Design of Experiment (DoE)

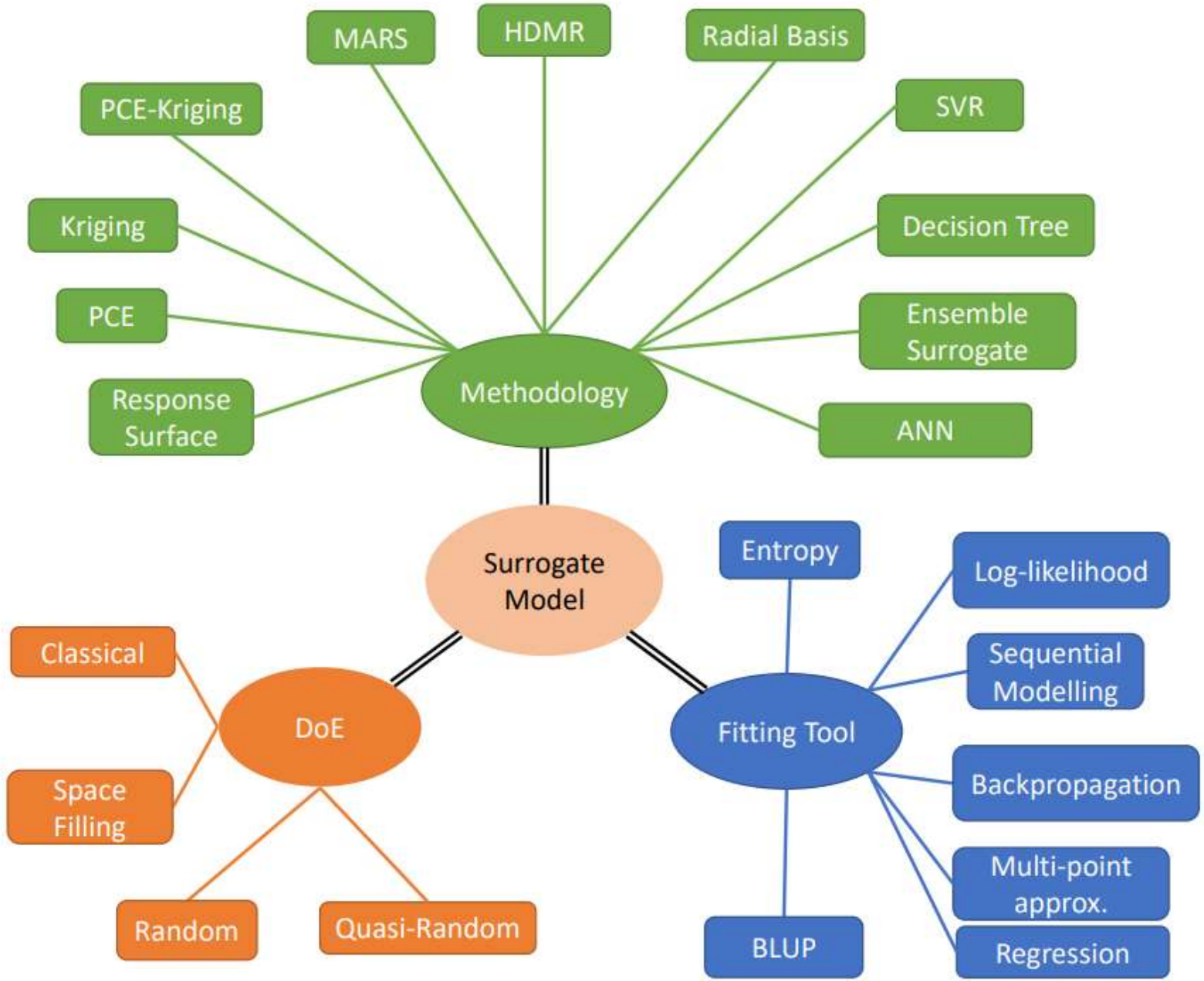
20 March 2024

Melik Kaan Şelale

Surrogate Model

- Monte Carlo Simulations is the oldest and most widely used technique in design optimization
  - But it requires large number of model evaluations and computationaly costly.

- Surrogate models, developed using DoE are worthwhile options


MARS
HDMR
Radial Basis
PCE-Kriging
SVR
Kriging
Decision Tree
PCE
Ensemble Surrogate
Response Surface
Methodology
ANN
Surrogate Model
Entropy
Log-likelihood
Classical
Sequential Modelling
DoE
Fitting Tool
Space Filling
Backpropagation
Multi-point approx.
Random
Quasi-Random
BLUP
Regression

Design of Experiment

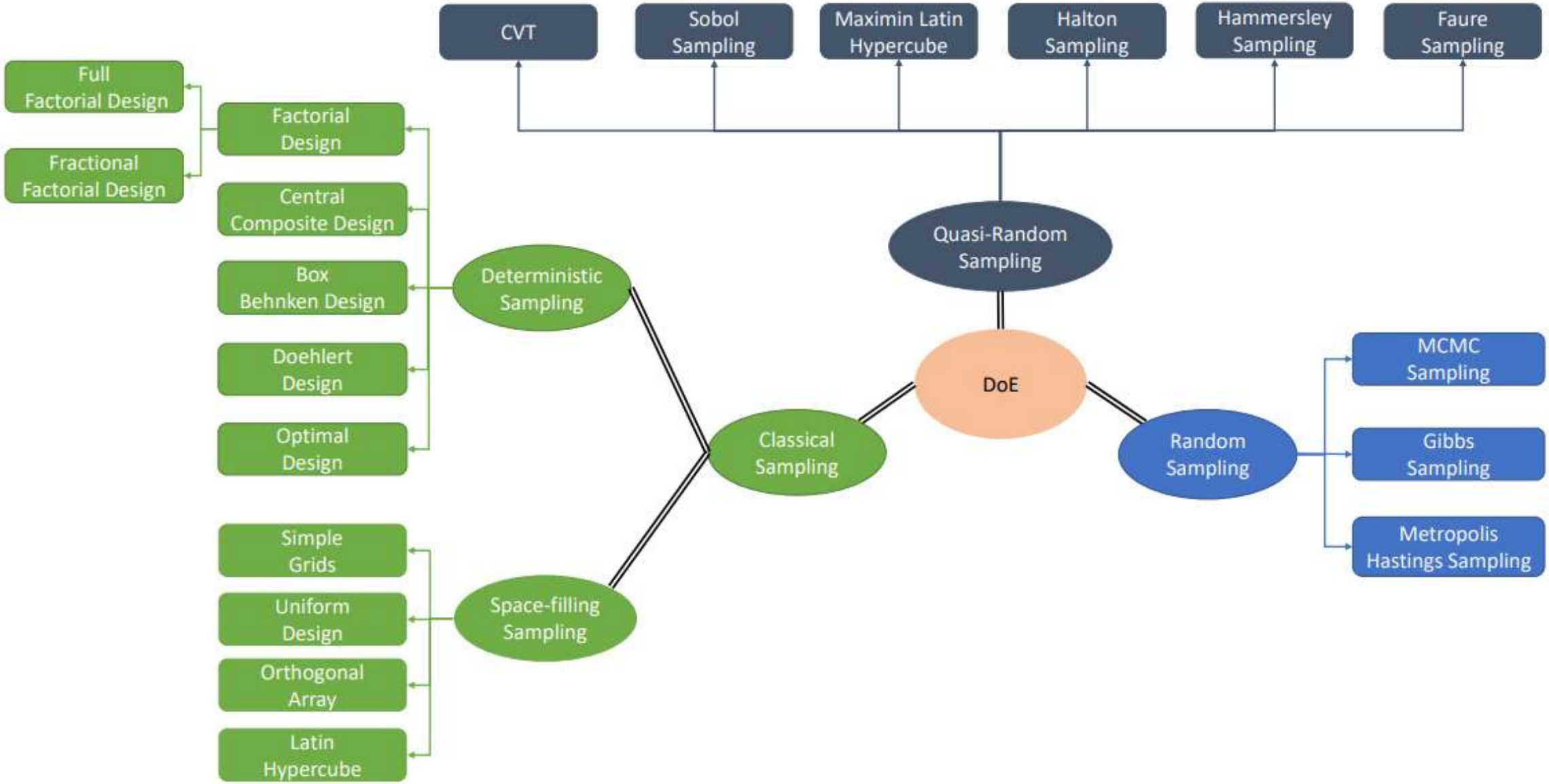
- Purpose of using DoE: the samples cover the entire desing space, error of predicted result decreases

- DoE contains different methods to generate train data set
- Then different methodologies can be used to build surrogate model
- Lastly minimize the noise between actual and predicted data using fitting tools based on knowledge of DoE samples.

DoE Sampling Strategy

- To explore on experimental region efficiently, the expertimental desing in a computer experiment is usually required to satisfy two properties:
  - Design should be space-filling, acquire the maximum amount of information on the expertimental domain
  - Design should be non-collapsing. Nearly identical responses offer little information, and that result in a waste of computation
- The concept of DoE introduced by the English statistician Ronald Fisher

- DoE => Classical DoE and Modern DoE
- DoE samples are generated using two approaches:
  - Domain Based: based on the information obtained from the design space
  - Response Based: based on the information obtained from the surrogate models.


CVT
Sobol Sampling
Maximin Latin Hypercube
Halton Sampling
Hammersley Sampling
Faure Sampling
Full Factorial Design
Factorial Design
Fractional Factorial Design
Central Composite Design
Box Behnken Design
Doehlert Design
Optimal Design
Quasi-Random Sampling
Deterministic Sampling
DoE
Classical Sampling
Random Sampling
MCMC Sampling
Gibbs Sampling
Metropolis Hastings Sampling
Simple Grids
Uniform Design
Orthogonal Array
Latin Hypercube
Space-filling Sampling

- Classical DoE
  - Deterministic Sampling: The sample points generated uniformly over the desing space. Therefore, most of the samples are located at the boundary of the design space
  - Space-filling Sampling: Mainly used to enhance the accuracy of the surrogate models. Iteratively updating surrogate model until a satisfactory accuracy is achieved.
- Modern DoE
  - The samples are generated with an equally likely probability of accurance of samles within the design space
  - This technique is suitable only when the proper knowledge of design space is available because poor coverage regions of samples ahve been observed for high dimensional problems

Deterministic Sampling

- **Full Factorial Design**
  - Variables discretized into two or three levels and layers.
  - Min and max values on axes
  - Difference between two and three levels: Center value
  - Curse of dimensionality!
- **Fractional Factorial Design**
  - Lowers computational cost where fewer sampling point are generated
  - Still curse of dimensionality

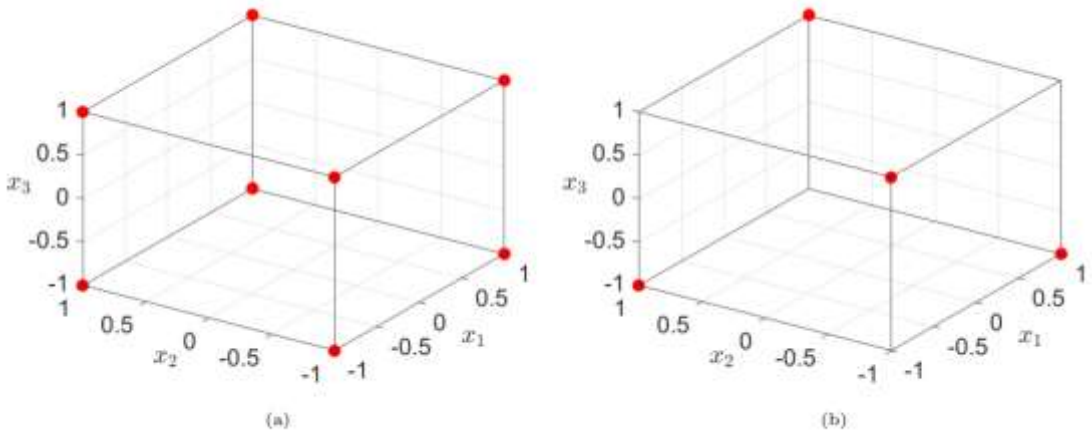


Fig. 3 2-levels (a) full and (b) fractional factorial design

- Number of simulations:
  - Full Factorial:
    - Two Level: 2^n
    - Three Level: 3^n
  - Fractional Factorial:
    - Two Level: 2^{n-k}
    - Three Level: 3^{n-k}
- Factorial design fails to capture the non-linearity of the system

- **Control Composite Desing (CCD)**
  - Similar to factorial design, with additional sampling points at the center and axial directions
  - Two additional axial points for each variable and the center point of hypercube

  - Circumscribed(CCC): Axial points are located outside of the cube where the cube points take (-1) and (+1) values
  - Inscribed(CCI): Same as CCC, but scaled where axial points are taken (-1) and (+1) values
  - Faced(CCF): The axial points are placed on the face of the cube

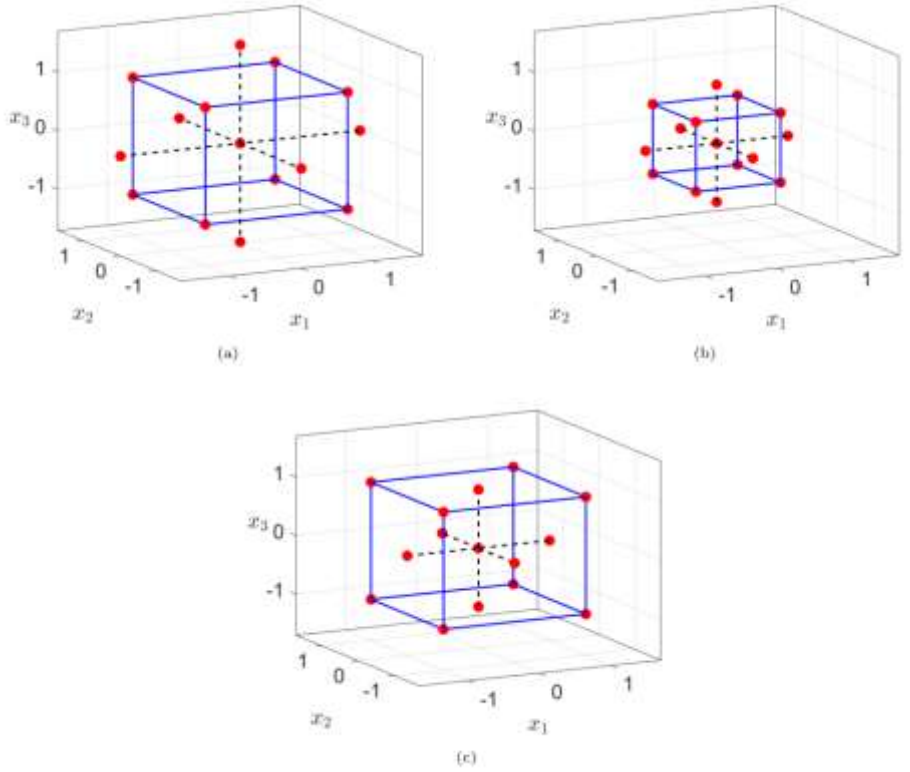


Fig. 4 Central composite design: (a) Circumscribed, (b) Inscribed and (c) Faced

- **Number of simulations:**

  Two Level: $2^n + 2n + N_0$

  Where N_0 number of central point

- **Box-Behnkend Design**
  - Similar to CCD
  - Sample points located at midpoints not at min-max points
  - Requires less number of simulations
  - Also suffers from curse of dimensionality

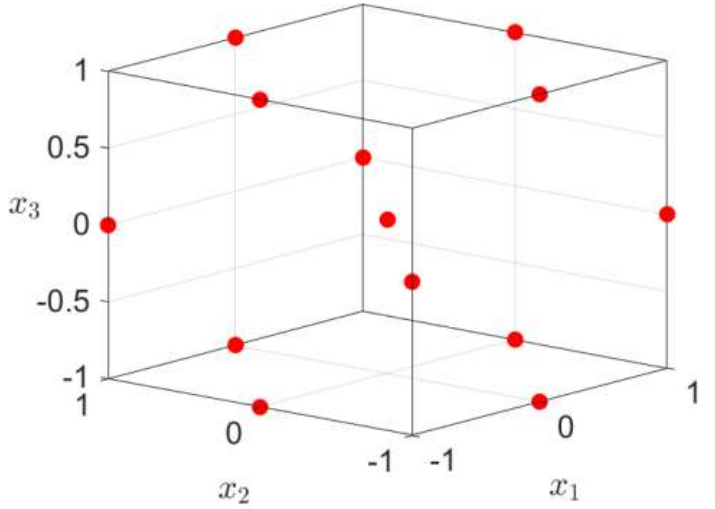
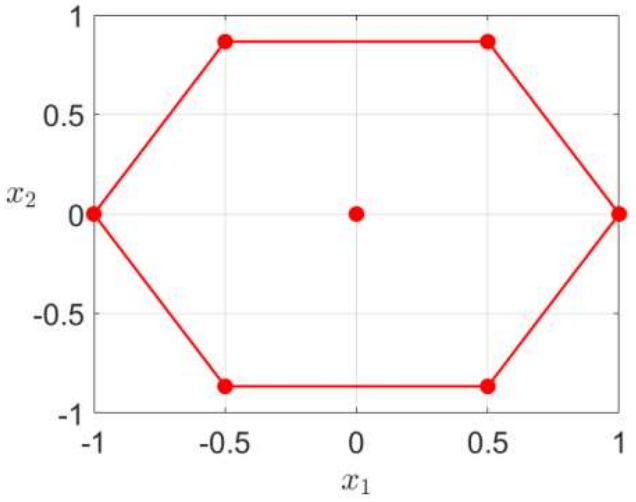
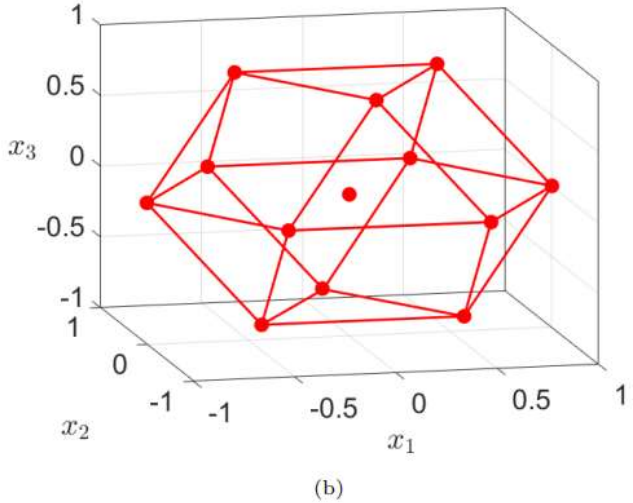


