice de coefficient :

$$\hat{A} = \begin{bmatrix} \hat{a}_0 \\ \hat{a}_1 \\ \vdots \\ \hat{a}_j \\ \vdots \\ \hat{a}_p \end{bmatrix}$$

3) Le vecteur des écarts :

$$e = \begin{bmatrix} e_1 \\ \vdots \\ e_i \\ \vdots \\ e_n \end{bmatrix}$$

Nous voulons résoudre le système :

$$y = X\hat{A} + e \tag{1.1}$$

par le critère de moindre carré qui sert à minimiser ${}^t ee$

D'après l'équation (1.1) nous avons :

$$\begin{aligned} {}^t ee &= {}^t (y - X\hat{A})(y - X\hat{A}) \\ &= ({}^t y - {}^t \hat{A}{}^t X)(y - X\hat{A}) \\ &= {}^t yy - {}^t yX\hat{A} - {}^t X{}^t \hat{A}y - {}^t \hat{A}{}^t XX\hat{A} \end{aligned}$$

Vu que ${}^t ee$ est un scalaire donc

$$^tyX\hat{A} = \ ^tX^t\hat{A}y$$

Au final nous obtenons :

$$^tee =^t yy - 2^t\hat{A}^tXy +^t \hat{A}^tXX\hat{A}$$

Nous calculons la dérivée de ^tee par rapport à $\hat{A}$

$$\frac{d\ ^tee}{d\hat{A}} = \frac{d\ ^tyy}{d\hat{A}} - 2\frac{d\ ^t\hat{A}^tXy}{d\hat{A}} + \frac{d\ ^t\hat{A}^tXX\hat{A}}{d\hat{A}}$$

Où :

- $\frac{d\ ^tyy}{d\hat{A}} = 0$,
- $\frac{d\ ^t\hat{A}^tXy}{d\hat{A}} = \ ^tXy$,
- $\frac{d\ ^t\hat{A}^tXX\hat{A}}{d\hat{A}} = 2^tXX\hat{A}$.

Donc

$$\frac{d\ ^tee}{d\hat{A}} = -2^tXy + 2^tXX\hat{A}$$

Maintenant nous minimisons ^tee

$$\frac{d\ ^tee}{d\hat{A}} = 0 \implies {}^tXX\hat{A} = \ ^tXy$$

Finallement nous obtenons :

$$\hat{A} = (^tXX)^{-1}\ ^tXy$$

1.4.2 Espérance mathématique des coefficients

D'après le résultat précédent, l'espérance mathématique de $\hat{A}$ définie par :

$$\begin{aligned} E(\hat{A}) &= E[(^tXX)^{-1}\ ^tXy] \\ &= (^tXX)^{-1}\ ^tXE(y) \end{aligned} \tag{1.2}$$

Nous avons $y = XA + \epsilon$, donc :

$$E(y) = E(XA + \epsilon) = E(XA) = XE(A) \tag{1.3}$$

(car E(ϵ)=0 par hypothèse)

Nous remplaçons l'équation (1.3) dans l'équation (1.2), nous obtenons :

$$E(\hat{A}) = ({}^tXX)^{-1}\ {}^tXXA = A$$

1.4.3 Variance des coefficients

Par définition :

$$var(\hat{A}) = E[(\hat{A} - A)^t(\hat{A} - A)]$$

Nous remplaçons $\hat{A}$ par sa formule et y aussi nous trouvons :

$$\begin{aligned}(\hat{A} - A) &= ({}^tXX)^{-1}\ {}^tX(XA + \epsilon) - A \\ &= A + ({}^tXX)^{-1}\ {}^tX\epsilon - A \\ &= ({}^tXX)^{-1}\ {}^tX\epsilon\end{aligned}$$

Donc :

$$\begin{aligned}var(\hat{A}) &= E[({}^tXX)^{-1}\ {}^tX\epsilon\ {}^t\epsilon X({}^tXX)^{-1}] \\ &= ({}^tXX)^{-1}\ {}^tXE(\epsilon\ {}^t\epsilon)X({}^tXX)^{-1}\end{aligned}$$

Vu que $E({}^tee) = E[(\epsilon - 0)^t(\epsilon - 0)] = var(\epsilon) = \sigma^2$ Alors :

$$var(\hat{A}) = ({}^tXX)^{-1}\ {}^tX\sigma^2X({}^tXX)^{-1}$$

$\implies var(\hat{A}) = \sigma^2({}^tXX)^{-1}$

1.5 Tests statistiques

1.5.1 Le coefficient de corrélation multiple R^2

Le coefficient de corrélation multiple R^2 est une mesure de la qualité de l'ajustement d'un modèle de régression linéaire multiple aux données. Le coefficient de détermination multiple R^2 est calculé comme suit :

$$R^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} = \frac{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

tel que

- SSE (Sum of Squared Errors) est la somme des carrés des résidus du modèle définit par :

$$SSE = \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- SST (Total Sum of Squares) est la somme des carrés des différences entre les valeurs observées de la variable dépendante et la moyenne de la variable dépendante.

$$SST = \sum_{i=1}^{n}(Y_i - \bar{Y})^2$$

- SSR (Sum of Squares due to Regression) est la somme des carrées due à la régression.

$$SSR = \sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2$$

1.5.2 Le F de Fisher

La formule de Fisher est définie par [Dodge and Rousson, 1999] :

$$F = \frac{\frac{SSR}{p-1}}{\frac{SSE}{n-p}}$$

tel que :

- $(p-1)$ est le degrè de liberté de SSR
- $(n-p)$ est le degrè de liberté de SSE

Si F de Fisher est élevé, les réponses calculées est plus grande que la variance des résidus. Pour avoir des coefficients significatifs il faut un F de Fisher élevé, c'est à dire une probabilité faible.

1.6 Critères d'optimalités

Les critères d'optimalité jouent un rôle crucial dans la conception des plans d'expérience, permettant de déterminer la disposition des points expérimentaux pour obtenir des résultats fiables et informatifs. Ces critères visent généralement à maximiser l'efficacité de l'expérience en minimisant le nombre de points nécessaires pour estimer les effets des facteurs et optimiser les réponses du système étudié.

1.6.1 Critère D-optimalité

Nous disons qu'une matrice d'expériences respecte le critère D-optimalité si le déterminant de la matrice de dispersion $({}^tXX)^{-1}$ est le plus faible ou bien le determinant de la matrice d'information $({}^tXX)$ est le plus fort.

1.6.2 Critère A-optimalité

Nous disons qu'une matrice d'expériences respecte le critère A-optimalité si la trace de sa matrice de dispersion $({}^tXX)^{-1}$ est minimale.

1.6.3 Critère E

Une matrice d'expériences est dite E-optimale, si elle conduit à la valeur propre maximale de $({}^tXX)^{-1}$ la plus faible possible.

1.6.4 Critère M

Ce critère permet de comparer deux matrices d'expériences qui n'ont pas nécessairement le même nombre d'expériences. Supposons que M_1 et M_2 soient deux matrices de moments associées à deux matrices d'expériences constituées respectivement de N_1 et N_2 expériences.
Sanchant que la matrice des moments est définie par :

$$M = \frac{({}^tXX)}{n}$$

et

$$M_1 = \frac{({}^tX_1X_1)}{N_1} \qquad\qquad M_2 = \frac{({}^tX_2X_2)}{N_2}$$

Si $|M_1| > |M_2|$ on dit que le premier plan est plus efficace que le deuxième plan.

1.6.5 Critère d'orthogonalité

Une matrice est orthogonale si ses vecteurs lignes ou colonnes sont orthogonaux deux à deux. L'orthogonalité de deux vecteurs x_1 et x_2 est telle que : ${}^tx_1x_2 = 0$.

1.6.6 Critère presque orthogonalité

On dit qu'une matrice respecte le critère de presque orthogonalité si nous retirons la première ligne et la première colonne de sa matrice de dispersion $({}^tXX)^{-1}$ nous obtenons une matrice diagonale [PLARD, 2014].

1.6.7 Critère d'isovariance par rotation

Nous cherchons à obtenir des réponses calculées à partir du modèle expérimental avec une erreur de prédiction identique pour des points situés à la même distance au centre du domaine d'étude. Ce type de plan est appelé plan isovariance par rotation.

1.6.8 Critère de recouvrement (cov)

Permet de mesurer la distance entre les points du plan et ceux d'une grille régulière, ce critère est nul pour une grille régulière. L'objectif est donc de le minimiser pour se rapprocher d'une grille régulière, tout en maintenant une distribution uniforme, notamment en projection sur les axes factoriels [Gunzburger and Burkardt, 2004] :

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

Pour une grille régulière, $\sigma_0 = \sigma_1 = \cdots = \sigma_n$, alors cov = 0.

Dans le même contexte, nous pouvons utiliser le ratio R, définie par :

$$R = \frac{MAX\sigma_i}{MIN\sigma_i}$$

Pour une grille régulière $R = 1$, Ainsi, plus R est proche de 1, et plus les points sont proches de ceux d'une grille régulière.

1.6.9 Critère de distance (Mindist)

Représente la plus petite distance entre une paire parmi n points du plan [Johnson M.E. and D., 1990], ce critère est défini par :

$$Mindist = min\, min\, d(x_i, x_j)$$

Avec, $d(x_i, x_j)$ est la distance Euclidienne entre le point x_i *et* x_j . Une valeur plus élevée de Mindist devrait correspondre à une dispersion plus uniforme des points du plan.

1.6.10 Critère de discrépance (Disc)

L'écart entre la fonction de répartition empirique des points du plan et celle de la loi uniforme est mesuré par la discrépance. La discrépance, à la différence des deux derniers critères précédents,

ne repose pas sur la distance entre les points. Il y a différentes formes de discrétion. La discrétion est retenue dans la norme L2 [Warnock, 1995].

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

Chapitre 2

LES PROCESSUS PONCTUELS

Un processus ponctuel est une séquence d'actions déclenchées par des événements particuliers, temporaire et non structuré. Contrairement aux processus standards, qui sont répétitifs et structurés, un processus ponctuel est un type de processus stochastique servant à répondre à des situations d'urgence et à gérer des cas exceptionnels qui ne peuvent pas être traités efficacement par les processus standards. Un tel processus peut être utilisé dans de nombreux domaines tels que les télécommunications, la gestion des ressources humaines, le service client, la géographie, la démographie, etc.

Dans ce qui suit, nous commencerons par aborder quelques notions générales sur les processus ponctuels, puis nous citerons quelques exemples de processus ponctuels de référence. Ensuite, nous exposerons les processus ponctuels marqués, les processus ponctuels et ponctuels marqués de Markov, ainsi que les algorithmes de simulation pour les chaînes de Markov.

2.1 Quelques définitions et notations

Définition 2.1. (Configuration). *Tout ensemble* $\mathbf{x} = (x_1, \ldots, x_n, \ldots)$ *est appelé configuration où* $x_n \in (\chi, d)$ *sont des points issus d'une expérience aléatoire. Il est considéré comme localement fini si une configuration possède un nombre fini de points dans n'importe quel borélien borné* B *de* (χ, d)*. La famille des configurations localement finis est noté* $\mathbf{E}^{lf}$*.*
Il n'existe donc pas de points d'accumulation dans une configuration localement fini x*.*

$$\mathbf{E}^{lf} = \{x \subset \chi : n(x_B) < \infty, \forall B \subseteq \chi\}$$

Définition 2.2. (Processus ponctuels). *Une application* X *sur un espace probabilisé* $(\Omega, \mathbb{A}, P)$ *vers la famille de configurations de points de* χ *localement finies est appelée processus ponctuel si*

pour tout borélien $A \subseteq \chi$, le nombre de points en A, $N_x(A)$ soit une variable aléatoire discrète finie.

Exemple 2.1. L'exemple initial qui peut nous venir à l'esprit est celui d'un processus ponctuel qui envoie de manière uniforme un point dans un segment : χ=$[0, a]$ et X=$\{U\}$ où U : $\Omega \to$ [0,a] et $\Omega \to u$ suit une loi uniforme. Dans ce cas : $N(A) = 1$.

Exemple 2.2. Regardons le processus $X = \{X_1; X_1 + X_2; X_1 + X_2 + X_3; \dots\}$ où $\{X_i : \Omega \to R^+\}_{i \in \mathbb{N}^*}$ désigne une suite de loi exponentielle de paramètre λ à termes indépendants. Un tel processus est de Poisson, d'intensité λ.

Définition 2.3. (Loi d'un processus ponctuel). *Nous appelons loi du processus ponctuel la mesure de probabilité induite π sur N^{lf}.*

Remarque 2.1 : D'autres définitions de processus ponctuels existent. Une des méthodes les plus courantes est celle qui réalise un processus ponctuel en utilisant des mesures aléatoires, en associant des masses aux points du processus. L'ensemble des mesures de comptage aléatoires, réparties de manière locale, est représenté par N^{lf}. Il existe aussi une formulation en termes d'un ensembles aléatoires, dont une est liée à la famille $\{N(A)\}_A$ qui compte le nombre de points dans les boréliens A. Cette définition constructive requiert néanmoins la condition suivante : lorsque $A \cap B = \varnothing$; $N(A \cup B) = N(A) + N(B)$ presque sûrement.

2.2 Fidis

Dans cette brève section, nous définissons un exemple similaire des fonctions de répartition pour les processus ponctuels.

Définition 2.4. (Fidis). *Supposons qu'un processus ponctuel soit* **X** *sur (χ ; d). La famille des lois marginales de dimensions finis, également connue sous le nom de fidis (fidis pour Finite Dimensional Distributions), regroupant toutes les lois marginales $(N(A_1); \dots; N(A_m))$, quelque soit le vecteur de taille finie m prenant en compte $(A_1; \dots; A_m)$. $A_1, \dots, A_m$ sont des boréliens bornées de χ.*

L'intérêt de cette définition est justifié par le théorème suivant [Jaulin, 2008].

Théorème 2.1. *Les données de la famille des fidis déterminent entièrement la loi d'un processus ponctuel sur un espace métrique complet séparable.*

Remarque 2.2. *Ce théorème implique que si deux processus ponctuels ont les mêmes fidis, alors ils suivent la même loi.*

2.3 Les différents types de processus ponctuels

Il existe plusieurs types de processus ponctuels, chacun ayant des propriétés et des applications spécifiques. Voici quelques-uns des principaux types de processus ponctuels :

2.3.1 Processus ponctuels de Poisson

Le processus ponctuel de Poisson est le plus simple et le plus universel des processus ponctuels. Soit $\nu(\cdot)$ une mesure de Borel sur $(\chi\,;\,d)$ espace métrique complet séparable, vérifiant $\nu(\chi) > 0$ et $\nu(A)<\infty$ pour tout borélien A borné de χ.

Un processus ponctuel de mesure de l'intensité ν est un processus ponctuel de poisson si :

- $\forall$ A borélien borné, $N(A)$ suit une loi de Poisson de paramètre $\nu(A)$;
- $\forall A_1, \ldots, A_k$ boréliens bornés disjoins $N(A_1), \ldots, N(A_k)$ sont indépendantes.

Processus ponctuels de Poisson homogène

Nous disons qu'un processus ponctuel X de Poisson est homogène d'intensité $\lambda > 0$ si :

- $\forall$ A borélien borné, $N(A)$ suit une loi de Poisson de paramètre $\lambda\mu(A)$;
- $\forall A_1, \ldots, A_k$ boréliens bornés disjoins $N(A_1), \ldots, N(A_k)$ sont indépendantes.

Processus ponctuels de Poisson inhomogène

Nous disons q'un processus ponctuel X de Poisson est inhomogène si :

- $\forall$ A borélien borné, $N(A)$ suit une loi de poisson de paramètre $\nu(A)$;
- $\forall A_1, \ldots, A_k$ boréliens bornés disjoins $N(A_1), \ldots, N(A_k)$ sont indépendantes.

