SoftwareX 31 (2025) 102294

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

"LHS in LHS": A new expansion strategy for Latin hypercube sampling in simulation design

Matteo Boschini [a,b,*], Davide Gerosa [a,b], Alessandro Crespi [c], Matteo Falcone [a]

[a] Dipartimento di Fisica "G. Occhialini", Universitá degli Studi di Milano-Bicocca, Piazza della Scienza 3, Milano, 20126, Italy
[b] INFN, sezione di Milano-Bicocca, Piazza della Scienza 3, Milano, 20126, Italy
[c] Dipartimento di Informatica, Sistemistica e Comunicazione, Universitá degli Studi di Milano-Bicocca, Viale Sarca 336, Milano, 20126, Italy

ARTICLE INFO

Keywords:
Simulation design
Latin hypercube sampling
Space-filling

ABSTRACT

Latin Hypercube Sampling (LHS) is a prominent tool in simulation design, with a variety of applications in high-dimensional and computationally expensive problems. LHS allows for various optimization strategies, most notably to ensure space-filling properties. However, LHS is a single-stage algorithm that requires a priori knowledge of the targeted sample size. In this work, we present "LHS in LHS," a new expansion algorithm for LHS that enables the addition of new samples to an existing LHS-distributed set while (approximately) preserving its properties. In summary, the algorithm identifies regions of the parameter space that are far from the initial set, draws a new LHS within those regions, and then merges it with the original samples. As a by-product, we introduce a new metric, the LHS degree, which quantifies the deviation of a given design from an LHS distribution. Our public implementation is distributed via the Python package EXPANDLHS.

Code metadata

Current code version	v1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00198
Legal Code License	MIT
Code versioning system used	git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	numpy, scipy, numba (>0.57)
If available Link to developer documentation/manual	https://m-boschini.github.io/expandLHS
Support email for questions	m.boschini1@campus.unimib.it

1. Motivation and significance

The continuous growth of available data is a common trend across disciplines, often enabling the study of increasingly complex phenomena. A useful strategy to address these challenges is to perform simulated experiments, where one analyzes the outcomes of a model and/or the predictions of a theory while controlling the effects of input parameters. Simulations are widely used in various fields, ranging from mathematics to physics, medicine, economics, and education [1]. More often than not, simulating complex phenomena carries a high computational cost, making it critical to determine where to place simulations in the parameter space.

To this end, simulation design is a branch of statistics that aims to optimize computational experiments [2,3]. Simulation design assists in developing algorithmic strategies that explore the parameter space effectively, ensuring an optimal distribution of samples (where optimality needs to be appropriately defined), while mitigating computational costs. A set of samples, where simulations will be placed, can therefore be preferred over another by enforcing specific properties such as space-filling, one-dimensional projection, and low correlation. There is a vast literature on simulation design, with a variety of implementations and model-independent algorithms based on different notions of optimality [2]. In this paper, we focus on one such approach: Latin hypercube sampling (LHS) [4].

* Corresponding author at: Dipartimento di Fisica "G. Occhialini", Universitá degli Studi di Milano-Bicocca, Piazza della Scienza 3, Milano, 20126, Italy.
E-mail address: m.boschini1@campus.unimib.it (M. Boschini).

https://doi.org/10.1016/j.softx.2025.102294
Received 26 March 2025; Received in revised form 8 July 2025; Accepted 28 July 2025
Available online 14 August 2025