Fig. 5 Box-Behnken design

- **Doehlert Design**
  - More efficient in filling space uniforml
  - Samples are not rotatable due to the number of estimations for different input variables
  - Provides different structures for the different number of factors.
    - For two variables: circular domain
    - For three variables: spherical domain
    - For four or more: hyperspherical domain



(a)



(b)

Fig. 6 Doehlert design for (a) two and (b) three factors

- **Number of simulations:**

  Two Level: $n^2 + n + N_0$

  Where N_0 number of central point

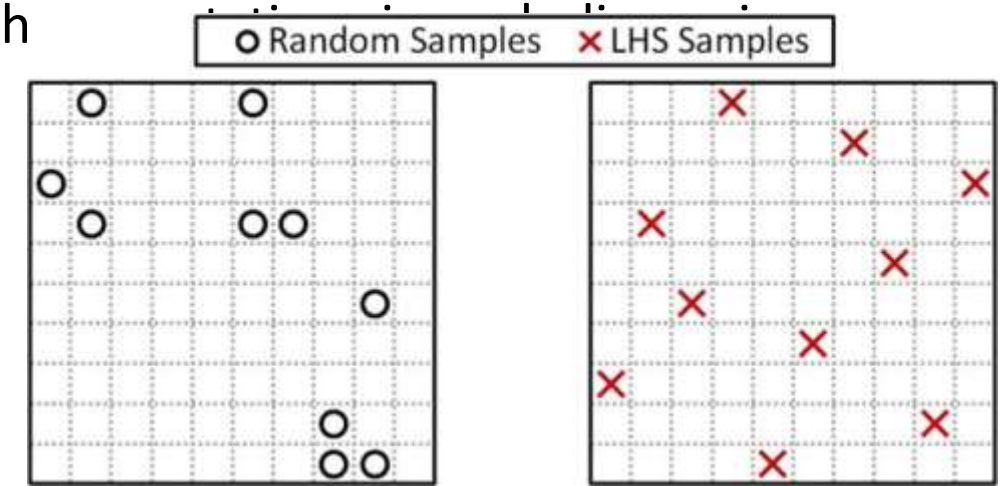
No. of Factors	No. of Coefficient	No. of Runs/ Simulations			
		Factorial	CCD	BBD	Doehlert
2	6	9	9	-	7
3	10	27	15	13	13
4	15	81	25	25	21
5	21	243	43	41	31
6	28	729	77	61	43
7	36	2187	143	85	57
8	45	6561	273	113	73

The Modern Things

- **Quasi-Random Sampling**
  - Also known as low-discrepancy or deterministic sampling
  - Neither fully random nor regular
  - More deterministic way, aiming to achieve more even distribution than random
  - Advantages:
    - Even coverage: reduces clustering and ensures a more uniform distribution
    - Low Discrepancy: reduction of sampling errors and more accurate representation
    - Less sensitivity to initial conditions
    - Effective in Higher Dimensions

Maximin Latin Hypercube Sampling

- Latin Hypercube Sampling
  - Idea: Recreate the distribution through less samples with determining samples randomly and single on each stratification
  - To generate N samples in d dimensions using Latin hypercube sampling, each dimension's range is divided into N equispaced intervals. Within these intervals, coordinate values are selected, either randomly or at the interval center. In the Latin hypercube approach, a permutation is applied in which a random value is chosen from N equispaced bins in the [0, 1] range. Subsequently, sample points are formed by randomly selecting one coordinate value per dimension without replacement, achieved through

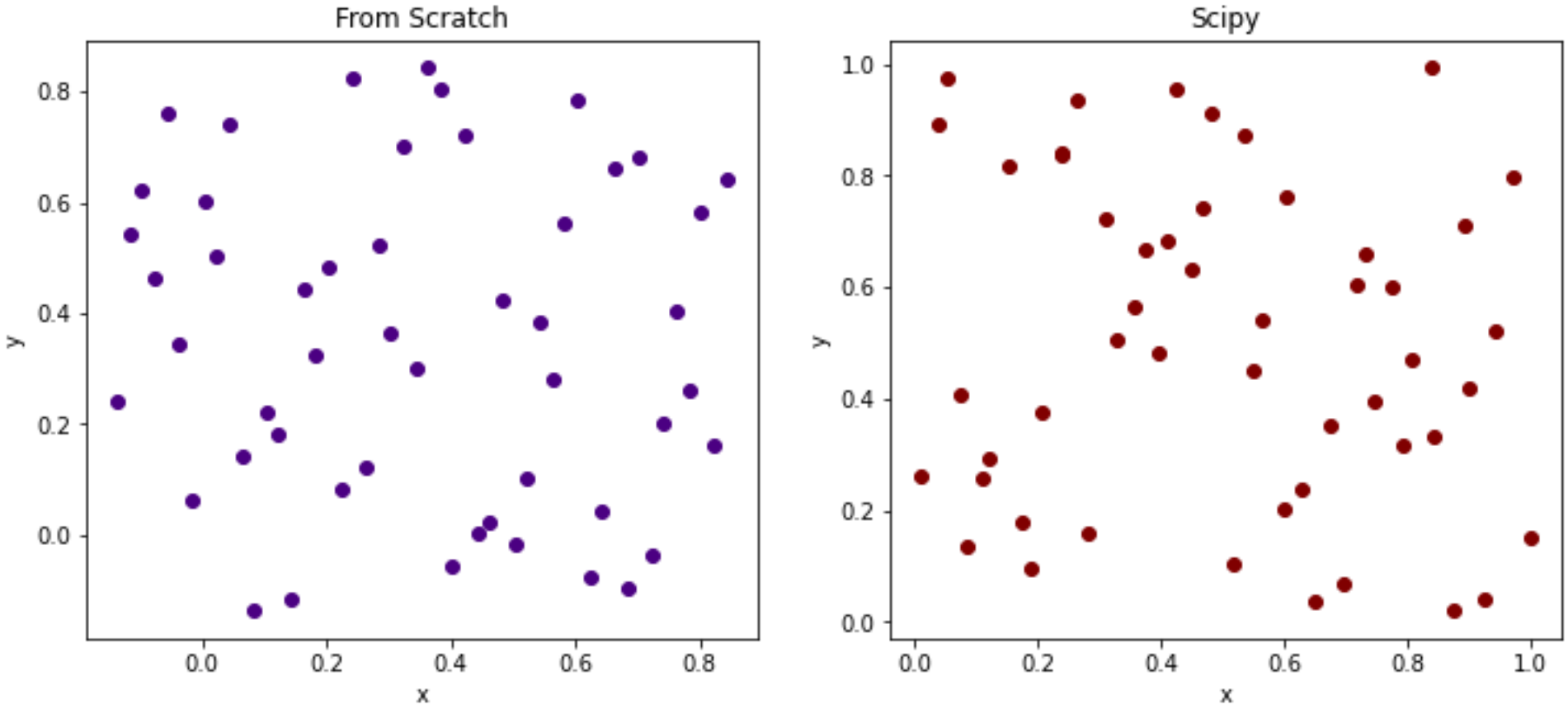


Pseudo-code of the LHS

Algorithm 1 Basic Latin hypercube sampling.

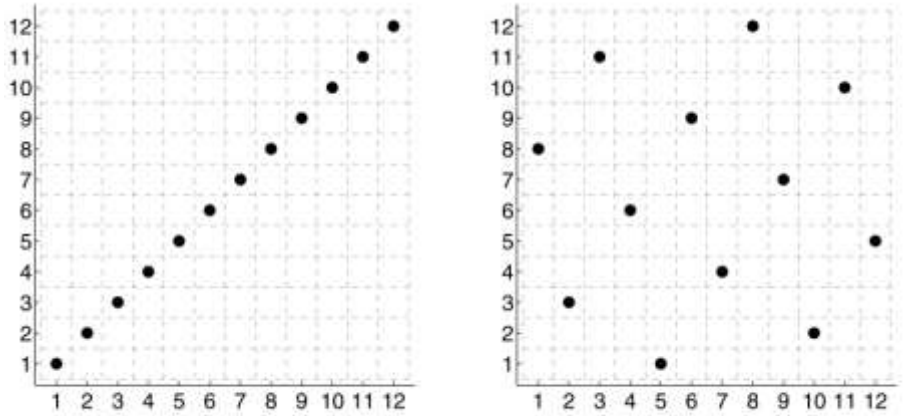
1: Goal: generate N samples in a d-dimensional space
2: Let x_{ij} denote the j-th coordinate of the i-th sample $\mathbf{x}_i$.
3: **for** $j = 1$ to d **do**
4: Generate P_j, a random permutation of the set $\{1, \ldots, N\}$, with the i-th element denoted by $P_j(i)$.
5: **end for**
6: **for** $j = 1$ to d **do**
7: **for** $i = 1$ to N **do**
8: Generate $x_{ij} = \frac{P_j(i) - U_j(i)}{N}$, where $U_j(i)$ is a uniform random number in $[0, 1]$.
9: **end for**
10: **end for**

My LHS Samples vs. Scipy LHS Samples



Github link of the code:https://github.com/kaans7/Latin-Hypercube-Sampling-from-Scratch

- **Problems of LHS**
  - Latin hypercube designs (LHDs) are designed to be non-collapsing, but randomness in their generation may lead to non-space-filling outcomes
  - Achieving a high-quality LHS is particularly challenging, especially in high-dimensional spaces or when dealing with a large number of samples.



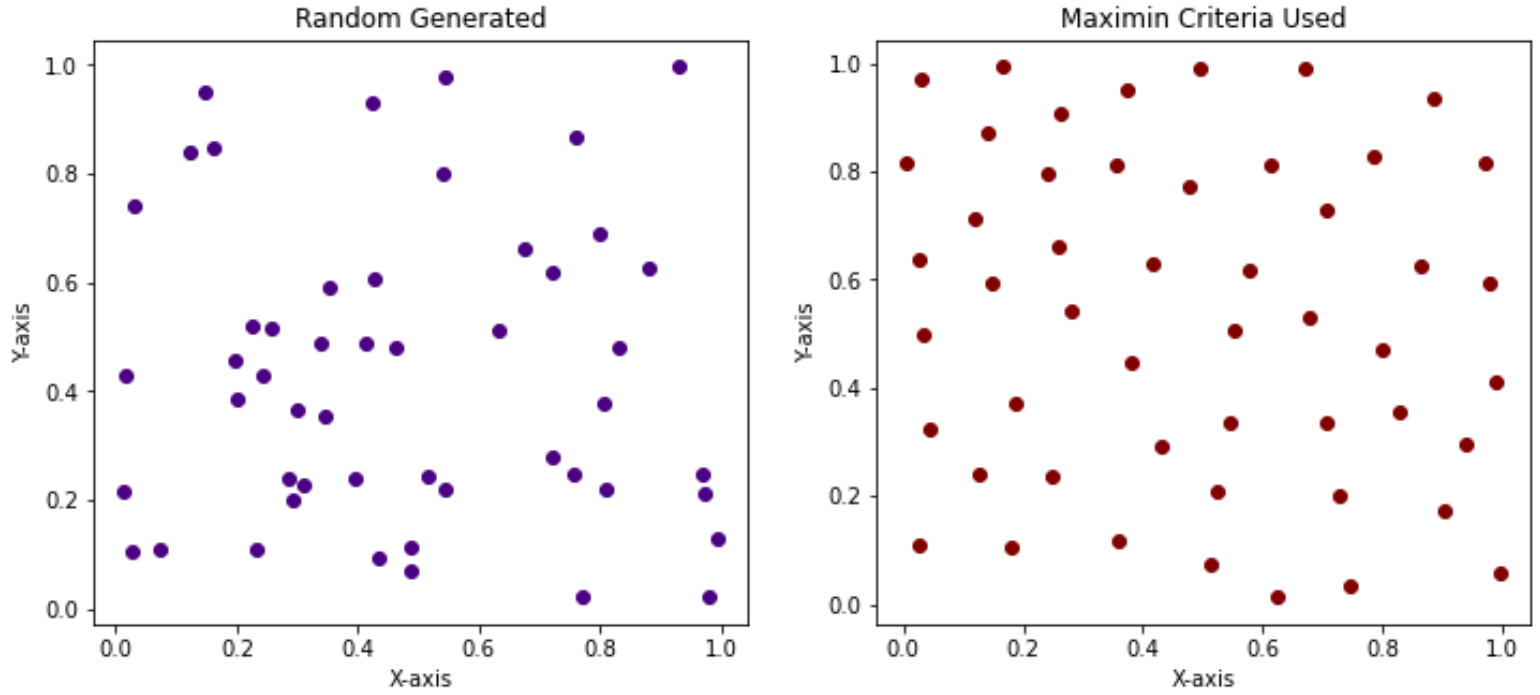
(a) A non-space-filling LHD (b) Max-min LHD

  - Various criteria, such as the max-min criterion, are employed to enhance the space-filling property.(Also the Audze-Eglais criteria, the Entropy criteria, the centered L2-discrepancy)
  - Although different methods, the common challange is the huge number of feasible design renders the search for the optimal LHDs as non-trivial especially for large n or k

Maximin Criterion

$$\max_{D}\left\{\min_{\forall j>i} d(\mathbf{x}_i, \mathbf{x}_j)\right\},$$

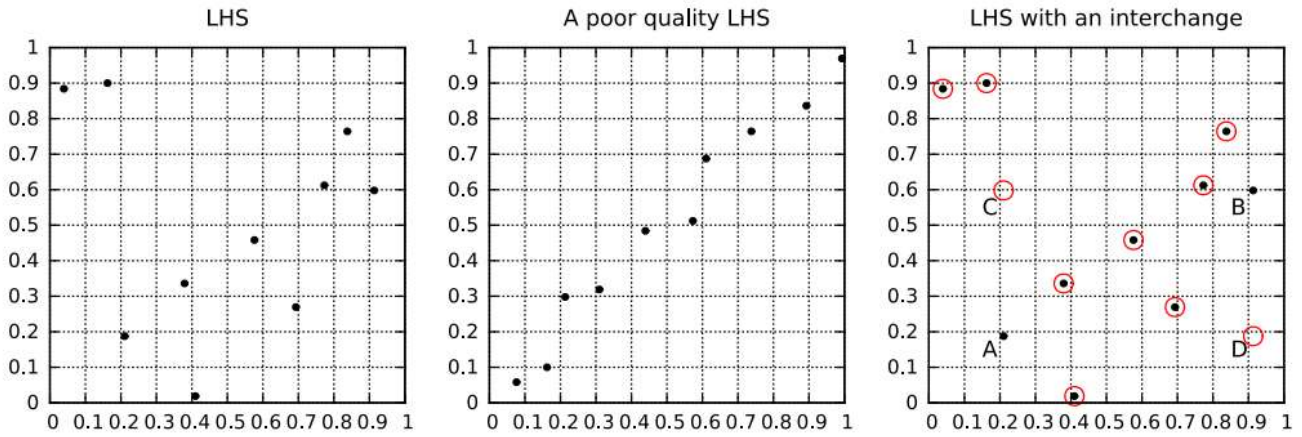
- The maximin criterion is a decision-making approach that involves selecting an option by identifying the maximum possible outcome for the minimum payoff. This technique aims to minimize risk and focuses on choosing alternatives with the most favorable worst-case scenario, prioritizing robustness in the face of uncertainty.



