Planning (Part 1)

Introduction to Automated Science

SLAS 2024

Example: Optimizing stem cell differentiation

Our goal is to improve the efficiency of differentiating ESCs into mature, insulin-producing beta cells.

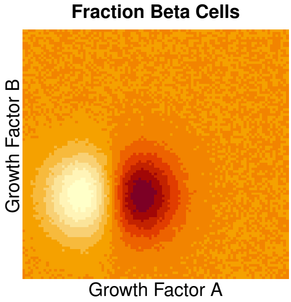
- **Factors**: [Growth Factor A] and [Growth Factor B], both added during differentiation.
- **Response**: Fraction of beta cells after 40 days [0.0–1.0].

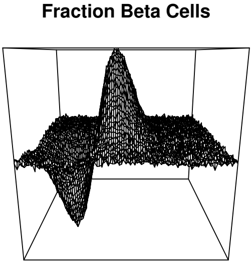
Example: Optimizing stem cell differentiation

Our goal is to improve the efficiency of differentiating ESCs into mature, insulin-producing beta cells.

- **Factors**: [Growth Factor A] and [Growth Factor B], both added during differentiation.
- **Response**: Fraction of beta cells after 40 days [0.0–1.0].

For illustration, pretend we know the “true” response surface:





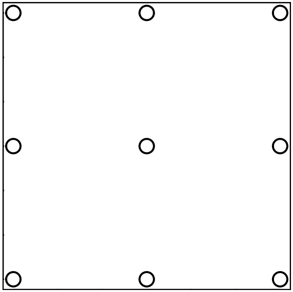
Starting with an initial design

We need an initial set of data to train our model, but we have no model to plan these experiments.

What is the best option for a starting “random” design?

How about evenly-spaced designs (grids)?

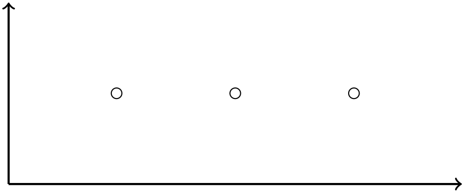
Evenly-spaced designs have two big drawbacks:

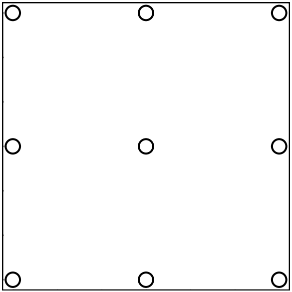


How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

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

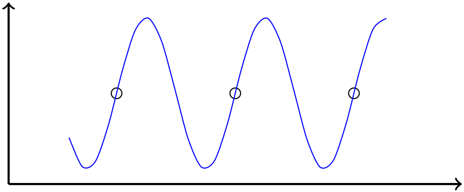


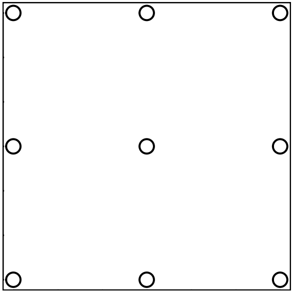


How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

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

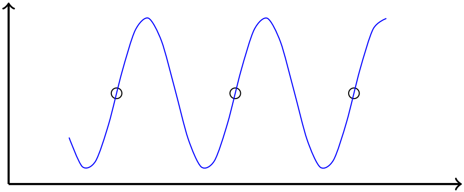




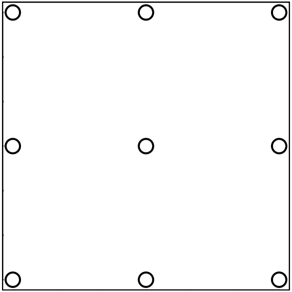
How about evenly-spaced designs (grids)?

Evenly-spaced designs have two big drawbacks:

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

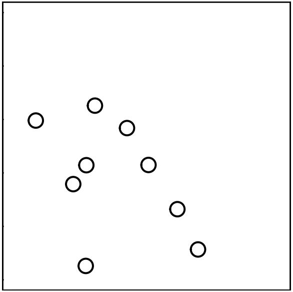


2. Regular designs have poor **projection spacing**. This is a problem because not all factors affect the resonse.



Why not random designs?

