# OPTIMIZATION TECHNIQUES

Design of experiments

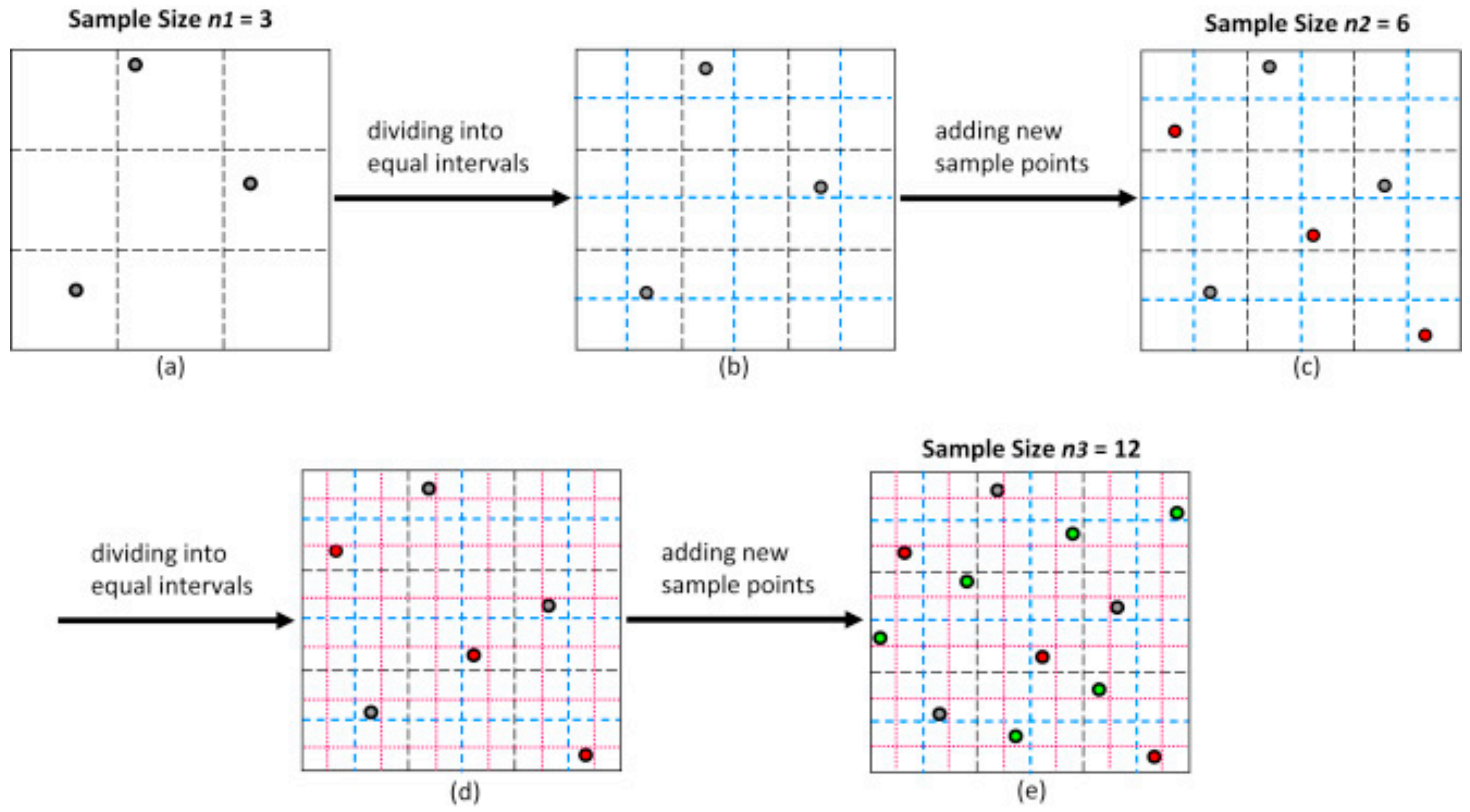
Prof. Giovanni Iacca

giovanni.iacca@unitn.it

UNIVERSITY OF TRENTO - Italy

**Information Engineering**
**and Computer Science Department**

# Design of Experiments (DOEs)



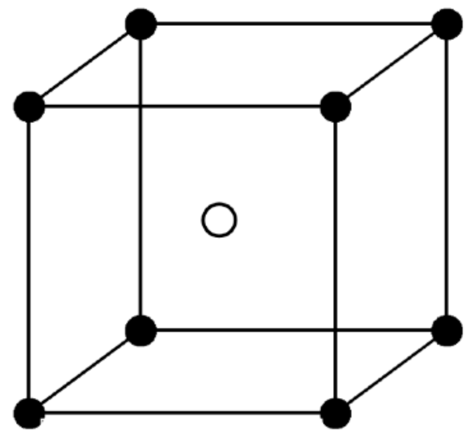
# DESIGN OF EXPERIMENTS

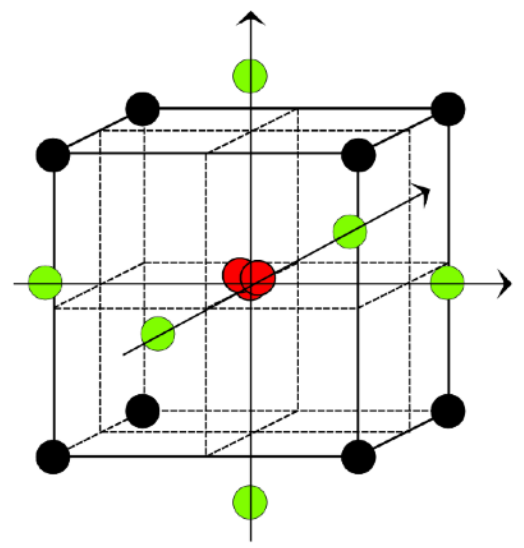
## INTRODUCTION / MOTIVATION

- Design of Experiments (DOEs) is an important activity for any scientist, engineer, or statistician planning to conduct experimental analysis. It is a crucial element in data science, statistical modeling and machine learning.

- Essentially, DOEs methods aim at gaining maximum information with the minimum number of experiments (i.e., samples, or points; in optimization: problem solutions in a given search space), spreading them out with the aim of encouraging a diversity of data. A well-planned DOEs can give a user meaningful data (that cover any possible corner case and reduce possible bias) to act upon, preserving critical resources.

- Main concerns in DOEs include *validity*, *reliability*, and *replicability* (important to choose the right variables, reduce errors, ensuring the method is sufficiently detailed). Another important requirement is achieving appropriate levels of *statistical power* and *sensitivity*.

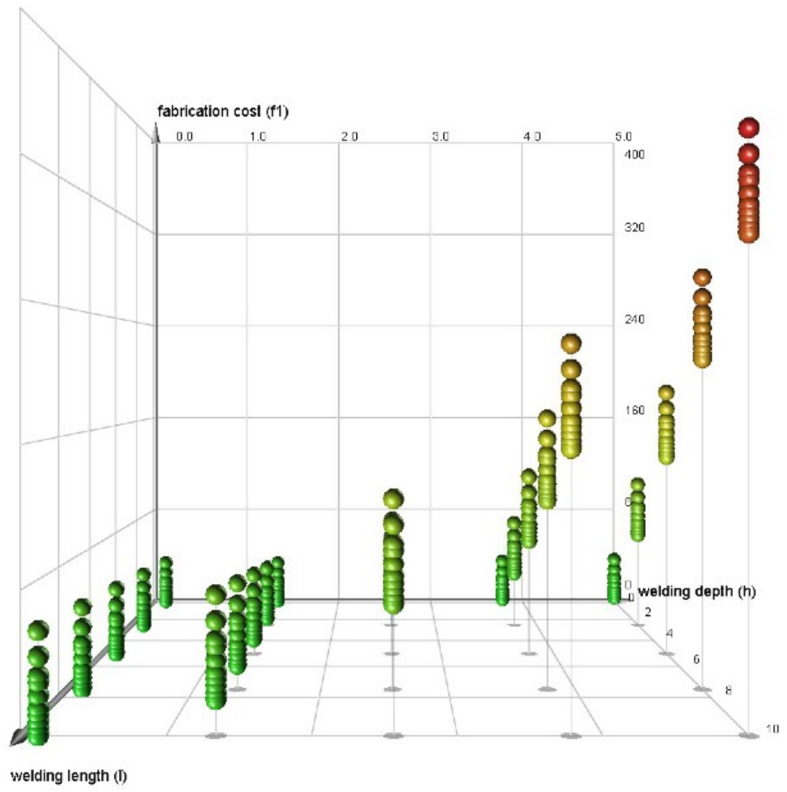




# DESIGN OF EXPERIMENTS

## INTRODUCTION / MOTIVATION

- DOEs methods are also called *space-filling sampling* methods.
- They can be used in the initialization step of many optimization algorithms (e.g. BO, SA, GAs, etc.)
- They can be also used as pre-processing phase, to analyze properties of the search space (e.g., sensitivity analysis, dimensionality reduction, etc.) and gain insight in the problem at hand.
- Sometimes, they can replace optimization altogether as they may find good enough (or even optimal!) solutions per se.



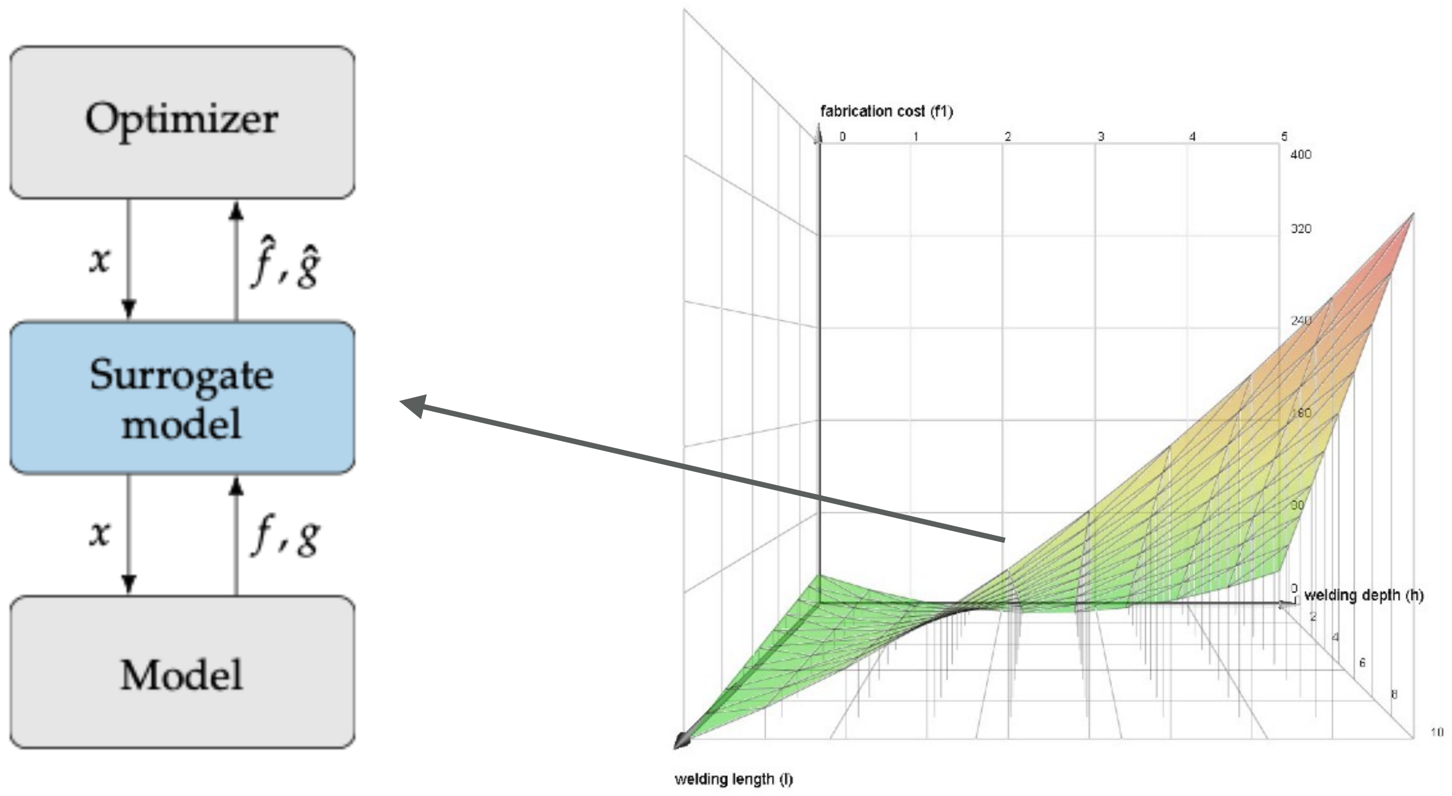
# DESIGN OF EXPERIMENTS

## SURROGATE MODEL-BASED OPTIMIZATION

- DOEs methods are often used to initialize *surrogate* models (*as seen in Bayesian optimization*), i.e., computationally cheap approximations of the objective function. These are also called **Response Surface Models** (RSM), (response surface) approximation models, meta-models, design and analysis of computer experiments (DACE) models, or emulators.

