Optimally weighted ensembles of surrogate models for sequential parameter optimization

Echtenbruck, M.M.

Chapter 2

Preliminaries

In the following, all tools and basics are introduced that build the fundamentals of this thesis. This chapter is structured as follows: In Section 2.1 the general concept of a surrogate model is introduced as well as the concrete models used in this thesis. In Section 2.2 SPO in general is introduced as well as the SPO framework that was specially developed for the experiments carried out in this work. Finally, in Section 2.3 an overview of the different objective functions is given.

2.1 Surrogate Modeling

The term surrogate model denotes a function $\hat{y} : \mathbb{R}^d \to \mathbb{R}$ which is an approximation of the true objective function[1] $y : \mathbb{R}^d \to \mathbb{R}$, learned from a finite set of evaluations of the objective function. Surrogate models are used, when the objective function is not known, and a complete evaluation is not feasible since they approximate the behavior of the objective function and are therefore cheaper and faster respectively to evaluate.

For the experiments carried out in this thesis a large set of heterogeneous surrogate models is used. To facilitate the choice while at the same time ensuring that the resulting set of surrogate models is most diverse, all models that are available as part of the SPOT package [21] are included. Since this work also

[1] Typically continuous functions are considered, but there are exceptions. [19, 20]

aims at handling unknown models, all models, except for the Kriging based models, are plugged into the system using default settings as provided by the SPOT interfaces. For the Kriging models, the correlation function is specified via the settings. Models that would not run on default settings or fail during experiments are discarded. The surrogate models that are used for the experiments are briefly introduced in the following sections.

2.1.1 Linear Regression Based Models

Linear regression models are the simplest approach to modeling data. A thorough introduction to linear regression models and related topics is given in [22, 23], which the following introduction is also based on.
For a start, we consider an easy example, where the relationship between two variables is to be modeled, i.e., income and years of education. Assuming that the relationship between those variables is of a linear nature, a model of the form

$$y = \beta_0 + \beta_1 x + \varepsilon$$

can be used to model the data. Predictions based on this model will be denoted as $\hat{y} = \beta_0 + \beta_1 x$. Here, y is the response variable, for a given x value, or predictor variable. β_0 and β_1 are model coefficients that represent the intercept and slope of the linear model, and ε denotes the model error. In order to gain the model, these parameters have to be estimated. This can, for example, be done by minimizing the least squares criterion. To do this, we consider the prediction errors $r_i = y_i - \hat{y}_i$, or residuals, between the observed response and the predicted response from the model. Figure 2.1 shows an example fit of a linear regression model and the resulting prediction errors.

Using the least squares criterion, the sum of squared prediction errors has to be minimized:

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} (r_i)^2 = \min_{\beta_0,\beta_1} \sum_{i=1}^{n} (\beta_0 + \beta_1 x_i - y_i)^2 \ .$$

This can be calculated as a simple mathematical optimization problem [22, p. 62]. Given that more than one predictor variable has to be considered, the linear regression model would take the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \varepsilon \ ,$$

where n is the number of predictor variables and β_i refers to the i-th predictor variable.

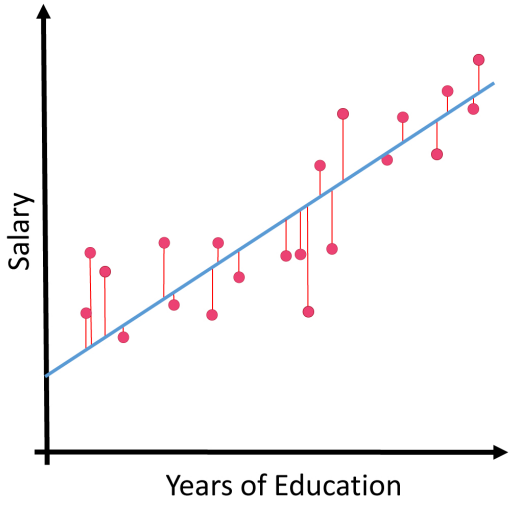


Figure 2.1: Example for a fit of a linear model. The blue line represents the linear model, the red dots the known data and the red lines between them depict the residuals or error made by the prediction for each known point. The model tries to find a linear relationship between the data that minimizes the sum of squared errors and thus obtains a straight line that models the data best under the assumption that the underlying relationship is linear.

