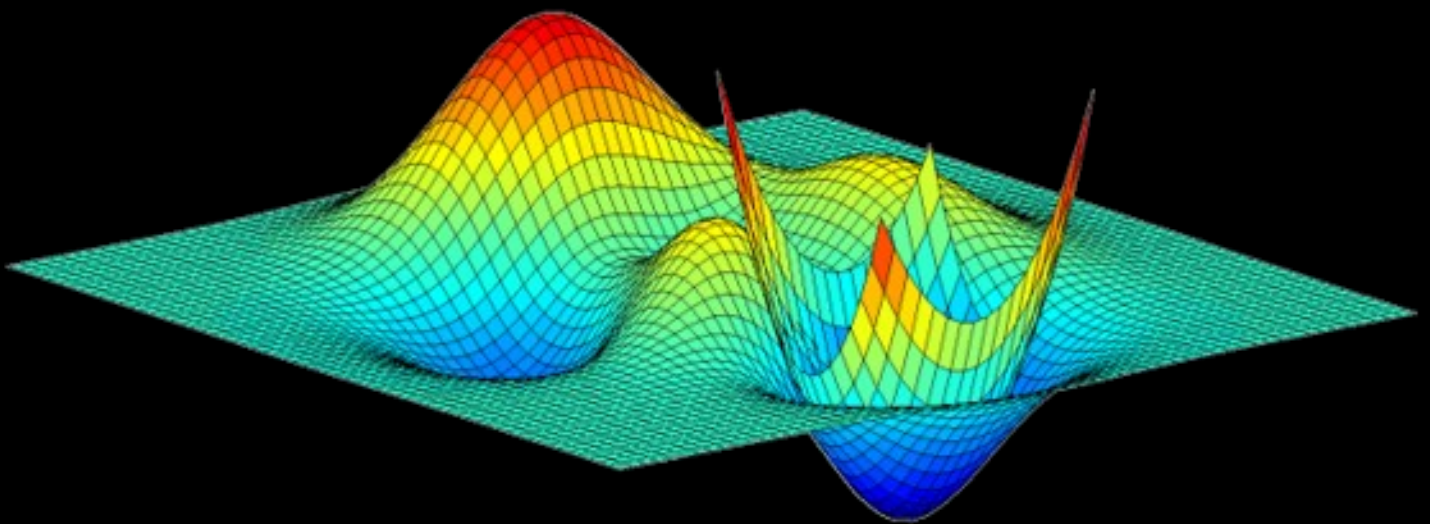


An Introduction to
Surrogate Modeling and
Response Surface
Methodology
in
Engineering Design
Optimization



Johan A. Persson

About the Author

Johan A. Persson, PhD is an assistant professor at Linköping University with both research and education that focus on Engineering Design Optimization. Most aspects of Engineering Design Optimization are of interest, but the focus is on how to enable optimization of engineering applications that include computationally expensive models.

About the Cover

A figure consisting of a surf plot of the Peaks function as well as a second order response surface of the optimum. This illustrates how surrogate models can replace the original engineering model to get almost instantly prediction with high accuracy.

An Introduction to Surrogate Modeling and Response Surface Methodology in Engineering Design Optimization

Johan Persson



Linköping University Electronic Press
Linköpings universitet, SE-581 83 Linköping, Sweden
Linköping 2026

First Edition: February 2026, Linköping University Electronic Press 2026

© Johan Alexander Persson 2026

Creative Commons Attribution 4.0 International (CC BY 4.0)
Some Rights Reserved.

Attribution Notice:

When reusing or distributing this work, you must provide appropriate credit to Johan Persson, provide a link to the license, and indicate if changes were made.

Cite like this:

Persson, J. A. (2026). *An Introduction to Surrogate Modeling and Response Surface Methodology in Engineering Design Optimization*. Linköping University Electronic Press.
<https://doi.org/10.3384/9789181185201>

ISBN 978-91-8118-520-1 (PDF)

<https://doi.org/10.3384/9789181185201>

Preface

The first and foremost aim with this book is to serve as a guide and reference for engineers that want to use surrogate modeling techniques when they design or develop products, systems and components.

Additionally, the book constitutes part of the literature for the courses in Design Optimization and Multidisciplinary Design Optimization at Linköping University.

It is the intention of the author that the content should serve as an easily understood introduction to surrogate modeling. The more advanced details are briefly mentioned but can be found in the scientific papers referenced in this book.

There exist numerous books and review articles regarding surrogate modeling, e.g. Forrester, Sobester, and Keane, [2008](#) and Krawczyk and Arabas, [2025](#), which means that all methods presented here can be found elsewhere. The two main advantages of this book are

- Many methods are exemplified using numerical examples
- Some practical guidelines recommended by the author are presented

The book has been reviewed by a few researchers at Linköping University, but it might still contain errors or mistakes. Generative AI has been used to assess the content as well as to suggest improvements to it and the language. All figures have been created by the author.

Contents

1	Introduction	3
1.1	Notations	4
1.2	Demonstration Problems	5
1.2.1	Problem 1 - A linear problem	5
1.2.2	Problem 2 - Rosenbrock's Banana	5
1.2.3	Problem 3 - Paper Airplane	6
2	Surrogate Model Types	11
2.1	Performance Example of Different Surrogate Models	14
3	Polynomial Response Surfaces (PRS)	19
3.1	Advantages and Disadvantages with PRS	20
3.2	Example 1	20
3.3	Example 2	22
4	Interpolating Surrogate Models	25
4.1	Inverse Distance Weighting	25
4.1.1	Advantages and Disadvantages	26
4.1.2	Example 1	26
4.2	Radial Basis Functions	27
4.2.1	Advantages and Disadvantages with RBF	29
4.2.2	Example 1	30
4.3	Kriging	32
4.3.1	Kriging with Trends	34
4.3.2	Covariance Functions	34
4.3.3	The Maximum Likelihood Estimation Training Approach	35
4.3.4	Prediction Accuracy	37
4.3.5	Advantages and Disadvantages with Kriging	37
4.3.6	Example 2: Predicting a Point on the Rosenbrock Function	38

5	Other Surrogate Models	43
5.1	Decision Trees for Regression	43
5.1.1	Mathematical Formulation	43
5.1.2	Training Criterion	44
5.1.3	Pruning and Regularization	44
5.1.4	Advantages and Disadvantages	44
5.2	Random Forests for Regression	45
5.2.1	Bagging and De-correlation	45
5.2.2	The Ensemble Prediction	45
5.2.3	Out-of-Bag (OOB) Error	46
5.2.4	Advantages and Disadvantages	46
5.3	High-Dimensional Model Representation (HDMR)	46
5.4	Neural Networks	50
5.4.1	The Neuron: Core Building Block	51
5.4.2	Making Predictions by Forward Propagation	51
5.4.3	Activation Functions	51
5.4.4	Overall Workflow for Surrogate Modelling	53
5.4.5	Data Pre-processing and Normalization	53
5.4.6	Dataset Creation and Management	54
5.4.7	Architecture Algorithm Selection	54
5.4.8	Training - Backpropagation and Early Stopping	55
5.4.9	Performance Evaluation Verification	58
5.4.10	Data Augmentation	58
5.4.11	For CFD: Physics-Informed Neural Networks (PINNs)	58
5.4.12	Advantages and Disadvantages	59
5.5	Polynomial Chaos Expansion	59
5.5.1	Training	60
5.5.2	Advantages and Disadvantages	60
5.6	Support Vector Regression (SVR)	61
5.6.1	Training Methodology and Optimization	61
5.6.2	The Kernel Trick and Common Functions	62
5.6.3	Advantages and Disadvantages	63
6	Design of Experiments	65
6.1	One-Shot Sampling Methods	65
6.1.1	Classical Approaches	66
6.1.2	Space-Filling Designs	70
6.2	Sequential or Adaptive Sampling	74
7	Assessing the Accuracy of a Surrogate Model	75
7.1	Visualizing Accuracy	78
7.1.1	Plot Estimations and True Values	78
7.1.2	Plot Estimations VS True Value (a QQ plot)	79
7.1.3	Plot Residuals	79

7.1.4	Dolan-Moré Performance Profile Plot	79
7.2	Accuracy Measures	81
7.2.1	Mean Squared Error (MSE)	83
7.2.2	Root Mean Squared Error (RMSE)	83
7.2.3	Normalized Root Mean Squared Error (NRSME)	83
7.2.4	R Squared	83
7.2.5	Relative Average Absolute Error (RAAE)	84
7.2.6	Relative Maximum Absolute Error (RMAE)	84
7.2.7	Cross-Validated Q^2	85
7.2.8	Example of Accuracy Measures	85
7.3	Assessment Strategies	86
7.3.1	Iterative Methods	86
7.3.2	Cross-Validation	87
7.3.3	Resampling Methods	88
8	Practical Guidelines	91
8.1	How to Utilize the Dataset	91
8.2	Selecting Surrogate Model	92
8.2.1	Guidelines Based on Dimensionality and Sample Size	92
8.2.2	Comparison Summary	92
8.3	Selecting Sampling Plan	93
8.3.1	Aligning Sampling Strategy with Model Choice	93
8.3.2	Recommended Pairings	94
8.4	Sampling Strategies	94
8.4.1	Accuracy vs. Complexity	95
8.4.2	Global vs. Local Models	95
8.4.3	Model Interpolation and Extrapolation	95
8.5	Handling Training Data before SM Training	95
8.5.1	Reveal Dependencies and Trends	96
8.5.2	Handling of Outliers	98
8.5.3	Removing trends?	99
8.5.4	Normalization of the Training Data	100
8.6	Using Multiple Surrogate Models for the Same Entity	100
8.6.1	Ensembles of Surrogate Models	101
8.6.2	Two-step Surrogate Model Estimations	102
8.7	Using one SM for each entity	102
9	Optimization with Surrogate Models	103
9.1	More Advanced Surrogate Based Optimization	103

Chapter 1

Introduction

Surrogate Models (SMs) are mathematical models that are used to map how an entity varies when one or more other entities are varied. Therefore, they are essential techniques in engineering design optimization, particularly when dealing with computationally expensive simulations, experimental data, or complex problems. Krawczyk and Arabas (2025) divides the usage of SMs into three categories:

- Regression - Estimate the value of a new design
- Ranking - Is design A better or worse than design B?
- Classification - Which areas of the design space are promising or not?

The focus in this work is on how to use SMs for regression (to estimate the value of a new design).

An example is the first demonstration problem (Eq. 1.1), where \hat{y} is modeled as a function of x_1 and x_2 . This yields an estimated value of y for any given values of x_1 and x_2 .

$$\hat{y} = 2x_1 - x_2 + 1 \tag{1.1}$$

The key benefit of using SMs is that they are computationally efficient models. Consequently, they are excellent choices when many designs need to be evaluated in a short time, for example in optimization or to visualize how their output varies in the design space. Additionally, no information about the system is needed to perform surrogate modelling since only the training data is needed to create an SM. However, underlying knowledge of the entity that should be modeled could be useful to select an appropriate surrogate model or normalize the data.

Additionally, SMs can be used to combine data from different models / experiments AND handle data from models with different fidelity.

SMs are fitted from (or trained with) test data so that they reanimate the data as well as possible. This means that the quality of the test data

affects the accuracy of the SM predictions. The reasoning of which experiments/samples should be performed to yield the test data is explained in Chapter 6.

It cannot be stressed enough that ALL predictions or suggested optima that will be used/chosen by an engineer or designer should be verified by conducting experiments or simulating the original model.

Viana et al. (2014) conducted a survey on the history of surrogate modeling in modeling and simulation. Interested readers are encouraged to read it. A more recent review of surrogate models with a focus on their use in optimization has been written by Krawczyk and Arabas (2025).

1.1 Notations

The authors have attempted to use a common notation for all entities that are presented in this work. However, some terms are used interchangeably, e.g.

- Design/Point to describe a location in the design space
- Design Variable/Parameter to describe x_1, x_2, \dots, x_n

The methods are used to handle data from experiments and/or simulations of systems. Both are referred to as experiments in this section.

The following symbols are used in this work.

- m - number of experiments/samples
- n - number of design variables
- x_i - design variable i
- $x_i^{(j)}$ - value of design variable x_i for experiment/point j
- y_k - output k
- $y^{(j)}$ - output value of experiment/point j
- $\hat{y}^{(j)}$ - estimated output value of experiment/point j

It is recommended to store the data from each system experiment similar to Table 1.1. Each row in the table corresponds to one experiment, and the value of each design variable is found in that row. Each column corresponds to a design variable or output.

Experiment one is for example performed with an x_1 value of 1.1 and results in an output of $y=1.2$.

The table is then divided into two matrices, one with the design variables and one with the output as shown in Eq. 1.2. These matrices are used to create SMs.

Table 1.1: Example of how to store data from experiments

No.	x_1	x_2	x_3	y
1	1.1	2.3	0.1	1.2
2	0.7	2.5	1.1	0.9
3	1.5	1.5	1.5	1.7
4	0.9	1.9	0.8	1.4

$$\begin{aligned}
 X &= \begin{bmatrix} 1.1 & 2.3 & 0.1 \\ 0.7 & 2.5 & 1.1 \\ 1.5 & 1.5 & 1.5 \\ 0.9 & 1.9 & 0.8 \end{bmatrix} \\
 Y &= [1.2 \quad 0.9 \quad 1.7 \quad 1.4]^T
 \end{aligned} \tag{1.2}$$

1.2 Demonstration Problems

The methods for SMs are demonstrated for three problems - two mathematical functions and one engineering application. Their equations are presented here, but the equations are considered unknown in the following chapters.

1.2.1 Problem 1 - A linear problem

The function is shown in Eq. 1.3 and is a simple linear combination of x_1 and x_2 .

$$\begin{aligned}
 y(\mathbf{x}) &= 2x_1 - x_2 + 1 \\
 -1 &\leq x_1, x_2 \leq 1
 \end{aligned} \tag{1.3}$$

Some example training data from this function that will be used in the example calculations are presented in Table 1.2 and Figure 1.1.

1.2.2 Problem 2 - Rosenbrock's Banana

Equation 1.4 describes a polynomial that consists of a flat, banana-shaped, valley with steep slopes (Rosenbrock (1960) as displayed in Figure 1.2.

$$\begin{aligned}
 y(\mathbf{x}) &= 100(x_1^2 - x_2)^2 + (1 - x_1)^2 \\
 -2 &\leq x_1, x_2 \leq 2
 \end{aligned} \tag{1.4}$$

The samples used for this example are presented in Table 1.3 and Figure 1.3.

Table 1.2: Samples drawn from the linear function by using Latin Hypercube Sampling

x_1	x_2	y
-2	-1.6	-1.4
-1.6	1.2	-3.4
-1.2	-0.4	-1
-0.8	0.4	-1
-0.4	1.6	-1.4
0	-0.8	1.8
0.4	-2	3.8
0.8	2	0.6
1.2	0	3.4
1.6	0.8	3.4
2	-1.2	6.2

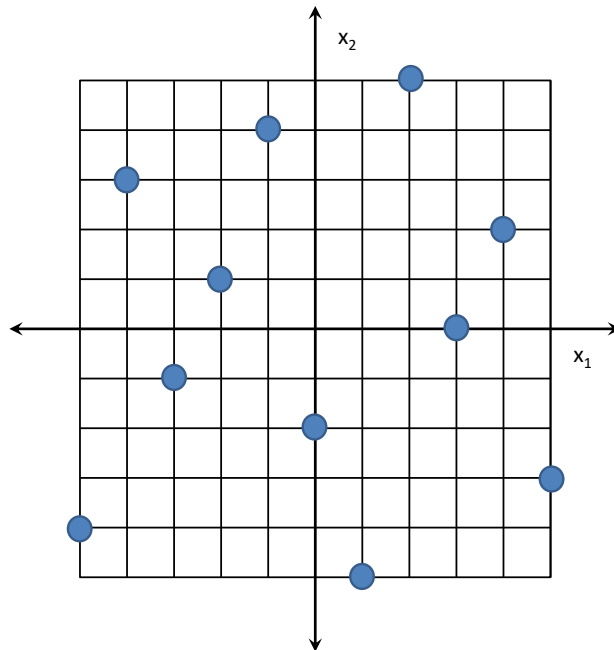


Figure 1.1: Example of a Latin Hypercube Sampling for a problem with two design variables

1.2.3 Problem 3 - Paper Airplane

The task is to design a paper airplane so it flies as far as possible by placing paper clips along the bottom edge. The first design variable is the position of the paper clips defined as the distance in mm from the front tip. The

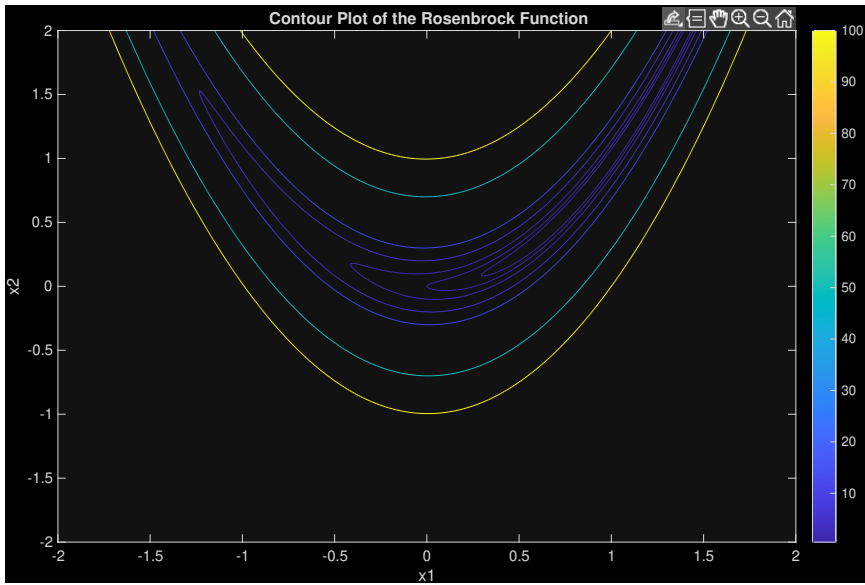


Figure 1.2: A contour plot of the Rosenbrock function

Table 1.3: Samples drawn from Rosenbrock’s Banana using Latin Hypercube Sampling

x1	x2	y
-2	2	409
-1.6	-1.6	1737.32
-1.2	0.8	45.8
-0.8	-0.4	111.4
-0.4	1.2	110.12
0	0	1
0.4	-1.2	185.32
0.8	0.4	5.8
1.2	-0.8	501.8
1.6	1.6	92.52
2	-2	3601

second design variable is the number of paper clips that are placed on the paper airplane. This is presented as an optimization problem in Eq. 1.5.

$$\begin{aligned}
 \max y(\mathbf{x}) &= \text{flightLength}(x) \\
 0 &\leq x_1 \leq 297 \\
 1 &\leq x_2 \leq 5
 \end{aligned}
 \tag{1.5}$$

Several throws with each paper airplane were performed and the median

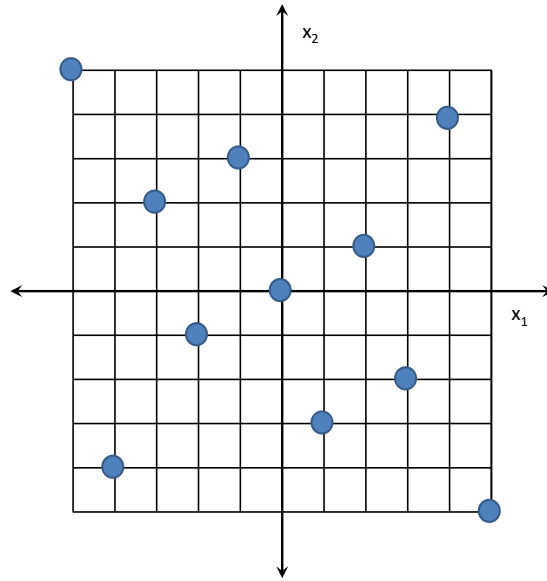


Figure 1.3: Example of a Latin Hypercube Sampling for a problem with two design variables

values were chosen as the flight length of that paperclip configuration. This was done to reduce the variability in the throws since it is difficult to throw the paper airplane with the same force and angle every time. This resulted in the dataset in Table 1.4. The x_1 , x_2 and y columns are used as the training data.

Table 1.4: The paper airplane dataset. x_1 is the position of the paperclips, x_2 is the number of paperclips. y is the median of the ten throws of each paper airplane configuration.

x_1	x_2	y	Throw									
			1	2	3	4	5	6	7	8	9	10
0	4	10.755	12.52	10.45	9.04	9.2	10.3	13.41	10.2	11.4	14.8	11.06
30	1	10	12.8	9.15	8.3	10.86	9.38	10.3	9.5	14.15	9.7	11.05
60	2	11.115	9.8	9.5	10.85	10.05	12.73	10.95	11.28	13.5	13.85	13.82
90	5	12.375	12.7	12.3	12.12	13.24	8.1	13.97	14.58	7.65	7.5	12.45
120	3	11.315	10.97	12.95	11.3	13.3	10.27	12.65	9.7	11.3	11.33	11.33
150	4	10.53	11	9.1	14.05	10.06	9.2	17	8.95	9.8	14	12.1
180	3	10.025	9.92	9.95	16	7	13.18	7.5	13.95	6.5	12.4	10.1
210	1	9.775	9.43	12.8	10.09	8.4	9.25	10.33	10.95	8.85	9.8	9.75
240	2	8.86	9.3	9.3	8.37	9.4	6.63	10.15	9.95	8.42	8.05	5.3
270	5	5.625	5.45	3.85	5.3	4.05	6.15	5.4	5.8	6.8	6.23	8.5

Chapter 2

Surrogate Model Types

Numerous SM types have been proposed, and this chapter summarizes some of the most popular. Each method is described in more detail in the following chapters.

Here, SMs are classified into three categories:

- Regression
- Interpolation
- Other SMs

Regression and Interpolation are exemplified in figure 2.1, where the stars represent training data, whereas the lines represent the corresponding surrogate model estimations.

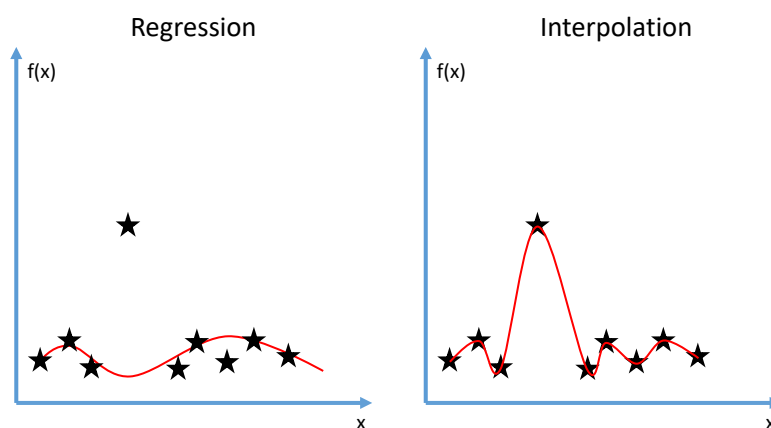


Figure 2.1: Example of regression and interpolation.

Regression models are usually used to capture the overall behavior or trend of the function. They smooth the data, which also make them suitable

for noisy data. This smoothing makes them unsuitable for modeling multimodal functions since the regression model might not capture the overall behavior. The function will instead be smoothed by the regression.

Interpolating models estimate the value of a point based on the values of the closest points. Consequently, the estimated value of a point whose value is already known will be that value, since the distance is zero. This means that pure interpolation is not recommended for noisy functions.

The advantages and disadvantages of some common SM types are presented in Table 2.1. Authors such as Krawczyk and Arabas, 2025 have listed additional advantages and disadvantages.

Alizadeh, Allen, and Mistree (2020) recommends to use Support Vector Machines for problems with many design variables, Polynomial Response Surfaces if little computation time is available and kriging when high accuracy is needed.

This is presented in section 8.2 but a short summary is: PRS is a good choice if the output is expected to follow a polynomial behavior. Otherwise, Kriging is recommended for datasets with fewer than 20 design variables. SVR is recommended for problems with many design variables but few samples in the dataset, whereas Neural Networks are recommended if hundreds of training samples are available.

Table 2.1: Overview of some common Surrogate Modelling techniques.

Method	Advantages	Disadvantages
Polynomial Response Surfaces	Easy to understand Smooths noisy functions	Needs extremely many samples to handle problems with many variables
Pure Interpolation	Good accuracy close to known points	Does not require training Cannot suggest values that are higher (or lower) than the highest (lowest) point
Radial Basis Functions	Differentiable	Estimates constant values far from training points
Kriging	Good global accuracy The prediction uncertainty can be extracted	Have trouble with samples that are too close to each other The function should be stationary (have the same order of magnitude in the whole design space)
Neural Networks	Good for modeling non-linear behavior	Long training times

2.1 Performance Example of Different Surrogate Models

The performances of some of the more commonly used SMs are demonstrated here for the second demonstration problem, Rosenbrock’s banana 1.4.