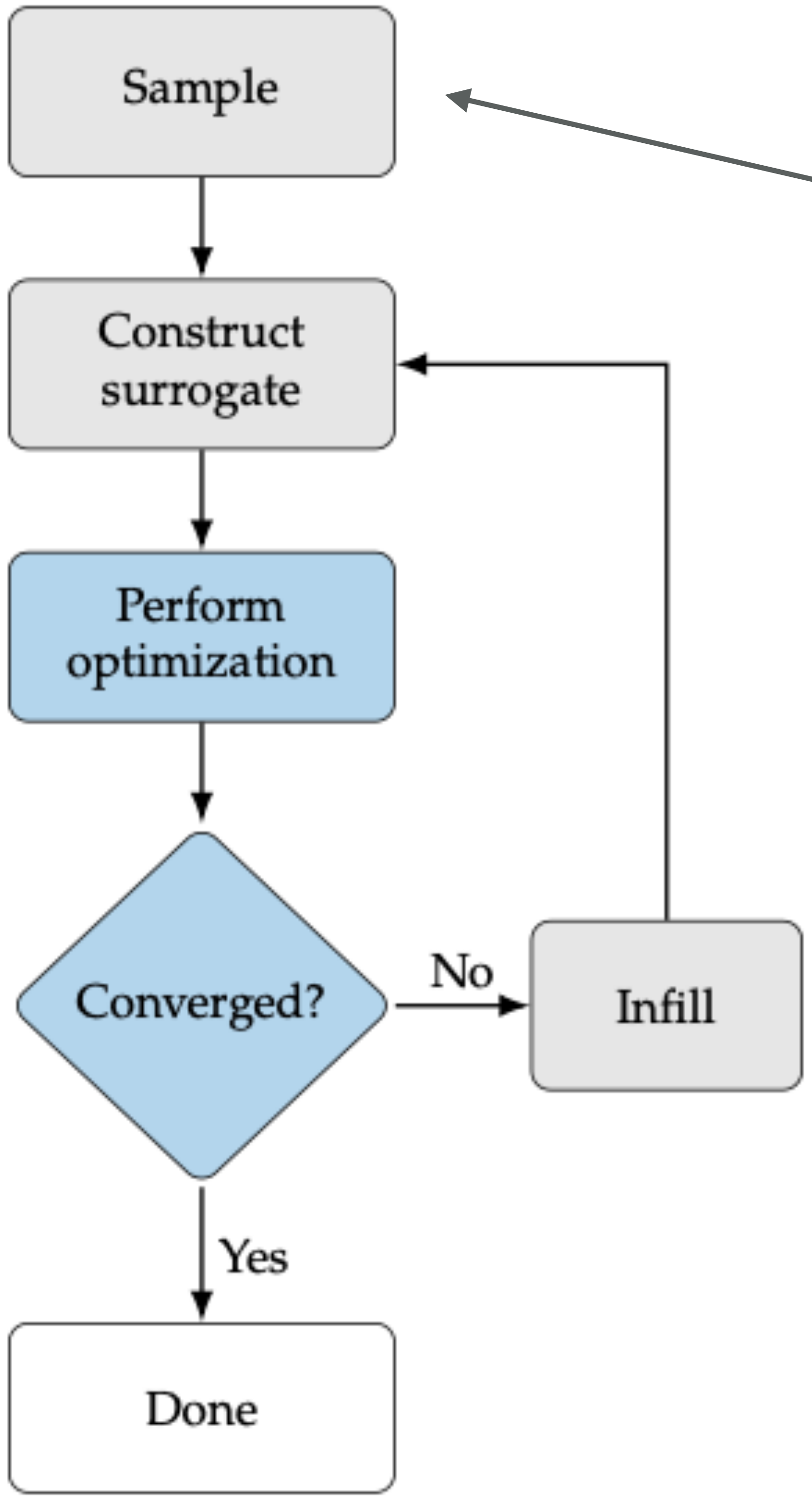
- These models can take many forms (Gaussian Processes aka "Kriging", k-nearest, tree-based models, neural networks, etc.) and need to be trained on the data sampled from DOEs.

- They are typically updated during the optimization through **active learning** (aka **query learning**), which aims at maximizing information gathered from new samples (*as in BO*).



# DESIGN OF EXPERIMENTS

## SURROGATE MODEL-BASED OPTIMIZATION



This lecture is about this sample step

# DESIGN OF EXPERIMENTS

## SURROGATE MODEL-BASED OPTIMIZATION

Exploitation-driven surrogate-based optimization

**Inputs:**

$n_s$: Number of initial samples

$\underline{x}, \overline{x}$: Variable lower and upper bounds

$\tau$: Convergence tolerance

**Outputs:**

$x^*$: Best point identified

$f^*$: Corresponding function value

```
$x^{(i)} = \text{sample}(n_s, n_d)$                                              Sample
$f^{(i)} = f\left(x^{(i)}\right)$                                                Evaluate function
$k = 0$
**while** $k < k_{\max}$ and $\left(\hat{f}^* - f_{\text{new}}\right) / \hat{f}^* < \tau$ **do**
    $\hat{f} = \text{surrogate}\left(x^{(i)}, f^{(i)}\right)$                     Construct surrogate model
    $x^*, \hat{f}^* = \min \hat{f}(x)$                                           Perform optimization on the surrogate function
    $f_{\text{new}} = f(x^*)$                                                    Evaluate true function at predicted optimum
    $x^{(i)} = x^{(i)} \cup x^*$                                                 Append new point to training data
    $f^{(i)} = f^{(i)} \cup f_{\text{new}}$                                      Append corresponding function value
    $k = k + 1$
**end while**
```

# DESIGN OF EXPERIMENTS

## SURROGATE MODEL-BASED OPTIMIZATION

Efficient global optimization (*essentially, an instance of Bayesian Optimization*)

**Inputs:**

$n_s$: Number of initial samples

$\underline{x}, \overline{x}$: Lower and upper bounds

$\tau$: Minimum expected improvement

**Outputs:**

$x^*$: Best point identified

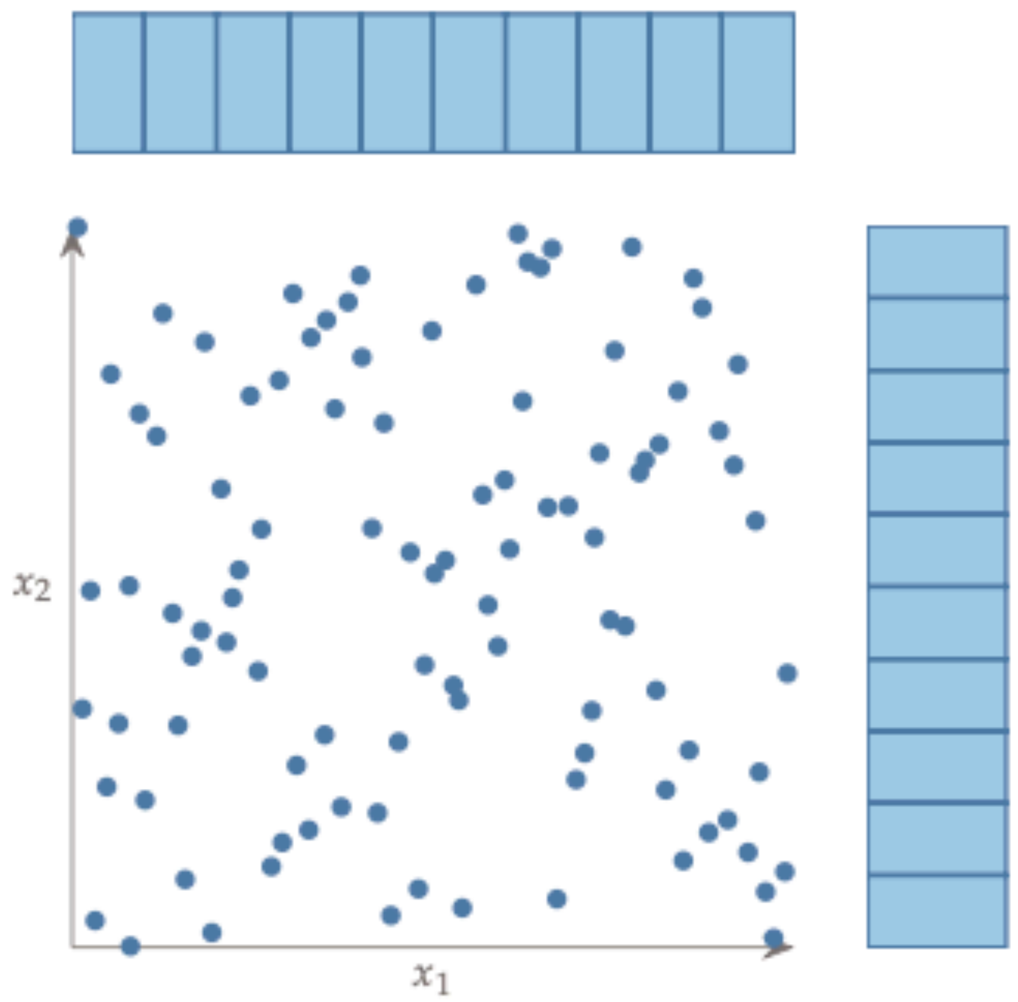
$f^*$: Corresponding function value

```
$x^{(i)} = \text{sample}(n_s, n_d)$                    Sample
$f^{(i)} = f(x^{(i)})$                                 Evaluate function
$f^* = \min\{f^{(i)}\}$                                Best point so far; also update corresponding $x^*$
$k = 0$
while $k < k_{\max}$ and $f_{ei} > \tau$ do
    $\mu(x), \sigma(x) = \text{GP}(x^{(i)}, f^{(i)})$   Construct Gaussian process surrogate model
    $x_k, f_{ei} = \max EI(x)$                         Maximize expected improvement
    $f_k = f(x_k)$                                     Evaluate true function at predicted optimum
    $f^* = \min\{f^*, f_k\}$                           Update best point and $x^*$ if necessary
    $x^{(i)} \leftarrow [x^{(i)}, x_k]$                Add new point to training data
    $f^{(i)} \leftarrow [f^{(i)}, f_k]$
    $k = k + 1$
end while
```

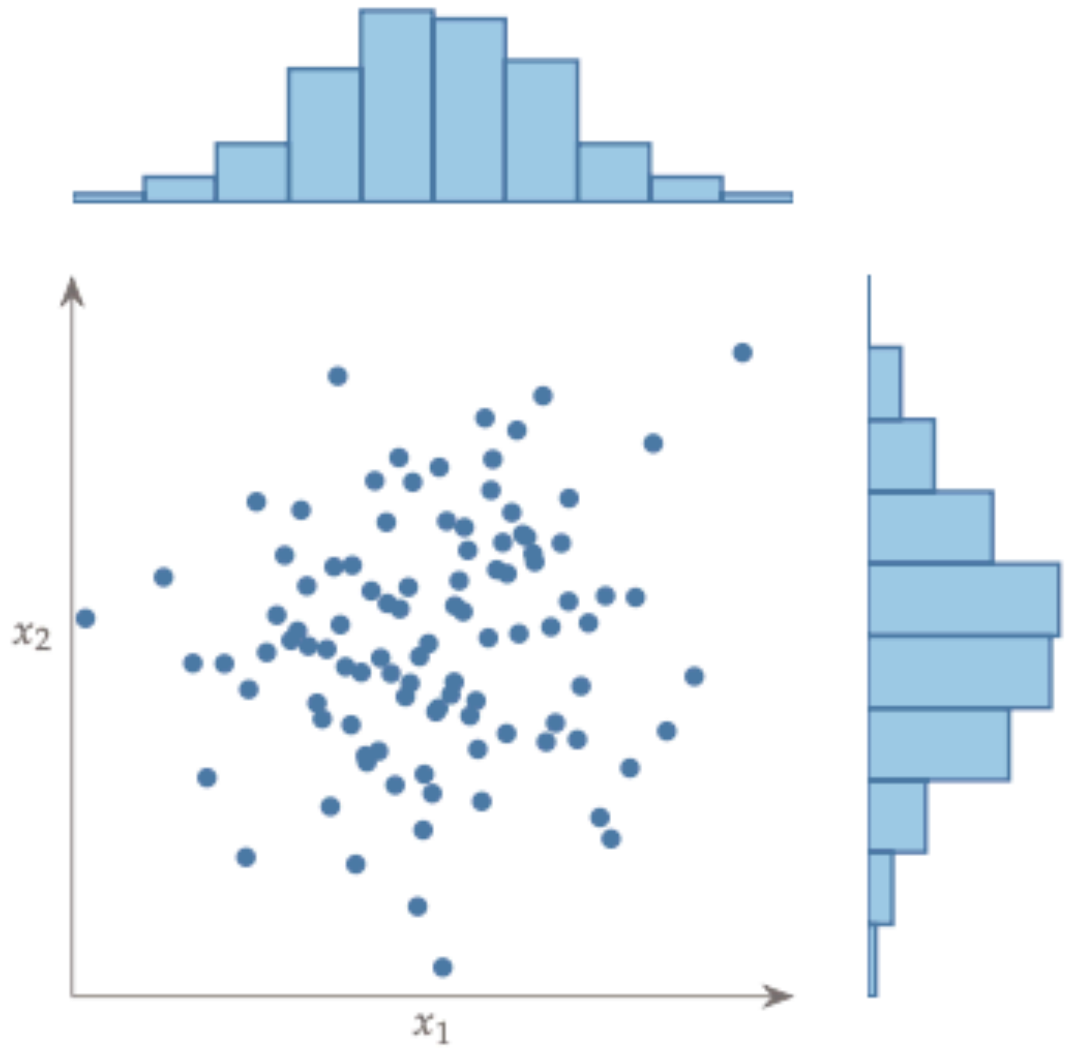
# DESIGN OF EXPERIMENTS

## THE NAÏVE APPROACH: RANDOMIZED DESIGNS

Which one is better? Why?