Github link of the code: https://github.com/kaans7/Maximin-Criteria-from-Scratch

• Maximin Latin Hypercube Sampling

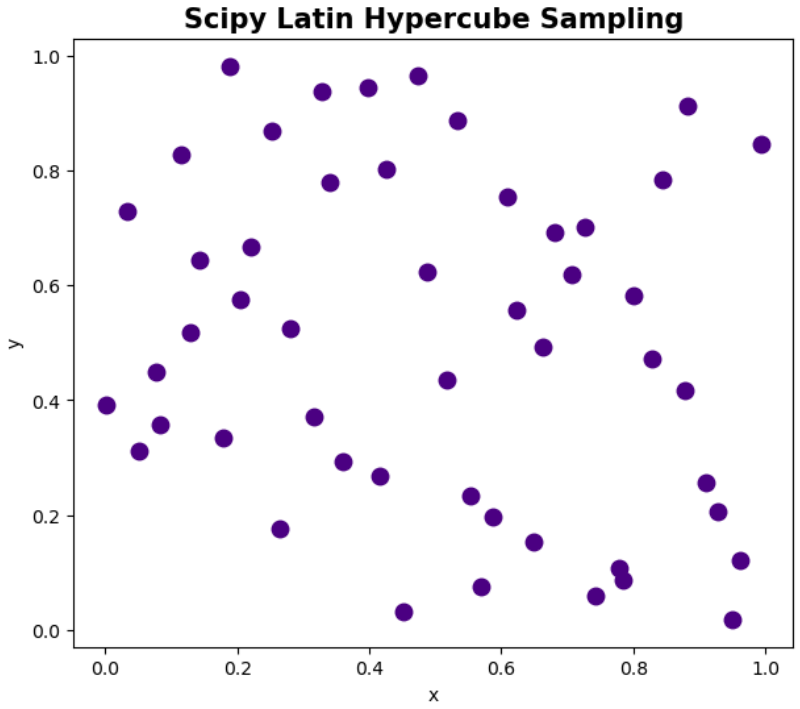
- As a result of the problems, much of the research in LHS has focused on improving the coverage of the sampling
- LHS research aims to improve sampling coverage by employing interchanges, which modify an initial LHS to optimize a defined quality metric.
- In an interchange, a random dimension is chosen, and the coordinate values of two randomly-selected rows in that dimension are swapped, preserving the Latin property.
- Interchanges are accepted based on their ability to enhance sampling quality aiming to maximize the minimum distance between samples
- However, optimizing this process is challenging due to the large space of possible interchanges.

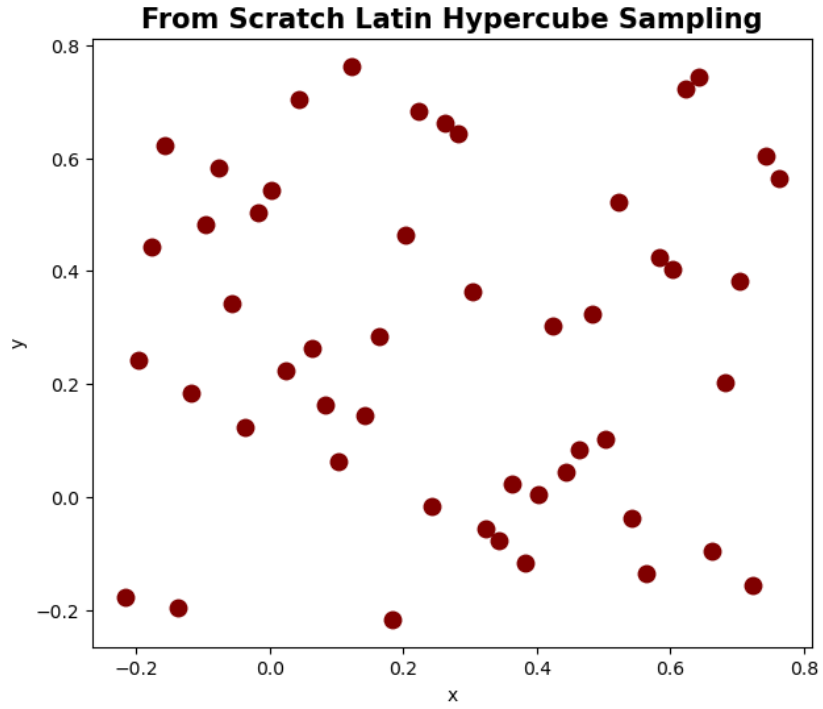


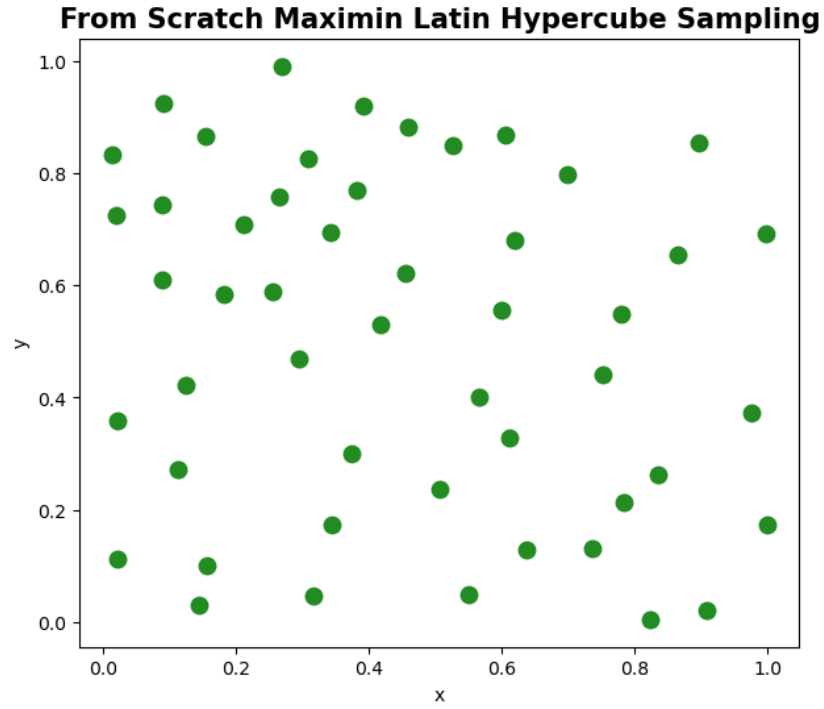
Algorithm 3 Pseudo-code for generation of Maximin Latin Hypercube Sampling [109]

Aim: Generate n samples over the N-dimensional space in which the minimum distance between samples are maximized.
Define the number of iteration (N_{Iter}) and the number of interchanges for every iteration (m).
Begin
for Iteration $= 1$ to N_{Iter} **do**
    Generate n samples using basic Latin hypercube sampling, shown in Algorithm 2.
    Assume p_1 and p_2 are the indices of samples with minimum distance.
    for $i = 1$ to m **do**
        Select index random either p_1 or p_2.
        For a random integer, r_{col} in $[1, \ldots, N]$, and a random row, r_{row} in $[1, \ldots, n]$, interchange the values in column r_{col}, rows index and r_{row}.
        If the minimum distance of samples is increased, accept the interchanges and update the values of p_1 and p_2.
    end for
    If the minimum distance between samples is smaller than previous iteration, terminate.
end for

LHS vs. Maximin LHS





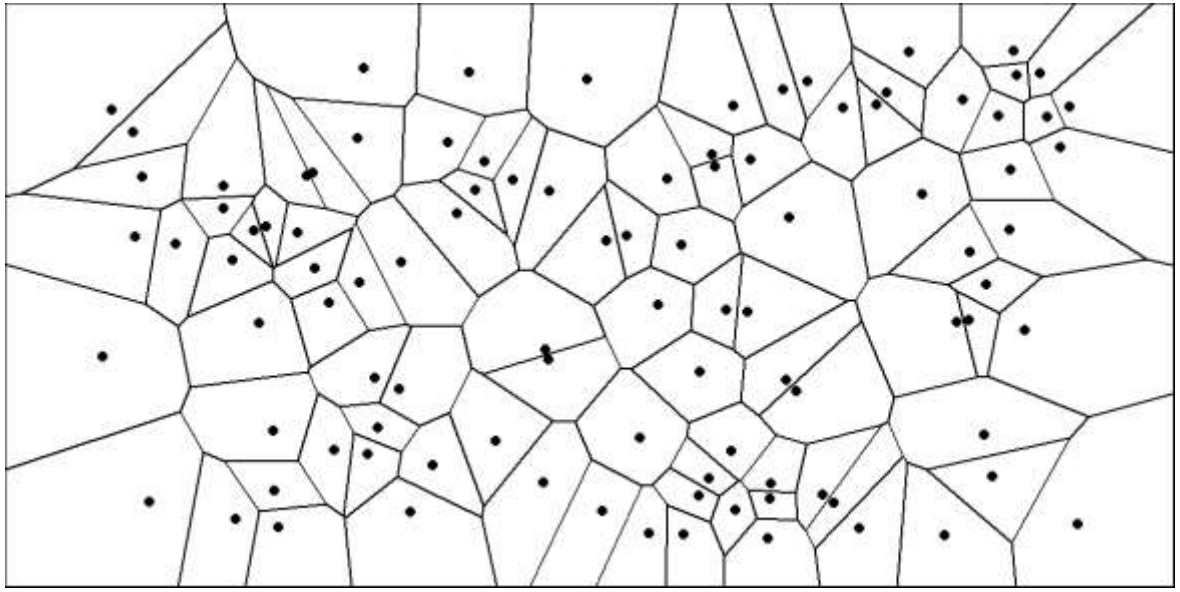


Github link of the code:https://github.com/kaans7/Latin-Hypercube-Sampling-from-Scratch

Centroidal Voronoi Tesselation Sampling(CVT)

- Voronoi Tesselations
  - Given a set of points(generators) and a distance function, Voronoi tesselations are subdivisions of another set of points into subsets such that the points in each subset are closest

- Key point: Each point one a region is closest to its own generator
- Generally generators are not the center of mass points of voronoi tesselations
- Centroidal Voronoi Tesselations(CVTs) are special voronoi tesselations such that generators are also correspond the center of mass of voronoi tesselations



- Algorithms for constructing CVTs could be both deterministic and probabilistic. But in both case the algorithms are iterative

- Determinstic (Lloyd's Method):
  + Small of number iterations
  - Cost per iteration

- Probabilistic (McQueen's Method):
  + Cost per iteration, no need to construction of Voronoi Tesselations and the apporximate multi dimensional integral
  - Number of iteration is huge

Mathematics of CVTs

Centroidal Voronoi Tesselations

Given an open set $\Omega \subset R^N$ and a set of points $\{z_i\}_{i=1}^k$ belonging to $\bar{\Omega}$ let

$$V_i = \{x \in \Omega \mid |x - z_i| < |x - z_j| \text{ for } j = 1, \dots, k, j \neq i\} \quad i = 1, \dots, k$$

Clearly, we have

$$V_i \cap V_j = \emptyset \text{ for } i \neq j \quad \text{and} \quad \bigcup_{i=1}^k \bar{V}_i = \bar{\Omega}$$

The set $\{V_i\}_{i=1}^k$ is referred to as a Voronoi tesselation or Voronoi Diagram of Ω.

Voronoi Diagram: $\{V_i\}_{i=1}^k$

Generators: $\{z_i\}_{i=1}^k$

Given a density function $\rho(x) \geq 0$ defined on $\bar{\Omega}$

Her bir Voronoi Region (V_i) için center of mass tanımlayabiliriz (z_i^*).

CoM of Voronoi Regions:

$$z_i^* = \frac{\int_{V_i} dx \times \rho(x)}{\int_{V_i} dx \; \rho(x)} \qquad \text{for } i = 1, \ldots, k$$

Zaman üstte tanımladığımız Voronoi tesselation, altteki şarta uyması durumunda CVT'dir diyebiliriz. if and only if.

$$z_i = z_i^* \qquad \text{for } i = 1, \ldots, k$$

2 Generatorlerin arbitrary seçimleri genel olarak CVT durumunu vermez.

Dolayısıyla şöyle bir problemimiz olur:

given a region $\Omega \subset R^N$, $K \in 2^+$, $\rho(x) : x \in \bar{\Omega}$

determine a K-point CVT of Ω with respect to the given density function.

(In general CVTs of a given set are not uniquely defined)

Let:

$$\mathcal{K}\left(\{z_i\}_{i=1}^{k}, \{V_i\}_{i=1}^{k}\right) = \sum_{i=1}^{k} \int_{V_i} \rho(x) |x - z_i|^2 dx$$

$\{V_i\}_{i=1}^{k}$: tesselation of Ω

$\{z_i\}_{i=1}^{k}$: points in $\bar{\Omega}$

$\mathcal{K}$: energy, variance, cost.

★★ It was proved that a necessary condition for $\mathcal{K}$ to be minimized is that $\{z_i, V_i\}_{i=1}^{k}$ is a CVT of Ω

In practice, CVTs can only be approximately constructed, the energy is often used to monitor the quality of the results.

Methods for Determining CVTs

- To adjust the positions of generators to central of voronoi diagrams, some sequential methods need to be disuss
- Help from variations of K-means methods, and build iterative methods

K-means Method

- K-means is a clustering technique used in data analysis where data points are grouped into K clusters based on similarity, with each cluster represented by its centroid

Pseudo Code For K-Means

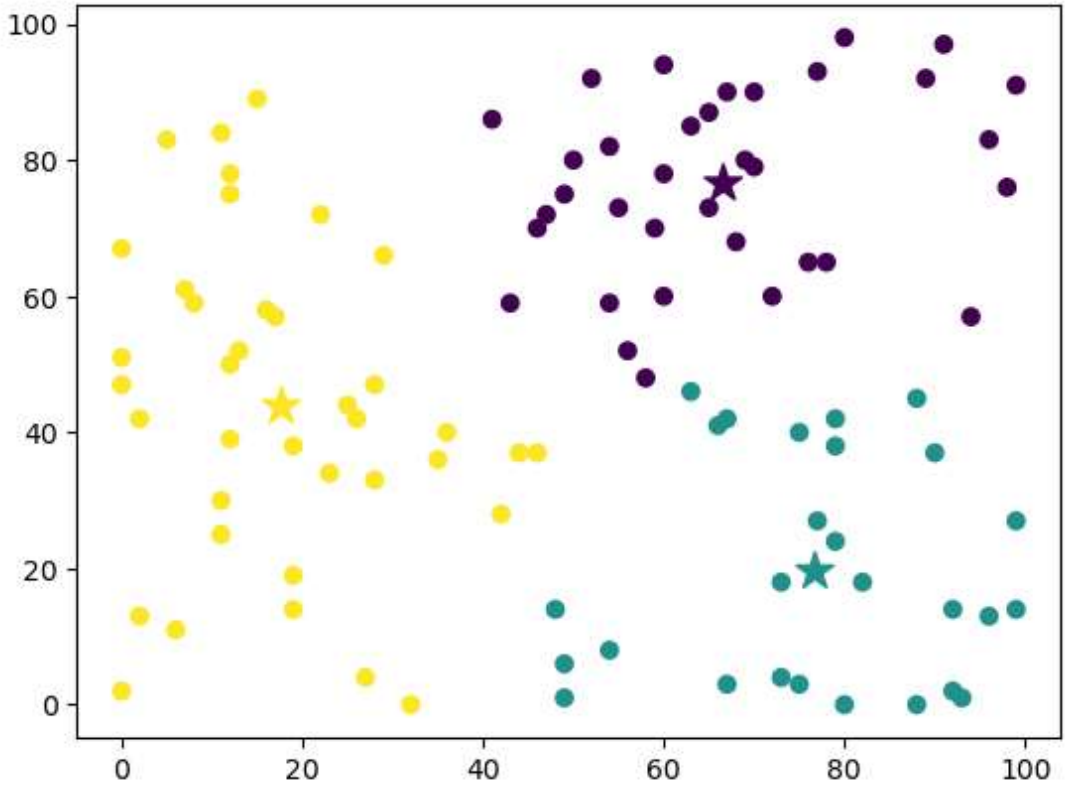
1. Initialize:
   - Choose the number of clusters K
   - Randomly initialize K cluster centroids (one for each cluster)

2. Repeat until convergence:
   a. Assign each data point to the nearest centroid:
      for each data point x_i:
         assign x_i to cluster j, where j is the index of the nearest centroid

   b. Update the centroids:
      for each cluster j:
         calculate the new centroid as the mean of all points assigned to cluster j

   c. Check for convergence:
      If the centroids do not change significantly, break the loop

3. Output the final clusters and centroids



Github link of the code: https://github.com/kaans7/Kmeans-from-Scratch/blob/main/Kmeans.ipynb

MacQueen's Method

- Divides the samples into k sets or clusters by taking means of sampling point

Algorithm 1. (*MacQueen's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, and a positive integer k,

0. Choose an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$ in Ω, e.g., by using a Monte Carlo method; set $j_i = 1$ for $i = 1, \ldots, k$;
1. Determine a point $\mathbf{y}$ in Ω at random, e.g., by a Monte Carlo method, according to the probability density function $\rho(\mathbf{x})$;
2. Find a $\mathbf{z}_{i^*}$ among $\{\mathbf{z}_i\}_{i=1}^k$ that is the closest to $\mathbf{y}$;
3. Set

$$\mathbf{z}_{i^*} \leftarrow \frac{j_{i^*}\mathbf{z}_{i^*} + \mathbf{y}}{j_{i^*} + 1} \quad \text{and} \quad j_{i^*} \leftarrow j_{i^*} + 1;$$

the new $\mathbf{z}_{i^*}$, along with the unchanged $\{\mathbf{z}_j\}, j \neq i^*$, form the new set of points $\{\mathbf{z}_i\}_{i=1}^k$;

4. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

- Any stage of the MacQueen's Method, each z is the mean of all sampling point that have been found to closest to that z (On the region)
- Therefore, this subset of all sampling points forms a cluster and the point z is the mean of its cluster. (CVTs condition)
- The almost sure convergence of the energy for the random MacQueen's method has been proved

Table 1
Maximum running times and average energies over four realizations of MacQueen's method vs. number of iterations for three density functions

No. of iterations	1		e^{-10x^2}		$e^{-20x^2} + (1/20)\sin^2(\pi x)$	
	Average energy	Maximum time (s)	Average energy	Maximum time (s)	Average energy	Maximum time (s)
0	22.514E−5	0.00	8.211E−5	0.00	9.323E−5	0.00
200,000	5.888E−5	2.93	2.285E−5	3.62	2.802E−5	3.83
800,000	5.658E−5	11.59	2.159E−5	13.94	2.702E−5	16.48
3,200,000	5.475E−5	46.34	2.059E−5	55.21	2.622E−5	63.44

Algorithm 1 (MacQueen's method); 128 generators on $(-1, 1)$.