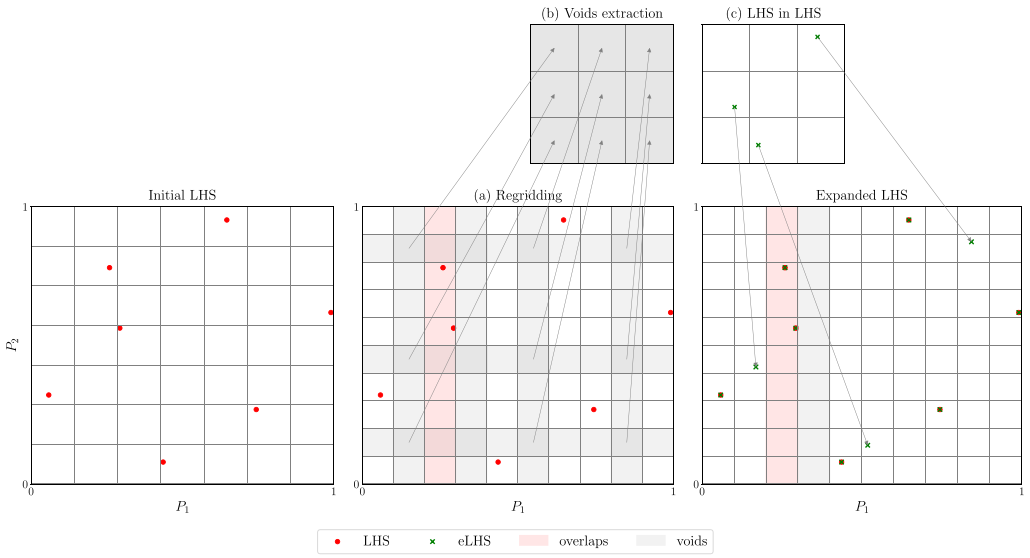

Fig. 1. The "LHS in LHS" expansion algorithm. The left panel shows an initial LHS with $N = 7$ points in $P = 2$ dimensions. This LHS is here expanded with $M = 3$ new points. In step (a), we regrid the original space with $M + N$ bins in each dimension. This creates at least M empty intervals in both dimensions (shaded in gray). Along dimension P_1, two samples overlap in the same bin (shaded in red), and consequently, there is a fourth empty interval. In step (b), $M = 3$ empty intervals are randomly selected along both dimensions. In step (c), a new set is drawn in this subspace via LHS. Finally, in the right panel, the new samples are added to the original ones to obtain the targeted expansion eLHS$(P, M + N)$. The expansion shown in this figure has a degree $\mathcal{D} = 0.95 < 1$ and is thus not a perfect Latin hypercube.

LHS was first introduced in the 1970s [5,6] and was further developed to optimize projection properties, improve space-filling design, and avoid spurious correlations [7–14]. The concept of Latin square comes from combinatorial mathematics — an $N \times N$ square with N different symbols appearing only once per column and row. A Latin hypercube generalizes this property to a P-dimensional hypercube, where each dimension is binned into N disjoint intervals $[i/N, (i+1)/N)$ where $i = 0, 1, \dots, N-1$ with marginal probability $1/N$. LHS distributes points to tile the space while preserving the one-dimensional projection property—i.e., ensuring that each marginalized one-dimensional bin contains exactly one sample. Multiple designs with this property are possible for a given N, and one can further optimize the final result by imposing additional criteria, such as orthogonality and distance optimization [4]. However, these are typically *single-stage* sampling algorithms, requiring all samples to be drawn simultaneously or, at most, in a finite number of discrete steps. This implies that the number of targeted samples must be known *a priori*, and once the set is drawn, additional samples generally cannot be added while preserving the desired projection property.

This is a crucial limitation in many practical applications. In most realistic scenarios, simulation design does not happen all at once but rather gradually and iteratively, with subsequent batches that need to be properly initialized. Suppose one initially plans for N simulations based on the available computational budget and distributes them using LHS. Later, realizing that this is insufficient for the targeted application, one might secure additional computing time for M more simulations. How should these M additional simulations be integrated into the initial LHS-distributed set of N simulations so that the combined set of $N + M$ samples still retains desirable space-filling properties?

In this paper, we propose a new model-free expansion algorithm for LHS, accompanied by an open-source package for the Python programming language, which we dubbed expandLHS. Compared to other proposed LHS expansion strategies [15–20], our approach allocates new samples by rebinning the original hypercube to preserve the projection property. Building on this idea, we can further leverage the versatility of Latin hypercubes by essentially designing an LHS within the existing LHS. As a by-product, this paper introduces the notion of "LHS degree" to quantify how closely a given set of samples resembles an LHS with the same number of points.

2. Software description

2.1. Algorithm

Let us denote by LHS(P, N) an initial set of N points in P dimensions, defined within the hypercube $[0, 1)^P$ and satisfying the one-dimensional projection property of LHS. That is, there is one and only one sample per interval if each dimension is uniformly binned into N disjoint intervals. We wish to distribute M new samples to obtain an extended set eLHS$(P, N + M)$.

The algorithm we propose consists of three main steps, as illustrated in Fig. 1:

(a) *Regridding*. Each dimension is binned into $N + M$ equal-width, disjoint intervals. The initial samples LHS(P, N) now fall onto a different grid, which creates $Q \geq M$ empty bins and may lead to overlaps of samples in other bins.
(b) *Void extraction*. We consider all empty intervals and select M of them.
(c) *"LHS in LHS"*: In this M-dimensional subgrid, we distribute M samples using, once more, an LHS strategy. These are added to the initial LHS to obtain the targeted extended set eLHS$(P, N + M)$.

The probability of obtaining multiple samples in the same interval after regridding depends on both the initial set LHS(P, N) and the value of M. When this happens, the final extended set eLHS$(P, N + M)$ partially loses the one-projection property. In fact, if $Q > M$ in a certain dimension, there will necessarily be $Q - M$ bins with multiple points and, consequently, empty bins after the expansion.