Uniform distribution in each direction

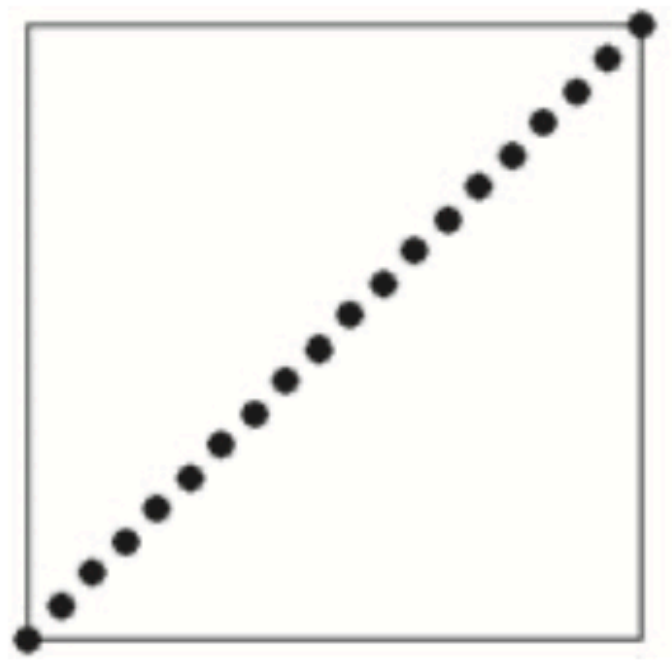


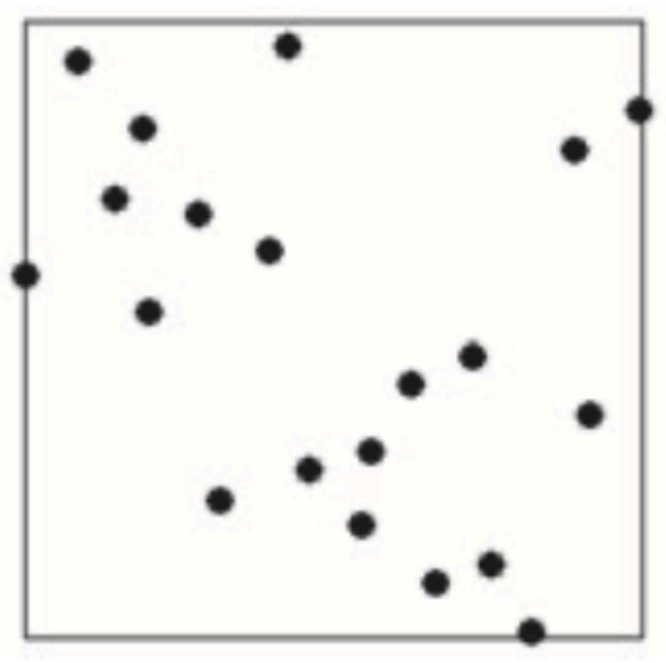
Normal distribution in each direction

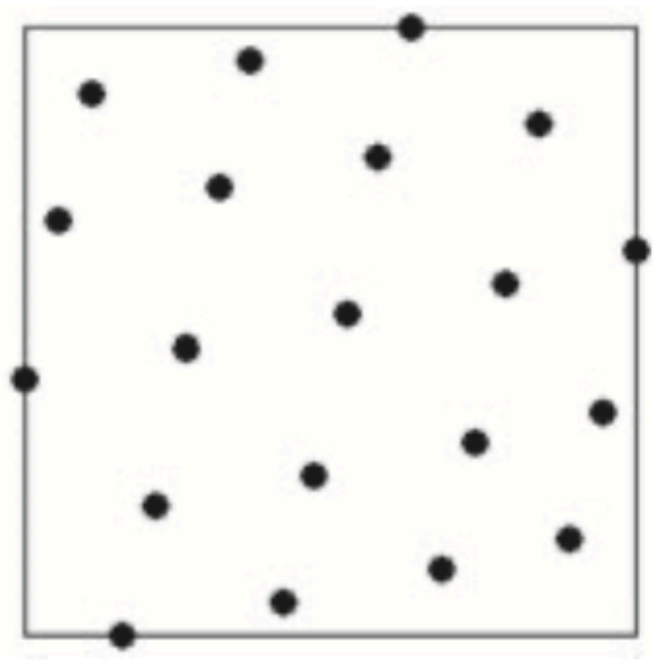
# DESIGN OF EXPERIMENTS

## WHAT ELSE CAN WE DO (APART FROM RANDOM)?

Which one is better? Why?


(a) Design with very poor space filling properties.

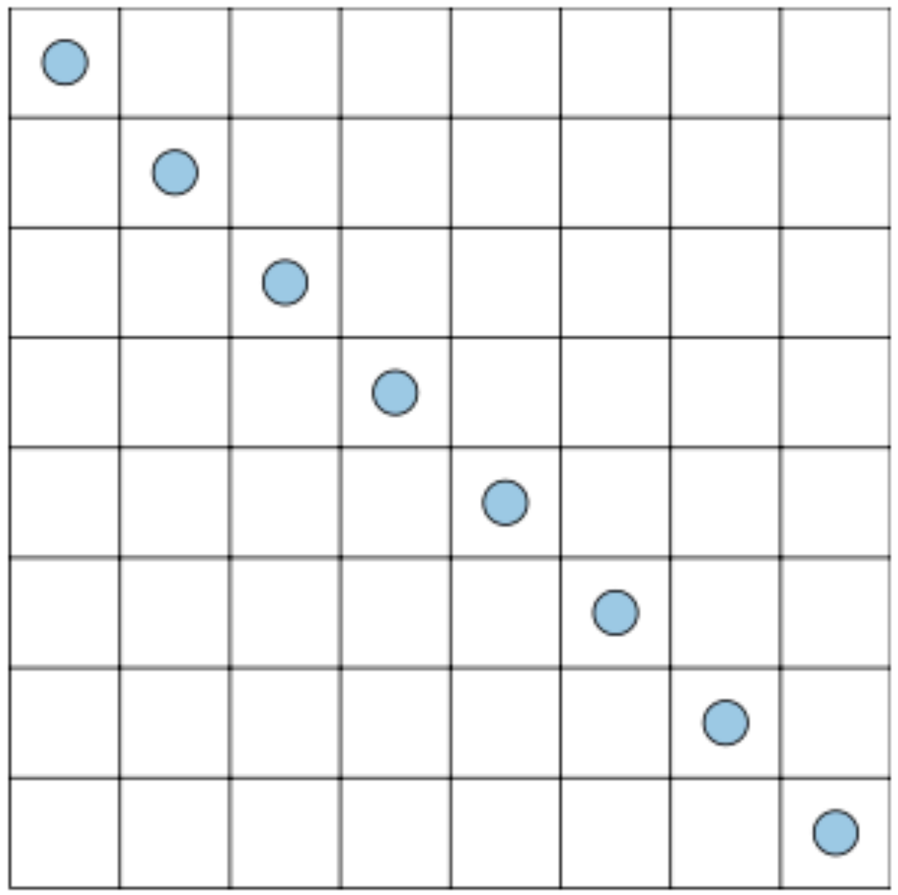

(b) Randomized design.


(c) Design with good space filling properties.

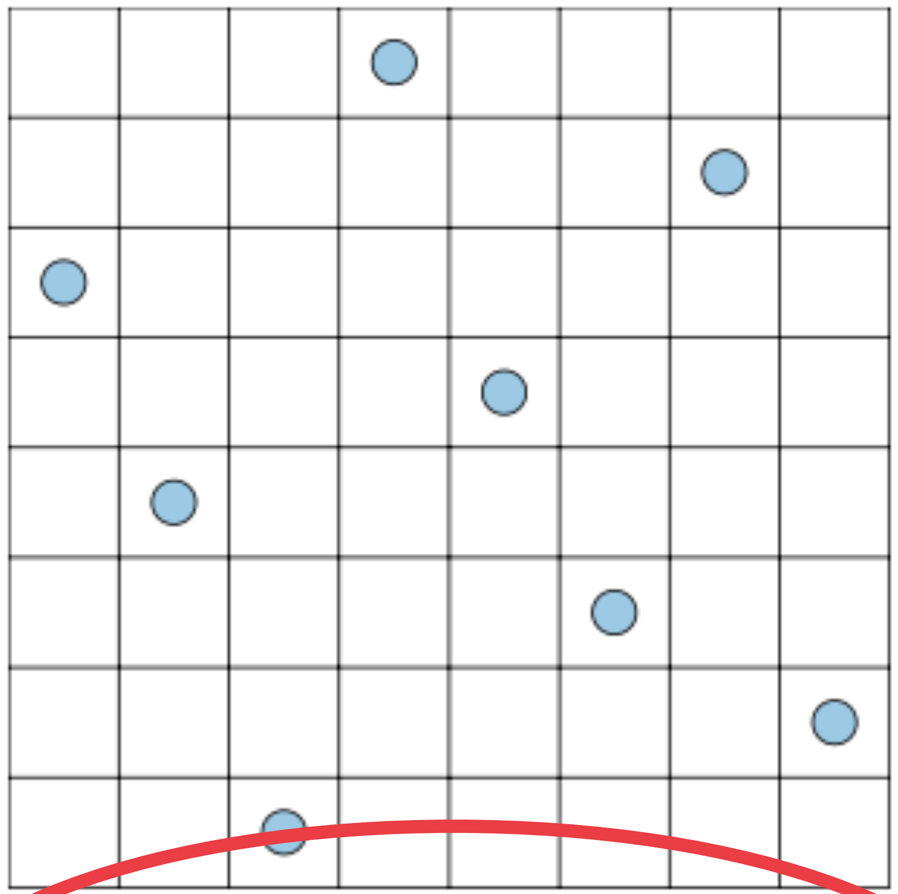
# DESIGN OF EXPERIMENTS

## POOR VS GOOD SPACE-FILLING

Goals: better coverage/regularity, reduce bias, avoid redundancies.



A sampling strategy whose projection uniformly spans each dimension but does not fill the space well



A sampling strategy whose projection uniformly spans each dimension and fills the space more effectively

# DESIGN OF EXPERIMENTS

## HALTON SEQUENCES

- Used to generate samples in a normalized space, e.g. $(0,1)\times(0,1) \subset \mathbb{R}^2$
- These sequences of samples are constructed *deterministically*, based on *coprime numbers* (i.e., numbers s.t. the only positive integer that is a divisor of both of them is 1)
- Even though these sequences are deterministic, they are quasi-random (formally, they are *low-discrepancy sequences*) → But, they yield better coverage than random!
- First introduced in 1960, they generalize the one-dimensional van der Corput sequences, introduced in 1935. The latter are constructed by *reversing* the base-*b* representation of the sequence of natural numbers (1, 2, 3, …). The *n*-th number in the van der Corput sequence is:

$$g_b(n) = \sum_{k=0}^{L-1} d_k(n) b^{-k-1}$$

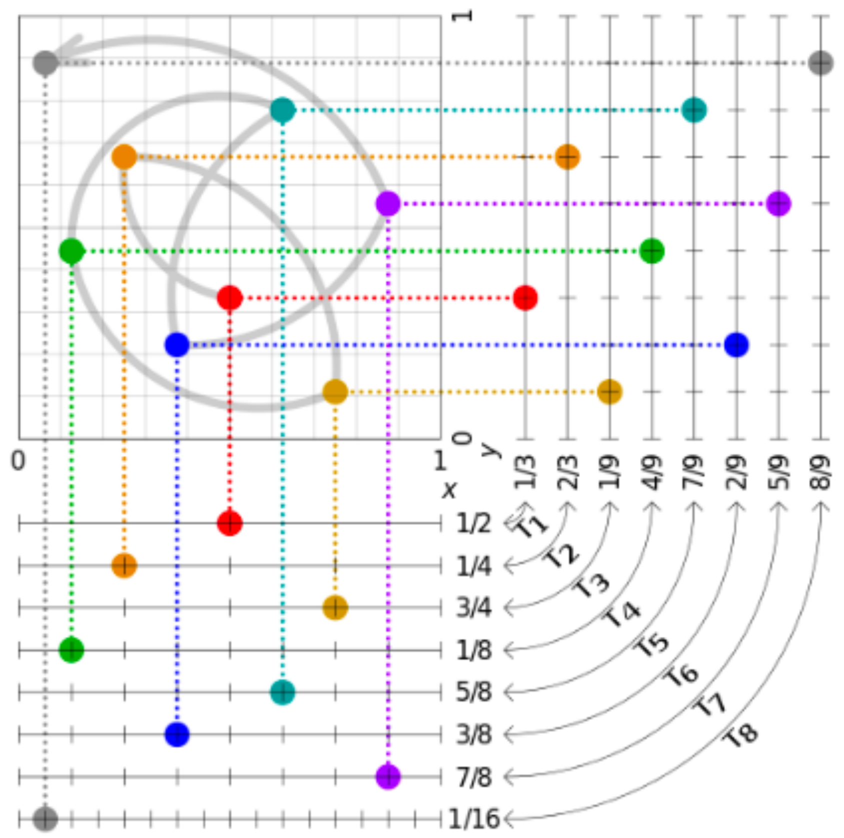
where *b* is the base in which the number *n* is represented, and $d_k(n) \in [0, b)$ is the *k*-th digit in the *b*-ary expansion of *n*.