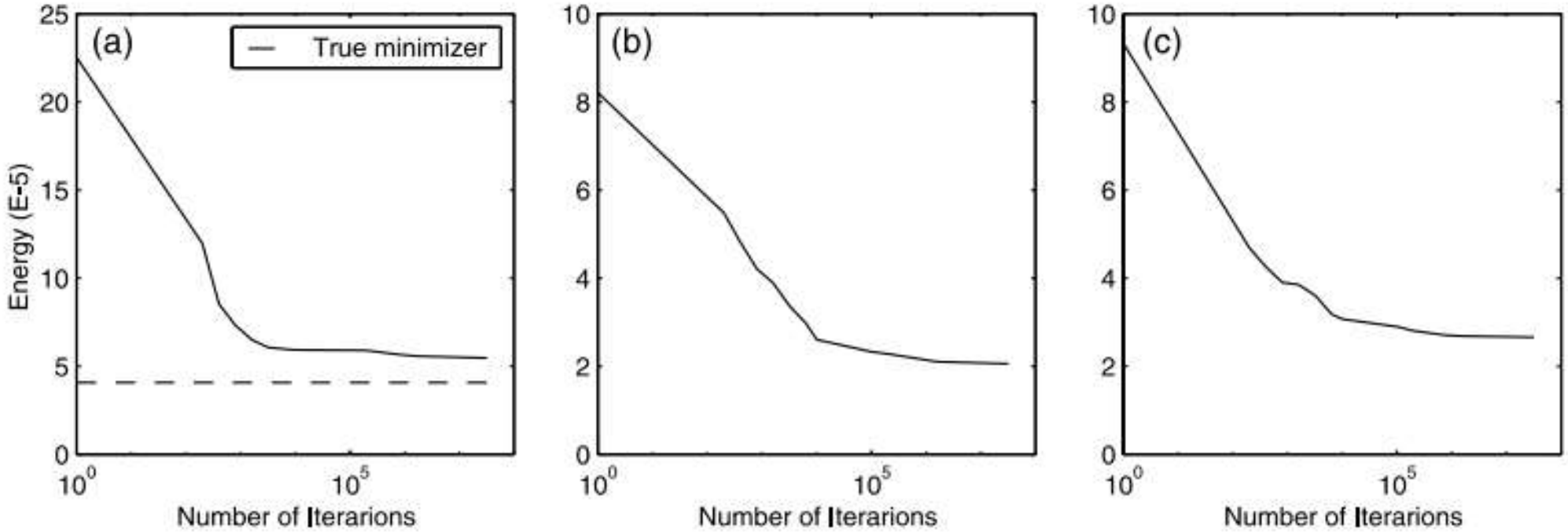


Fig. 1. Energy vs. number of iterations for MacQueen's method: (a) $\rho(x) = 1$; (b) $\rho(x) = \mathrm{e}^{-10x^2}$; (c) $\rho(x) = \mathrm{e}^{-20x^2} + 0.05\sin^2(\pi x)$.

Lloyd's Method

- Deterministic algorithm for determining CVTs

Algorithm 2. (*Lloyd's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, and a positive integer k,

0. Select an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$, e.g., by using a Monte Carlo method;
1. Construct the Voronoi sets $\{V_i\}_{i=1}^k$ associated with $\{\mathbf{z}_i\}_{i=1}^k$;
2. Determine the mass centroids of the Voronoi sets $\{V_i\}_{i=1}^k$; these centroids form the new set of points $\{\mathbf{z}_i\}_{i=1}^k$;
3. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

- In step 1: Voroni tesselations must be explicitly constructed
- In step 2: Numerical integraions over polyhedral domain must be employed to calculate centroids

- These steps increases the computational cost of each iterations, but also decreases the number of iterations

- Therefore, Lloyd's method requires fewer iterations than MacQueen's method, but on the MacQueen's method, consturction of voronoi tesselations not required

- Also, if a completely probabilistic procedure such as the rejection method is used to determine random sampling points, then MacQueen's method also does not involve any numerical integration

Quasi-Random Sequences

- Low Discrepancy Sequences
  - Idea: To generate a theoretically infinite sequence of numbers such that every subsequence {1,...,m} the subsequence has low discrepancy
  - This allows to increase the accuracy by just adding more and more functional evaluations

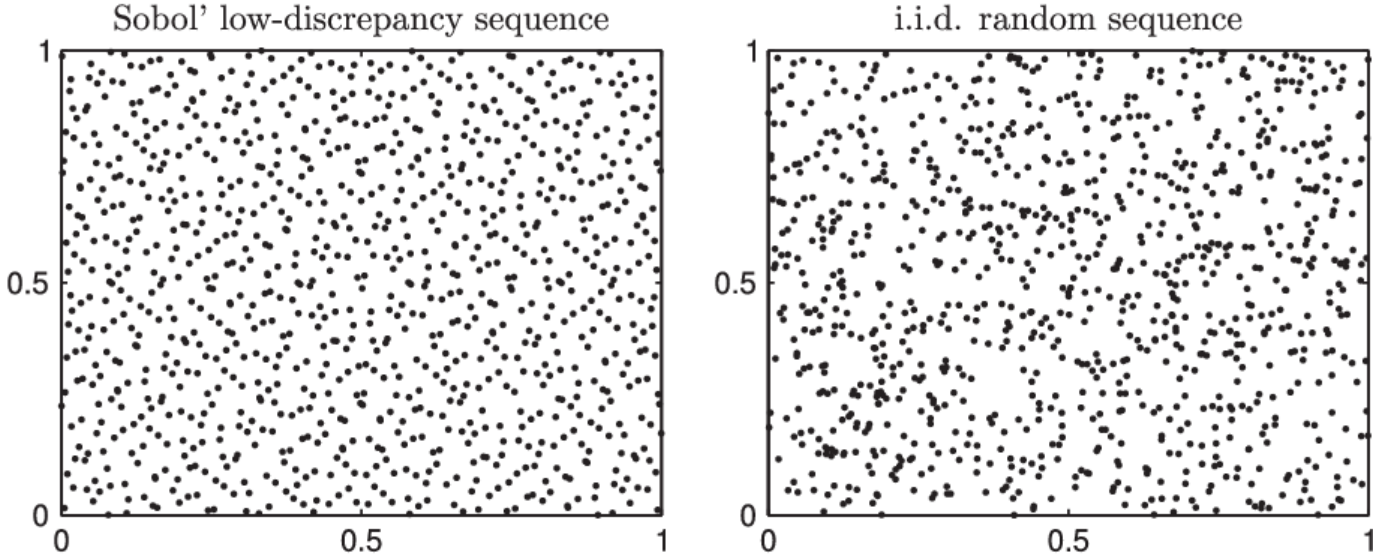


Fig. 1. Comparison between Sobol' and i.i.d. uniform samplings of the 2-dimensional unit cube.

Koksma-Hlowka Inequality

- Provides an upperbound on the error incurred when approximating definite integrals by numerical methods

$$\left| \frac{1}{N} \sum_{j=1}^{N} f(x_j) - \int_0^1 f(x)\,dx \right| \leq D(\{x_j\}_{j=1}^{N})\, V(f)$$

$$D(\{x_j\}_{j=1}^{N}) = \sup_{0<t<1} \left\{ \left| N^{-1} \sum_{j=1}^{N} \chi_{[0,t]}(x_j) - t \right| \right\} : \text{Discrepancy of the points } 0 \leq x_j \leq 1$$

$$V(f) = \sup_{0=y_0<y_1<\cdots<y_n=1} \left\{ \sum_{k=1}^{n} |f(y_k) - f(y_{k-1})| \right\} : \text{The total Variation of the function}$$

- The discrepancy measures how well the distribution of points in P reflects the behavior of the function f
- The Koksma-Hlowka Inequality is useful in analyzing the accuracy of numerical integraiton methods and understanding the relationship between quality of point set and the error in the approximation of integrals
- The sequence is uniformly distributed on [0,1) **if and only** if Discrepancy --> 0 as N--> ∞ . However limit to how well distributed any sequence can be

Van der Corput Sequence

- J.G. Van der Corput introduce to generate 1 dimensional low-discrepancy sequences on [0,1) in 1935
- Considered to be among the best distributed over [0,1) and no other infinitely generated sequences can have discrepancy of smaller order of magnitude than it.

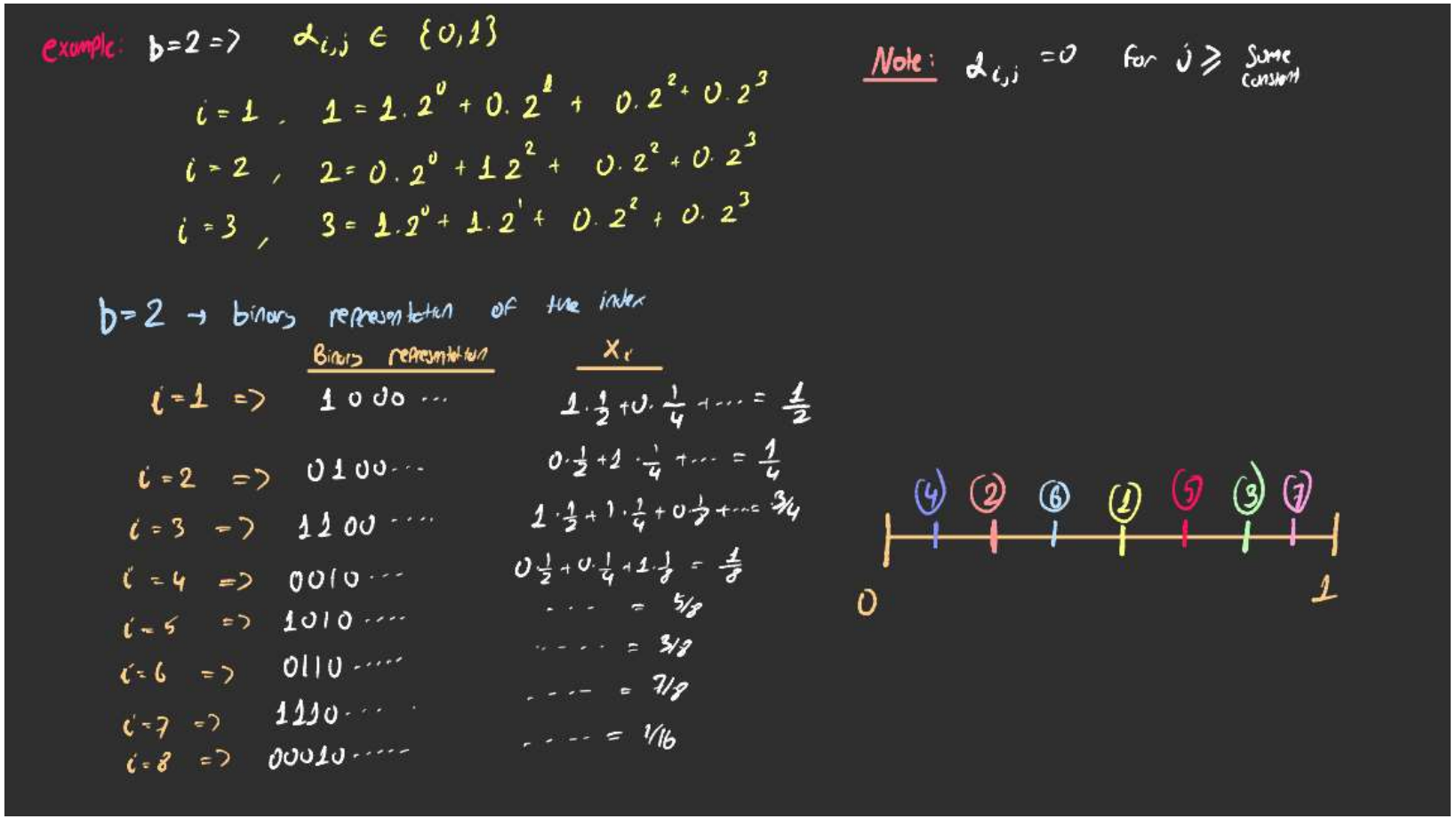
Definition:

Let $b \in \mathbb{N}$, $b > 1$. The Van der Corput sequence $\{x_i\}_{i \in \mathbb{N}} \subset [0,1]$ with base b is defined by

$$x_i = \sum_{j=1}^{\infty} a_{ij} b^{-j}$$

where $a_{i,j}$ are such that $i = \sum_{j=1}^{\infty} a_{ij} b^{j-1}$ with $a_{ij} \in \{0, \dots, b-1\}$

Example for Generating Van der Corput Sequence



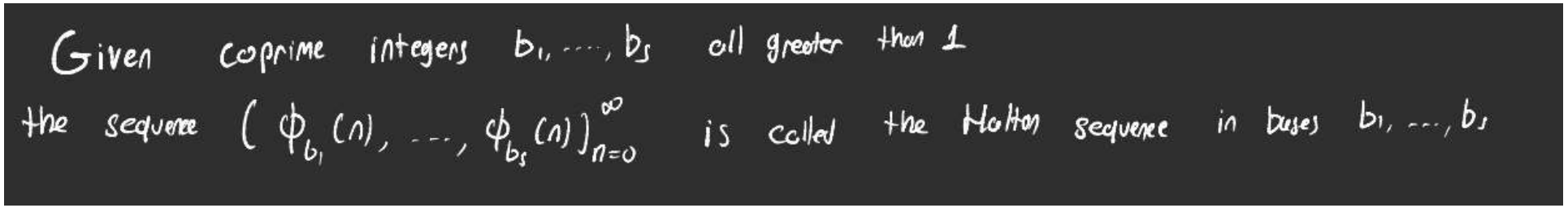
- Binary Representation; b=2
- Works on only in 1 dimension
- Discrepancy:

$$\text{Discrepancy}: O\left(\frac{\log(n)}{n}\right)$$

- It is and infinite low-discrepancy sequence. We can improve the accuracy of the integration by just adding new evaluation points, re-using the previous result

Halton Sequence

- Generalization of the Van der Corput sequence to higher dimensions is obtained by John Halton in 1960



- Key point: Generating Van der Corput sequences for each dimensions with choosing some bases. Bases must be relatively prime with each other
- How to determine the bases?