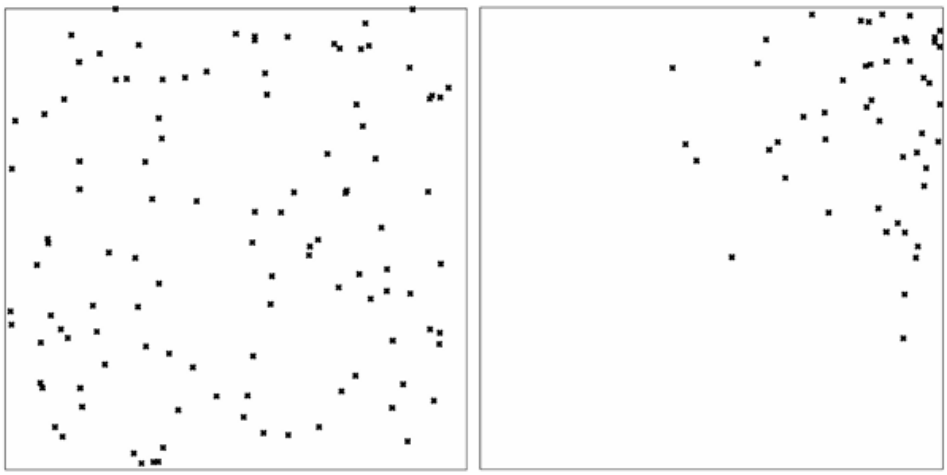

Figure 2.1 – Processus homogène (gauche) et processus inhomogène (droite)

2.3.2 Processus Binomial

Dans un processus ponctuel binomial, chaque événement est considéré comme un "succès" ou un "échec". Ce type de processus ponctuel est définit comme union $X = \{X_1, \ldots, X_n\}$ d'un nombre fixe $n \in \mathbb{N}$ de points $X_1, \ldots, X_n$ qui sont indépendant et distribué de manière uniforme, étant donné que $P(X_i = X_j = 0)$ pour tout $i \neq j$. En outre, étant donné que $P(N(x) = n) = 1$ et $P_m = \sigma_n^m$ et $j_n(x_1, x_2, \ldots, x_n) = (\frac{1}{\mu(\chi)})^n$ du fait que les points X_i sont uniformément distribués alors :

$$N(A) = \sum 1\{X_i \in \mathbb{A}\}$$

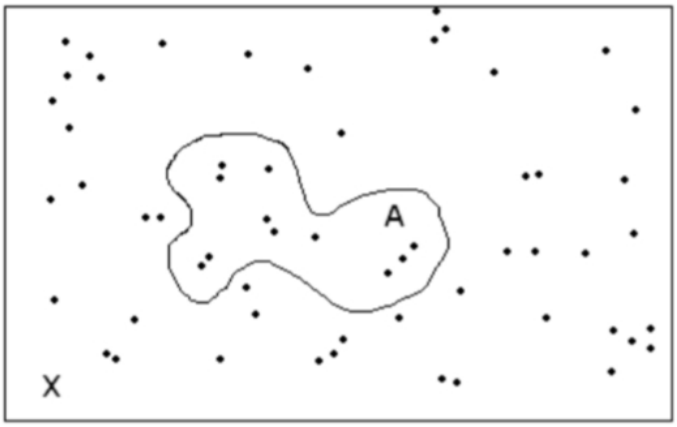

Figure 2.2 – Processus ponctuel Binomial

2.3.3 Processus de Cox

Un processus de Cox est un processus stochastique généralisant le processus de Poisson dans lequel la moyenne n'est pas constante mais varie dans l'espace ou le temps.

Prenons l'exemple d'un processus X sur l'espace mesuré (χ, v) avec une intensité μ stochastique. Si X est un processus ponctuel inhomogène de Poisson conditionnellement à μ, nous le nommons processus de Cox ou processus de Poisson.

Avec une double stochastique. Les processus temporels de Cox ne nécessitent pas de mémoire car ils sont d'intensité stochastique et ne font pas référence au passé.

2.3.4 Processus de Strauss

Selon Ripley et Kelly [Ripley and Kelly, 1977], le processus de Strauss est pris en compte avec une interaction de type répulsion dans le domaine $[0, 1]^p$. La loi $\pi(\cdot)$ de ce processus est définie en fonction du nombre n de points, ce qui est représentée par :

$$\pi(\mathbf{x}) = k\gamma^{s(\mathbf{x})}$$

Avec, $0 < \gamma < 1$ est considéré comme un coefficient de répulsion, k est une constante de normalisation et la fonction $s(\mathbf{x})$ est considérée comme un potentiel global d'énergie.

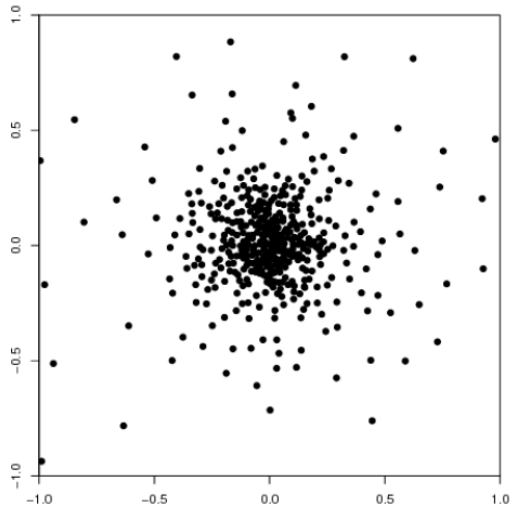

Figure 2.3 – Résultat d'une simulation à partir d'un processus de Strauss à échelle locale sur [-1, 1].

2.3.5 Processus de Hawkes

Les processus de Hawkes sont des processus stochastiques qui ont des applications dans de différents domaines comme la sismologie, la finance et les réseaux sociaux. Un processus de Hawkes, de manière informelle, est un processus ponctuel sur R qui a une intensité stochastique λ telle que $\lambda(t)$ dépend du processus restreint à $]-\infty, t[$. En d'autres termes, si un point du processus se trouve

à un instant t, ce point va influencer les valeurs de l'intensité après t. Deux concepts de processus de Hawkes seront examinés : Les processus de Hawkes dont la dynamique est négligée sur R, ainsi que ceux dont la dynamique est négligée sur R_+ et dont le comportement sur R_- sera imposé par ce qu'on nommera une condition initiale.

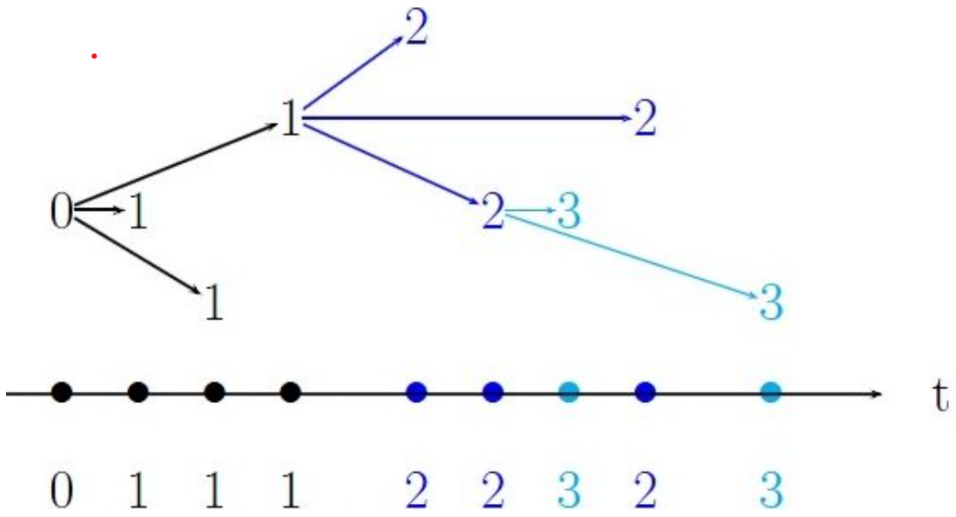

Figure 2.4 – Illustration du processus de branchement

2.4 Processus ponctuels marqués

Afin de décrire des objets plutôt que des points, il suffit simplement d'ajouter à chaque point une "marque" qui exprime les caractéristiques de l'objet (sa taille, son orientation, sa forme, etc.). Selon Daley et Vere-Jones [et D. Vere-Jones., 2002], ce processus est désigné sous le nom de processus ponctuels marqués ou processus d'objets.

Voici quelques exemples de processus ponctuels marqués :

- Répartition des arbres dans une forêt.
- Répartition des étoiles dans une galaxie.
- Répartition des bâtiments dans une ville.
- Répartition des clients dans un magasin.

Définition 2.5. *Un processus ponctuel marqué peut être représenté par une série de paires (x_i, m_i) où x_i représente la position spatiale du point et m_i sa marque associée.*

Remarque 2.3. *En règle générale, les points x_i sont définis dans un espace avec des dimensions finies ou infinies, tandis que les marques m_i peuvent être définies dans un espace donné.*

Exemple 2.3. Voici un exemple de processus ponctuel marqué, où il y a deux types de marques :

une marque correspondant à un processus ponctuel sur $K = \mathbb{R}$ (représenté par l'intermédiaire des différents plans) et une marque correspondant à un processus ponctuel sur $\chi = \mathbb{R}^2$ (représenté par les points sur chaque plan associé à une marque).

Il est possible d'obtenir le processus non marqué en projectant les points spatiaux de chaque marque sur un plan de référence, tel que décrit par $P \times \{0\}$ dans la Figure 2.5. En évitant de charger le schéma, les projections des différents points marqués ne sont pas toutes exactes.

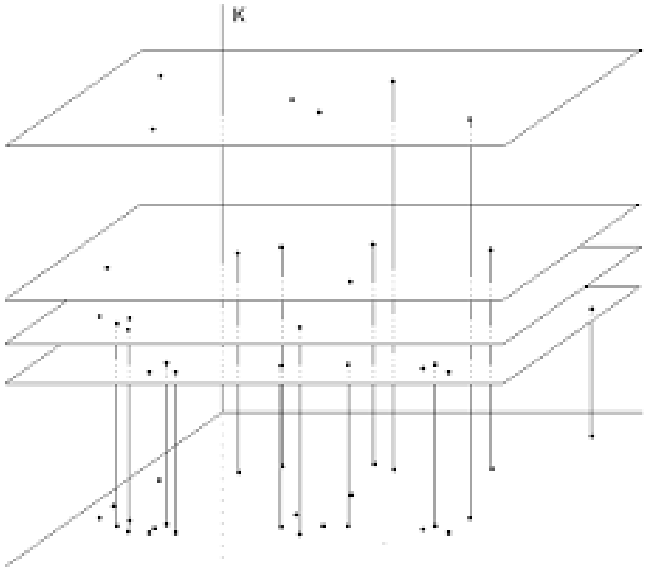

Figure 2.5 – Exemple d'un processus ponctuel marqué

2.5 Processus ponctuels de Markov

Une classe intéressante de processus ponctuels est celle des processus ponctuels de Markov. Ils regroupent les processus ponctuels finis définis par une densité pouvant s'écrire sous forme énergétique comme une somme de potentiels d'interactions. Ce sont les plus utilisés en traitement des images puisqu'ils permettent de modéliser les interactions entre les objets du processus, et qu'ils sont facilement programmables. Cette notion de processus ponctuel de Markov a été introduite par Ripley et Kelly . Sa généralisation à la propriété de Markov aux plus proches voisins est due à Baddeley et Moller .

2.5.1 Propriété de Markov au sens de Ripley-Kelly

Nous sommes maintenant en mesure de définir le processus ponctuel de Markov [et F. P. Kelly., 1977]. Définissons d'abord la notion de voisinage suivante :

Définition 2.6. *Soit $\sim$ une relation binaire symétrique et réflexive sur χ. Deux points u et v sont voisins si $u \sim v$. Le voisinage de $A \subset \chi$ est donné par :*

$$\partial(A) = \{u \in \chi \text{ et } u \notin A : \exists\, v \in A \text{ tel que } v \sim u\}$$

Nous notons $\partial\{u\} = \partial u$ pour $u \in \chi$. La propriété de Markov au sens de Ripley-Kelly est la suivante :

Définition 2.7. *Soit X un processus ponctuel de densité f par rapport à un processus ponctuel de Poisson de loi $\pi_v(\cdot)$ et d'intensité $(\cdot)$. Le processus X est markovien pour la relation $\sim$ si, pour toute configuration $x \in N^{lf}$, nous avons* :

1. $f(x) > 0$ implique $f(y) > 0$ pour tout $y \subset x$ (Nous disons que $f(\cdot)$ est héréditaire).
2. Si $f(x) > 0$, alors : $\lambda(u, x) = \frac{f(x \cup \{u\})}{f(x)}$ ne dépend que de u et de $\partial u \cap x$. Où
   $\partial u \cap x = \{v \in x : v \sim u\}$.

La condition (1) traduit que si une configuration peut se produire, alors toutes les sous configurations qu'elle contient peuvent se produire aussi. Le quotient $\lambda(u, x)$ dans la condition (2) appelé l'intensité conditionnelle de Papangelou [**?**], est la densité de probabilité qu'il ait un point u sachant que x est réalisé ailleurs. Cette condition exprime une propriété de Markov locale : le comportement d'un point u par rapport à la configuration entière ne dépend que de ses proches voisins dans cette configuration.

Nous appelons une clique, une configuration dont tous les points sont voisins les uns des autres par rapport à une relation symétrique et réflexive notée $\sim$. L'ordre dune clique est le nombre d'objets qu'elle contient. Le théorème suivant, équivalent de celui d'Hammersley-Clifford [Van Lieshout, 2000] pour les processus ponctuels, permet d'exprimer la densité d'un processus de Markov sous une forme énergétique décomposée sur les cliques de la configuration x :

Théorème 2.2.(Hammersley - Clifford).
Une densité de processus ponctuel $fE^f \to [0, +\infty[$ est markovienne sous une relation de voisinage $\sim$ si et seulement s'il existe une fonction mesurable $\varphi : N^f \to [0, +\infty[$ telle que :

$$f(x) = \alpha \prod_{cliquesy \subseteq x} \phi(y)$$

pour tout $x \in E^f$. $\phi(y)$ sont les potentiels d'interaction.

Ce résultat permet de construire facilement des modèles markoviens, par exemple à interaction de paires :

$$f(x) = \alpha \prod_i \beta(x_i) \prod_{x_i \sim x_j, i<j} \gamma(x_i, x_j)$$

Où, α est une constante de normalisation qui fait de f une densité.

2.5.2 Propriété de Markov pour un processus ponctuel marqué

La définition de la propriété de Markov tout comme le théorème de Hammersley-Clifford restent inchangés si $Y = (X, M)$ est un processus ponctuel marqué sur $E^{lf} \times K$ pour $\sim$, une relation de voisinage symétrique sur $E^{lf} \times K$. Par exemple si (X, M) est à marques indépendantes et si X est markovien pour une relation de voisinage $\sim$ sur est un processus ponctuel marqué markovien pour la relation $(x, m) \sim (y, o) \iff x \sim y$ sur E^{lf}. Un exemple de modèle isotrope à interaction de paires et à nombre fini de marques $M = 1, 2, ..., K$ est donné par la densité en $y = \{(x_i, m_i)\}$:

$$f(x) = \alpha \prod_i \beta_{m_i} \prod_{x_i \sim x_j, i<j} \gamma_{x_i, x_j} ||(x_i - x_j)||$$

Si, l'interaction $\gamma_{kl}(d) = 1$ pour $d > r_{kl}$, où les réels $r_{kl} > 0, k, l \in K, k \neq l$, sont fixés. Les deux conditions de la définition 2.7 sont vérifiées pour la relation de voisinage $\sim$.

$$(x, m) \sim (x', m') \iff ||x - x'|| \leq r_{m,m'}$$

Donc, Y est un processus ponctuel Markovien.

2.6 Chaines de Markov

Toute suite de variables aléatoires $(X_N)_{N \in \mathbb{N}}$ avec des valeurs dans Ω, telle que, pour tout entier $k \in \mathbb{N}$ et $A \in \mathbb{A}$, est appelée chaîne de Markov à temps discret. Cette suite respecte la propriété markovienne suivante :

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

Autrement dit, la valeur d'une variable aléatoire de cette suite ne dépend que de celle qui la précède. Nous nous intéresserons ici aux chaînes dites homogènes, c'est-à-dire aux chaînes dont l'évolution ne dépend pas de la position dans la chaine, mais seulement de l'état actuel :

$$P(X_{t_1}, X_{t_2}, \ldots, X_{t_k}/X_{t_0}) = P(X_{t_1-t_0}, X_{t_2-t_0}, \ldots, X_{t_k-t_0}/X_{t_0})$$

D'un point de vue informatique, une telle chaine présente bien entendu l'avantage de rendre inutile le rappel de l'ensemble des configurations antérieures puisqu'elle utilise uniquement l'état actuel pour générer une nouvelle configuration. La génération de la nouvelle configuration réclame la définition d'un noyau de transition. Un noyau de transition est une fonction $P : \chi \times A \rightarrow [0,1]$, tel que :

- Pour tout $A \in \mathbb{A}$, $P(\cdot, A)$ est mesurable.
- Pour tout $X \in \chi$, la fonction $P(X, \cdot)$ est une mesure de probabilité.

Ainsi, une chaîne de Markov homogène est complètement définie par la valeur ou la distribution de X_0 et son noyau de transition P. Chaque fois que nous sommes en mesure de construire une transition P, telle que, pour toute loi initiale $X_0 = v, \nu P^k \rightarrow \pi$. Avant de présenter les différents résultats théoriques et algorithmes de simulation, rappelons quelques définitions préliminaires des chaînes de Markov lors de la simulation.

- **Invariance :**

  Une loi π est invariante pour la chaine de Markov si :

  $$\pi = \pi P$$

  Cette condition est nécessaire pour obtenir la convergence de la chaine vers π.

- **Réversibilité :**

  La chaine est réversible pour π si le noyau de transition P vérifie :

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

  Cette condition implique l'invariance pour π, et signifie que sous la distribution stationnaire π la probabilité de passer de A à B est la même que de passer de B à A. La plupart des algorithmes de simulation sont en réalité construits pour produire des chaines de Markov réversibles [Geyer and Møller, 1994, Moller, 1999].

- **Irréductible :**

  Nous qualifions la chaîne de π-irréductible si :

$$\forall x \in \Omega \text{ et } \forall A \in \mathbb{A} \text{ tel que } \pi(A) > 0, \exists\ t \text{ tel que } {}^{t}p(x, A) > 0$$

Cela signifie que la chaine a une probabilité non nulle d'atteindre en temps fini tout ensemble π-probable. Cette condition est clairement nécessaire pour que la chaine converge en distribution vers π avec n'importe quelle condition initiale. Dans le cadre de ce projet, nous verrons uniquement des chaînes irréductibles, c'est-à-dire des chaînes ne possédant qu'une seule classe d'états.