**Example of Halton sequence in 2D**

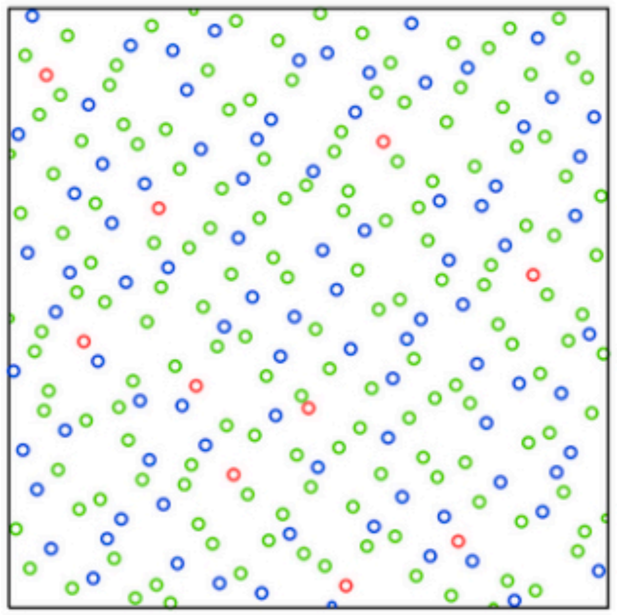
- Let's assume one dimension of the sequence to be based on 2 and the other on 3.
  To generate the sequence for 2, we divide the interval (0,1) in half, fourths, eighths, etc.:
  1⁄2, 1⁄4, 3⁄4, 1⁄8, 5⁄8, 3⁄8, 7⁄8, 1⁄16, 9⁄16,…
- To generate the sequence for 3, we divide the interval (0,1) in thirds, ninths, twenty-sevenths, etc.:
  1⁄3, 2⁄3, 1⁄9, 4⁄9, 7⁄9, 2⁄9, 5⁄9, 8⁄9, 1⁄27,…
- When we pair them up, we get a sequence of samples in a unit square:
  (1⁄2, 1⁄3), (1⁄4, 2⁄3), (3⁄4, 1⁄9), (1⁄8, 4⁄9), (5⁄8, 7⁄9), (3⁄8, 2⁄9), (7⁄8, 5⁄9), (1⁄16, 8⁄9), (9⁄16, 1⁄27), …

# DESIGN OF EXPERIMENTS

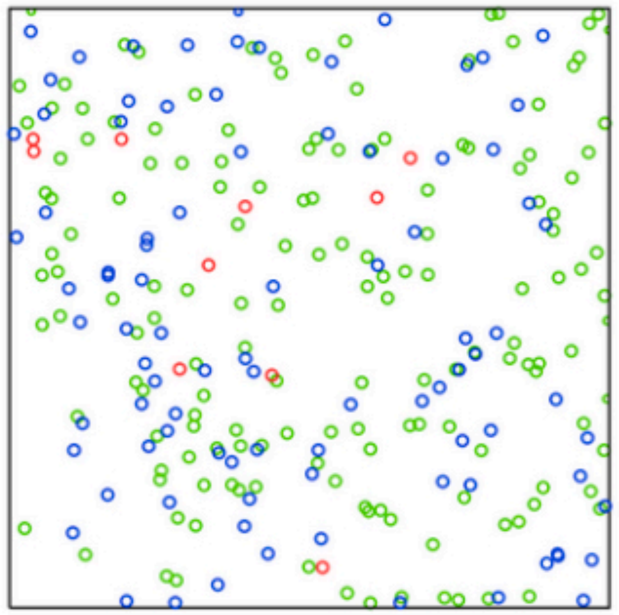
## HALTON SEQUENCES



The first 8 samples of the 2,3 Halton sequence



Halton



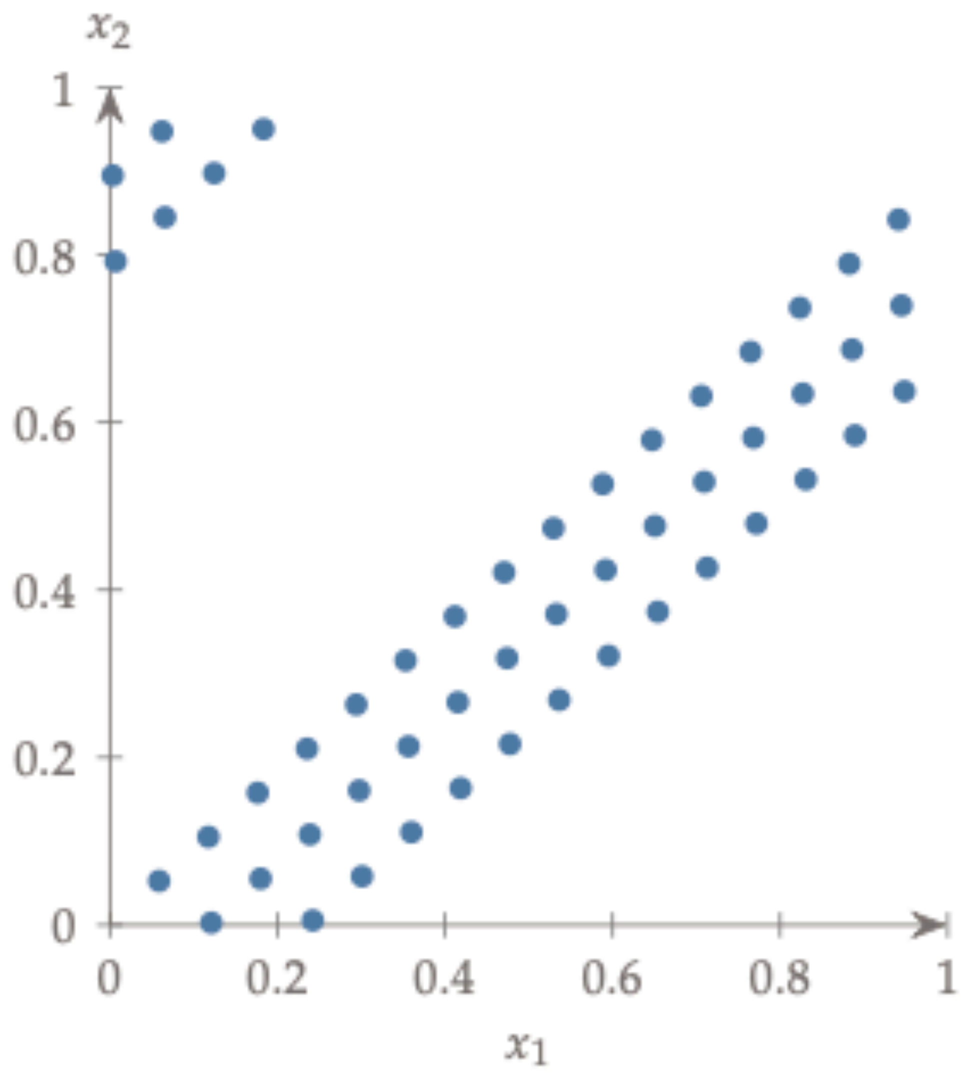
Random

The first 256 samples of the 2,3 Halton sequence vs 256 pseudorandom numbers. The Halton sequence covers the space more evenly (red=1,...,10, blue=11,...,100, green=101,...,256).

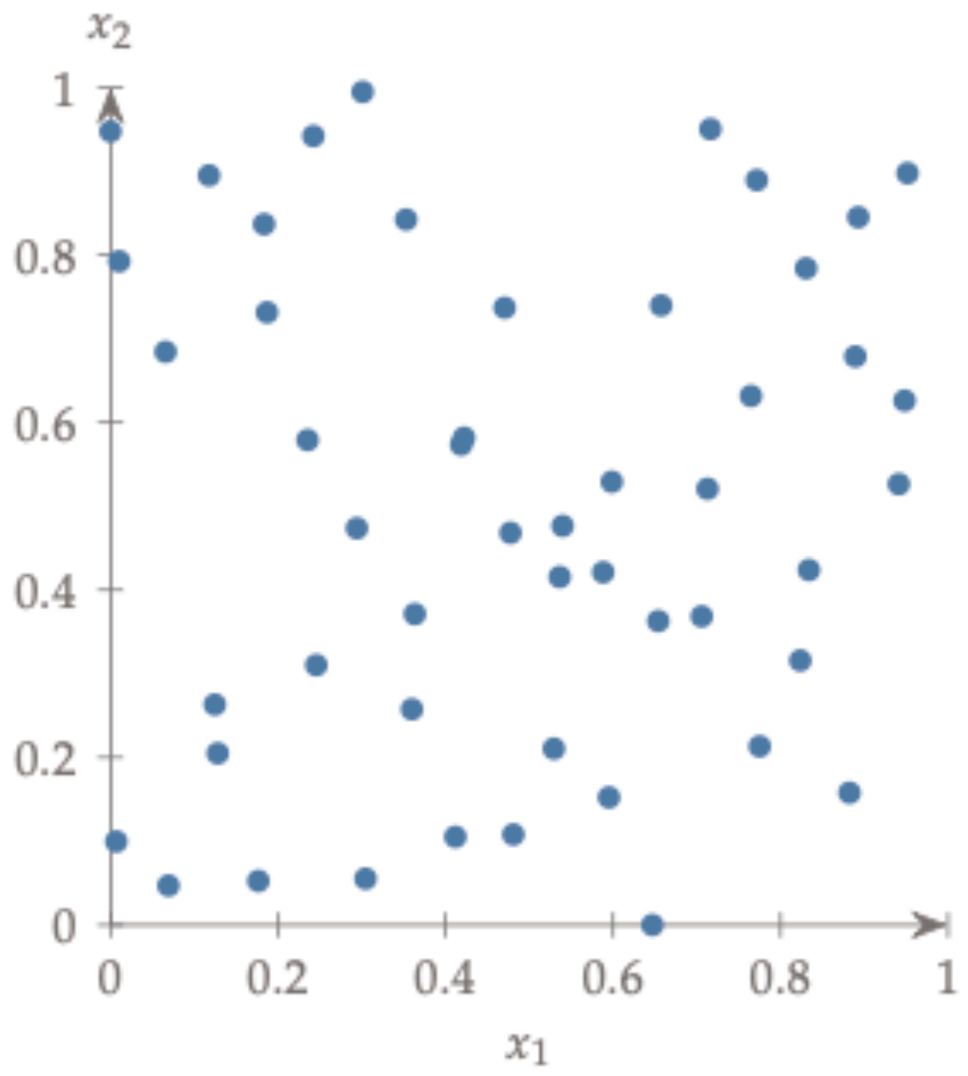
**NOTE**: Even though standard Halton sequences perform very well in low dimensions, correlation problems have been noted between sequences generated from higher primes. To avoid this, it is common to drop a predetermined quantity of initial numbers, depending on the primes chosen. Alternatives: *scrambled Halton sequence*, which uses permutations of the coefficients used in the construction of the standard sequence; *leaped Halton sequence*, which skips samples in the standard sequence.

# DESIGN OF EXPERIMENTS

## HALTON SEQUENCES



**Standard Halton sequence**



**Scrambled Halton**

Halton sequence with base 17 for $x_1$ and base 19 for $x_2$

# DESIGN OF EXPERIMENTS

## HALTON SEQUENCES - PYTHON GENERATOR

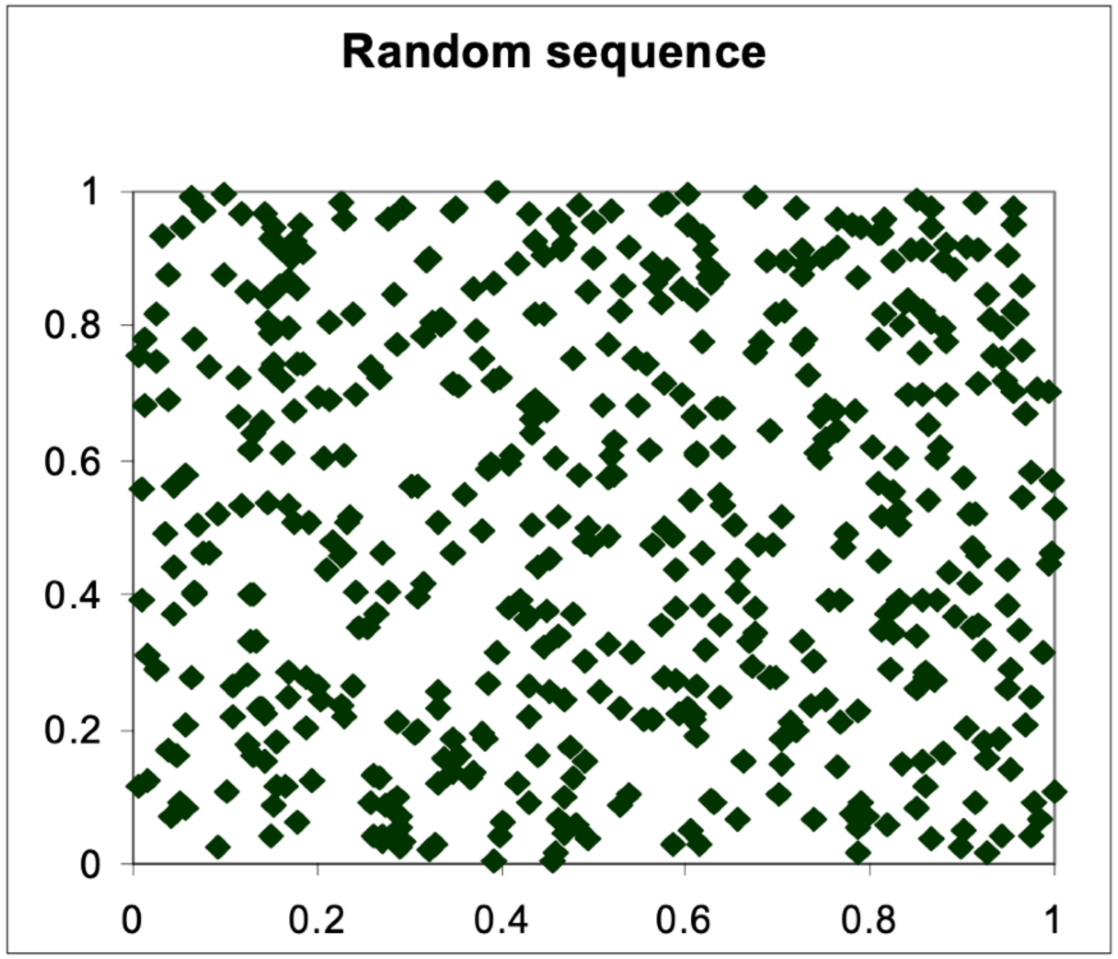
```python
def Halton(b):
    """Generator function for Halton sequence."""
    n, d = 0, 1
    while True:
        x = d - n
        if x == 1:
            n = 1
            d *= b
        else:
            y = d // b
            while x <= y:
                y //= b
            n = (b + 1) * y - x
        yield n / d
```