- Determinin bases of Halton Sequence
  - Discrepancy of the first N elements of the Halton Sequence:

$$C(b_1, \dots, b_s)\frac{(\log N)^s}{N} + O\left(\frac{(\log N)^{s-1}}{N}\right)$$

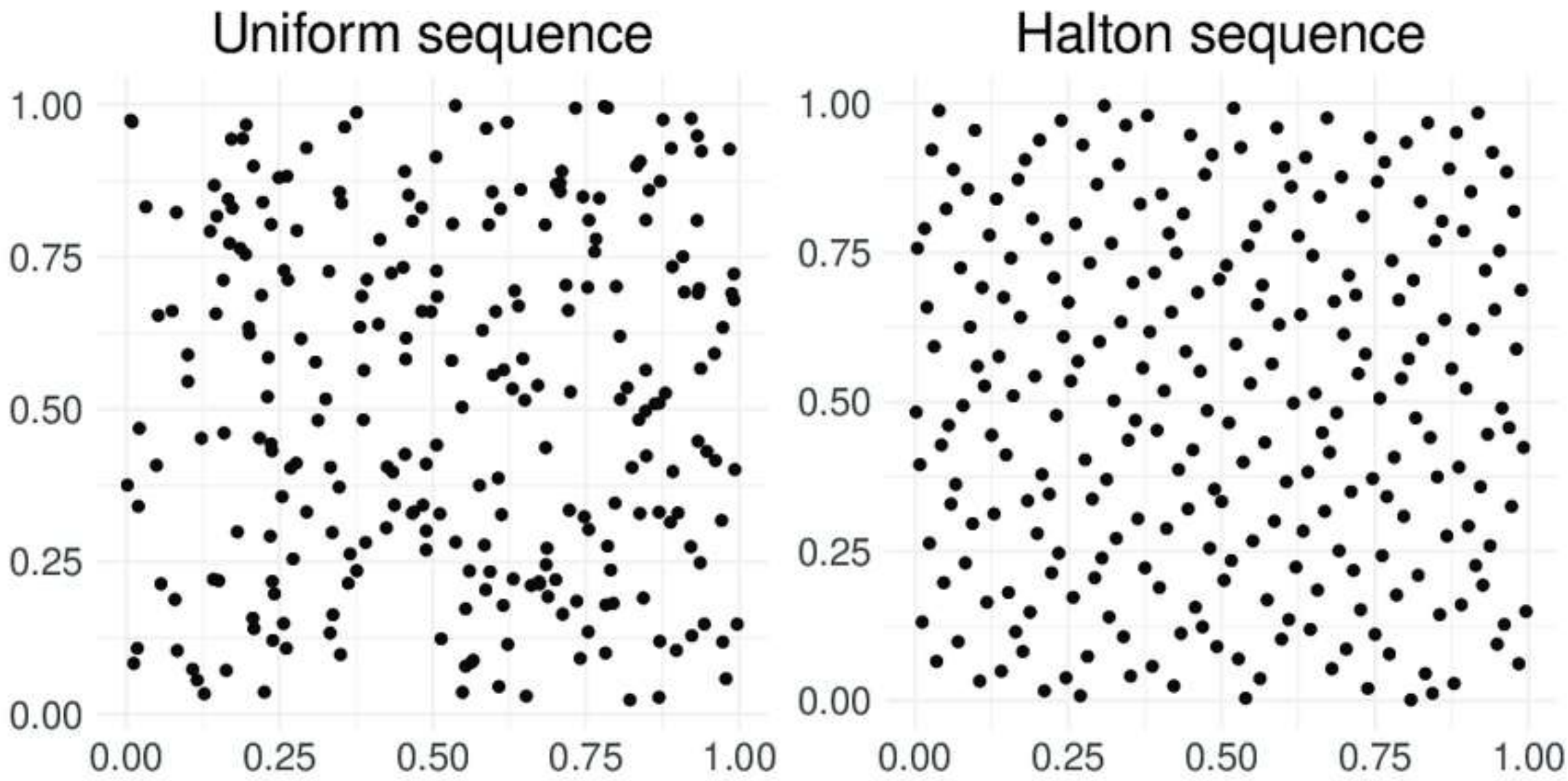
For some constant $c(b_1, \dots, b_s) > 0$

  - We shall call an infinite sequence w a low-discrepancy if $D_N(w) = O((\log N)^s/N)$

  - To achieve this how small $c(b_1, \dots, b_s) > 0$ can be found
  - This constant depends very strongly on the dimension s
  - The minimal value can be obtained if we choose bases to be the first s prime numbers. First s prime numbers also relatively prime with each other.
  - Even in this case c grows very fast to infinity as s increases
    - This problem solved by Atanassov

$$C(b_1, \dots, b_s) = \frac{1}{s!}\prod_{i=1}^{s}\frac{b_i - 1}{\log b_i}$$

When $b_1, \dots, b_s$ are the first s prime numbers
$c(b_1, \dots, b_s) \to 0$ as $s \to \infty$


Uniform sequence
Halton sequence
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00

Hammersley Sequence

- Presented by John Hammersley who was advisor of John Halton
- This sequence is also obtained from Van der Corput sequence, and very similar to Halton sequence

$$\Phi_p(k) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \cdots + \frac{a_r}{p^{r+1}}$$

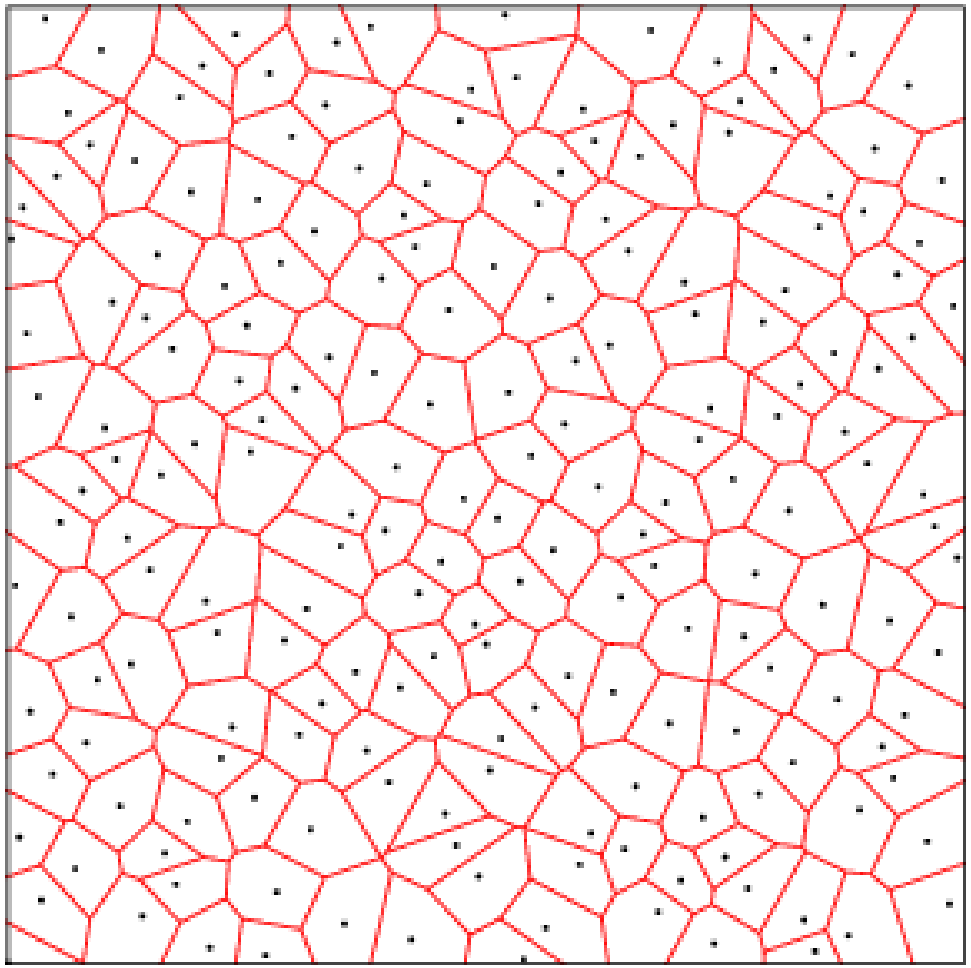
if $p=2$, the sequence of $\Phi_2(k)$, for $k = 0, 1, 2, \ldots$, is called the Van der Corput sequence

Let d be the dimension of the space to be sampled. Any sequence $P_1, P_2, \ldots, P_{d-1}$ of prime numbers defines a sequence $\Phi_{P_1}, \Phi_{P_2}, \ldots, \Phi_{P_{d-1}}$ of functions, whose corresponding k-th d-dimensional Hammersley point is

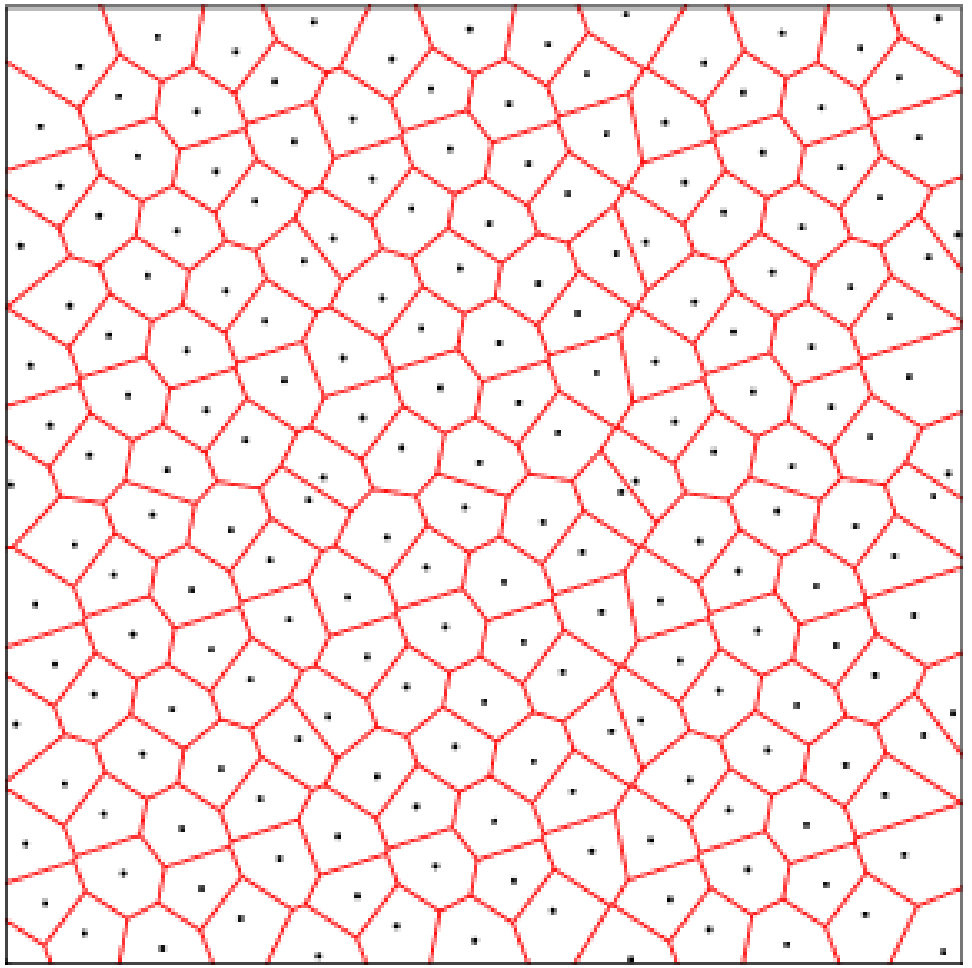
$$\left(\frac{k}{n}, \Phi_{P_1}(k), \Phi_{P_2}(k), \ldots, \Phi_{P_{d-1}}(k) \right) \quad \text{for} \quad k = 0, 1, 2, \ldots, n-1$$

$P_1, P_2, \ldots, P_{d-1}$ relatively prime.

Halton Sampling vs. Hammersley Sampling



(a) 196 Halton points



(b) 196 Hammersley points

Sobol Sequence

- **Primitive Polynomial**: An irreducible polynomial over a finite field such that the powers of its root generate all non-zero elements of that field
- Let's choose a primitive polynomal of some degree s

$$X^{s_j} + a_{1,j} X^{s_j-1} + a_{2,j} X^{s_j-2} + \cdots + a_{s_j-1,j} X + 1$$

- Where coefficients $a_{1,j}, \ldots, a_{s-1,j}$ are either 0 or 1. Then:

$$m_{k,j} := 2a_{1,j} m_{k-1,j} \oplus 2^2 a_{2,j} m_{k-2,j} \oplus \cdots \oplus 2^{s_j-1} a_{s_j-1,j} m_{k-s_j+1,j} \oplus 2^{s_j} m_{k-s_j,j} \oplus m_{k-s_j,j}$$

- Where $\oplus$ is the Bitwise XOR operator

- The initial values of $m_{1,j}, \ldots, m_{s,j}$ can be choosen freely. Then we can define direction numbers:

$$v_{k,j} := \frac{m_{k,j}}{2^k}$$

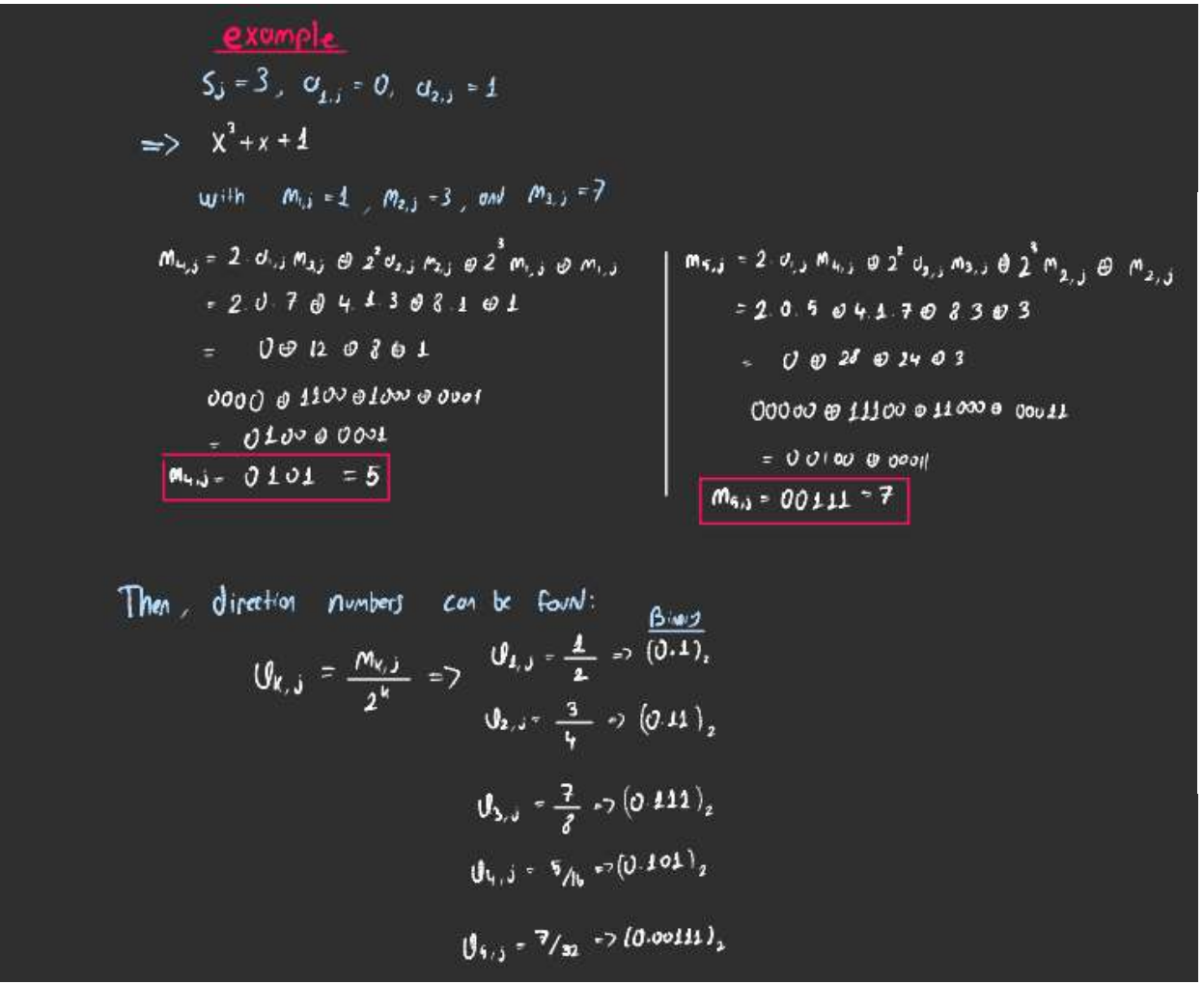
- Finally, Sobol Sequence is given by

$$X_{i,j} := i_1 v_{1,j} \oplus i_2 v_{2,j} \oplus \cdots$$

Where i_k is the k^{th} digit from the right when i is written in binary $i = (\cdots i_3 i_2 i_1)_2$

- Sobol Sequence can be enhanced by implementing grey code

Example for Generating Sobol Sequence



$$
\begin{aligned}
x_{0,j} &= 0, \\
x_{1,j} &= (0.1)_2 = 0.5, \\
x_{2,j} &= (0.11)_2 = 0.75, \\
x_{3,j} &= (0.1)_2 \oplus (0.11)_2 = (0.01)_2 = 0.25, \\
x_{4,j} &= (0.111)_2 = 0.875, \\
x_{5,j} &= (0.1)_2 \oplus (0.111)_2 = (0.011)_2 = 0.375,
\end{aligned}
$$

