Submitted to Department of Informatics, Systems and Communication (DISCo), Bachelor's Degree in Computer Science, in association with Department of Physics "Giuseppe Occhialini"

Properties and Implementation of Sequential Expansion of Latin Hypercube Sampling for Simulation Design

Supervisor:
Prof. Davide Gerosa

Co-supervisor:
Prof. Davide Elio Ciucci

Co-supervisor:
Dott. Matteo Boschini

Authored by:
Alessandro Crespi

student number 879186

Academic year 2023/2024

Contents

Abstract

Stochastic methods are crucial for managing computationally demanding simulations because they effectively handle complex systems with many involved variables, offering high scalability where deterministic methods fall short. On the other hand, deterministic methods cut result variance. Latin Hypercube Sampling (LHS), a quasi-Monte Carlo technique, combines the benefits of both stochastic and deterministic approaches. It draws random samples from a model's parameter space within predetermined specific intervals, or strata, a property known as stratification. This research aims to horizontally scale an existing LHS distribution while maintaining stratification as much as possible, a challenging task. The proposed technique, named expanded 'Latin Hypercube Sampling' (eLHS), enhances the flexibility of LHS-based simulations in the later stages of an evolutionary simulation process. Evolutionary apparatus are involved when a simulation's surrogate model is insufficient for experiments, requiring the simulation to continue. The eLHS algorithm, available in the authors' Python package 'latinexpansion', allows users to expand a hyperparameter set and provides control over other desired properties.

1 Introduction

Simulation design is a branch of Statistics focused on building better simulations to enhance the comprehension of phenomena. These simulations are widely used in mathematics, physics, economics, mechanics, and other scientific fields as tools for proving theories, interpolating real sampled values, and generating predictive models to explore uncharted traits and features, perhaps intuitively or roundly developed in the early stages of the study of a specific problem.

Since the advent of hybrid mechanical-electrical programmable calculators such as IBM's machines, the first of their kind to be really useful in engineering, scientists have used them to perform heavy computations for experiments. In 1969, Kennar and Stone's Computer Aided Design for Experiments (CADEX)[15] proposal for computer-driven experiments led to spread up a broad variety of Computer-based simulation methods.

Eventually, computer-based simulations are a set of strategies that benefit from mathematical modeling techniques based on discrete known points placed in a limited parameter space, hereafter samples, and the computer programmability advantage has been used to design, shape and enhance a specific subset of samples that satisfies the desired properties, hereafter sample sets.

The general concept of computer simulation has been defined in the past few decades, it's based on the following key ideas: taking into account a desired behavior F the experimenters have an interest in, F has to be explored through its P real parameter space; the algorithm takes samples from a standard NP-dimensional hyperspace Ω and arranges the sample set for the simulation; afterwar, the simulation is carried out by evaluating F over the sample set and

eventually producing a so-called surrogate model. The class of algorithms meant to implement this abstraction is commonly labeled with *sampling methods*.

To the category of sampling methods belongs the fixed-step (or determined-step) samplers, which include a $h(x)$ function that evenly space samples across the hyperspace or somehow deterministically. For instance, consider $h(x)$ constant function or a Chebyshev nodes [22]. For the matter of this paper, we have focused on the sampling methods whose points are drawn with a specific random distribution.

A critical consideration when evaluating sampling methods is the trade-off between exploration and exploitation. An exploration-oriented sample set maximizes the simulation expertise of seeking key features over the studied behavior. Exploration has been depicted as a model-free practice, so that it does not base its own actions on the model (behavior) evolution or any other on-site response. On the other hand, exploitation is an auxiliary mechanism that aims to better assist the simulation by deploying samples in strategic placements that prevent exploration from exceeding the prediction surrogate made upon a key region (such as overshooting an optima or mismatching a discontinuity for a steep slope).

The most iconic sampling method for simulations is the pseudo-random sampling, lightened of every other criteria, namely the Monte Carlo Sampling or MCS (*Metropolis (1987)*[11]) which has been outlined as basic design of sampling methods. Quasi-Monte Carlo methods are a class of sampling algorithms based on Monte Carlo, indeed, but without a proper random drawing of sample points from the parameter space, instead, points are sequentially extracted in order to satisfy one or multiple criteria as best as the computational time required remains acceptable. Many criteria have been theorized and tested; each of them has a proper application context.

In the scope of this matter, the authors will focus on the space-filling class of criteria and, particularly, on the stratification procedure (also known as non-overlapping property); The former measures the quality of a sample set to be spread evenly across an hyperspace; the way space-filling is defined determines the final aspect of the sampling. On the same hand, a sample set, in P-dimensional space, admits the non-collapsing property if and only if the projections of the samples on a specific dimension fall into distinct intervals I_i. These I_i intervals are fixed-width slices of the limited volume of parameter space taken into examination (e.g., an hyper-volume $[0,1)^N$) and the number of intervals is equal to the number of samples. So, the non-overlapping property prevents samples to fall into a busy area (which has been occupied by another sample). Furthermore, given that the number of intervals and the size of the sample set is equal, it does ensure there are no empty intervals across the parameter space. This property is widely known because it is the fundamental property which Latin Hypercube Sampling sets (LHS) are based on, topic of this paper.

The authors used to work with sampling methods for simulation design, more likely LHS designs [12 23], in Astrophysics related experiments, such as simulating black hole binary mergers, which obviously requires a massive amount of computational time and many different pa-

rameters. The authors often experienced difficulties predicting how long it would take to run a full simulation given N known sampling points in a P-dimensional space. This situation forces them to reserve more machine time on a shared company supercomputer for experiments than they really need. In order to better spend the reserved machine time left after the execution of the first run of sampling points, the authors designed an algorithm that adds up points to the initial LHS sample set using another one that has been drawn by the expansion algorithm in order to preserve stratification of both sample sets together.
Differently, they have experienced another issue related to the accuracy of the surrogate model, result of the simulation consumed after the LHS set. Sometimes, it just happens that the model does not satisfy an eventual accuracy threshold. Instead of throwing the simulation away and compute another one, they would like to expand the current initial set with additional points to aid, the updated model, converging.

The authors of this paper propose a algorithm called *Expansion of an LHS* that, indeed, takes a already existing Latin Hypercube sample set and propose a new set of points samples in the same parameter hyperspace which are supposed to maintain properties stability altogether. The original sample set is referred as *starting set* and the add-on samples as *expansion set*. The whole sample set joined together by both is called *expanded set*.

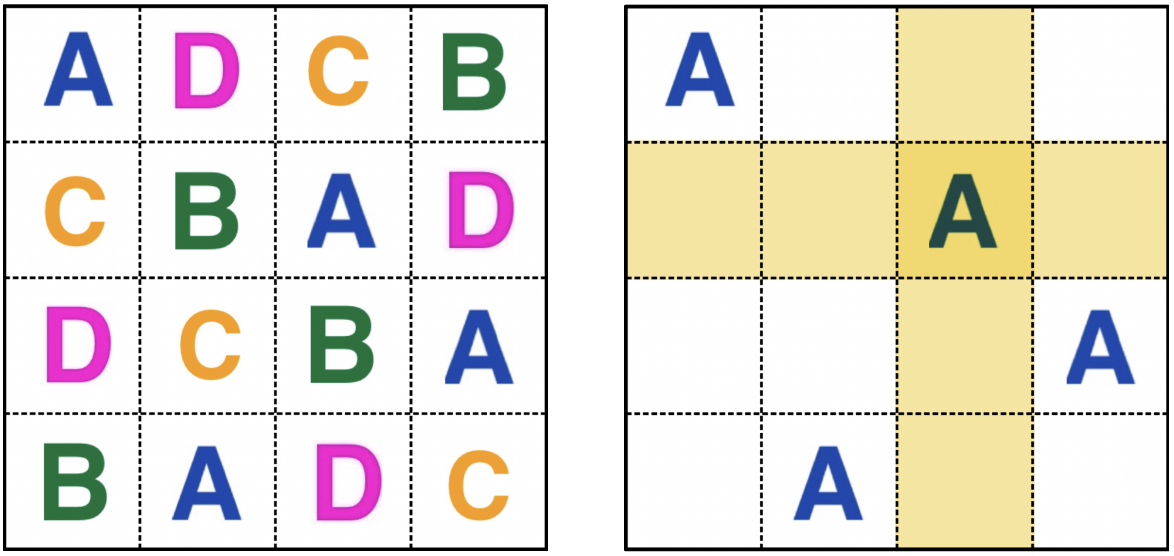
The paper is structured as follows: section 2 yields a Latin Hypercube Sampling brief history and formal definition; in section 3 is shown the research results and discussion of the expansion task issued; section 4 contains experimental and example assessments on the applied expansion algorithm; section 5 the C module package code implementation; and section 6 held the conclusions.

2 *Latin Hypercube Sampling*

2.1 What is an LHS?

According to the Handbook of Combinatorial Designs [1], the first appearance in history of the "Latin Square" has to be attributed to the Korean mathematician Choi Seok-jeong, who described it, using modern terminologies, as a $N \times N$ matrix with N distinct symbols, appearing N times each but precisely once per type for each row and column. The suffix "Latin" has been inspired by the efforts that Leonhard Euler has put into this topic while defining a general theory for Latin Squares [4] and using Latin letters as symbols to fill the square up with. See Fig.1 for an example with 4 objects.

In the scope of this paper, which does not aim to study combinatorial properties of N symbols, we consider the placement of a sample set, which has to well stratifies the matrix. The symbols must occur in the matrix exactly N times each. Instead of consider all symbols, we will look to a single one only.



(a) A Latin 4x4 Square with 4 distinct symbols, both letters and colors either, arranged so that no letter occurs more than once in a row or a column

(b) The skeleton of (a) Latin Square highlighting only the As (or BLUE) symbol positions, it's easily noticeable how the positions does not overlap onto each other's row and column both

Figure 1

Generally, we can speak of modern Latin Hypercube designs as the Euler's Latin Square's matrix concept (depicted as a grid in Fig.1b) but deeming a more complex multidimensional matrix. N symbols are placed in the same way of earlier: each symbol lies exactly once on each fiber (in literature, a "fiber" of a multidimensional matrix is the general term for a one-dimensional substructure in any dimension [or mode] of a tensor. e.g. In a 2D matrix, the fibers of the first and second dimensions are respectively "rows" and "columns"). Moreover, LHS is no longer supposed to place symbols in matrices but, instead, put N random samples on a sectioned hyperspace, whose sections are familiar with the matrix's fibers. The *hyperspace* represents the examined multi-parameter space of the interested model, whereof each axis is associated with a different parameter.

By definition of LHS, the parameter hyperspace has a limited span, which is the hypervolume $[0,1)^P$, commonly used in literature. However, some texts use a different standard, and they let the parameter space take place in a $[-1,1)^P$ hypercube. Every parameter axis is sliced into smaller consecutive intervals of width $\frac{1}{N}$ that consequently depict the sub-region where each coordinate of the points is sampled randomly. In anticipation for further definitions, we provide a indexing system for intervals; in any dimension of the hypercube with N intervals, for each i from 0 to $N-1$, the i-th interval's boundaries are:

$$I_i = \left[\frac{i}{N}, \frac{i+1}{N}\right) \tag{1}$$

for further usage, the right term $\frac{i+1}{N}$ of the interval was called *frontier of the i-th interval* which is shared with the left hand term of I_{i+1}. With no loss of generality, the examples and considerations designed by the authors of this paper assess the random distribution to be uniform in all intervals. The uniformly distributed samples can be transformed by associated transformation functions for any other distribution (e.g. Gaussian distribution).

2.2 How to build a Latin Hypercube Sample Set

In this section it has been marked out mathematically the construction of an LHS sample. This description is widely used for introducing the topic on many textbooks and lectures and it takes inspiration from the work of X.Kong at al.[9]. Let $S = \{S_1, S_2, ..., S_N\}$ be the Latin Hypercube sample set with N number of samples, where $\|S_i\| = P$ number of dimensions. It is comfortable to use the matrix notation S_{ij}, whereof rows are the i-th sample and columns, instead, represents the projection of every sample on each j-th dimension. Then, we introduce the sorted index matrix $A = \{a_{ij} = i\}$ of $N \times P$ dimension as a tool for trace the intervals index, its purpose will be clearer soon.
Given an A index matrix, the preliminary design matrix S is given by:

$$S_{ij} = \frac{u_{ij} + a_{ij} - 1}{N} \tag{2}$$

where u_{ij} is a uniform distributed variable U[0,1). This preliminary design has the peculiarity to have the samples always placed on the diagonal of the hypercube $[0,1)^P$ (Fig.2a). Now, the indexes matrix comes into use, typically shuffling the original A we attain $B = \{b_{ij}\}$ random permutation, which plugging it into eq. (2) describe the uniform random variable S_{ij} living inside the statistical bin identified by the interval index b_{ij}. We explicitly write down the uniform random variable S_{ij} involving eq. (1) boundaries for the ij-th interval as well:

$$S_{ij} \sim U\Big[\frac{b_{ij} - 1}{N}, \frac{b_{ij}}{N}\Big) \tag{3}$$

Along this paper, the authors have used the notation $R \in MC(N,P)$ that implies the R be a Monte-Carlo sample set, then of N points in P, hence R is a Monte-Carlo sample set.
Similarly, the expression $S \in LHS(N,P)$ states that S is a Latin Hypercube sample set of the same parameters. By definition of LHS, which stress that it is a quasi-Monte-Carlo sample set distribution, we can write that:

$$LHS(N,P) \subset MC(N,P)$$

2.3 Degree of a Sample Set

For the purpose of this research, the authors introduced a tool to measure how much a Monte Carlo sample set is close to a Latin Hypercube one, namely *degree of a sample set* with $\mathscr{D}$ identifier. This metric (Eq.6) is designed to assign to a sample set S of N elements in P dimensions an index that ranges from the worst possible distributions of points when approaching 0 (it happens if samples overcrowd into the same very intervals and let many more empty) to a fully conventional LHS when degree equals to 1, such that:

$$0 < \mathscr{D}(S) \leq 1 \ , \ S \in MC(N,P) \tag{4}$$

such that

$$\mathscr{D}(S) = 1 \ \Leftrightarrow \ S \in LHS(N,P) \tag{5}$$

The degree formula cast the arithmetical average of the presence (with numerical value 1) of the projection of every sample in each interval for each dimension:

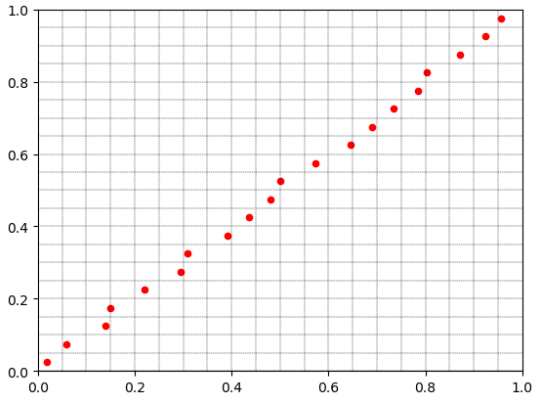
$$\mathscr{D}(S) = \frac{\sum_{j=1}^{P} \sum_{q=1}^{N} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{N}, \frac{q}{N})}(S_{ij}), 1)}{P \cdot N} \tag{6}$$

where $\mathbf{1}$ is the indicator function (see Appendix 7.1), the variable q is another way to represent the sorted index matrix $A = \{a_{ij} = i\}$ for a more immediate comprehension. The *min* operator states that the presence of several samples' projections in a specific interval q does not weight up the whole term, which would be at most 1 even if multiple samples lie in q. Hence, the degree formula ignores the overlapping samples onto the same interval, hereafter only *overlaps*.