The graphs in figure 2.2 are based on a grid with 80 levels for both x_1 and x_2 . The different SMs have been created from a training set consisting of 100 designs created using a LHS. Pure interpolation uses the five closest training points to estimate the values and the DACE toolbox (Lophaven, Nielsen, and Söndergaard (2005)) has been used for kriging.

It can be noted that Kriging performs very well, whereas the second-order polynomial response surface model can not capture the correct behavior of the Rosenbrock’s banana function.

It is also possible to compare the performance using accuracy measures on the estimated values. One example of this is Table 7.3, where the 80x80 points used to create contour plots in figure 2.2 have been compared to the true values. According to all the presented accuracy measures, the Kriging model has the best accuracy, whereas the second-order polynomial response has the worst. This is not surprising, since the accuracy measures are based on the same data as the contour plots.

Table 2.2: Samples drawn from Rosenbrock’s Banana using Latin Hypercube Sampling. The estimations are based on a training set with 100 samples

x_1	x_2	y_{True}	PRS	PI	RBF	Krig	NN
-2	-1.6	3145	1406.6	1796.3	1386.7	3145.0	2988.1
-1.6	1.2	191.72	412.4	253.9	276.7	191.7	179.9
-1.2	-0.4	343.4	381.9	309.8	341.2	343.4	342.0
-0.8	0.4	9	-10.5	6.1	3.0	9.0	-3.7
-0.4	1.6	209.32	-372.2	198.8	210.7	209.3	202.6
0	-0.8	65	45.4	92.5	66.3	65.0	74.9
0.4	-2	466.92	334.2	374.0	333.2	466.9	468.8
0.8	2	185	-226.5	101.1	29.2	185.0	169.5
1.2	0	207.4	412.8	239.3	90.0	207.4	222.9
1.6	0.8	310.12	648.3	375.2	317.4	310.1	318.0
2	-1.2	2705	1482.0	2067.1	1344.8	2705.0	2703.3
		MAE	448	218	330	0.0028	21.98
		NRMSEmean	0.970	0.635	0.947	8.18E-06	0.068
		R2	0.570	0.693	0.659	0.744	0.795
		RAAE	1.101	0.264	0.357	1.51E-05	0.172

It was quite easy to create surrogate models with reasonable accuracy when 100 samples are used. With 20 samples, as shown in Figure 2.3, it is

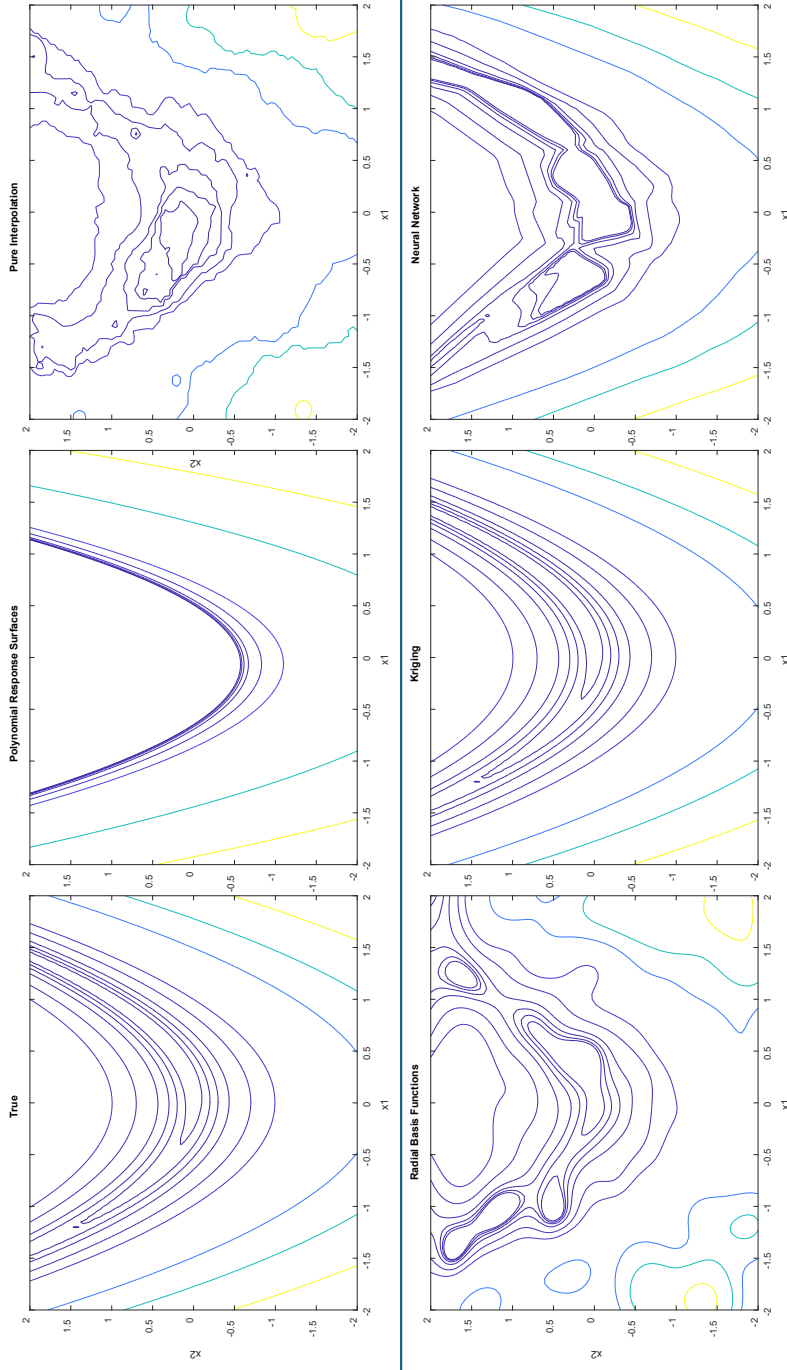


Figure 2.2: Contour plots for Rosenbrock's Banana, where the values for each contour plot have been estimated using a surrogate model trained with 100 samples. The value levels are 2, 5, 10, 20, 50, 100, 500, 1000 and 2000 where yellow corresponds to higher values 2000.

much more difficult. PRS fails dramatically, whereas the rest find the valley somewhat. NN can be considered the only model that is close to finding the optimum.

Rosenbrock Function - Surrogate Model Comparison
 Training samples: 30

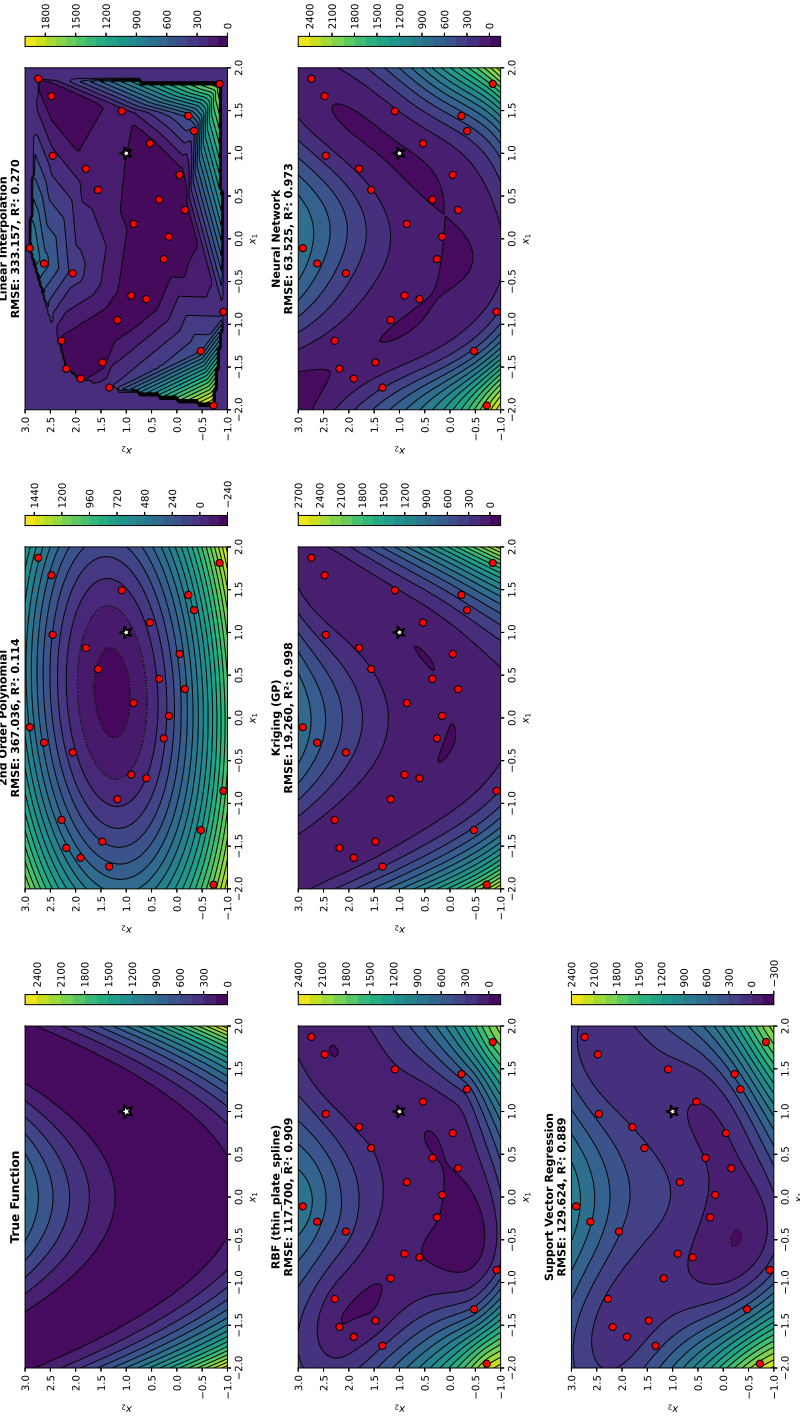


Figure 2.3: Contour plots for Rosenbrock's Banana, where the values for each contour plot have been estimated using a surrogate model. The red dots represent the training data whereas the white star represents the optimum.

Chapter 3

Polynomial Response Surfaces (PRS)

PRS (also known as response surfaces) try to match the test data with a polynomial of a given degree. A second-order polynomial with n number of variables is exemplified in Eq. 3.1. The terms with coefficients β_{ij} are called combination terms if $i \neq j$ since they represent the change due to the combination effect of the design variables i and j .

$$\hat{y}(\mathbf{x}) = \beta_0 + \sum_{i=1}^n \beta_i x_i + \sum_{i=1}^n \sum_{j=1}^n \beta_{ij} x_i x_j \quad (3.1)$$

Eq. 3.2 exemplifies a second-order polynomial with two design variables.

$$\hat{y}(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 \quad (3.2)$$

The process of determining the coefficients can be done by setting up the equation system in Eq. 3.3. Each row corresponds to the realization of Eq. 3.2 for one sample. See the examples to see how it may look.

$$\mathbf{X} \boldsymbol{\beta} = \mathbf{Y} \quad (3.3)$$

This matrix system can be solved by an easy matrix operation as shown in Eq. 3.4.

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \mathbf{Y} \quad (3.4)$$

The main disadvantage with PRS is the combinational terms β_{ij} . They increase the number of coefficients that need to be determined to create the PRS tremendously if the polynomial should be of a high degree or should include many variables. An example of this effect can be seen in Table 3.1, where the required number of samples are listed versus the order of the polynomial.

Table 3.1: Required number of samples for a problem with n design variables for polynomial of different degrees.

Order of Polynomial	Required No of samples / data points
1	$n+1$
2	$(n+1)(n+2)/2$
3	$(n+1)(n+2)(n+3)/6$

This can somewhat be remedied by reducing the number of design variables included in the surrogate models. Subset selection is a method that tries to find the optimal variables to include in the polynomial regression model. Read for example Bhosekar and Ierapetritou (2018) for an overview of this. Regularization is another method that can be used for over-fitting and also to filter away unnecessary design variables.

For additional information on things that can be done with PRS, the interested reader can consult *Response surface methodology: process and product optimization using designed experiments* by Myers and Anderson-Cook, 2009.

3.1 Advantages and Disadvantages with PRS

Advantages

- It is easy to understand the influence from each design variable.
- Is is easy to implement

Disadvantages

- Not suitable to model entities that depend on many variables if combination terms should be included
- It might not be possible to describe be underlying phenomena accurately with a polynomial (of low order)
- The matrix inversion needed to find the coefficients can be computationally expensive for large data sets.

3.2 Example 1

Fit a second order polynomial to the data received from a Latin Hypercube Sampling of Function 1 (Eq. 3.5) and estimate the value at the point $x = [0 \ 0]$.

$$\begin{aligned}
y(\mathbf{x}) &= 2x_1 - x_2 + 1 \\
-1 &\leq x_1, x_2 \leq 1
\end{aligned}
\tag{3.5}$$

The eleven sample points are presented in Table 3.2.

Table 3.2: Samples of Function 1 that has been drawn from a Latin Hypercube Sampling DoE

x_1	x_2	y
-2	-1.6	-1.4
-1.6	1.2	-3.4
-1.2	-0.4	-1
-0.8	0.4	-1
-0.4	1.6	-1.4
0	-0.8	1.8
0.4	-2	3.8
0.8	2	0.6
1.2	0	3.4
1.6	0.8	3.4
2	-1.2	6.2

We want to determine the coefficients, β_i , by solving the equation system in Eq. 3.6.

$$\mathbf{X} \beta = \mathbf{Y} \quad \leftrightarrow \quad \hat{\beta} = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \mathbf{Y}
\tag{3.6}$$

The matrix should look like in Eq. 3.7

$$\begin{bmatrix}
1 & x_1^{(1)} & x_2^{(1)} & x_1^{(1)2} & x_1^{(1)}x_2^{(1)} & x_2^{(1)2} \\
1 & x_1^{(2)} & x_2^{(2)} & x_1^{(2)2} & x_1^{(2)}x_2^{(2)} & x_2^{(2)2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
1 & x_1^{(m)} & x_2^{(m)} & x_1^{(m)2} & x_1^{(m)}x_2^{(m)} & x_2^{(m)2}
\end{bmatrix}
\begin{bmatrix}
\beta_0 \\
\beta_1 \\
\beta_2 \\
\beta_{11} \\
\beta_{12} \\
\beta_{22}
\end{bmatrix}
=
\begin{bmatrix}
y^{(1)} \\
y^{(2)} \\
\vdots \\
y^{(m)}
\end{bmatrix}
\tag{3.7}$$

With numbers inserted, the matrices \mathbf{X} and \mathbf{Y} should look like in Table 3.3.

By solving the equation system in Eq. 3.6, the coefficients should be the ones in Table 3.4.

$$\hat{y}(\mathbf{x}) = 1 + 2x_1 - x_2 + 5 * 10^{-17}x_1^2 - 5.2 * 10^{-17}x_2^2 + 1.4 * 10^{-16}x_1x_2
\tag{3.8}$$

Table 3.3: The matrices \mathbf{X} and \mathbf{Y} used to estimate the values of the coefficients, β_i , of a PRS.

1	x_1	x_2	x_1^2	x_2^2	x_1x_2	y
1	-2	-1.6	4	2.56	3.2	-1.4
1	-1.6	1.2	2.56	1.44	-1.92	-3.4
1	-1.2	-0.4	1.44	0.16	0.48	-1
1	-0.8	0.4	0.64	0.16	-0.32	-1
1	-0.4	1.6	0.16	2.56	-0.64	-1.4
1	0	-0.8	0	0.64	0	1.8
1	0.4	-2	0.16	4	-0.8	3.8
1	0.8	2	0.64	4	1.6	0.6
1	1.2	0	1.44	0	0	3.4
1	1.6	0.8	2.56	0.64	1.28	3.4
1	2	-1.2	4	1.44	-2.4	6.2

Table 3.4: The coefficients, β_i , of a second order polynomial based on the training data.

β_0	β_1	β_2	β_{11}	β_{12}	β_{22}
1	2	-1	5.1E-17	-5.2E-17	1.4E-16

This means that the resulting polynomial is the one shown in Eq. 3.8. This leads to an estimated value at $\mathbf{x}=[0 \ 0]$ of 1. The analytical value is 1, so it is a quite good estimation.

It can be argued that the polynomial is really $y = 2x_1 - x_2$ since the other coefficients are extremely close to zero. This would mean that the polynomial is exactly the tested function. This is not so surprising since Example 1 is a first order polynomial.

3.3 Example 2

Fit a second order polynomial to the data received from a Latin Hypercube Sampling of Function 2 - Rosenbrock's Banana (Eq. 3.5) and estimate the value at the point $\mathbf{x} = [1 \ 1]$.

We want to determine the coefficients, β_i , by solving the equation system in Eq. 3.9.

$$\mathbf{X} \beta = \mathbf{Y} \quad \leftrightarrow \quad \hat{\beta} = (\mathbf{X}^t \mathbf{X})^{-1} \mathbf{X}^t \mathbf{Y} \quad (3.9)$$

With numbers inserted, the matrices \mathbf{X} and \mathbf{Y} should look like in Table 3.5.

Table 3.5: The matrices X and Y used to estimate the values of the coefficients, β_i , of a PRS.

1	x_1	x_2	x_1^2	x_2^2	x_1x_2	y
1	-2	2	4	4	-4	409
1	-1.6	-1.6	2.56	2.56	2.56	1737.32
1	-1.2	0.8	1.44	0.64	-0.96	45.8
1	-0.8	-0.4	0.64	0.16	0.32	111.4
1	-0.4	1.2	0.16	1.44	-0.48	110.12
1	0	0	0	0	0	1
1	0.4	-1.2	0.16	1.44	-0.48	185.32
1	0.8	0.4	0.64	0.16	0.32	5.81
1	1.2	-0.8	1.44	0.64	-0.96	501.8
1	1.6	1.6	2.56	2.56	2.56	92.52
1	2	-2	4	4	4	3601

By solving the equations system in Eq. 3.6, the coefficients should be the ones in Table 3.6. This means that the resulting polynomial is the one shown in Eq. 3.10.

Table 3.6: The coefficients, β_i , of a second order polynomial based on the training data.

β_0	β_1	β_2	β_{11}	β_{22}	β_{12}
-177.87	91.72	-513.83	264.05	217.75	-54.29

$$\hat{y}(\mathbf{x}) = -177.87 + 91.72x_1 - 513.83x_2 + 264.05x_1^2 + 217.75x_2^2 - 54.29x_1x_2 \quad (3.10)$$

This leads to an estimated value at $\mathbf{x}=[1 \ 1]$ of -172.46. The analytical value is 0, so it is a very poor estimation. However, at least it identifies that $\mathbf{x} = [1 \ 1]$ has a lower value than all training points.

Chapter 4

Interpolating Surrogate Models

Interpolating methods use the value at the closest known points to estimate the value at a new location where the value is unknown. The estimation is often a linear combination of the values at the known points as seen in Eq. 4.1.

$$\hat{y} = \sum_{i=1}^m w_i y^{(i)} \tag{4.1}$$
$$\sum_{i=1}^m w_i = 1$$

The sum of the weights should be equal to 1 to ensure that the estimation is unbiased. It can be seen as taking a percentage of the value of each neighbouring point and sum them together. The sum of the weights will then be 100%.

Note that the estimation of a known point will be the already known value of that point since the distance to that training point is zero (and the contribution thereby 100%).

4.1 Inverse Distance Weighting

Inverse Distance Weighting (IDW) is also called Shepard's method (Hwang and Martins (2018), Shepard (1968)) and the weights, w_i , are determined by Equation 4.2. The main idea is that the points that are close to the unknown point are assigned high weights, whereas faraway points are given lower weights. This is due to the reasoning that the closest points should be the ones that are most similar to our unknown point, and therefore we trust these points the most.

$$w_i = \frac{1}{\sum_{j=1}^m d_j} \quad (4.2)$$

It is common that different variables have different units and therefore different orders of magnitude. Therefore, the design variables usually have to be normalized to ensure that all design variables are of equal importance.

Example 1 below results in an accuracy of 9%. But the accuracy improves the closer our known points are to our desired point. Additionally, this model gives reasonable values regardless of the complexity of the underlying function or model.

4.1.1 Advantages and Disadvantages

Advantages

- It is rational that the value of a point is similar to the closest known points.
- No parameter needs to be tuned since it only depends on the distance

Disadvantages

- Pure interpolation can never estimate values that are higher than the highest known value (or lower than the lowest value).
- It is only the distance to the unknown point that is used in the estimation of the weights, so it does not find trends etc.
- It ignores the data's own structure. It does not care if your 5 closest points are all in a line or clustered together.

4.1.2 Example 1

Use the test data from 1.2.1 and try to estimate the value at the point $x = [0 \ 0]$ by interpolation.

The samples are seen in the three leftmost columns of Table 4.1. The fourth column presents the distance from each point to $[0 \ 0]$ whereas the fifth column inverts the distance.

The sixth column presents the weights, calculated according to Eq. 4.2. The seventh column contains the weight of each point multiplied by the value of the point.

The point $x = [0 \ 0]$ is estimated as 0.912 whereas the value of the underlying function is 1. This corresponds to an error of 9% which is a decent but not good estimation for this simple problem.

Table 4.1: Samples drawn from function 1 together with the distance to $x = [0 \ 0]$, the weights given to each point and the value contribution from each point to the estimate of $y(x = [0 \ 0])$

x_1	x_2	y	dist	1/dist	w	w*y
-2	-1.6	-1.4	2.56	0.39	0.053	-0.074
-1.6	1.2	-3.4	2	0.5	0.067	-0.229
-1.2	-0.4	-1	1.26	0.79	0.106	-0.106
-0.8	0.4	-1	0.89	1.118	0.150	-0.150
-0.4	1.6	-1.4	1.65	0.606	0.082	-0.114
0	-0.8	1.8	0.8	1.25	0.168	0.303
0.4	-2	3.8	2.04	0.49	0.066	0.251
0.8	2	0.6	2.15	0.46	0.062	0.037
1.2	0	3.4	1.2	0.83	0.112	0.381
1.6	0.8	3.4	1.79	0.56	0.075	0.256
2	-1.2	6.2	2.33	0.43	0.058	0.358
			18.68	7.43	1	0.912

4.2 Radial Basis Functions

Radial Basis Functions (RBF) are special cases of interpolation where the relationship between the distance to known points and the contribution from each point to the estimated value at the desired point can vary according to a distance function (Broomhead and Lowe (1988)).

The estimation of of a new point can be made using Eq. 4.3

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^m w_i \phi(\mathbf{x}) \quad (4.3)$$

where w_i are weights that determines the contribution from each distance function, \mathbf{x} is the point that should be estimated. The distance function $\phi(\mathbf{x})$ is defined by Eq. 4.4 and should be chosen by the user.

$$\phi(\mathbf{x}) = \phi(\|\mathbf{x} - \mathbf{x}^i\|) \quad (4.4)$$

In Eq. 4.4, \mathbf{x} is the point that should be estimated, \mathbf{x}^i is reference point i , and $\|\cdot\|$ denotes the Euclidean distance between the two points.

The weights, w_i , are calculated by solving the matrix system in Eq. 4.5

$$\Phi \mathbf{W} = \mathbf{Y} \quad (4.5)$$

where Φ is an $n \times n$ matrix with the distance function values of all pairwise combinations of the the training points \mathbf{x} . \mathbf{W} is a column vector with the weights that should be determined and \mathbf{Y} is a column vector with

the function value of each training point. This is exemplified in matrix form in Eq. 4.6 and 4.7.

$$\Phi = \begin{bmatrix} \phi(\|\mathbf{x}^{(1)} - \mathbf{x}^{(1)}\|) & \phi(\|\mathbf{x}^{(1)} - \mathbf{x}^{(2)}\|) & \dots & \phi(\|\mathbf{x}^{(1)} - \mathbf{x}^{(n)}\|) \\ \phi(\|\mathbf{x}^{(2)} - \mathbf{x}^{(1)}\|) & \phi(\|\mathbf{x}^{(2)} - \mathbf{x}^{(2)}\|) & \dots & \phi(\|\mathbf{x}^{(2)} - \mathbf{x}^{(n)}\|) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\|\mathbf{x}^{(n)} - \mathbf{x}^{(1)}\|) & \phi(\|\mathbf{x}^{(n)} - \mathbf{x}^{(2)}\|) & \dots & \phi(\|\mathbf{x}^{(n)} - \mathbf{x}^{(n)}\|) \end{bmatrix} \quad (4.6)$$

$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_2 \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad (4.7)$$

The training process consists of finding the optimal weights, w_i , according to Eq. 4.5, given the predetermined distance functions. However, for the highest accuracy, the training process should include an optimization process where the optimal combination of weights AND distance function parameters are received. In practice, the optimization algorithm suggests parameter values, which are used to calculate the weights, which in turn are used to estimate a set of validation points to determine an accuracy measure that can be used as objective function for the optimization.