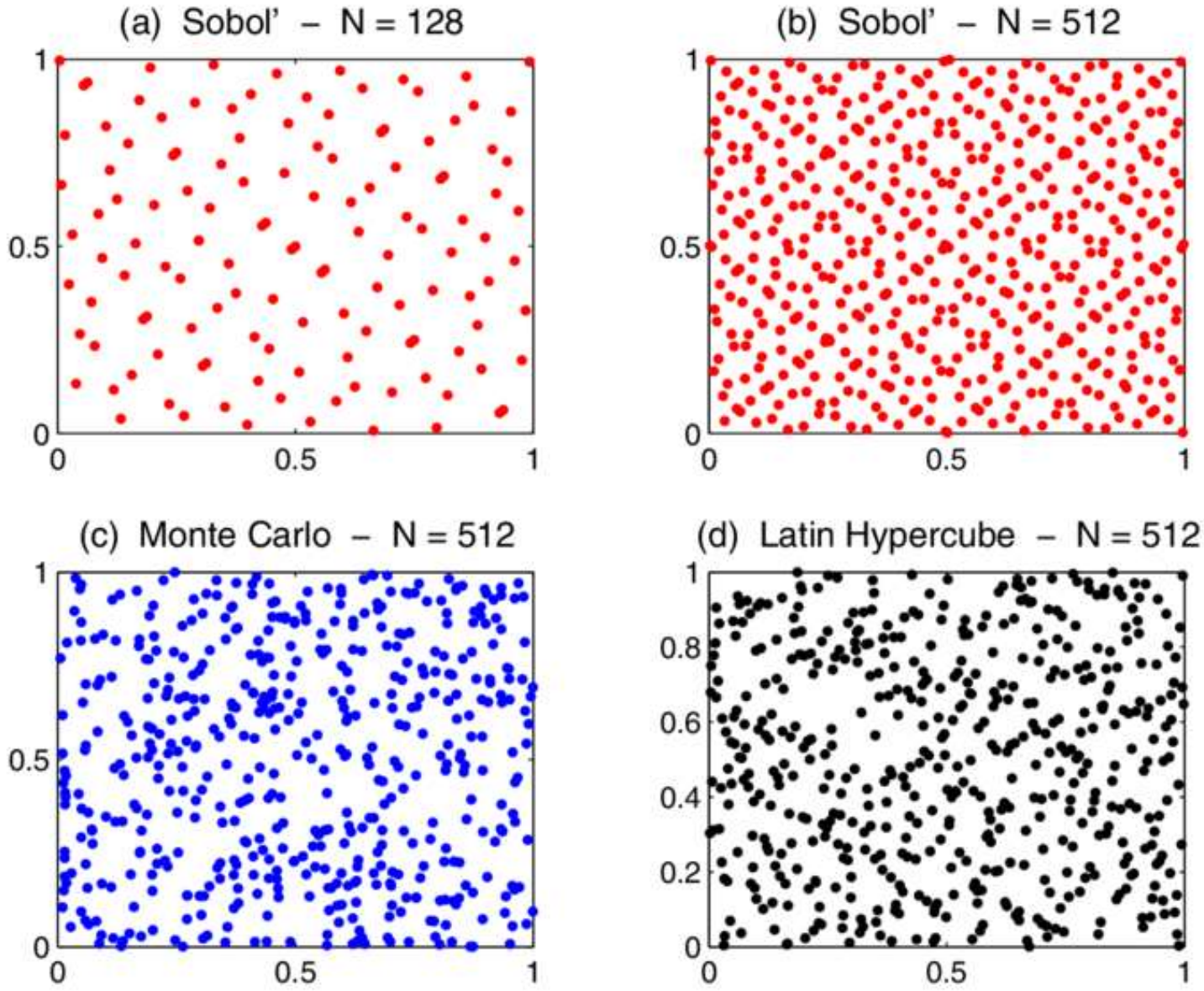
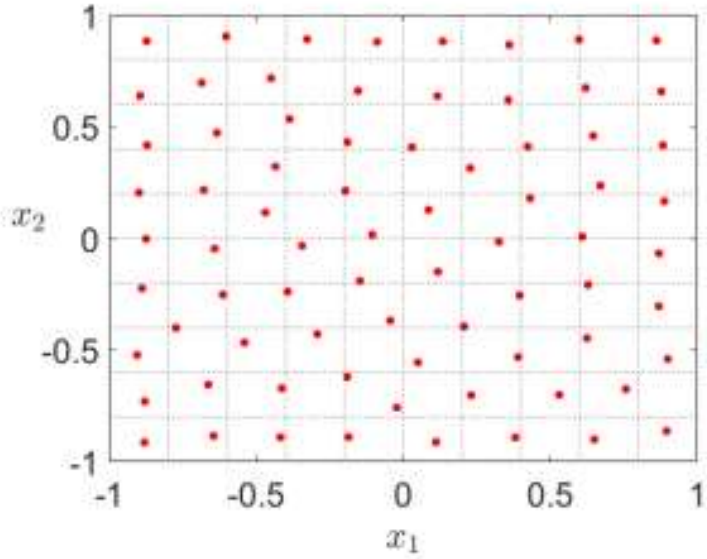
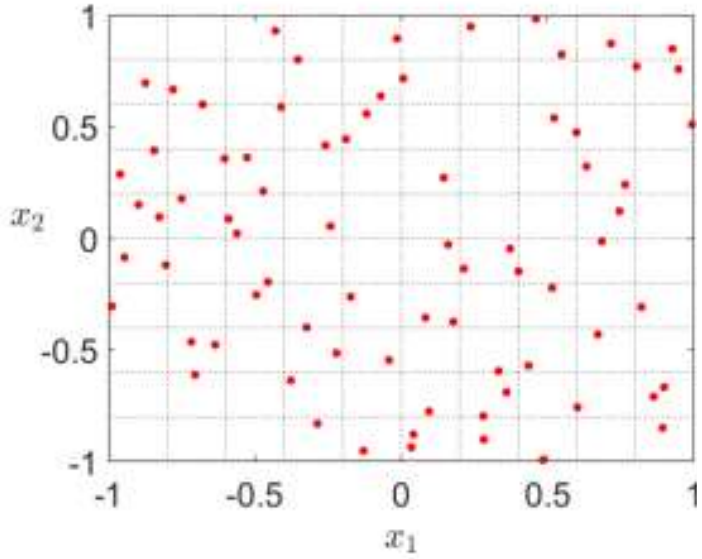


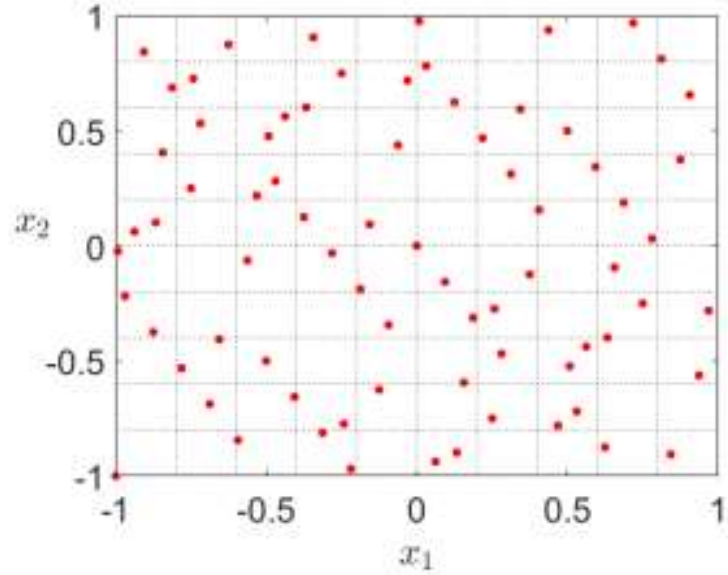
Fig. 2. Space-filling process of $[0, 1]^2$ using a two-dimensional Sobol' sequence, compared to MCS and LH



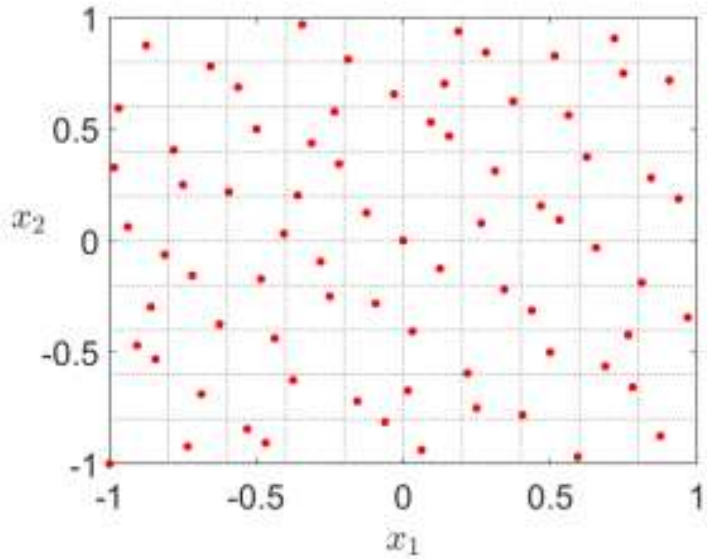
(a) Centroidal Voronoi Tesselation



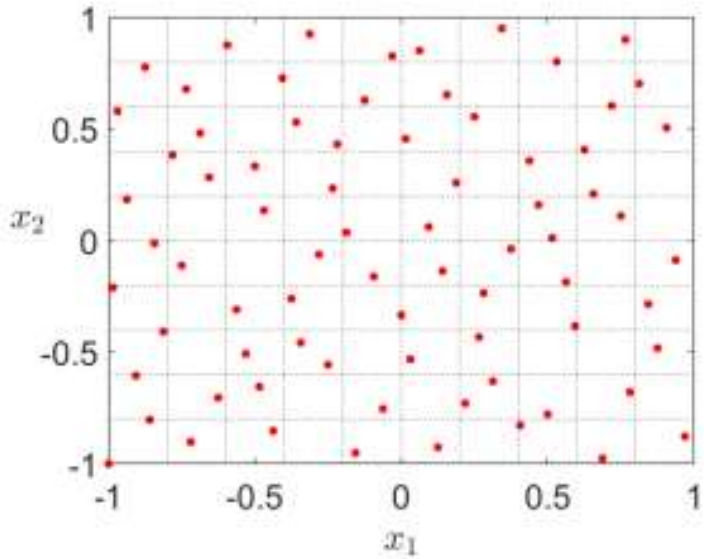
(b) Maximin Latin Hypercube



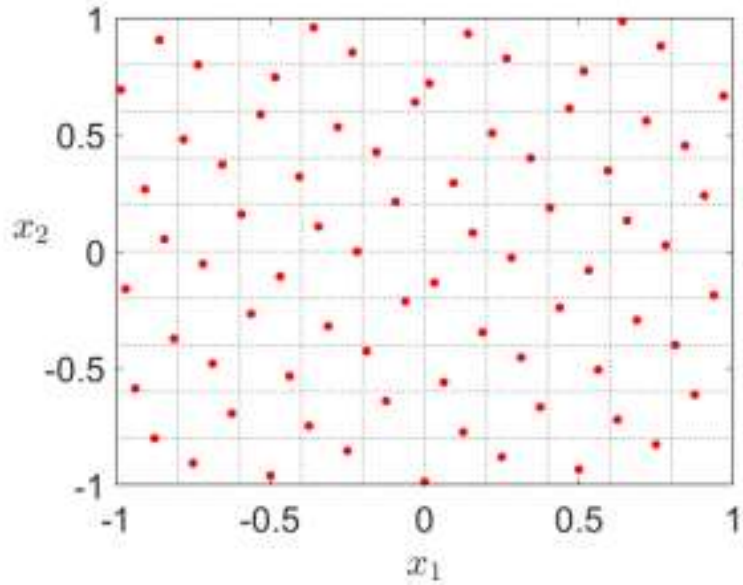
(f) Faure Samples



(c) Sobol Samples



(d) Halton Samples



(e) Hammersley Samples

Applications of Quasi-random Sequence Sampling

Codes: https://github.com/kaans7/Quasi-random-Sequence-Sampling/blob/main/Sequences.ipynb

- Van der Corput Sequence:

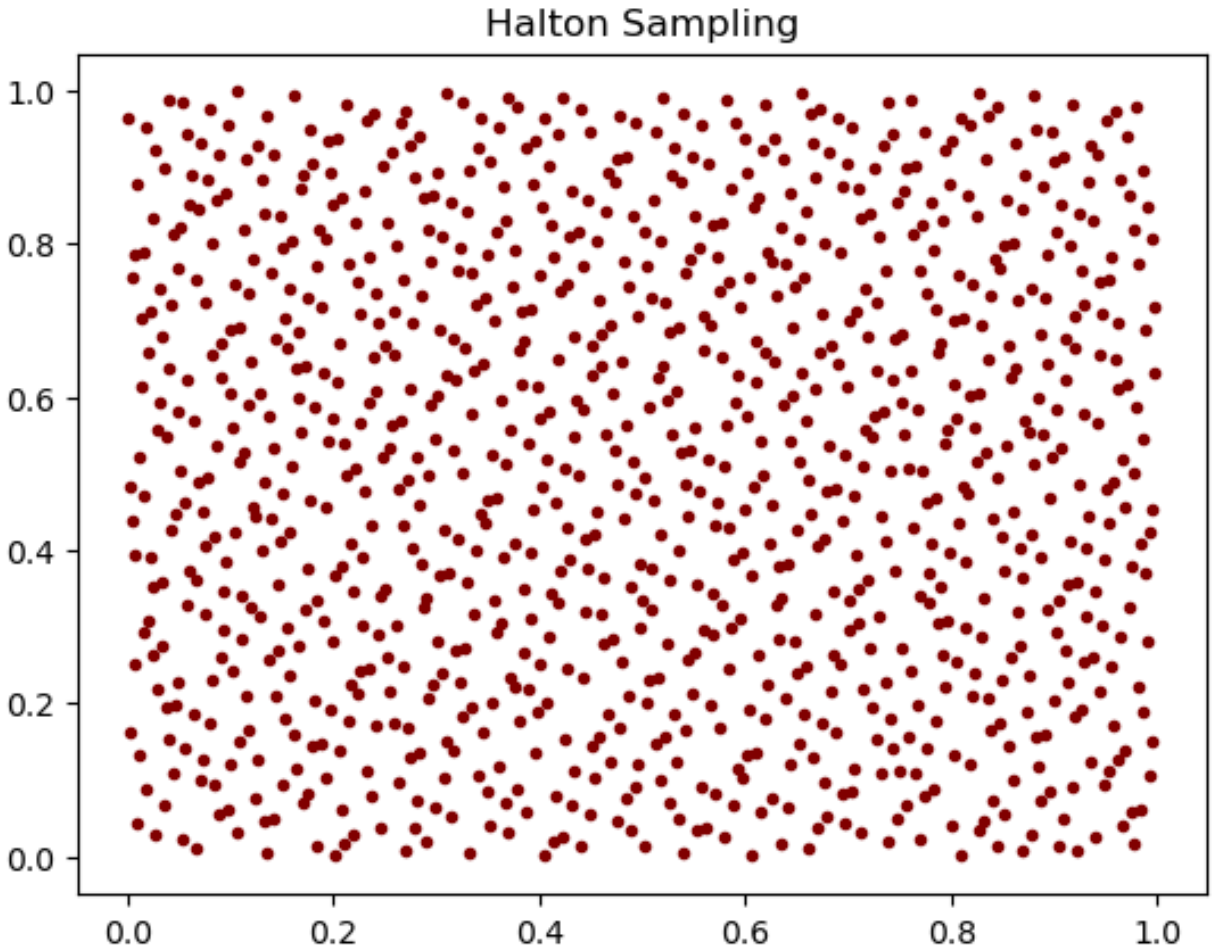
```python
def van_der_corput(self, n_sample, base=2):

    vdc_sequence = []
    for i in range(1,n_sample+1):
        n_th_number = 0.
        denominator=1.

        while i > 0:
            # Binary representation of i:
        # Firstly divide it with 2. Find the remainder and the result of the division and set it as new i
        # Then repeat the same process until i=1 and the division gives i=0. This gives the elements of
        # binary representation for each iteration. For instance, i=7 gives 1110 for remainders and 7,3,1,0 for i values
            i, remainder = divmod(i, base)
            denominator *= base
            n_th_number += remainder / denominator # x_i = a_ij * base^-1

        vdc_sequence.append(n_th_number)

    return vdc_sequence
```

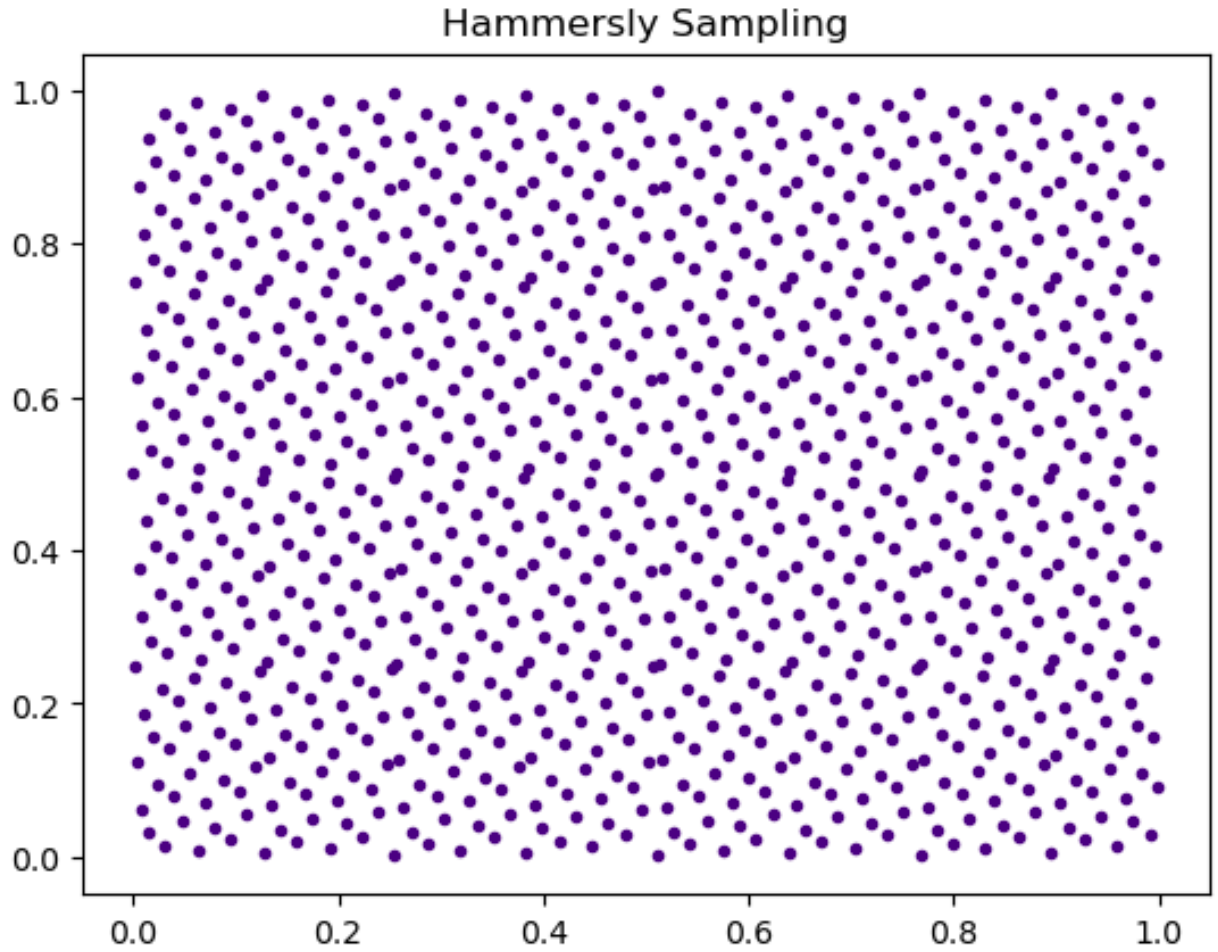
- Halton Sequence:
  - Halton sequence can be obtained with the help of Van der Corput sequences. Firstly, determining the first n prime numbers, then creating Van der Corput sequences with bases respect to these prime numbers. Lastly, implementing each Van der Corput sequences on different dimensions to get Halton sequence.