- **Apériodicité :**

  A périodicité assure que les déplacements entre états n'ont pas trop de contraintes. Formellement, la chaine est apériodique s'il n'existe de partition disjointe $A = U_{i=0}^{k} A_i$ pour $r \geq 2$ comme suit :

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

  Il existe une relation entre l'irréductibilité et l'apériodicité. En effet, si une chaîne de Markov est irréductible et qu'il existe un état X_k tel que la probabilité $P(X_k, X_k)$ soit strictement positive, alors la chaîne de Markov est fortement apériodique. Donc tous les processus pour lesquels il est possible de rester dans un état sont fortement apériodiques (c'est le cas des algorithmes de type Metropolis-Hasting que nous étudierons plus loin).
- Une chaine π-irréductible et π-invariante est récurrente positive si, $\forall A \in A\ t.q\ \pi(A) > 0$, elle vérifie :

  $$\forall x, P_x\{X \in A\ i.s\} > 0\ et\ P_x\{X \in A\ i.s\} = 1$$

  (i.s : infiniment souvent)
- Sur un espace discret le noyau de transition P est primitif (régulier) si $\exists k \geq 1$ tel que P^k a tous ses termes strictement positifs.

2.6.1 Convergence d'une Chaine de Markov

Avant de présenter les méthodes de simulation des chaînes MCMC, nous présentons les conditions nécessaires pour qu'une chaîne puisse converger et atteindre la distribution recherchée π. Si, P est π-irréductible, π-invariante, récurrente. Dans ce cas, la mesure invariante est unique et la chaine est dite récurrente positive si la masse totale de cette mesure est finie. C'est les cas si π est une probabilité. Nous obtenons les résultats d'ergodicité [Chib and Greenberg, 1996] :

Proposition 2.1 : *Si, P est π-irréductible, et π-invariante, alors P est récurrente positive et π*

est l'unique loi invariante de P. Si P est apériodique, alors $x \in \chi$, nous avons :

$$||p^m(c,\cdot) - \pi|| \to 0 \; quand \; m \to \infty$$

Le contrôle de la convergence vers 0 de $||p^m(c,\cdot) - \pi|| \to 0$ est une question centrale et très difficile. Ce contrôle permet en effet d'assurer que νp^n pour m assez grand réalise une simulation acceptable de π. Il existe de nombreux résultats théoriques. Certains utilisent des minorations de P sur un petit ensemble E. Le coefficient de contraction sur un espace d'états fini permet le contrôle de cette convergence [Meyn and Tweedie, 2012].

- **Ergodicité géométrique, egodicité uniforme**

  L'ergodicité géométrique est caractérisée par :

$$\|P^t(x,.) - \pi\| < Mr(x)l^m$$

Où $M(x)$ est π- intégrable et $l < 1$. L'ergodicité est uniforme si nous pouvons choisir pour M une constante finie.

- **Coefficient de contraction**

Dans un espace d'états fini, le coefficient de contraction pour un noyau de transition P est donné par [Dobrushin, 1956] :

$$C(P) = \frac{1}{2} \; max||P(x,\cdot) - P(y,\cdot)||$$

Lemme 2.1 : Soient ν et μ deux distributions, P et Q deux noyaux de transition. Alors

$$||\mu P - \nu P|| \leq ||\mu - \nu||C(P) \; et \; C(PQ) \leq C(P)C(Q)$$

En particulier,

$$||\mu P - \nu P|| \leq ||\mu - \nu|| \; et \; ||\mu P - \nu P|| \leq 2C(P)$$

Et si P est primitif alors $C(P) < 1$.

D'après ce dernier résultat, si nous prenons $\mu = \pi$, la loi invariante de P, Nous en déduirons :

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

Dans ce cas, quand $m \to \infty$ la chaine est uniformément ergodique si P est primitif.
Signalons à la fin de cette section, que ces dernières conditions, délicates à obtenir en général, sont satisfaites pour les deux principales familles de simulateurs : l'algorithme de Metropolis-Hastings et l'algorithme de Gibbs, l'objectif de la section suivante.

2.7 MCMC - Metropolis Hasting

2.7.1 Méthodes de Monte-Carlo par chaînes de Markov

Les méthodes de Monte-Carlo par chaînes de Markov (MCMC) offrent la possibilité de simuler une distribution en utilisant une chaîne de Markov ergodique qui a une distribution stationnaire en tant que distribution. Afin de créer un algorithme de cette nature, il est donc nécessaire de définir un ensemble de probabilités de transition P adéquat, c'est-à-dire irréductible, ergodique et ayant une distribution stationnaire appropriée.

Définition 2.8. (MCMC).
Nous appellerons MCMC toute méthode permettant de simuler une distribution en utilisant une chaîne de Markov ergodique $(X_N)_{N\in\mathbb{N}}$ ayant celle-ci comme distribution stationnaire

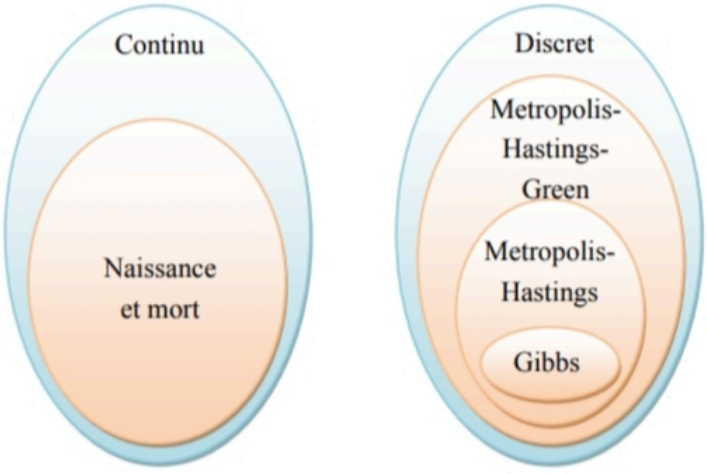

Figure 2.6 – Les familles d'algorithme pour la simulation des chaînes MCMC

2.7.2 Metropolis Hasting général

En statistique, l'algorithme de Metropolis-Hasting est une méthode MCMC dont le but est d'obtenir un échantillonnage aléatoire d'une distribution de probabilité quand l'échantillonnage direct en est difficile.

L'algorithme Metropolis a été introduit par Metropolis, et al. en 1953 [Metropolis, 1953] et généralisé par Hastings en 1970 [Hastings, 1970a]. L'algorithme repose sur la proposition d'un nouvel état en perturbant légèrement l'état actuel, puis nous pouvons l'accepter ou le refuser. Le coefficient d'acceptation $R(x, y)$ correspond à la probabilité que la chaîne passe d'un état actuel x à l'état y. Le modèle suggéré par Geyer et Moller crée une chaîne de Markov capable d'explorer toutes les configurations de l'espace χ. La naissance et la mort sont les perturbations suggérées (ajout ou suppression d'un élément de la configuration actuelle). De manière générale, l'algorithme se sert d'une transition P qui est réversible à π et invariante à π. La création de P est réalisée en deux phases :

- Transition de proposition de changement : nous commençons par proposer un changement x $\rightarrow$ y avec une probabilité $Q(x, dy)$.
- Probabilité d'acceptation du changement : ensuite, nous acceptons ce changement avec la probabilité $a(x, y)$ où $a : \Omega \text{ Œ } \Omega \rightarrow]0,1]$. Si non nous restons en x.

Les deux paramètres de l'algorithme sont Q, la transition de proposition de changement et a la probabilité d'accepter ce changement. Si γ_x(dy) est la mesure de Dirac en x la transition P de MH s'écrit :

$$P(x, dy) = a(x, y)Q(x, dy) + \gamma_x(dy) \int 1 - a(x, z)Q(x, dz)dz$$

Notons que : $\nu(dx, dy) = \mu(dx, dy) + \mu(dy, dx)$, où $\mu(dx, dy) = Q(x, dy)\pi(dx)$ et μ admet une densité $h(x, y)$ par rapport à ν.

Le ratio r de Metropolis-Hastings est défini par :

$$r(x, y) = \frac{h(x, y)}{h(y, x)} \; sur \; R = \{(x, y) \; h(x, y) > 0 \; et \; h(y, x) > 0\}$$

La transition P est π-réversible si et seulement si la probabilité d'acceptation a vérifie :

$$a(x, y)r(x, y) = a(y, x)$$

Si Q et π sont respectivement à densité q et π, et si $q(x,y) > 0$ équivaut $q(x,y) > 0$, alors la π-réversibilité de P s'écrit alors :

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

Et, si Q est symétrique, l'algorithme est le suivant :

Algorithme

- En partant d'un état initial $x_0 = x$, Choisissez un état y en fonction de $Q(x, \cdot)$.
- Choisissez un noyau de proposition Q ayant une probabilité p.
- Effectuer la génération de $y \sim Q(x, \cdot)$.
- Évaluer le rapport d'acceptation $a(x,y) = \frac{\pi(y)}{\pi(x)}$.
- Accepter l'état y en tenant compte de la probabilité $a = min(1, a)$.
- Recommencer avec l'état actuel.

2.7.3 L'échantillonneur Gibbs

L'échantillonneur de Gibbs est en fait un cas particulier d'un algorithme Metropolis-Hastings (MH). Dans le cas de l'échantillonneur de Gibbs, nous utilisons une méthode de MH simplifiée qui est plus efficace lorsque l'on veut échantillonner à partir d'une distribution jointe en utilisant des distributions conditionnelles simples.

L'échantillonneur de Gibbs est efficace lorsque les distributions conditionnelles sont connues et relativement simples à échantillonner. Dans ce cas, chaque étape d'échantillonnage est acceptée avec une probabilité de 1, ce qui signifie que l'acceptation ou le rejet explicite des échantillons n'est pas nécessaire, contrairement à l'algorithme de Metropolis-Hastings général.

La probabilité d'acceptation pour la dynamique de Metropolis est alors donnée par :

$$a(x,y) = min\left(1, \frac{\pi(y) \times q_i(y, x_i)}{\pi(x) \times q_i(x, y_i)}\right)$$

tel que : i un site choisit au hasard et y_i suivant une transition de densité $q_i(x, y_i)$.

Voici l'algorithme Gibbs à partir d'une configuration $X = \{x_1, \ldots, x_n\}$:

- Choisir un objet à modifier $x_j \in \mathbb{X}$
- Générer une nouvelle valeur x_j selon $\pi\{x_1, \ldots, x_{j-1}, \ldots, x_{j+1}, \ldots, x_n\}$
- Recommencer avec la nouvelle configuration $X = \{x_1, \ldots, xj, \ldots, x_n\}$

Chapitre 3

NOUVEAUX PLANS D'EXPÉRIENCES NUMÉRIQUES À PARTIR DE PROCESSUS PONCTUELS MARQUÉS

Dans ce chapitre, nous examinons comment les Processus Ponctuels Marqués (PPM) sont utilisés pour simuler les n expériences qui composent les plans d'expériences suggérés. À la différence de l'approche basée sur les processus ponctuels [Franco, 2008], les PPM nous offrent la possibilité d'acquérir non seulement des connaissances géométriques, mais aussi des connaissances préalables sur les points. Il est possible de prendre en compte les processus caractérisés de Strauss [Strauss, 1975] qui englobent la notion d'interaction entre paire de points. Afin de produire de tels plans, nous ferons appel aux méthodes de simulation par Chaine de Markov (MCMC), plus spécifiquement à l'algorithme de Métropolis-Hasting [Chib and Greenberg, 1995b, Hastings, 1970b].

3.1 Plan d'expériences numériques par processus ponctuels markoviens marqués de Strauss à trois marques

L'idée fondamentale consiste à prendre chaque expérience x_i comme un point ou une particule défini sur $[0,1]^p$, et chaque configuration x comme une matrice d'expériences où chaque point de cette configuration sera caractérisé par trois marques m_i et m'_i et m''_i définies sur l'espace de marques M. Les trois marques et le point constituent un objet défini par (x_i, m_i, m'_i,m''_i). Les n objets

(plan d'expériences) sont donc comparés à la réalisation d'un processus ponctuel défini par X. Les processus utilisés par Straus incluent des possibilités d'interaction. Nous considérons ces interactions comme des caractéristiques de voisinage telles qu'elles sont définies par un champ de Markov au sens de Ripley-Kelly [Hawkes, 1971]. L'interaction entre paires d'objets est le potentiel d'interaction le plus couramment utilisé. La modélisation des phénomènes répulsifs nécessite l'utilisation de ces processus objets. La densité de probabilité d'un processus ponctuel marqué de Strauss pour une configuration x de points est donnée par :

$$\pi(x) = \alpha \prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j \geq i}}^{3} \gamma_{ij}^{m_{ij}(x)} \tag{3.1}$$

Avec :

- α est la constante de normalisation,
- $0 < \gamma_{kl} \leq 2$, k $\in\{1, 2, 3\}$ et l $\in\{1, 2, 3\}$ sont des coefficients d'interaction,
- β_k,k $\in\{1, 2, 3\}$ l'intensité du processus,
- $m_k(x)$ est le nombre de points de type k dans x (de marque k),
- $m_{kl}(x)$ est le nombre de paires de voisins par rapport à la relation $\sim$.

3.1.1 Choix des marques

Dans cette étude, nous identifions les points en utilisant trois marques distinctives : la première sera la valeur de l'erreur de prédiction $\hat{y}_{x_i}$ au point x_i. Nous nous souvenons que cette valeur est déterminée par :

$$var(\hat{y}_{x_i}) = f(x_i)^t ({}^tXX)^{-1} f(x_i)$$

Avec,

- $X = [f(x_1), \ldots, f(x_i)\ ]^t$ est la matrice de calcul, qui dépend des points expérimentaux choisis et du modèle postulé,
- $(X^tX)^{-1}$ la matrice de dispersion,
- $f(x_i)$ est le vecteur modélisé du point x_i

Dans ce cas nous définissons $m_1(x)$ pour une configuration x par :

$$m_1(x) = \sum_{i=1}^{n} 1_{var(\hat{y}_{x_i})` \leq \epsilon}$$

Comme deuxième marque, nous prenons la moyenne des distances à densité normal entre le point x_i et les autres points de la configuration x. Cette marque sera donnée par :

$$m_2(x) = \sum_{i=1}^{n} 1_{\mu(x_i) \geq r}$$

Tel que $\mu(x_i) = \frac{1}{n-1} \sum_{j=1}^{n-1} \gamma(x_i, x_j)$ avec $\gamma(x_i, x_j) = \int_0^l \phi(t)dt$, tel que l est la distance usuel entre les points x_i et x_j. Et ϕ la densité de la loi normale où ϵ et r sont des valeurs fixées.

Pour la troisième marque, nous considérons le minimum des distances entre les points. Cette marque est donnée par :

$$m_3(x) = \sum_{i=1}^{n} 1_{\lambda(x_i) \leq r} \tag{3.2}$$

Tel que $\lambda(x_i) = min||x_i - x_j||$

3.1.2 Simulation des processus ponctuels par la méthode MCMC et l'algorithme de Metropolis-Hasting

Il s'agit de créer une chaîne de configurations $\{X_0, X_1, \ldots, X_N\}$ qui converge vers la distribution recherchée π donnée par (3.1). Cette construction est possible grâce à l'algorithme de Metropolis Hasting, qui utilise un noyau de transition P_{MH} qui est π-réversible. Cet algorithme suit deux étapes :

- Nous suggérons une transition d'état de x à y d'après la loi de probabilité $Q(x, \cdot\ )$,
- Nous acceptons y avec la probabilité $a(x, y)$, sinon nous restons dans l'état x (Où $a : \Omega$ Œ Ω $\rightarrow$ [0, 1]).

Notons $q(x, y)$ la densité de $Q(x, \cdot)$, la transition P_{MH} s'écrit [Meyn and Tweedie, 2012, Chib and Greenberg, 1

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

Avec $\delta_x(\cdot)$ est la masse du point en x. Afin de faciliter les calculs, la mesure de Dirac en x est utilisée. Le choix de (Q, a) assurera la π-réversibilité de P_{MH} si l'équation d'équilibre suivante est satisfaite :

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

Le choix de la probabilité d'acceptation a est plus limité : il est dicté essentiellement par l'objectif de simuler (asymptotiquement) une loi de probabilité π donnée. C'est le cas du choix usuel, où :

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

Deux éléments essentiels à souligner. Tout d'abord, le calcul de $a(x, y)$ ne nécessite pas la connaissance de la constante de normalisation de (3.1). Ensuite, dans cette étude, nous prenons en compte le cas où deux configurations x et y sont différentes d'un point précis. La dynamique de renversement de spin (en anglais, spin flop Dynamics) est alors connue, ce qui signifie que la densité q est symétrique :

$$q(y, x) = q(x, y),$$

Si tel est le cas, la probabilité d'acceptation est de :

$$a(x, y) = \frac{\prod_{i=1}^{3} \beta_i^{m_i(y)} \prod_{\substack{j=1 \\ j \geq i}}^{3} \gamma_{ij}^{m_{ij}(y)}}{\prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j \geq i}}^{3} \gamma_{ij}^{m_{ij}(x)}}$$

3.2 L'algorithme de construction du plan proposé

Les plans d'expériences numériques proposé dans ce travail [appelé le plan d'expériences marqué à trois marques] sont générés à l'aide de l'algorithme suivant :

Algorithme.