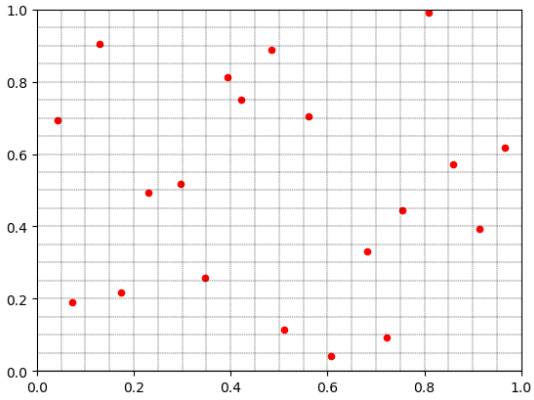
2.4 Additional properties

In this section, the authors present some very important additional properties that could greatly improve the accuracy - or the quality in other terms - of the surrogate model produced by the consumed simulation.

A well known issue with LHS designs is well depicted in Fig.2a. It's obvious that it must be considered a poor simulation design for the most of the experiments. Usually, implementations of LHS add some peculiar properties to the design. We can highlight two superclasses of properties which LHS may be joined with: model-free properties and model-based properties.



(a) A poor Latin Hypercube sampling in 2 dimension. Despite it stratifies completely the square, it busts the experiment parameter space.
The diagonal sampling depicts how insufficient a pure LHS is to achieve a good simulation.



(b) LHS with space-filling property prevents samples to cluster together, low-discrepancy reduces unintentional patterns. This sampling has been generated using scipy.stats.qmc.LatinHypercube utility class. The sample generator bears random permutations of coordinates to lower the centered discrepancy [24].

Figure 2

2.4.1 Model-free and model-based simulation designs

As the names may suggest, the latter implies the experimental design to involve some the peculiarities of the function to evaluate e.g. the shape that is expected, the initial or boundaries conditions, any well known critical region. Given that this approach is knowledge-driven, ”What do we already know about?”, can knock down pretty much the computational time required for the whole experiment with eventually high accuracy, but it can be equally easy that results may be biased, based on assumptions that may prove wrong later, or make hard any further effort from other scientists to retake the experiment and confirm the results if the assumptions are not completely clear. On the other hand, the model-free is the formal way to depict a fully independent sampling of the parameter space from the experiment which it has been designed for. Because of not many criteria have been left, all of the properties in this class are based on inter-element relationships, which convey that for each of them has a heuristic meant to quantify how much each sample is well-placed among the others. The shorthand for such a class of criteria is *space-filling properties*.

Beside the many interesting considerations and suggestions that the model-based class of criteria has to offer to mathematicians, the space-filling properties has highly excited the experimenter community through the decades, producing creative and curious features that the samples could experience among each others. A well-known criteria is, for instance,the maximum minimum distance among points, so-called maximin distance criterion, and its typical

counterpart that minimize the maximum hole, so-called maximin distance. The assumed metric is the P-dimensional Euclidean distance in the hyperspace.

By the way, the characteristic non-overlapping property of the LHS itself is actually a space-filling property. It ensures that, for example, the average distance between the consecutive projections of a group of 3 samples on the same axis is at least 1/2 distance units and at most 3/2 d.u.

2.4.2 On the optimal search tree

Despite the ease to describe a criteria for a set of elements, for instance the maximin distance, generating from random drawn a quasi-optimal solution is a nowdays a very serious issue to deal with. Actually, the algorithm should *seek* for the best candidate among a broad pool of possibilities.

In Computer Science, all the possible variants of a decision-demanding problem is mapped in a search tree, a graph-based structure that is explored by querying at each node which subpath should the algorithm undertake [6]. Some trivial well-defined path search algorithm adopts a nice binary search tree solution which, in the worst setups, is solved in linear time $O(n)$ *Daskalakis et al. (2007)*[2]. However, many other undeterministic, or let's say stochastic, algorithms are easily NP-complete problem, in their average-case. Such that is the synonym of computational-demanding algorithm for non-binary search trees. Some solution was proposed by John McCarthy, and then commented by *Knuth et al. (1975)*[7], namely the Alpha-Beta pruning algorithm which, in short, when the algorithm deepen and evaluate the tree branches it overestimates the current solution found at some extent, in the end the algorithm cuts off a lot of branches deemed not useful anymore. Alpha and Beta are two hyper-parameters that ponders the current solution value.

3 *Expansion of a Latin Hypercube Sampling*

The LHS paradigm allows to implement several criteria over a rigorous grid, which forms the basis for the sequential creation of samples. Thus it's frequently utilized in engineering environments for surrogate manufacture of complex systems - see also section 2.4. For example, the LHS is used in hyperparameter tuning/optimization of Machine Learning models (*Koch et al. (2018)*[8]), environmental and water system analysis (*Sheikholeslami et al. (2017)*[18]), structural reliability analysis (*Olsson et al. (2003)*[13]).

There are ongoing efforts to improve the LHS capability. As previously stated, add criterions first. Then, by rethinking the foundations of the algorithm. The reader has to acquaint that Latin Hypercube technique, widely implemented, is labeled as a *one-stage* algorithm in literature. The fact that all samples are distributed and assessed "on the first run" bestows this

adjective. It is relevant to clear out that the actual creation and propagation of points is not properly implemented as the resulting of a single sampling random variables in agreement with Eq.3, but instead it is a sequential drawing of points - or, possibly, a parallel drawing of several ones for optimization reasons - given a desired N_1 number of samples, the newest one has to be pulled out from a pool of optimal candidates in order to improve the criteria applied, such as maximin space-filling distance (see section 2.4.1). The feature "one-stage" highlights that it could be possible to have many more "stages" of the algorithm. It's a game of perspectives: inside a current stage, the policy for drawing a fixed number of data points is aimed at pulling out point given the other ones; instead, a multistage policy is related to a more evolutionary approach, aiming to enhance the sample set stage by stage. Manipulating an already instantiated LHS may sound difficult because LHS is not designed to add points - or, at least, it is not supposed to consider it. Indeed, it is reasonable to assess that by adding points, one by one, over a full grid of N_1 intervals of an LHS, and then reshaping such a grid in $N_2 > N_1$ intervals, will lead to collisions (overlaps), which represents an issue - see section 3.1.

The process that embodies the evolution from a precursor LHS to his next-stage has been called *Expansion* by the authors; the resultant of the expansion process is the so-called *expanded set*. Please refer to Fig.5 for the visual explanation of expansion.

Along this section, the authors used to refer several times to the expansion process without precising how the new samples are going to be placed, the actual process is described in section section 3.3.3.

3.1 The task of multistaging sampling

The multistage approach raises concerns regarding the consistency of the non-collapsing property, which is valid for one-staged setups. Fig.3 depicts an experiment carried out upon scipy's $S \in LHS(N_1 = 10, P = 2)$ with samples displayed over its appropriate grid of N_1 intervals per dimension (Fig.3a). The experiment consists in evaluating the behavior of the fixed S sample set while the grid is "growing" - intended as creating a brand new grid with a greater number of intervals - one by one for three times. Light grey-colored rows and columns are vacant, meaning that no projections are located there; rows and columns marked in red indicate the location of overlaps. After the first add-on (Fig.3b), the grid shows two overlaps in row #3 and in column number #3 as well; the next stage (Fig.3c) shows overlaps too, four in total, more than before. In the end, Fig.3d has no overlaps at all. The distribution of occurrences of overlaps when the grid grows, depends on the initial sample set. The authors discuss about this topic later in section 4 [⟵ be more specific].

Regarding the LHS directive, the multistage experiment proposed shows that growing the interval grid could lead to downgrade the stratification in some extent (Fig.3b and Fig.3c) or give a new opportunity to fill the new empty space with a set of brand new samples without

compromise the whole set optimality.

3.2 Degree of an Expansion

A precise heuristic that expresses how near the current expansion is to a perfect expansion, given the desired expansion magnitude M and the beginning state, is necessary for developing a new sample set. Since the metric in Eq.6 is no longer sufficient, the variation *expanded degree* has been offered. The purpose of the *expanded degree* is to convert the sample set S of N elements into an index when the sample set is compared against a P-dimensional hypercube grid of $N+M$ intervals per axis - this is where it deviates from Eq.6. Here it follows:

$$\mathscr{D}(S,M) := \frac{\sum_{j=1}^{P}\sum_{q=1}^{N+M} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{N+M},\frac{q}{N+M})}(S_{ij}),1)}{P\cdot(N+M)} \tag{7}$$

Each I interval contributes to Eq.7 value with a share of:

$$\mu_I = \begin{cases} \frac{1}{P\cdot(N+M)} & \text{if any } x \in I \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

which makes Eq.7 a discrete quantity that ranges from $\frac{1}{N+M}$, if every sample of a non-empty set lies in one single interval per axis, to 1, which represents the perfect LHS expansion.

A perfect expansion for $S \in MC(N,P)$ over a $N+M$ space grid of a number M of intervals (per dimension) empty, given that, by definition of LHS, N samples fall in N distinct intervals (per dimension). Then, the only valuable μ shares are given by N intervals while the other M set of intervals are empty. From this statement, we can provide an upper limit for the expanded degree by starting from 1 - the maximum possible index of a perfect LHS sample set - minus the total weight lost during the growing of the grid, which corresponds to the total number of *voids* times Eq.8. The total number of *voids* is equal to $M\cdot P$ because they are evenly spread across each dimension. So, Z is a *perfect expansion* of S with M additional intervals if and only if:

$$gr_{max}(S,M) = 1 - \frac{M}{N+M} \tag{9}$$

The *initial set* $S \in MC(N,P)$ plotted over P hypercube of $M+N$ intervals (per dimension) is called *perfect expansion* if and only if S expanded degree Eq.7 is equal to the *upper expanded degree limit* Eq.9.

As the name may suggest, the perfect expansion is the best possible candidate to produce an optimal stratified sample set, so an LHS, as result of an expansion process described in section 3.3.2.