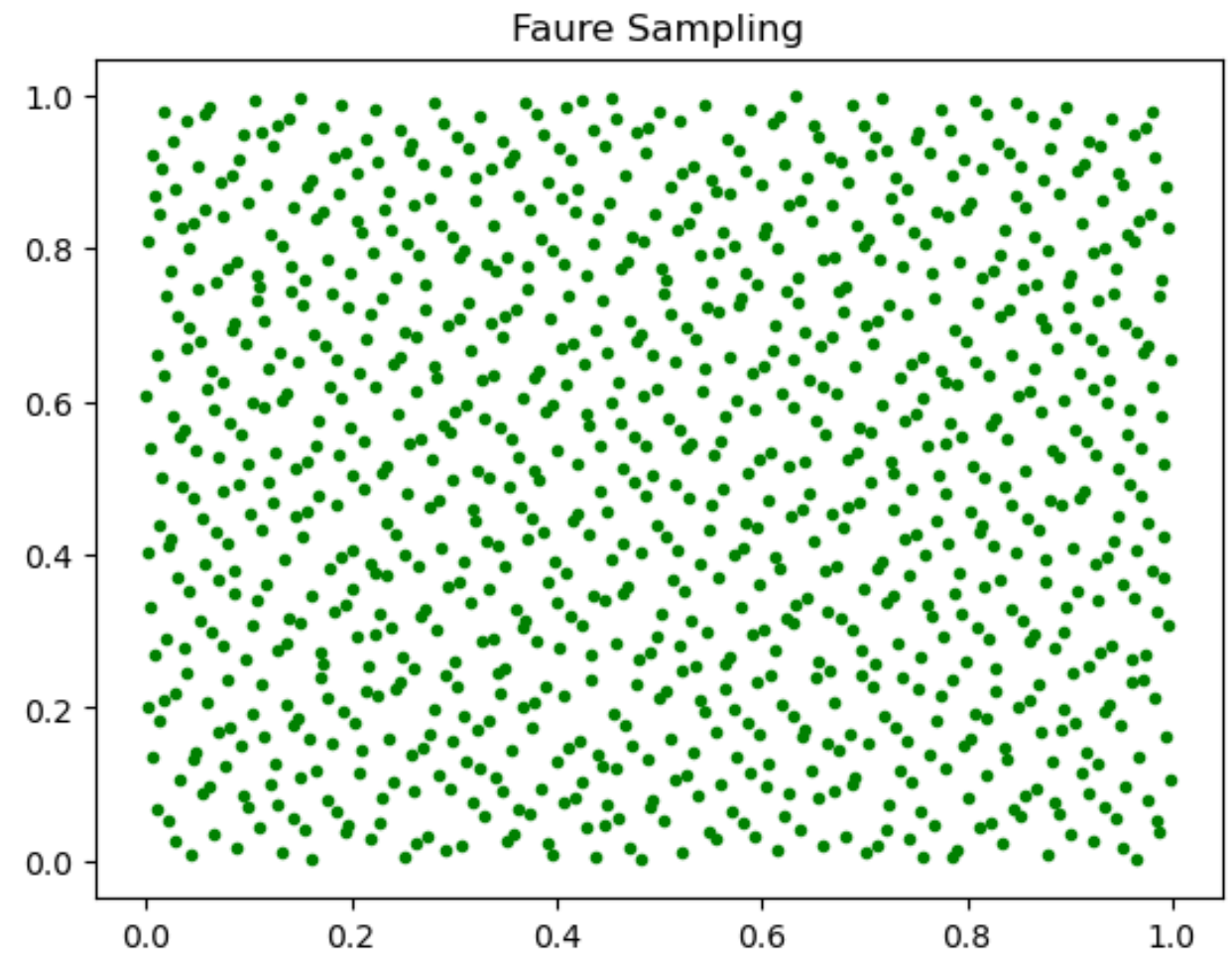
```
def halton(self):

    bases=self.first_n_prime_number(self.dimension)
    halton_sequence=np.zeros((self.n_sample,self.dimension))
    for i in range(self.dimension):
        halton_sequence[:,i] = self.van_der_corput(n_sample=self.n_sample,base=bases[i])
    return halton_sequence
```

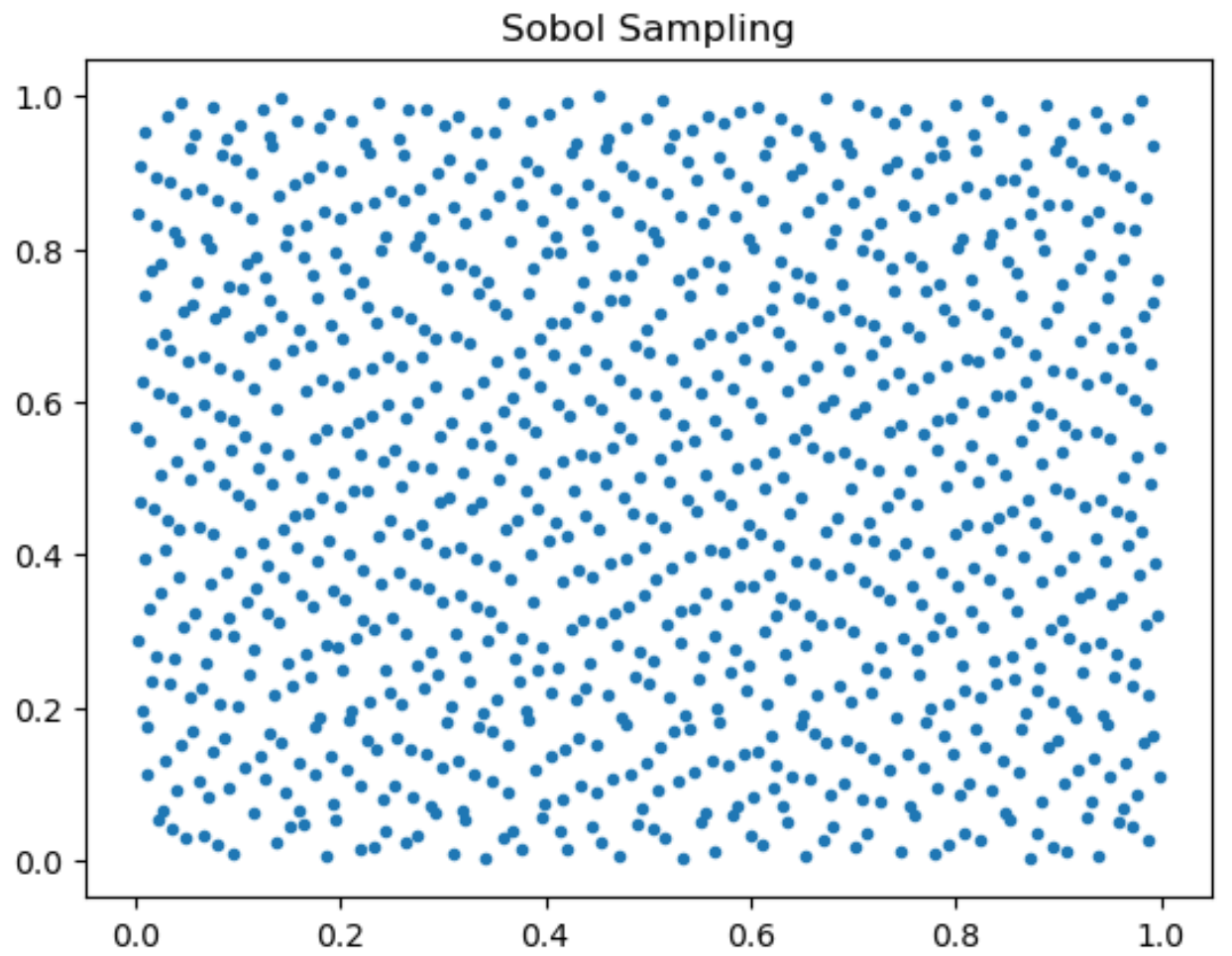
- Hammersly Sequence
  - Hammersly sequence can be created with specialization of the first dimension of the Halton sequence.
  - This approach involves splitting an axis into n equal parts and then normalizing each point based on the number of points in its segment.

```
def hammersly(self):

    bases=self.first_n_prime_number(self.dimension-1)
    hammersly_sequence=np.zeros((self.n_sample,self.dimension))

    hammersly_sequence[:,0]=np.arange(self.n_sample)/self.n_sample

    for i in range(1,self.dimension):
        hammersly_sequence[:,i] = self.van_der_corput(n_sample=self.n_sample,base=bases[i-1])
    return hammersly_sequence
```


Halton Sampling
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Hammersly Sampling
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Faure Sampling
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Sobol Sampling
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

Monte Carlo Integration

Monte Carlo Integration

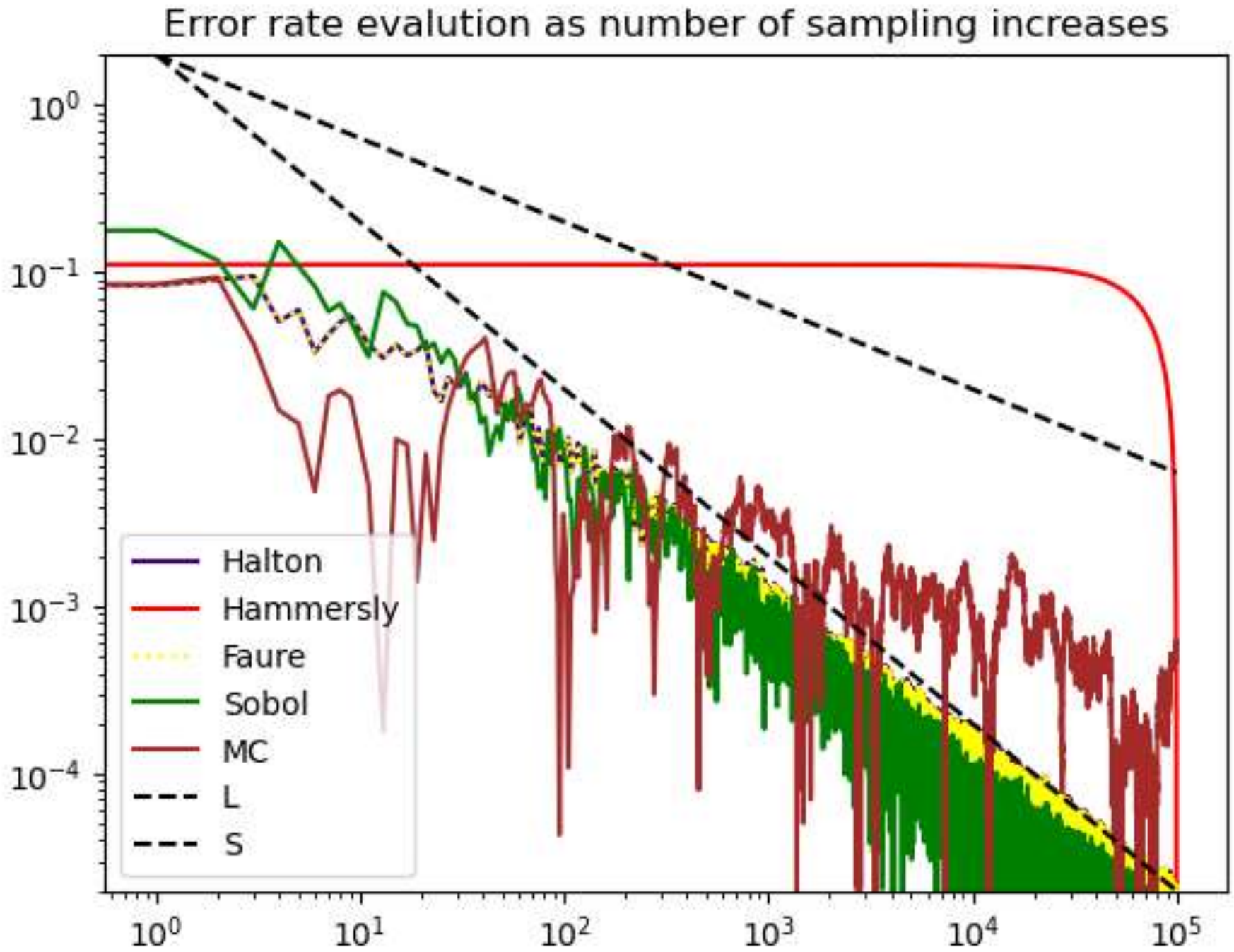
Integral of a function can be obtained with the monte carlo method. To find the result, sample points is neceserray which can be obtained by randomly or from a sequence sampling

$$f(x, y) = -x^2 y^2$$

$$F = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i)$$
$$x_i \in [a, b]$$

Sampling range: [0,1] on each dimension
Analytical Solution: -1/9
N=100000 points are used for sampling
Results:

```
Halton:  -0.1110912056817631 9 Error:  -1.9905429347918946e-05
Hammersly:  -0.111109379960684 7 Error:  -1.7311504263983757e-06
Faure:  -0.1110912056817631 9 Error:  -1.9905429347918946e-05
Sobol:  -0.1111068630614084 6 Error:  -4.2480497026431 42e-06
Random:  -0.1116823558039626 Error:  0.0005771244692851507
```


Error rate evalution as number of sampling increases
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^0
10^1
10^2
10^3
10^4
10^5
Halton
Hammersly
Faure
Sobol
MC
L
S

Numerical Quadrature

- Process of computing approximations to integral

$$\int_{\Gamma} f(\boldsymbol{x})\omega(\boldsymbol{x})\mathrm{d}x \approx \sum_{j=1}^{n} f(\boldsymbol{x}_j)w_j,$$

- Gaussian quadrature rules satisfy many of the desirable properties in one dimension.

```
def gauss_leg(m,n,a,b):
    h=(b-a)/n
    slope=h/2.
    vx=np.zeros(m*n,dtype=float)
    weights=np.zeros(m*n,dtype=float)
    r,w=gauss(m)
    for k in range(m):
        vx[k::m]=a+slope*(r[k]+1.)+h*np.arange(n)
        weights[k::m]=w[k]*slope
    return(vx,weights)

(x,w)=gauss_leg(8,5,0.,5.)
def f(x):
    return 1/(1-x**2)
s1=(w*f(x)).sum()
```

```
Analytic solution => arctanh(5) 1.373400766945016
Numeric solution with gauss-legendre = 1.373400766927361
Error:  1.7654988582194164e-11
```