- **Etape d'initialisation :** Choisir une configuration initiale (plan expérimental initial) ($X_0 = x$ ou $x = (x_1, x_2, \ldots, x_n)$ et $x \in [0,1]^p$) selon une distribution de probabilité donnée, par exemple la distribution uniforme.
- **Étape d'itération** :

for $N = 1, 2, ..., N_{MCMC}$ **do**

for *chaque configuration* x **do**

Échantillonner y en utilisant la dynamique de retournement de spin :

- Choisir un spin s uniformément au hasard parmi $\{1, \cdots, n\}$.
- Simuler une expérience y_j selon la distribution uniforme sur $[0,1]^p$. Prendre ensuite ceci comme nouvelle configuration : $y = (x_1, x_2, \ldots x_{s-1}, y_j, x_{s+1} \ldots, x_n)$.

end

- Calcul de la probabilité d'acceptation
  $a(x,y) = \min\left(1; \pi(x)a(x,y)q(x,y) = \alpha \prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j \geq i}}^{3} \gamma_{ij}^{m_{ij}(x)}\right)$.
- Prendre $x = \begin{cases} y \text{ avec une probabilité } a \\ x \text{ avec une probabilité } 1-a \end{cases}$.

Répéter ces deux dernières étapes n fois pour chaque itération N.

Prendre $X_N = x$

end

Exemple 3.1

Pour N=1000, la figure 3.1 montre la convergence vers une configuration qui caractérise la réalisation d'un processus ponctuel marqué de Strauss à trois marques à partir d'une configuration initiale de 20 points choisis uniformément sur $[0\ 1]^2$.

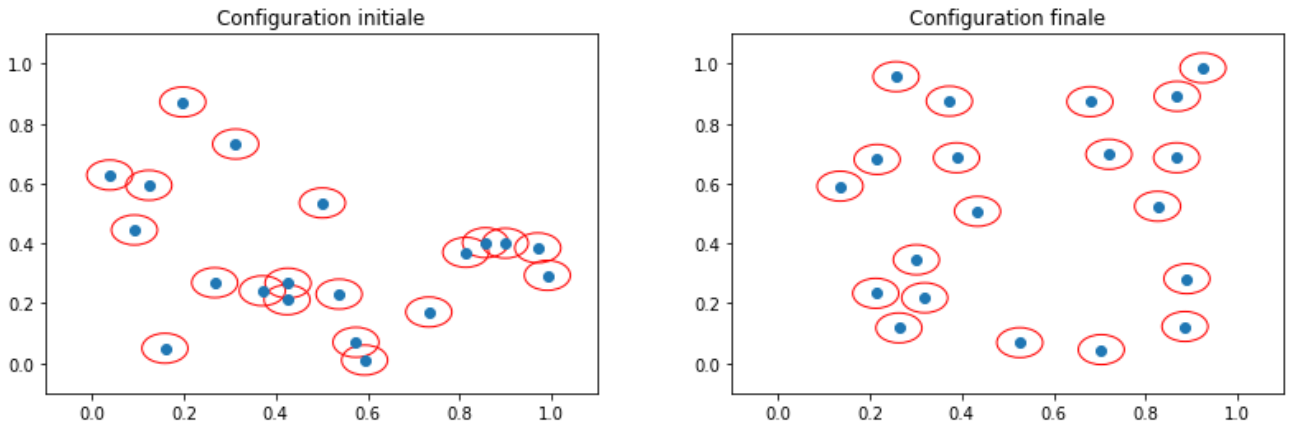

Figure 3.1 – A gauche, une configuration initiale aléatoire de 20 points et à droite, une configuration finale pour $\beta_1 = 0.5, \beta_2 = 0.5, \beta_3 = 0.5, \gamma_{11} = 0.1, \gamma_{12} = 0.02, \gamma_{13} = 0.05, \gamma_{23} = 0.2, \gamma_{22} = 0.02, \gamma_{33} = 0.08, R_1 = 0.05, R_2 = 0.08, R_3 = 0.05$ *et* $r = 0.1$

Exemple 3.2

Pour N=1000, la Figure 3.2 montre la convergence vers une configuration qui caractérise la réalisation d'un processus ponctuel marqué de Strauss à trois marques à partir d'une configuration initiale de 50 points choisis uniformément sur $[0\ 1]^2$.

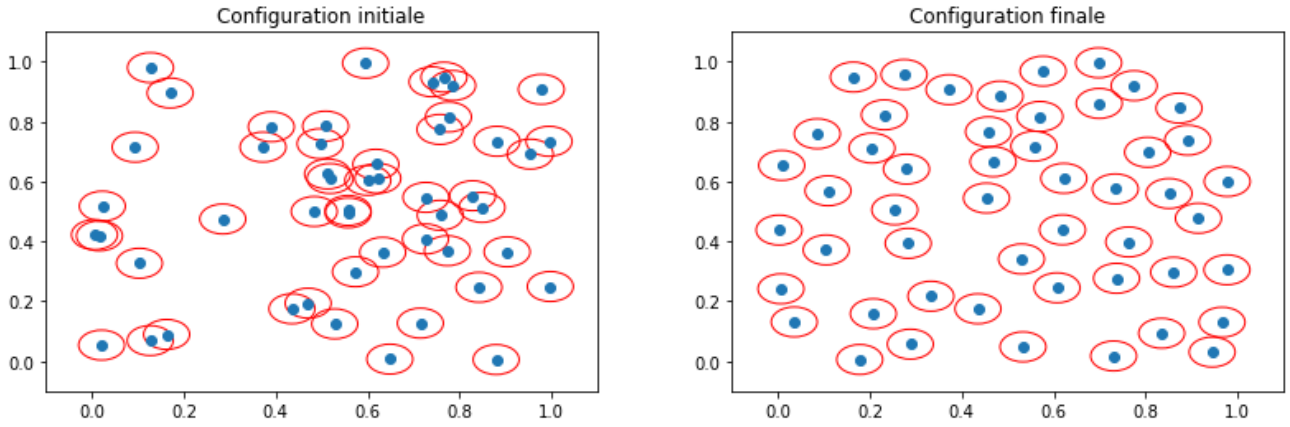

Figure 3.2 – A gauche, une configuration initiale aléatoire de 50 points et à droite, une configuration finale pour $\beta_1 = 0.5, \beta_2 = 0.5, \beta_3 = 0.5, \gamma_{11} = 0.1, \gamma_{12} = 0.02, \gamma_{13} = 0.05, \gamma_{23} = 0.2, \gamma_{22} = 0.02, \gamma_{33} = 0.08, R_1 = 0.07, R_2 = 0.09, R_3 = 0.04$ *et* $r = 0.1$

Remarques

- Sur la figure 3.1, nous matérialiserons les interactions entre expériences en traçant les cercles de rayon r , l'intersection de deux cercles correspond précisément à une interaction.
- Il est important de bien fixer le paramètre de répulsion qui est situé entre 0 et 1, nous avons constaté selon plusieurs exemples qu'il est facile de générer une distribution répondant au critère de remplissage de l'espace avec un paramètre de répulsion faible.
- Pour le choix du rayon r, un rayon trop petit engendre une distribution sans interaction. Par

contre, un rayon trop grand conduit à une distribution avec des agglomérats.

3.3 Etude de convergence

Nous réalisons n transformations de base pour chaque itération N de l'algorithme de construction mentionné précédemment. D'où la chaîne des plans d'expériences $(X_N)_{N\geq 0}$ ainsi générée est la réalisation d'une chaîne de Markov de noyau de transition :

$$P(x,y) = P_{MH}^{n}(x,y)$$

La question principale qui se pose à ce stade est de déterminer si la chaîne converge vers la distribution $\pi(x)$ définie en (3.1). La chaîne est convergente vers la distribution invariante π si :

$$P^{t}(x,A) \xrightarrow[t\to\infty]{} \pi(A)$$

Où A un borélien de $\mathbb{A}$ et $P^{t}(x,A) = P(X_t = A, X_0 = x)$ est un noyau de transition de pas t.

Récapitulons le résultat principal qui nous intéresse ici :

Proposition 3.1. *Sur un espace fini, le noyau de transition $P = P_{MH}^{n}$ de la chaîne de Markov $(X_N)N \leq 0$ obtenue à partir de l'algorithme de construction est $\pi-$irréductible et positivement récurrent. La distribution π est la seule distribution stationnaire de P et P étant un noyau apériodique et primitif.*

Preuve

Tout d'abord, nous montrons trois propriétés essentielles du noyau P_{MH} : π-réversibilité, π-stationnarité et π-irréductibilité.

- **La π-réversibilité** : Par définition, la transition P_{MH} est π -réversible si :

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

Soit $x \in \Omega$ et $B \in A$ nous avons :

$\int_\Omega 1_{B(x,y)}\pi(x)P_{MH}(x,y)dx$

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

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

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

Et puisque :

$$\pi(x)a(x,y)q(x,y) = \alpha \prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(x)} min\left(1, \prod_{i=1}^{3} \beta_i^{m_i(y)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(y)-m_{ij}(x)}\right) q(x,y)$$

$$= \alpha min\left(\prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(x)}, \prod_{i=1}^{3} \beta_i^{m_i(y)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(y)}\right) q(x,y)$$

$$= \alpha \prod_{i=1}^{3} \beta_i^{m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(x)} min\left(\prod_{i=1}^{3} \beta_i^{m_i(x)-m_i(y)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(x)-m_{ij}(y)}, 1\right) q(x,y)$$

$$= \pi(y) min(1; \left(\prod_{i=1}^{3} \beta_i^{m_i(x)-m_i(y)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(x)-m_{ij}(y)}, 1\right) q(x,y)$$

$$= \pi(y)a(y,x)q(x;y)$$

En prenant en considération que : $q(x,y) = q(y,x)$ donc :

$$\pi(x)a(x,y)q(x,y) = \pi(y)a(y,x)q(y,x)$$

Nous obtenons :

$\int_\Omega 1_{B(x,y)}\pi(x)P_{MH}(x,y)dx$

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

$= \int_\Omega 1_{B(x,y)}\pi(y)P_{MH}(y,x)dy$

Donc $\pi(x)P_{MH}(x,y) = \pi(y)P_{MH}(y,x)$, alors la chaine est π-réversible.

- **La π-stationnarité :** La transition P_{MH} est π-stationnaire si :

$$\forall x, y \in \Omega \; et \; B, A \in B : \int_\Omega 1_{B(x,y)}\pi(x)P_{MH}(x,A)dx = \int_\Omega 1_{B(x,y)}\pi(x)dx$$

Soit $x \in \Omega$ et $B \in A$. Nous avons alors :

$\int_\Omega 1_{B(x,y)}\pi(x)P_{MH}(x,y)dx$

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

$= \int_\Omega \int_\Omega 1_{B(x,y)}\pi(x)a(x,y)q(x,y)dydx + \int_\Omega 1_{B(x,x)}\pi(x) - \int_\Omega \int_\Omega \pi(x)a(x,z)q(x,z)dzdx$

$= \int_\Omega 1_{B(x,x)}\pi(x)$

Ainsi, la chaine admet π comme distribution stationnaire.

- **La π-irréductibilité** : La transition P_{MH} est π-irréductible si :

$$\forall A \in \mathcal{B} : \pi(A) > 0 \implies \exists t, P_{MH}^t(x,A) > 0$$

Soit A un borélien $\in \mathcal{B}$ et pour $t = 1$ nous obtenos :

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

vu que

$$a(x,A) = min\left(1, \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)}\right)$$

et

$$a(x,z) = min\left(1, \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)}\right)$$

Quatre cas sont donc possibles :

Si $a(x,A) = 1$ et $a(x,z) = 1$ alors :

$$\begin{aligned}\int_\Omega 1_{B(x,A)} P_{MH}(x,A)dx &= \int_\Omega 1_{B(x,A)} q(x,A)dx + 1 - \int_\Omega \int_\Omega q(x,z)dzdx \\ &= \int_\Omega 1_{B(x,A)} q(x,A)dx > 0\end{aligned}$$

Si $a(x,A) = 1$ et $a(x,z) = \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)}$ alors :

$\int_\Omega 1_{B(x,A)} P_{MH}(x,A)dx = \int_\Omega 1_{B(x,A)} q(x,A)dx + 1 - \int_\Omega \int_\Omega \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)}$
$q(x,z)dzdx$

$= \int_\Omega 1_{B(x,A)} q(x,A)dx + 1 - \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)} > 0$

Si $a(x,z) = 1$ et $a(x,A) = \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)}$ alors :

$\int_\Omega 1_{B(x,A)} P_{MH}(x,A)dx = \int_\Omega 1_{B(x,A)} \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)} q(x,A)dx + 1$
$- \int_\Omega \int_\Omega q(x,z)dzdx$

$= \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)} \int_\Omega 1_{B(x,A)} q(x,A)dx > 0$

Si $a(x,z) = \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)}$ et $a(x,A) = \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)}$

Alors :

$$\int_\Omega 1_{B(x,A)} P_{MH}(x,A)dx = \int_\Omega 1_{B(x,A)} \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)} q(x,A)dx + 1$$

$$- \int_\Omega \int_\Omega \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)} q(x,z)dzdx$$

$$= \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)} \int_\Omega 1_{B(x,A)} q(x,A)dx + 1 - \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)}$$

$$\int_\Omega \int_\Omega q(x,z)dzdx$$

$$= \prod_{i=1}^{3} \beta_i^{m_i(A)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(A)-m_{ij}(x)} \int_\Omega 1_{B(x,A)} q(x,A)dx + 1 - \prod_{i=1}^{3} \beta_i^{m_i(z)-m_i(x)} \prod_{\substack{j=1 \\ j\geq i}}^{3} \gamma_{ij}^{m_{ij}(z)-m_{ij}(x)} > 0$$

Alors P_{MH} est π-irréductible.

Étant donné que π est la loi invariante de P_{MH} , elle est également la loi invariante pour P_{MH}^n. Effectivement, $\pi P_{MH} = \pi$ et par récurrence sur l'entier n, nous pouvons obtenir :

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

Et, comme $P = P_{MH}^n$, alors nous obtenons $\pi P = \pi$. D'autre part π réversible de P_{MH} conduit à π réversible de P, c'est à dire :

$$\pi(x)P_{MH}(x,y) = \pi(y)P_{MH}(y,x) \implies \pi(x)P(x,y) = \pi(y)P(y,x)$$

Puisque $\pi P_{MH} = \pi P_{MH}^n = \pi$ alors nous obtenons :

$$\pi(x)P_{MH}(x,y) = \pi(x)P_{MH}^n(x,y) = \pi(y)P(y,x) = \pi(y)P_{MH}^n(y,x)$$

et puisque $P_{MH}^n = P$, alors nous obtenons : $\pi(x)P(x,y) = \pi(y)P(y,x)$

Donc $\pi P = \pi$. Par construction de P_{MH}^n, la π-irréductibilité de P_{MH} entraine la π-irréductibilité de P. Si P est π-irréductible et possède une distribution π invariante. Alors P est récurrent positive et π est l'unique distribution invariante de P (Proposition 2.1).

D'autre part, la chaîne générée par l'algorithme de construction doit également être apériodique, à condition qu'il existe au moins une paire de configurations (x,y) telle que $a(x,y) < 1$, et nous obtiendrons finalement $P(x,x) > 0$. Nous remarquons rapidement que la chaîne est apériodique puisque

l'événement $X_{(N+1)} = X_{(N)}$ est probable à presque tout moment. En fait, chaque état peut alors être visité à deux itérations successives, donc $P^1(x, x) > 0$, et leur période est alors égale à 1. Puisque la chaîne générée par l'algorithme est irréductible et apériodique, son noyau de transition P est alors primitif (une caractérisation plus courante du noyau de Markov primitif en théorie des probabilités est de dire qu'il est irréductible et apériodique [Senata, 1981]).

Théorème 3.1.

La chaine de markov $(X_k)_{k>0}$ obtenue à partir de l'algorithme de construction est uniformément ergodique et son noyau P réalise la simulation du processus ponctuel à trois marques de densité :

$$\pi(x) = \alpha\beta_1^{m_1(x)}\beta_2^{m_2(x)}\beta_3^{m_3(x)}\gamma_{11}^{m_{11}(x)}\gamma_{12}^{m_{12}(x)}\gamma_{13}^{m_{13}(x)}\gamma_{22}^{m_{22}(x)}\gamma_{23}^{m_{23}(x)}\gamma_{33}^{m_{33}(x)}$$

C.à.d υP^m tend vers π quand m tend vers l'infini, où υ est une distribution initiale et nous avons :

$$||\upsilon^m(x, .) - \pi|| \to 0, m \to \infty$$

Preuve

Pour tout entier m et $\forall x \in N^{lf}$, considérons υ comme une distribution initiale. Nous avons d'après lemme 2.1 :

$$||\upsilon p^m(x, .) - \pi|| = ||\upsilon p^m - \pi p^m|| \leq 2C(p^m) \leq 2(C(p^m))$$

Avec $0 \leq C(p) < 1$ est le coefficient de contraction Dobrushin [Dobrushin, 1956]. Donc la chaine est uniformément ergodique. Et, $||\upsilon^m(x, .) - \pi||$ tend vers zéro quand m tend vers l'infini. Donc, la chaine converge vers la distribution de processus marqué multiple définie en (3.1).

Chapitre 4

RESULTATS ET COMPARAISONS

Dans ce chapitre, nous présenterons les résultats numériques des différents critères d'optimalité afin d'évaluer la qualité des plans d'expériences numériques proposés dans ce mémoire.