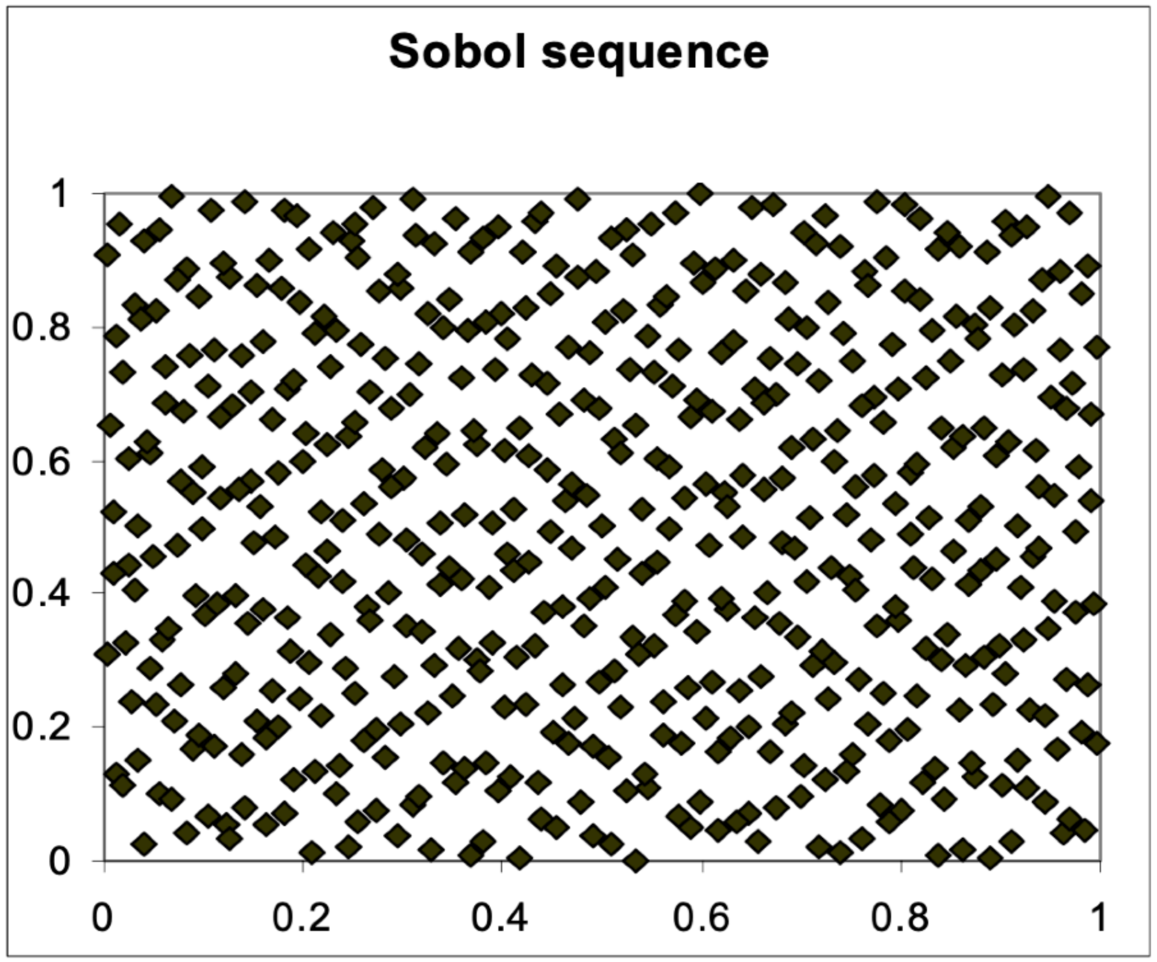
## DESIGN OF EXPERIMENTS

### SOBOL SAMPLING

Another algorithm for generating low-discrepancy sequences.

→ Again, better coverage than random! (at least in low dimensionalities)



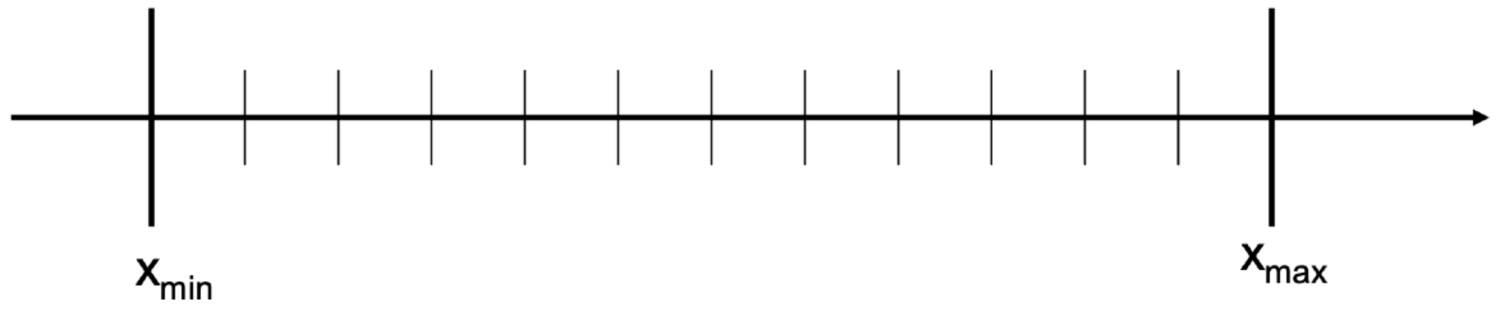


# DESIGN OF EXPERIMENTS

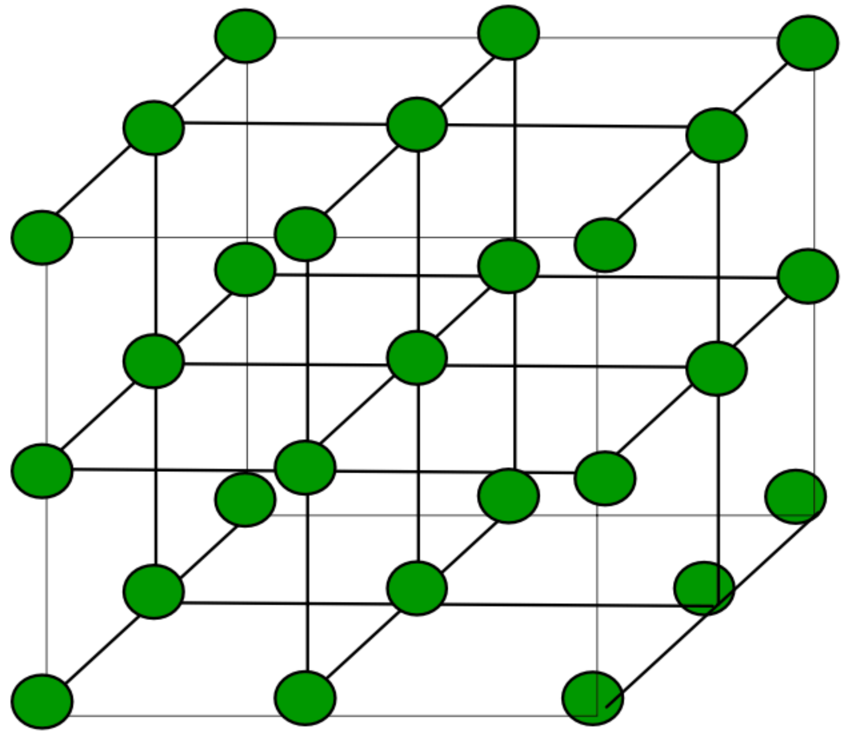
## FULL FACTORIAL DESIGN

Given $N$ variables and $M$ (base) divisions per variables, the **Full Factorial** design creates all possible $M^N$ combinations. ➞ Feasible only for small $N$ and $M$.

With $M$=2 it is possible to calculate 1st-order variable interactions on the obj. function (e.g. $x_1 x_2$).
With $M$=3 it is possible to calculate 2nd-order variable interactions on the obj. function (e.g. $x_1^2 x_2$).



Example: $M$=11 divisions



Example: N=3, M= 3 ➞ 27 samples

# DESIGN OF EXPERIMENTS

## FRACTIONAL (AKA REDUCED) FACTORIAL DESIGN

In contrast to Full Factorial, **Fractional (aka Reduced) Factorial** design creates all $M^P$ combinations, with $P < N$. → Computationally cheaper than Full Factorial, but less coverage and not all variable interactions can be analyzed.

Basically a Full Factorial design on $P < N$ variables. The remaining $N$-$P$ variables can be obtained e.g. as combinations (typically, products) of the selected $P$ variables.

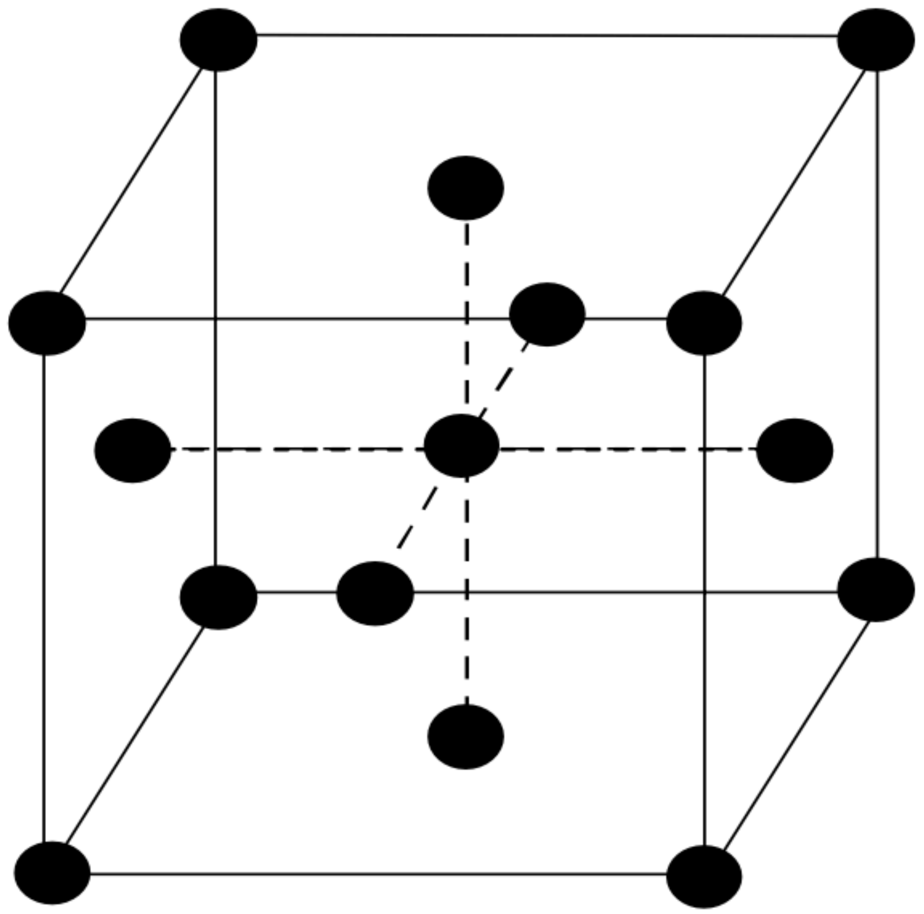
| n. design | x1 | x2 | x3 | x4 (=x1*x2) |
|---|---|---|---|---|
| 1 | + | + | + | + |
| 2 | + | + | - | + |
| 3 | + | - | + | - |
| 4 | + | - | - | - |
| 5 | - | + | + | - |
| 6 | - | + | - | - |
| 7 | - | - | + | + |
| 8 | - | - | - | + |

Example of Reduced Factorial with $N$=4, $P$=3.