However, this model, as specified so far, is based on the strict assumption that the relation between the predictor variables and the response variable is additive and linear [22].

Extensions to the linear model enable a relaxation of these assumptions by allowing interaction terms, i.e.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2 + \varepsilon$$

or quadratic terms, i.e.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \varepsilon,$$

also.

The experiments carried out in this thesis also use a linear model ('LM') with a response-surface component, based on the `rsm` package [24], which again is an extension to the R core functionality `stats::LM`. This model allows for interaction terms and quadratic terms also, if the number of predictor variables is sufficient.

2.1.2 Tree Based Models

Classification and Regression trees were first introduced by Breiman et al. [25]. James et al. also give a thorough and comprehensible introduction to both types of trees [22]. Later, Breiman et al. introduced a method to combine a large number of trees into one more accurate prediction model, known as random forest [26].

Tree-based models, as many other models also, can be used for classification as well as for regression. In the case of classification, a binary tree is learned, where nodes and adjoint labeled branches represent decisions while their terminal nodes, which will be referred to as leaves, represent the class that the data is assigned to. Figure 2.2 shows an intuitive example for such a classification tree, given by

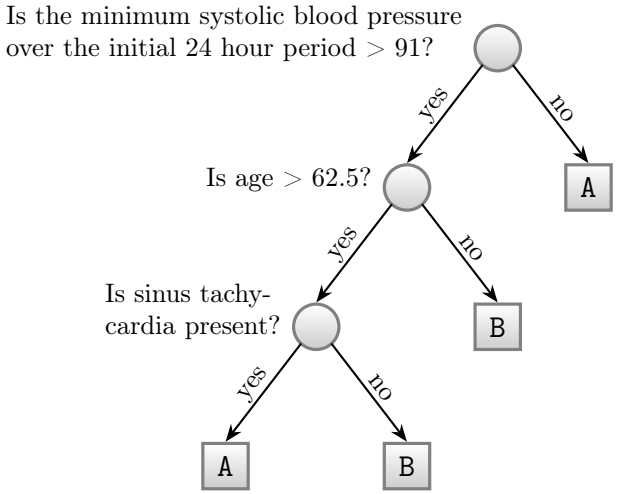
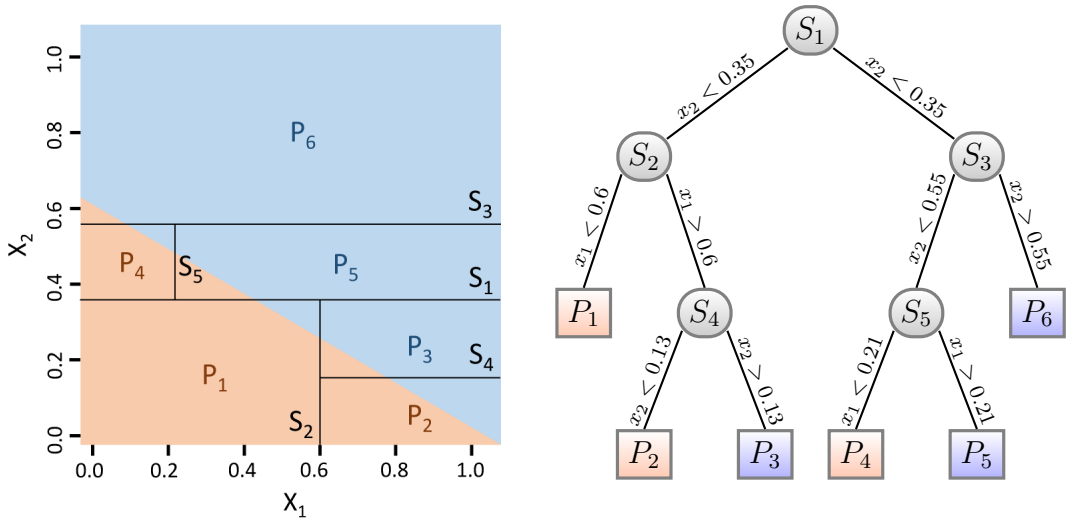