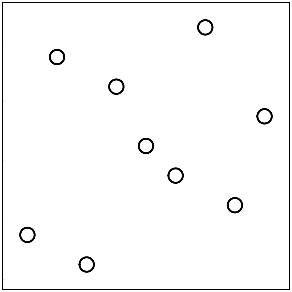
Random design are “clumpy”, especially in high dimensions.



Latin Hypercube Designs

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

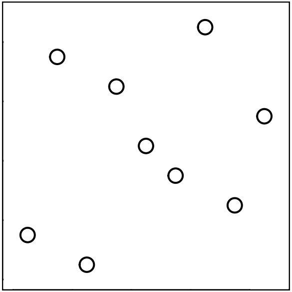
- Each dimension is divided into n intervals.
- Points are placed randomly, but only one point is allowed in each interval along each dimension.
- Points can be placed in the center or a random position in each "square".
- LHDs are like a simplified Sudoku puzzle!

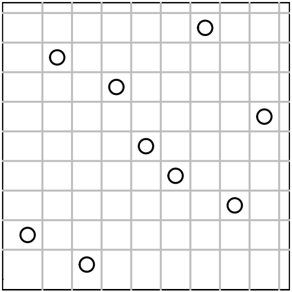


Latin Hypercube Designs

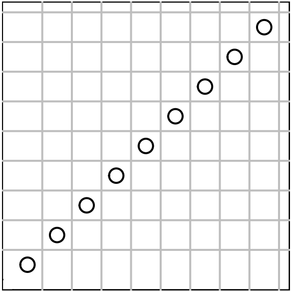
A Latin Hypercube Design (LHD) is a semi-random design that guarantees uniform projection.

- 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!



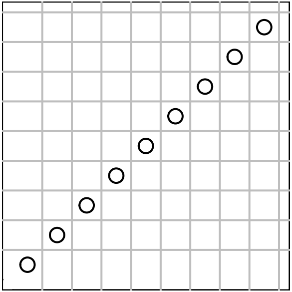


Beware of randomness (again)

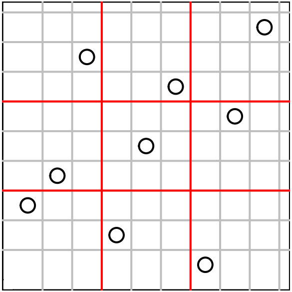


The above design is a LHD, but it is not very random. Rarely we can get permutations that do not space-fill.

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 minimum 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 minimum 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 minimum 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

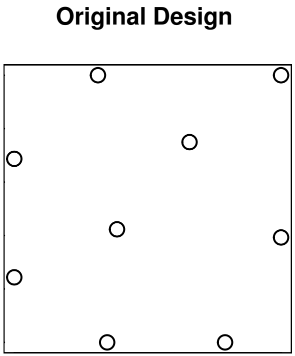
$$d(x, x') = \|x - x'\|^2 = \sum_{j=1}^{k} (x_j - x'_j)^2$$

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

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

Augmenting maximin designs

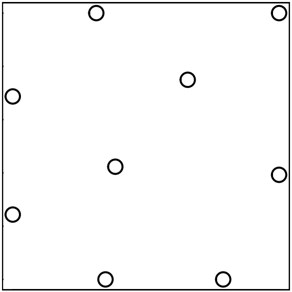
We can add more points to an existing design by spacing them apart from previous points.



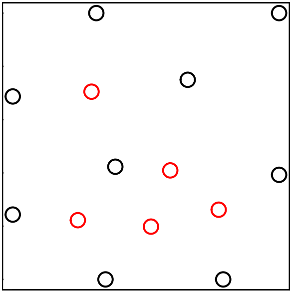
Augmenting maximin designs

We can add more points to an existing design by spacing them apart from previous points.