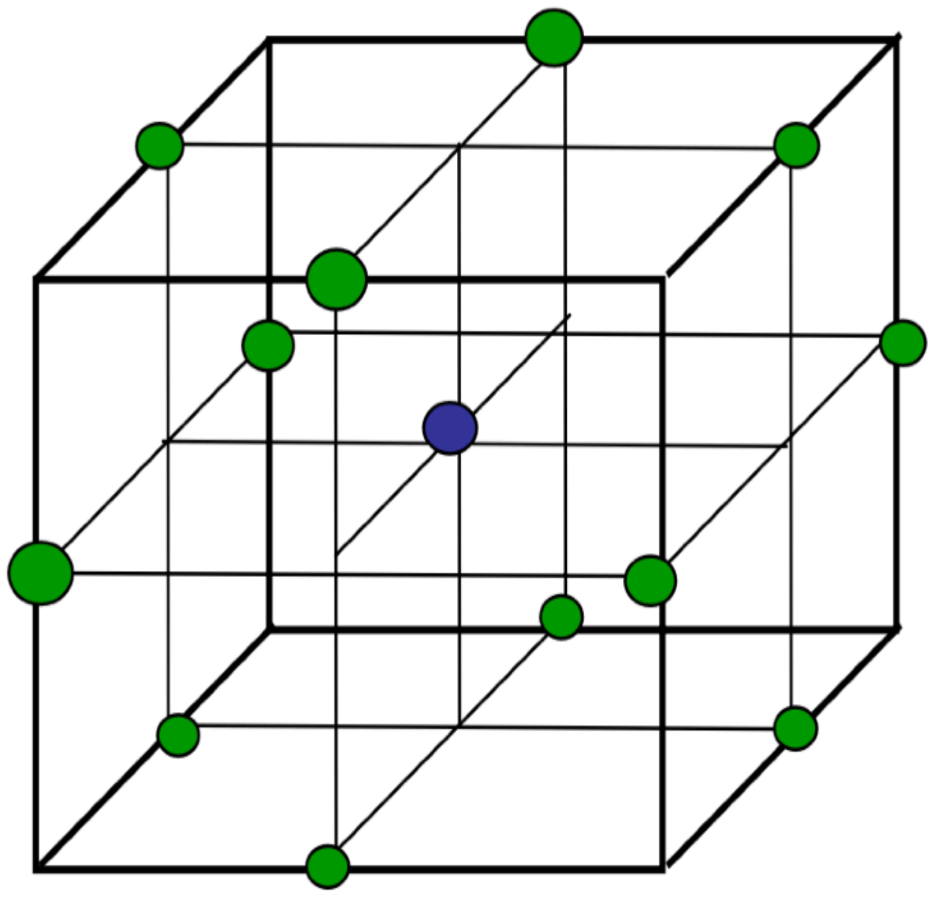
# DESIGN OF EXPERIMENTS

## CENTRAL COMPOSITE DESIGN AND BOX-BEHNKER

Both methods are less expensive than Full Factorial (and allow analysis of second-order interactions), but still their complexity grows exponentially with *N*, i.e., the number of variables.



**Cubic face centered**
($2^N + 2N + 1$ samples)
E.g. $N$=3 → 15 samples



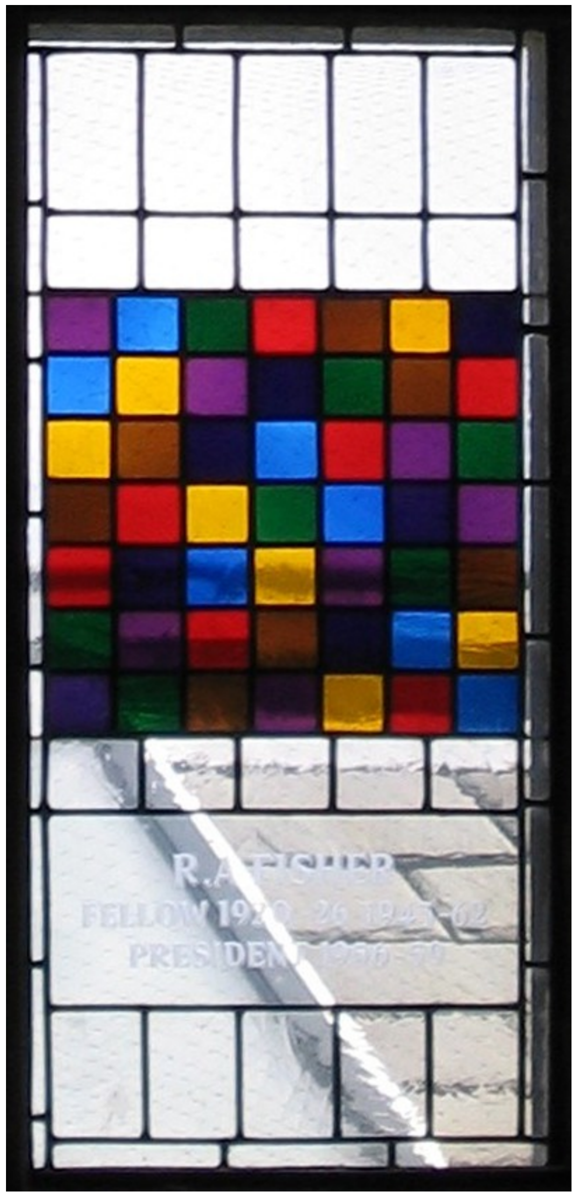
**Box-Behnker**
($2^N + 2(N-1) + 1$ samples)
$N$=3 → 13 samples

# DESIGN OF EXPERIMENTS

## LATIN SQUARE

An *N*×*N* array filled with *N* different symbols, each occurring *exactly once* in each row and exactly once in each column (basically, just like Sudoku).

| | | | | | |
|---|---|---|---|---|---|
| A | B | F | C | E | D |
| B | C | A | D | F | E |
| C | D | B | E | A | F |
| D | E | C | F | B | A |
| E | F | D | A | C | B |
| F | A | E | B | D | C |



A 7×7 Latin square on a stained-glass window at Gonville and Caius College, Cambridge commemorated Sir Ronald Fisher, whose *Design of Experiments* discussed Latin squares. The window was removed in 2020 because of Fisher's connection with eugenics.

# DESIGN OF EXPERIMENTS

## LATIN SQUARE - GENERALIZATIONS

- **Latin rectangle**: a generalization of a Latin square in which there are $N$ columns and $N$ possible values, but the number of rows may be smaller than $N$. Each value still appears at most once in each row and column.

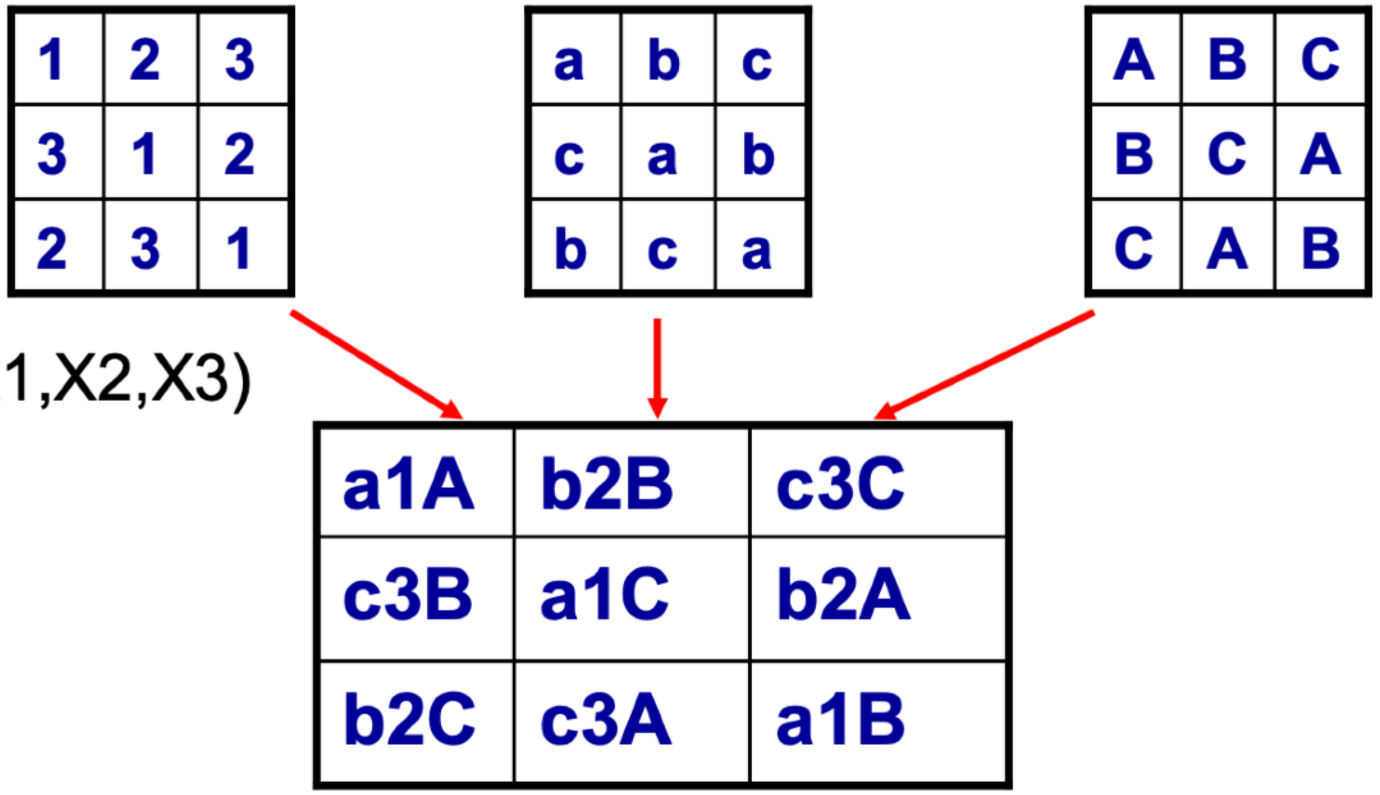
- **Latin hypercube**: a generalization of a Latin square from 2D to $N$>2 D dimensions. Each sample is the only one in each axis-aligned hyperplane containing it.
  - When sampling a function of $N$ variables, the range of each variable is divided into $M$ equally probable intervals; $M$ samples are then placed to satisfy the Latin hypercube requirements; this forces the number of divisions, $M$, to be equal for each variable.
  - This sampling scheme *does not require more samples for more dimensions*. In fact, the total number of samples is $M^2$.
  - This independence from $N$ is one of the main advantages of this sampling scheme (differently from Factorial schemes). Another advantage is that random samples can be taken one at a time, remembering which samples were taken so far.

# DESIGN OF EXPERIMENTS

## LATIN HYPERCUBE SAMPLING (LHS)

Latin Square for 3 variables (X1,X2,X3)
with 3 levels :

X1(1,2,3),
X2(A,B,C),
X3(a,b,c)



| | | |
|---|---|---|
| 1 | 2 | 3 |
| 3 | 1 | 2 |
| 2 | 3 | 1 |

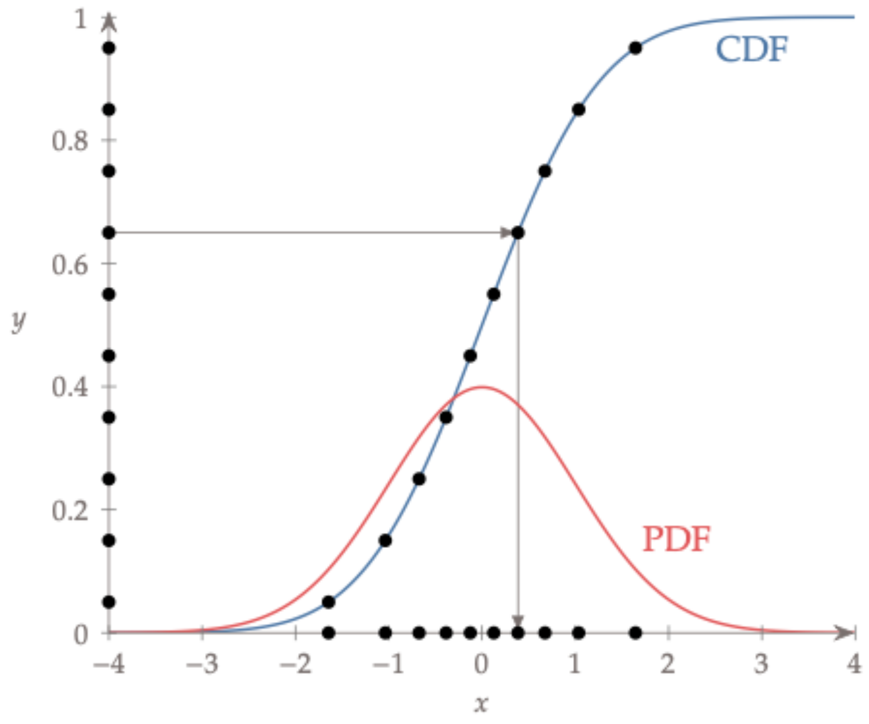
| | | |
|---|---|---|
| a | b | c |
| c | a | b |
| b | c | a |

| | | |
|---|---|---|
| A | B | C |
| B | C | A |
| C | A | B |

| | | |
|---|---|---|
| a1A | b2B | c3C |
| c3B | a1C | b2A |
| b2C | c3A | a1B |