To quantify this loss, we introduce a new metric $\mathcal{D}$, which we refer to as the "LHS degree" of a sample set. Consider a generic sample set S with N elements in P dimensions; this is composed of real numbers S_{ij} with $i = 0, \dots, N-1$ and $j = 0, \dots, P-1$. We define

$$\mathcal{D}(S) = \frac{1}{N\,P} \sum_{j=0}^{P-1} \sum_{l=0}^{N-1} \min \left[\sum_{i=0}^{N-1} I_{\left[\frac{l}{N}, \frac{l+1}{N}\right)}(S_{ij}), 1 \right], \tag{1}$$

where I is the indicator function defined as

$$I_{[a,b)}(x) = \begin{cases} 1 & x \in [a, b) \\ 0 & x \notin [a, b) \end{cases}.$$

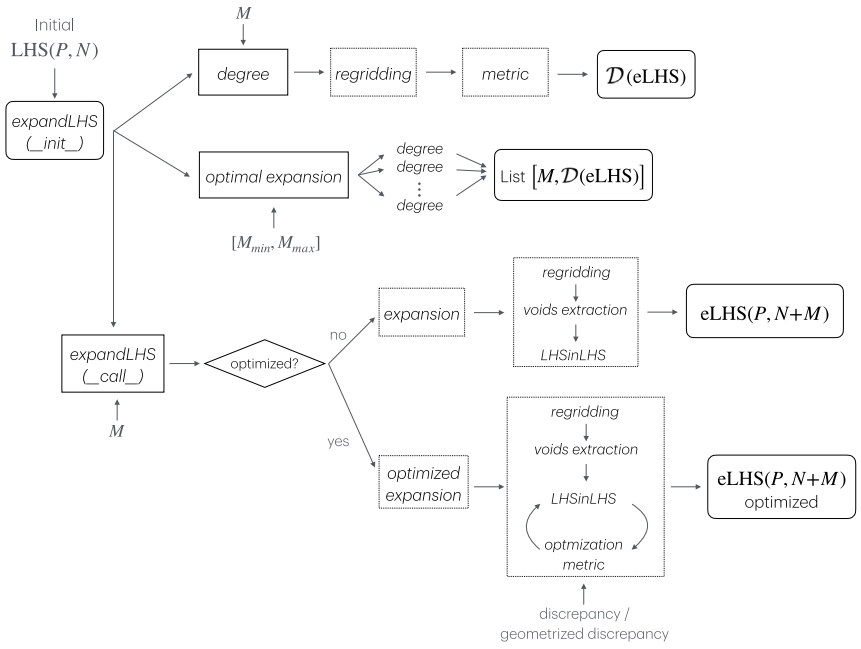

Fig. 2. Schematic representation of the expandLHS package. Functionalities are implemented in the `ExpandLHS` class. Users have access to three main methods (solid black rectangles): `degree`, `optimal_expansion`, and the `ExpandLHS()` call. These compute the degree D of a putative expansion, estimate the optimal expansion size, and perform the actual expansion, respectively. For additional details see Section 2.2 and the code documentation (m-boschini.github.io/expandLH).

In particular, one has $0 < D(S) \leq 1$ and

$$D(S) = 1 \iff S = \mathrm{LHS}(P, N), \tag{2}$$

The degree D can thus be interpreted as the fractional closeness to an LHS set. Essentially, this procedure checks all the i-intervals for each dimension j. If they are populated by a sample S_{ij}, a weight $1/(NP)$ is assigned; otherwise, the weight is 0. The metric is the sum of all the weights.

After the expansion, one has $D(\mathrm{eLHS}) \leq 1$. Indeed, the sums in Eq. (1) trace the presence of at least one sample per interval. An interval containing overlapping samples still contributes a weight of 1 to the weighted sum described above, due to the use of the min function. For each such interval, there exists a corresponding empty interval in the same dimension that receives a weight of zero. Consequently, the degree $D(\mathrm{eLHS})$ is reduced. Fig. 1 shows an example with $N = 7$, $M = 3$, and $P = 2$. In this case, the final degree is $D = 0.95$. A "perfect" expansion with $D(\mathrm{eLHS}) = 1$ corresponds to the case where the new $N+M$ set of points is also LHS-distributed. In general, this is not always possible.

The selection of M empty intervals is not unique whenever $Q < M$. Among the possible choices, we select the one that maximizes either the centered discrepancy or the geometric discrepancy [2,21]. The former measures the uniformity of the sample set as a proxy for its space-filling properties, while the latter tracks the minimum Euclidean distance between samples. In practice, we generate multiple expansions and select the one that minimizes (maximizes) the discrepancy (geometric discrepancy), up to a certain tolerance level. This becomes more relevant in a larger number of dimensions P and as the number of new samples M approaches the initial sample size N.

2.2. Implementation

Our software, expandLHS, is implemented in Python 3 and leverages Numpy's arrays and Numba's just-in-time compilation for fast computations. expandLHS is distributed via the Python Package Index and can be installed via:

```
pip install expandLHS
```

Dependencies include Numpy [22], Scipy [23], and Numba (version $\geq$ 0.57.0) [24]. If not present, these libraries will be installed/updated along with the package.