4.1 Résultats numériques et qualités des plans proposés

Afin d'évaluer la qualité du plan d'expériences numérique proposé en utilisant des critères courants qui assurent un remplissage optimal de l'espace et une répartition homogène des points. Cette partie vise à évaluer les valeurs de ces critères présentés dans le premier chapitre, pour cela, nous employons trois catégories de critères :

- Critère de distance.
- Critère de recouvrement.
- Critère de discrépance.

Nous rappelons que :

- Le critère de distance : consiste à maximiser la distance minimale entre deux points du plan. Plus la valeur de ce critère est grande, plus les points seront éloignés les uns des autres.
- Le critère de recouvrement (Cov) : permet de mesurer l'écart entre les points du plan et ceux d'une grille régulière. Ce critère est nul pour une grille régulière. Le but est donc de minimiser le recouvrement pour se rapprocher d'une grille régulière, et ainsi assurer le remplissage de l'espace.
- Critère de discrépance : la discrépance permet de mesurer l'écart entre la fonction de répartition empirique des points du plan et celle de la loi uniforme. Plus la discrépance est faible, plus les points sont répartis uniformément. Il existe différentes mesures de discrépance. Nous

retenons la discrépance en norme L2.

4.2 Résultats de comparaison pour les suites à faible discrépance

Le tableau 4.1 présente une comparaison suivant le critère de discrépance entre les plans proposés dans ce travail, (notés THMD : Three Marked Strauss Designs), avec les suites de faible discrépance (séquence de Halton [Halton, 1960], séquence de Sobol [Sobol, 1976] et séquence de Faure [Faure, 1982]). Il est intéressant d'observer que les plans proposés ont une faible discrépance comparable à celle des suites à faible discrépance.

Table 4.1 – La valeur de la discrépance pour les plans proposés (THMD), les suites de Halton, les suites de Sobol et les suites de Faure pour quatre, sept et dix dimensions.

Dimension	Nombre de points	THMD	Séquence de Halton	Séquence de Sobol	Séquence de Faure
4	32	0,001817465	0,00177969	0,000843615	0,001567004
7	64	0,000104268	0,000480034	0,00022491	0,000480034
10	128	5,94766 $\times 10^{-}6$	0,000109686	6,05211$\times 10^{-}5$	0,000138046

4.3 Résultats de comparaison entre les plans stochastiques

Concernant les plans stochastiques, les critères ont été calculés sur 100 plans afin de donner un sens aux résultats. Ces plans sont :

- Plans aléatoires (RD)
- Hypercubes latins (LHS) [Loh, 1996]
- Plans Maximin LHS (mLHS) [Morris and Mitchell, 1995]
- Plans à entropie maximale (Dmax) [Shewry and Wynn, 1987]
- Plans de Strauss (SD) [Franco, 2008]
- Plans de Strauss marqué (MSD)
- Plan pour modèle à interaction de connexité (CCD) [Elmossaoui and Oukid, 2023]
- Plans proposés (THMD)
- Plans de Strauss de deux marques (TMD) [Elmossaoui et al., 2020]

4.4 Plans avec 20, 50 et 100 points en 5 dimensions.

Les figures ci-dessous présentent visuellement les critères les plus significatifs qui ont été calculés. Ces représentations graphiques aident à mieux comprendre et interpréter les résultats, en mettant en lumière la distribution et les variations observées pour chaque critère.

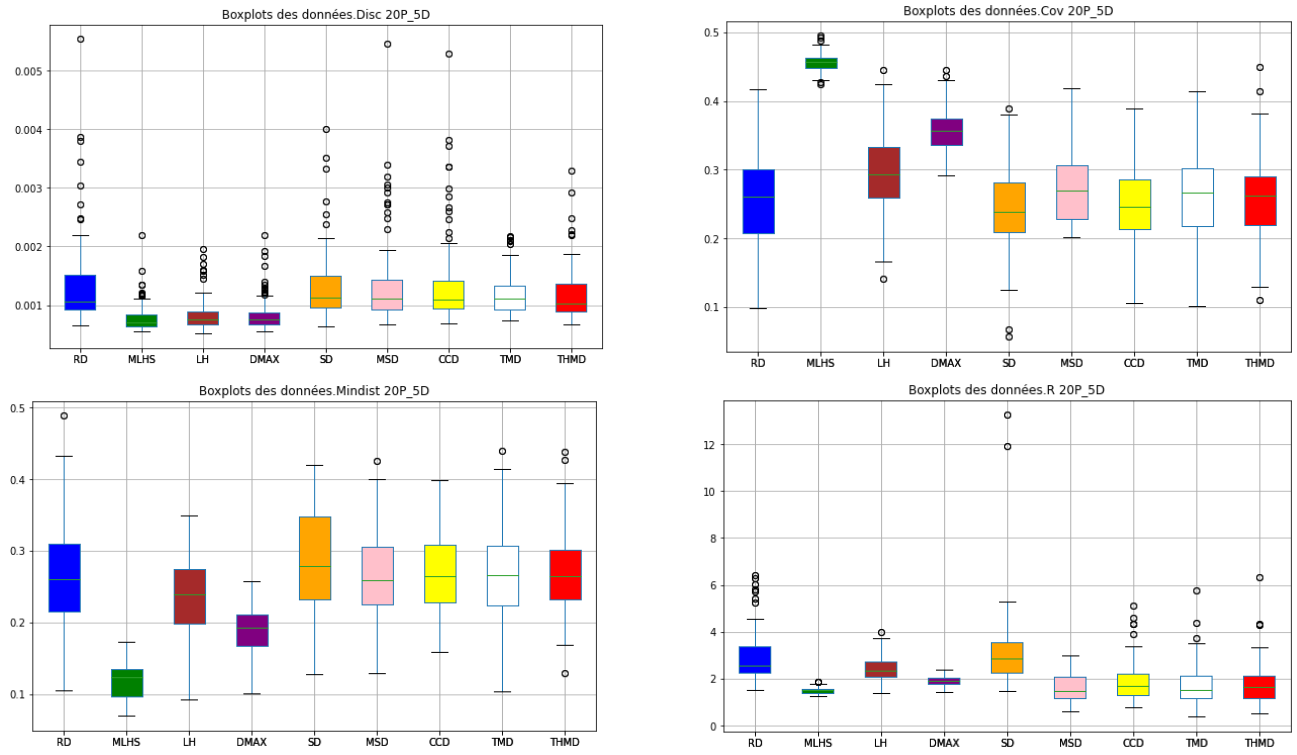

Figure 4.1 – Box plots des critères de qualité calculés sur les 100 plans à 20 points en dimension 5

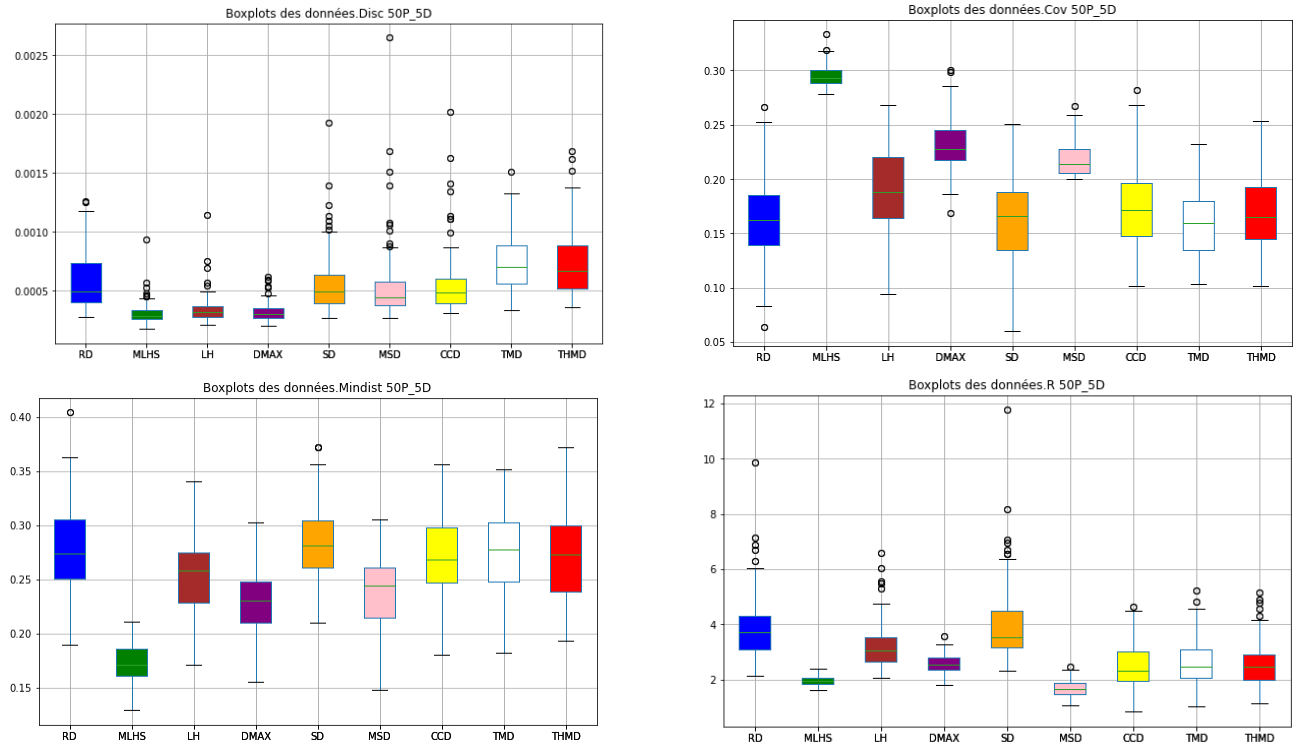

Figure 4.2 – Box plots des critères de qualité calculés sur les 100 plans à 50 points en dimension 5

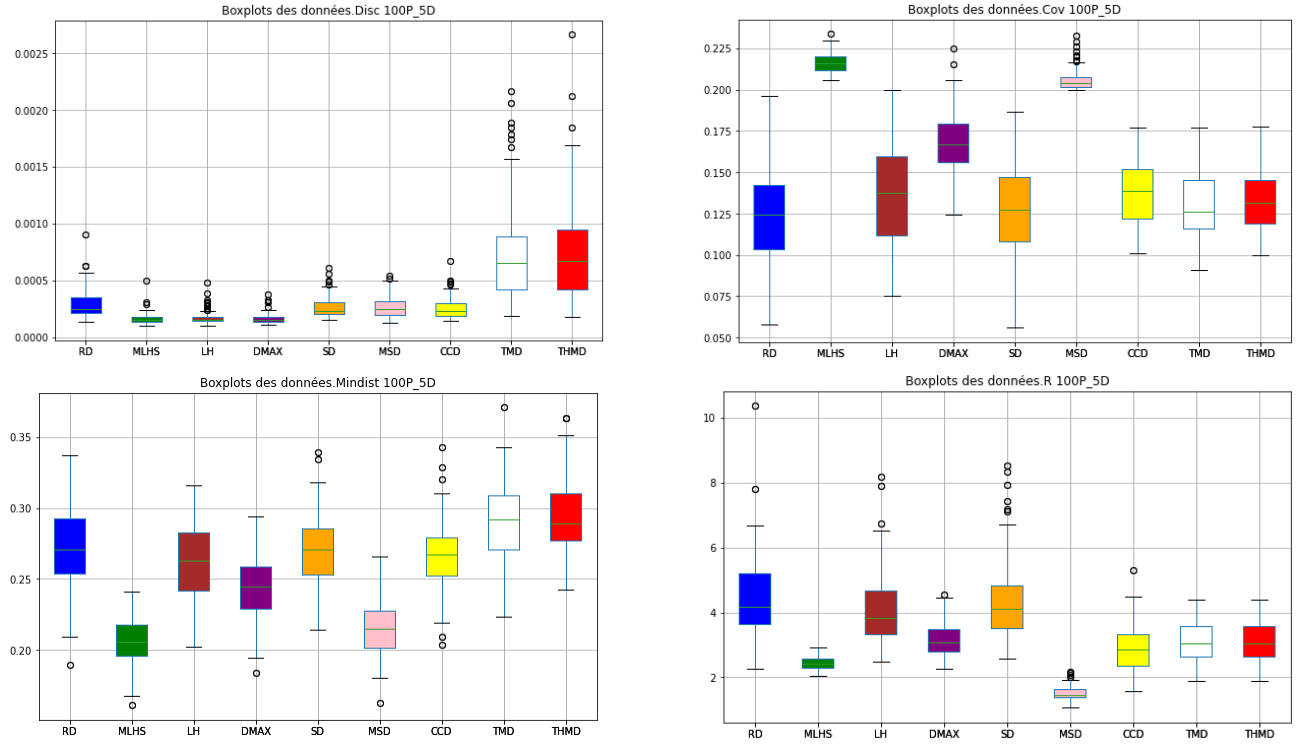

Figure 4.3 – Box plots des critères de qualité calculés sur les 100 plans à 100 points en dimension 5

Les box plots offrent une comparaison visuelle entre notre plan d'expérience numérique THMD et plusieurs autres plans tels que MSD, LH, RD, MLHS, DMAX, CCD, SD et TMD, en se basant sur quatre critères clés : la discrépance, le recouvrement, la distance et le critère R, à l'exception des plans Maximin LHS pour une configuration en 5 dimensions. Néanmoins, notre plan s'est montré

très compétitif par rapport aux autres plans en termes de critère de discrépance.

4.5 Plans avec 20, 50 et 100 points en 7 dimensions.

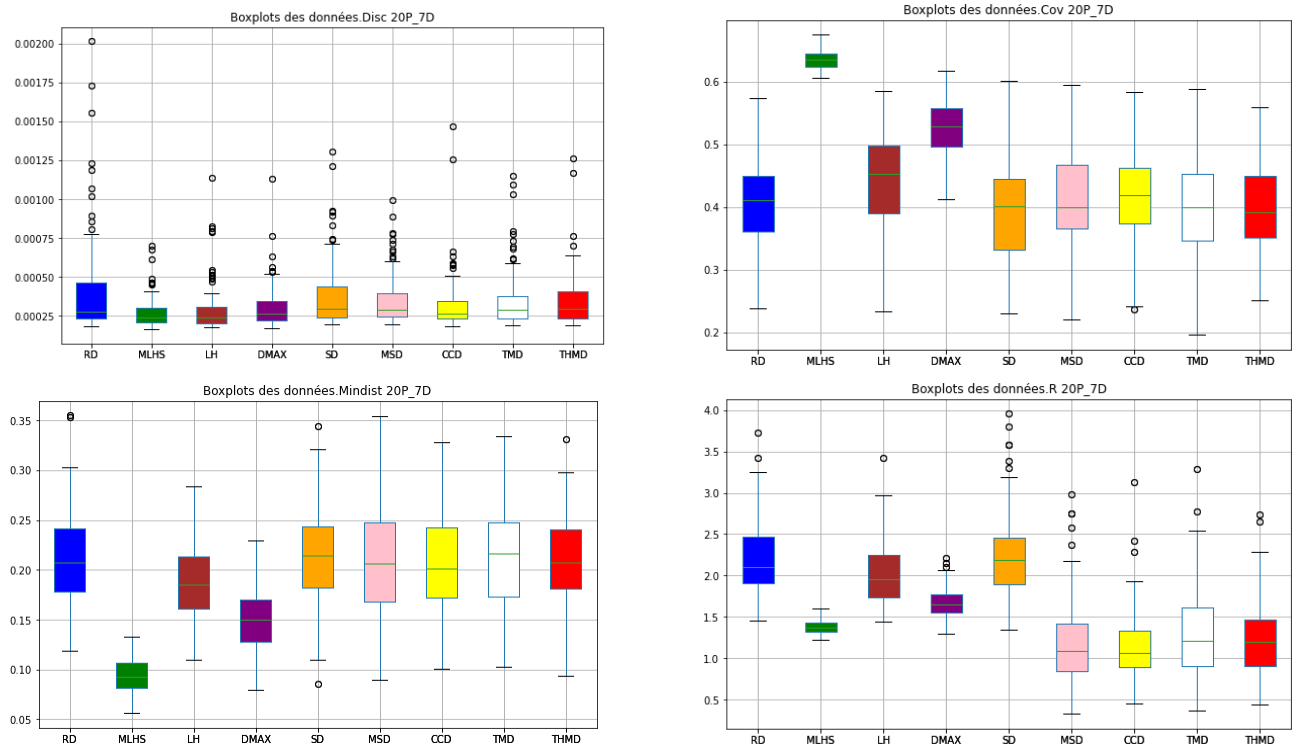

Figure 4.4 – Box plots des critères de qualité calculés sur les 100 plans à 20 points en dimension 7