# DESIGN OF EXPERIMENTS

## LATIN HYPERCUBE SAMPLING (LHS)



**Inputs:**

$n_s$: Number of samples

$n_d$: Number of dimensions

$P = \{P_1, \ldots, P_{n_d}\}$: (optionally) A set of cumulative distribution functions

**Outputs:**

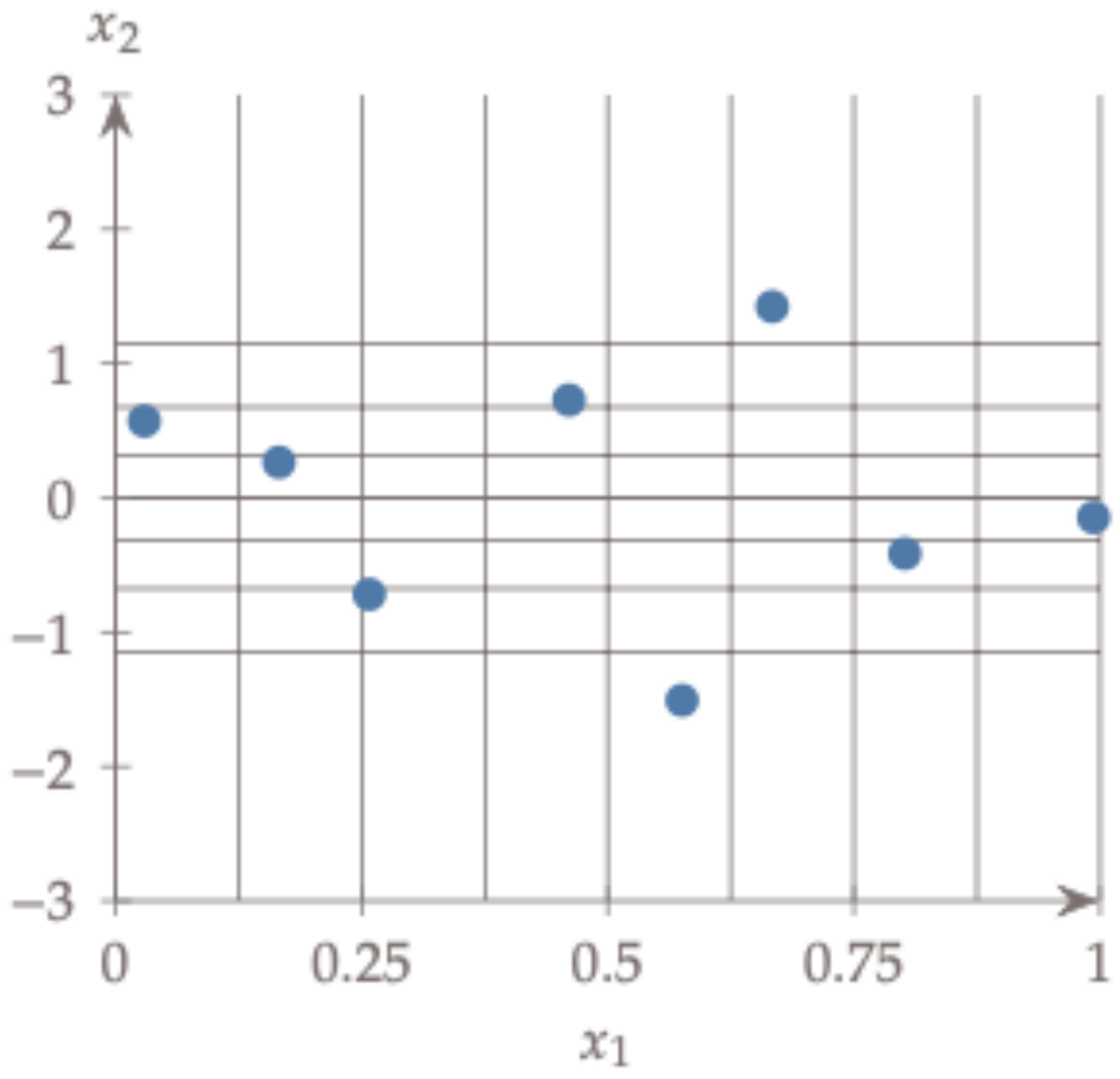
$X = \{x_1, \ldots, x_{n_s}\}$: Set of sample points

---

```
for $j = 1$ to $n_d$ do
    for $i = 1$ to $n_s$ do
        $V_{ij} = \frac{i}{n_s} - \frac{R_{ij}}{n_s}$ where $R_{ij} \in \mathbb{U}[0,1]$     Randomly choose a value in each equally spaced cell from uniform distribution
    end for
    $X_{*j} = P_j^{-1}(V_{*j})$ where $P_j$ is a CDF     Evaluate inverse CDF
    Randomly permute the entries of this column $X_{*j}$     Alternatively, permute the indices $1 \ldots n_s$ in the prior for loop
end for
```

# DESIGN OF EXPERIMENTS

## LATIN HYPERCUBE SAMPLING (LHS)

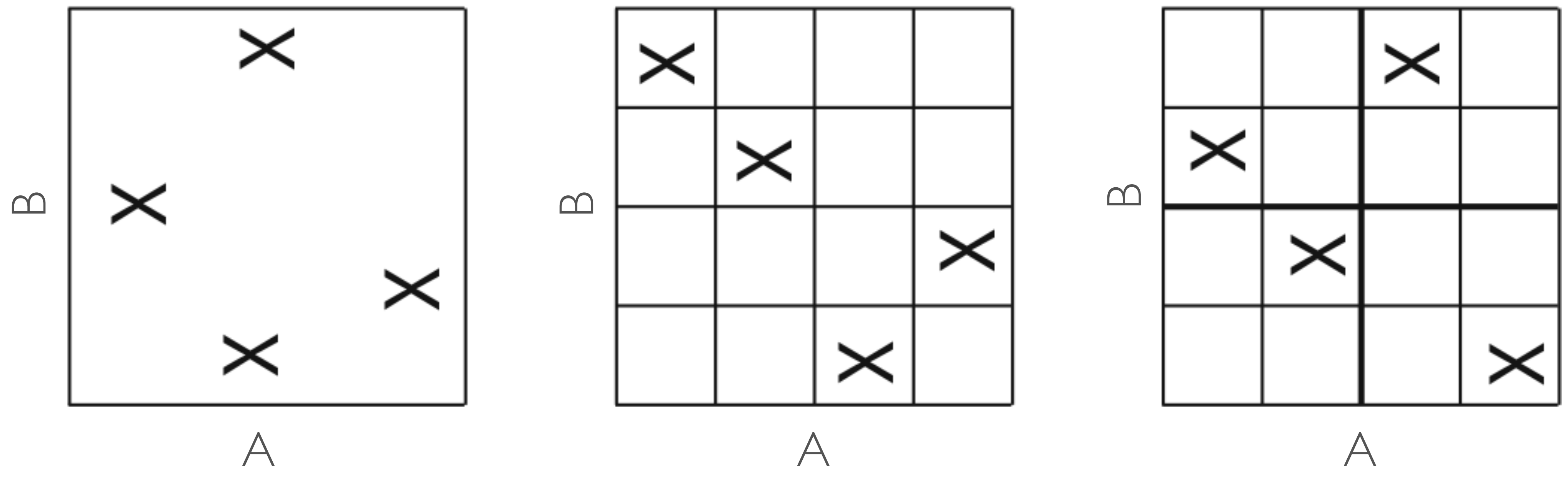


An example from the LHS algorithm showing uniform distribution in $x_1$ and a Gaussian distribution in $x_2$ with 8 samples. The equiprobable bins are shown as grid lines.

## DESIGN OF EXPERIMENTS

### LATIN HYPERCUBE SAMPLING (LHS)

- A sampling method that ensures that each sampling space dimension is roughly evenly sampled.
- Further generalization: **orthogonal sampling.** Basically an extension to LHS that ensures that each subspace is evenly sampled. This also ensures that correlation between sampling dimensions is minimized.



Example of random sampling (left) vs LHS (center) vs orthogonal sampling (right):
- **Random and LHS**: Strong (negative) correlation between A and B; the combinations of (low A, low B) and (high A, high B) are not sampled at all.
- **Orthogonal sampling**: Very little correlation between A and B; all the 4 subspaces are evenly sampled.

Note that the subspaces are somewhat arbitrary, but larger than a single sample, and smaller than the entire sample space.

# DESIGN OF EXPERIMENTS

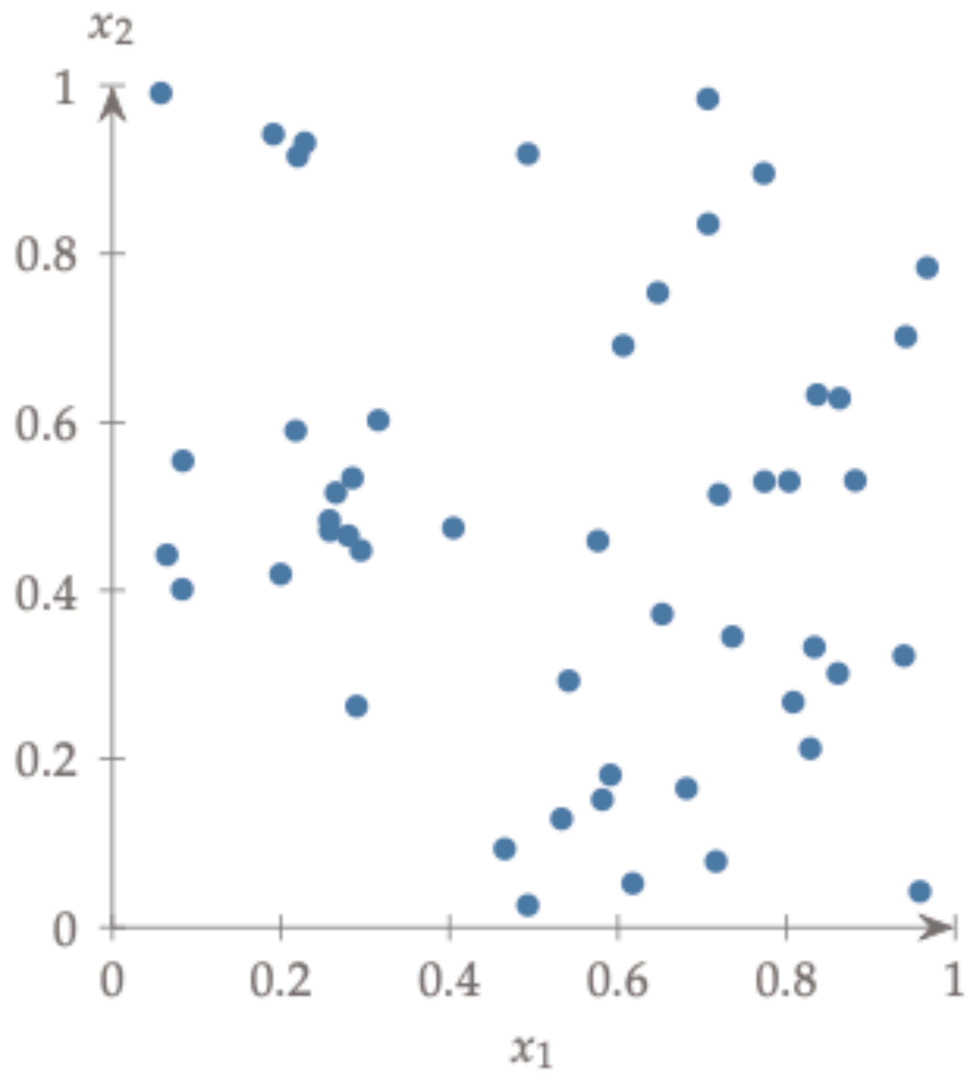
## LATIN HYPERCUBE SAMPLING (LHS)

- In two dimensions the difference between random sampling, Latin hypercube sampling, and orthogonal sampling can be explained as follows:
  - In **random sampling** new samples are generated *without taking into account the previously generated samples*. One does not necessarily need to know beforehand how many samples are needed.
  - In **Latin hypercube sampling** one must first decide how many samples to use and for each sample *remember in which row and column the sample was taken*. Such configuration is similar to having *N* rooks ♖ on a chess board without threatening each other.
  - In **orthogonal sampling**, the sample space is divided into equally probable subspaces. All samples are then chosen simultaneously making sure that the total set of samples is a Latin hypercube sample and that *each subspace is sampled with the same density*.