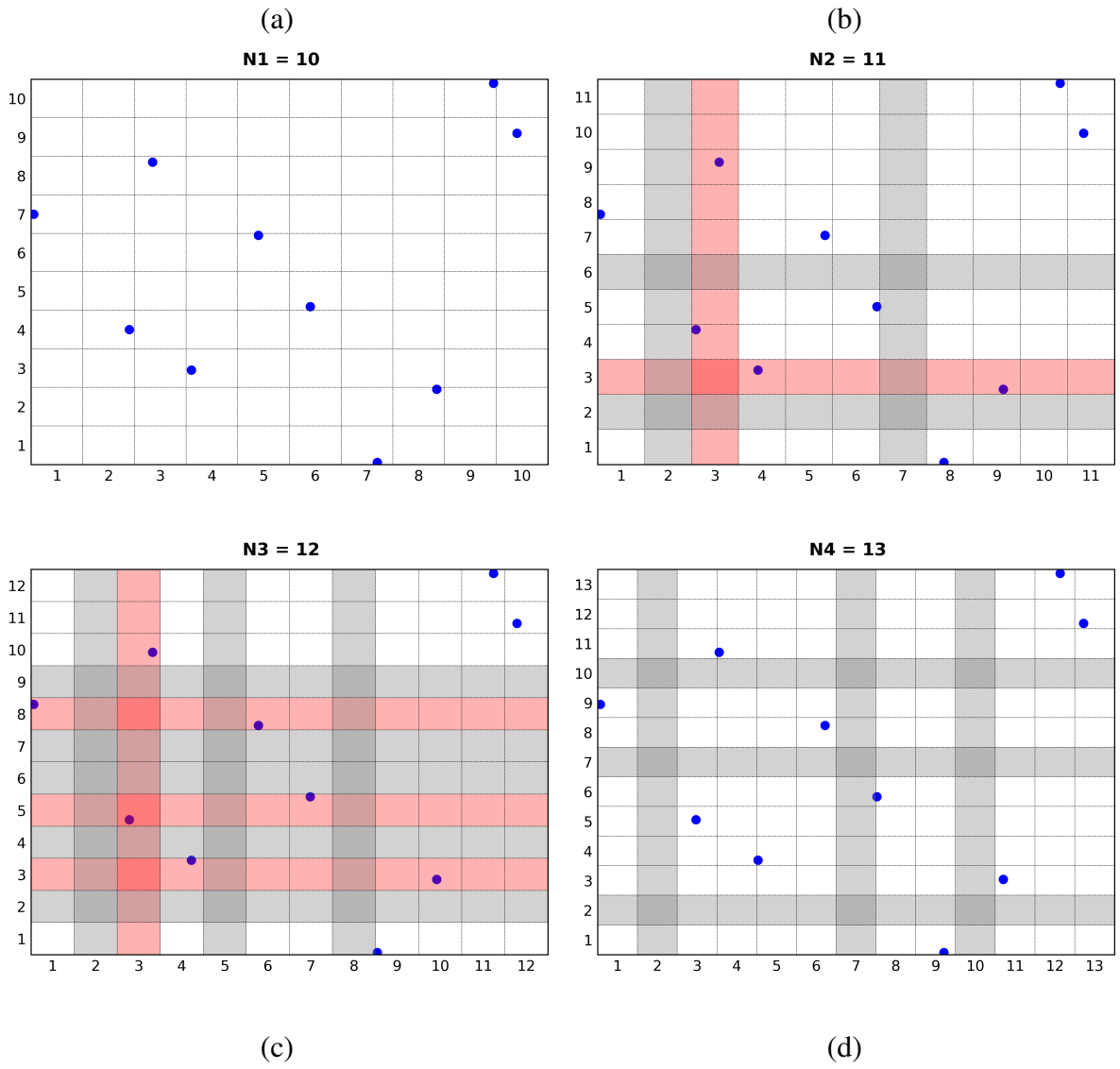


Figure 3: To demonstrate the behavior of the grid, $S \in LHS(N_1 = 10, P = 2)$ sample set is allocated and projected over the grid as it grows one by one. Rows and columns marked in red indicate the location of overlaps, while light grey-colored rows and columns are vacant. In (a) $N_1 = 10$ LHS samples are displayed over a grid of N_1 intervals per axis; (b) shows the first step of the growing grid: two overlaps occur in the horizontal and vertical intervals #3, two intervals per dimension are vacant one of which derives from the expansion vacancy and the other one is caused by the overlap; (c) samples displayed over $N_3 = N_1 + 2$ grid: on the vertical axis there are 3 overlaps and 2 (from the expansion) + 3 (from the overlaps) vacancies; (d) the grown $N_4 = N_1 + 3$ grid have no collisions, then S is a perfect expansion with $M = 3$ expansion magnitude

3.3 The expansion process

The expansion task was initially handed at the very beginning of section 3.1 as a evolutionary process which augments the S starting sample set to a next-stage state with increased number of elements M, placed as better as it can to maximize criteria. However, we show in section 3.2 that an expansion may demote the stratification of the resultant expanded set Z, which can be measured with the metric Eq.7. In this section the authors propose how to place the new samples to achieve maximum "LHS-ness" over any M expansion magnitude needed and introduce its potentialities for future researches - see more in section 6.

In first place, the proposal is delivered by studying the basic case of a perfect expansion in section 3.3.1, then extend to the general case with any non-perfect expansion outcome.

3.3.1 State case - Perfect Expansion

An initial instanced sample set S of N elements can be perfect expanded if and only if S after the M-th growing step of the P-hypercube grid has the maximum degree (Eq.9). Furthermore, Fig.3d well depicts what the experimenters would expect before setting down an expansion set E. We notice the best candidate extent is likely the empty intervals (in light grey) because they do not interfere with other well-formed intervals. It is also imperative that the new samples do not overlap onto each other over the vacant space (*vacancies* or *voids*, as we previously called them).

First of all, it's necessary to trace each void's index. Here it comes into use again the permuted index matrix, previously used in section 2.2 to scatter the samples across the hyper-parameter space. The matrix of voids $P \times M$ is composed by the row vectors:

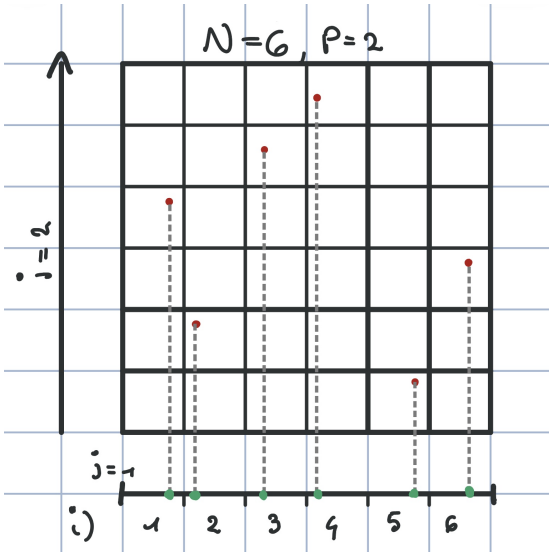
$$\mathbf{V}_j = \left(q : \nexists\, x \in S_{ij}\ s.t.\ x \in \left[\frac{q-1}{N+M}, \frac{q}{N+M} \right) \right) \quad , \quad \forall j = 1...P \tag{10}$$

which should be element-wise permuted along each row to prevent Fig.2a diagonalized situation. Then, the expansion set E collects all the newly generated samples which are a distribution likewise Eq.3 but using V_{ji}.

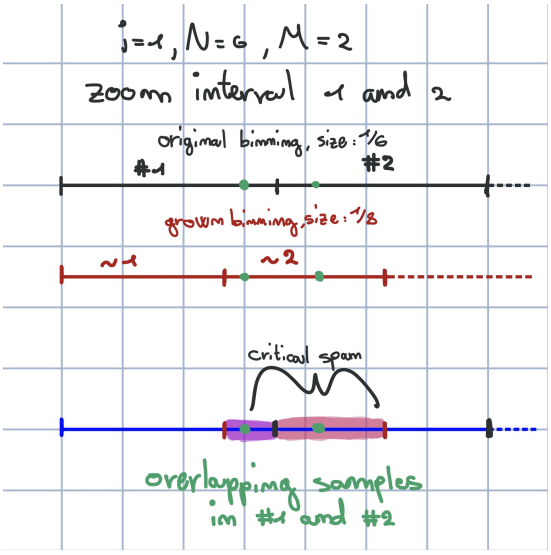
In the end, the initial S set is concatenated with expansion set E, originated from the voids of the $N+M$ growing grid, in a perfect expansion, that guarantees the fulfilling of maximum stratification (degree = 1).

A perfect expansion is a intuitively rare event: each sample has to avoid overlaps. This could eventually happen if two sample projections are spaced more than $\frac{1}{N+M}$ apart (new size of the intervals), But if they are closer, another condition has to be fulfilled: for any M close enough to N, a *critical span* exists across the *i-th frontier* (see right after Eq.1 for definition) - which shares its boundary with the next interval. The critical span is the intersection area between two neighbor samples intervals and the eventual interval raised across them (Fig.4b). Given

that LHS is not supposed to consider further additional points, it might happen that samples are placed in critical areas (after a M growing step). An example of how can a couple of collapsing samples can occur is shown in Fig.4. In the example, the first sample (in Fig.4b) has been generated in the critical area that is the intersection between ~ 2 and #1 original interval. On its hand, the second sample lies upon the same critical area between ~ 2 and #2 original subspace. Hence, in the example the two samples are overlapped.



(a) An LHS of $N = 6$ samples expanded for $M = 2$. on the horizontal axis are projected the horizontal component of all samples. [SKETCH - ne farò in digitale di migliori]



(b) Zooming in #1 and #2 interval, it's show how both shares a critical area (on the bottom) whereof each one may have been distributed in it. [SKETCH - ne farò in digitale di migliori]

Figure 4

The number and displacement of overlaps and vacancies are pretty hard to predict: they are strongly correlated to the Eq.2 sample random distribution. However, we experimentally observed that some expansions are peculiar: exactly all expansions with multiples of N are always perfect. That's because critical spans - again, that are given by the intersection of the specific new intervals that cross the old frontiers with the previous intervals - are always void. Take Fig.4b but with grown binning size equals to $\frac{1}{12}$ instead of $\frac{1}{8}$ (so, with $N = 6$ then $M = 12$): new intervals divide perfectly the old ones, without ever crossing any frontier. This concept is properly explained in Appendix 7.2. Explicitly:

$$\forall K \in \mathbb{N}^+ \quad : \quad S \in LHS(N,P) \Rightarrow \mathscr{D}(S, K \cdot N) = gr_{max} \tag{11}$$

[I'm not exactly sure if it should be $\Rightarrow$ or $\Leftrightarrow$]

3.3.2 State case - General Expansion

During the upscale of S from N to $N+M$ number of elements, it comes along with a variable number of overlaps on j-th axis O_j. In section 3.2 the authors stress the relation between tuples $\mathbf{V}_j$, M, and the amount of collisions O_j on each j-th dimension. Specifically:

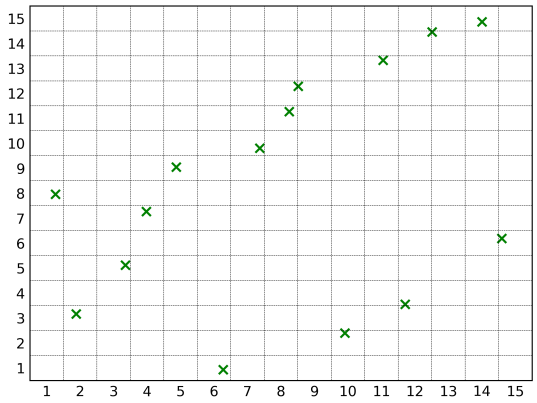
$$\| \mathbf{V}_j \| = O_j + M \tag{12}$$

whereof the overlaps count equals to zero, the expansion has been perfect (see section 3.1). In a general case of expansion, the overlaps amount is most likely not equal to zero in some dimensions.

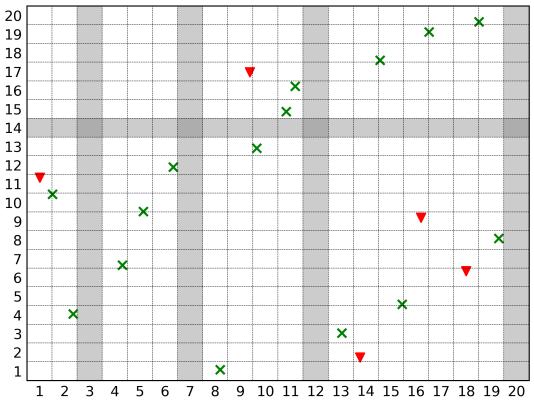
The case study implies that the creation of the expansion set iE is not trivial anymore because of the irregularity of the vacancies set. By referring to Eq.12, unlike what was stated before, V would probably be no matrix at all but, instead, a set of heterogeneous tuples. If V components was homogeneous, it would have been a inherited matrix. The number of $\mathbf{V}_j$ interval indexes would likely be more than M sample's projections to commit. The expansion algorithm has to pick up a reasonable subset of M void entries, and thus to discard an amount of intervals equal to the number of overlaps O_j. Therefore, given the sub-hyperspace settled by the joined selected voids, in order to plot an M amount of new samples, it will pick up an $P \times M$ submatrix (that mimics Eq.10) of V set. The submatrix should be handled being aware that it would effect the samples layout which may better improve another coherent criteria chosen (such as low-discrepancy or Maximin space-filling).
The selection process of vacancies from an irregular V voids set is described by a function $\sigma : N \times P,\ M \rightarrow P \times M$, namely *perfectify* or *vacancy reduction* (for the matter of giving names to anything).
In this section, σ reduce function trivially picks up M intervals randomly per dimension and build up a permuted Eq.10 vacancies matrix, which will be plugged into Eq.2 to produce an expansion set.

In other words, σ criterion extracts from each j-th axis of the vacancies set $\mathbf{V}_j$ a fixed M number of elements which are going to compose the sub-hyperspace where M samples will be placed using at least the non-overlapping property. The function σ discards O_j number of intervals (Eq.12), then creating an amount of voids of the same quantity. Hence, the number of overlaps O_j determines the number of void intervals after the expansion is consumed.

In this paper, the term *quasi-LHS* refers to a non full-degreed in-th stage of expansion descended from a proper LHS.

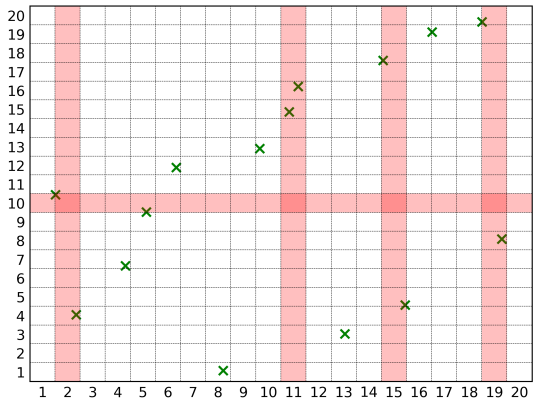


(a) First-stage LHS of $N = 15$ samples in $P = 2$ generated with scipy's qmc library.