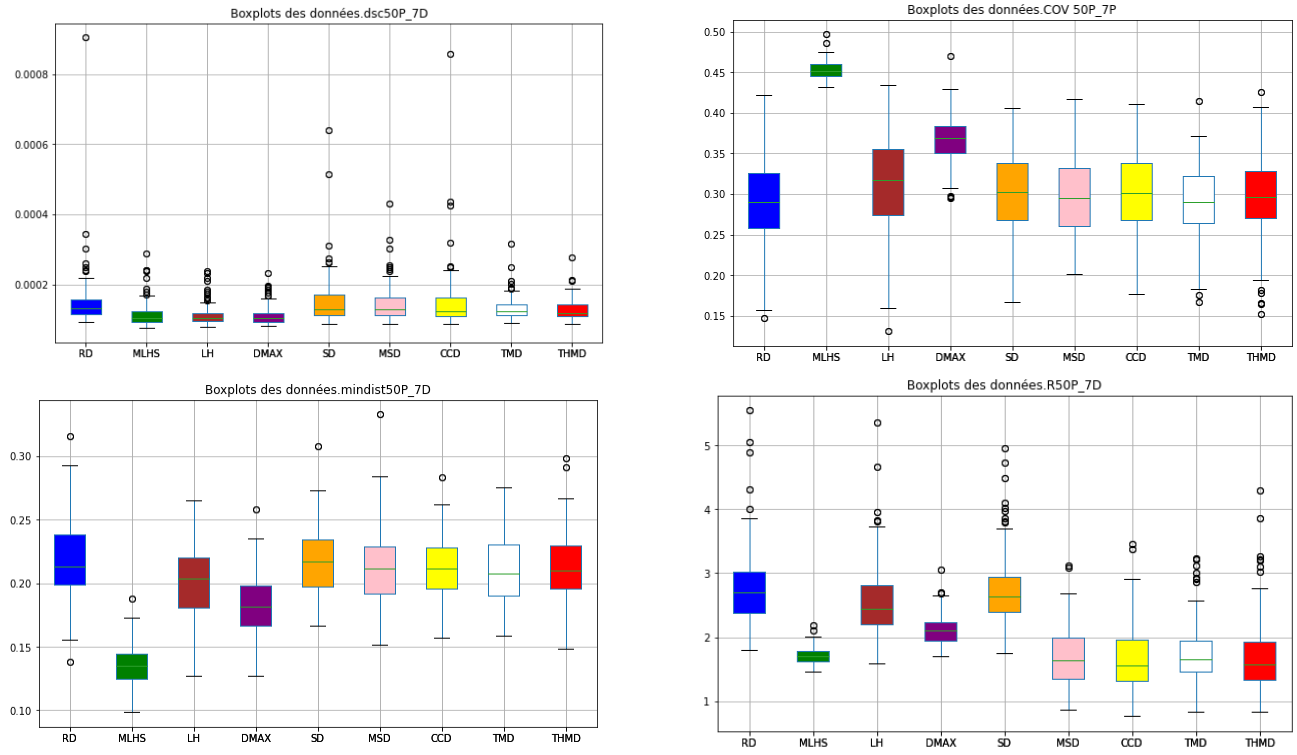

Figure 4.5 – Box plots des critères de qualité calculés sur les 100 plans à 50 points en dimension 7

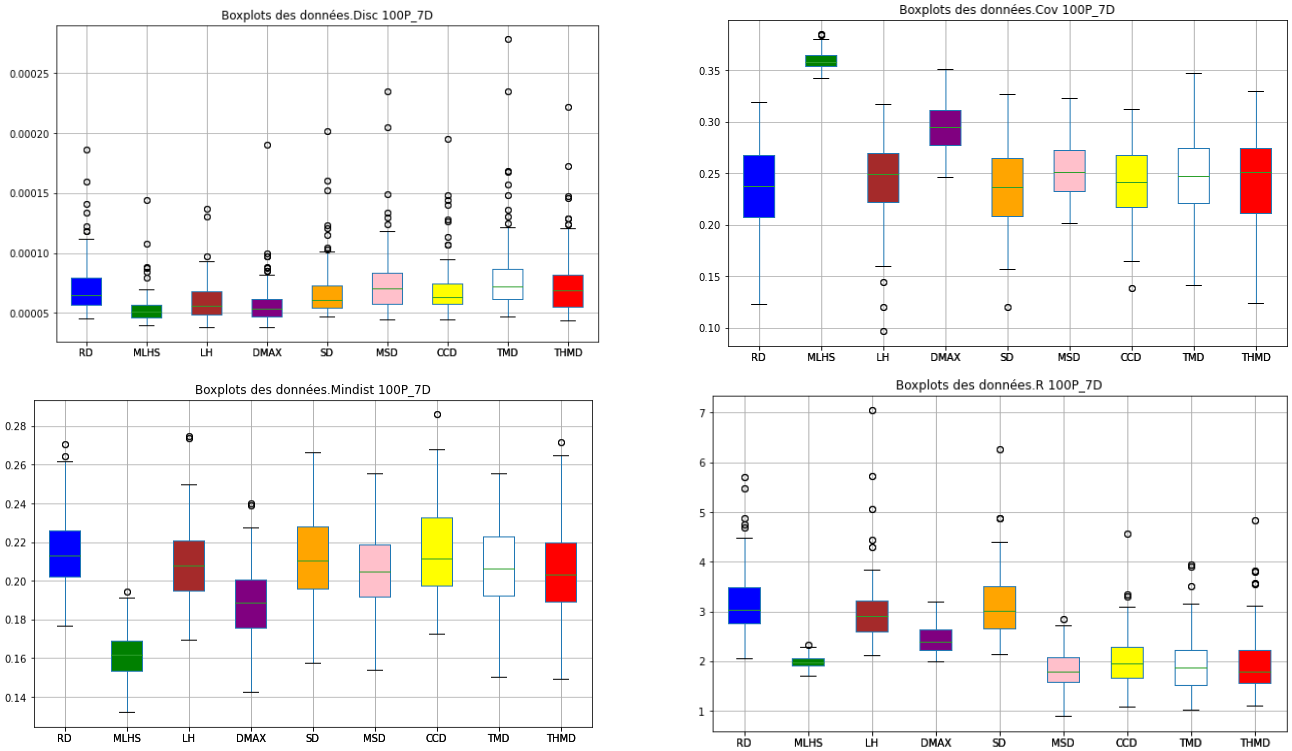

Figure 4.6 – Box plots des critères de qualité calculés sur les 100 plans à 100 points en dimension 7

Selon les box plots ci-dessus, en ce qui concerne le critère de recouvrement, notre plan proposé a surpassé pratiquement tous les autres plans stochastiques en se basant sur quatre critères clés : la discrépance, le recouvrement, la distance et le critère R. Plus précisément, en ce qui concerne le critère de discrépance sur des ensembles de 20, 50 et 100 points dans 7 dimensions, notre plan d'expérience s'est avéré être remarquablement performant.

Conclusion

En analysant un système uniquement par ses entrées (facteurs) et ses sorties (réponses), sans tenir compte de sa structure interne, les Plans d'expériences se révèlent extrêmement puissants. Cette approche permet d'analyser des systèmes de diverses disciplines (physique, chimie, génie civil, mécanique, biologie, gestion, etc.). Bien que la méthode statistique ne cherche pas à remplacer les résultats spécifiques de chaque domaine, elle offre des résultats complémentaires essentiels pour valider les théories, obtenir des résultats pratiques et optimiser les systèmes étudiés. Le modèle statistique, souvent sous forme de polynôme, simplifie et unifie les informations apportées par chaque facteur, qu'ils agissent seuls ou ensemble.

La méthodologie de la recherche expérimentale, ou méthode des plans d'expériences, est une branche des mathématiques relevant de la statistique inférentielle. Cette méthode est en perpétuelle évolution et a gagné en importance grâce aux logiciels spécialisés.

Dans cette étude, nous avons d'abord exploré la théorie des processus ponctuels marqués et les méthodes MCMC pour créer de nouvelles matrices d'expériences numériques basées sur les processus ponctuels marqués markoviens de Strauss. Ces plans permettent une répartition optimale et uniforme des points expérimentaux dans l'espace d'étude, tout en répondant à deux objectifs : la distribution des points et la caractérisation de leurs marques, pour maximiser la prédictibilité du modèle. Les résultats comparatifs avec d'autres plans numériques se sont avérés très satisfaisants.

Les résultats obtenus dans ce mémoire ouvrent de nouvelles perspectives de recherche, telles que la caractérisation des points expérimentaux par d'autres types de marques, ou encore l'utilisation d'autres modèles de processus ponctuels marqués.

Annex A

Code En PYTHON pour les résultats présentés en chapitre trois et quatre

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

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

```

```
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):
66          t[i] = np.min(M[i, :])
67
68      q = np.sum(t)
69      q1 = q / n
70      lamda = 0
71      for i in range(n):
72          lamda += (t[i] - q1) ** 2
73
74      y = (1 / q1) * ((1 / n) * lamda) ** 0.5
75      return y
76
77  def generate_maximal_entropy_matrix(n, d):
78      # Generate random orthogonal matrix
79      matrix = lhs(d, samples=n, criterion='centermaximin')
80      return matrix.tolist()
81
82  def generate_random_matrix(n, d):
83      matrix = np.random.rand(n, d)
84      return matrix.tolist()
85
86  def generate_maximin_lhs_matrix(n, d):
87      matrix = lhs(d, samples=n, criterion='maximin', iterations=1000)
88      return matrix.tolist()
89
90  def generate_strauss_matrix(n, d, r):
91      matrix = np.zeros((n, d))
92      matrix[0] = np.random.rand(d)
93
94      for i in range(1, n):
95          candidate = np.random.rand(d)
96          distances = np.linalg.norm(matrix[:i] - candidate, axis=1)
97          while np.any(distances < r):
98              candidate = np.random.rand(d)
99              distances = np.linalg.norm(matrix[:i] - candidate, axis=1)
100         matrix[i] = candidate
```

```
101
102     return matrix.tolist()
103     def rap(X):
104     n = X.shape[0]
105     w = X.shape[1]
106     M = np.zeros((n, n))
107     for i in range(n-1):
108         for k in range(i+1, n):
109             s = 0
110             for j in range(w):
111                 s += (X[i, j] - X[k, j])**2
112             M[i, k] = np.sqrt(s)
113             M[k, i] = np.sqrt(s)
114     np.fill_diagonal(M, np.inf)
115     t = np.min(M, axis=1)
116     t1 = np.min(t)
117     t2 = np.max(t)
118     y = t2 / t1
119     return y
120
121 # Parameters
122 n1 = 100
123 dim1 = 7
124 n = int(n1)
125 dim = int(dim1)
126
127 # Create empty DataFrames to store values
128 df = pd.DataFrame(columns=[])
129
130 for _ in range(100):
131     print("Iteration:", _ + 1)
132     rd_matrix = np.array(generate_random_matrix(n, dim))
133     latiner = stats.qmc.LatinHypercube(dim, scramble=False)
134     latin = latiner.random(n)
135     mlhs_matrix = np.array(generate_maximin_lhs_matrix(n, dim))
136     dmax_matrix = np.array(generate_maximal_entropy_matrix(n, dim))
137     sd_matrix = np.array(generate_strauss_matrix(n, dim, r=0.05))
138
```

```
    a = mdist(rd_matrix)
    b = mindist(rd_matrix)
    c = dsc(rd_matrix)
    d = rap(rd_matrix)

    a1 = mdist(mlhs_matrix)
    b1 = mindist(mlhs_matrix)
    c1 = dsc(mlhs_matrix)
    d1 = rap(mlhs_matrix)

    a2 = mdist(latin)
    b2 = mindist(latin)
    c2 = dsc(latin)
    d2 = rap(latin)

    a3 = mdist(dmax_matrix)
    b3 = mindist(dmax_matrix)
    c3 = dsc(dmax_matrix)
    d3 = rap(dmax_matrix)

    a4 = mdist(sd_matrix)
    b4 = mindist(sd_matrix)
    c4 = dsc(sd_matrix)
    d4 = rap(sd_matrix)

    new_row = {'mdist(RD)': a, 'mindist(RD)': b, 'dsc(RD)': c, 'R(RD)': d,
               'mdist(mlhs)': a1, 'mindist(mlhs)': b1, 'dsc(mlhs)': c1, 'R(mlhs)'
                  : d1,
               'mdist(latin)': a2, 'mindist(latin)': b2, 'dsc(latin)': c2, 'R(
                  latin)': d2,
               'mdist(dmax)': a3, 'mindist(dmax)': b3, 'dsc(dmax)': c3, 'R(dmax)'
                  : d3,
               'mdist(sd)': a4, 'mindist(sd)': b4, 'dsc(sd)': c4, 'R(sd)': d4}

    df = pd.concat([df, pd.DataFrame(new_row, index=[0])], ignore_index=True)

# Export the DataFrame to an Excel file
nom_fichier = 'resultats100P_7d.xlsx'
```

```
df.to_excel(nom_fichier, index=False)
print(f"Les résultats ont été exportés vers {nom_fichier}")
```

CCD

```
import numpy as np
import pandas as pd

def mdist(x):
    n = x.shape[0]
    w = x.shape[1]
    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]
    w = x.shape[1]
    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]
    w = x.shape[1]
    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):
```

```
70          lamda += (t[i] - q1) ** 2
71
72      y = (1 / q1) * ((1 / n) * lamda) ** 0.5
73      return y
74
75  def rap(X):
76      n = X.shape[0]
77      w = X.shape[1]
78      M = np.zeros((n, n))
79
80      for i in range(n - 1):
81          for k in range(i + 1, n):
82              s = 0
83              for j in range(w):
84                  s += (X[i, j] - X[k, j]) ** 2
85              M[i, k] = np.sqrt(s)
86              M[k, i] = np.sqrt(s)
87
88      np.fill_diagonal(M, np.inf)
89
90      t = np.min(M, axis=1)
91      y = np.ptp(t) / np.min(t)
92      return y
93
94  def nn2(x, r):
95      n, w = x.shape
96      a = np.zeros((n, n))
97
98      for i in range(n - 1):
99          for j in range(i + 1, n):
100             D = x[i] - x[j]
101             d = np.linalg.norm(D)
102
103     if d < r:
104     a[i, j] = 1
105     a[j, i] = 1
106
107 ncx = 0
```

```
k = 0
t = np.zeros(n+1, dtype=int)  # Modifier ici pour n+1

for i in range(n):
    if t[:k+1].tolist().count(i) == 0:
        k += 1
        ncx += 1
        t[k] = i
        for j in range(i + 1, n):
            if a[i, j] == 1 and t[:k+1].tolist().count(j) == 0:
                k += 1
                t[k] = j

return ncx
n = 100
r = 0.1
gamma = 0.04

w = 7

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'])
df_d = pd.DataFrame(columns=['Valeur'])

# Execute the code 20 times
results_a = []
results_b = []
results_c = []

np_values = np.zeros(NMC)
for _ in range(100):
    # initialisation de la configuration
    X = np.random.rand(n, w)
    print("Iteration:", _ + 1)
```

```
146     for N in range(NMC):
147         print("Ite:", N + 1)
148         # choix d'un point au hasard
149         k = np.random.randint(n)
150         # création d'une nouvelle configuration Y
151         y = np.random.rand(1, w)
152         Y = np.copy(X)
153         Y[k] = y
154         print(N)
155         # calcul du nombre d'interactions pour X et Y
156         bx = nn2(X, r)
157         by = nn2(Y, r)
158 
159         # calcul de la probabilité d'acceptation
160         a = min(1, gamma ** (bx - by))
161         if a == 1:
162             X[k] = y
163             np_values[N] = by
164         else:
165             np_values[N] = bx
166 
167     # Calcul de la valeur de a et b
168     a = mdist(X)
169     b = mindist(X)
170     c = dsc(X)
171     d = rap(X)
172     print(d)
173 
174     # Store the results in DataFrames
175     df_a = pd.concat([df_a, pd.DataFrame({'Valeur': [a]})], ignore_index=True)
176     df_b = pd.concat([df_b, pd.DataFrame({'Valeur': [b]})], ignore_index=True)
177     df_c = pd.concat([df_c, pd.DataFrame({'Valeur': [c]})], ignore_index=True)
178     df_d = pd.concat([df_d, pd.DataFrame({'Valeur': [d]})], ignore_index=True)
179 
180 # Save the DataFrames to a single Excel file
181 with pd.ExcelWriter('CCD.100P_7dBD.xlsx') as writer:
182     df_a.to_excel(writer, sheet_name='mdist', index=False)
183     df_b.to_excel(writer, sheet_name='mindist', index=False)
```

```
    df_c.to_excel(writer, sheet_name='dsc', index=False)
    df_d.to_excel(writer, sheet_name='R', index=False)
```

MSD

```
import numpy as np
import pandas as pd

def rap(X):
    n = X.shape[0]
    w = X.shape[1]
    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)

    np.fill_diagonal(M, np.inf)

    t = np.min(M, axis=1)
    y = np.ptp(t) / np.min(t)
    return y

def mdist(x):
    n = x.shape[0]
    w = x.shape[1]
    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]
    w = x.shape[1]
    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]
    w = x.shape[1]
    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

n = 100
R = 0.2
t = 0.1
b = 2
w = 7
eps = 0.04
NMC = 1000

df_a = pd.DataFrame(columns=['Valeur'])
df_b = pd.DataFrame(columns=['Valeur'])
df_c = pd.DataFrame(columns=['Valeur'])
df_d = pd.DataFrame(columns=['Valeur'])

for _ in range(100):
```

```
    X = np.random.rand(n, w)
    for N in range(NMC):
        k = np.random.randint(n)
        y = np.random.rand(1, w)
        Y = np.copy(X)
        Y[k] = 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

        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

        f1 = b ** somme
        by = t ** Sy * f1
        bx = t ** Sx * somme
        a = \min(1, \frac{by}{bx})

# mise à jour de la configuration
if a == 1:
    X[k] = y
a = mdist(X)
b = mindist(X)
c = dsc(X)
d = rap(X)
print(a)
print(b)
print(c)
```

```
print(d)

# Store the results in DataFrames
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)
df_d = pd.concat([df_d, pd.DataFrame({'Valeur': [d]})], ignore_index=True)