The expandLHS module is implemented as a single Python class. This is imported within a Python console, script, or Jupyter Notebook using e.g.

```
from expandLHS import ExpandLHS
```

Our software is in its v1.1 release. The source code is distributed under the git version control system and the MIT permissive license at github.com/m-boschini/expandLHS. See Fig. 2 for a flowchart describing the overall implementation.

The expandLHS class has additional functionalities implemented. The method `ExpandLHS.degree` computes the degree D defined in Eq. (1). The method `ExpandLHS.optimal_expansion` iterates over possible values of M to identify the expansion strategy that maximizes the degree of the resulting eLHS. Finally, the inner LHS in step (c) of the algorithm above can be optimized to achieve a user-defined threshold in the chosen metric. At present, our package implements two different metrics from the `scipy.stats.qmc` submodule: `discrepancy` and `geometric_discrepancy`.

3. Illustrative examples

3.1. LHS expansion

The code snippet below introduces the basic usage of expandLHS. We extend an initial LHS set with $N = 20$ points in $P = 2$ dimensions, adding $M = 18$ new samples.

```
from scipy.stats.qmc import LatinHypercube
from expandLHS import ExpandLHS

# initial LHS set
P = 2 # hypercube dimension
N = 20 # initial sample size
lhs_sampler =  LatinHypercube(P)
lhs_set = lhs_sampler.random(N)
```

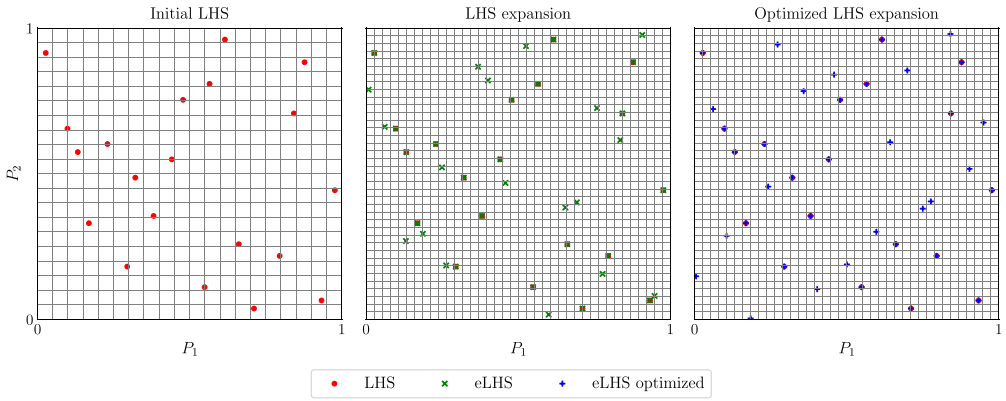

Fig. 3. Examples of LHS expansion. The left panel shows a standard LHS set with $N = 20$ samples in $P = 2$ dimensions (red circles). The middle panel shows a possible expansion with $M = 18$ new data points (green crosses), without further optimization. The right panel shows a different expansion, which was optimized to minimize the overall discrepancy. We report centered discrepancies of 1.0×10^{-3} and 4.7×10^{-4} for the unoptimized and the optimized expansions, respectively.

```
# LHS expansion
M = 18 # expansion size
eLHS = ExpandLHS(lhs_set)
elhs_set = eLHS(M)
elhs_opt = eLHS(M, optimize='discrepancy')
```

Outputs are illustrated in Fig. 3, where we show the initial set, an unoptimized expanded set, and the optimized expanded set. Optimizing using the discrepancy criterion increases the uniformity of the expanded sample set. For this case, we report a discrepancy estimate of $1.0{\times}10^{-3}$ for the unoptimized expansion and $4.7{\times}10^{-4}$ for the optimized expansion. This particular case happens to be a perfect expansion, $\mathcal{D}(\text{eLHS}) = 1$.

3.2. How many new samples?

In the following example, we simulate different expansions for the same initial set by adding a number of new points M between 4 and 12, ranking our attempts according to the resulting degree $\mathcal{D}$. Note that this metric depends only on the initial set and the value of M, and not on the added samples.

```
# optimal expansion size with M in [4, 12]
# verbose = False returns the optimal expansion
# verbose = True returns all the estimates
eLHS.optimal_expansion((4,12), verbose=True)
    M  degree
   (0,  1.0), # no expansion
   (12, 1.0), # best option
   (9,  0.9828),
   (7,  0.9815),
   (10, 0.9667),
   (6,  0.9615),
   (11, 0.9516),
   (5,  0.92),
   (4,  0.9167),
   (8,  0.9107) # worst option
```

We obtain a perfect expansion for $M = 12$, while the worst case is that with $M = 8$. This functionality offers some interesting use cases. In the example above, suppose one has budgeted computational time for $M = 8$ new simulations. It turns out a minor increase to $M = 9$ ensures substantially better coverage of the parameter space, with $\mathcal{D}$ increasing from ~ 0.91 to ~ 0.98.