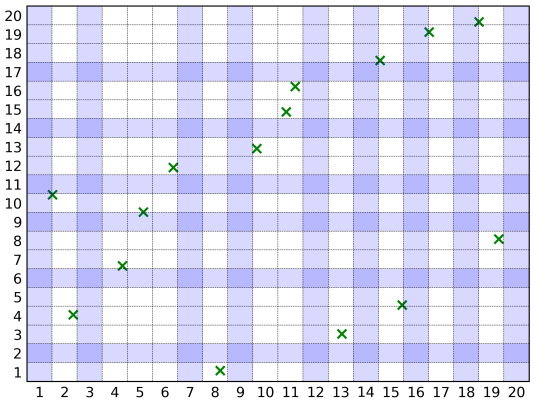


(b) Expansion of (a)'s LHS with section 3.3.3 eLHS algorithm given $M = 5$ new samples. Note that the light grey marked intervals are empty and, according to section 3.3.2, they are related with the overlaps distribution in (c)

Above is shown a two-staged quasi-LHS. (a) is the first original LHS and (b) is next stage expansion of it.



(c) Red intervals have two sample's projections in it and reduce the space stratification.



(d) Blue intervals are empty. They represents the best candidate spots to place new LHS samples. Every interval all together has been referred as the sub-hyperspace of vacancies.

Re-binned (a)'s grid with $N + M$ intervals and plotted against the starting LHS.

Figure 5

3.3.3 The eLHS algorithm

The LHS expansion algorithm, namely *eLHS*, pushes a starting Latin Hypercube S to the next stage $Z = eLHS(S,M)$ by maximizing the hyperspace stratification at most, along with other eventual criterions.

1. Instance a V vacancies set of P tuples - which may have different lengths (Eq.10) because every dimension has an arbitrary number of voids (Eq.12). The list of all indexes of the $N+M$ grid is filtered accordingly with Eq.10.
   Visually, the algorithm re-bins the original S grid to achieve $N+M$ intervals. By plotting it against S, the reader can visualize where samples overlap and where there are voids.

2. Reduce V vancancies set to a suitable indexes matrix $V' \in Matrix(P,M)$ by extracting from each $\mathbf{V}_j$ tuple M elements - which are going to compose the expansion binning grid - using σ reduction criteria (see section 3.3.1). If there are no overlaps (meaning S has maximum expanded degree Eq.9) then V is implicitly equal to matrix V', so no reduction criterion is required to be applied.
   We propose a paradigm which σ can be built on. Reduce (abuse of notation) the issue to an harder problem with integer constraints so that solutions are the void interval's indexes. Before generating optimal samples (in step 3.), select the optimal void intervals by considering a trivial "puppet" sample that lies in each possible combination of void intervals. Then apply Branch&Bound methodology [TODO: find good refs for BnB] to find the best integer solution(s).

3. Generate new points over the sub-hyperspace outlined by the permuted V' indexes matrix. Currently, Scipy does not implement the instancing of a LHS over a discontinuous space yet.
   The eLHS' implementers should achieve drawing optimal data from a discontinuous space on their own, based on desired additional criteria and time complexity requirements. However, we suggest the following algorithm structures.

   (a) This procedure was inspired by Shang et al.[17] work based on the pioneering CADEX research (*R. W. Kennard (1969)*[15]). It was designed to be fast and flexible given some appropriate criteria to optimize:

      i. set the variable $\Lambda = M$, iteration index $k = 1$ and initialize the expanded set E empty;

      ii. generate Γ_k a random pool of, let's say, α_k number of new random sample points. The hyper-parameter α_k should be much greater than Λ. We would recommend a fast-and-reliable MCS;

      iii. ignore all points that fall outside V' space;

iv. from Γ_k select a subset of optimal samples γ_k of $\lambda_k \leq \Lambda$ number of points.

† If $\lambda_k = 1$ it is like drawing the very optimal point each iteration;

† If $\lambda_k > 1$ the algorithm should yield a bunch of samples which satisfies or optimize any eventual sub-property. Definitely, the basic rule that γ_k has to satisfy is the non-collapsing property after being joined with $S \cup E$;

v. pop out from V' the intervals where $gamma_k$ elements lie in;

vi. append the optimal subset: the sample set $E = E \cup \gamma_k$;

vii. set $\Lambda = \Lambda - \lambda_k$.
If $\Lambda = 0$, return E.
Otherwise, set $k = k + 1$ and go to step (ii).

(b) The other way to generate E recalls what was said in section 2.4 [not said yet, I'll do it] about search trees. Indeed, as many sampling methods implementations adopt, use a search tree enhances any desired criteria by a lot but trade-offs with time complexity.
However, following there are some further suggestions that the authors have considered notable:

i. A basic search three, branches take a random not yet used vacant space from V' (a P tuple of intervals for each dimension) and shoot K samples, then selects whose the optimal. The deeper the tree goes, the higher is the number of samples drawn. On the leaves there are every expansion sets E computed, the number of leaves is $(M!)^P$. The time complexity is $O\left(K^M \cdot (M!)^P\right)$. This search tree should came along with a reasonable branch pruning rule that reduces computational time and approximate the optimal solution.

ii. If multiple criterions are given, build a multi-agent adversarial search algorithm [drop some references from AI here] where each agent (one for each property) tries to optimize its own criteria against the other's (without compromising the global score, the actual objective function linear composed by every criteria).

4 Examples

This section seeks to evaluate eLHS' sampling performance against other common options, beyond other use cases of the algorithm we have proposed. Specifically, we adopt the classic Monte-Carlo Sampling in the role of mark the performance lower limit whereof other methods might not exceed. In fact, pure MCS is widely deemed as a very poor sampling procedure of an hyperspace [21]. On the other hand, Sobol' are a type of low-discrepancy sequences (for specific details, see Appendix 7.4), whose optimality on the topic of sensitivity analysis was observed, rather than LHS [16], and suitable for singular integrals evaluation [14].

Scipy's implementation (*scipy.stats.qmc.Sobol*) adopts a pre-computed list of base direction numbers (*Sobol et al. (2011)*[19]) which act as generative seeds. Furthermore, the Python class *scipy.stats.qmc.LatinHypercube* - LHS sample set generator used in this paper - adopts low-discrepancy optimization criteria along with non-overlapping property. In the end, the Monte-Carlo generator used belongs to Numpy's *numpy.random* utility package, the specific method is *numpy.random.rand* .

The experiments were designed to explore two different important sides of the expansion task, beyond the proposed algorithm:

(1) The expected performance of E expanded set, descendant of an LHS, with number of samples $N_{tot} = N_{init} + \sum M_e$, should be at least *comparable* with another different standard S Latin Hypercube sample set with same number of samples N_{tot} and built with coherent criteria. The actual performances of the E should be determined only by the effectiveness of the properties, as usual, and do not wane as the evolution proceeds.

(2) The expansion process gives the experimenters a new dimension for developing sampling algorithms. Govern the expansion stages may unleash inner properties and high simulation performances that are hard to replicate with monolithic techniques.

Along this section, we will refer to the above sentences as Reason (1) and Reason (2).

The evolution procedure that we used for every example that involves a growing number of samples is different for some experimental set. First of all, a eLHS sample set is deployed as an LHS in first place, then, at each new iteration a brand new expansion is provided, built on the very initial sample set.

A variant of this first experimental eLHS is considered. On this scope, the evolution does not descend from the very initial LHS anymore, instead it is cumulative: at each iteration the samples generated during the previous stage works as new starting set. So, when the simulation advances, the samples are "Forwarded" to the next stage, namely *eLHS-F*. In the writers opinion, while eLHS experimental set is the pure essence of how a sole expansion stage behaves, eLHS-F represents the very "applied use" of the algorithm: if, during a simulation, an expansion is needed most likely many more are coming along with it.

Sobol' sequences, MC sets and for some LHS sets the expansion paradigm is ruled by adding a fixed number of random samples across the space, without any other specification. The reason for such a trivial expansion is introduced in **??**, briefly: we, the authors, aims to evolve an ongoing simulation which, for instance, has not achieved error requirements or we want only spend a limited more computational time at some extent. Obviously, treating an expansion of Sobol', MC or LHS as a whole new simulation with brand new data does not fit with our perspective.

We highlight that some example results are the arithmetic average of a fixed bunch others of equal parameters, so-called agents. This was done because of the high variance of the results

detected during the simulations. Take into account that eLHS, eLHS-F and LHS are all based on the same initial Latin Hypercube sample set, then the respective expansion can be deemed as branch choices of the same sample set.

The eLHS software used was implemented by the authors section 5 following the guidelines in **??** whereof σ functions randomly pops vacancies out the voids set. The optimal search criteria adopted is pure maximin distance of the whole expanded set. The optimal solution is pulled out from a pool of $K = 10$ acceptable expansions. Clearly, such an implementation is trivial and improvable, pondered over time complexity limits assumed and to embrace a real criterion augmentation sufficiently considered for the sake of this paper.

4.1 Example I - Stratification evolution

An important topic of this research was about how to preserve stratification while the sampling evolves. To give an experimental observation about the trend of the degree metric function - again, which quantify the non-overlapping property - a sample set pool has been instanced in first place, then the sample is grown by a step factor equal to 5.

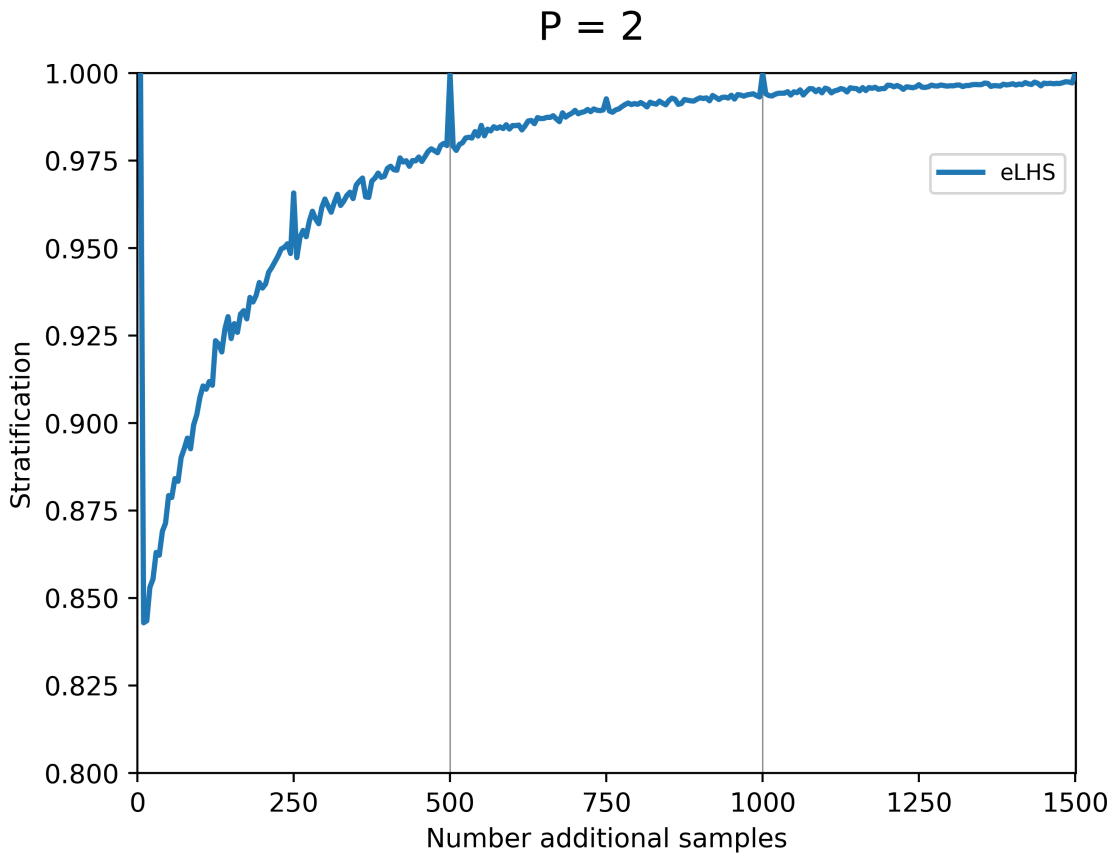
Fig.7 depicts 3 scenario with increasing dimension number P equals to 2, 5 and 100. On the horizontal axis varies the M expansion number of samples, on the vertical axis is plotted the result of $\mathscr{D}(S,M)$ for each sample set. The initial sample size is $N = 500$. Whatever a curve touches the upper stratification heuristic limit 1.0 means that the expanded set at the given M is a perfect expansion.