Figure 2.2: Classification tree for the identification of high-risk heart attack patients. Based on [25]. Each internal node marks a split based on the decision to be made for that node, here the related label shows this decision. The terminal nodes, or leaves, represent the class of the observations that fall here. In this example 'A' denotes low-risk patients whereas 'B' represents high-risk patients.

Breiman et al. [25]. The tree shown here was created during a study that analyzed data of a medical study on heart attacks to identify high-risk patients. During this study, 19 variables were measured in the first 24 hours after admission to the medical center. The tree uses only three of the variables to effectively classify low-risk patients 'A' and high-risk patients 'B'.
To generate such a classification tree, for each set of data the one variable is

searched for, that separates the two classes best. In general, the process is repeated for each branch until the part of data on this path is separated or when splitting no longer improves the splitting result and with such, the prediction. However, other rules also may be applied.

In the case of regression also a binary tree is learned, but the tree does not classify the data in that sense but instead partitions the parameter space of the data such, that every partition, represented by a leaf, contains data that is similar or somehow close to each other, and the prediction error is minimized. Here, the nodes and adjoint labeled branches define the partitions of the data.



(a) Depicted are the partitions generated by a regression tree on a mostly flat, two-dimensional function. The colors indicate the underlying function.

(b) An exemplary regression tree that corresponds to the partitions shown in (a). The colors of the leaves indicate the class that the observed data is classified to.

Figure 2.3: The figures show an example of a regression process on a two-dimensional function. Figure (a) shows the partitions constructed by the regression tree, while Figure (b) depicts the related tree.

Figure 2.3 gives an example of how such a regression using a regression tree may work. Given a two-dimensional function that mainly defines two plane areas separated by a diagonal. Figure 2.3a shows the regarded function, indicated by the colors. The regression tree may partition the parameter space by splitting at $S_1 : x_2 = 0.35$ first. The predictions for the upper partition may have improved, while the prediction quality for the lower part may not have changed or got even worse. So in the next step, the related partition is split $S_2 : x_1 = 0.6$, and so on.

A few steps later the regression tree may have partitioned the parameter space as shown in Figure 2.3a. The regression tree that was constructed, and relates to the partitions created during this process looks like depicted in Figure 2.3b. Each internal node with adjoint labeled branches represents the Splits S_1 - S_5 specified during the partitioning of the data. The leaves specify the prediction that will be made for observations in the related partition P_i, $i \in \{1, 2, \cdots 6\}$.

In order to build a random forest, the available data is bootstrapped so that a set of distinct training data sets is generated. On each of the training data sets generated this way, a single tree is learned. However, this alone does not ensure a high variance in the trees generated. Assumed that in the available parameters one turns out to be a very strong predictor while several others only are moderately strong, the trees would always tend to use this very strong predictor. To counteract on this behavior, another tweak is added to the tree generating process, that enables the tree to choose from a randomly selected subset of predictors only, to generate the next junction in the tree. These subsets are generated randomly for each junction.
To gain a single prediction from the set of trees generated this way, the predictions of all trees are aggregated using majority voting, in the case of classification, and averaging, in the case of regression, respectively.

For the experiments carried out for this thesis two different kinds of tree-based models are used. These are for one a single tree model ('Tree'), that is based on the `rpart` package [27] and in most details follows Breiman et. al [25] quite closely.
And for another, a random forest model ('RandomForest'), which is based on the `randomForest` package [28], and implements the random forest algorithm by Breiman et al. [29] for regression.