Common choices of distance functions are listed here. The distance function 4.3 is represented by r in the formulas. The parameter ϵ is a shape parameter that controls the RBF's width. This value is different for each RBF between the training point and the point at which the value should be estimated. When the RBF interpolation model is trained, the purpose is to find optimal values of ϵ_i .

- Linear: $\phi(r) = r$
- Cubic: $\phi(r) = r^3$
- Gaussian RBF: $\phi(r) = e^{-\epsilon r^2}$
- Inverse Multiquadric RBF: $\phi(r) = \frac{1}{\sqrt{1 + \epsilon r^2}}$
- Multiquadric RBF: $\phi(r) = \sqrt{1 + \epsilon r^2}$
- Thin Plate Spline RBF: $\phi(r) = r^2 \log r$

The coefficients of the distance functions should ideally be optimized using for example a cost function and steepest descent. A shortcut is to use the Gaussian distance function (Eq. 4.8) and the estimation of s suggested by Nakayama, Arakawa, and Sasaki (2001). This estimation is presented in Eq. 4.9, where d_{max} is the distance between the two points that are furthest

from each other, m is the number of training samples and n is the number of design variables.

$$\phi(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{s^2}\right) \quad (4.8)$$

$$s = \frac{d_{max}}{(m * n)^{1/n}} \quad (4.9)$$

It is also possible to include trends in the RBF which turns the estimation equation into Eq. 4.10 (Park and Dang (2010)).

$$\hat{y}(\mathbf{x}) = \sum_{i=1}^m w_i \phi(\mathbf{x}) + \sum_{i=1}^n \beta_i x_i + \beta_0 \quad (4.10)$$

where β_i are the coefficients for the linear terms and β_0 is the constant term. This converts the matrix system needed to determine the weights into Eq. 4.11.

$$\begin{bmatrix} \Phi & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} W \\ \beta \end{bmatrix} = \begin{bmatrix} Y \\ \mathbf{0} \end{bmatrix} \quad (4.11)$$

The matrix P and the column vector β are displayed in Eq. 4.12

$$P = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix} \quad (4.12)$$

4.2.1 Advantages and Disadvantages with RBF

Advantages

- Most RBFs are differentiable, which means that the resulting interpolating functions become smooth.
- The RBF estimation mainly depends on the closest points, which means that it is not affected by points that are far away.
- Since RBFs depend mainly on the distance between the points in the training set, they can be applied to data with any number of dimensions, which makes them suitable for high-dimensional problems.
- RBF is quite easy to implement

Disadvantages

- Determining the optimal parameters for the distance functions chosen is an optimization in itself.
- The choice of the distance functions and their parameters can significantly affect the accuracy of the estimations.
- RBF is not suitable for large training data sets since the linear system of equations that should be solved to determine the weights, w_i , can become ill-conditioned, leading to numerical instability and / or becoming computationally expensive to solve.

4.2.2 Example 1

Use the test data from 1.2.1 and try to estimate the value at the point $x = [0 \ 0]$ by using Radial Basis Functions.

The samples are found in Table 4.2 and the last two columns are the normalized design variable values.

Table 4.2: Samples drawn from function 1 together with the normalized coordinates

x_1	x_2	y	x_{1norm}	x_{2norm}
-2	-1.6	-1.4	0	0.1
-1.6	1.2	-3.4	0.1	0.8
-1.2	-0.4	-1	0.2	0.4
-0.8	0.4	-1	0.3	0.6
-0.4	1.6	-1.4	0.4	0.9
0	-0.8	1.8	0.5	0.3
0.4	-2	3.8	0.6	0
0.8	2	0.6	0.7	1
1.2	0	3.4	0.8	0.5
1.6	0.8	3.4	0.9	0.7
2	-1.2	6.2	1.0	0.2

The matrix with normalized distances between the 11 points is presented in table 4.3.

Then, the exponential distance function in Eq. 4.8 is chosen to allow the use of the approximation of the range parameter. So the range parameter will be calculated with Eq. 4.13.

$$s = \frac{d_{max}}{(m * n)^{1/n}} = \frac{1.14}{(11 * 2)^{1/2}} = 0.243 \quad (4.13)$$

The resulting distance function is then the one in Eq. 4.14.

Table 4.3: The normalized distances between the 11 training points

0	0.707	0.361	0.583	0.894	0.539	0.608	1.140	0.894	1.082	1.005
0.707	0	0.412	0.283	0.316	0.640	0.943	0.632	0.762	0.806	1.082
0.361	0.412	0	0.224	0.539	0.316	0.566	0.781	0.608	0.762	0.825
0.583	0.283	0.224	0	0.316	0.361	0.671	0.566	0.510	0.608	0.806
0.894	0.316	0.539	0.316	0	0.608	0.922	0.316	0.566	0.539	0.922
0.539	0.640	0.316	0.361	0.608	0	0.316	0.728	0.361	0.566	0.510
0.608	0.943	0.566	0.671	0.922	0.316	0	1.005	0.539	0.762	0.447
1.140	0.632	0.781	0.566	0.316	0.728	1.005	0	0.510	0.361	0.854
0.894	0.762	0.608	0.510	0.566	0.361	0.539	0.510	0	0.224	0.361
1.082	0.806	0.762	0.608	0.539	0.566	0.762	0.361	0.224	0	0.510
1.005	1.082	0.825	0.806	0.922	0.510	0.447	0.854	0.361	0.510	0

$$\phi(x^{(i)}, x^{(j)}) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{0.243^2}\right) \quad (4.14)$$

By using the distance function in Eq. 4.14 in matrix Eq. 4.6, the resulting Φ matrix in Table. 4.4 is received.

Table 4.4: The Φ matrix for the training data

1	2.1e-4	0.111	0.0032	1.3e-6	0.007	0.002	3e-10	1.3e-6	2.5e-9	3.8e-8
2.1e-4	1	0.056	0.258	0.184	9.7e-4	2.9e-7	0.001	5.5e-5	1.7e-5	2.5e-9
0.11	0.056	1	0.429	0.007	0.184	0.004	3.3e-5	0.002	5.5e-5	1.0e-5
0.003	0.258	0.429	1	0.184	0.11	4.9e-4	0.004	0.012	0.002	1.7e-5
1.3e-6	0.184	0.007	0.184	1	0.002	5.7e-7	0.184	0.004	0.007	5.7e-7
0.007	9.7e-4	0.184	0.11	0.002	1	0.184	1.3e-4	0.11	0.004	0.012
0.002	2.9e-7	0.004	4.9e-4	5.7e-7	0.184	1	3.8e-8	0.007	5.5e-5	0.034
3e-10	0.001	3.3e-5	0.004	0.184	1.3e-4	3.8e-8	1	0.012	0.11	4.3e-6
1.3e-6	5.5e-5	0.002	0.012	0.004	0.11	0.007	0.012	1	0.43	0.11
2.5e-9	1.7e-5	5.5e-5	0.002	0.007	0.004	5.5e-5	0.11	0.43	1	0.012
3.8e-8	2.5e-9	1.0e-5	1.7e-5	5.7e-7	0.012	0.034	4.3e-6	0.11	0.012	1

This means that the weights, w_i , for the RBF model are the ones in Table 4.5. These weights are the same regardless of which design should be estimated with the RBF model.

Table 4.5: The weights, w_i , for each distance function in the trained RBF.

w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}
-1.3	-3.2	-1.0	0.31	-0.97	1.1	3.4	0.47	1.5	2.7	5.9

Now it is time to estimate the point $x_{des} = [0 \ 0]$. The values after

each step are presented in Table 4.6. The Euclidean distance between each training point and the desired point are presented in the dist column. The distances are then inserted into Eq. 4.14 which results in the distance function contributions in the ϕ column. The contributions from each training point to the estimation of $\hat{y}(x_{des})$ are presented in the last column and they sum up to 0.8719. Hence, The estimated value at $x_{des} = [0 \ 0]$ is 0.8719. The true value is 1, so it is not a very good approximation.

Table 4.6: The contribution from each point when the point $x = [0 \ 0]$ is estimated using an RBF

x_1	x_2	y	w	dist	ϕ	$w * \phi$
-2	-1.6	-1.4	-1.30	0.64	0.0010	-0.0013
-1.6	1.2	-3.4	-3.25	0.5	0.0145	-0.0472
-1.2	-0.4	-1	-1.02	0.32	0.1841	-0.1875
-0.8	0.4	-1	0.31	0.22	0.4291	0.1325
-0.4	1.6	-1.4	-0.97	0.41	0.0563	-0.0544
0	-0.8	1.8	1.10	0.2	0.51	0.5574
0.4	-2	3.8	3.39	0.51	0.0123	0.0417
0.8	2	0.6	0.47	0.54	0.0074	0.0035
1.2	0	3.4	1.46	0.3	0.2180	0.3188
1.6	0.8	3.4	2.65	0.45	0.0339	0.0898
2	-1.2	6.2	5.88	0.58	0.0032	0.0186
						0.8719

4.3 Kriging

Kriging was originally developed by Krige (1951) to model the ore ratio in the ground using drill-core samples at different locations as training data. Sacks et al. (1989) introduced it into design optimisation, and it is now one of the most popular surrogate modeling techniques in engineering. The method can be described from either a geostatistical or a design-optimisation perspective. Here, the latter is adopted, following the notation of Sacks et al. (ibid.) and Jones, Schonlau, and Welch (1998). The derivations can be found in Sacks et al. (1989), whereas Jones, Schonlau, and Welch (1998) includes a good description of the reasoning behind Kriging.

Generally, a PRS is a regression model with an error term mainly due to leaving out high-order terms in the polynomial. The magnitude of this error term should be close for points that are close to each others. Consequently, kriging assumes that the errors have high correlation for two close points and low correlation for points that are far from each other. Therefore, an important choice when using kriging is which correlation function to select.

The kriging model is a stochastic process that can include linear regression and can be described by Eq. 4.15.

$$\hat{y}(x) = \hat{f}(x) + \mathbf{r}(x)^T \mathbf{R}^{-1}(\mathbf{y} - \mathbf{1}\hat{f}(x)) \quad (4.15)$$

where

- \mathbf{x} is the query point whose response is to be estimated,
- $\hat{f}(\mathbf{x})$ is a polynomial regression trend (e.g. a constant),
- $\mathbf{r}(\mathbf{x})$ is an $m \times 1$ vector of correlations between the query point and each of the m training points,
- \mathbf{R} is the $m \times m$ correlation matrix of the training points,
- \mathbf{y} is the $m \times 1$ vector of training responses, and
- $\mathbf{1}$ is an $m \times 1$ vector of ones.

To be specific, \mathbf{R} is an $m \times m$ matrix where entry i,j is the correlation between the regression errors $\epsilon(x^i)$ and $\epsilon(x^j)$ as shown in Eq. 4.16.

$$\mathbf{R}_{ij} = \text{Corr}[\epsilon(x^i), \epsilon(x^j)] \quad (4.16)$$

Similarly, $\mathbf{r}(x)$ is an $m \times 1$ vector with the correlation between the errors in the point that should be estimated, $\epsilon(x)$ and each training point $\epsilon(x^i)$ as shown in Eq. 4.17.

$$\mathbf{r}(x)_i = \text{Corr}[\epsilon(x), \epsilon(x^i)] \quad (4.17)$$

An example of a correlation function is an exponential of the form in Eq. 4.18, where the distance, d , is calculated using Eq. 4.19. θ_h and p_h are parameters that determine the shape of the distance function for design variable x_h . In particular, θ_h can be seen as a measure of how important variable h is as a higher value means that shorter distances approach zero correlation.

$$\text{Corr}[\epsilon(x), \epsilon(x^i)] = e^{-d(x, x^i)} \quad (4.18)$$

$$d(x^i, x^j) = \sum_{h=1}^k \theta_h |x_h^i - x_h^j|^{p_h}, \quad (\theta_h > 0, 1 \leq p_h \leq 2) \quad (4.19)$$

This choice of correlation function (and many other suggested correlation functions) has the desired properties. For small distances between x^i and x^j , the correlation function will be close to one. If the distance is large, the correlation function will instead approach zero.

The correlation parameters $\{\theta_h, p_h\}$ are determined from the training data during fitting, as explained in Section 4.3.3.

4.3.1 Kriging with Trends

Depending on the form of the trend $\hat{f}(\mathbf{x})$, different variants arise:

- **Simple Kriging:** assumes a known, fixed mean (usually zero).
- **Ordinary Kriging:** introduces an unknown constant mean $\hat{f}(\mathbf{x}) = \mu$, which is estimated from the data.
- **Universal Kriging:** uses a polynomial trend.

Isaaks and Srivastava (1989) argues that it is better to remove a trend explicitly. Isaaks and Srivastava (ibid.) argues that it is better to remove a trend explicitly before fitting the Kriging model than to embed it in Universal Kriging. This is supported by Chen et al. (2016), who found no systematic improvement in prediction accuracy from complex regression terms and noted that they can introduce multiple local optima in the likelihood function.

4.3.2 Covariance Functions

The choice of covariance (correlation) function determines the smoothness of the surrogate and is a key modeling decision. The most common options in engineering surrogate modeling are listed below.

- Exponential
- Gaussian
- Spherical
- Matérn

Exponential Covariance Function

Produces a continuous but non-differentiable predictor. Appropriate when the underlying response is expected to have sharp features.

$$C(h) = \exp\left(-\frac{\|h\|}{\theta}\right) \quad (4.20)$$

where:

- h is the distance between two points,
- θ is the range parameter, controlling the rate of decay of correlation.

Gaussian Covariance Function

Corresponds to $p_h = 2$ in Eq. (4.19). The predictor is infinitely differentiable and is the standard choice for smooth engineering responses (e.g. aerodynamic coefficients). It is the kernel used in the example at the end of this section.

$$C(h) = \exp\left(-\frac{\|h\|^2}{\theta^2}\right) \quad (4.21)$$

Spherical Covariance Function

The correlation is exactly zero beyond the range θ , giving a compact support that can be computationally advantageous for large datasets.

$$C(h) = \begin{cases} 1 - \frac{3\|h\|}{2\theta} + \frac{\|h\|^3}{2\theta^3} & \text{if } \|h\| \leq \theta \\ 0 & \text{if } \|h\| > \theta \end{cases} \quad (4.22)$$

Matérn Covariance Function

The Matérn covariance function encompasses a family of covariance functions, where the smoothness of the predictions can be controlled through a smoothness parameter.

$$C(h) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|h\|}{\theta}\right)^\nu K_\nu\left(\frac{\sqrt{2\nu}\|h\|}{\theta}\right) \quad (4.23)$$

where:

- $\nu > 0$ is a parameter that controls the smoothness of the function,
- K_ν is a modified Bessel function.

The Matérn family generalises both the exponential ($\nu = 0.5$) and Gaussian ($\nu \rightarrow \infty$) kernels and is preferred when the analyst is uncertain about the smoothness of the response.

4.3.3 The Maximum Likelihood Estimation Training Approach

With the choice of the regression model as a constant value, that is, $\hat{f}(x) = \mu$, the kriging model becomes the stochastic process in Eq. 4.24.

$$\hat{y}(x^{(i)}) = \mu + \epsilon(x^{(i)}), \quad i = 1, \dots, n \quad (4.24)$$

μ is the mean value of the stochastic process and $\epsilon(x^{(i)})$ is a normal distribution with 0 mean and variance σ^2 . This is the basic model for Kriging

and the correlation between the errors of the different points depends on the chosen correlation functions.

The stochastic process model has $2k+2$ parameters, where k is the number of design variables that need to be determined by using the training data. These are μ , σ^2 , $\theta_1, \dots, \theta_k$, and p_1, \dots, p_k . These parameters can be seen in Eq. 4.19 and section 4.3.2

Log-likelihood The log-likelihood for the parameters $(\mu, \sigma^2, \boldsymbol{\theta})$ given observations \mathbf{y} is calculated according to Eq. 4.25.

$$\mathcal{L}(\mu, \sigma^2, \boldsymbol{\theta}) = -\frac{n}{2} \log(2\pi) - \frac{n}{2} \log(\sigma^2) - \frac{1}{2} \log |\mathbf{R}| - \frac{1}{2\sigma^2} (\mathbf{y} - \mu \mathbf{1})^T \mathbf{R}^{-1} (\mathbf{y} - \mu \mathbf{1}) \quad (4.25)$$

For fixed $\boldsymbol{\theta}$, the optimal μ and σ^2 have the closed-form solutions (Eq. 4.27):

$$\hat{\mu} = \frac{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{y}}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}} \quad (4.26)$$

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \hat{\mu} \mathbf{1})^T \mathbf{R}^{-1} (\mathbf{y} - \hat{\mu} \mathbf{1})}{n} \quad (4.27)$$

Substituting these back into Eq. (4.25) yields the *concentrated* log-likelihood, which depends only on $\boldsymbol{\theta}$. A numerical optimizer (e.g. L-BFGS-B or Differential Evolution) is then used to find the $\boldsymbol{\theta}$ that maximizes it.

Practical notes

- Because MLE optimizes the likelihood of the training data, it simultaneously calibrates the length scales (how quickly correlation decays in each dimension) and the process variance. A large $\hat{\theta}_h$ indicates that dimension h strongly influences the response.
- The MLE objective is generally multi-modal; a global optimizer or multiple random restarts should be used.
- If the wrong kernel is chosen, MLE will still find optimal parameters for that kernel, but the surrogate may exhibit poor generalization.
- For large datasets ($n=1000$), computing \mathbf{R}^{-1} becomes expensive. Approximations such as sparse covariance functions or inducing point methods are then required.

4.3.4 Prediction Accuracy

The main advantage of kriging is that it is possible to estimate the uncertainty of the prediction. Generally, it is desirable that the prediction is more accurate the closer the estimated point is to one or more of the training points. The general formula for the mean squared error of the predictor, s^2 , fulfills this (Jones, Schonlau, and Welch (1998)) and is shown in Eq. 4.28 .

$$s^2(\mathbf{x}) = \hat{\sigma}^2 \left[1 - \mathbf{r}^T(\mathbf{x}) \mathbf{R}^{-1} \mathbf{r}(\mathbf{x}) + \frac{(1 - \mathbf{1}^T \mathbf{R}^{-1} \mathbf{r}(\mathbf{x}))^2}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}} \right] \quad (4.28)$$

where

- $\hat{\sigma}^2(x)$ is the process variance
- $\mathbf{r}(x)$ is the vector of correlations between the point x and each training point
- \mathbf{R} is the matrix of correlations between the training.
- $\mathbf{1}$ is a vector of ones.

The three terms in Eq. 4.28 have clear interpretations:

- The leading 1 represents the baseline process variance $\hat{\sigma}^2$.
- The middle term $\mathbf{r}^T \mathbf{R}^{-1} \mathbf{r}$ is a *reduction* in uncertainty due to the spatial correlation between \mathbf{x} and the training points. At a training point, this term equals 1 and $s^2 = 0$ (the model interpolates exactly).
- The last term accounts for the additional uncertainty introduced because the mean μ is estimated from data rather than known exactly.

The quantity $s(\mathbf{x}) = \sqrt{s^2(\mathbf{x})}$ is sometimes called the *Kriging standard error*. An approximate 95% prediction interval is $\hat{y}(\mathbf{x}) \pm 1.96 s(\mathbf{x})$.

4.3.5 Advantages and Disadvantages with Kriging

Kriging has three major advantages:

- It provides exact interpolation at the training points
- Good estimations for uniformly distributed samples
- The ability to estimate the uncertainty in the prediction of a point. This is unbelievable

Kriging works well for uniformly distributed samples, but not for data that is unevenly sampled (Havinga, Klaseboer, and Boogaard (2013)). A higher sampling density in an area will increase the accuracy of Kriging in that area but at the same time deteriorate the accuracy in the other parts of the design space. What happens is that the spatial correlation parameters, θ_h , are decreasing in size, which narrows the Gaussian basis functions.

This might be a problem if additional samples are added sequentially as described in Section 9. The Kriging model might be refined with additional samples in a promising region. This means that the accuracy in that region will improve and there is a higher chance of finding the best value in that region. But at the same time we might lose accuracy in other parts of the design space, and the true optimum might actually be present there.

4.3.6 Example 2: Predicting a Point on the Rosenbrock Function

This section walks through a complete Ordinary Kriging prediction, including uncertainty quantification, using a small dataset sampled from the 2D Rosenbrock function (section 1.2.2:).

The goal is to predict the response at $\mathbf{x}^* = [1.0 \ 1.0]$ (where the true value is $f = 0$) from 11 training points in table 4.7 generated by Latin Hypercube Sampling.

Table 4.7: Training data: 11 LHS samples from the 2D Rosenbrock function, $x_1, x_2 \in [-2, 2]$.

i	$x_{i,1}$	$x_{i,2}$	y_i
1	-2.0	2.0	409.00
2	-1.6	-1.6	1737.32
3	-1.2	0.8	45.80
4	-0.8	-0.4	111.40
5	-0.4	1.2	110.12
6	0.0	0.0	1.00
7	0.4	-1.2	185.32
8	0.8	0.4	5.80
9	1.2	-0.8	501.80
10	1.6	1.6	92.52
11	2.0	-2.0	3601.00

Step 1 – Standardise the Response

Because the responses span several orders of magnitude, the outputs are standardized to a zero mean and unit variance using Eq. 4.29. The mean

of the data before conversion is $\bar{y} = 618.28$ and the standard deviation is $s_y = 1107.66$.

$$\mathbf{y}_s = \frac{\mathbf{y} - \bar{y}}{s_y} = \frac{\mathbf{y} - 618.28}{1107.66} = \mathbf{y}_s = \frac{\mathbf{y} - \bar{y}}{s_y} = \begin{pmatrix} -0.1889 \\ 1.0103 \\ -0.5168 \\ -0.4576 \\ -0.4588 \\ -0.5573 \\ -0.3909 \\ -0.5529 \\ -0.1052 \\ -0.4747 \\ 2.6928 \end{pmatrix} \quad (4.29)$$

All subsequent computations are performed on \mathbf{y}_s . The final prediction is back-transformed at the end.

Step 2 – Select and Fit the Correlation Model

A Gaussian correlation function ($p_h = 2$ in Eq. 4.19) of the form in Eq. 4.30 is used

$$R_{ij} = \exp\left(-\theta_1(x_{i,1} - x_{j,1})^2 - \theta_2(x_{i,2} - x_{j,2})^2\right) \quad (4.30)$$

A Maximum Likelihood Estimation (Section 4.3.3) is applied to the standardized data. The concentrated log-likelihood is maximized by a numerical optimizer, yielding the theta values in Eq. 4.31 for the two design variables.

$$\hat{\theta}_1 = 0.5, \quad \hat{\theta}_2 = 0.5 \quad (4.31)$$

These equal length scales indicate that both input variables contribute similarly to the response variation over the sampled domain.

Step 3 – Build the Correlation Matrix \mathbf{R}

With $\hat{\theta}_1 = \hat{\theta}_2 = 0.5$, each entry is computed via Eq. 4.30.

$$R_{ij} = \exp\left(-0.5(x_{i,1} - x_{j,1})^2 - 0.5(x_{i,2} - x_{j,2})^2\right) \quad (4.32)$$

For example:

$$R_{1,2} = \exp\left(-0.5(-2.0 + 1.6)^2 - 0.5(2.0 + 1.6)^2\right) = \exp(-6.56) = 0.0014$$

$$R_{8,10} = \exp\left(-0.5(0.8 - 1.6)^2 - 0.5(0.4 - 1.6)^2\right) = \exp(-1.04) = 0.3535$$

$$R_{8,9} = \exp\left(-0.5(0.8 - 1.2)^2 - 0.5(0.4 + 0.8)^2\right) = \exp(-0.80) = 0.4493$$

The full 11×11 matrix \mathbf{R} is (values below 0.001 shown as ≈ 0):

$$\mathbf{R} \approx \begin{pmatrix} 1.000 & 0.001 & 0.354 & 0.027 & 0.202 & 0.018 & 0.000 & 0.006 & 0.000 & 0.001 \\ 0.000 & 1.000 & 0.052 & 0.354 & 0.010 & 0.077 & 0.125 & 0.008 & 0.014 & 0.000 \\ 0.001 & 0.001 & 1.000 & 0.449 & 0.670 & 0.354 & 0.038 & 0.125 & 0.016 & 0.014 \\ 0.354 & 0.052 & 0.052 & 1.000 & 0.449 & 0.670 & 0.354 & 0.038 & 0.125 & 0.016 \\ 0.000 & 0.000 & 0.000 & 0.449 & 1.000 & 0.257 & 0.670 & 0.354 & 0.202 & 0.125 \\ 0.027 & 0.354 & 0.449 & 1.000 & 0.257 & 0.670 & 0.354 & 0.202 & 0.125 & 0.008 \\ 0.006 & 0.010 & 0.670 & 0.257 & 1.000 & 0.449 & 0.041 & 0.354 & 0.038 & 0.125 \\ 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 1.000 & 0.449 & 0.670 & 0.354 \\ 0.018 & 0.077 & 0.354 & 0.670 & 0.449 & 1.000 & 0.449 & 0.670 & 0.354 & 0.077 \\ 0.018 & 0.018 & 0.000 & 0.125 & 0.038 & 0.354 & 0.041 & 0.449 & 1.000 & 0.257 \\ 0.000 & 0.125 & 0.038 & 0.354 & 0.041 & 0.449 & 1.000 & 0.257 & 0.670 & 0.010 \\ 0.202 & 0.202 & 0.006 & 0.008 & 0.125 & 0.202 & 0.354 & 0.670 & 0.257 & 1.000 \\ 0.006 & 0.008 & 0.125 & 0.202 & 0.354 & 0.670 & 0.257 & 1.000 & 0.449 & 0.354 \\ 0.027 & 0.027 & 0.000 & 0.014 & 0.016 & 0.125 & 0.038 & 0.354 & 0.670 & 0.449 \\ 0.000 & 0.014 & 0.016 & 0.125 & 0.038 & 0.354 & 0.670 & 0.449 & 1.000 & 0.052 \\ 0.354 & 0.354 & 0.001 & 0.000 & 0.014 & 0.008 & 0.125 & 0.077 & 0.010 & 0.354 \\ 0.001 & 0.000 & 0.014 & 0.008 & 0.125 & 0.077 & 0.010 & 0.354 & 0.052 & 1.000 \\ 0.001 & 0.001 & 0.000 & 0.000 & 0.000 & 0.018 & 0.202 & 0.027 & 0.354 & 0.001 \\ 0.000 & 0.001 & 0.000 & 0.006 & 0.000 & 0.018 & 0.202 & 0.027 & 0.354 & 0.001 \\ 1.000 & 1.000 & 1.000 & 1.000 & 1.000 & 1.000 & 1.000 & 1.000 & 1.000 & 1.000 \end{pmatrix}$$

Note how the diagonal is always 1 (a point is perfectly correlated with itself), and how nearby points such as $\mathbf{x}_8 = [0.8, 0.4]$ and $\mathbf{x}_{10} = [1.6, 1.6]$ share a moderate correlation of 0.354, while distant pairs such as \mathbf{x}_1 and \mathbf{x}_{11} are nearly uncorrelated.