3.3. General behavior of eLHS

We now present some general properties of our expansion algorithm with a particular focus on the degree metric introduced in Section 2. Fig. 4 shows the degree $\mathcal{D}$ as a function of the expansion size M, for initial LHS sets with different numbers of samples N and parameter-space dimensions P. For stability, results are averaged over multiple realizations.

Lower values of M correspond to a higher likelihood of overlaps, leading to a lower degree and thus a less space-filling eLHS. This effect diminishes as M increases; when $M > N$, the expansion dominates over the initial set, and the degree approaches 1. Fig. 4 shows "spikes" with $\mathcal{D} = 1$ taking place at multiples of the initial set size, i,e. $M = kN$ with $k \in \mathbb{N}$. This is not surprising: in those expansions, the existing boundaries of each bin remain unchanged and each initial interval is divided into k subintervals. This implies that overlaps never occur, resulting in a perfect expansion. Hints of this behavior were already presented in Refs. [18,25], even though the authors only considered the case with $M = N$. Less trivially, Fig. 4 shows that expansions with sizes $M = \left(k + \frac{1}{2}\right) N$ exhibit higher degrees than adjacent values, although they do not reach $\mathcal{D} = 1$.

We find that the LHS degree is largely independent of the number of dimensions P and presents a self-similar behavior with N. In other words, $\mathcal{D}$ depends on the ratio M/N and not on any of these two parameters independently. In particular, the degree is well predicted by the following phenomenological expression[1]

$$\mathcal{D} = 1 - \frac{1}{6\,(1 + M/N)^3}\,, \tag{3}$$

which, however, does not capture the high-$\mathcal{D}$ spikes described above.

Finally, the middle panel of Fig. 4 compares our results against repeated unitary expansions, that is, expanding the LHS M times with one new sample at a time, instead of performing a direct one-step expansion of size M as considered here. As expected, a unitary expansion strategy is highly suboptimal, as it leads to an overall increase in the likelihood of overlaps. Furthermore, it drastically increases the computational time by a factor $\propto M$.

3.4. Computational performance

Fig. 5 illustrates the computational cost of expandLHS as a function of M and P. Just-in-time compilation via Numba significantly speeds up the computation, particularly when the expansion is performed without optimization. Unsurprisingly, the execution time increases when optimization is required. In particular, optimization degrades the algorithm's scaling with the dimensionality of the hypercube.

Both of our optimization schemes show a similar trend as the number of samples grows, with discrepancy performing slightly better

[1] More precisely, a least-square fit to the curves of Fig. 4 with ansatz $\mathcal{D} = 1 + a(b + M/N)^c$ returns $a = -0.167$, $b = 1.01$, and $c = -2.99$.