2D Gauss – Legendre Quadrature

- Full Grid Desing

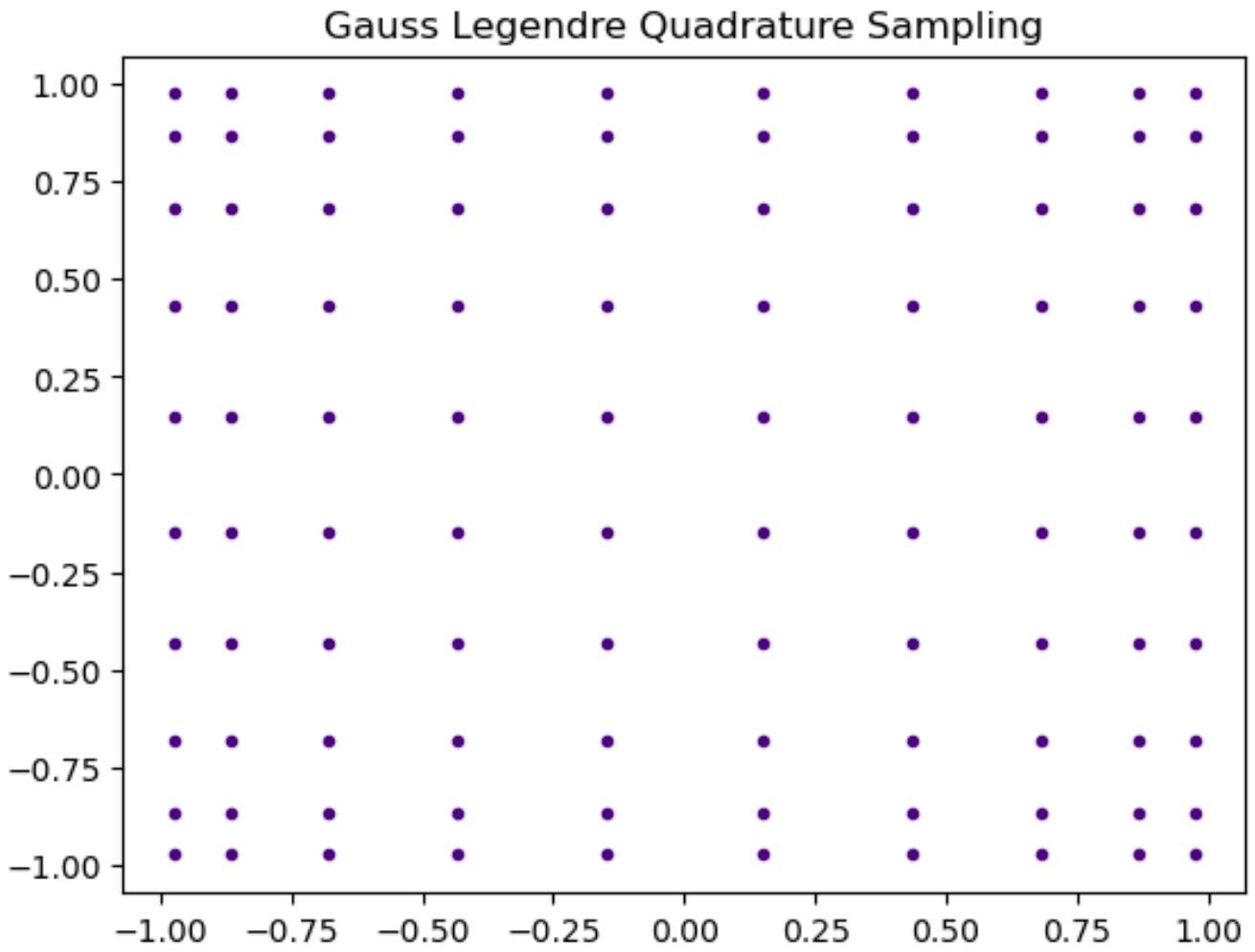
- Integration:

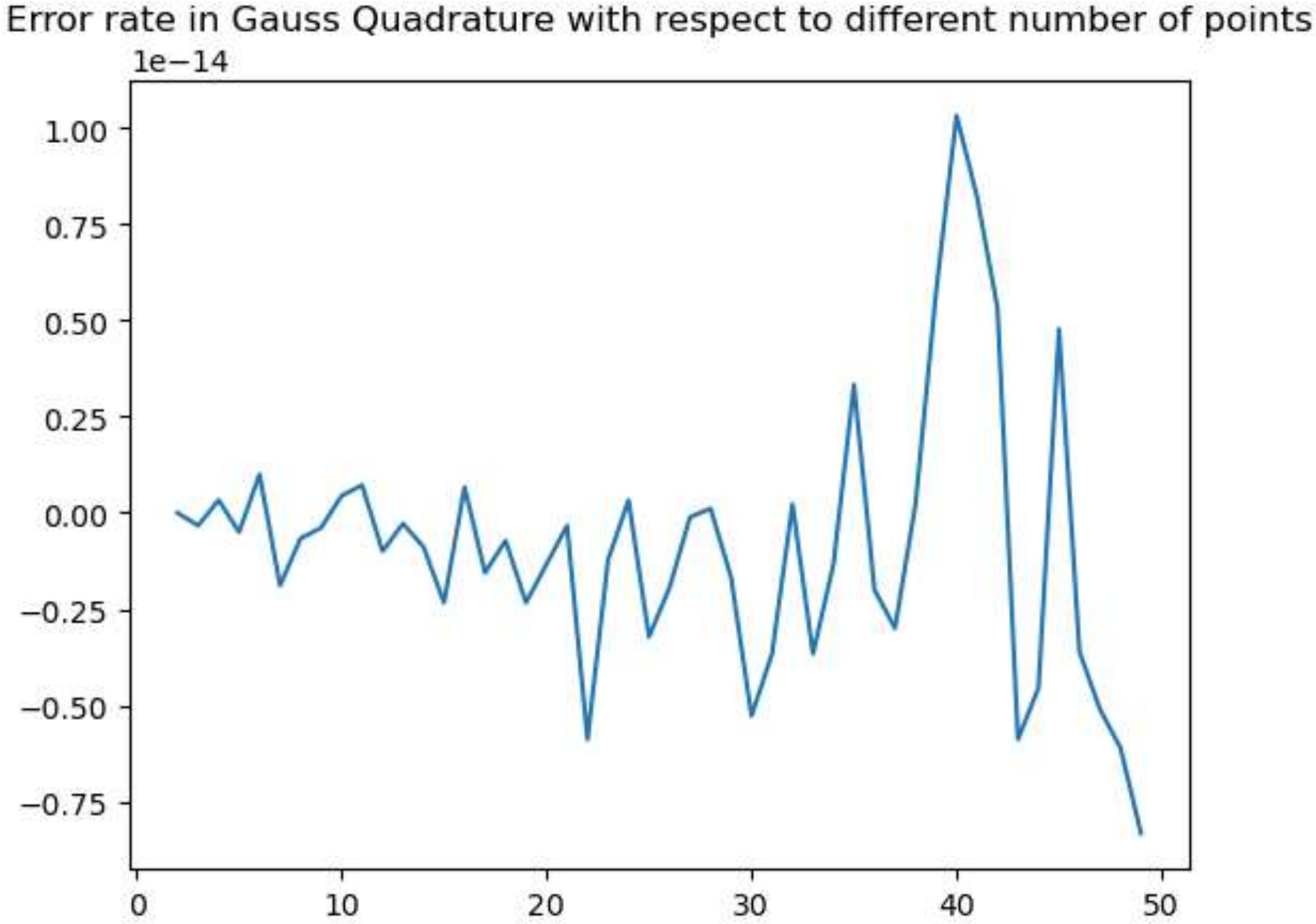
$$f(x, y) = -x^2 y^2$$

- Analytical Result: -4/9

```
error=[]
analytic=-4/9
for i in range(2,50):
    w,gauss=gauss2D(i)
    legendre=np.sum(f(gauss[:,0],gauss[:,1])*w)
    error.append(legendre-analytic)

print('Error:',legendre-analytic)
```

```
Error: -8.326672684688674e-15
```




Error rate in Gauss Quadrature with respect to different number of points
1e−14
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
10
20
30
40
50

What happens in higher dimensions?

- Possible reasons why gauss quadrature does not work well in higher dimensions!

- Question: Can quadrature methods be implemented in higher dimensions that perform as good and efficient as quasi monte carlo methods?
  - The problem is not just how quadrature points distributed well in n-dimensional space
  - Finding optimal method for numerical integration in higher dimensions!
  - Rather than randomization, how a function can be interpolated in high dimensions?
  - https://arxiv.org/pdf/1804.06501.pdf

My Implementation

- I have built a model for Viscous Burger's Equation

Definition of PDE:

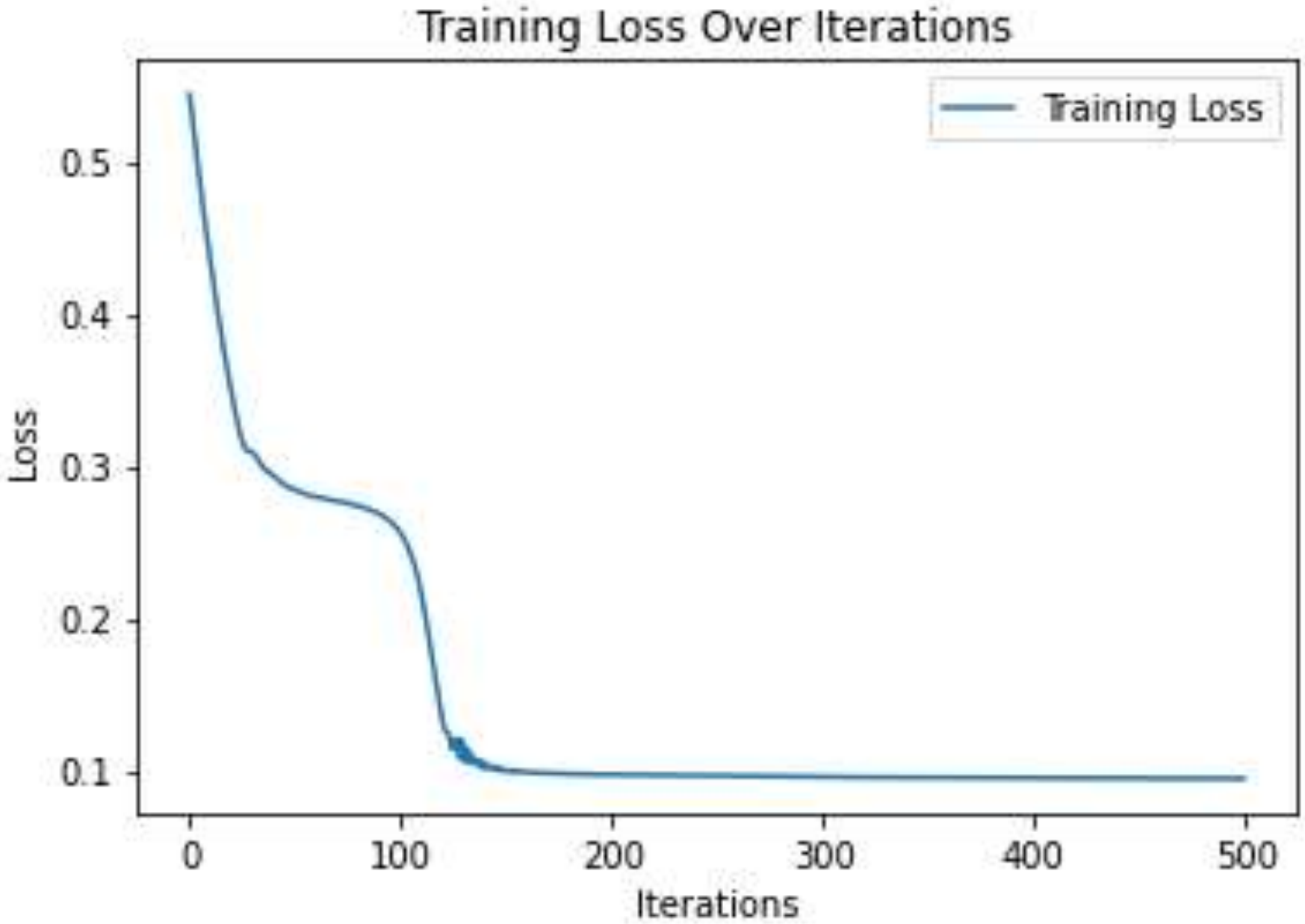
```
def net_f(self, x, t):
    lambda_1 = self.lambda_1
    lambda_2 = self.lambda_2

    u = self.net_u(x, t)

    du_dt = torch.autograd.grad(u, t, torch.ones_like(u), create_graph=True, retain_graph=True)[0]
    du_dx = torch.autograd.grad(u, x, torch.ones_like(u), create_graph=True, retain_graph=True)[0]
    d2u_dx2 = torch.autograd.grad(du_dx, x, torch.ones_like(du_dx), create_graph=True)[0]

    f = du_dt + u * du_dx -  self.v * d2u_dx2
    return f
```

Physical Information through loss:

```
def loss(self):
    u = self.net_u(self.x, self.t_0)

    loss1 = (torch.square(torch.abs(torch.squeeze(u) - (torch.sin(torch.pi * self.x)))))
    u = self.net_u(self.x_1, self.t)
    loss2 = (torch.square(torch.squeeze(u)))
    u = self.net_u(self.x_1_neg, self.t)
    loss3 = (torch.square(torch.squeeze(u)))
    f_pred = self.net_f(self.x, self.t)
    loss4 = torch.mean(torch.square(f_pred))

    loss = torch.mean(loss1 + loss2 + loss3) + loss4
    loss.backward(retain_graph=True)

    return loss
```

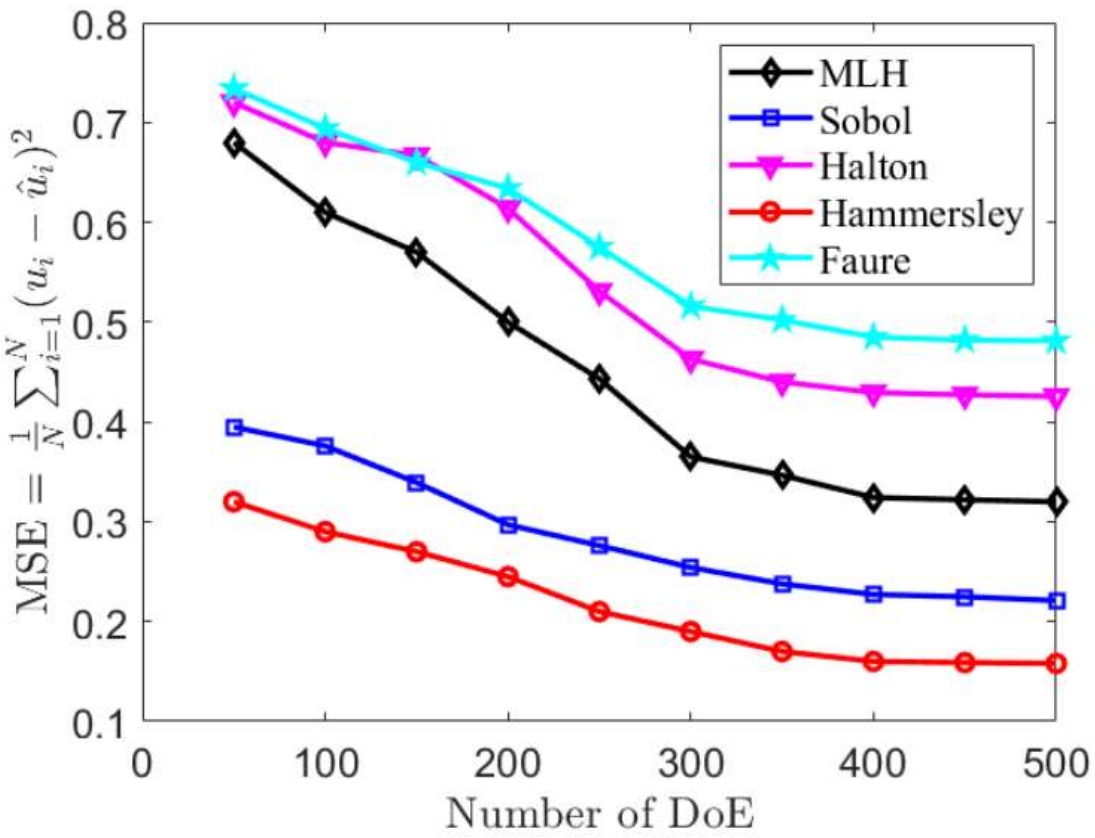
- I had problems on obtaining analytical result and comparing it with my model that trained with different quasi-random sampling methods

- However, the sampling that I used did not show differences on training process.


Training Loss Over Iterations
Training Loss
Loss
0.5
0.4
0.3
0.2
0.1
0
100
200
300
400
500
Iterations

Results form Paper

- When we look at the middle values, we see that FD, CCD, and CVT are not doing so well. They tend to overfit.
- Hammersley stands out as the best among them because it has a lower median.
- After Hammersley, SGD and Sobol sampling come next, giving good results.



Plan for further study

Codes: https://github.com/kaans7?tab=repositories

References:

- https://arxiv.org/abs/2202.06416
- https://www.researchgate.net/publication/257665445_Optimizing_Latin_hypercube_designs_by_particle_swarm
- https://arxiv.org/abs/2306.04066
- https://people.sc.fsu.edu/~jburkardt/classes/urop_2016/du_faber_gunzburger.pdf
- https://people.sc.fsu.edu/~mgunzburger/files_papers/gunzburger-cvt-probalgo.pdf
- https://www.comp.nus.edu.sg/~tants/cvt_files/cvt.pdf
- https://livrepository.liverpool.ac.uk/3002524/1/HaltonSequencesDiscrepancy01.pdf
- https://ttwong12.github.io/papers/udpoint/udpoint.pdf
- https://www.cs.fsu.edu/~mascagni/Hammersley-Handscomb.pdf
- https://web.maths.unsw.edu.au/~fkuo/sobol/joe-kuo-notes.pdf
- https://link.springer.com/article/10.1007/BF01386213