Step 4 – Estimate $\hat{\mu}$ and $\hat{\sigma}^2$

Using the closed-form MLE expressions (Eq. (4.27)) applied to the standardized response \mathbf{y}_s yield the values in Eq. 4.33 and Eq. 4.34.

$$\hat{\mu}_s = \frac{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{y}_s}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}} = 0.3537 \quad (4.33)$$

$$\hat{\sigma}_s^2 = \frac{(\mathbf{y}_s - \hat{\mu}_s \mathbf{1})^T \mathbf{R}^{-1} (\mathbf{y}_s - \hat{\mu}_s \mathbf{1})}{11} = 0.9134 \quad (4.34)$$

The subscript s is used as a reminder that these estimates are in the standardized space.

Step 5 – Compute the Correlation Vector $\mathbf{r}(\mathbf{x}^*)$

For query point $\mathbf{x}^* = [1.0, 1.0]^T$, each entry of \mathbf{r} is calculated according to Eq. 4.35

$$r_i = \exp\left(-0.5(1.0 - x_{i,1})^2 - 0.5(1.0 - x_{i,2})^2\right) \quad (4.35)$$

The two most influential entries are r_8 and r_{10} , since $\mathbf{x}_8 = [0.8, 0.4]$ and $\mathbf{x}_{10} = [1.6, 1.6]$ are the training points closest to \mathbf{x}^* :

$$\begin{aligned} r_8 &= \exp(-0.5 \cdot 0.04 - 0.5 \cdot 0.36) = \exp(-0.20) = 0.8187 \\ r_{10} &= \exp(-0.5 \cdot 0.36 - 0.5 \cdot 0.36) = \exp(-0.36) = 0.6977 \end{aligned}$$

The full vector is:

$$\mathbf{r}(\mathbf{x}^*) = \begin{pmatrix} 0.0067 \\ 0.0012 \\ 0.0872 \\ 0.0743 \\ 0.3679 \\ 0.3679 \\ 0.0743 \\ 0.8187 \\ 0.1940 \\ 0.6977 \\ 0.0067 \end{pmatrix}$$

Step 6 – Compute the Prediction

Inserting into the Kriging predictor (Eq. 4.15) with $\hat{f}(\mathbf{x}) = \hat{\mu}_s = 0.3537$ yields the calculations in Eq. 4.36

$$\hat{y}_s(\mathbf{x}^*) = \hat{\mu}_s + \mathbf{r}^T \mathbf{R}^{-1}(\mathbf{y}_s - \hat{\mu}_s \mathbf{1}) = 0.3537 + \mathbf{r}^T \mathbf{R}^{-1}(\mathbf{y}_s - 0.3537 \mathbf{1}) = -0.5510 \quad (4.36)$$

Back-transforming this into the original scale gives us the final estimation according to Eq. 4.37

$$\hat{y}(\mathbf{x}^*) = \hat{y}_s \cdot s_y + \bar{y} = -0.5510 \times 1107.66 + 618.28 \approx \boxed{8.0} \quad (4.37)$$

The true value is $f(1, 1) = 0$. The prediction is close given the highly non-linear nature of the Rosenbrock function and the small training set.

Step 7 – Compute the Prediction Uncertainty

We now use Eq. 4.28 to estimate the prediction uncertainty:

$$s^2 = \sigma^2(x) \left[1 - \mathbf{r}^T(x) \mathbf{R}^{-1} \mathbf{r}(x) + \frac{(1 - \mathbf{1}^T \mathbf{R}^{-1} \mathbf{r}(x))^2}{\mathbf{1}^T \mathbf{R}^{-1} \mathbf{1}} \right]$$

With the standardized process variance $\hat{\sigma}_s^2 = 0.9134$ this yields the following:

$$\begin{aligned} \mathbf{r}^T \mathbf{R}^{-1} \mathbf{r} &= 0.9286 \\ \mathbf{1}^T \mathbf{R}^{-1} \mathbf{r} &= 1.0610 \quad \Rightarrow \quad 1 - \mathbf{1}^T \mathbf{R}^{-1} \mathbf{r} = -0.0610 \\ \mathbf{1}^T \mathbf{R}^{-1} \mathbf{1} &= 5.0751 \end{aligned}$$

Finally, the standardized prediction uncertainty is calculated as:

$$\begin{aligned} s_s^2(\mathbf{x}^*) &= 0.9134 \left[1 - 0.9286 + \frac{(-0.0610)^2}{5.0751} \right] = \\ &= 0.9134 [0.0714 + 0.0007] \\ &= 0.9134 \times 0.0721 \\ &= 0.0659 \end{aligned}$$

This number needs to be transformed back:

$$\begin{aligned} s^2(\mathbf{x}^*) &= s_s^2 \cdot s_y^2 = 0.0659 \times 1107.66^2 = 80\,849 \\ s(\mathbf{x}^*) &= \sqrt{80\,849} \approx 284 \end{aligned}$$

The approximate 95% prediction interval can therefore be calculated according to Eq. 4.38, where the 95% yields the 1.96 number.

$$\hat{y} \pm 1.96 s = 8 \pm 1.96 \times 284 \approx [-549, 565] \quad (4.38)$$

This wide interval (from -549 to 565) has a reason. With only 11 training points spread over a very non-linear function, the surrogate is uncertain far from any training point. The interval correctly contains the true value $f(1, 1) = 0$. In practice, additional samples would be placed near \mathbf{x}^* (using, e.g., EGO infill criteria. See sec. 9.1) to reduce $s(\mathbf{x}^*)$ and tighten the interval.

Chapter 5

Other Surrogate Models

There exist numerous SMs that do not rely on interpolation. Some of them are presented in this chapter:

- Decision Tree
- Random Forest
- High-Dimensional Model Representation (HDMR)
- Neural Networks (NN)
- Polynomial Chaos Expansion (PCE)
- Support Vector Regression (SVR)

5.1 Decision Trees for Regression

Decision Trees (Classification and Regression Trees, or CART) divide the design space into a set of hyperspaces, similar to how it is possible to follow a tree from its root to the leaves. Each branching has a yes or no statement that determines the direction. This makes it easy to follow the decision process, but also means that the tree can only predict a discrete number (equal to the number of leaves) of values.

5.1.1 Mathematical Formulation

For a regression tree, the design space is partitioned into J distinct and non-overlapping regions, R_1, R_2, \dots, R_J . For any input falling into region R_j , the prediction \hat{y} is the mean of the training observations y_i in that region as calculated by Eq. 5.1. $I(\cdot)$ is the indicator function that is either 1 or 0 depending on whether the input is in that region or not.

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^J \bar{y}_{R_j} I(\mathbf{x} \in R_j) \quad (5.1)$$

5.1.2 Training Criterion

The split at each node is determined by minimizing the total Residual Sum of Squares (RSS) after the splits. The algorithm searches for the predictor X_j and the split point s that minimize Eq 5.2.

$$RSS = \sum_{i:x_i \in R_1(j,s)} (y_i - \bar{y}_{R_1})^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \bar{y}_{R_2})^2 \quad (5.2)$$

A decision formula is suggested, which splits the dataset from the previous level in the decision tree into two. The mean value of the points belonging to the set R1 is compared to the actual value of each point in the set to get the RSS of the suggested set R1. The same is done for R2, and RSS_1 and RSS_2 are then added together. This gives a performance measure of how good the decision formula was.

5.1.3 Pruning and Regularization

To prevent overfitting, trees are often pruned using *Cost-Complexity Pruning*. This adds a penalty term for the number of terminal nodes (tree leaves) $|T|$ to the objective function:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha|T| \quad (5.3)$$

where α is a tuning parameter controlled via cross-validation.

5.1.4 Advantages and Disadvantages

Some advantages are:

- You can visually trace every decision. If you want to investigate why the model predicted a certain value, it is possible to follow the path along the tree.
- Decision Trees do not require feature scaling (normalization or standardization). Whether the pressure is in Pascals or Mega-Pascals, the tree splits exactly the same way.
- The algorithm naturally picks the most informative variables first. Unimportant inputs are simply ignored and never appear in the tree. This means that it performs its own screening of the variables.
- Decision Trees can capture complex, non-linear relationships without the need for high-order polynomials or complex kernels.

Some disadvantages are:

- Decision Trees are extremely sensitive to the training data. A single new data point can completely change the structure of the entire tree.
- Without strict "pruning," a tree will grow until it perfectly memorizes every data point, including the numerical noise in your simulations.
- Because they predict the mean of a "bucket," the output is a set of discrete steps. This is often physically unrealistic for phenomena like temperature gradients or fluid flow.
- They cannot predict any value higher or lower than the range in the training set.

5.2 Random Forests for Regression

A Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction of the individual trees. This makes it a robust model since it can average out the estimations of the individual trees Ho, 1998.

5.2.1 Bagging and De-correlation

The algorithm utilizes Bootstrap Aggregation (Bagging, Breiman, 2001) to reduce the variance of the estimator. Given a training set $Z = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, bagging repeatedly selects a random sample with replacement from the training set and fits trees to these samples.

For $b = 1, \dots, B$:

1. Sample, with replacement, n training examples from Z ; call these Z_b .
2. Train a regression tree f_b on Z_b .
3. At each node, select m variables at random from the p available variables (typically $m \approx p/3$ for regression).

5.2.2 The Ensemble Prediction

The final prediction for a new input \mathbf{x} is the average of the predictions from all individual trees:

$$\hat{f}_{RF}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(\mathbf{x}) \quad (5.4)$$

5.2.3 Out-of-Bag (OOB) Error

A unique advantage of Random Forests is the OOB error estimate. Since each tree is trained on roughly 63% of the data, the remaining 37% (the "out-of-bag" samples) can be used to validate that tree. This provides a built-in cross-validation mechanism without the need for a separate validation set.

5.2.4 Advantages and Disadvantages

Some advantages are:

- By averaging hundreds of trees, the "noise" in individual trees cancels out. This makes it much more stable than a single Decision Tree.
- You get a validation score for "free" because the model tests itself on data it didn't use for training (Out-of-Bag samples).
- It is resistant to overfitting. You can add 1 000 trees to a forest, and it generally will not overfit more. It will just become more stable.
- It performs exceptionally well even when you have 50+ input variables but only a small number of expensive simulation samples.
- It provides a quantitative ranking of which variables actually drive your results, which is excellent when it is used for design optimization.

Some disadvantages are:

- When you have many trees, it is extremely difficult to understand the model. You trade interpretability for accuracy.
- While individual trees are fast, a large forest can be memory-intensive to store and slower to run during "inference" (making a prediction).
- Just like a single tree, a Random Forest can never predict a value outside the original training range (it is bounded by the y_{min} and y_{max} of the data).
- If your input variables are highly correlated, the "feature importance" rankings can sometimes be misleading.

5.3 High-Dimensional Model Representation (HDMR)

HDMR originates from Sobol (1993) and strives to be efficient models of problems with many variables.

The easiest model to use is a first-order polynomial with interaction terms of the second order as in Eq. 5.5.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \beta_{11} x_1 x_1 + \beta_{12} x_1 x_2 + \dots + \beta_{mm} x_m x_m \quad (5.5)$$

There exist two common types of HDMR, cut-HDMR and RS-HDMR (Random Sampling) Li, Rosenthal, and Rabitz (2001). Cut-HDMR represents $f(\mathbf{x})$ with reference to a cutting point in the design space, whereas RS-HDMR is based on the average value of $f(\mathbf{x})$ in the design space.

Higher terms than second order can be neglected (Li, Rosenthal, and Rabitz (ibid.)). This means that the mapping between the k x_i design variables and the estimation $f(\mathbf{x})$ can be written according to Eq. 5.6

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^k f_i(x_i) + \sum_{1 \leq i < j \leq k} f_{ij}(x_i, x_j) \quad (5.6)$$

cut-HDMR

Cut-HDMR uses a cutting point. It is recommended to use the point in the middle of the design space as the cutting point (Ulaganathan et al. (2016)). The different functions can then be attained by Eq. 5.7

$$\begin{aligned} f_0 &= f(\mathbf{x}_0) \\ f_i(x_i) &= f(x_i, \mathbf{x}_0^i) - f_0 \\ f_{ij}(x_i, x_j) &= f(x_i, x_j, \mathbf{x}_0^{ij}) - f(x_i) - f(x_j) - f_0 \end{aligned} \quad (5.7)$$

The required number of samples, m_{req} , can be calculated with Eq. 5.8, where n is the number of variables and s is the number of values used for each input variable x (Li, Rosenthal, and Rabitz (2001)). This means that m_{req} is reduced from s^n , which is exponential, but still has the same information.

$$m_{req} = 1 + ns + \frac{n(n-1)s^2}{2} + \dots \quad (5.8)$$

Example 1

Try to adapt a cut-HDMR to the function in Eq. 5.9

$$\begin{aligned} y &= 1 + x_1 + 3x_2 - x_3 - 4x_1 x_2 \\ -2 &\leq x_i \leq 2, \quad i = 1, 2, 3 \end{aligned} \quad (5.9)$$

We will use the simple form in Eq. 5.5, which means that our model in this case actually will be able to reanimate the function exactly. But this example can anyway be used to demonstrate the principle.

Seven samples are needed to fit the model. The samples are presented in Table 5.1. We use the recommended approach from Ulaganathan et al. (2016), and use the central point of the design space as the cutting point.

The sampling points for the first-order contributions are the maximum value of each variable and the combination points are the maximum of both variables.

Table 5.1: Samples drawn from eq. 5.9

x_1	x_2	x_3	$f(x)$
0	0	0	1
2	0	0	3
0	2	0	7
0	0	2	-1
2	2	0	-7
2	0	2	1
0	2	2	5

The first coefficient, β_0 is calculated by Eq. 5.7. It is calculated here by performing the calculations in Eq. 5.10

$$f_0 = f(\mathbf{x}_0) = f([0 \ 0 \ 0]) = 1 = \beta_0 \quad (5.10)$$

The coefficients for the first order terms, β_i are determined by Eq. 5.7. This means that Eq. 5.11 calculates these values for our problem and model.

$$\begin{aligned} f_1(x_1) &= f(x_1, \mathbf{x}_0^1) - f_0 = f([2 \ 0 \ 0]) - 1 = 3 - 1 = 2 = \beta_1 * x_1 = \beta_1 * 2 \quad \leftrightarrow \quad \beta_1 = 1 \\ f_2(x_2) &= f(x_2, \mathbf{x}_0^2) - f_0 = f([0 \ 2 \ 0]) - 1 = 7 - 1 = 6 = \beta_2 * x_2 = \beta_2 * 2 \quad \leftrightarrow \quad \beta_2 = 3 \\ f_3(x_3) &= f(x_3, \mathbf{x}_0^3) - f_0 = f([0 \ 0 \ 2]) - 1 = -1 - 1 = -2 = \beta_3 * x_3 = \beta_3 * 2 \quad \leftrightarrow \quad \beta_3 = -1 \end{aligned} \quad (5.11)$$

The coefficients for the combination terms, β_{ij} , are calculated by Eq. 5.7. Eq. 5.12 calculates the last coefficients that we need.

$$\begin{aligned} f_{12}(x_1, x_2) &= f(x_1, x_2, \mathbf{x}_0^{12}) - f_1(x_1) - f_2(x_2) - f_0 = f([2 \ 2 \ 0]) - 2 - 6 - 1 = \\ &\quad - 7 - 9 = -16 = \beta_{12} * x_1 * x_2 = \beta_{12} * 2 * 2 \quad \leftrightarrow \quad \beta_{12} = -4 \\ f_{13}(x_1, x_3) &= f(x_1, x_3, \mathbf{x}_0^{13}) - f_1(x_1) - f_3(x_3) - f_0 = f([2 \ 0 \ 2]) - 2 + 2 - 1 = \\ &\quad 1 - 1 = 0 = \beta_{13} * x_1 * x_3 = \beta_{13} * 2 * 2 \quad \leftrightarrow \quad \beta_{13} = 0 \\ f_{23}(x_2, x_3) &= f(x_2, x_3, \mathbf{x}_0^{23}) - f_2(x_2) - f_3(x_3) - f_0 = f([0 \ 2 \ 2]) - 6 + 2 - 1 = \\ &\quad 5 - 5 = 0 = \beta_{23} * x_2 * x_3 = \beta_{23} * 2 * 2 \quad \leftrightarrow \quad \beta_{23} = 0 \end{aligned} \quad (5.12)$$

This means that our approximation is $f(\mathbf{x}) = 1 + x_1 + 3x_2 - x_3 - 4x_1x_2$. This is exactly the underlying function.

Example 2

Now, let us try to create a second order (Eq. 5.13) cut-HDMR representation of Rosenbrock's function (Eq. 1.4).

$$\hat{y}(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_{12} x_1 x_2 \quad (5.13)$$

The drawn samples from Rosenbrock's function are presented in Table 5.2. Two samples per variable are needed since a second order behavior should be reanimated.

Table 5.2: Drawn samples from Rosenbrock's function

x_1	x_2	$f(x)$
0	0	1
-2	0	1609
2	0	1601
0	-2	401
0	2	401
2	2	401

The first coefficient, β_0 , is here calculated by performing the calculations in Eq. 5.14

$$f_0 = f(\mathbf{x}_0) = f([0 \ 0]) = 1 = \beta_0 \quad (5.14)$$

Eq. 5.15 calculates these values for our problem and model.

$$\begin{aligned}
f_1(x_1 = -2) &= f(x_1, \mathbf{x}_0^1) - f_0 = f([-2 \ 0]) - 1 = 1609 - 1 = 1608 \\
&= \beta_1 * x_1 + \beta_{11} * x_1^2 = -\beta_1 * 2 + \beta_{11} * 4 \\
f_1(x_1 = 2) &= f(x_1, \mathbf{x}_0^1) - f_0 = f([2 \ 0]) - 1 = 1601 - 1 = 1600 \\
&= \beta_1 * x_1 + \beta_{11} * x_1^2 = \beta_1 * 2 + \beta_{11} * 4 \\
&\Leftrightarrow \beta_1 = -2, \quad \beta_{11} = 401 \\
f_2(x_2 = -2) &= f(x_2, \mathbf{x}_0^2) - f_0 = f([0 \ -2]) - 1 = 401 - 1 = 400 \\
&= \beta_2 * x_2 + \beta_{22} * x_2^2 = -\beta_2 * 2 + \beta_{22} * 4 \\
f_2(x_2 = 2) &= f(x_2, \mathbf{x}_0^2) - f_0 = f([0 \ 2]) - 1 = 401 - 1 = 400 \\
&= \beta_2 * x_2 + \beta_{22} * x_2^2 = \beta_2 * 2 + \beta_{22} * 4 \\
&\Leftrightarrow \beta_2 = 0, \quad \beta_{22} = 100
\end{aligned} \quad (5.15)$$

Eq. 5.16 calculates the last coefficient we need.

$$\begin{aligned}
f_1(x_1, x_2 = 2) &= f(x_1, x_2, \mathbf{x}_0^{12}) - f_1(x_1 = 2) - f_2(x_2 = 2) - f_0 = \\
&f([2 \ 2]) - 1600 - 400 - 1 = 401 - 2001 = -1600 \quad (5.16) \\
&= \beta_{12} * x_1 * x_2 = \beta_{12} * 2 * 2 \leftrightarrow \beta_{12} = -400
\end{aligned}$$

This means that our approximation is $f(\mathbf{x}) = 1 - 2 * x_1 + 401 * x_1^2 + 100 * x_2^2 - 400 * x_1 * x_2$.

5.4 Neural Networks

Neural Networks (NN) are highly flexible models that are popular in machine learning and pattern detection (McCulloch and Pitts, 1943). They have grown tremendously in popularity since the beginning of this millennium. Nowadays, NNs are the go-to algorithms for image recognition, recommendation systems, and large language models. In the context of engineering surrogate modeling, an NN is a high-dimensional, non-linear interpolator.

An NN consists of interconnected layers of nodes, or neurons. Each neuron receives inputs from other neurons or an external source, applies a set of learnable synaptic weights, and uses a (usually non-linear) activation function to produce an output.

The network must have one neuron in the input layer for each input variable (feature) and one node in the output layer for each entity that should be predicted. This is illustrated in figure 5.1, which shows a neural network that models \hat{y} with respect to two input variables x_1 and x_2 . Note that there are two neurons (h_1 and h_2) in the single hidden layer and that there is only one input since we want to predict the value of y .

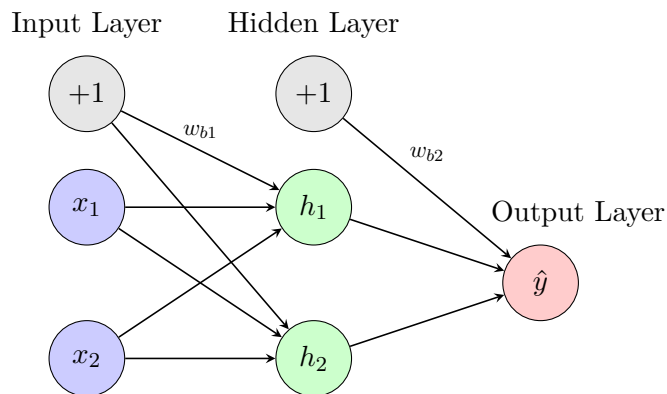


Figure 5.1: A Neural Network with architecture $2 \times 2 \times 1$: two input neurons (x_1, x_2), one hidden layer with two neurons (h_1, h_2), and one output neuron (\hat{y}). Bias nodes are shown explicitly (+1).

For a classification model there would be one output node per class (e.g. 10 outputs for digit recognition 0–9), where each output represents the probability that the input belongs to that class.

5.4.1 The Neuron: Core Building Block

The fundamental unit of a NN is the *neuron*, which loosely mimics the signal processing of biological neurons. For neuron j , the input signals x_i are combined into a weighted sum (the *pre-activation* according to Eq. 5.17 where w_{ji} is the weight from input i to neuron j and b_j is a bias term.

$$z_j = \sum_{i=1}^n w_{ji} x_i + b_j \quad (5.17)$$

This sum is then passed through a non-linear activation function σ (see Eq. 5.18). Without the non-linearity, a network of any depth reduces to a single linear transformation and cannot model complex physical phenomena.

$$a_j = \sigma(z_j) \quad (5.18)$$

5.4.2 Making Predictions by Forward Propagation

Forward propagation is the process of computing the output of the network given an input vector \mathbf{x} . In a Multi-Layer Perceptron (MLP), neurons are organized into layers. The activations of layer l are calculated using the outputs of the previous layer $l - 1$ as in Eq. 5.19.

$$\mathbf{a}^{(l)} = \sigma(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \quad (5.19)$$

$\mathbf{W}^{(l)}$ is the weight matrix and $\mathbf{b}^{(l)}$ is the bias vector for layer l . The final layer produces the surrogate prediction $\hat{\mathbf{y}}$ for a given set of design parameters.