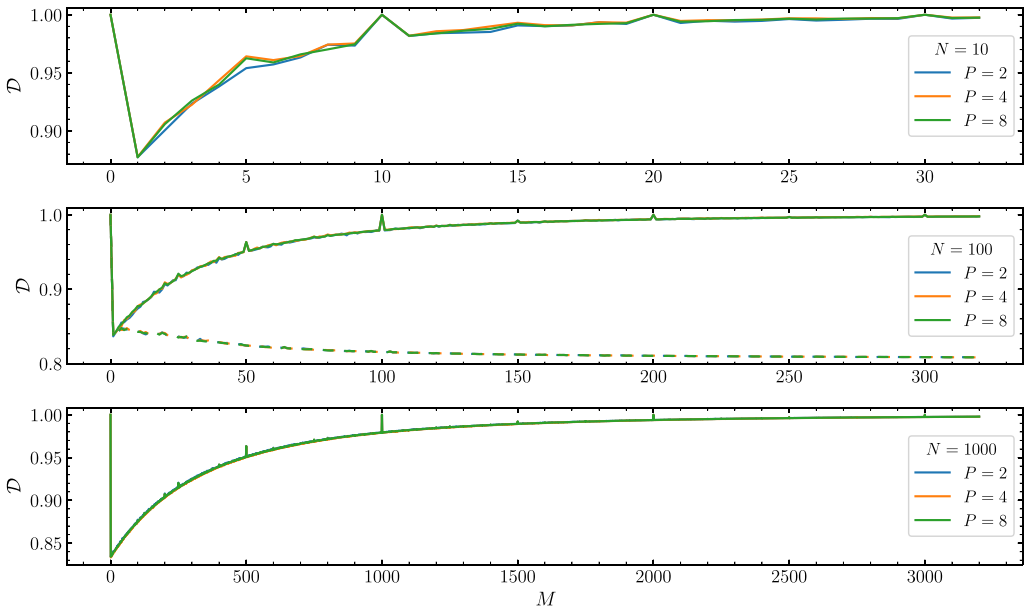

Fig. 4. The degree D of our LHS expansion as a function of the size M. The top, middle, and bottom panel shows results for an initial LHS set with $N = 10$, $N = 100$, and $N = 1000$ samples, respectively. We consider three hypercube dimensions: $P = 2$ (blue), $P = 4$ (orange), and $P = 8$ (green). The middle panel also considers the case of repeated unitary expansions, adding one sample M times until the required expansion size is reached (dashed curves). To avoid highlighting particularly poor or favorable configurations, results are averaged over 100 Latin hypercube realizations.

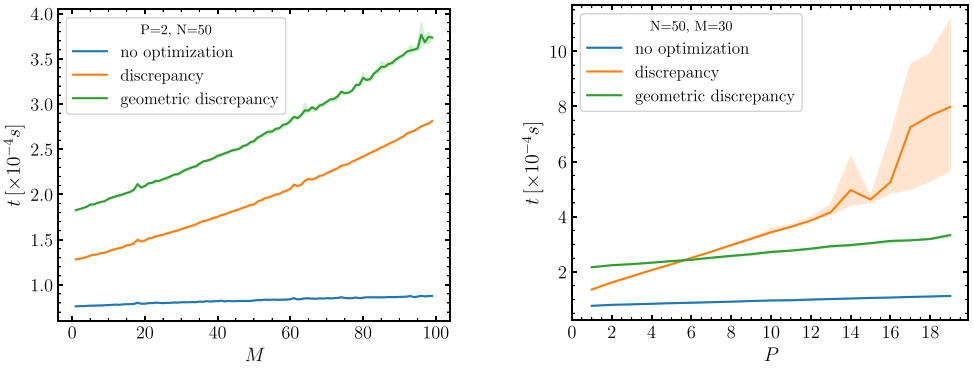

Fig. 5. Computational performance of the "LHS in LHS" expansion algorithm. The left panel shows computational times for different expansion sizes M of an initial LHS set with $N = 50$ samples in $P = 2$ dimensions. The right panel shows computational times for $N = 50$ and $M = 30$ when varying the number of hypercube dimensions P. Different curves represent variations of the expansion algorithm: no optimization in blue, discrepancy minimization in orange, and geometric discrepancy maximization in green. Solid curves show median values over 1000 LHS realizations, while the shaded regions encompass 90% of the cases. The time estimates reported in this figure were obtained by running expandLHS in a single thread on an AMD EPYC Rome processor.

than geometric discrepancy. This scaling behavior can be explained as follows. The discrepancy focuses on uniformly filling the hypercube and quantifies the difference between the input distribution of points and the expected uniform coverage. Its evaluation requires comparing the fraction of samples in a hypercube subset with the fraction of volume it occupies, which depends heavily on P and mildly on M. On the other hand, the geometric discrepancy estimates the minimum distance between any pair of points and is thus mainly affected by the number of samples $N + M$.

4. Impact

LHS plays a major role in simulation design, as evidenced by the vast literature on the topic (for a review see Ref. [4]). The additional flexibility provided by our expansion algorithms broadens the use cases of this powerful tool. One interesting application is in training machine learning models. A model initially trained on a set of N samples may fail to achieve the required accuracy. A common solution in machine learning is to increase the training set size or extend the coverage of the input space. An expansion algorithm for LHS can assist in selecting an appropriate distribution for new training samples while preserving existing information. A study on this topic is presented in Ref. [26]. To this end, expanded LHS might complement the usage of learning curves in machine learning [27]. Quoting the specific field of research of some of us, gravitational-wave astronomy, two activities to which we plan to apply expandLHS are the development of numerical-relativity surrogate models [28,29] and the interpolation of stellar-physics simulations [30,31].

Our expandLHS algorithm shares some similarities with the method presented in Ref. [20]. The key differences are that our approach assumes fewer constraints on the final samples and directly targets the required number of additional samples M while allowing for expansions that are not perfect Latin Hypercube sets. Ref. [18] also presented a related algorithm, which they dubbed Progressive LHS, supported by a public Python implementation. Their approach samples an LHS while also ensuring additional properties derived from Sliced Latin Hypercube sampling [32]. In particular, they construct a sequence of LHS slices whose progressive union still forms a Latin hypercube.

The complete set obtained by the union of all slices constitutes a Latin hypercube that maximizes space-filling properties. Crucially, this remains a *one-stage* sampling procedure, meaning that the size of the final Latin hypercube and the number of slices must be specified *a priori*. Moreover, the algorithm permits sampling only those configurations in which the total number of samples is an integer multiple of the number of slices. Our implementation is different; the regridding and "LHS in LHS" operations we propose are arguably closer to the original procedure for distributing points via LHS and, by relaxing the requirement of a perfect Latin hypercube, we allow for expansions of any size.