# Save the DataFrames to a single Excel file
with pd.ExcelWriter('MSD.100P_7dd.xlsx') as writer:
       df_a.to_excel(writer, sheet_name='mdist', index=False)
       df_b.to_excel(writer, sheet_name='mindist', index=False)
       df_c.to_excel(writer, sheet_name='dsc', index=False)
       df_d.to_excel(writer, sheet_name='R', index=False)
```

TMD

```
import numpy as np
from scipy.integrate import quad
import pandas as pd

def mdist(x):
    n = x.shape[0]
    w = x.shape[1]
    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
```

```
20
21      y = np.min(np.min(M))
22      return y
23
24  def dsc(x):
25      n = x.shape[0]
26      w = x.shape[1]
27      s1 = 0
28
29      for i in range(n):
30          p1 = 1
31          for j in range(w):
32              p1 *= (1 - x[i, j]) * (1 + x[i, j])
33          s1 += p1
34
35      s2 = 0
36      for i in range(n):
37          for j in range(n):
38              p2 = 1
39              for k in range(w):
40                  m = max(x[i, k], x[j, k])
41                  p2 *= (1 - m)
42              s2 += p2
43
44      y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n ** 2 * (w - 1))) * s1
45      return y
46      def mindist(x):
47      n = x.shape[0]
48      w = x.shape[1]
49      M = np.zeros((n, n))
50
51      for i in range(n - 1):
52          for k in range(i + 1, n):
53              s = 0
54              for j in range(w):
55                  s += (x[i, j] - x[k, j]) ** 2
56              M[i, k] = np.sqrt(s)
57              M[k, i] = np.sqrt(s)
```

```
58
59      for i in range(n):
60          M[i, i] = np.inf
61
62      t = np.zeros(n)
63      for i in range(n):
64          t[i] = np.min(M[i, :])
65
66      q = np.sum(t)
67      q1 = q / n
68      lamda = 0
69      for i in range(n):
70          lamda += (t[i] - q1) ** 2
71
72      y = (1 / q1) * ((1 / n) * lamda) ** 0.5
73      return y
74
75  def rap(X):
76      n = X.shape[0]
77      w = X.shape[1]
78      M = np.zeros((n, n))
79
80      for i in range(n - 1):
81          for k in range(i + 1, n):
82              s = 0
83              for j in range(w):
84                  s += (X[i, j] - X[k, j]) ** 2
85              M[i, k] = np.sqrt(s)
86              M[k, i] = np.sqrt(s)
87
88      np.fill_diagonal(M, np.inf)
89
90      t = np.min(M, axis=1)
91      y = np.ptp(t) / np.min(t)
92      return y
93      w = 7             # dimension
94  r = 0.05          # rayon d'interaction
95
```

```
# Définition des paramètres du modèle
n = 100          # nombre de points
eps = 0.07
R = 0.1           # rayon de la gaussienne de proposition
g11 = 0.01
g12 = 0.01         # coefficient d'interaction pour les paires de marque (1,2)
g22 = 0.05          # coefficient d'interaction pour les paires de marque (2,2)
b1 = 0.9            # intensité du processus pour la marque 1
b2 = 1.5            # intensité du processus pour la marque 1
NMC = 1000

# marques1
def m1(X):
    n = X.shape[0]
    m = np.zeros(n)
    XTX_inv = np.linalg.inv(np.matmul(X.T, X))
    for i in range(n):
        m[i] = np.matmul(np.matmul(X[i, :], XTX_inv), X[i, :].T)
    return m

# marques2
def distance(x1, x2):
    return np.linalg.norm(x1-x2)

def density(x):
    return np.exp(-(x)**2/2) / (np.sqrt(2*np.pi))
def m2(X):
    n = X.shape[0]
    mus = np.zeros(n)
    for i in range(n):
        sum_distances = 0
        for j in range(n):
            if i != j:
                dist = distance(X[i], X[j])
                integral, _ = quad(lambda t: density(t), 0, dist)
                sum_distances += integral
        mus[i] = sum_distances / (n-1)
    return mus
```

```
134
135 def m11(X, m1,r, eps):
136     n = X.shape[0]
137     count = 0
138     for i in range(n):
139         for j in range(i+1, n):
140             if np.linalg.norm(X[i] - X[j]) <= r and m1[i] <= eps and m1[j] <= eps
                :
141                 count += 1
142     return count
143
144 def m22(X, m1, eps, r):
145     n = X.shape[0]
146     count = 0
147     for i in range(n):
148         for j in range(i+1, n):
149             if np.linalg.norm(X[i] - X[j]) <= r and m1[i] >= R and m1[j] >= R:
150                 count += 1
151     return count
152
153 def m12(X, m1, m2, eps, r, R):
154     n = X.shape[0]
155     count = 0
156     for i in range(n):
157         for j in range(i+1, n):
158             if   m1[i] <= eps and m1[j] <= eps and m2[i] >= R and m2[j] >= R:
159                 dist = np.linalg.norm(X[i] - X[j])
160                 if dist <= r:
161                     count += 1
162     return count
163
164 # Génération d'une configuration initiale de points aléatoires dans un carré
    unité
165
166 # Create empty DataFrames to store values
167 df_a = pd.DataFrame(columns=['Valeur'])
168 df_b = pd.DataFrame(columns=['Valeur'])
169 df_c = pd.DataFrame(columns=['Valeur'])
```

```
df_d = pd.DataFrame(columns=['Valeur'])
# Execute the code 20 times
# Simulation de NMC étapes
for _ in range(100):
    X = np.random.rand(n, w)

    for N in range(NMC):
        print("Itera", _ + 1, N + 1)
# 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
        print(N)
    # Calcul des valeurs de m1, m2, m11, m22 et m12 pour la configuration
        actuelle
        m1_vals = m1(X)
        num_points_m1 = len(np.where(m1_vals <= eps)[0])
        m2_vals = m2(X)
        num_points_m2 = len(np.where(m2_vals >= eps)[0])
        m11_vals = m11(X, m1_vals, r, eps)
        m22_vals = m22(X, m2_vals, r, R)
        m12_vals = m12(X, m1_vals, m2_vals, eps, r, R)

    # Calcul des valeurs de m1, m2, m11, m22 et m12 pour la nouvelle
        configuration
    m1_vals_new = m1(Y)
num_points_m1_new = len(np.where(m1_vals_new <= eps)[0])
m2_vals_new = m2(Y)
num_points_m2_new = len(np.where(m2_vals_new >= eps)[0])
m11_vals_new = m11(Y, m1_vals, R, eps)
m22_vals_new = m22(Y, m2_vals_new, r, R)
m12_vals_new = m12(Y, m1_vals_new, m2_vals_new, eps, r, R)
# Calcul de la probabilité d'acceptation
by = b1**num_points_m1_new * b2**num_points_m2_new * g11**m11_vals_new * g12**
    m12_vals_new * g22**m22_vals_new
```

```
bx = b1**num_points_m1 * b2**num_points_m2 * g11**m11_vals * g12**m12_vals * g22**m22_vals
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)
d = rap(X)
print(c)

# Store the results in DataFrames
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)
df_d = pd.concat([df_d, pd.DataFrame({'Valeur': [d]})], ignore_index=True)

# Save the DataFrames to a single Excel file
with pd.ExcelWriter('P.P.M2.100P_7D.xlsx') as writer:
    df_a.to_excel(writer, sheet_name='mdist', index=False)
    df_b.to_excel(writer, sheet_name='mindist', index=False)
    df_c.to_excel(writer, sheet_name='dsc', index=False)
    df_d.to_excel(writer, sheet_name='R', index=False)
```

THMD

```
import numpy as np
from scipy.integrate import quad
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)

    np.fill_diagonal(M, np.inf)

    y = 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):
```

```
46      n = x.shape[0]  # Nombre de points dans x
47      w = x.shape[1]  # Dimension de x
48      M = np.zeros((n, n))
49
50      for i in range(n - 1):
51          for k in range(i + 1, n):
52              s = 0
53              for j in range(w):
54                  s += (x[i, j] - x[k, j]) ** 2
55              M[i, k] = np.sqrt(s)
56              M[k, i] = np.sqrt(s)
57
58      np.fill_diagonal(M, np.inf)
59
60      t = np.zeros(n)
61      for i in range(n):
62          t[i] = np.min(M[i, :])
63
64      q = np.sum(t)
65      q1 = q / n
66      lamda = 0
67      for i in range(n):
68          lamda += (t[i] - q1) ** 2
69
70      y = (1 / q1) * ((1 / n) * lamda) ** 0.5
71      return y
72
73  def rap(X):
74      n = X.shape[0]  # Nombre de points dans x
75      w = X.shape[1]  # Dimension de x
76      M = np.zeros((n, n))
77
78      for i in range(n - 1):
79          for k in range(i + 1, n):
80              s = 0
81              for j in range(w):
82                  s += (X[i, j] - X[k, j]) ** 2
83              M[i, k] = np.sqrt(s)
```

```
            M[k, i] = np.sqrt(s)

    np.fill_diagonal(M, np.inf)

    y = np.min(M)
    return y
    w = 7       # dimension
r = 0.1     # rayon d'interaction

# Définition des paramètres du modèle
n = 100     # nombre de points
R1 = 0.07
R2 = 0.09
R3 = 0.04

# rayon de la gaussienne de proposition
g11 = 0.1
g12 = 0.02
g13 = 0.05
g23 = 0.2   # coefficient d'interaction pour les paires de marque (1,2)
g22 = 0.02
g33 = 0.08  # coefficient d'interaction pour les paires de marque (2,2)
b1 = 0.5    # intensité du processus pour la marque 1
b2 = 0.5    # intensité du processus pour la marque 2
b3 = 0.5    # intensité du processus pour la marque 3
NMC = 1000

# Marques 1
def m1(X):
    n = X.shape[0]
    m = np.zeros(n)
    XTX_inv = np.linalg.inv(np.matmul(X.T, X))
    for i in range(n):
        m[i] = np.matmul(np.matmul(X[i, :], XTX_inv), X[i, :].T)
    return m

# Marques 2
def distance(x1, x2):
```

```
122     return np.linalg.norm(x1 - x2)
123
124 def density(x):
125     return np.exp(-(x)**2 / 2) / (np.sqrt(2 * np.pi))
126
127 def m2(X):
128     n = X.shape[0]
129     mus = np.zeros(n)
130     for i in range(n):
131         sum_distances = 0
132         for j in range(n):
133             if i != j:
134                 dist = distance(X[i], X[j])
135                 integral, _ = quad(lambda t: density(t), 0, dist)
136                 sum_distances += integral
137         mus[i] = sum_distances / (n - 1)
138     return mus
139
140 # Marques 3
141 def dist(x1, x2):
142     return np.linalg.norm(x1 - x2)
143
144 def denst(x):
145     return np.exp(-(x)**2 / 2) / (np.sqrt(2 * np.pi))
146
147 def m3(X):
148     n = X.shape[0]
149     mdt = np.zeros(n)
150
151     for i in range(n):
152         mus = np.zeros(n)
153         for j in range(n):
154             if i != j:
155                 dist = distance(X[i], X[j])
156                 mus[j] = dist
157         mdt[i] = np.min(mus[mus > 0])  # Correction de la sélection du minimum
158     return mdt
159     def m11(X, m1, r, R1):
```

```
    n = X.shape[0]
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if np.linalg.norm(X[i] - X[j]) <= r and m1[i] <= R1 and m1[j] <= R1:
                count += 1
    return count

def m22(X, m2, R2, r):
    n = X.shape[0]
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if np.linalg.norm(X[i] - X[j]) <= r and m2[i] >= R2 and m2[j] >= R2:
                count += 1
    return count
    def m33(X, m3, R3, r):
    n = X.shape[0]
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if np.linalg.norm(X[i] - X[j]) <= r and m3[i] >= R3 and m3[j] >= R3:
                count += 1
    return count

def m12(X, m1, m2, R1, r, R2):
    n = X.shape[0]
    count = 0
    for i in range(n):
        for j in range(i + 1, n):
            if m1[i] <= R1 and m1[j] <= R1 and m2[i] >= R2 and m2[j] >= R2:
                dist = np.linalg.norm(X[i] - X[j])
                if dist <= r:
                    count += 1
    return count

def m13(X, m1, m3, R1, r, R3):
    n = X.shape[0]
```

```
198     count = 0
199     for i in range(n):
200         for j in range(i + 1, n):
201             if m1[i] <= R1 and m1[j] <= R1 and m3[i] >= R3 and m3[j] >= R3:
202                 dist = np.linalg.norm(X[i] - X[j])
203                 if dist <= r:
204                     count += 1
205     return count
206 
207 def m23(X, m2, m3, R2, r, R3):
208     n = X.shape[0]
209     count = 0
210     for i in range(n):
211         for j in range(i + 1, n):
212             if m2[i] >= R2 and m2[j] >= R2 and m3[i] >= R3 and m3[j] >= R3:
213                 dist = np.linalg.norm(X[i] - X[j])
214                 if dist <= r:
215                     count += 1
216     return count
217 # Génération d'une configuration initiale de points aléatoires dans un carré
    unité
218 
219 # Create empty DataFrames to store values
220 df_a = pd.DataFrame(columns=['Valeur'])
221 df_b = pd.DataFrame(columns=['Valeur'])
222 df_c = pd.DataFrame(columns=['Valeur'])
223 df_d = pd.DataFrame(columns=['Valeur'])
224 # Execute the code 20 times
225 # Simulation de NMC étapes
226 for _ in range(100):
227     X = np.random.rand(n, w)
228 
229     for N in range(NMC):
230         print("Itera", _ + 1, N + 1)
231         # Choix d'un point au hasard
232         k = np.random.randint(n)
233 
234         # Création d'une nouvelle configuration Y
```

```
        y = np.random.rand(1, w)
        Y = np.copy(X)
        Y[k] = y
        print(N)
        # Calcul des valeurs de m1, m2, m3, m11, m22, m33, m12, m13, m23 pour la
            configuration actuelle
        m1_vals = m1(X)
        num_points_m1 = len(np.where(m1_vals <= R1)[0])
        m2_vals = m2(X)
        num_points_m2 = len(np.where(m2_vals >= R2)[0])
        m3_vals = m3(X)
        num_points_m3 = len(np.where(m3_vals >= R3)[0])
        m11_vals = m11(X, m1_vals, r, R1)
        m22_vals = m22(X, m2_vals, r, R2)
        m33_vals = m33(X, m3_vals, r, R3)
        m12_vals = m12(X, m1_vals, m2_vals, R1, r, R2)
        m13_vals = m13(X, m1_vals, m3_vals, R1, r, R3)
        m23_vals = m23(X, m2_vals, m3_vals, R2, r, R3)

        # Calcul des valeurs de m1, m2, m3, m11, m22, m33, m12, m13, m23 pour la
            nouvelle configuration
        m1_vals_new = m1(Y)
        num_points_m1_new = len(np.where(m1_vals_new <= R1)[0])
        m2_vals_new = m2(Y)
        num_points_m2_new = len(np.where(m2_vals_new >= R2)[0])
        m3_vals_new = m3(Y)
        num_points_m3_new = len(np.where(m3_vals_new >= R3)[0])
        m11_vals_new = m11(Y, m1_vals_new, r, R1)
        m22_vals_new = m22(Y, m2_vals_new, r, R2)
        m33_vals_new = m33(Y, m3_vals_new, r, R3)
        m12_vals_new = m12(Y, m1_vals_new, m2_vals_new, R1, r, R2)
        m13_vals_new = m13(Y, m1_vals_new, m3_vals_new, R1, r, R3)
        m23_vals_new = m23(Y, m2_vals_new, m3_vals_new, R2, r, R3)

        # Calcul de la probabilité d'acceptation
        by = (b1 ** num_points_m1_new) * (b2 ** num_points_m2_new) * (b3 **
            num_points_m3_new) * \
```

```
269             (g11 ** m11_vals_new) * (g22 ** m22_vals_new) * (g33 ** m33_vals_new
                    ) * \
270             (g12 ** m12_vals_new) * (g23 ** m23_vals_new) * (g13 ** m13_vals_new
                    )
271
272         bx = (b1 ** num_points_m1) * (b2 ** num_points_m2) * (b3 ** num_points_m3
                ) * \
273             (g11 ** m11_vals) * (g22 ** m22_vals) * (g33 ** m33_vals) * \
274             (g12 ** m12_vals) * (g23 ** m23_vals) * (g13 ** m13_vals)
275
276         a = min(1, by / bx)
277
278         # Mise à jour de la configuration
279         if a == 1:
280             X[k] = y
281             # print(m1_vals)
282
283     # Calcul des valeurs de a, b, c, d
284     a = mdist(X)
285     b = mindist(X)
286     c = dsc(X)
287     d = rap(X)
288     print(c)
289
290     # Store the results in DataFrames
291     df_a = pd.concat([df_a, pd.DataFrame({'Valeur': [a]})], ignore_index=True)
292     df_b = pd.concat([df_b, pd.DataFrame({'Valeur': [b]})], ignore_index=True)
293     df_c = pd.concat([df_c, pd.DataFrame({'Valeur': [c]})], ignore_index=True)
294     df_d = pd.concat([df_d, pd.DataFrame({'Valeur': [d]})], ignore_index=True)
295
296 # Save the DataFrames to a single Excel file
297 with pd.ExcelWriter('P.P.M3.100P_7D.xlsx') as writer:
298     df_a.to_excel(writer, sheet_name='mdist', index=False)
299     df_b.to_excel(writer, sheet_name='mindist', index=False)
300     df_c.to_excel(writer, sheet_name='dsc', index=False)
301     df_d.to_excel(writer, sheet_name='R', index=False)
```

BOXPLOTS

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

# Chemin vers le fichier Excel sur votre bureau
chemin_fichier = r'C:\Users\WIDOWS\Desktop\100P_7D\R 100P_7D.xlsx'

# Charger les données Excel dans un DataFrame pandas
donnees = pd.read_excel(chemin_fichier)

# Créer une liste de couleurs pour chaque boîte
couleurs = ['blue', 'green', 'brown', 'purple', 'orange', 'pink', 'yellow', '
    white', 'red']

# Créer les boxplots avec des couleurs différentes pour chaque boîte
plt.figure(figsize=(10, 6))
bp = plt.boxplot(donnees, patch_artist=True)

# Itérer sur chaque boîte pour lui attribuer une couleur différente
for i, box in enumerate(bp['boxes']):
    box.set_facecolor(couleurs[i])

donnees.boxplot()
plt.title('Boxplots des données.R 100P_7D')
plt.show()
```