5.4.3 Activation Functions

Activation functions are the key ingredient that gives a NN its power. Without them, no matter how many layers the network has, it collapses to a single linear equation. In engineering terms, without activation functions, the surrogate model could only fit hyperplanes, no matter how many neurons are used.

Common Activation Functions

Table 5.3 contains some common activation functions and their equations are presented below.

Table 5.3: Common activation functions with recommended use cases.

Function	Out Range	Best Use Case
Linear	$(-\infty, \infty)$	Output layer for regression (e.g., predicting Stress or Pressure).
ReLU	$[0, \infty)$	Standard choice for all hidden layers.
Sigmoid	$(0, 1)$	Output layer for classification (e.g., Fail/Pass).
Tanh	$(-1, 1)$	Good for hidden layers if the data is normalized around zero.

Linear Activation The output equals the pre-activation value z , enabling any real-valued output. It is the standard choice for the output neuron in regression problems.

ReLU (Rectified Linear Unit)

The current industry standard. It is described by Eq. 5.20 and is incredibly simple. If the input is negative, the output is zero. If positive, it passes the value through. ReLU is extremely fast to compute and avoids the vanishing gradient problem in deep networks. Its main drawback is the dying "ReLU" problem: if a neuron's pre-activation is consistently negative, the gradient is always zero and the neuron never updates.

$$f(z) = \max(0, z) \tag{5.20}$$

Sigmoid

The Sigmoid equation (5.21) creates an S-shaped curve between 0 and 1. It is often used in output layers for classification since the values will be close to 0 or 1.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{5.21}$$

Hyperbolic Tangent (Tanh)

The Tanh function (5.22) sets most values to -1 or 1. It is generally preferred over Sigmoid for hidden layers because it is zero-centered, which generally makes gradient descent converge faster.

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{5.22}$$

Leaky ReLU and Swish. These are modern variants designed to mitigate the dying ReLU problem by allowing a small, non-zero gradient for negative inputs.

5.4.4 Overall Workflow for Surrogate Modelling

The development of an Artificial Neural Network (ANN) as a surrogate for high-fidelity simulations (e.g. CFD/FEM) approximately follows this workflow:

1. **Data Pre-processing & Normalization:** Inputs \mathbf{x} and targets \mathbf{y} are normalized to a common scale.
2. **Dataset Partitioning:** The m available samples are split into Training (m_{tr}), Validation (m_{val}), and Test (m_{te}) subsets.
3. **Architecture & Algorithm Selection:** The network topology (number of layers and neurons) and the learning rate are chosen. This is typically an iterative process.
4. **Training - Backpropagation and Early Stopping:** Weights are optimized via backpropagation. Early stopping halts training when the validation loss \mathcal{J}_{val} begins to diverge from the training loss \mathcal{J}_{tr} .
5. **Performance Evaluation Verification:** The final model is evaluated on the unseen Test Set and accuracy metrics calculated.

All steps are described in more detail in the following sections.

5.4.5 Data Pre-processing and Normalization

Since activation functions such as Sigmoid and Tanh are sensitive to input magnitude, and because different engineering quantities can span several orders of magnitude (e.g. pressure at 10^6 Pa vs. temperature at 10^1 °C), the inputs and targets must be scaled before training. Otherwise, the network will implicitly weight the numerically largest feature far more than the others.

Two standard normalization strategies are:

- A mean of 0 and a standard deviation of 1. This is achieved by using Eq. 5.23 on the dataset.

$$z = \frac{x - \mu}{\sigma} \quad (5.23)$$

- Min-Max scaling to the interval $[c, d]$ according to Eq. 5.24.

$$x_{i,\text{norm}} = c + \frac{(x_i - \min(\mathbf{x})) (d - c)}{\max(\mathbf{x}) - \min(\mathbf{x})} \quad (5.24)$$

Setting $c = 0$, $d = 1$ maps all values to $[0, 1]$; setting $c = -1$, $d = 1$ maps to $[-1, 1]$.

5.4.6 Dataset Creation and Management

Data in engineering is often expensive to generate via experiments or simulations. To ensure the model generalizes well to unseen designs, the data must be split and scaled.

- **Training Set (70-80%):** Used to update the weights.
- **Validation Set (10-15%):** Used to monitor for overfitting. Training should stop when validation error begins to rise.
- **Test Set (10-15%):** A final check on a "hidden" dataset to report the model's true accuracy.

The model will be trained using the training set. However, it will use the validation set to ensure that the model is not over-fitted to the training set. This means that the validation set is indirectly used during training. Hence, it can not be used to assess the model's ability to predict on new data. That is why we save some designs for testing.

A useful analogy. The training set is the course material the model learns from. The validation set is the practice exam, used to tune early stopping and hyperparameters. The test set is the final exam: it is kept secret until the very end to verify that the model has learned the underlying physics, not just memorized the training noise.

5.4.7 Architecture Algorithm Selection

To create an NN, the number of layers and nodes need to be defined. The NN should have ONE input layer and it should have the same number of neurons as the number of input variables. Some configurations might have a bias term as well, but that can be included in the hidden layer instead. The NN should have ONE output layer and for regression, the number of nodes should equal the number of entities whose value the NN should predict. For classification, it is possible to have one output node for each class. That node should output a measure of how likely a design is to fall into that class.

The number of hidden layers and nodes in them is usually where the major decisions regarding the NN architecture need to be made. Sarle (2002) has edited a Usenet newsgroup where this topic has been investigated and discussed. One hidden layer might be enough, but in some cases two (or even three) layers might be needed. Generally, the number of parameters that should be trained should be less than the number of samples in the training data set and each layer introduces many new parameters.

There are different guidelines for the size (number of nodes) in the first hidden layer. A practical recommendation for a first attempt is as follows,

where N_{in} is the number of input variables and N_{out} is the number of outputs.:

1. Start with one hidden layer and $N_h = N_{in}$ neurons. Evaluate the R^2 score on the validation set.
2. If performance is unsatisfactory, add a second hidden layer with $N_h = N_{in}/2$ neurons.
3. If performance is still poor, the bottleneck is likely insufficient data, not network size. Collect more samples before enlarging the network.

Two additional heuristics from the literature for the first hidden layer size are:

- Use Eq. 5.25.

$$N_h^{(1)} \approx \sqrt{N_{in} \cdot N_{out}} \quad (5.25)$$

- Use Eq. 5.26.

$$N_h^{(1)} \approx \frac{2}{3}N_{in} + N_{out} \quad (5.26)$$

Avoiding overfitting through capacity control. To avoid overfitting, the total number of trainable parameters W should be constrained by the number of available training samples m_{train} . A common heuristic for surrogate models is:

$$W \leq \frac{m_{train}}{\rho} \quad (5.27)$$

where ρ is a safety factor, typically $\rho \in [5, 10]$. If the network capacity exceeds this limit, techniques such as *Dropout* or *Weight Decay* (ℓ_2 regularization) must be employed.

5.4.8 Training - Backpropagation and Early Stopping

The power of ANNs lies in their ability to learn the synaptic weights from data. The most common learning algorithm is *backpropagation* Rumelhart, Hinton, and Williams, 1986, which minimizes a cost function by adjusting the weights via gradient descent. Having analytical expressions for the derivatives of the activation functions is essential since it makes gradient computation exact and training orders of magnitude faster than finite-difference approaches.

Main Training Functions

The following three functions are essential for the training process:

1. **The Loss Function:** Typically, the Mean Squared Error (MSE) in Eq. 5.28 is used. The loss function is calculated as the sum of the squared errors between the predictions and the true values.

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{k=1}^m (y_k - \hat{y}_k)^2 \quad (5.28)$$

2. **Optimizer:** Parameters are updated iteratively. The simplest update rule is vanilla Gradient Descent using Gradient Descent with Eq. 5.29. η is the step length (learning rate) that scales the gradient. In practice, the Adam optimizer is preferred because it adapts the learning rate for each parameter individually and converges faster than plain gradient descent.

$$\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{J} \quad (5.29)$$

3. **Backpropagation:** The gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}$ is calculated using the chain rule, propagating the error from the output layer back to the input according to Eq. 5.30.

$$\frac{\partial \mathcal{J}}{\partial w_{ji}^{(l)}} = \frac{\partial \mathcal{J}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} \quad (5.30)$$

Training Procedure

Training involves finding the optimal weights \mathbf{W} and biases \mathbf{b} that minimize the difference between the surrogate's prediction $\hat{\mathbf{y}}$ and the ground-truth simulation data \mathbf{y} .

1. Randomly initialize all weights using Xavier (Glorot) initialization:

$$\epsilon = \frac{\sqrt{6}}{\sqrt{N_{in} + N_{out}}} \quad (5.31)$$

where N_{in} and N_{out} are the number of incoming and outgoing connections for the layer. Weights are drawn uniformly from $[-\epsilon, \epsilon]$.

2. Perform a forward pass to obtain $\hat{y}^{(i)}$ for all training samples.
3. Compute the loss $\mathcal{J}(\boldsymbol{\theta})$ on both the training set and the validation set.
4. Plot the loss values against the current epoch number.
5. Use backpropagation to compute $\partial \mathcal{J} / \partial w_{ji}^{(l)}$ for all weights.
6. Update all parameters using the optimizer (e.g. Adam).
7. One *epoch* is now complete. Return to Step 2.

Regularization via Early Stopping

Early stopping is used to stop the training process before the model is overfitting. The model iterates through the training data. After each iteration (epoch), it checks the Validation Set. If the validation error stops dropping, the "Early Stopping" trigger pulls the plug to prevent overfitting. This can create a decent NN even if the architecture is bad.

When you train a neural network, you typically split your data into a Training Set and a Validation Set. Training Loss represents how well the model is memorizing the data it sees. It will almost always continue to decrease as long as you keep training. Validation Loss represents how well the model generalizes to data it has not seen. In the beginning, both curves go down together. But eventually, the model starts learning the specific noise in your training data. At this point, the training loss keeps dropping, but the validation loss starts to rise. This u-turn in the validation curve is the exact moment the model transitions from learning physics to memorizing noise. This divergence, sometimes called the *generalization gap*, is the signal to stop:

$$\frac{\partial \mathcal{J}_{tr}}{\partial t} < 0 \quad \text{and} \quad \frac{\partial \mathcal{J}_{val}}{\partial t} > 0 \quad (5.32)$$

where t denotes the epoch counter.

It is not desirable to stop the very first time the validation loss ticks upward (it might just be a small bump in the optimization). Instead, we use a setting called Patience Monitor:

- Check (Monitor) the validation loss after every epoch (iteration)
- If the loss stops improving, wait for p additional epochs (this is your "Patience" parameter)
- If no improvement is observed within p epochs, stop training and *roll back* the weights to the epoch with the lowest validation loss.

This ensures that the surrogate model maintains the best possible generalization capability for unseen design points.

Early Stopping is free. Unlike L_2 regularization (Weight Decay), which requires you to tune a math coefficient (λ), Early Stopping only requires you to look at a graph. It's the single most effective way to ensure a Neural Network remains a "physically plausible" surrogate.

Early stopping requires no additional mathematical tuning (unlike ℓ_2 regularization (Weight Decay), which requires selecting a penalty coefficient λ). It is the recommended first-line defense against overfitting for engineering surrogate models. It is also very simple to interpret since it only requires you to look at a graph.

5.4.9 Performance Evaluation Verification

The loss curve (training and validation loss vs. epoch number) should always be plotted to confirm that training has converged and that early stopping triggered correctly.

The performance and accuracy of the NN need to be assessed by using a test set that has not been involved in the training of the NN. Then methods from section 7.2 are used to reveal the accuracy. Some examples are:

- R^2 Score: How much of the physical variance is captured?
- RMSE: What is the average real world error in physical units (e.g., ± 5 MPa)?
- Residual Plots: Scatter plots of prediction error vs. true value to reveal systematic biases (e.g. systematic under-prediction at high pressures).

5.4.10 Data Augmentation

In image recognition, data augmentation (rotating, flipping, or cropping images) is a powerful way to artificially expand the training set. The same image can appear multiple times in different orientations, all genuinely representing the same object. Applying this idea to engineering simulations requires caution. It is critical to distinguish between replication and augmentation.

- **Avoid Data Replication:** Simply duplicating existing FEA results to create a larger dataset for a Neural Network provides no new information. This leads to extreme overfitting and invalidates validation metrics, as the model uses identical points for training.
- **Use Physical Symmetries for Augmentation:** If the problem has a known symmetry (e.g. a symmetric airfoil or a symmetric bridge cross-section), mirroring existing results produces genuine new data points that the model has not seen.
- **Prioritize Screening:** If the number of input variables is large relative to the available data, perform a screening study (e.g. Morris One-at-a-Time, Section 8.5.1) to identify and remove the least influential variables before training.

5.4.11 For CFD: Physics-Informed Neural Networks (PINNs)

A significant limitation of standard NNs is that they are purely data-driven and can produce physically impossible results. Researchers have tried to remedy this by embedding governing Partial Differential Equations (PDEs)

directly into the loss function as in Eq. 5.33 where $\mathcal{J}_{physics}$ enforces physical laws (e.g., conservation of mass or energy).

$$\mathcal{J}_{total} = \mathcal{J}_{data} + \lambda \mathcal{J}_{physics} \quad (5.33)$$

This allows for accurate surrogates even with sparse training data.

5.4.12 Advantages and Disadvantages

ANNs are capable of modeling highly non-linear systems, are robust to measurement noise, and require no a priori knowledge of the underlying physical relationships. However, they are computationally intensive to train, require more data than simpler models such as Kriging or Response Surface Methods (RSM), and are difficult to interpret since the learned weights do not directly reveal which physical mechanisms govern the predictions.

A summary is provided in Table 5.4.

Table 5.4: Advantages and disadvantages of Artificial Neural Networks as engineering surrogate models.

Criterion	Advantages	Disadvantages
Modeling	Captures highly complex, non-linear relationships	"Black-box" nature makes physical interpretation difficult
Datasets	Highly scalable to high-dimensional problems	Requires more data points than Kriging or RSM
Predictions	Near-instantaneous evaluation after training	Very poor extrapolation outside training bounds

5.5 Polynomial Chaos Expansion

Polynomial Chaos Expansion (PCE) is a functional approximation method that represents the relationship between input variables and the output response as a sum of orthogonal polynomials (Xiu and Karniadakis, 2002). Whereas Neural Networks use "black-box" neurons, PCE builds an explicit, analytical polynomial expression of the form in Eq. 5.34, where c_j are the coefficients to be determined and Ψ_j are multivariate orthogonal polynomials.

$$Y \approx \sum_{j=0}^P c_j \Psi_j(\mathbf{X}) \quad (5.34)$$

The magic of PCE lies in the choice of polynomials. By choosing polynomials that are orthogonal with respect to the probability density functions

(PDF) of the inputs, the model provides global sensitivity information for free (Sudret, 2008). For example:

- If your input has a Gaussian (Normal) distribution, you use Hermite polynomials.
- If your input has a Uniform distribution, you use Legendre polynomials.

This alignment ensures that the expansion converges much faster than a standard Taylor series or a basic power series.

5.5.1 Training

There are two primary ways to find the coefficients c_j :

- Intrusive (Galerkin): Requires modifying the original governing equations (FEA/CFD code). This is rarely used by non-experts as it is mathematically grueling.
- Non-Intrusive (Collocation/Regression): The original simulation is treated as a black box. You run N samples and use *Least-Angle Regression (LAR)* or *Sparse Point Integration* to solve for the coefficients. This is the standard approach in modern engineering.

5.5.2 Advantages and Disadvantages

The greatest advantage of PCE is that once you have the coefficients c_j , you can calculate the Sobol Sensitivity Indices (section 6.1.2) analytically without any further simulations (*ibid.* Because the polynomials are orthogonal, the total variance is simply the sum of the squares of the coefficients as in Eq. 5.35.

$$\text{Var}(Y) = \sum_{j=1}^P c_j^2 \quad (5.35)$$

This allows you to say exactly what percentage of the output variance is caused by each input variable (e.g., "70 % of the stress variation is due to the tolerance on the thickness").

Other advantages are:

- Very few coefficients are needed for smooth engineering problems.
- It is excellent for characterizing material property uncertainties.

Some disadvantages are:

- It struggles with non-smooth responses (e.g., "if-else" logic or discontinuities).

- It suffers from the "curse of dimensionality" as polynomial terms grow exponentially.
- It requires the input probability distributions to be known beforehand.

PCE is essentially a smart regression. If your response surface is smooth and your inputs are well-defined probability distributions, PCE will likely be more accurate and more informative than a Neural Network. However, if the physics involves sharp changes (like buckling or fracture), PCE might show "oscillations" (Gibbs phenomenon) and a Neural Network or Kriging might be better.

5.6 Support Vector Regression (SVR)

SVR (Drucker et al., 1996 is an extension of Support Vector Machines (SVM, Cortes and Vapnik, 1995), originally developed for binary classification. In a classification context, SVMs seek a hyperplane that maximizes the margin between two distinct classes. The simplest version of an SVM is a line in a 2D plot, where all points on one side of the line belong to one class and the points on the other side belong to the second class. (In reality, it might be difficult to find a line that perfectly classifies the points.)

When support vectors are used for regression, this concept is adapted somewhat. SVR seeks to find a function $\hat{f}(\mathbf{x})$ that has at most ϵ deviation from the actually obtained targets y_i for all the training data. This creates an ϵ -insensitive tube around the regression line, where errors within the tube are ignored, providing the model with inherent robustness against noise and small fluctuations in the data. This is different from traditional Ordinary Least Squares (OLS) regression, such as PRS, which minimizes the sum of squared errors.

5.6.1 Training Methodology and Optimization

Training a Support Vector Regression (SVR) model involves solving a constrained convex optimization problem. The objective is to find a function that deviates from the actual observed targets y_i by a value no greater than ϵ for all training data, while simultaneously remaining as "flat" as possible to prevent overfitting.

The Optimization Problem: Minimize:

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m (\xi_i + \xi_i^*) \quad (5.36)$$

Subject to:

$$\begin{aligned} y_i - (\mathbf{w}^T \phi(\mathbf{x}_i) + b) &\leq \epsilon + \xi_i \\ (\mathbf{w}^T \phi(\mathbf{x}_i) + b) - y_i &\leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* &\geq 0 \end{aligned}$$

Where:

- \mathbf{w} is the weight vector, which determines the *flatness* (regularization) of the model.
- C is the *box constraint*, a penalty parameter that dictates the trade-off between model smoothness and the degree to which deviations larger than ϵ are tolerated.
- ϵ defines the *insensitive tube*; errors smaller than this threshold are ignored by the loss function.
- ξ_i, ξ_i^* are slack variables representing the distance of points falling outside the ϵ -tube.
- $\phi(\mathbf{x}_i)$ represents the Kernel Function (e.g., Linear, Polynomial, or Radial Basis Function), which maps the input data into a higher-dimensional space to capture non-linearities.

5.6.2 The Kernel Trick and Common Functions

The "Kernel Trick" allows SVR to perform non-linear regression by implicitly mapping the input data into a high-dimensional feature space $\phi(\mathbf{x})$ without ever explicitly computing the coordinates in that space. This is achieved by replacing the dot product of the mapping functions with a Kernel function: $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$.

Common kernel functions used in engineering surrogate models include:

- **Linear Kernel:** Used for simple, nearly linear physical relationships.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \quad (5.37)$$

- **Polynomial Kernel:** Useful for capturing interaction effects between variables.

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d \quad (5.38)$$

- **Radial Basis Function (RBF) / Gaussian:** The most popular choice for complex engineering surfaces due to its ability to handle highly non-linear behavior.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad (5.39)$$

5.6.3 Advantages and Disadvantages

The choice of SVR as a surrogate model is usually driven by the need for a global optimum and robustness against noisy simulation data.

Advantages:

- The ϵ -insensitive loss function makes the model highly resistant to outliers.
- The optimization problem is convex, ensuring a global optimum (no local minima).
- It is effective in high-dimensional spaces, even if the number of dimensions exceeds the number of samples.
- The Kernel functions allow the model to learn complex, non-linear physical responses.

Disadvantages:

- The performance is sensitive to the choice of the noise threshold ϵ .
- It is computationally expensive for very large datasets ($m > 10,000$).
- It requires careful cross-validation to tune the hyperparameters C , γ and ϵ .
- Unlike Kriging, standard SVR does not provide a direct measure of prediction uncertainty.

The interested reader could consult Smola and Schölkopf, [2004](#) for a tutorial.

Chapter 6

Design of Experiments

Design of Experiments (DoE) or Sampling Plans are strategies used to determine which experiments/samples should be carried out to obtain as accurate surrogate models as possible with as few samples as possible. A good overview can be found in Liu, Ong, and Cai (2018). The strategies presented in this chapter try to create a good sampling plan when the user defines the desired number of samples.

The user can choose to use all the computational budget on one DoE that can be used for prediction - One-shot sampling. The other alternative is to use Sequential sampling. This means that a majority of the computational budget is used to create an initial DoE. The remaining budget is then used to perform experiments/simulations according to a scheme with the aim of improving the SM in certain areas of the design space.

The disadvantage of one-shot sampling is that it is difficult to know the optimal or required sample size before the surrogate model is created. Additionally, it is often difficult to know where in the design space the SM should have high accuracy.

Sequential sampling methods try to use the information obtained from known data points to determine where additional samples should be placed. The main drawbacks are that it is more complicated to implement and that the initial SM may guide the sequential additions to the wrong area of the design space.

Methods for both categories are described in this chapter, with a focus on one-shot sampling since those methods are used for the initial DoE in the sequential approach as well.

6.1 One-Shot Sampling Methods

One-shot sampling methods can be divided into two categories.

- Classical approaches that follow a certain scheme.

- Space-filling techniques that aim to spread out the experiments/samples so that they cover the design space as thoroughly as possible.

It is generally preferable to use space-filling designs when samples are based on deterministic computer models instead of physical experiments, since they try to cover the entire design space rather than focusing on the outer boundaries (Sacks et al., 1989, Jin, Chen, and Simpson, 2001). This is especially true when interpolating SMs are used.

Alizadeh, Allen, and Mistree (2020) recommends using Latin Hypercube Sampling for problems with many design variables, Fractional Factorial Design if little computation time is available, and D-Optimal Design when high accuracy is needed.

Each of the methods mentioned is demonstrated by generating sampling plans for a problem with three design variables, each with limits between -1 and 1. The resulting plans are presented with three 2D projections per sampling method.

6.1.1 Classical Approaches

The classical approaches were developed to model physical experiments and contain methods such as:

- Full Factorial Design (FFD) - This method tests all possible combinations of input variables and levels. It is comprehensive but can be time-consuming and expensive, especially when the number of factors increases.
- Fractional Factorial Design - A subset of the full factorial design, this method tests only some combinations of the factors.
- Central Composite Design (CCD) - Places one point in the center of the design space and other points at the boundaries of the design space.
- Box-Behnken Design (BBD) - This is an alternative to CCD and is useful for three-level factorial designs.
- D-Optimal Design
- Plackett-Burman Design

While these methods try to reduce the random error of physical experiments, they have two main disadvantages (Liu, Ong, and Cai (2018)):

- They have been developed with polynomial response surfaces in mind, usually with a fixed predetermined model.
- They generally place many samples around the boundary of the design space instead of focusing on the interior.

Full Factorial Design

The n input variables are divided into k levels. FFD then performs one experiment for each combination, resulting in k^n experiments. The main advantage is that the interactions between all input variables, $x_i x_j$, are revealed. The main disadvantage of this method is the extremely high number of experiments required if there are many input variables. Hence, it is unsuitable for modeling entities that depend on many input variables.

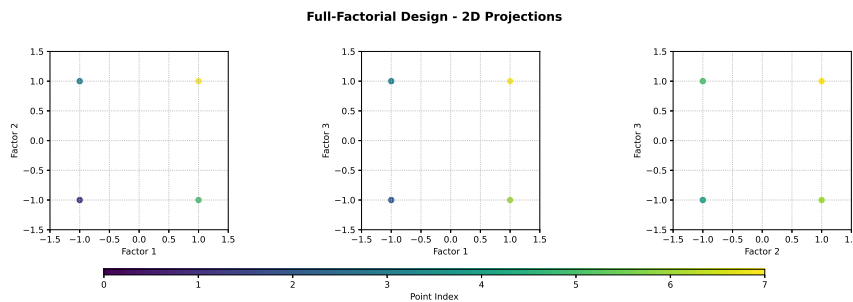


Figure 6.1: Example of a Full Factorial Design of a design space with three variables. Eight samples are drawn in this case.

Fractional Factorial Design

Fractional Factorial Design is a variant of FFD that requires fewer experiments. It tests only a fraction of the total possible combinations of input variables and levels. This fraction is chosen to capture the most critical information about the system's response.

It can be a suitable method for initial exploratory studies or when resources are limited. However, it might not reveal all interactions between the different input variables. Figure 6.2 shows that 4 designs are needed, which is half of what a Full Factorial Design requires for a DoE with 3 design variables.