5. Conclusions

In this paper, we presented a new algorithm for expanding an existing LHS while optimally preserving its space-filling properties. Our procedure leverages the flexibility of LHS by embedding a new LHS within the initial one. Loss of optimality arises from the potential occurrence of overlapping samples in the regridded space, as quantified by the LHS degree D.

Our new algorithm has broad applications in simulation design and machine learning, assisting in scenarios where a carefully planned initial simulation must later be expanded to meet targeted requirements. However, the expansion strategy introduced in Section 2 is specialized in sampling the uniform hypercube. Other LHS-like algorithms have been proposed to sample constrained spaces, handle non-uniform boundaries, and incorporate weighted samples (e.g. [33–35]). Generalizing our implementation to these non-standard LHS techniques is left to future work.

Our implementation, expandLHS, is available as an open-source package for the Python programming language, utilizing fast array manipulation and just-in-time compilation.

CRediT authorship contribution statement

Matteo Boschini: Writing – original draft, Software, Methodology, Formal analysis, Conceptualization. **Davide Gerosa:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Alessandro Crespi:** Investigation. **Matteo Falcone:** Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

M.B., D.G., A.C. and M.F. are supported by ERC Starting Grant No. 945155–GWmining, Cariplo Foundation, Italy Grant No. 2021-0555, MUR PRIN, Italy Grant No. 2022-Z9X4XS, MUR Grant "Progetto Dipartimenti di Eccellenza 2023-2027" (BiCoQ), Italy, and the ICSC National Research Centre funded by NextGenerationEU, Italy. D.G. is supported by MSCA Fellowship, Italy No. 101064542–StochRewind, MSCA Fellowship, Italy No. 101149270–ProtoBH, and MUR Young Researchers, Italy Grant No. SOE2024-0000125. Computational work was performed at CINECA with allocations through INFN and Bicocca.

References

[1] Mittal S, Durak U, Tuncer Ö. Guide to simulation-based disciplines. Springer Cham; 2017, http://dx.doi.org/10.1007/978-3-319-61264-5.

[2] Fang K, Li R, Sudjianto A. Design and modeling for computer experiments. CRC Press; 2005, http://dx.doi.org/10.1201/9781420034899.

[3] Garud SS, Karimi IA, Kraft M. Design of computer experiments: A review. Comput Chem Eng 2017;106:71–95. http://dx.doi.org/10.1016/j.compchemeng.2017.05.010.

[4] Devon Lin C, Tang B. Latin hypercubes and space-filling designs. In: Handbook of design and analysis of experiments. CRC Press; 2015, p. 593–626. http://dx.doi.org/10.48550/arXiv.2203.06334.

[5] Eglajs V, Audze P. New approach to the design of multifactor experiments. Probl Dyn Strengths 1977;35(1):104–7.

[6] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 1979;21(2):239–45. http://dx.doi.org/10.2307/1268522.

[7] Iman RL, Conover W. Small sample sensitivity analysis techniques for computer models.with an application to risk assessment. Commun Stat A-Theor 1980;9(17):1749–842. http://dx.doi.org/10.1080/03610928008827996.

[8] Florian A. An efficient sampling scheme: Updated Latin hypercube sampling. Probabilist Eng Mech 1992;7(2):123–30. http://dx.doi.org/10.1016/0266-8920(92)90015-A.

[9] Tang B. Orthogonal array-based Latin hypercubes. J Amer Statist Assoc 1993;88(424):1392–7. http://dx.doi.org/10.1080/01621459.1993.10476423.

[10] Park J-S. Optimal Latin-hypercube designs for computer experiments. J Statist Plann Inference 1994;39(1):95–111. http://dx.doi.org/10.1016/0378-3758(94)90115-5.

[11] Morris MD, Mitchell TJ. Exploratory designs for computational experiments. J Statist Plann Inference 1995;43(3):381–402. http://dx.doi.org/10.1016/0378-3758(94)00035-T.

[12] Ye KQ. Orthogonal column Latin hypercubes and their application in computer experiments. J Amer Statist Assoc 1998;93(444):1430–9. http://dx.doi.org/10.1080/01621459.1998.10473803.

[13] Ye KQ, Li W, Sudjianto A. Algorithmic construction of optimal symmetric Latin hypercube designs. J Statist Plann Inference 2000;90(1):145–59. http://dx.doi.org/10.1016/S0378-3758(00)00105-1.

[14] Cioppa TM, Lucas TW. Efficient nearly orthogonal and space-filling Latin hypercubes. Technometrics 2007;49(1):45–55. http://dx.doi.org/10.1198/004017006000000453.