Original Design



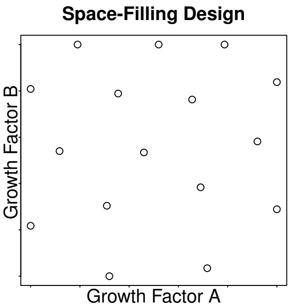
Augmented Design



Starting with an initial design

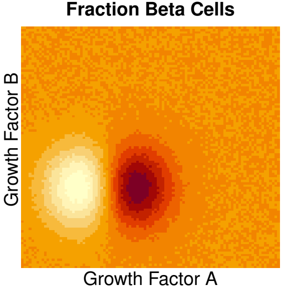
We need an initial set of data to train our model, but we have no model to plan these experiments.

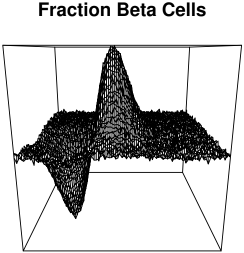
Let's move ahead with a space-filling maximin design.



Comparing our model after running the initial design

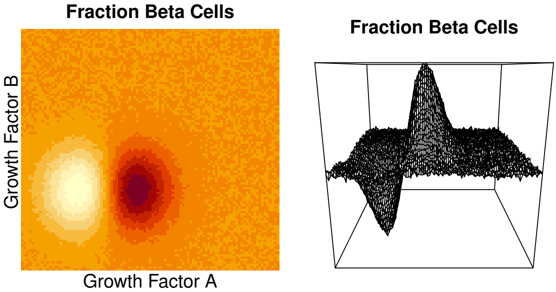
True Response



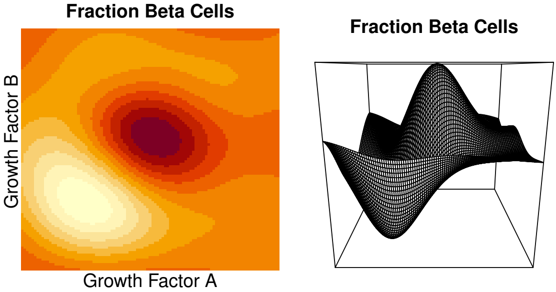


Comparing our model after running the initial design

True Response



Initial Model Prediction



Searching for the next run

We need to pick the next treatment (combination of Growth Factor A and Growth Factor B) to test. This should be the treatment **predicted** to give the best response.

Searching for the next run

We need to pick the next treatment (combination of Growth Factor A and Growth Factor B) to test. This should be the treatment **predicted** to give the best response.

Our model is a Gaussian Process Regression

$$\text{fraction beta cells} = \mathcal{GP}(\text{Growth Factor A, Growth Factor B})$$

trained on data from our initial design.

Searching for the next run

We need to pick the next treatment (combination of Growth Factor A and Growth Factor B) to test. This should be the treatment **predicted** to give the best response.

Our model is a Gaussian Process Regression

$$\text{fraction beta cells} = \mathcal{GP}(\text{Growth Factor A, Growth Factor B})$$

trained on data from our initial design.

Given an initial guess, a nonlinear optimizer like L-BFGS-B can find the nearest *local optimum*.

Searching for the next run

We need to pick the next treatment (combination of Growth Factor A and Growth Factor B) to test. This should be the treatment **predicted** to give the best response.

Our model is a Gaussian Process Regression

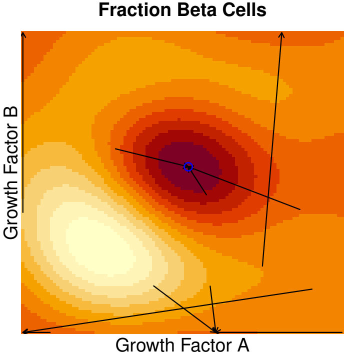
$$\text{fraction beta cells} = \mathcal{GP}(\text{Growth Factor A, Growth Factor B})$$

trained on data from our initial design.