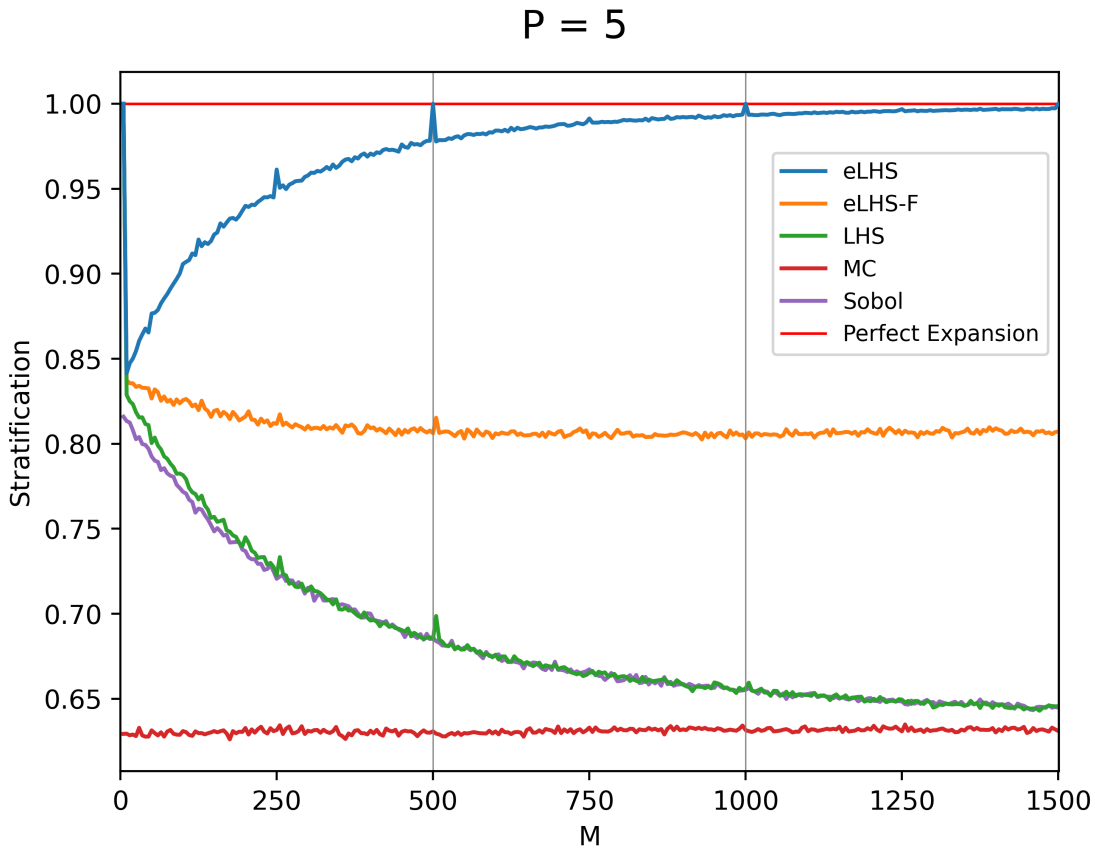
In the scope of eLHS: for every scenario, we have confirmed experimentally what Eq.11 stress: for each multiple of N, the eLHS expansion of any S is perfect.
The curve also shows that growing the initial LHS with a relatively small M drops the stratification index by approximately 17% . This evidence tell us that expansion should be held by almost doubling the sample space, at least.

Unexpectedly, eLHS plot shows evidence of other *peculiar* values of M, not yet identified, that seem to optimize stratification locally. Those points occur in every LHS based experimental setup (eLHS, eLHS-F and LHS) at $M \simeq \frac{N}{2}$ and for all P setup. Then, we observed other, less relevant peaks around $M \simeq \frac{N}{4}$ and $M \simeq 2N$. The behavior might suggest a fractal pattern in the degree distribution probability. Such a feature might be matter of further researches.

eLHS-F behavior differs from eLHS because of what we stressed earlier: for small Ms, it does not worth to expand at all. eLHS-F - that expands the previous expanded set by 5 at each iteration - drops suddenly to around 83% of stratification and then it is keeping decrease until stationing at 81% . In order to, underline the importance of the growing step, we will keep the non-optimal step size for eLHS-F next applications.

Sobol' sequences and LHS samples are completely unable to prevent stratification collapse, approaching Monte-Carlo results, which poor performed the tests at every step and bottom limit the simulation. So, unguided Sobol' and LHS both will converge to a Monte-Carlo performance at a swift rate which even increase with dimension.
Notice that Sobol' and LHS trend to converge. This could suggest how Scipy's produce both: LHS sets are generated from precomputed direction numbers as Sobol' do, then the software filter for those optimized sequences which are Latin Hypercube also. Then Scipy's might be LHS and quasi-Sobol' as well. Disrupting stratification by adding random points each step compromises the non-overlapping property, then the former LHS sets would drop as Sobol' sequences do.


P = 2
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
0.800
Stratification
eLHS
0
250
500
750
1000
1250
1500
Number additional samples


P = 5
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
Stratification
eLHS
eLHS-F
LHS
MC
Sobol
Perfect Expansion
0
250
500
750
1000
1250
1500
M

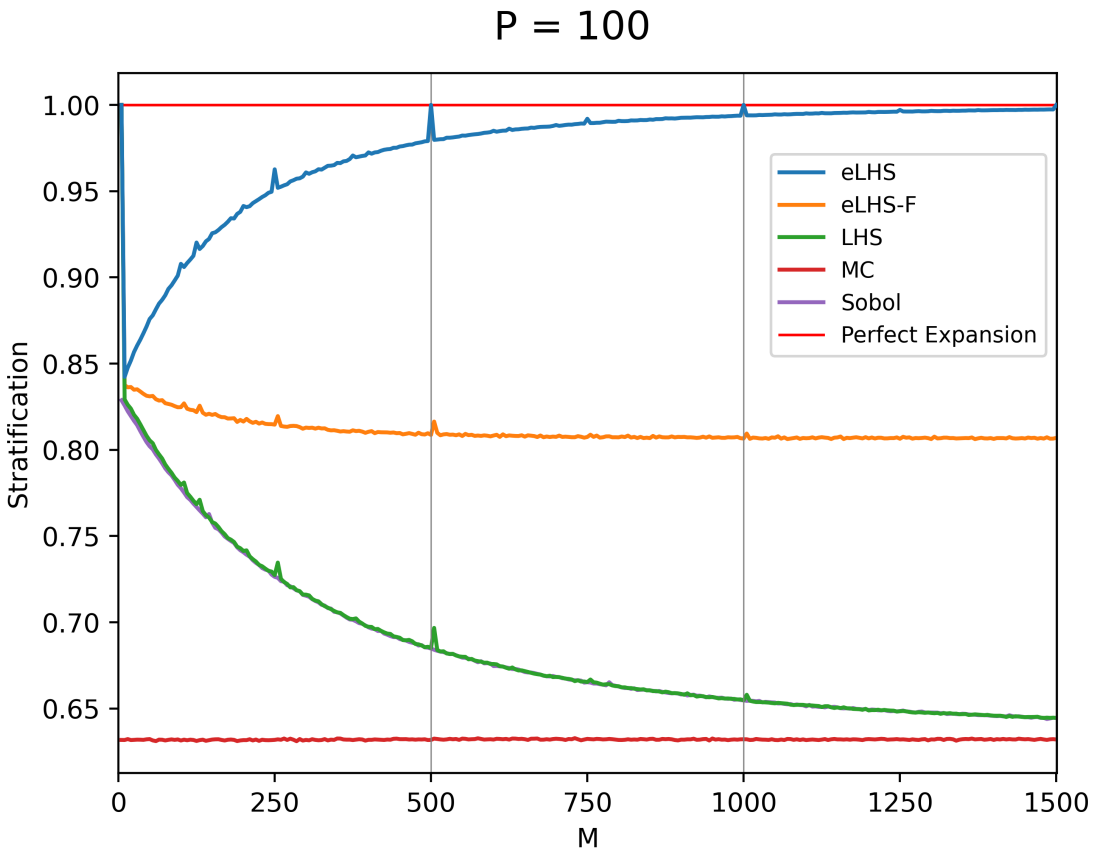


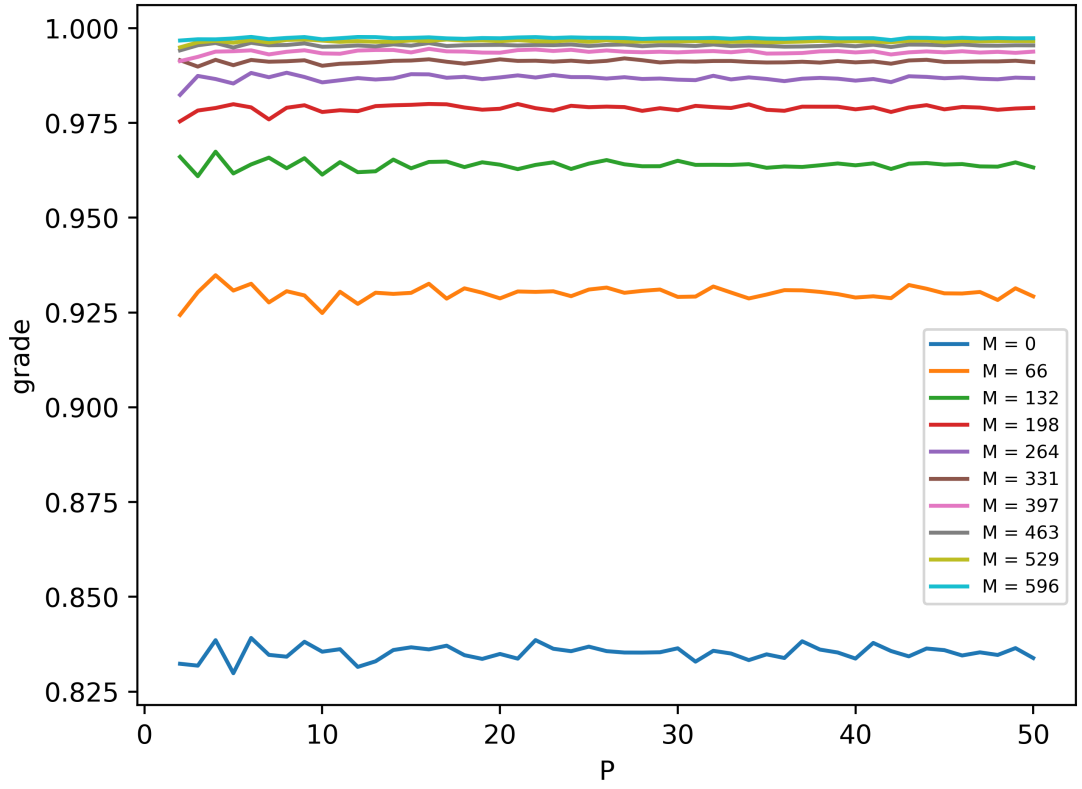
Figure 7

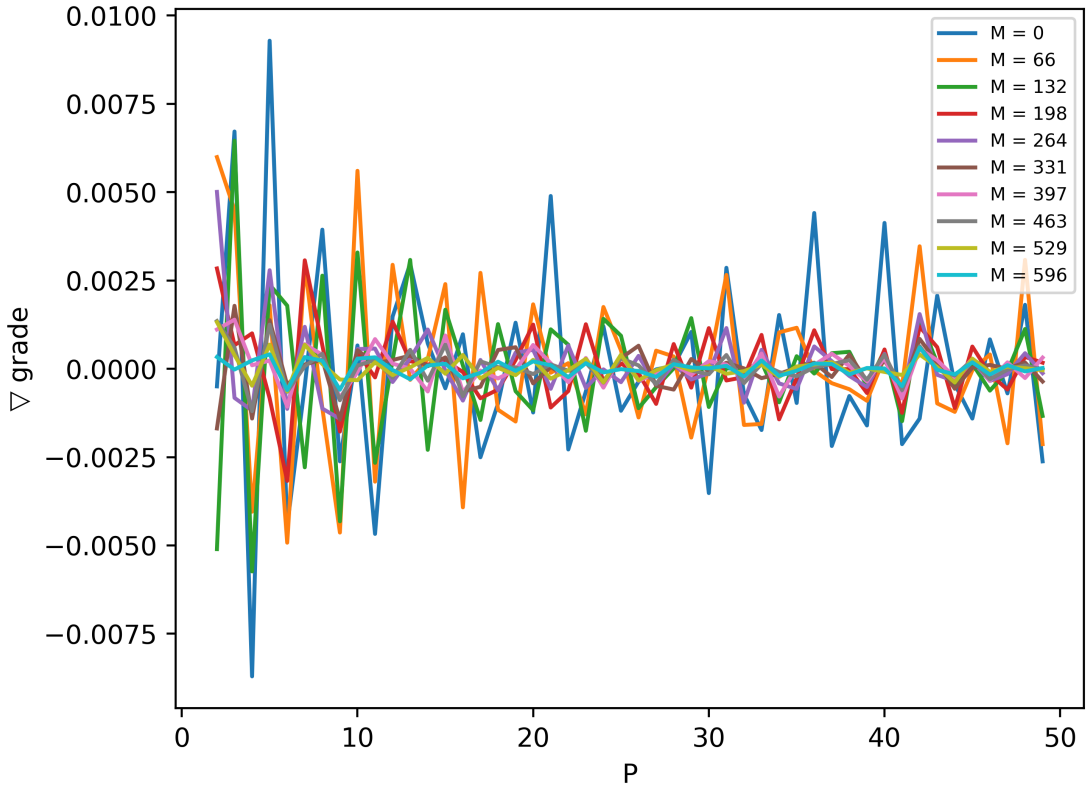
4.1.1 Stratification across dimensions

In Fig.**??** is plotted the same degree function depicted in Fig.7 but considering all dimension P between 2 and 50. Then it was sectioned 10 times at M equally spaced. The projected sections are shown in Fig.**??** as long as their prime derivative Fig.**??**.

The example confirms that even for higher-dimensions sample sets, the mean of stratification at fixed Ms is stationary, meanwhile the variance is effected.

We can conclude that stratification is not effected from P dimensions, but in higher-dimensions the degree is more stable.