2.1.3 Artificial Neural Networks

Artificial neural networks, also referred to as neural networks, describe information processing systems that are composed of units or nodes respectively, that communicate their information over directed connections between each other. Zell [30] and Anderson et al. [31] give an exhaustive overview of artificial neural networks and its history.
The general idea is derived from the cerebral structure of mammals, rigorously simplified. Different types of bipolar and multipolar cells of mammals are depicted in Figure 2.4. All of them feature the same components of cell body, axon,

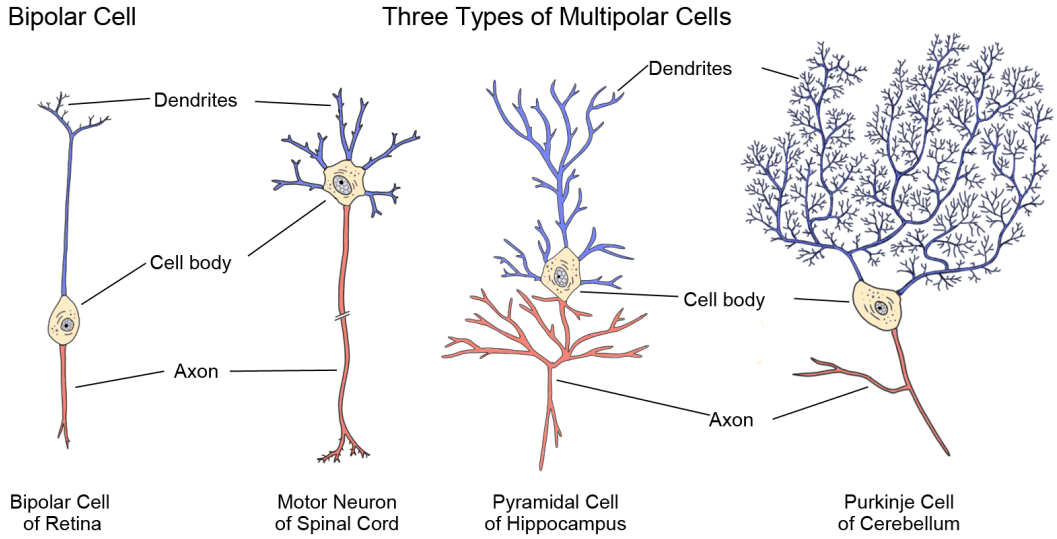


Figure 2.4: The illustration shows examples for different types of neurons (based on [32, 33])

and dendrites. The cell body ensures the energy supply of the cell and has the ability to process and transmit nerve impulses. With the dendrites, incoming impulses are received, with the axon the impulses are transmitted to other cells in the target area.

In order to build an artificial neural network, the neurons are vigorously abstracted. Figure 2.5 shows a schematic representation of two cells of a neural network. The cells i and j both have a network in_i and in_j of incoming connec-

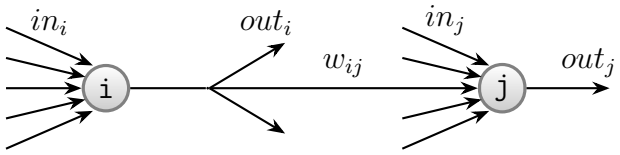


Figure 2.5: Depicted is a schematic representation of two simplified cells of a neural network (based on [30]). The incoming connections *in* represent the dendrites, the outgoing connections *out* the axons. The connection strength between the two cells is specified by a weight w.

tions representing the Dendrites, and a network of outgoing connections out_i and out_j representing the Axons. A weight w_{ij} specifies the transmission strength of the connection from the cell i to the cell j. With such cells, a network is built

that in many cases is arranged in layers with connections heading in the direction of the output neurons only, also referred to as feed-forward networks. Figure 2.6 shows an example of such a network.
This network has three layers of trainable connections, in this particular case every cell of one layer is connected to every cell of the next layer, and four layers of cells with the cells in the lower layer being the input cells and the cells in the upper layer the output cells. The inner layers are referred to as hidden layers.

