Surrogate Optimization: Space-Filling Designs

BIOE 498/598 PJ

Spring 2022

From local to global optimization

- Steepest Ascent/RSM finds the optimal operating conditions in a **local** design space.
- It's possible a better operating condition exists far away.
- **Global optimization** searches the entire design region.

From local to global optimization

- Steepest Ascent/RSM finds the optimal operating conditions in a **local** design space.
- It's possible a better operating condition exists far away.
- **Global optimization** searches the entire design region.

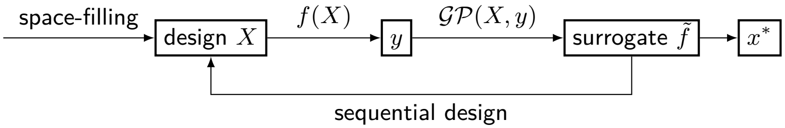
Method	**Search**	**# of samples**	**Sampling**
Steepest Ascent/RSM	local	10–100's	very expensive, noisy
Surrogate Optimization	global	100–1000's	moderately expensive
Reinforcement Learning	global	10,000+	very inexpensive

Surrogate Optimization

- Assume we are trying to optimize a function f that is **expensive** to evaluate.
- Instead, we use evaluations of f to build a surrogate model $\tilde{f}$ that is **cheap** to evaluate.
- We optimize $\tilde{f}$ to find good candidates for evaluation by f.

Surrogate Optimization

- Assume we are trying to optimize a function f that is **expensive** to evaluate.
- Instead, we use evaluations of f to build a surrogate model $\tilde{f}$ that is **cheap** to evaluate.
- We optimize $\tilde{f}$ to find good candidates for evaluation by f.



Why use surrogates?

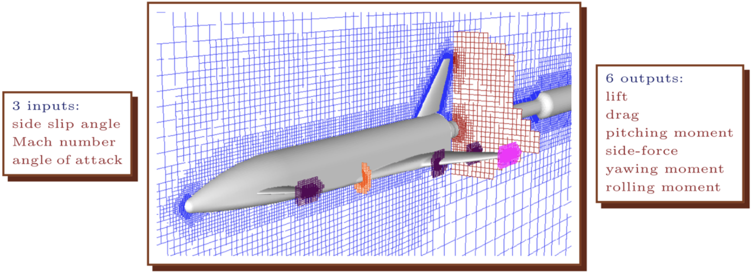


FIGURE 2.1: Drawing of the LGBB computational fluid dynamics computer model simulation. Adapted from Rogers et al. (2003); used with permission from the authors.

Surrogate optimization by many names

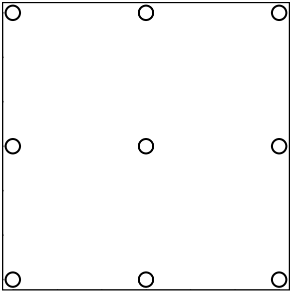
- **Computer experiments**, **emulation**, or **metamodeling** based on historical usage.
- **Kriging** from geostatistics (refers to prediction with GPR).
- **Nonparametric Bayesian optimization** to impress your manager.
- **Sequential design** in the DOE field.
- **Active learning** in the ML field.

Today: Space-Filling Designs

- Global optimization requires data from every part of the design space.
- We want to cover as much of the space as possible with the fewest number of points.
- **Space-Filling Designs** spread samples over a multidimensional space $[0,1]^k$.
  - Other design spaces can be rescaled to match this hypercube.
- Later we will use the surrogate to *intelligently* augment the initial design.

How about evenly-spaced designs (grids)?

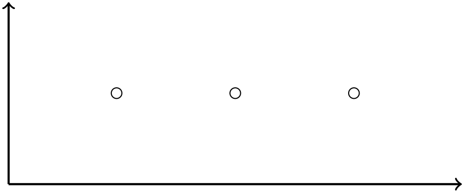
Evenly-spaced designs have two big drawbacks:

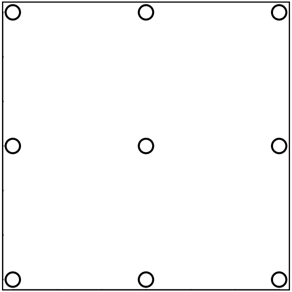


How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

1. Regular spacing can alias patterns in the response surface.

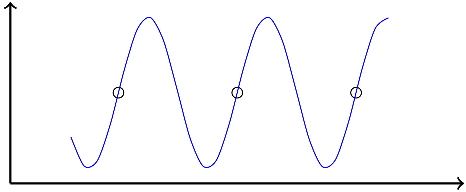


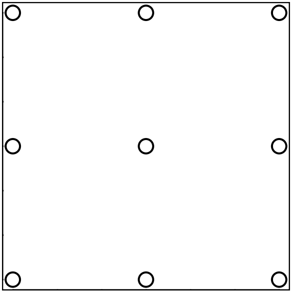


How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

1. Regular spacing can alias patterns in the response surface.

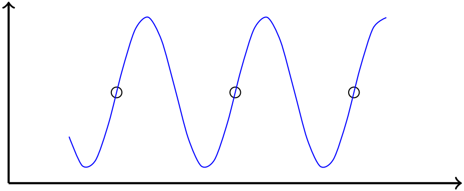




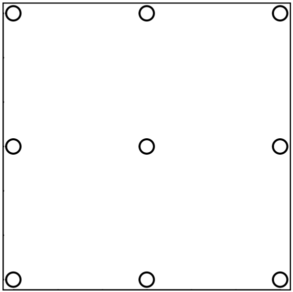
How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

1. Regular spacing can alias patterns in the response surface.



2. Regular designs have poor **projection spacing**. This is a problem because of effect sparsity!

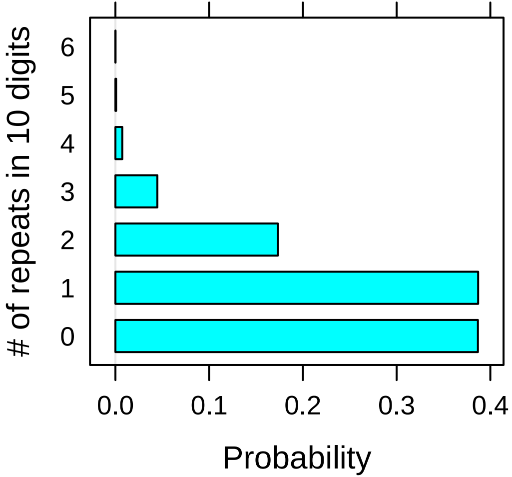


Why not random locations?

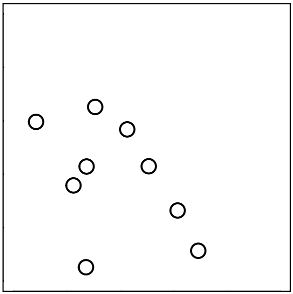
People often overestimate the randomness of random processes. Take our "random" string of digits as an example. How many repeated digits did you find in your sequence?

Why not random locations?

People often overestimate the randomness of random processes. Take our "random" string of digits as an example. How many repeated digits did you find in your sequence?



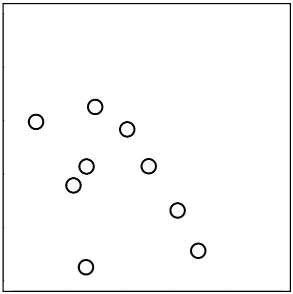
Random designs are "clumpy"



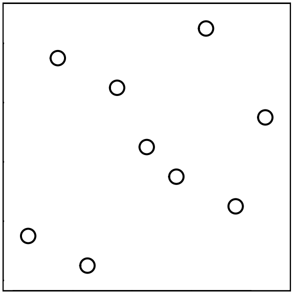
Random designs are "clumpy"

We need a **Space-Filling Design** that

1. Places points semi-randomly to avoid aliasing
2. Avoids "clumps" of points
3. Projects well onto lower dimensions



Latin Hypercube Designs



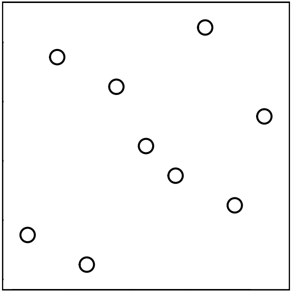
A Latin Hypercube Design (LHD) is a semi-random design that guarantees uniform projection.

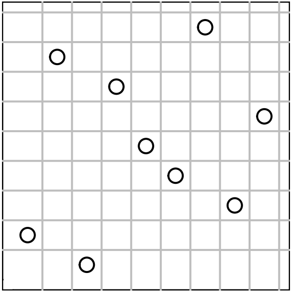
- Each dimension is divided into n intervals.
- Points are placed randomly, but only one point is allowed in each interval along each dimension.
- Points can be placed in the center or a random position in each "square".
- LHDs are like a simplified Sudoku puzzle!

Latin Hypercube Designs

A Latin Hypercube Design (LHD) is a semi-random design that guarantees uniform projection.

- Each dimension is divided into n intervals.
- Points are placed randomly, but only one point is allowed in each interval along each dimension.
- Points can be placed in the center or a random position in each "square".
- LHDs are like a simplified Sudoku puzzle!