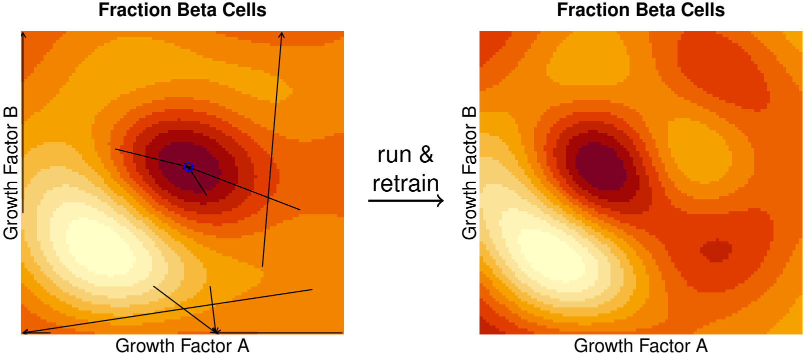
Given an initial guess, a nonlinear optimizer like L-BFGS-B can find the nearest *local optimum*.

To find the *global optimum*, we restart the optimizer many times at random points.

Searching for a global optimum by L-BFGS-B

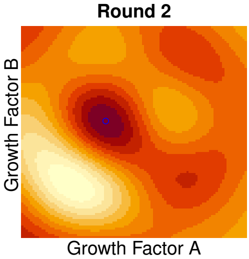


Searching for a global optimum by L-BFGS-B

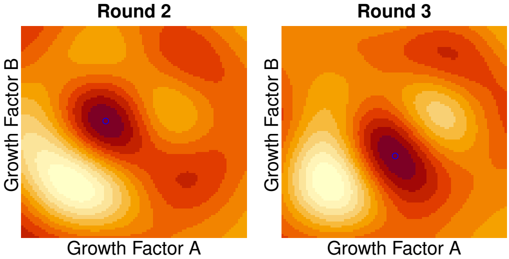


The true optimum was not where the model predicted. Retraining the model moved the model's estimate of the optimum for the next round.

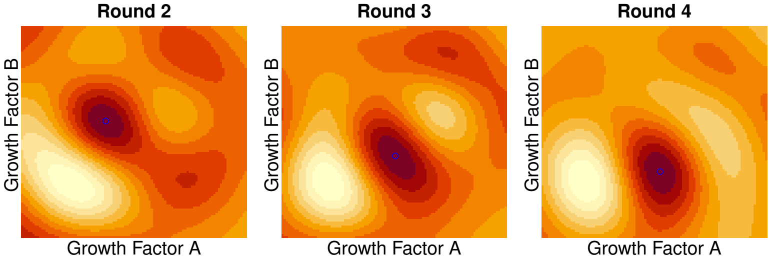
Sequential experiments and model updates



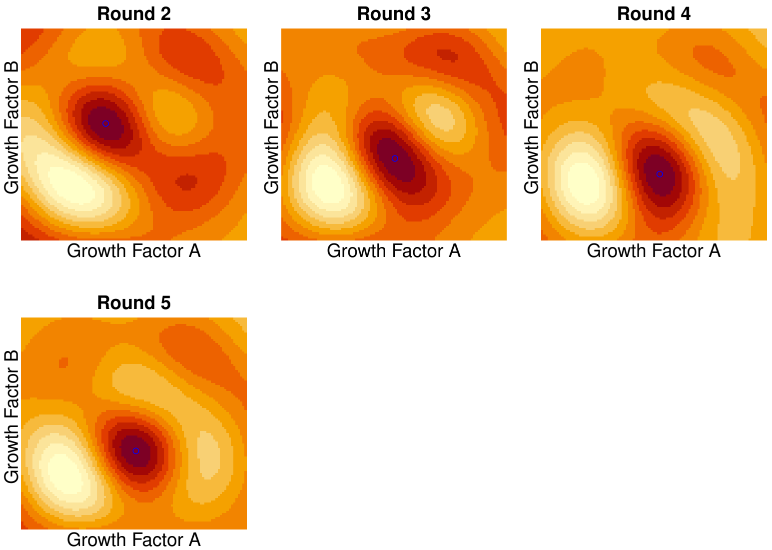
Sequential experiments and model updates