1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
grade
0
10
20
30
40
50
P
M = 0
M = 66
M = 132
M = 198
M = 264
M = 331
M = 397
M = 463
M = 529
M = 596


0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
▽ grade
0
10
20
30
40
50
P
M = 0
M = 66
M = 132
M = 198
M = 264
M = 331
M = 397
M = 463
M = 529
M = 596

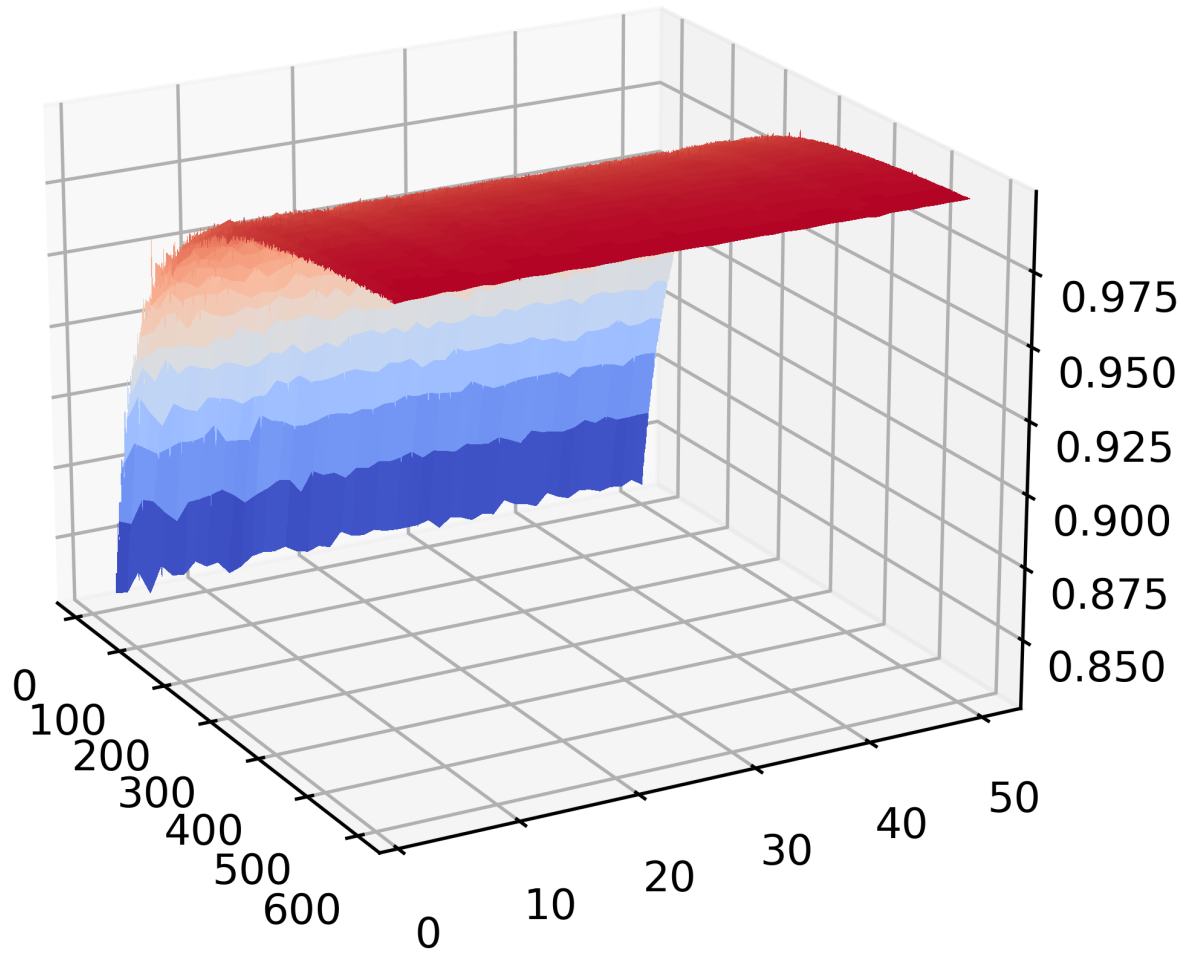
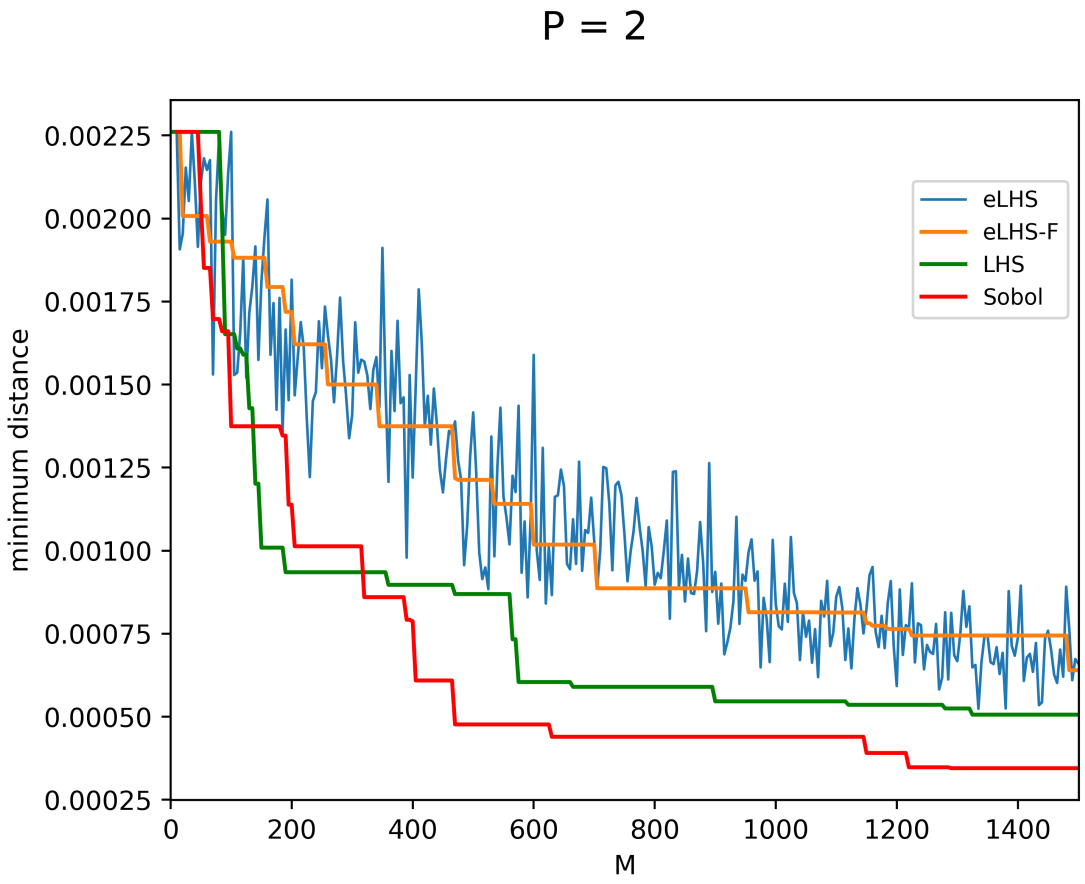


Figure 9: The function $\mathscr{D}$(LHS(N, P), M) with P and M variables and $N = 500$ is shown in (c). 10 equally spaced sections were taken perpendicular to M and parallel to P (a). Prime derivatives for each M section in (b)

4.2 Example II - Properties mutation

The aim of this investigation is to test how much the implemented eLHS with its the criteria embedded - which is maximize the minimum distance between sample points - effects the whole sample set. The example wants to empathize how much good new samples are a key aspect to seriously take into account.

The criteria involved are space-filling maximin distance and L2-star discrepancy (Fig.11). In first place, every sample set involved is optimized to have good discrepancy properties. Then, during the expansion process, both eLHS and eLHS-F bears new samples with space-filling properties, meanwhile both Sobol and LHS are expanded with Monte-Carlo sample points.


P = 2
minimum distance
0.00225
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
0
200
400
600
800
1000
1200
1400
M
eLHS
eLHS-F
LHS
Sobol

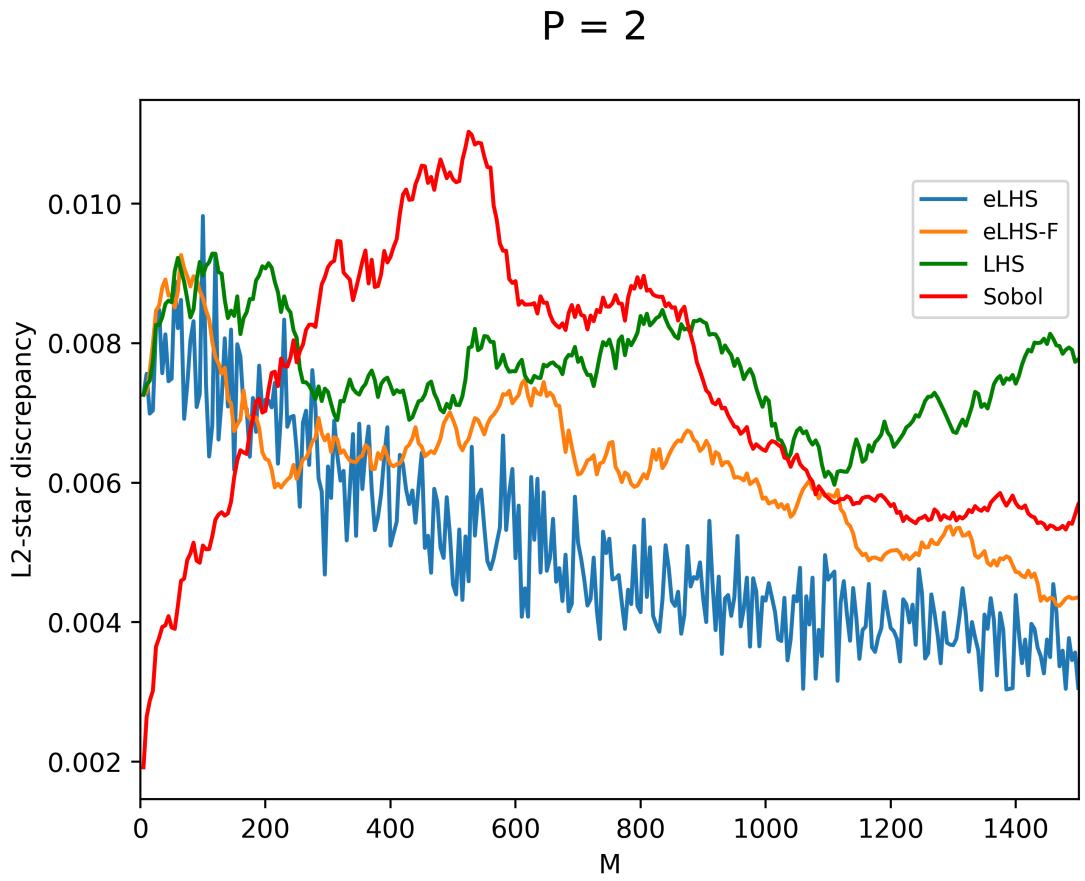

P = 2
L2-star discrepancy
0.010
0.008
0.006
0.004
0.002
0
200
400
600
800
1000
1200
1400
M
eLHS
eLHS-F
LHS
Sobol

Figure 10

There is evidence that both eLHS and eLHS-F outperform the unguided expansion scheme by far regarding maximin distance and minimizing L2-star discrepancy. The characteristic step function is due to the initial sample sets, which have definite minimum distance of fixed points, hence the expanded sets cannot improve it. At least it cannot worsen it. The initial minimum distance represents the upper bound for space-filling optimization search. Despite what observed in what is stressed in section 4.1 about the demoting degree of eLHS-F with respect on the optimal eLHS, the space-filling property does not manifest any relevant disrupting, instead, the minimum distance metric is more stable than eLHS's.

On the other hand, Sobol' and LHS's discrepancy is outperformed as well, even if the eLHS software was not designed to improve that. Discrepancy performance is relevant in the subjcect of sensitivity analysis, topic of section 4.3.

This investigation shows the potential of expansion process on the every properties of a sample set. In the matter of Reason (2), the initial eLHS and eLHS-F have achieved a better discrepancy and space-filling properties than the optimal experimental setups Sobol' and standard LHS enriched with trivial Monte-Carlo sample points.

4.3 Example III - Expansion of Monte Carlo Integration

In the scope of this example, the Monte-Carlo integration adopts a quasi-MC sample distribution to compute numerically the integral of the multivariate function:

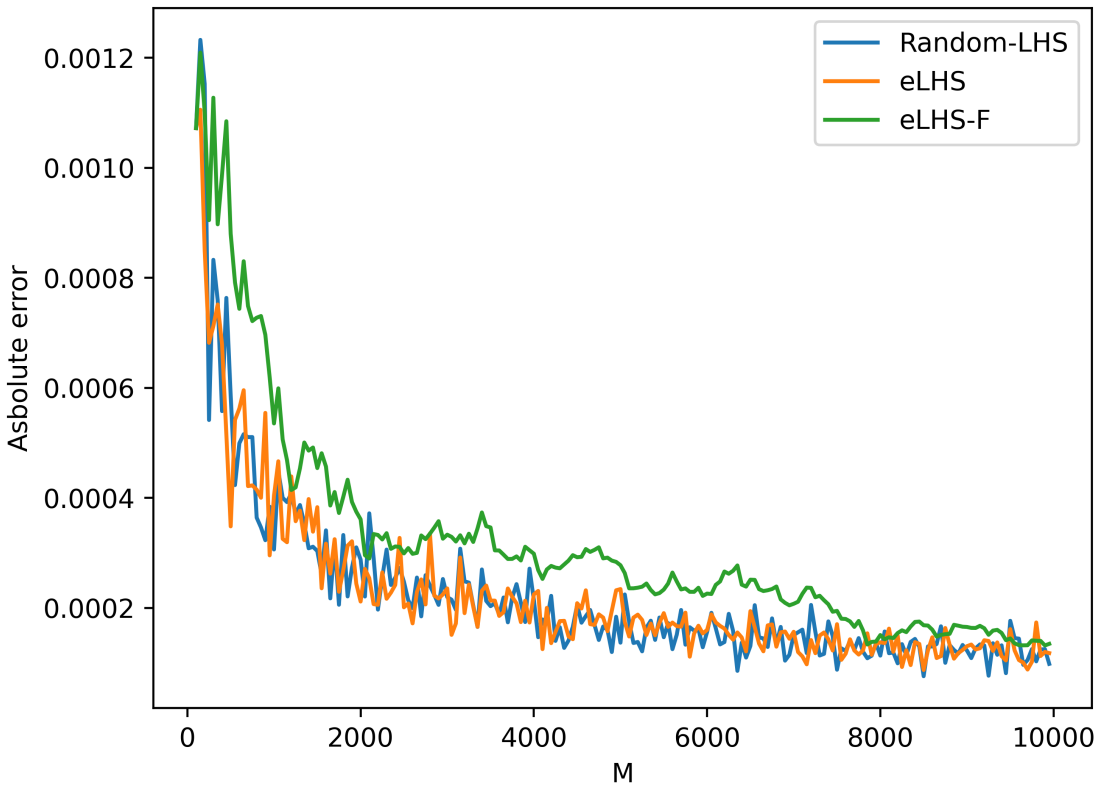
$$f(x,y,z) = e^{-x^2-y^2-z^2} \cdot sin(x) \cdot cos(y) \cdot sin(z)$$

on the integration interval $[0,1]^3$. For technical formulation of this technique and how it is applied to carry out the simulation, please refer to Appendix 7.3 .