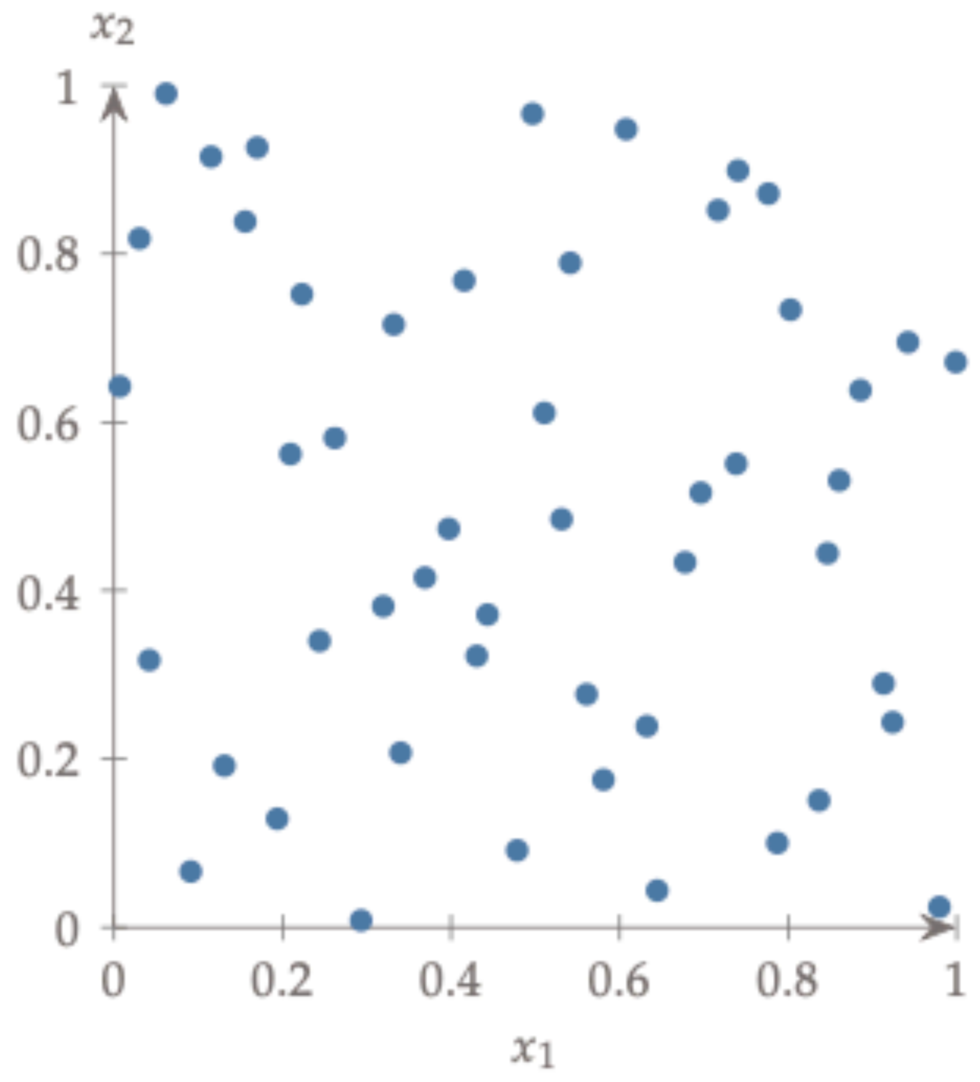
- Thus, orthogonal sampling ensures that the set of random numbers is a <u>very good representative</u> of the real variability, LHS ensures that the set of random numbers is <u>representative</u> of the real variability whereas traditional random sampling (aka *brute force*) is <u>just a set of random numbers without any guarantees</u>.

# DESIGN OF EXPERIMENTS

## LATIN HYPERCUBE SAMPLING (LHS) - 2D EXAMPLE (50 SAMPLES)



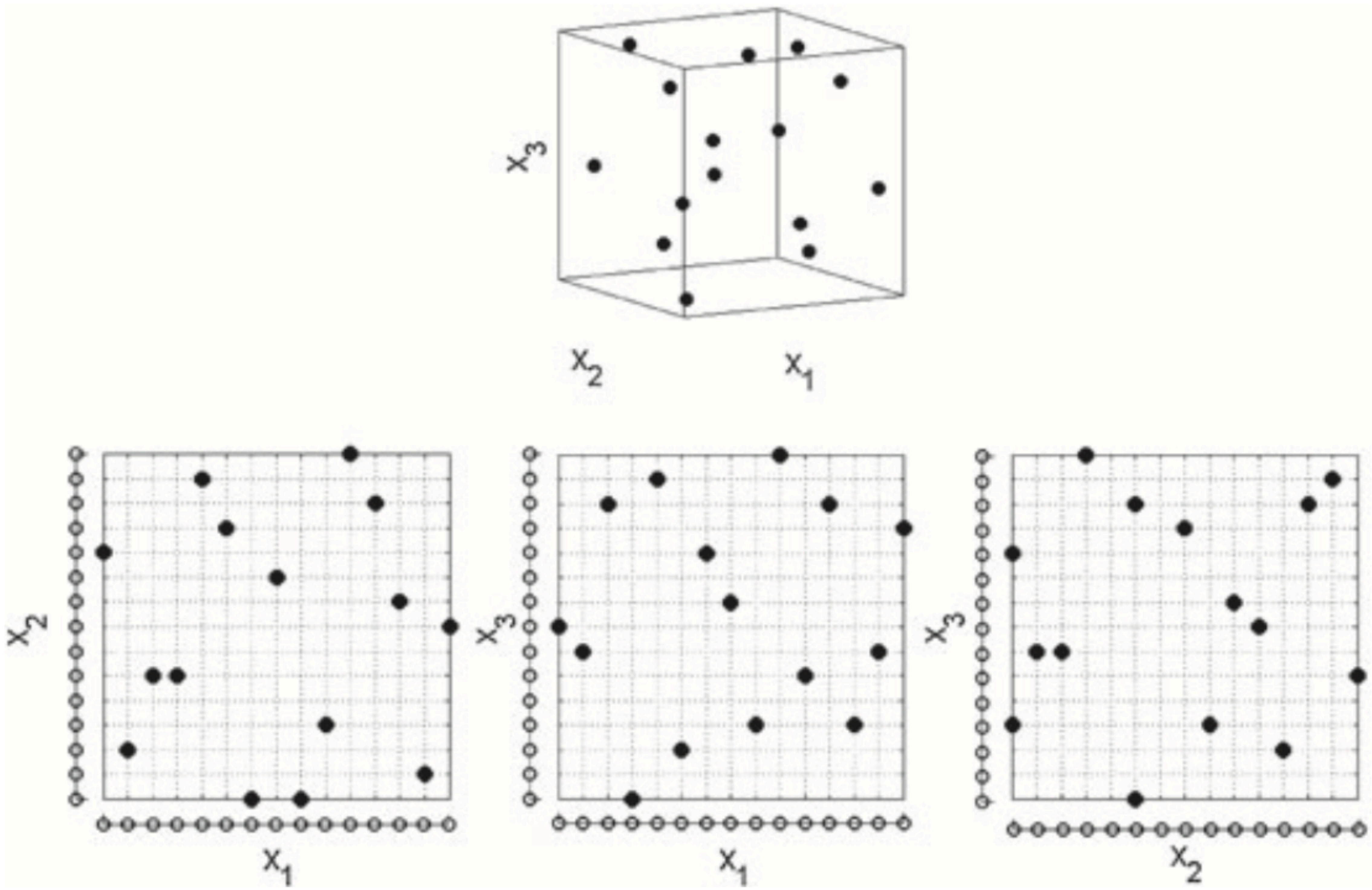
Random



Latin hypercube sampling

# DESIGN OF EXPERIMENTS

## LATIN HYPERCUBE SAMPLING (LHS) - 3D EXAMPLE (15 SAMPLES)



# DESIGN OF EXPERIMENTS

## WHY DOE IS USED AND COMMON APPLICATIONS (BEYOND OPTIMIZATION)

- DOEs can be used in general to:
  - Gain knowledge → *What are the most significant variables? What's their range?*
  - Increase understanding → *What's the relationship between variables and objective(s)?*

- DOE applies to various areas such as:
  - Estimation of the proper operating conditions of a system/process/product
  - Development of new products and processes
  - Enhancement of existing products and processes
  - Robustness testing of products and processes
  - Optimization of quality and performance of a product
  - Optimization of an existing manufacturing procedure (e.g. to reduce production cost)

# DESIGN OF EXPERIMENTS

## CONCLUDING REMARKS

- DOEs results in a principled set of experiments:
  - All variables are varied, systematically and independently
  - The number and type of variables and the sampling model specify the DOEs
  - DOEs defines the optimal number (and the best combinations) of samples

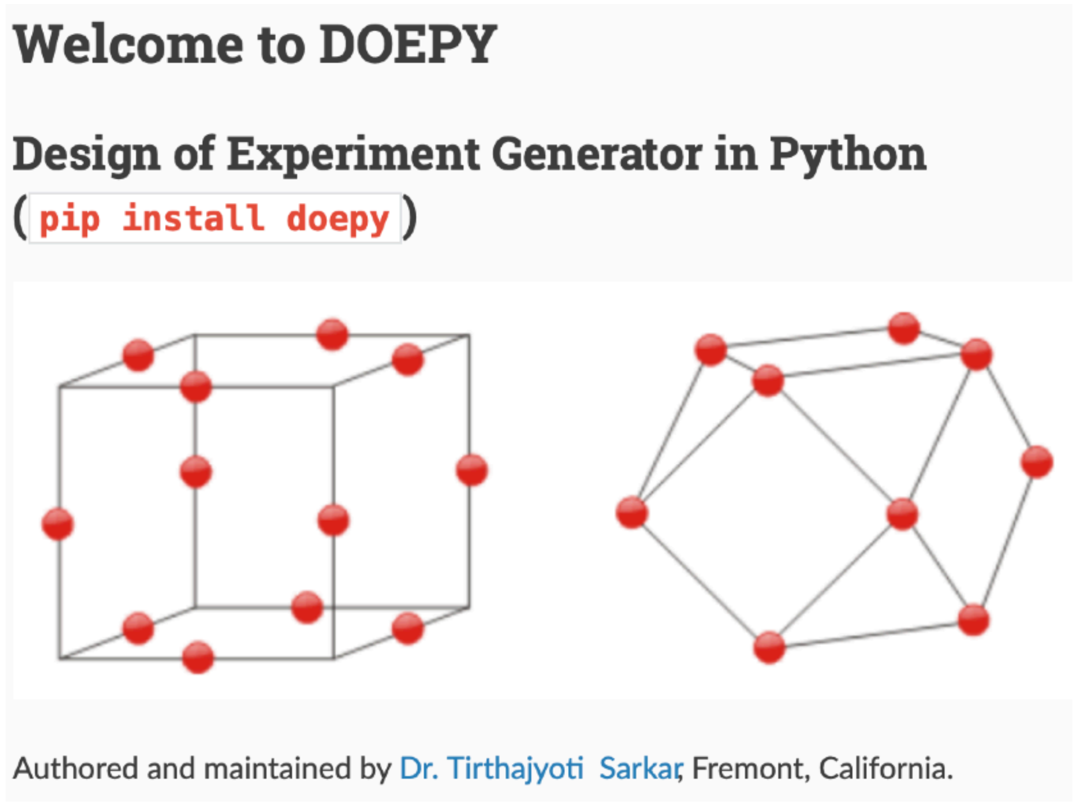
- DOE is used for three primary experimental objectives:
  - Screening: which variables are important, and what are their appropriate ranges?
  - Optimization: what are the optimal samples?
  - Robustness testing: how sensitive is a response to small variable changes?

- Advantages with DOEs:
  - Organized approach which defines experiments/samples in a rational manner
  - The influence of and interactions between all variables can be estimated
  - More precise information is acquired in fewer samples
  - Results are evaluated in the light of variability
  - Support for decision-making → response surfaces/contour plots
  - Seen effects and noise are separable and estimable probability analysis

# DESIGN OF EXPERIMENTS

## DOES IN PYTHON

- DOEPy https://doepy.readthedocs.io/en/latest/
- pyDOE https://pythonhosted.org/pyDOE/

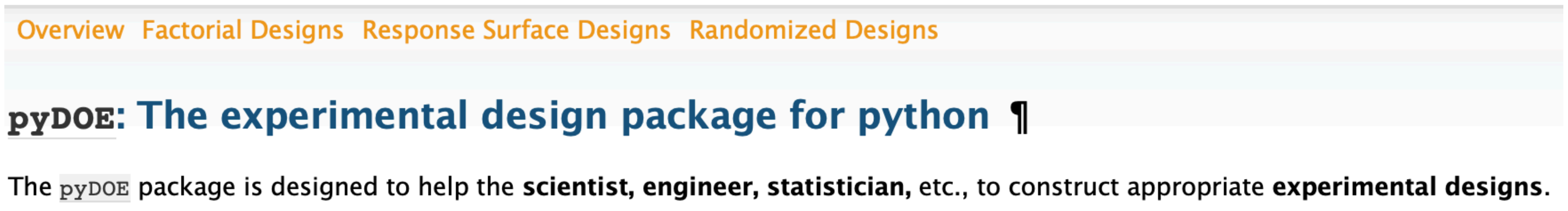


**Welcome to DOEPY**

**Design of Experiment Generator in Python**

(`pip install doepy`)

Authored and maintained by Dr. Tirthajyoti Sarkar, Fremont, California.



Overview Factorial Designs Response Surface Designs Randomized Designs

**`pyDOE`: The experimental design package for python ¶**

The `pyDOE` package is designed to help the **scientist, engineer, statistician,** etc., to construct appropriate **experimental designs**.

# DESIGN OF EXPERIMENTS

## FURTHER READING

- https://bookdown.org/rbg/surrogates/chap4.html
- https://online.stat.psu.edu/stat503/
- Martins, Joaquim RRA, and Andrew Ning. Engineering design optimization. Cambridge University Press, 2021. **(chapter 10)**
- Viana, Felipe AC. "A tutorial on Latin hypercube design of experiments." Quality and reliability engineering international 32.5 (2016): 1975-1985.

Questions?