Central Composite Design

CCD consists of three types of points:

- A point in the center of the design space
- Axial points that are placed at the lower and upper limits of each input variable while the other variables are held at their values at the center point.
- Corner points that are placed at the corners of the design space to capture interactions between the different input variables. These points can be placed in a Full or Fractional Factorial Design manner.

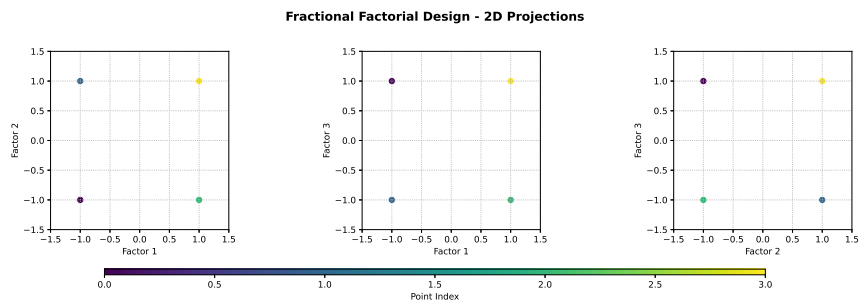


Figure 6.2: Example of a Fractional Factorial Design of a design space with three variables. Four samples are drawn in this case.

CCD is specifically designed to build and explore quadratic response surfaces and may not be suitable for other types of models. Figure 6.3 shows that one design is placed in the middle of the design, whereas all other designs are placed on the outskirts of the design space.

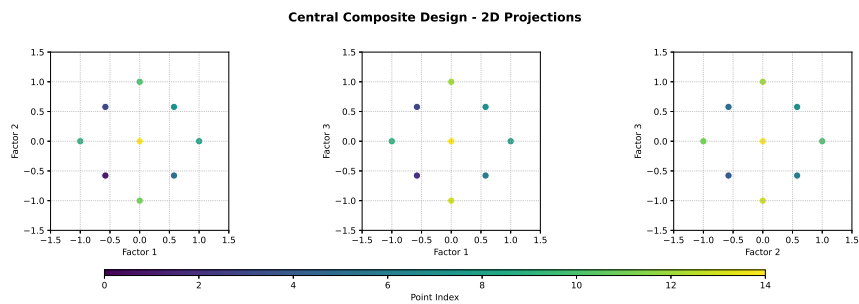


Figure 6.3: Example of a Central Composite Design of a design space with two variables. Note the center sample, the 6 axis samples and the 8 corner samples

Box-Behnken Design

This design was developed by Box and Behnken, 1960 and is intended to fit a quadratic response surface. It is especially useful for cases where fewer experiments are desired without compromising the ability to explore quadratic relationships between variables. BBD is designed to use three levels for each input variable and avoids placing points at the edges of the design space. Its main disadvantage is that it is not suitable for modeling phenomena that depend on many input variables.

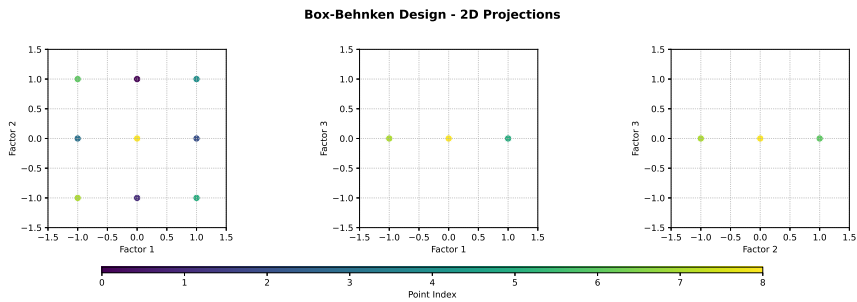


Figure 6.4: Example of a Box Behnken design of a design space with three variables. Nine samples are drawn.

D-Optimal Design

This design is tailored for regression models of the form in Eq. 6.1. Also see the Polynomial Response Surfaces section (3)

$$\mathbf{Y} = \mathbf{X}\beta + \epsilon \quad (6.1)$$

D-Optimal Design tries to place the training points to maximize the determinant of the information matrix ($\mathbf{X}'\mathbf{X}$ in linear regression; hence, the D in "D-Optimal" stands for determinant).

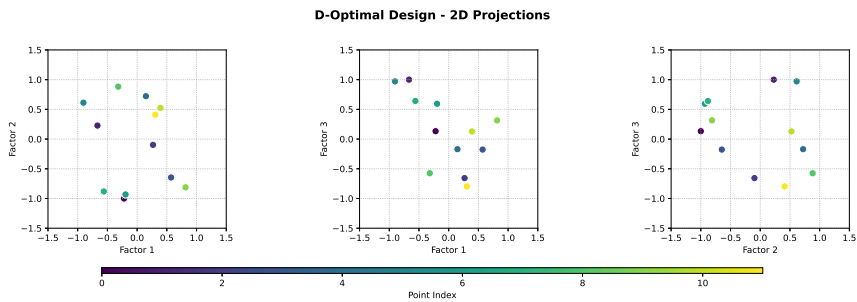


Figure 6.5: Example of a D-optimal Design of a design space with three variables.

Plackett-Burman Design

Plackett-Burman Design (Plackett and Burman, 1946) is a type of design used particularly for screening a large number of factors to identify the most important ones with a relatively small number of runs. This design is especially useful in the early stages of experimental investigation when the goal is to identify the few significant factors from among many possible ones.

This design has two major disadvantages:

- The interaction between two or more variables is not revealed but is included in the main effects
- No second order or higher terms are included, which means that only first order impacts are included.

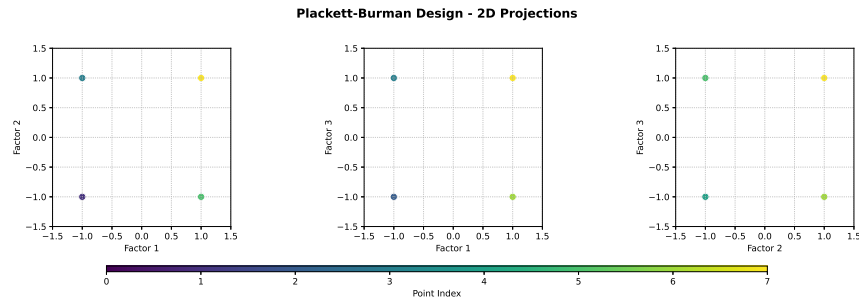


Figure 6.6: Example of a Plackett-Burman Design of a design space with three variables.

6.1.2 Space-Filling Designs

Space filling designs are design strategies that attempt to cover the design space as effectively as possible. This typically include

- Spreading the samples uniformly
- Maximizing the distance between the individual samples

The space-filling designs are especially suitable for interpolating SMs, as the points that should be estimated ideally should be close to the training points.

Latin Hypercube Sampling (LHS)

LHS divides the range of each variable into equal probability intervals and samples one point from each interval (McKay, Beckman, and Conover (1979)).

This is illustrated in figure 6.7 for two design variables when three points are desired. In the 2D case, the intervals can be represented as rows and columns. The LHS criterion is that there should be one point present in each row and column.

Figure 6.8 presents an example where 20 samples have been created for a problem with 3 input variables. It is difficult to see, but there are 5 samples in each row and column in each 2D projection.

The main drawback with a pure LHS is that a DoE with all designs along the diagonal fulfills the LHS criteria. However, the points are not covering

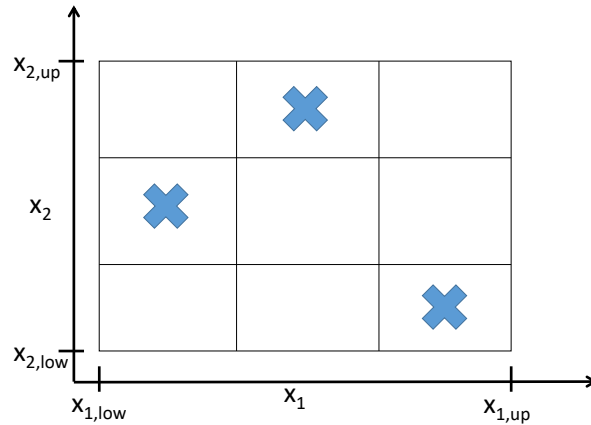


Figure 6.7: Example of an Latin Hypercube Sampling that creates 3 samples for a dataset with two input variables.

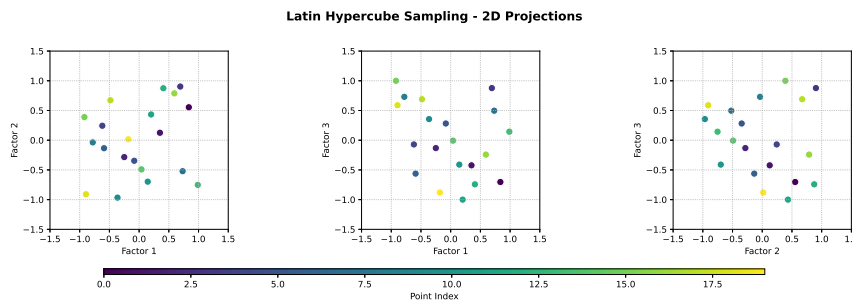


Figure 6.8: Example of a Latin Hypercube Sampling of a design space with three variables. 20 samples are drawn.

the design space very well in that case. Consequently, the LHS can have poor design space coverage, although the user has control over the number of points. One solution is to add another criterion, i.e. that the distance between the two closest points should be maximized.

Other drawbacks are that

- The results are not repeatable since the points are created randomly
- It is difficult to add points afterwards without destroying the LHS criteria. If the criteria should be maintained, the number of added points should be a multiple of the original number of points.

Orthogonal Sampling

Orthogonal sampling can be seen as an extension to LHS. The samples need to be created simultaneously since they must fulfill two conditions:

- The LHS criteria (One sample in each row and column for the 2D example)
- Each region of the design space should be sampled with the same density.

The second criterion is exemplified in Figure 6.9.

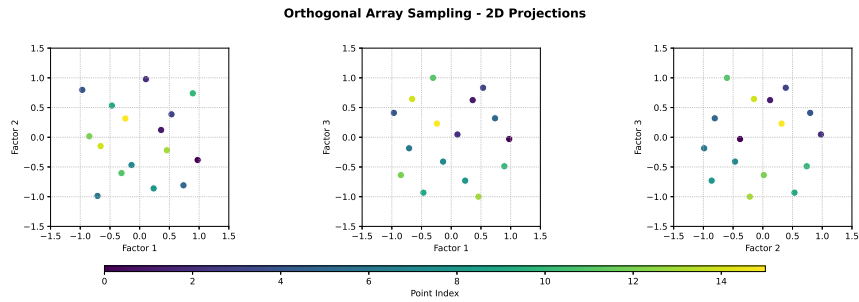


Figure 6.9: Example of a Orthogonal Sampling of a design space with two variables. Three samples are drawn.

Hammersley Sampling

Hammersley sampling is a quasi-random process for sampling points in the design space (Hammersley (1960)). It is constructed using the Van der Corput sequences, which mirrors numbers across a decimal point in a specific base (usually base 2).

The points are spaced perfectly evenly along the first axis ($0, 1/m, 2/m, \dots$). The latter dimensions are calculated using the Van der Corput bit-reversal strategy according to Eq. 6.2. For m points in n dimensions, the k -th point \mathbf{x}_k is defined by Eq. 6.2, where $\Phi_b(k)$ is the radical inverse function in base b .

$$\mathbf{x}_k = \left(\frac{k}{m}, \Phi_{b_1}(k), \Phi_{b_2}(k), \dots, \Phi_{b_{n-1}}(k) \right) \quad (6.2)$$

This creates an extremely uniform grid that avoids placing points close to each other. However, you must decide the total number of points (m) before you start. If you add one more point later, the entire structure's optimality breaks.

An example of a Hammersley sampling for a 3-dimensional design space is shown in figure 6.10.

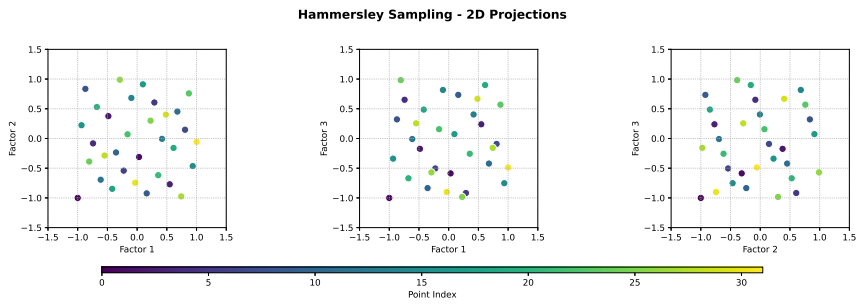


Figure 6.10: Example of a Hammersley Sampling of a design space with three variables

Sobol Sequences

The Sobol sequence Sobol, 1967 is a recommended approach to spread the points as much as possible. The sequences use a base-2 digital construction and linear shift registers to achieve uniformity across all dimensions. They are *incremental*, meaning the sequence can be stopped at any m while maintaining low-discrepancy properties.

Sobol sequences converge much faster towards the mean than the standard Monte Carlo method does. Its main benefit is for problems where the required number of samples is not known beforehand since it can be stopped whenever desired. This means that you can begin with 10 points. If the model's accuracy is too small, you can add 10 additional points and the 20 points will still be spread out.

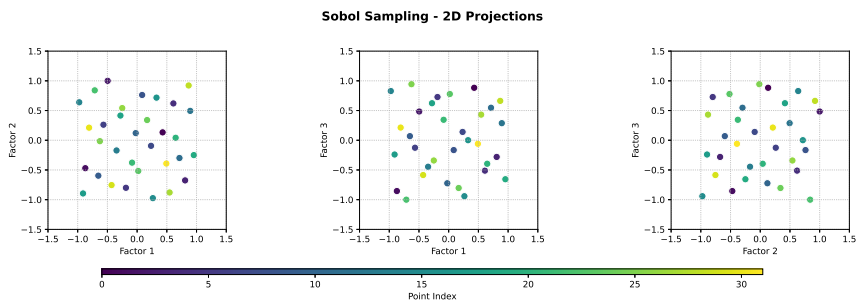


Figure 6.11: Example of a Sobol Sequence Sampling of a design space with three variables.

An example of a Sobol Sequence sampling for a 3-dimensional design space is shown in figure 6.11.

Hammersley VS Sobol Analogy

Imagine you are seating people in a cinema.

- Randomly: People sit anywhere; some sit together, leaving huge gaps
- Hammersley: You know 100 people are coming, so you pre-assign seats perfectly.
- Sobol: The first person sits in the middle. The second sits as far as possible from the first. The third sits in the middle of the largest remaining gap. The room always stays balanced no matter when you stop letting people in.

6.2 Sequential or Adaptive Sampling

A Sequential or Adaptive Sampling method begins by using part of the experimental budget to create an initial DoE. Clever schemes or optimizations are then used to decide where infill points should be placed to increase the accuracy of the SM. Some common one are shown in section [9.1](#)

Query-by-committee based adaptive sampling uses an ensemble of surrogate models and the point where the difference in the estimation of the surrogate models is the largest is selected as the new point. Usually, the difference in estimation is calculated as the variance in the estimations by the different surrogate models.

Chapter 7

Assessing the Accuracy of a Surrogate Model

An SM has several benefits, but it is important to know its limits and disadvantages. The first and foremost disadvantage is that there will almost always be a modeling error since the SM should be able to represent a phenomenon from just a few data points. Therefore, it is highly recommended to estimate its accuracy.

The accuracy of regression models can be estimated by comparing the training data with the SM estimation at the training points.

Interpolation models must always have their accuracy measured by validating it to a data set that was not used to create the model. The reason is that an interpolation model always estimates the value at its training locations to exactly the value of the training data. This is exemplified in figure 7.1 where the interpolation model passes through all training points, whereas the regression model does not.

The methods in this chapter will be exemplified with the data sets in Table 7.1 and Figure 7.2. The blue diagonal crosses in the figure represent the training data from which the surrogate models have been trained, and the red circles represent points that should be evaluated.

The two columns to the left in Table 7.1 present the training data set, whereas the third and fourth columns present the validation data set. The training data has been used to train a Polynomial Response Surface and a Radial Basis Functions model. The predictions for the validation set from using these SMs can be found in the two rightmost columns.

It is recommended that the training data set is several times larger than the validation data set, e.g. a 80-20 ratio. The 50-50 ratio here is mainly to demonstrate how the accuracy measures and visualization work.

This chapter is divided into three parts

- Visualizing Accuracy

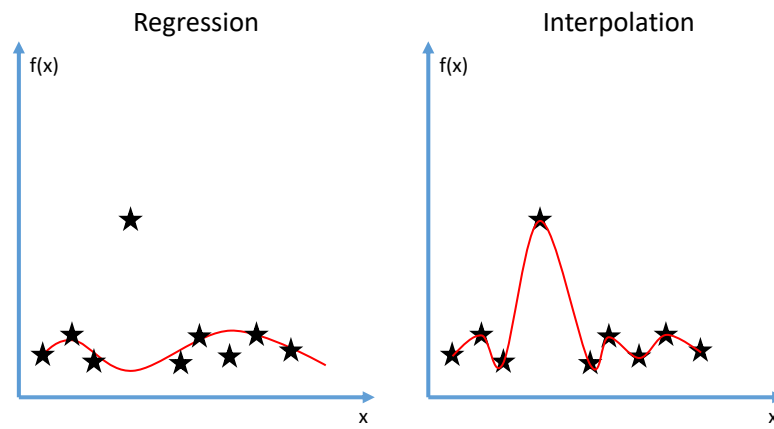


Figure 7.1: Schematic example of how interpolation and regression models adapts to the training data.

- Accuracy Measures
- Assessment Strategies

Table 7.1: The x and y values of the training and validation data set respectively. The two rightmost columns include estimated values for the validation set by using a Polynomial Response Surface and a Radial Basis Function model respectively.

X_{train}	Y_{train}	X_{val}	Y_{val}	yPRS	yRBF	
-2	2	409	-2 -1.6	3145	994.9	1592.7
-1.6	-1.6	1737.32	-1.6 1.2	191.72	100.3	207.6
-1.2	0.8	45.8	-1.2 -0.4	343.4	184.7	276.4
-0.8	-0.4	111.4	-0.8 0.4	9	-183.3	-24.9
-0.4	1.2	110.12	-0.4 1.6	209.32	-611.3	119.2
0	0	1	0 -0.8	65	59.0	3.1
0.4	-1.2	185.32	0.4 -2	466.92	536.7	568.7
0.8	0.4	5.8	0.8 2	185	-614.5	71.7
1.2	-0.8	501.8	1.2 0	207.4	312.4	-40.5
1.6	1.6	92.52	1.6 0.8	310.12	338.5	18.0
2	-2	3601	2 -1.2	2705	1547.4	2159.0

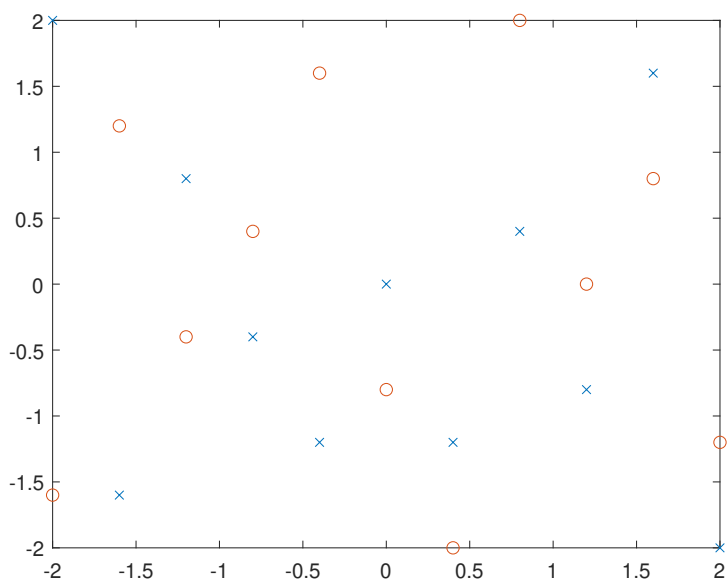


Figure 7.2: Two LHS DoEs for a two dimensional problem with variable limits -2 to 2.

7.1 Visualizing Accuracy

While numerical measures help to assess the accuracy of a surrogate model and also compare different surrogate models, the analysis can be aided by visualizing the accuracy.

Numerical values might be misleading even if it is somewhat unlikely if several metrics are used, but it could for example be that some regions are of extra importance.

Here follows some plots that can help to visualize the accuracy of the surrogate model.

7.1.1 Plot Estimations and True Values

By having the true values and the predicted values as two separate series, with the design number on the x-axis, it can be seen how close each prediction is to the true value. The two series should be as close together. This is exemplified in figure 7.3 where the true values can be compared with the predictions of the PRS and RBF models.

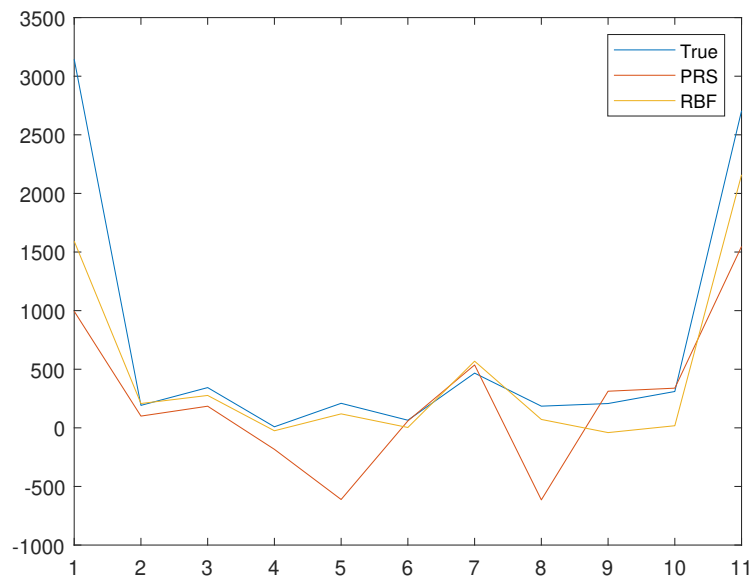


Figure 7.3: Example of a plot where the true values and the predictions using the different surrogate models are plotted with one line for each dataset. The x-axis is the point ID number and the y-axis shows the true and predicted values.

7.1.2 Plot Estimations VS True Value (a QQ plot)

By having the true value on the x-axis and the predicted value on the y-axis, it can be seen how close each prediction is to the true value. This is called a quintile-quintile (QQ) plot and should resemble a straight line if the predictions are 100 % accurate. This is exemplified in 7.4, where both the straight line and the evaluated points are shown.

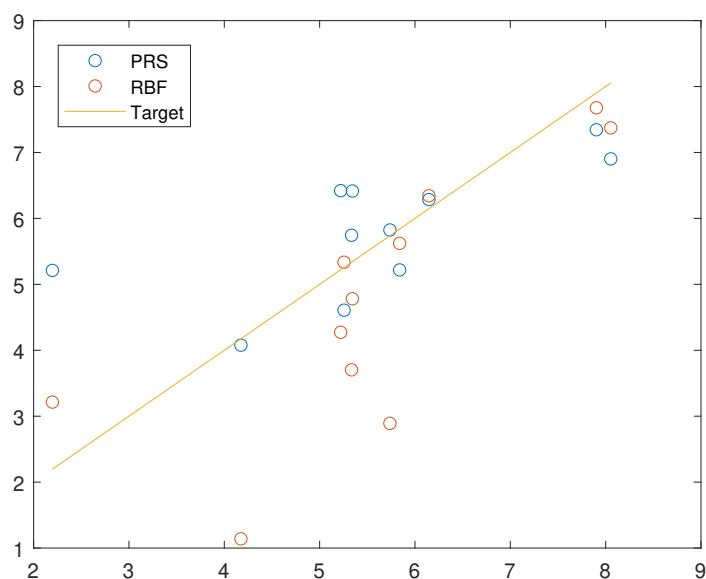


Figure 7.4: Example of a QQ-plot where the true values are on the x axis and the predicted values are on the y axis. The straight line corresponds to a 100 % accuracy.

7.1.3 Plot Residuals

It is possible to calculate the residuals between the predictions and the true values by subtracting one from the other. These can then be plotted as in figure 7.5 to reveal how large the prediction errors are.