The growing step is set to 50, the total number of samples ranges from 100 to 10000. The total number of agents is 10. All experimental setups share the same initial LHS sets.

Fig.**??** depicts absolute error evolution throughout simulation expansions. Instead, Fig.**??** shows the computational time measured at the end of every step. The experimental sets are eLHS-F and eLHS as usual and the Random-LHS distribution, which is a sample set pool pulls out a new sample set at each step. Random-LHS was adopted to simulate what the experimenter should expect if, for instance, the MC integration does not satisfy the maximum error requirement (or any other) and thus they decide to redeploy the whole simulation by creating a new sample set with increased sample size. This kind of expansion pool is in direct contrast with the assumption stated in section 4. That's because the below tests can justify Reason (1).

The high-precision result of the integral which the outcomes have been compared to is yielded by *scipy.integrate.nquad* adaptive multidimensional integration quadrature method. The resultant error is the absolute difference between the deployed MC Integration over the handed sample set and Scipy's result.



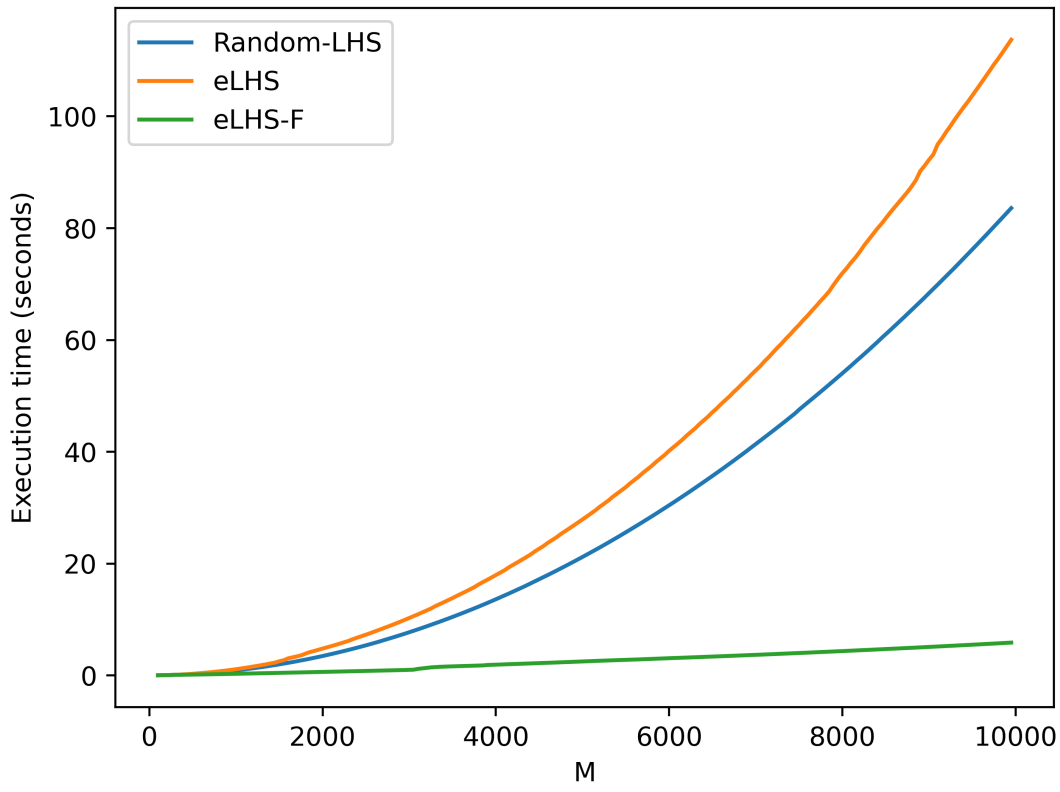


Figure 11: The example was ran on a single unit Macbook Air (2020), CPU/GPU: Silicon M1

The discrepancy in the earlier stages of evolution between eLHS and eLHS-F is justified by the Monte-Carlo integration's high variance predictions. On the other hand, eLHS keeps the error rate comparable to Random-LHS.
By stressing this, we have fulfilled Reason (1) requirements: eLHS and LHS are yet comparable and, in this specific case study, the former outperforms the latter.

eLHS-F close perform the other ones but never exceeding them, the reason is linked to the observation in section 4.1. In the first-stage, the fixed step expansion size (50) is equal to the 50% of the initial sample size (100); on the second-stage, the ratio $\frac{M}{N}$ drops to 33%, then 25%, 20% etc. So, stratification consistency does not hold as well as it should do. Hence, accuracy performance trade-offs with execution time.

On the computational time, the authors achieved a so less execution effort using eLHS-F because the experimental setup is affected by the numerical Monte-Carlo integration next-step rule (Eq.16).

5 Code implementation

In this section, the code implementation of eLHS algorithm and degree function is shown and commented.

The interface language is Python 3.1 and later versions, widely used among scientist and lender of many very advanced mathematics libraries and optimized data structures. While dealing with large bunch of data, it is recommended to develop their own software solution using C/C++ programming languages. By default, Python exports API for porting C language, distributed with Python.h header file. Python APIs are not subject of this research, although it may happen that some code refer to owned Python data structures. The developed C module file for Python has been compiled using clang 15.0.0 (clang-1500.3.9.4) stable release. Furthermore, the flag *-fno-vectorize* for clang compiler has been enable to cast loop vectorization on a data vector with variable width, decided on compile time by a cost model evaluation on computational resources available and cost of memory write outs [5].

Future versions would implement parallelization with pthreads to optimize large data structure CRUD operations and concurrent search trees. We do also have assessed that the algorithm could be reorganized to better match Superword-Level-Parallelism (SPL) prerequisites which is incompatible with loop vectorization by LLVM project.

The full project code is publicly available online and ruled by MIT License on my personal GitHub respository: `github.com/alecrespi/PLHS` . The package will be distributed with Python package manager *pip* under the name of *latinexpansion*.

5.1 eLHS - Expansion Algorithm

The software was implemented accordingly with guidelines in section 3.3.3.

The Python function signature is:

```
def eLHS(S:numpy.ndarray, M:int, throws:int = 10) -> numpy.ndarray
```

The *vacancies_list* struct type has been used to build the irregular set of tuples referred as V throughout this paper. It tracks down how many vacancies has been found in a specific dimension. A tuple (array) of integers indexes of the same size is allocated. Definition:

```
typedef struct vlist{
    int len;
    int* vacancies;
} vacancies_list;
```

To enhance code readability, some pre-processor macros have been defined as well:

```
// given q interval index, get interval lower bound
#define I(q) (((double)q)/(N+M))

// given x sample, get its interval index
#define Q(x) (floor(((double)x)*(N+M)))

// var swapping, do-while block triggers pre-processor
// multi-line rescripting
#define SWAP(a, b) do { typeof(a) tmp = a; a = b; b = tmp; }
   while (0)

// given q interval index, generate a random sample in it
#define R(q) ((double)I(q) + ((double)rand()*I(1) / RAND_MAX))
```

The variable identifiers are

N, M, P $\rightarrow$ number of samples in ss, number of additional points, number of dimensions. All strictly positive.

ss $\rightarrow$ shorthand for *sample set*, $N \times P$ matrix of double;

exp $\rightarrow$ shorthand for *expansion set*, $M \times P$ matrix of double;

vs $\rightarrow$ shorthand for *vacancies set*, P-long array of *vacancies_list* structs;

v_prime $\rightarrow$ the V' vacancies matrix, $M \times P$ matrix of integers;

The C core body module of eLHS is given below:

```c
static PyObject* method_eLHS (...) {
  // variable declaration block
    double max_distance = 0.0;
  // interface with Python.h API
  ...
  // getting the whole vacancies
    vs = build_vacancies_matrix(ss, N, P, M);

  /* try 'throws' times to keep maximin distance optimal */
    for(t = 0; t < throws; t++){
        /* shuffle and subset vacancies set at each iteration
   */
        v_prime = shuffle_subset_vacancies(vs, M, P);

        // sowing new points
        for(i = 0; i < M; i++)
            for(j = 0; j < P; j++)
                exp[i][j] = R(v_prime[j][i]);

      // maximin distance search
        if(throws != 1){
            dist = min_distance(ss, exp, N, M, P);
            if(dist > max_distance)
                max_distance = dist;
        }
    }
    ...
  // returning exp PyObject* data struct as eLHS result
}
```

The core module in C is then subdivided in the following 4 macro routines.

5.1.1 build_vacancies_matrix

The first step of the algorithm is to track down every vacant interval in every dimension. Thus,

```c
vacancies_list* build_vacancies_matrix
  (double** ss, int N, int P, int M)
{
    vacancies_list *vs;
    int i, j, q;

    vs = (vacancies_list*)malloc(P*sizeof(vacancies_list));

    for(j = 0; j < P; j++){
        int vindex = 0;
        int *vptr = (int*) malloc((N+M)*sizeof(int));
```

```
        // masks vptr elements if they are void
        bool *vmask = (bool*) malloc((N+M)*sizeof(bool));
        // set mask to false
        memset(vmask, 0, (N + M) * sizeof(bool));

        // map down the busy intervals (busy is TRUE)
        for(i = 0; i < N; i++)
            vmask[(int) Q(ss[i][j])] = true;

        // mask the interval indexes to gather only voids
        for(q = 0; q < N+M; q++){
            // if q is a void, then add it
            if(!vmask[q])
                vptr[vindex++] = q;
        }

        // realloc voids sequence to fit void numbers
        vptr = (int*) realloc(vptr, vindex * sizeof(int));
        if(!vptr)   return NULL;
        // build up vacancies_list entry
        vs[j].len = vindex;
        vs[j].vacancies = vptr;
    }
    return vs;
}
```

5.1.2 shuffle_subset_vacancies

Shuffle the vacancies set *vs* and apply the σ criterion chosen. The implementation permutes randomly each vacancy_list array attribute and then cut off the exceeding intervals.

```
int **shuffle_subset_vacancies
  (vacancies_list* vs, int M, int P)
{
    if(!vs || M == 0) return NULL;
    int **v_prime = (int **)malloc(P * sizeof(int*));

    /* for each dimension, shuffle and reshape the vacancy list
    to a regular size M */
    for(int j = 0; j < P; j++){
        if(!vs[j].vacancies || !vs[j].len)
            return NULL;

        int *vptr = (int*)malloc(vs[j].len * sizeof(int));
        /* copy elements to preserve data integrity for further
    steps*/
```

```
        for(int i = 0; i < vs[j].len; i++)
            vptr[i] = vs[j].vacancies[i];

        /* shuffling all items */
        for (int i = vs[j].len; i > 0; i--){
            // generate a random index that ranges from [0, M
   -1]
            int r = rand() % i;
            SWAP(vptr[i-1], vptr[r]);
        }
        /* truncate the list to M */
        vptr = (int *)realloc(vptr, M*sizeof(int));
        v_prime[j] = vptr;
    }
    return v_prime;
}
```

5.1.3 sample_sowing

Based on V' shuffled matrix, generate a brand new expansion set and store it in *exp* matrix. This routine is hard coded in the eLHS module body.

```
// in the declaration block
// allocate exp result matrix
exp = (double **)malloc(M * sizeof(double*));
for(i = 0; i < M; i++)
  exp[i] = (double *)malloc(P * sizeof(double));
...
//sample_sowing
/* for each new sample */
for(i = 0; i < M; i++)
  /* for each dimension */
  for(j = 0; j < P; j++)
    /* generate sample in the random interval v_prime[j][i] */
    exp[i][j] = R(v_prime[j][i]);
...
```

5.1.4 minimax_search

```
// in the declaration block
double max_distance = 0.0, dist;
...
while (condition){
  ...
```

```
  dist = min_distance(ss, exp, N, M, P);
  if(dist > max_distance)
    max_distance = dist;
  ...
}

// somewhere else in function initialization section
/* computes minimum distance among every points, only expansion
    points are considered beacuse initial points have fixed
   minimum distance */
double min_distance
(double **ss, double **exp, int N, int M, int P)
{
    int i, k;
    /* __DBL_MAX__ will force the first minimum if-condition */
    double min = __DBL_MAX__, dist = 0.0;

    for(k = 0; k < M; k++){
      /* search minimum distance among expansion points and
   initial points */
        for(i = 0; i < N; i++){
          // P-degree euclidean distance as metric
            dist = euclid(exp[k], ss[i], P);
            if( dist < min ) min = dist;
        }
      /* search minimum distance among expansion points
   themselves */
        for(i = k + 1; i < M; i++){
            dist = euclid(exp[k], exp[i], P);
            if( dist < min ) min = dist;
        }
    }
    return min;
}
```

5.2 Degree function - Stratification heuristic

The degree heuristic software formally implements Eq.7 definition. It was deployed to carry out section 4.1 Example I simulations.