[15] F. Xiong WC, Yang S. Optimizing Latin hypercube design for sequential sampling of computer experiments. Eng Optim 2009;41(8):793–810. http://dx.doi.org/10.1080/03052150902852999.

[16] Crombecq K, Laermans E, Dhaene T. Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling. European J Oper Res 2011;214(3):683–96. http://dx.doi.org/10.1016/j.ejor.2011.05.032.

[17] Vořechovskỳ M. Hierarchical refinement of Latin hypercube samples. Comput-Aided Civ Infrastruct Eng 2015;30(5):394–411. http://dx.doi.org/10.1111/mice.12088.

[18] Sheikholeslami R, Razavi S. Progressive Latin hypercube sampling: An efficient approach for robust sampling-based analysis of environmental models. Env Model Softw 2017;93:109–26. http://dx.doi.org/10.1016/j.envsoft.2017.03.010.

[19] Wei Li XX, Yang M. A novel extension algorithm for optimized Latin hypercube sampling. J Stat Comput Sim 2017;87(13):2549–59. http://dx.doi.org/10.1080/00949655.2017.1340475.

[20] Zhou X, Lin DKJ, Hu X, Ouyang L. Sequential Latin hypercube design with both space-filling and projective properties. Qual Reliab Eng Int 2019;35(6):1941–51. http://dx.doi.org/10.1002/qre.2485.

[21] Zhou Y-D, Fang K-T, Ning J-H. Mixture discrepancy for quasi-random point sets. J Complexity 2013;29(3):283–301. http://dx.doi.org/10.1016/j.jco.2012.11.006.

[22] Harris CR, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[23] Virtanen P, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nat Methods 2020;17:261–72. http://dx.doi.org/10.1038/s41592-019-0686-2.

[24] Lam SK, Pitrou A, Seibert S. Numba: a LLVM-based Python JIT compiler. In: LLVM compiler infrastructure in HPC. LLVM '15, New York, NY, USA: Association for Computing Machinery; 2015, http://dx.doi.org/10.1145/2833157.2833162.

[25] Sallaberry C, Helton J, Hora S. Extension of Latin hypercube samples with correlated variables. Reliab Eng Syst Saf 2008;93(7):1047–59. http://dx.doi.org/10.1016/j.ress.2007.04.005, Bayesian Networks in Dependability.

[26] Iordanis I, Koukouvinos C, Silou I. On the efficacy of conditioned and progressive Latin hypercube sampling in supervised machine learning. Appl Numer Math 2025;208:256–70. http://dx.doi.org/10.1016/j.apnum.2023.12.016.

[27] Mohr F, van Rijn JN. Learning curves for decision making in supervised machine learning: a survey. Mach Learn 2024;113(11):8371–425. http://dx.doi.org/10.1007/s10994-024-06619-7.

[28] Varma V, Field SE, Scheel MA, Blackman J, Gerosa D, Stein LC, et al. Surrogate models for precessing binary black hole simulations with unequal masses. Phys Rev Res 2019;1(3):033015. http://dx.doi.org/10.1103/PhysRevResearch.1.033015.

[29] Boschini M, Gerosa D, Varma V, Armaza C, Boyle M, Bonilla MS, et al. Extending black-hole remnant surrogate models to extreme mass ratios. Phys Rev D 2023;108(8):084015. http://dx.doi.org/10.1103/PhysRevD.108.084015.

[30] Taylor SR, Gerosa D. Mining gravitational-wave catalogs to understand binary stellar evolution: A new hierarchical Bayesian framework. Phys Rev D 2018;98(8):083017. http://dx.doi.org/10.1103/PhysRevD.98.083017.

[31] Mould M, Gerosa D, Taylor SR. Deep learning and Bayesian inference of gravitational-wave populations: Hierarchical black-hole mergers. Phys Rev D 2022;106(10):103013. http://dx.doi.org/10.1103/PhysRevD.106.103013.
[32] Qian PZG. Sliced Latin hypercube designs. J Amer Statist Assoc 2012;107(497):393–9. http://dx.doi.org/10.1080/01621459.2011.644132.
[33] Petelet M, Iooss B, Asserin O, Loredo A. Latin hypercube sampling with inequality constraints. AStA Adv Stat Anal 2010;94(4):325–39. http://dx.doi.org/10.1007/s10182-010-0144-z.
[34] Minasny B, McBratney AB. A conditioned Latin hypercube method for sampling in the presence of ancillary information. Comput Geosci 2006;32(9):1378–88, URL https://10.1016/j.cageo.2005.12.009.
[35] Schenk C, Haranczyk M. A novel constrained sampling method for efficient exploration in materials and chemical mixture design. Comput Mater Sci 2025;252:113780. http://dx.doi.org/10.1016/j.commatsci.2025.113780.