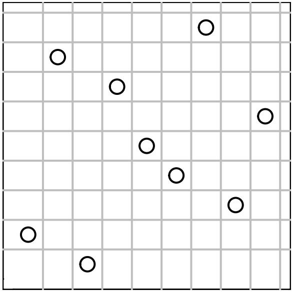




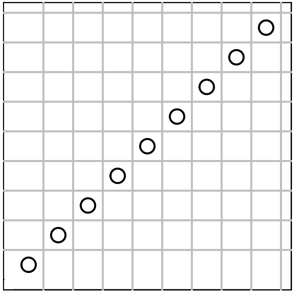
Building a LHD

1. The interval along the first dimension is simply $1 \ldots n$; there is no need to randomize.
2. For each subsequent dimension, select a random permutation of $\{1 \ldots n\}$

x_1	x_2
1	2
2	8
3	1
4	7
5	5
6	4
7	9
8	3
9	6

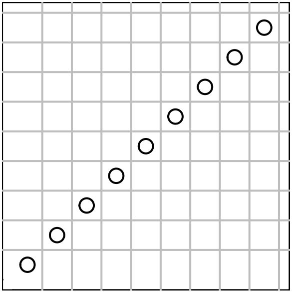


Beware of randomness (again)

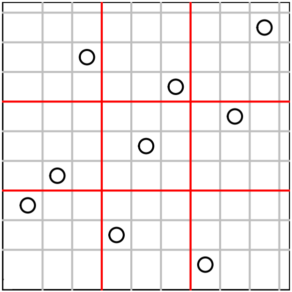


The above design is a LHD, but it is not very random. Rarely we can get permutations that do not space-fill.

Beware of randomness (again)



The above design is a LHD, but it is not very random. Rarely we can get permutations that do not space-fill.



One alternative is an Orthogonal Array LHD.

Another option: Maximin Designs

What we really want is to maximize the distance between the points in the final design. We can achieve this directly via optimization.

Another option: Maximin Designs

What we really want is to maximize the distance between the points in the final design. We can achieve this directly via optimization.

The *Euclidean distance* between any two points x and x' is

$$d(x, x') = \|x - x'\|^2 = \sum_{j=1}^{k} (x_j - x'_j)^2$$

Another option: Maximin Designs

What we really want is to maximize the distance between the points in the final design. We can achieve this directly via optimization.

The *Euclidean distance* between any two points x and x' is

$$d(x, x') = \|x - x'\|^2 = \sum_{j=1}^{k} (x_j - x'_j)^2$$

The *maximin* design matrix with n samples, called X_n is

$$\arg\max_{X_n} \min\{d(x, x'), \forall x \neq x'\}$$

Another option: Maximin Designs

What we really want is to maximize the distance between the points in the final design. We can achieve this directly via optimization.

The *Euclidean distance* between any two points x and x' is

$$d(x, x') = \|x - x'\|^2 = \sum_{j=1}^{k} (x_j - x'_j)^2$$

The *maximin* design matrix with n samples, called X_n is

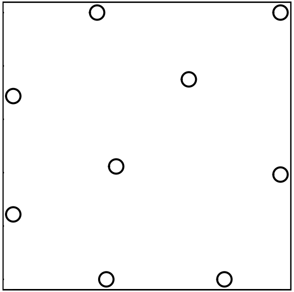
$$\arg\max_{X_n} \min\{d(x, x'), \forall x \neq x'\}$$

Computationally, we begin with a random set of n points and iteratively move points until a local optimum is found.

The `maximin` package

The `maximin` package creates sequential space-filling designs.

```
X1 <- maximin::maximin(
        n=9, p=2, T=100)
```



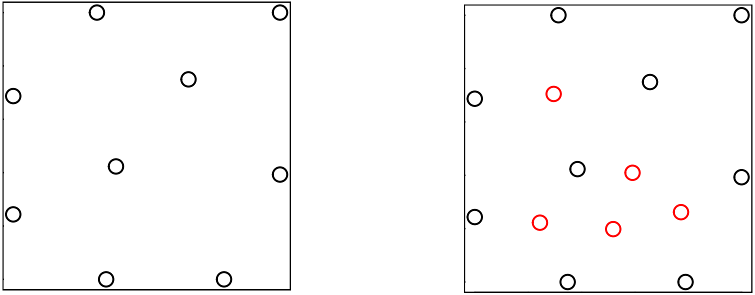
`n`=number of samples, `p`=number of dimensions, `T`=number of iterations for the optimizer.

The `maximin` package

The `maximin` package creates sequential space-filling designs.

```
X1 <- maximin::maximin(
        n=9, p=2, T=100)
```

```
X1 <- maximin::maximin(
        n=5, p=2, T=100,
         Xorig=X1$Xf)
```



`n`=number of samples, `p`=number of dimensions, `T`=number of iterations for the optimizer.

Summary

- Surrogate optimization begins with a space-filling design.
- Grid and random designs are not good.
- LHD and Maximin designs spread points globally and project well to lower dimensions.
- **Next time:** Building a surrogate model from an initial space-filling design.