SUITES A FAIBLE DISCREPANCE

```
import numpy as np
import pandas as pd
from scipy.stats import qmc

def van_der_corput(n, base):
```

```
    """Génère le n-ième terme de la suite de Van der Corput pour une base donnée.
       """
    result = 0
    fraction = 1.0 / base
    while n > 0:
        result += (n % base) * fraction
        n //= base
        fraction /= base
    return result

def faure_sequence(dimensions, num_points, primes):
    """Génère les num_points premiers points d'une suite de Faure pour les
       dimensions spécifiées."""
    points = np.zeros((num_points, dimensions))
    for i in range(num_points):
        for j in range(dimensions):
            points[i, j] = van_der_corput(i, primes[j])
    return points

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
```

```
42
43      y = (3 ** (-w)) + (1 / (n ** 2)) * s2 - (1 / (n ** 2 * (w - 1))) * s1
44      return y
45
46  # ... (the rest of the functions remain the same)
47
48  n = 128
49  d = 10
50  primes = [2, 3, 5, 7, 11, 13, 17]
51  # , 19, 23, 29]
52  # , 31, 37, 41, 43, 47]
53  # , 53, 59, 61, 67, 71,
54  #           73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
        151,
55  #           157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229]
56  dimensions = len(primes)
57
58  # Halton
59  haltoner = qmc.Halton(d, scramble=False)
60  halton = haltoner.random(n)
61
62  # Sobol
63  soboler = qmc.Sobol(d, scramble=False)
64  sobol = soboler.random(n)
65
66  # Faure
67  faur = faure_sequence(dimensions, n, primes)
68
69  # ... (the rest of the code remains the same)
70
71      # Calcul de la valeur de a et b
72
73  # Calcul de la valeur de a et b
74  c = dsc(halton)
75  print('halton=', c)
76
77  c1 = dsc(sobol)
78  print('sobol=', c1)
```

```
79
80 c2 = dsc(faur)
81 print('faur=', c2)
82
83 # Create a DataFrame to store the results
84 results_df = pd.DataFrame({
85     'Sequence': ['Halton', 'Sobol', 'Faure'],
86     'Distance Measure': [c, c1, c2]
87 })
88
89 # Save the DataFrame to an Excel file
90 output_filename = 'quasi_monte_carlo_results128p10d.xlsx'
91 results_df.to_excel(output_filename, index=False)
```

Code Python pour dessiner une configuration initiale et finale sur un plan avec des cercles de rayon R

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from scipy.integrate import quad
4
5  r = 0.1  # rayon d'interaction
6  # Définition des paramètres du modèle
7  n = 20  # nombre de points
8  R1 = 0.05
9  R2 = 0.08
10 R3 = 0.05
11 # rayon de la gaussienne de proposition
12 g11 = 0.1
13 g12 = 0.02
14 g13 = 0.05
15 g23 = 0.2  # coefficient d'interaction pour les paires de marque (1,2)
16 g22 = 0.02
17 g33 = 0.08  # coefficient d'interaction pour les paires de marque (2,2)
18 b1 = 0.5  # intensité du processus pour la marque 1
```

```
b2 = 0.5  # intensité du processus pour la marque 2
b3 = 0.5  # intensité du processus pour la marque 3
NMC = 1000

# marques1
def m1(X):
    n = X.shape[0]
    m = np.zeros(n)
    XTX_inv = np.linalg.inv(np.matmul(X.T, X))
    for i in range(n):
        m[i] = np.matmul(np.matmul(X[i, :], XTX_inv), X[i, :].T)
    return m

# marques2
def distance(x1, x2):
    return np.linalg.norm(x1-x2)

def density(x):
    return np.exp(-(x)**2/2) / (np.sqrt(2*np.pi))
def m2(X):
    n = X.shape[0]
    mus = np.zeros(n)
    for i in range(n):
        sum_distances = 0
        for j in range(n):
            if i != j:
                dist = distance(X[i], X[j])
                integral, _ = quad(lambda t: density(t), 0, dist)
                sum_distances += integral
        mus[i] = sum_distances / (n-1)
    return mus

# marques3
def dist(x1, x2):
    return np.linalg.norm(x1-x2)

def denst(x):
    return np.exp(-(x)**2/2) / (np.sqrt(2*np.pi))
```

```
57 def m3(X):
58     n = X.shape[0]
59     mdt = np.zeros(n)
60     for i in range(n):
61         mus = np.zeros(n)
62         for j in range(n):
63             if i != j:
64                 dist = distance(X[i], X[j])
65                 mus[i] = dist
66         mdt[i] = np.min(mus[i])
67     return mdt
68     def m11(X, m1, r, R1):
69     n = X.shape[0]
70     count = 0
71     for i in range(n):
72         for j in range(i+1, n):
73             if np.linalg.norm(X[i] - X[j]) <= r and m1[i] <= R1 and m1[j] <= R1:
74                 count += 1
75     return count
76
77 def m22(X, m2, R2, r):
78     n = X.shape[0]
79     count = 0
80     for i in range(n):
81         for j in range(i+1, n):
82             if np.linalg.norm(X[i] - X[j]) <= r and m2[i] >= R2 and m2[j] >= R2:
83                 count += 1
84     return count
85
86 def m33(X, m3, R3, r):
87     n = X.shape[0]
88     count = 0
89     for i in range(n):
90         for j in range(i+1, n):
91             if np.linalg.norm(X[i] - X[j]) <= r and m3[i] >= R3 and m3[j] >= R3:
92                 count += 1
93     return count
94
```

```
95 def m12(X, m1, m2, R1, r, R2):
96     n = X.shape[0]
97     count = 0
98     for i in range(n):
99         for j in range(i+1, n):
100             if m1[i] <= R1 and m1[j] <= R1 and m2[i] >= R2 and m2[j] >= R2:
101                 dist = np.linalg.norm(X[i] - X[j])
102                 if dist <= r:
103                     count += 1
104     return count
105 
106 def m13(X, m1, m3, R1, r, R3):
107     n = X.shape[0]
108     count = 0
109     for i in range(n):
110         for j in range(i+1, n):
111             if m1[i] <= R1 and m1[j] <= R1 and m3[i] >= R3 and m3[j] >= R3:
112                 dist = np.linalg.norm(X[i] - X[j])
113                 if dist <= r:
114                     count += 1
115     return count
116 
117 def m23(X, m2, m3, R2, r, R3):
118     n = X.shape[0]
119     count = 0
120     for i in range(n):
121         for j in range(i+1, n):
122             if m2[i] <= R2 and m2[j] <= R2 and m3[i] >= R3 and m3[j] >= R3:
123                 dist = np.linalg.norm(X[i] - X[j])
124                 if dist <= r:
125                     count += 1
126     return count
127     # Génération d'une configuration initiale de points aléatoires dans un carré
        unité
128 X = np.random.rand(n, 2)
129 
130 # affichage de la configuration initiale avec cercles
131 fig1, ax1 = plt.subplots()
```

```
ax1.set_xlim([-0.1, 1.1])
ax1.set_ylim([-0.1, 1.1])
ax1.set_title("Configuration initiale")
for i in range(n):
    ax1.add_artist(plt.Circle((X[i, 0], X[i, 1]), r/2, color='r', fill=False))
ax1.scatter(X[:, 0], X[:, 1])

# Simulation de NMC étapes
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, 2)
    Y = np.copy(X)
    Y[k] = y
    plt.show()
    # Calcul des valeurs de m1, m2, m3, m11, m22,m33 , m12,m13,m23 pour la
        configuration actuelle
    m1_vals = m1(X)
num_points_m1 = len(np.where(m1_vals <= R1)[0])
m2_vals = m2(X)
num_points_m2 = len(np.where(m2_vals >= R2)[0])
m3_vals = m3(X)
num_points_m3 = len(np.where(m3_vals >= R3)[0])
m11_vals = m11(X, m1_vals, r, R1)
m22_vals = m22(X, m2_vals, r, R2)
m33_vals = m33(X, m3_vals, r, R3)
m12_vals = m12(X, m1_vals, m2_vals, R1, r, R2)
m13_vals = m13(X, m1_vals, m3_vals, R1, r, R3)
m23_vals = m23(X, m2_vals, m3_vals, R2, r, R3)
    # Calcul des valeurs de m1, m2, m3, m11, m22,m33 , m12,m13,m23 pour la
        nouvelle configuration
    m1_vals_new = m1(Y)
num_points_m1_new = len(np.where(m1_vals <= R1)[0])
m2_vals_new = m2(Y)
num_points_m2_new = len(np.where(m2_vals >= R2)[0])
m3_vals_new = m3(Y)
num_points_m3_new = len(np.where(m3_vals >= R3)[0])
```

```
m11_vals_new = m11(Y, m1_vals, r, R1)
m22_vals_new = m22(Y, m2_vals, r, R2)
m33_vals_new = m33(Y, m3_vals, r, R3)
m12_vals_new = m12(Y, m1_vals, m2_vals, R1, r, R2)
m13_vals_new = m13(Y, m1_vals, m3_vals, R1, r, R3)
m23_vals_new = m23(Y, m2_vals, m3_vals, R2, r, R3)
    # calcul de la probabilité d'acceptation
  by=b1**num_points_m1_new*b2**num_points_m2_new*b3**num_points_m3_new*g11**
     m11_vals_new*\\
   g22**m22_vals_new*g33**m33_vals_new*g12**m12_vals_new*g23**m23_vals_new*g13**
      m13_vals_new
   bx=b1**num_points_m1*b2**num_points_m2*b3**num_points_m3*g11**m11_vals*g12**
      m12_vals*\\
   g33**m33_vals*g12**m12_vals*g23**m23_vals*g13**m13_vals
    a = min(1, by / bx)
     #print(a)
    # mise à jour de la configuration
    if a == 1:
        X[k] = y

# affichage de la configuration finale avec cercles
fig2, ax2 = plt.subplots()
ax2.set_xlim([-0.1, 1.1])
ax2.set_ylim([-0.1, 1.1])
ax2.set_title("Configuration finale")
for i in range(n):
    circle = plt.Circle(X[i], r/2, color='r', fill=False)
    ax2.add_artist(circle)
ax2.scatter(X[:, 0], X[:, 1])
plt.show()
```

Bibliographie

[Box and Behnken, 1960] Box, G. E. and Behnken, D. W. (1960). Some new three level designs for the study of quantitative variables. *Technometrics*, 2(4) :455–475.

[Chib and Greenberg, 1995a] Chib, S. and Greenberg, E. (1995a). Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4) :327–335.

[Chib and Greenberg, 1995b] Chib, S. and Greenberg, E. (1995b). understanding the metropolishastings algorithm. *Am. Stat.*, volume 49.

[Chib and Greenberg, 1996] Chib, S. and Greenberg, E. (1996). Markov chain monte carlo simulation methods in econometrics. *Econometric theory*, 12(3) :409–431.

[Daley et al., 2003] Daley, D. J., Vere-Jones, D., et al. (2003). *An introduction to the theory of point processes : volume I : elementary theory and methods*. Springer.

[Dobrushin, 1956] Dobrushin, R. L. (1956). Central limit theorem for nonstationary markov chains. i. *Theory of Probability & Its Applications*, 1(1) :65–80.

[Dodge and Rousson, 1999] Dodge, Y. and Rousson, V. (1999). *Analyse de régression appliquée*. Dunod.

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

[Elmossaoui et al., 2020] Elmossaoui, H., Oukid, N., and Hannane, F. (2020). Construction of computer experiment designs using marked point processes. *Afrika Matematika*, 31 :917–928.

[et D. Vere-Jones., 2002] et D. Vere-Jones., D. D. (2002). *ń An Introduction to the Theory of Point Processesż*. Springer Verlag.

[et F. P. Kelly., 1977] et F. P. Kelly., B. D. R. (1977). markov point processes. *Journal of the London Mathematical Society*, volume 15.

[Faure, 1982] Faure, H. (1982). Discrépance de suites associées à un système de numération (en dimension s). *Acta arithmetica*, 41(4) :337–351.

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

[George E. P. Box, 2007] George E. P. Box, N. R. D. (2007). *<Response surfaces, mixtures, and ridge analyses>*. -2nd ed. John Wiley & Sons, Inc., Hoboken, New Jersey.

[Geyer and Møller, 1994] Geyer, C. J. and Møller, J. (1994). Simulation procedures and likelihood inference for spatial point processes. *Scandinavian journal of statistics*, pages 359–373.

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

[Gunzburger and Burkardt, 2004] Gunzburger, M. and Burkardt, J. (2004). Uniformity measures for point sample in hypercubes. *Rapp. tech. Florida State University (cf. p. 73)*.

[Halton, 1960] Halton, J. H. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2 :84–90.

[Hastings, 1970a] Hastings, W. K. (1970a). Monte carlo sampling methods using markov chains & their applications.

[Hastings, 1970b] Hastings, W. K. (1970b). Monte carlo sampling methods using markov chains & their applications.

[Hastings, 1970c] Hastings, W. K. (1970c). Monte carlo sampling methods using markov chains and their applications.

[Hawkes, 1971] Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1) :83–90.

[Jacques, 2000] Jacques, G. (2000). *<Introduction aux Plans d'expériences : plans des plans des doehlerte>*. Dunod. Paris.

[Jaulin, 2008] Jaulin, F. (juin 2008). Processus ponctuels markoviens.

[Johnson M.E. and D., 1990] Johnson M.E., M. L. and D., Y. (1990). *<Minimax and maximin distance designs>*. J. Stat. Plann. Inference 26.

[Kai-Tai Fang., 2006] Kai-Tai Fang., Runze Li., A. S. (2006). *ń.Design and Modeling for Computer Experiments, Computer Science and Data Analysis Seriesż*. Chapman & Hall/CRC is an imprint of Taylor & Francis Group.

[Kiefer, 1959] Kiefer, J. (1959). Optimum experimental designs. *Journal of the Royal Statistical Society : Series B (Methodological)*, 21(2) :272–304.

[Lin., 2008] Lin., R.-B. C. Y.-J. T. . D. K. J. (2008). ńconditionally optimal small composite designsż. *Statistics and Applications*, volume 6.

[Loh, 1996] Loh, W.-L. (1996). On latin hypercube sampling. *The annals of statistics*, 24(5) :2058–2080.

[Metropolis, 1953] Metropolis, N. Rosenbluth A. W., T. A. H. E. (1953). equation of state calculations by fast computing machines. *Journal of Chemical Physics,*.

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

[Moller, 1999] Moller, J. (1999). *ńNotes on Markov Chain Monte Carlo Methods.ż*. Dina Research Summer.

[Morris and Mitchell, 1995] Morris, M. D. and Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3) :381–402.

[PLARD, 2014] PLARD, J. . (2014). *ńApport de la chimiométrie et des plans dexpériences pour lévaluation de la qualité de lhuile dolive au cours de différents processus de vieillissementż*. Thèse de Doctorat en Chimie analytique et Chimiométrie .Université Aix Marseille.

[Ray-Bing Chen., 2008] Ray-Bing Chen., Yu-Jen Tsai, . D. K. J. L. (2008). <conditionally optimal small composite designs>. *Statistics and Applications Nos.1 & 2,*, volume 6 :35–56.

[Ripley and Kelly, 1977] Ripley, B. D. and Kelly, F. P. (1977). Markov point processes. *Journal of the London Mathematical Society*, 2(1) :188–192.

[Senata, 1981] Senata, E. (1981). *Non-Negative Matrices and Markov Chains, 2nd edition.* Springer, New York Heidelberg Berlin,.

[Shewry and Wynn, 1987] Shewry, M. C. and Wynn, H. P. (1987). Maximum entropy sampling. *Journal of applied statistics*, 14(2) :165–170.

[Sobol, 1976] Sobol, I. M. (1976). Uniformly distributed sequences with an additional uniform property. *USSR Computational mathematics and mathematical physics*, 16(5) :236–242.

[Strauss, 1975] Strauss, D. J. (1975). A model for clustering. *Biometrika*, 62(2) :467–475.

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

[Van Loggerenberg-Hattingh, 2003] Van Loggerenberg-Hattingh, A. (2003). Examining learning achievement and experiences of science learners in a problem-based learning environment. *South African Journal of Education*, 23(1) :52–57.

[Warnock, 1995] Warnock, T. T. (1995). Computational investigations of low-discrepancy point sets. *In : Niederreiter H Shiue P.J.S Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing. Lecture Notes in Statistics*.