The Python signature for this method is:

```
def degree(S:numpy.ndarray, M:int = 0) -> float
```

The function's body follows:

```
...
```

```
double gr = 0;
// compute degree
for(j = 0; j < P; j++){
    for(q = 0; q < N+M; q++){
        for(i = 0; i < N; i++){
          /* check if sample's projection lies in q interval */
            if(I(q) <= ss[i][j] && ss[i][j] < I(q+1)){
                gr += 1;
                /* if found, break the samples' loop (simulates
    'min' operand of the formula */
                break;
            }
        }
    }
}
gr /= (double) P * (N + M);
...
// return PyObject* representing 'gr' result
```

6 *Conclusions*

The expansion process has demonstrated high capabilities of enhancing an ongoing simulation thanks to the evolutionary analysis that the authors depict. The number of additional points are a crucial variable to ponder: too low Ms annihilate the original LHS stratification property across the space; instead, higher Ms, roughly above N - so by doubling the sample size - the accuracy increases and stratification is almost kept but may fail on the computational limits requested.

Expansion may be ruled by a optimization criteria which can differ from the ones which the initial sample set was built on. This heterogeneous mixture of properties can be crucial in solving complex simulation dynamically, by improving a different criteria on the need. Example II (section 4.2) experimental setups exhibit this flexibility.

The research has also proposed the degree and expanded degree heuristics, along with their properties, which demonstrated to be useful tool for quantify stratification in any quasi-Monte-Carlo sampling.

Referring to the innovative work of [18] about *Progressive Latin Hypercube Sampling* - which is a theoretical class of sampling methods with adaptive sample size based on intrinsic properties of the sample set - the multistage evolutionary approach that this paper discuss can represent the very fundamental computational aspect of enlarging (or shrinking) the sample set size on the need.

This research allowed the authors to develop and release the first official latinexpansion package for Python API (**crespi2024**) that include *expanded degree* stratification metric and *eLHS* expansion algorithm for any sample set.

Research might turn toward the analysis of the probabilistic distribution of the degree stratification index, explaining the optimal stratification spikes observed in Example I (section 4.1), and looks for an optimal pool of Latin Hypercube sample sets which are best to expand.

7 APPENDIX

7.1 Indicator function

The indicator function ***1*** of a set **A** indicates whether the input belongs to A or not, specifically:

$$\mathbf{1}_A(x) := \begin{cases} 1 & if\, x \in A \\ 0 & if\, x \notin A \end{cases} \tag{13}$$

As in the matter of sectioning a space into continuous intervals in the shape of [a, b), it is useful to redefine the indicator function as an operation that occurs with the boundaries of A using the Heaviside step function which does not involve set operators but only logical ones. It's important to remark that it does not matter what happens precisely on the boundaries. The Heaviside function is defined:

$$H(x) := \begin{cases} 1 & if\, x \geq 0 \\ 0 & if\, x < 0 \end{cases} \tag{14}$$

So the indicator function can be also produced:

$$\mathbf{1}_{[a,b)}(x) = H(x-a) \cdot H(b-x) \tag{15}$$

7.2 Known perfect expansion explained: Multiples of N

Given $S \in LHS(N,P)$ set with $N,P \in \mathbb{N}$ respectively number of samples and dimensions, by definition (Eq.5), has maximum $\mathscr{D}$ degree. Experimentally, it has been observed that the only well-known perfect expansions (maximum expanded degree Eq.7) of S are those which the M increment of the sample space is a multiple of N (*M. Falcone (2023)*[10]). Experimental evidences are depicted in Fig.7.

The reason of this behavior is linked to the unique distribution characteristic that intervals the $M = K \cdot N$ growing grid and the initial S' grid share: the former, actually, sections - with abuse of notation - *perfectly* the latter's intervals space. (*) By considering the same amount of samples N for both situations, we can state that the sum of the shares along any dimension of the regridded set is equal to the initial's one, which is equal to N.
Let's stress out the degree and expanded degree equations:

$$\mathscr{D}(S) = \frac{1}{P \cdot N} \cdot \sum_{j=1}^{P} \sum_{q=1}^{N} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{N}, \frac{q}{N})}(S_{ij}), 1) = 1$$

the total share of all intervals along a fixed dimension is:

$$\sum_{q=1}^{N} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{N}, \frac{q}{N})}(S_{ij}), 1) = N$$

Given the unique property cited above (*), also given $N + M = (K+1) \cdot N$, the expanded grid should have the exact same amount of share:

$$\sum_{q=1}^{(K+1)\cdot N} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{(K+1)\cdot N}, \frac{q}{(K+1)\cdot N})}(S_{ij}), 1) = N$$

then, the expanded degree should be:

$$\mathscr{D}(S, K \cdot N) = \frac{1}{P \cdot (K+1) \cdot N} \cdot \sum_{j=1}^{P} \sum_{q=1}^{(K+1)\cdot N} min(\sum_{i=1}^{N} \mathbf{1}_{[\frac{q-1}{(K+1)\cdot N}, \frac{q}{(K+1)\cdot N})}(S_{ij}), 1) = \frac{1}{K+1}$$

which corresponds exactly to the upper limit of an $M = K \cdots N$ growing step grid for an S LHS set Eq.9:

$$\mathscr{D}(S, K \cdot N) = \frac{1}{K+1} = 1 - \frac{K \cdot N}{(K+1) \cdot N} = \mathscr{D}_{max}$$

Hence, any S expansion of $K \cdot N$ magnitude has maximum expanded degree, which makes them *perfect expansions* by definition.

7.3 Monte-Carlo Integration

Monte-Carlo Integration is a powerful numerical integration technique widely adopted in engineering and physics. It is deployed in high-dimensional spaces where traditional methods become too much computational demanding and the trade-off between computational time and estimating gain become infeasible. Basically, this is a stochastic method where N points are randomly sampled in a parameter space, a so-called Monte-Carlo Sampling.

Given a $f(\bar{x})$ of P parameters, the integral:

$$I = \int \Omega f(\bar{x}) d\bar{x}$$

is approximated by the Monte-Carlo integral:

$$I \simeq \frac{V}{N} \cdot \sum_{i}^{N} f(\bar{x}_i)$$

If two quasi-MC sample sets are involved, let's say X and Y of N_1 and N_2 sample size respectively, the integral take the form:

$$I = \frac{V}{N_1 + N_2} \cdot \sum_{i}^{N_1+N_2} f(\bar{x}_i)$$

If I_1 integral computed for X is known, then to compute the estimated I is only necessary to compute for Y and combine both using:

$$I = \frac{N_1}{N_1 + N_2} \cdot I_1 + \frac{V}{N_1 + N_2} \sum_{i=N_1}^{N_2} f(\bar{x}_i) \tag{16}$$

7.4 Low-Discrepancy sequences

Low-discrepancy sequences are a class of quasi-MC sampling methods which adopt a discrepancy minimization criteria, a measure of the deviation of the distribution of sample points from uniformity. One commonly used metric for evaluating this uniformity is the L2-star discrepancy, which quantifies how well the sample points cover the integration domain.

Sobol' sequences are a type of low-discrepancy sequence, developed by *Sobol' (1967)*[20], that provides a highly uniform distribution of points in multi-dimensional spaces. They are particularly effective in numerical integration and sensitivity analysis. The construction of Sobol' sequences involves the use of direction numbers and primitive polynomials to achieve a uniform spread of points [3].

References

[1] Charles J. Colbourn. *Handbook of Combinatorial Designs*. 2nd. CRC Press, 2006.

[2] Constantinos Daskalakis et al. *Sorting and Selection in Posets*. 2007. arXiv: 0707.1532 [cs.DS]. URL: https://arxiv.org/abs/0707.1532.

[3] Josef Dick and Friedrich Pillichshammer. *Digital nets and sequences: discrepancy theory and quasi–Monte Carlo integration*. Cambridge University Press, 2010, pp. 263–267.

[4] Leonhard Euler. *On magic squares*. 2005. arXiv: math/0408230 [math.CO]. URL: https://arxiv.org/abs/math/0408230.

[5] University of Illinois. *Auto-Vectorization in LLVM — LLVM 19.0.0git documentation*. URL: https://llvm.org/docs/Vectorizers.html.

[6] Tobias Jacobs et al. "On the Complexity of Searching in Trees: Average-Case Minimization". In: *Automata, Languages and Programming*. Ed. by Samson Abramsky et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 527–539. ISBN: 978-3-642-14165-2.

[7] Donald E. Knuth and Ronald W. Moore. "An analysis of alpha-beta pruning". In: *Artificial Intelligence* 6.4 (1975), pp. 293–326. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(75)90019-3. URL: https://www.sciencedirect.com/science/article/pii/0004370275900193.

[8] Patrick Koch et al. "Autotune: A Derivative-free Optimization Framework for Hyperparameter Tuning". In: KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 443–452. ISBN: 9781450355520. DOI: 10.1145/3219819.3219837. URL: https://doi.org/10.1145/3219819.3219837.

[9] Xiangshun Kong, Mingyao Ai, and Kwok Leung Tsui. "Design for Sequential Follow-Up Experiments in Computer Emulations". In: *Technometrics* 60.1 (Apr. 2017), pp. 61–69. DOI: 10.1080/00401706.2016.1258010. URL: https://doi.org/10.1080/00401706.2016.1258010.

[10] M. Boschini M. Falcone D. Gerosa. *Progressive Latin hypercube sampling for simulation design*. 2023.

[11] N. Metropolis. "The beginning of the Monte Carlo Method". In: *Los Alamos Science* Special Issue (1987).

[12] Matthew Mould, Davide Gerosa, and Stephen R. Taylor. "Deep learning and Bayesian inference of gravitational-wave populations: Hierarchical black-hole mergers". In: *Physical Review D* 106.10 (Nov. 2022). ISSN: 2470-0029. DOI: 10.1103/physrevd.106.103013. URL: http://dx.doi.org/10.1103/PhysRevD.106.103013.

[13] A. Olsson, G. Sandberg, and O. Dahlblom. “On Latin hypercube sampling for structural reliability analysis”. In: *Structural Safety* 25.1 (2003), pp. 47–68. ISSN: 0167-4730. DOI: https://doi.org/10.1016/S0167-4730(02)00039-5. URL: https://www.sciencedirect.com/science/article/pii/S0167473002000395.

[14] Art B. Owen. “Scrambling Sobol’ and Niederreiter–Xing Points”. In: *Journal of Complexity* 14.4 (1998), pp. 466–489. ISSN: 0885-064X. DOI: https://doi.org/10.1006/jcom.1998.0487. URL: https://www.sciencedirect.com/science/article/pii/S0885064X98904873.

[15] L. A. Stone R. W. Kennard. “Computer Aided Design of Experiments”. In: *Technometrics* 11.1 (1969).

[16] Marissa Renardy et al. “To Sobol or not to Sobol? The effects of sampling schemes in systems biology applications”. In: *Mathematical Biosciences* 337 (2021), p. 108593. ISSN: 0025-5564. DOI: https://doi.org/10.1016/j.mbs.2021.108593. URL: https://www.sciencedirect.com/science/article/pii/S0025556421000419.

[17] Boyang Shang and Daniel W. Apley. “Fully-sequential space-filling design algorithms for computer experiments”. In: *Journal of Quality Technology* 53.2 (2021), pp. 173–196. DOI: 10.1080/00224065.2019.1705207.

[18] Razi Sheikholeslami and Saman Razavi. “Progressive Latin Hypercube Sampling: An efficient approach for robust sampling-based analysis of environmental models”. In: *Environmental Modelling & Software* 93 (2017), pp. 109–126. ISSN: 1364-8152. DOI: https://doi.org/10.1016/j.envsoft.2017.03.010. URL: https://www.sciencedirect.com/science/article/pii/S1364815216305096.

[19] Ilya M. Sobol et al. “Construction and Comparison of High-Dimensional Sobol’ Generators”. In: *Wilmott* 2011.56 (Nov. 2011), pp. 64–79. DOI: 10.1002/wilm.10056. URL: https://doi.org/10.1002/wilm.10056.

[20] I.M Sobol’. “On the distribution of points in a cube and the approximate evaluation of integrals”. In: *USSR Computational Mathematics and Mathematical Physics* 7.4 (1967), pp. 86–112. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(67)90144-9. URL: https://www.sciencedirect.com/science/article/pii/0041555367901449.

[21] Chenxiao Song and Reiichiro Kawai. “Monte Carlo and variance reduction methods for structural reliability analysis: A comprehensive review”. In: *Probabilistic Engineering Mechanics* 73 (2023), p. 103479. ISSN: 0266-8920. DOI: https://doi.org/10.1016/j.probengmech.2023.103479. URL: https://www.sciencedirect.com/science/article/pii/S0266892023000681.

[22] G. W. Stewart. *Afternotes on Numerical Analysis*. SIAM, Jan. 1996, pp. 151–155. URL: http://books.google.ie/books?id=VEHBOUOAL-EC&printsec=frontcover&dq=Stewart,+Gilbert+W.+(1996).+Afternotes+on+Numerical+Analysis&hl=&cd=1&source=gbs_api.

[23] Stephen R. Taylor and Davide Gerosa. "Mining gravitational-wave catalogs to understand binary stellar evolution: A new hierarchical Bayesian framework". In: *Physical Review D* 98.8 (Oct. 2018). ISSN: 2470-0029. DOI: 10.1103/physrevd.98.083017. URL: http://dx.doi.org/10.1103/PhysRevD.98.083017.

[24] Pauli Virtanen et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17.3 (Feb. 2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2. URL: https://doi.org/10.1038/s41592-019-0686-2.