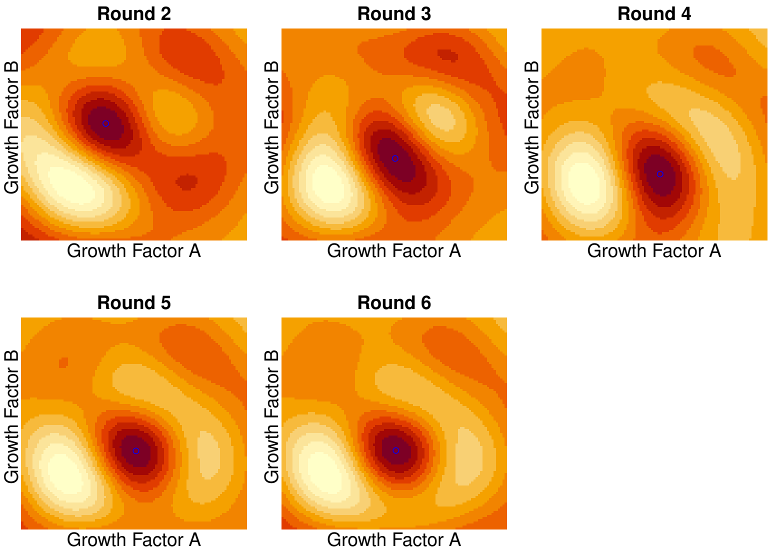
Sequential experiments and model updates



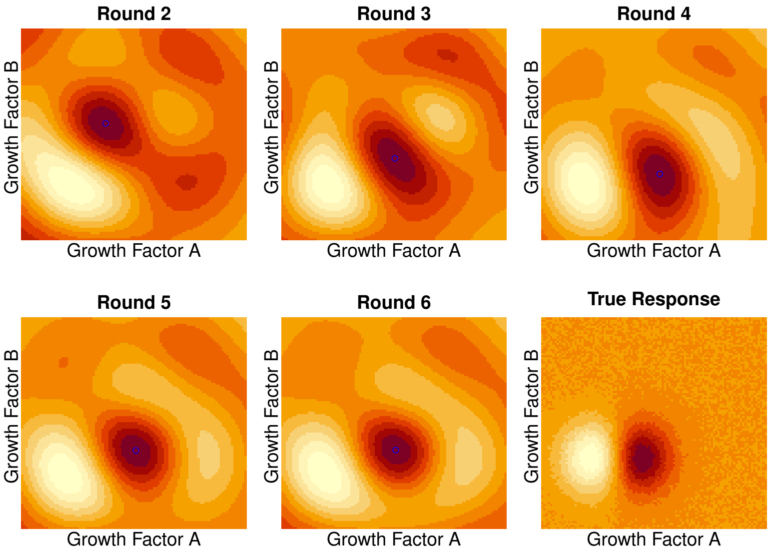
Sequential experiments and model updates



Sequential experiments and model updates



Sequential experiments and model updates



Leveraging the low cost of model predictions

Cost of our automated search:

	Initial design	16 runs
$+$	Sequential search	6 runs
	Real-world total	22 runs

Leveraging the low cost of model predictions

Cost of our automated search:

	Initial design	16 runs
+	Sequential search	6 runs
	Real-world total	22 runs

However, our nonlinear search used **3,235 model predictions** to select these 22 runs.

Exploitation vs. Exploration

The previous example used pure **exploitation**—using the model's knowledge to find the best predicted response.

Exploitation vs. Exploration

The previous example used pure **exploitation**—using the model's knowledge to find the best predicted response.

Models can also be improved by **exploration**—placing runs in regions where the model is most uncertain.

Summary

- Planning begins with a **space-filling design** when we lack prior knowledge.

Summary

- Planning begins with a **space-filling design** when we lack prior knowledge.
- Prior knowledge is used to find a new experiment at a better response.

Summary

- Planning begins with a **space-filling design** when we lack prior knowledge.
- Prior knowledge is used to find a new experiment at a better response.
- Restarting a local optimizer can find (approximately) global optima.

Summary

- Planning begins with a **space-filling design** when we lack prior knowledge.
- Prior knowledge is used to find a new experiment at a better response.
- Restarting a local optimizer can find (approximately) global optima.
- True global optimization requires **exploitation** and **exploration**.