7.1.4 Dolan-Moré Performance Profile Plot

A Dolan-Moré performance profile plot is a graphical tool used to compare the performance of multiple methods, usually optimization algorithms or surrogate models (**dolan2002benchmarkingempty citation**). It provides a visual representation of how well each algorithm or model performs relative to the others across a set of test problems or samples. The creators

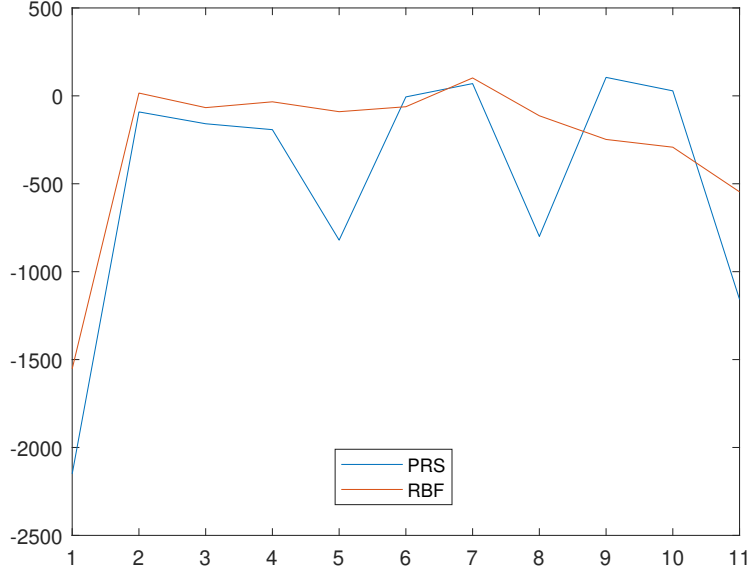


Figure 7.5: Example of a residual plot where the y-axis represents the error values and the x-axis represents the design IDs.

of the method used it to compare the solving times of different solvers on a set of problems.

Figure 7.6 contains a Dolan-Moré chart of the accuracies of the predictions of the example dataset using the PRS and RBF models. It can be interpreted that PRS has the best accuracy for 37% of the points and RBF for 63%. Another takeaway is that 63% of the PRS predictions are less 2.5 times worse than the best prediction. For RBF, that number is 82%. Ideally, the curve should be as close to the top left corner as possible. This means that RBF is better than PRS overall.

To create a Dolan-Moré performance profile plot, the following steps are performed:

1. For each test problem, the performance metric, such as the objective function value or computational cost, is computed for each algorithm or surrogate model.
2. The performance ratio, $r_{i,j}$, is calculated as follows:

$$r_{i,j} = \frac{p_{i,j}}{\min_k p_{i,k}} \quad (7.1)$$

where $p_{i,j}$ is the performance of the j -th algorithm or model on the i -th test problem, and the minimum value is taken over all algorithms

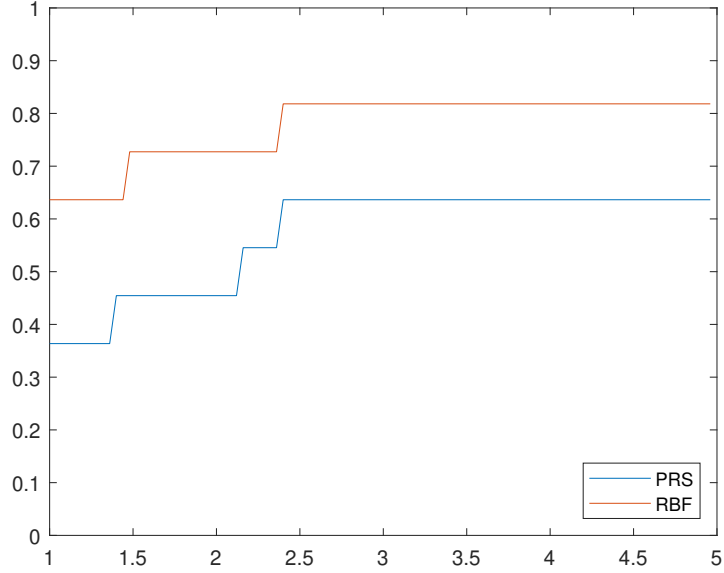


Figure 7.6: Example of a Latin Hypercube Sampling of a design space with two variables. Three samples are drawn.

or models, k .

3. The performance profile, $\rho_j(\tau)$, is defined as the fraction of problems for which the performance ratio is at most τ :

$$\rho_j(\tau) = \frac{1}{m} \sum_{i=1}^m I(r_{i,j} \leq \tau) \quad (7.2)$$

where m is the number of test problems or estimations, and $I(\cdot)$ is an indicator function that takes the value 1 if the condition inside the parentheses is true, and 0 otherwise.

4. Plot $\rho_j(\tau)$ versus τ for each algorithm or model in the comparison.

The resulting plot will be an informative way to show the performance of several models in the same graph. A disadvantage is that the models are compared to the best of the other models, which makes it a relative measure and not an absolute one.

7.2 Accuracy Measures

There are numerous measures that can be used to assess the accuracy of an SM and some of them are presented in 7.2. Each of the methods is presented below the table.

Table 7.2: Summary of different popular accuracy measures for Surrogate Models

Metric	Accuracy Focus	Target Value	Outlier Sensitive
R^2/Q^2	Global	1.0	Low / Medium
RMSE	Global	0.0	High
NRMSE	Global (Scaled)	0.0	High
MAE / RAAE	Global	0.0	Low
Max Error	Local	0.0	Extreme
RMAE	Local (Scaled)	0.0	Extreme

The metrics are described in more detail below, but first there are some considerations.

- Metrics that assess global accuracy (such as R^2 and MAE) indicate how the model performs on average. They are great for general design exploration, where a small error in one corner is not devastating.
- Metrics that assess local accuracy (such as Max Error and RMAE) indicate the worst-case scenario. These are vital for engineering, where a single large prediction error could lead to physical failure in the real world.
- Metrics that are sensitive to outliers (such as RMSE/NRMSE) square the errors. This means that one bad prediction pulls the score down much harder than MAE. This is good if you want to penalize large errors heavily.
- Metrics that focus on the maximum error (such as Max Error and RMAE) are entirely defined by the single worst outlier. If these values are high while your R^2 is good, it means your model is terrible in a specific small region of the design space.

It is recommended to use at least three metrics: Q^2 (to prove the model can predict unseen data). RAAE (to show the overall average accuracy). RMAE (to show the worst-case error). These metrics can be replaced with others, but the combination of overall accuracy, worst-case error and capability to predict new points are important information.

If your model has a low (good) RAAE value but a very high (bad) RMAE value, it usually indicates that the model is fitting well everywhere except for one outlier or one specific non-linear "kink" in the physics (such as a sudden transition from laminar to turbulent flow). This is a cue to add more sampling points in that specific region.

7.2.1 Mean Squared Error (MSE)

The MSE is the average of the squared differences between the model predictions and the true function values. It is defined according to Eq. 7.3, where m is the number of test points, $y(\mathbf{x}_i)$ is the true function value, and $\hat{y}(\mathbf{x}_i)$ is the surrogate model prediction.

$$\text{MSE} = \frac{1}{m} \sum_{i=1}^m (y(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i))^2 \quad (7.3)$$

7.2.2 Root Mean Squared Error (RMSE)

The RMSE is the square root of the MSE and provides an indication of the average prediction error in the same units as the function values. It is defined according to Eq. 7.3.

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{m} \sum_{i=1}^m (y(\mathbf{x}_i) - \hat{y}(\mathbf{x}_i))^2} \quad (7.4)$$

7.2.3 Normalized Root Mean Squared Error (NRSME)

The Normalized Root Mean Squared Error normalizes the RMSE by the range of the true function values. It is defined as (Eq. 7.5), but could also use the mean value of the data or the standard deviation as the denominator. The mean value is often used when the dataset has the same order of magnitude is not close to 0. The standard deviation makes the error measure a relation to the natural spread of the data.

$$\text{NRMSE} = \frac{\text{RMSE}}{\max(y) - \min(y)} \quad (7.5)$$

7.2.4 R Squared

R Squared (also called the Coefficient of Determination) is calculated according to Eq. 7.6 and is used to determine how much of the variance of y can be explained by the model \hat{y} . \bar{y} is the average value of y . $R^2 = 1 - FVU$ where FVU is the fraction of unexplained variance. An R^2 value of 0.9 means that the model \hat{y} models 90% of y and therefore ignores 10% of the variance of the problem.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2}{\sum_{i=1}^m (\bar{y} - y^{(i)})^2} \quad (7.6)$$

SS_{res} is the residual sum of squares and SS_{tot} is the total variance in the data.

An R^2 value close to 1 indicates a good fit, whereas a value close to 0 indicates a poor fit. A negative R^2 value indicates that the surrogate model is worse than just using \hat{y} regardless of the values of the design variables at the point that should be estimated.

Adjusted R Squared

R^2 increases when the number of terms increases (Parnianifard et al. (2019)) and a solution might be to use the adjusted R^2 instead. The variable m is the number of designs, whereas n is the number of design variables that were used to build the \hat{y} model.

$$R^2 = 1 - \frac{m-1}{m-n}(1 - R^2) \quad (7.7)$$

The adjusted R^2 increases when the addition of a design variable improves the surrogate model estimation more than pure chance. It can be used to find the optimal design variables to include in the surrogate model by fitting surrogate models with different combinations of included design variables. The combination that results in the highest adjusted R^2 is the one with the best fit.

7.2.5 Relative Average Absolute Error (RAAE)

This provides a global measure of model accuracy. This is calculated for all m points as in Eq. 7.8.

$$RAAE = \frac{\sum_{i=1}^m |y_i - \hat{y}_i|}{m \cdot \sigma_y} \quad (7.8)$$

It is possible to replace the σ_y in the denominator to get Eq. 7.9, which can be seen as the average percentage error of the model's predictions.

$$RAAE = \frac{1}{m} \sum_{i=1}^m \frac{|y_i - \hat{y}_i|}{y_i} \quad (7.9)$$

7.2.6 Relative Maximum Absolute Error (RMAE)

This measure assesses the local performance and identifies the worst-case prediction error in the design space.

$$RMAE = \frac{\max(|y_i - \hat{y}_i|)}{\sigma_y} \quad (7.10)$$

7.2.7 Cross-Validated Q^2

This metric measures the model's predictive power on unseen data. A significant gap between R^2 and Q^2 indicates overfitting. The equation (7.11) is similar to the R^2 but is used in Cross-Validation as described in section 7.3.1

$$Q^2 = 1 - \frac{SS_{press}}{SS_{tot}} \quad (7.11)$$

7.2.8 Example of Accuracy Measures

The estimated values from PRS and RBF can be inserted into the accuracy measures together with the true values. The different measures are compiled in Table 7.3 together with the underlying values.

Table 7.3: The true and predicted values for the validation set using the PRS and RBF surrogate models. The bottom part displays numerical measures of the accuracy of the two surrogate models.

yVal	yPRS	yRBF
3145	994.9	1592.7
191.72	100.3	207.6
343.4	184.7	276.4
9	-183.3	-24.9
209.32	-611.3	119.2
65	59.0	3.1
466.92	536.7	568.7
185	-614.5	71.7
207.4	312.4	-40.5
310.12	338.5	18.0
2705	1547.4	2159.0
R^2	0.397	0.580
R^2_{ADJ}	0.330	0.534
$NRSE_{mean}$	1.148	0.720
$NRSE_{range}$	0.261	0.164
RAAE	2.955	0.826
MAE	507.2	283.8

It can be noted that the Radial Basis Functions model outperforms the Polynomial Response Surface for all measures. Therefore, the RBF model should be the one the engineer chooses to use.

7.3 Assessment Strategies

The easiest way to perform a validation is to divide the available samples into two sets, one training set and one validation set. This may be called hold-out sampling since some designs are held out and saved for validation. A common distribution is 80% of the samples for training and 20% for validation. The procedure is such that:

1. One or more surrogate model(s) is/are trained using the training set
2. The surrogate model is used to estimate the values of the validation set
3. The predictions are compared to the true values to assess the performance of the surrogate model
4. The best surrogate model is selected as the one that should be used

The disadvantages are that the statistics are not averaged over several data sets and that not all points are used for training the SM.

Engineering models are often computationally expensive, which means that the datasets are small. This makes it undesirable to save samples for validation instead of using all samples for training.

7.3.1 Iterative Methods

The other methods are iterative methods where m different models are created using different parts of the dataset. The models that are created are only intended as diagnostic tools to assess the accuracy of the SM and tweak its parameters.

This will yield m error values that can be used with Eq. 7.12. This reveals how much the prediction of a model can be trusted if this surrogate model type is trained with these settings. This also means that this metric can be used to prove that the model is robust and not only use the final model's accuracy measures to measure how useful the model is.

$$\text{RMSE}_{cv} = Q^2 = \sqrt{\frac{1}{m} \sum_{i=1}^m (y_i - \hat{y})_i^2} \quad (7.12)$$

The model that should be used for predictions afterwards should be created using ALL samples in the dataset. The reason for not using the best of the m models created earlier is that each of them has one or more points that were left out. It is preferable to use all samples for training. Additionally, it might be that one of the models performed well because the point(s) left out of the training was easy to predict accurately.

7.3.2 Cross-Validation

Cross-Validation is a collection of approaches in which one or more designs are omitted from the training of the SM and are instead used for validation. This procedure is repeated in rounds, with different designs selected as the validation set for each round.

Cross-validation is mostly useful for small data sets, where saving samples for validation makes the SM too inaccurate because too few samples are left to train the SM.

Leave-1-Out Cross-Validation (LOOCV)

This method means that one training point at a time is set aside from the training set and a surrogate model created using all other points.

1. Set aside 1 point from the dataset
2. Create a surrogate model using all other points in the dataset
3. Compare the prediction to the value of the point you set aside
4. Repeat this for all m points in your dataset
5. Calculate the overall error

This provides an almost unbiased prediction of how the model should perform on new data.

Leave-P-Out Cross-Validation

The leave-p-out method means that the training dataset is divided into two different datasets, where the training set contains $m-p$ designs and the validation set contains p designs. It operates similarly to LOOCV with the exception that all combinations of p designs should be tested.

This means that the required number of combinations (and models) that need to be tested is calculated according to Eq. 7.13. A dataset with 30 samples and a chosen p -value of 3 results in 4 060 evaluations, which is a high number. Hence, p should be set to a low value.

$$N_{\text{eval}} = \binom{m}{p} = \frac{m!}{p!(m-p)!} \quad (7.13)$$

K-fold Cross-Validation

The k-fold cross-validation method means that the training dataset is divided into k different datasets. k models are then created using $k-1$ of the datasets for training and one of the folds to calculate accuracy measures.

It is similar to the LOOCV method, except that more than one point is left for accuracy assessment. If you have 60 samples and choose 5 folds, you will train each of the five models using 48 points and save 12 samples for accuracy.

7.3.3 Resampling Methods

Resampling methods are very similar to the cross-validation methods mathematically but serve a different purpose. Cross-validation aims to estimate the predictive performance of the SM, whereas resampling methods try to estimate the statistical properties (such as bias, variance and confidence intervals) for a prediction or model parameter.

If you are checking whether your model is accurate, you are performing Cross-Validation. If you are checking if your model is biased or uncertain, you are performing Resampling.

Resampling has three different advantages:

- It is possible to give a confidence interval for the predictions. For example - The predicted stress is $250 \text{ MPa} \pm 15 \text{ MPa}$
- You can identify high-risk regions where your model is uncertain if your resampling surrogate models agree in large areas of the design space, but differs in other regions.
- If the surrogate model's parameters vary drastically when one of the training points have been left out, the resampling method has found an outlier.

This section presents two methods - Jackknifing and Bootstrapping. If the engineering data is mostly normal and there are very few samples ($m < 15$), the Jackknife is usually more stable. If the model is highly complex and non-linear (like a Neural Network predicting turbulent flow), the Bootstrap is more honest because it captures the actual, often messy, distribution of errors.

Jackknifing

Jackknifing is a resampling method proposed by Quenouille, 1949. The goal is to estimate the bias and variance of a property or variable. In surrogate modeling for engineering, it is the same as LLOCV if the predicted and true values of the training dataset are compared to obtain an estimate of the model accuracy. However, it can also be used to calculate the mean value, standard deviation and confidence interval of the parameters of the surrogate model.

The general formulation of Jackknifing is as follows. Given a dataset of size m , the jackknife method involves iteratively leaving out one observation,

computing the statistic of interest, and repeating this process for all m observations.

The jackknife estimate of a parameter θ is given by Eq. 7.14 where $\theta_{(i)}$ is the statistic computed by leaving out the i -th observation.

$$\theta_{(\text{jack})} = \frac{1}{m} \sum_{i=1}^m \theta_{(i)}, \quad (7.14)$$

When used to estimate the model parameters, the workflow is as follows:

1. Compute Pseudo values. For a global estimate $\hat{\theta}$ and m leave-one-out estimates $\hat{\theta}_{(i)}$, the i -th pseudo value is calculated according to Eq. 7.15

$$P_i = m\hat{\theta} - (m-1)\hat{\theta}_{(i)} \quad (7.15)$$

2. Estimate Jackknife Variance. The variance of the estimate is derived from the variance of the pseudo values according to Eq. 7.16, where \bar{P} is the mean of the pseudo values.

$$\sigma_{jack}^2 = \frac{1}{n-1} \sum_{i=1}^n (P_i - \bar{P})^2 \quad (7.16)$$

3. Define the Interval. The $100(1-\alpha)\%$ confidence interval is then calculated according to Eq. 7.17, where t is the student's t-distribution.

$$\left[\bar{P} - t_{\alpha/2, m-1} \frac{\sigma_{jack}}{\sqrt{m}}, \quad \bar{P} + t_{\alpha/2, m-1} \frac{\sigma_{jack}}{\sqrt{m}} \right] \quad (7.17)$$

If the confidence interval is very wide at a specific design point, it indicates that the surrogate model is highly sensitive to the presence of specific training points in that region. This is a clear signal that more local sampling is required.

Bootstrapping

Bootstrapping is a non-parametric resampling technique used to estimate the sampling distribution of a statistic or the uncertainty of a surrogate model's predictions (Efron, 1992). It is similar to Jackknifing except that it selects n training points with replacement. This means that the same point can be selected several times in the same sub dataset.

The overall method is as follows:

1. Create a bootstrap sample \mathcal{D}^* by selecting n observations from the original dataset \mathcal{D} at random with replacement.

2. Train the surrogate model or calculate the statistic $\hat{\theta}^*$ using the sample \mathcal{D}^* .
3. Repeat this process B times (typically $B \geq 1000$) to obtain a distribution of estimates $\{\hat{\theta}_1^*, \hat{\theta}_2^*, \dots, \hat{\theta}_B^*\}$.

The possibility of repeating the process means that enough repetitions can be made to create a probability distribution based on the data. For a prediction \hat{y} , the $(1 - \alpha)\%$ confidence interval can be derived directly from the percentiles of the bootstrap distribution:

$$CI = [\hat{y}_{(\alpha/2)}^*, \hat{y}_{(1-\alpha/2)}^*] \quad (7.18)$$

In engineering, Bootstrapping can be used to:

- Determine the "error bars" of a Neural Network prediction without needing a validation set.
- Create a forest of models (the same reasoning as Random Forest) where the final prediction is the average of all bootstrap models, significantly reducing the variance of the prediction.
- Extract maximum statistical information from expensive, low- m experimental data.

Chapter 8

Practical Guidelines

This chapter presents comments and considerations that may be more or less useful when applying surrogate modeling techniques.

The overall thinking for the model creation should often be.

1. What is my hypothesis for the appearance of the output with regards to the input?
2. This gives a recommendation for a suitable SM type
3. When an SM type is selected, we can use a recommended sampling plan.

8.1 How to Utilize the Dataset

If it takes a long time to perform simulations/experiments to add more designs to the dataset, it is important to use the data efficiently. This can be done in several ways. The methods are exemplified with a dataset containing 100 samples.

1. Skip screening and try to fit a model with all 100 points. This is recommended for beginners and problems with few variables.
2. Use 30 designs for screening and 70 for model training. This is good if the screening reveals that only five design variables matter and you can create an accurate model with the remaining 70 samples.
3. Use 30 designs for screening and use all 100 designs for training a model anyway. This is recommended for average users.
4. Use your 100 designs to create a surrogate model with all variables. Then perform a sensitivity analysis on the surrogate model (not the original model) to identify the important variables. The final step is to create a surrogate model based on all 100 training points, but

only including the important variables. This is recommended for more advanced users.

The last method is not valid if you have many ($n > 20$) design variables. The first model will probably be too inaccurate to get a correct sensitivity information. In this case, it is recommended to use Morris trajectories to get the physics right first.

If you have a low-fidelity version of your code, run Morris (section 8.5.1 trajectories on the cheap version to pick variables, then run the expensive sampling plan on the high-fidelity version.

8.2 Selecting Surrogate Model

Choosing the correct surrogate model is a trade-off between the probable appearance of the output, the number of inputs (n), the available computational budget for samples (m), and the required accuracy. There is no "one-size-fits-all" solution, but the following guidelines is suitable for many engineering projects.

8.2.1 Guidelines Based on Dimensionality and Sample Size

The selection process generally follows a trajectory based on the density of the data relative to the input space:

- **Low-Dimensional Space ($d < 20$):** For most mechanical and structural problems with fewer than 20 variables, **Kriging (Gaussian Process Regression) is a great choice**. It is particularly effective at local interpolation and also provides an estimate of prediction uncertainty, which is invaluable for adaptive sampling and optimization.
- **High-Dimensional, Small Dataset ($d > 20, n < 500$):** This is a challenging situation. When the number of design variables is large but simulations are expensive (limited n), **Support Vector Regression (SVR)** is often a good choice. SVR's complexity is controlled by the number of support vectors rather than the dimensionality of the input, making it resistant to the "Curse of Dimensionality."
- **High-Dimensional, Large Dataset ($d > 20, n > 1000$):** When we have a huge amount of data, **Neural Networks (NN)** are the most powerful tool. They can capture complex, multi-modal, and non-linear interactions that might be too computationally expensive for Kriging to solve or too complex for SVR kernels to map effectively.

8.2.2 Comparison Summary

Table 8.1: Selection Matrix for Surrogate Modeling

Model	Samples (m)	Dim (d)	Primary Use Case
Kriging	Small/Med	Low/Med	Optimization needing uncertainty bounds.
PCE	Small/Med	Low	Global sensitivity analysis (Sobol indices).
SVR	Small	High	Robust modeling with noisy simulation data.
NN	Large	High	Highly non-linear, multi-physics surrogates.

8.3 Selecting Sampling Plan

If your DoE does not provide the "right" kind of information for your chosen model's mathematical structure, the surrogate will struggle regardless of how much you tune its hyperparameters.

8.3.1 Aligning Sampling Strategy with Model Choice

A critical, yet often ignored, step in surrogate modeling is ensuring that the Design of Experiments (DoE) provides the specific type of information required by the model's mathematical architecture.

This can be summarized in table 8.2 and the following points

- Global Interpolators (Kriging/Gaussian Processes): These require Space-Filling Designs (LHS, Sobol). Since Kriging relies on the spatial correlation between points, it needs to "see" the entire landscape evenly to build an accurate covariance matrix.
- Global Approximators (RPS): These are often best served by classical designs (Central Composite, Box-Behnken). These designs focus on the boundaries and center of the design space to stabilize the estimation of polynomial coefficients.
- Expansion Methods (PCE): These frequently require Quadrature or Collocation points—specific locations dictated by the roots of orthogonal polynomials (like Gauss-Legendre) to calculate coefficients with high mathematical precision.
- Data-Driven Models (NN/SVR): These are less picky about where points are, but they need a dataset with high density of points in the design space. However, using a random or quasi-random (Sobol) sequence is preferred over a grid to avoid "collapsing" dimensions where one variable does not matter.

8.3.2 Recommended Pairings

The following table outlines some recommendations regarding combinations of sampling strategies and surrogate model types:

Table 8.2: Recommended DoE and Surrogate Model Pairings.

Model Type	Preferred DoE	Reasoning
Polynomials	CCD, Box-Behnken	Minimizes the variance of the predicted coefficients (β).
Kriging / GP	LHS, Sobol, Halton	Needs space-filling to accurately model the spatial correlation (kriging kernel).
PCE	Gauss-Quadrature	Uses specific integration points to solve for coefficients analytically.
NN / SVR	LHS, Sobol	Prevents aliasing and ensures a high-density coverage of non-linear regions.

For modern engineering surrogates, LHS is the industry standard. Since LHS ensures that each input variable is sampled at m different levels it results in a projection-robust design. However, if the goal is optimization or creating a model that is accurate in interesting areas in the design space, then an adaptive sampling method is recommended. This means that an initial small DoE is performed, and subsequent points are added iteratively in regions where the surrogate is either highly uncertain or predicts an optimal value. This is described in [9.1](#).

8.4 Sampling Strategies

Whereas DoEs are methods used to find the best possible experiment configuration to create an accurate surrogate model, there are things the user should consider when using them.

Some considerations are

- accuracy vs complexity
- local vs global accuracy
- interpolation vs extrapolation
- interpolation vs regression
- normalization of the training data
- handling of outliers

8.4.1 Accuracy vs. Complexity

Generally, a larger number of training samples increases the accuracy of an SM, especially when using space-filling DoEs. The main disadvantage is that each additional sample requires an additional simulation or experiment. Furthermore, too many samples might result in overfitting (especially in PRS) or distortion of the model (e.g., kriging).

8.4.2 Global vs. Local Models

The accuracy of a surrogate model is usually higher closer to the training points. This is especially true for interpolating surrogate models, where the predicted value depends on the distance from the training points. Consequently, the placement of the training points is important for the accuracy of the surrogate model. It is usually too computationally expensive to perform enough experiments/simulations to get enough training samples to cover the whole design space accurately. Ideally, there should be at least one sample close to the region of the design space which the decision maker considers to be the most interesting region. At the same time, the SM should capture the overall behavior of the original model to be able to guide the optimization properly. The latter is especially important in MDO, where it is usually difficult to predict where in the design space the optimization will end up. Therefore, the decision maker must carefully balance global accuracy versus local. A recommendation is to use most of the computational budget on global accuracy and just a few samples in promising areas. If the SM is used for constraints, the actual value is important since it will be compared to a limit.

8.4.3 Model Interpolation and Extrapolation

Regardless of whether the SM is an interpolating model or not, it is NOT recommended to use SMs for extrapolation. SMs are usually created to approximate an entity in a design space. Estimations outside of the original design space should not be trusted too much. Consequently, the sampling plan should cover the intended usage of the SM, and it is important to communicate the intended design space and usage for the SM when sharing it with someone else.

8.5 Handling Training Data before SM Training

Once a dataset has been obtained, it might be useful to do a few tests and modify the data before creating an SM.

8.5.1 Reveal Dependencies and Trends

It could be useful to identify dependencies, correlations and trends to see if the dataset can be reduced. If an input variable has a low impact on the output, or a 100% correlation with another variable, it might be possible to eliminate it from the dataset. This is also true if the variable has close to zero impact on the entity that the SM should predict.

Useful tools for discovering correlations and impacts include, among others, calculating correlations numerically or using a Scatter Plot Matrix.

The Morris Method for Global Sensitivity Screening

The Morris, [1991](#), method is a sensitivity analysis technique used to categorize input variables into three groups:

- negligible
- linear and additive
- non-linear and/or involved in interactions

The method is based on the Elementary Effect (EE) concept. For a given design point \mathbf{x} , the elementary effect of the i -th input is calculated by varying that variable by a relative distance Δ according to Eq. [8.1](#). This is the formula for a partial derivative and estimates the partial derivative of f versus x_i .

$$EE_i(\mathbf{x}) = \frac{f(x_1, \dots, x_i + \Delta, \dots, x_n) - f(\mathbf{x})}{\Delta} \quad (8.1)$$

Instead of sampling the entire space randomly, the Morris method uses r paths/trajectories. Each trajectory starts at a random point and moves through the design space by changing one variable at a time, requiring $n + 1$ simulations per trajectory.

The method completes r trajectories in the design space and then calculates two measures for each variable.

1. The Mean of Absolute Values (μ^*): This measures the overall influence of the variable on the output (Campolongo, Cariboni, and Saltelli, [2007](#)) and is calculated using Eq. [8.2](#).

$$\mu^* = \frac{1}{r} \sum_{j=1}^r |EE_{i,j}| \quad (8.2)$$

2. The Standard Deviation (σ): This measures the non-linearity of the variable or its interactions with other variables. A high σ indicates

that the variable's effect depends strongly on the values of other inputs. This measure is calculated according to Eq. 8.3.

$$\sigma_i = \sqrt{\frac{1}{r-1} \sum_{j=1}^r (EE_{i,j} - \mu_i)^2} \quad (8.3)$$

The results are typically visualized on a μ^* vs. σ graph. Variables located near the origin are not important and should be removed from the design space before building the final surrogate model.

The total number of simulations required for the Morris method is calculated according to Eq. 8.4, where r is typically between 5 and 15. This makes the method significantly cheaper than variance-based methods (like Sobol) for high-dimensional problems.

$$N = r \times (n + 1) \quad (8.4)$$

,

Scatter Plot Matrix

It might be a good idea to plot the dataset in, for example, a scatter plot matrix. Humans are excellent at identifying trends and correlations from graphs such as scatter plots. This may help to identify global trends or input variables that do not significantly affect the output. Such variables could be eliminated from the dataset, making the design space coverage of the SM one dimension smaller.

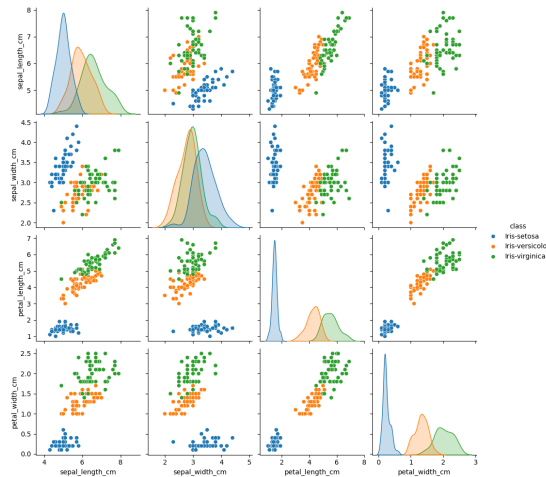


Figure 8.1: Example of a scatter plot matrix with three different "products". The data is from the Iris dataset which depicts flower species.

Calculating Correlations

Two common correlation metrics are the Pearson correlation and the Spearman correlation.

Pearson Correlation The Pearson correlation coefficient, ρ , reveals how much the dependence of two entities, x and y , resembles a straight line. The formula is shown in Eq. 8.5, where \hat{y} is the mean value of y .

$$\rho = \frac{\sqrt{\sum_{i=1}^n (x_i - \hat{x})^2 (y_i - \hat{y})^2}}{\sqrt{\sum_{i=1}^n (x_i - \hat{x})^2} \sqrt{\sum_{i=1}^n (y_i - \hat{y})^2}} \quad (8.5)$$

Spearman Correlation The Spearman's rank correlation coefficient, r_s reveals how strongly two set of ranks are correlated. This means that the two entities, x and y , are sorted in ascending order and then their orders compared. If the orders are equal, ρ equals 1. If the orders of x and y are totally inverted, r_s equals -1.

Correlation example The two correlation coefficients are exemplified with the data in Table 8.3, where x and y have been sampled at four points. y is always increasing when x is increasing in this case, which results in a Spearman's rank correlation coefficient, $r_s = 1$. However, the Pearson correlation coefficient, $\rho = 0.7544$. The reason is that a scatter plot between x and y is not a straight line.

Table 8.3: An example of Pearson and Spearman Rank correlation coefficients

x	y
0	1
10	100
101	500
102	2000

8.5.2 Handling of Outliers

Outliers in the dataset can affect the SMs tremendously. Therefore, it is strongly recommended to identify and handle outliers before an SM is created. This is exemplified in figure 8.2, which consists of three sub-figures;

- The leftmost sub-figure shows what happens if we fit a model without handling the outlier
- The middle sub-figure shows what happens if the outlier is removed
- The rightmost sub-figure shows what happens if the value of the bad point is changed

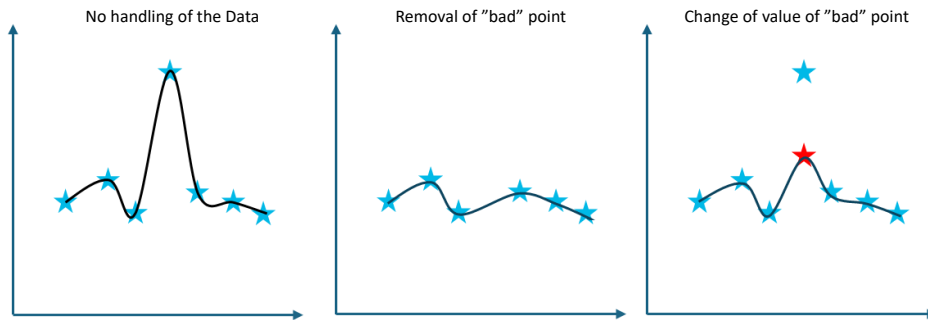


Figure 8.2: Three figures representing how to handle outliers in a dataset. The leftmost figure is without handling the outlier. The middle figure shows that we can remove the outlier, whereas the rightmost figure represent a change of the value of the outlier.

The SM type in the figure is an interpolating model since it goes through all points but the reasoning would be almost similar for a PRS. The outlier distorts the model locally, and if the magnitude of the outlier is large enough, it also distorts it globally.

If the outlier is removed, it does not distort the surrogate model, but we get a hole in the dataset. This means that there will be a hole in the dataset where the outlier was. This reduces the accuracy in that region of the design space, but it also means that we lose the information that this part of the design space has extreme values.

If we instead change the value of the outlier to the same order of magnitude as the data points surrounding it, but still make it worse than the other samples, we retain the information that this is a bad/suspicious area. The SM will be somewhat distorted in that area, but hopefully not globally. This would be our dream scenario.

8.5.3 Removing trends?

Does the data seem to follow any polynomial trend? In that case, remove the trend from the data and let another SM model the local deviations. This means that our model is a combination of a polynomial and another SM.

Computer models are often deterministic, whereas experiments are stochastic. See the paper airplane, for example. This makes it desirable to recreate each experiment a few times to reduce statistical uncertainty, or to choose

a regression model that smooths out the noise/variation. The problem with repeating experiments is that it is time-consuming and the whole idea with using SMs is to minimize the number of experiments/simulations of the original model.

8.5.4 Normalization of the Training Data

Data where the inputs are of different orders of magnitude should usually be normalized before training the SM. However, most software intended for surrogate modeling has this implemented into them. This means that the user does not need to consider normalization. However, in some rare cases, it might be beneficial to normalize the data anyway if the user wants to control the normalization.

8.6 Using Multiple Surrogate Models for the Same Entity

The idea with SMs is that they should be used in cases where it is faster to train a new SM and call it many times than to perform a new evaluation of the original model. With a large enough difference in wall clock time, it is possible to create several SMs and utilize the information from all of them.

This is especially true when the SMs are used in optimization since different SMs often return different optima. If several optima are clustered, it means that several SMs found the same optima and it is therefore probable that it is a good solution. If we do not find any clustered optima, it means that it was clever to use several SMs since they return different optima. It is probable that more training data is needed since the SMs cannot agree upon where the best areas in the design space are.

Additionally, the spread in estimated value when a point is estimated by different SMs is important. A small spread increases the confidence that the estimation(s) is accurate, while a large spread indicates that the accuracy of the SMs in that area of the design space is poor. Consequently, it is recommended to add at least one training point there.

The strategy when using multiple SMs can be divided into two categories.

- Select the best SM for the evaluations
- Use a combination of all SMs to evaluate the points - An ensemble of SMs

In both categories, the accuracies of the different SMs need to be estimated using metrics from section 7.2.

8.6.1 Ensembles of Surrogate Models

It is recommended to use an ensemble of SMs when estimating the value of an interesting design unless the training time of creating an additional model is too large. In most cases, it should be realistic to train a few different SMs, e.g. a PRS, a Kriging model and one of NN/RBF/SVM.

While the main drawback is increased computational time since each design needs to be estimated using more than one SM, there are several benefits to using an ensemble of SMs;

- Improved Accuracy – By aggregating multiple models, the ensemble can average out individual errors and better approximate the true system response.
- Uncertainty Quantification – Variability among surrogate predictions can provide a proxy for model confidence, helping engineers identify regions where additional simulation or testing is needed.
- Robustness Across Design Space – Different surrogates may perform better in different regions (e.g., low-dimensional vs. highly nonlinear subspaces). An ensemble adapts to these variations.
- Reduced Risk of Overfitting – Ensembles often generalize better than a single over-tuned model.

The easiest method to use an ensemble of SMs is to simply take the average of their estimations. This is exemplified in Eq. 8.6 where \hat{y} is the estimation and is based on the \hat{y}_i estimations from the n SMs.

$$\hat{y} = \frac{1}{n} \sum_{k=1}^n \hat{y}_i \quad (8.6)$$

It is also possible to combine SMs with a linear weighting as exemplified in Eq. 8.7. The weights, λ_i , are determined such that the SM that is believed to have the highest accuracy has the highest λ_i value and the worst SM has the lowest.

$$\hat{y} = \sum_{k=1}^n \lambda \hat{y}_i \quad (8.7)$$

$$\sum_{\lambda=1}^n = 1 \quad (8.8)$$

$$\lambda_i > 0 \quad (8.9)$$

8.6.2 Two-step Surrogate Model Estimations

Yan et al. (2020) describes an approach similar to the reasoning by Isaaks and Srivastava (1989) that the trend in the data should be handled and then use an interpolating SM. Yan et al. (2020) begins by fitting a polynomial response surface to the data. The difference between the regression model $\hat{y}_r^{(i)}$ and the true value $y_{true}^{(i)}$ at training point i , (the error), here denoted $y_d^{(i)}$, is defined according to Eq. 8.10.

$$y_d^{(i)} = \hat{y}^{(i)} - y_{true}^{(i)}, \quad (8.10)$$

An interpolating SM is then trained using the $y_d^{(i)}$ as output it should predict. The estimation of a point will then be made using both the regression model and the interpolation model. Consequently, the regression handles the global trend and the interpolation model the local variations. This is exemplified in Eq. 8.11 where $\hat{y}_{(r)}$ is the regression model and $\hat{y}_{(i)}$ is the interpolation model.

$$\hat{y} = \hat{y}_{(r)} + \hat{y}_{(i)}, \quad (8.11)$$

8.7 Using one SM for each entity

It is almost always recommended to model each entity using a separate SM instead of attempting to aggregate several disciplines into one SM.

For FEM and CFD models, there are some additional factors to consider. A mesh convergence test is often required to ensure that the FEM/CFD calculations are not affected by a poor mesh. It is often useful to model different parts of the geometry with different SMs, since the location of the highest stress or deflection might change when the geometry or boundary conditions change.

Chapter 9

Optimization with Surrogate Models

Surrogate models are often used to reduce the wall clock time of simulations to allow optimization. This means that the SM is used instead of the original model whenever the objective value of a point should be evaluated.

The easiest way to perform an optimization with an SM is to:

1. Choose a Sampling Plan / DoE that determines which samples should be created.
2. Perform simulations/experiments according to the Sampling Plan
3. Optional: Plot the data to see the shape of the phenomena. This can guide the selection of the most appropriate surrogate model
4. Fit a surrogate model to the data
5. Perform an optimization where the SM is used instead of the original model/experiment
6. Verify the optimum by simulating the original model or experiment

This approach is fairly straightforward and relatively easy to understand and implement. However, since the purpose of using a SM is to replace experiments or computationally expensive models, it is desirable to utilize the gathered data as much as possible. Therefore, it is recommended to use a more advanced strategy.

9.1 More Advanced Surrogate Based Optimization

This section describes a strategy for performing optimization with surrogate models in a structured and efficient manner. This method is only valid if

you can perform new evaluations of your original model(s)

1. Choose Sampling Plan
2. Perform simulations/experiments according to the Sampling Plan
3. Plot the data to identify bad designs
4. Change the value of bad designs so that they do not degenerate the Surrogate Models
5. Fit Surrogate Models to data
6. Perform optimizations on each Surrogate Model
7. Identify potential clusters of optima
8. Perform simulations/experiments to confirm/validate the optima
9. Optional: Add data points in unknown areas of the design space
10. Use these to update the Surrogate Models and go to step 4. This process continues until the user is satisfied

This iterative way of updating surrogate models is called several things depending on the variant, e.g., Efficient Global Optimization (EGO) by Jones (2001) or Bayesian Optimization, but here it will be referred to as Sequential Approximate Optimization (SAO).

Adding points at the optima of the SMs serve two purposes - verifying that the optima of the SMs actually are good points and to increase the accuracy of the SMs at their optima. This is similar to the exploitation function of an optimization algorithm.

Adding data points in areas of the design space where not much is known might be useful to ensure that there are no better areas in the design space. This is similar to the exploration function of an optimization algorithm.

There are several strategies to choose the experiments / simulations that should be added to have an efficient SAO (Jones (*ibid.*)). Some of them and their advantages and drawbacks are shown in Table 9.1. The difference between the two last criteria is that the last criterion tries to add a point where the probability that the SM will improve is the largest. The second last criteria tries to add a point where the accuracy of the SM probably is the largest, i.e. the location in the design space where the RSME will decrease the most when a point is added to the SM.

- The optimum of the SM - this is an interesting area
- Furthest from the known points - we hope to gain information in an area where we do not know much

Table 9.1: A few popular infill criteria for surrogate based optimization

Criterion	Advantages	Disadvantages
Optimum of the SM	It is an interesting area	The SM might not capture the global appearance correctly
Furthest from known points	The SM is probably inaccurate there	Probably not an interesting area
Maximize the probable improvement of the SM	The SM will probably improve	Probably not an interesting area
Maximize the probability of improvement of the SM	The SM will probably improve	Probably not an interesting area

- The point where the probability of an improvement of the accuracy of the SM is the largest.
- The point where the expected improvement of the SM is the largest.
- A combination of some of the points above

Bibliography

- Alizadeh, Reza, Janet K Allen, and Farrokh Mistree (2020). “Managing computational complexity using surrogate models: a critical review”. In: *Research in Engineering Design* 31.3, pp. 275–298.
- Bhosekar, Atharv and Marianthi Ierapetritou (2018). “Advances in surrogate based modeling, feasibility analysis, and optimization: A review”. In: *Computers & Chemical Engineering* 108, pp. 250–267.
- Box, George EP and Donald W Behnken (1960). “Some new three level designs for the study of quantitative variables”. In: *Technometrics* 2.4, pp. 455–475.
- Breiman, Leo (2001). “Random forests”. In: *Machine learning* 45.1, pp. 5–32.
- Broomhead, David S and David Lowe (1988). “Multi-variable functional interpolation and adaptive networks”. In: *Complex Systems* 2, pp. 321–355.
- Campolongo, Francesca, Jessica Cariboni, and Andrea Saltelli (2007). “An effective screening design for sensitivity analysis of large models”. In: *Environmental modelling & software* 22.10, pp. 1509–1518.
- Chen, Hao et al. (2016). “Analysis Methods for Computer Experiments: How to Assess and What Counts?” In: *Statistical Science* 31.1, pp. 40–60. DOI: [10.1214/15-STS531](https://doi.org/10.1214/15-STS531).
- Cortes, Corinna and Vladimir Vapnik (1995). “Support-vector networks”. In: *Machine learning* 20, pp. 273–297. DOI: [10.1007/BF00994018](https://doi.org/10.1007/BF00994018).
- Drucker, Harris et al. (1996). “Support vector regression machines”. In: *Advances in neural information processing systems* 9.
- Efron, Bradley (1992). “Bootstrap methods: another look at the jackknife”. In: *Breakthroughs in statistics: Methodology and distribution*. Springer, pp. 569–593.
- Forrester, Alexander, András Sóbester, and Andy Keane (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons. DOI: [10.2514/4.479557](https://doi.org/10.2514/4.479557).
- Hammersley, John M (1960). “Monte Carlo methods for solving multivariable problems”. In: *Annals of the New York Academy of Sciences* 86.3, pp. 844–874.

- Havinga, Jos, Gerrit Klaseboer, and AH Van den Boogaard (2013). “Sequential optimization of strip bending process using multiquadric radial basis function surrogate models”. In: *Key engineering materials*. Vol. 554. Trans Tech Publ, pp. 911–918.
- Ho, Tin Kam (1998). “The random subspace method for constructing decision forests”. In: *IEEE transactions on pattern analysis and machine intelligence* 20.8, pp. 832–844.
- Hwang, John T and Joaquim RRA Martins (2018). “A fast-prediction surrogate model for large datasets”. In: *Aerospace Science and Technology* 75, pp. 74–87.
- Isaaks, Edward H and R Mohan Srivastava (1989). *Applied geostatistics*. Oxford University Press.
- Jin, Ruichen, Wei Chen, and Timothy W Simpson (2001). “Comparative studies of metamodelling techniques under multiple modelling criteria”. In: *Structural and Multidisciplinary Optimization* 23.1, pp. 1–13.
- Jones, Donald R (2001). “A taxonomy of global optimization methods based on response surfaces”. In: *Journal of global optimization* 21.4, pp. 345–383. DOI: [10.1023/A:1012771025575](https://doi.org/10.1023/A:1012771025575).
- Jones, Donald R, Matthias Schonlau, and William J Welch (1998). “Efficient global optimization of expensive black-box functions”. In: *Journal of Global optimization* 13.4, pp. 455–492.
- Krawczyk, Konrad and Jarosław Arabas (2025). “Single-Objective Surrogate Models for Continuous Metaheuristics: An Overview”. In: *Applied Sciences* 15.16, p. 9068.
- Krige, DG (1951). “A statistical approach to some basic mine valuation problems on the Witwatersrand”. In: *Journal of Chemical, Metallurgical, and Mining Society of South Africa*.
- Li, Genyuan, Carey Rosenthal, and Herschel Rabitz (2001). “High dimensional model representations”. In: *The Journal of Physical Chemistry A* 105.33, pp. 7765–7777.
- Liu, Haitao, Yew-Soon Ong, and Jianfei Cai (2018). “A survey of adaptive sampling for global metamodelling in support of simulation-based complex engineering design”. In: *Structural and Multidisciplinary Optimization* 57.1, pp. 393–416.
- Lophaven, Sören, Hans Bruun Nielsen, and Jacob Søndergaard (2005). “Automatic mapping of monitoring data”. In.
- McCulloch, Warren S and Walter Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.
- McKay, Michael D, Richard J Beckman, and William J Conover (1979). “Comparison of three methods for selecting values of input variables in the analysis of output from a computer code”. In: *Technometrics* 21.2, pp. 239–245.

- Morris, Max D (1991). “Factorial sampling plans for preliminary computational experiments”. In: *Technometrics* 33.2, pp. 161–174.
- Myers Raymond H; Montgomery, Douglas C and Christine M Anderson-Cook (2009). *Response surface methodology: process and product optimization using designed experiments*. Vol. 705. John Wiley & Sons.
- Nakayama, Hirotaka, Masao Arakawa, and Rie Sasaki (2001). “A computational intelligence approach to optimization with unknown objective functions”. In: *Artificial Neural Networks-ICANN 2001*. Springer, pp. 73–80.
- Park, Hong-Seok and Xuan-Phuong Dang (2010). “Structural optimization based on CAD-CAE integration and metamodeling techniques”. In: *Computer-Aided Design* 42.10, pp. 889–902.
- Parnianifard, Amir et al. (2019). “Recent developments in metamodel based robust black-box simulation optimization: An overview”. In: *Decision Science Letters* 8.1, pp. 17–44.
- Plackett, Robin L and J Peter Burman (1946). “The design of optimum multifactorial experiments”. In: *Biometrika* 33.4, pp. 305–325.
- Quenouille, Maurice H (1949). “Problems in plane sampling”. In: *The annals of mathematical statistics*, pp. 355–375.
- Rosenbrock, HoHo (1960). “An automatic method for finding the greatest or least value of a function”. In: *The Computer Journal* 3.3, pp. 175–184.
- Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). “Learning representations by back-propagating errors”. In: *nature* 323.6088, pp. 533–536.
- Sacks, Jerome et al. (1989). “Design and analysis of computer experiments”. In: *Statistical science*, pp. 409–423. DOI: [10.1214/ss/1177012413](https://doi.org/10.1214/ss/1177012413).
- Sarle, Warren S. (2002). “comp.ai.neural-nets FAQ”. In.
- Shepard, Donald (1968). “A two-dimensional interpolation function for irregularly-spaced data”. In: *Proceedings of the 1968 23rd ACM national conference*. ACM, pp. 517–524.
- Smola, Alex J and Bernhard Schölkopf (2004). “A tutorial on support vector regression”. In: *Statistics and computing* 14.3, pp. 199–222.
- Sobol, Ilya M (1967). “Distribution of points in a cube and approximate evaluation of integrals”. In: *USSR Computational mathematics and mathematical physics* 7, pp. 86–112.
- (1993). “Sensitivity estimates for nonlinear mathematical models”. In: *Mathematical modelling and computational experiments* 1.4, pp. 407–414.
- Sudret, Bruno (2008). “Global sensitivity analysis using polynomial chaos expansions”. In: *Reliability engineering & system safety* 93.7, pp. 964–979.
- Ulaganathan, Selvakumar et al. (2016). “A hybrid sequential sampling based metamodeling approach for high dimensional problems”. In: *Evolutionary Computation (CEC), 2016 IEEE Congress on*. Ieee, pp. 1917–1923.

- Viana, Felipe AC et al. (2014). “Special section on multidisciplinary design optimization: metamodeling in multidisciplinary design optimization: how far have we really come?” In: *AIAA Journal* 52.4, pp. 670–690.
- Xiu, Dongbin and George Em Karniadakis (2002). “The Wiener–Askey polynomial chaos for stochastic differential equations”. In: *SIAM journal on scientific computing* 24.2, pp. 619–644.
- Yan, Cheng et al. (2020). “Ensemble of regression-type and interpolation-type metamodels”. In: *Energies* 13.3, p. 654.