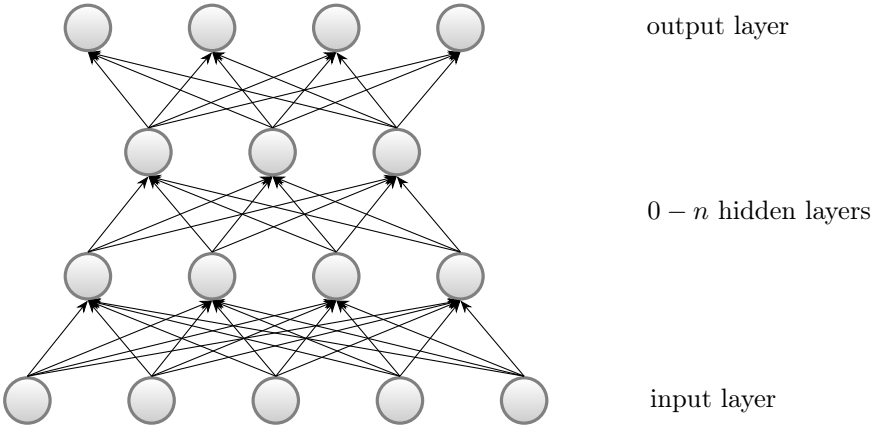


Figure 2.6: Depicted is a schematic feed-forward network with three connection layers and four neuron layers (based on [30]). The four cells in the bottom row represent the input layer; the four neurons in the topmost row the output layer. The inner layers are so-called hidden layers, and their number may strongly vary. Here, every cell of one layer is connected to every cell of the next layer.

For the experiments carried out in this thesis from the neural network type of models three different neural networks are used. These are a neural network ('neuralnet') based on the `neuralnet` package [34]. It is trained using resilient backpropagation without weight backtracking. The model is instantiated with two hidden neurons in each layer and a threshold of 0.001 that corresponds to the defaults that are set by the SPOT interface and thus slightly differs from the default provided by the `neuralnet` package.

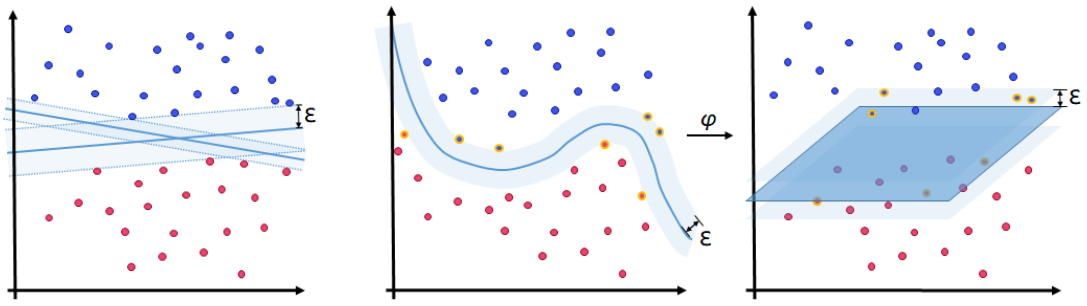
Additionally, an ensemble of twenty multi-layer perceptron neural network models ('MLP') based on the `monmlp` package [35] is used. Each is using two hidden-layers. The implementation applies early stopping together with bootstrap aggregation to control overfitting. The SPOT interface invokes the neural net using two hidden neurons in the first and one in the second layer.

And lastly, a censored quantile regression neural network model ('Qrnn') based on the `qrnn` package [36]. The model is instantiated with a single layer of hidden nodes.

2.1.4 Support Vector Machines for Regression

Support Vector Machines (SVM) implement concepts for regression as well as for classification. They were introduced by Boser et al. in 1992 [37]. A detailed introduction to SVMs can be found in [38], a comprehensive introduction to SVMs for Regression is given in [39, 40].

Here, data points are regarded as vectors in space. In the case of classification, the algorithm constructs a hyperplane that is separating the known vectors optimally. In this case, optimally means, that the distance ε of the input vectors that are closest to this separating hyperplane, also referred to as support vectors, is maximized. Figure 2.7a depicts such a situation where a set of vectors representing two different classes have to be separated. The vectors are directly linearly separable. Two possible separating lines are shown, the line with the broader separation space allows for the largest possible separation distance ε and therefore is the best choice for this example.



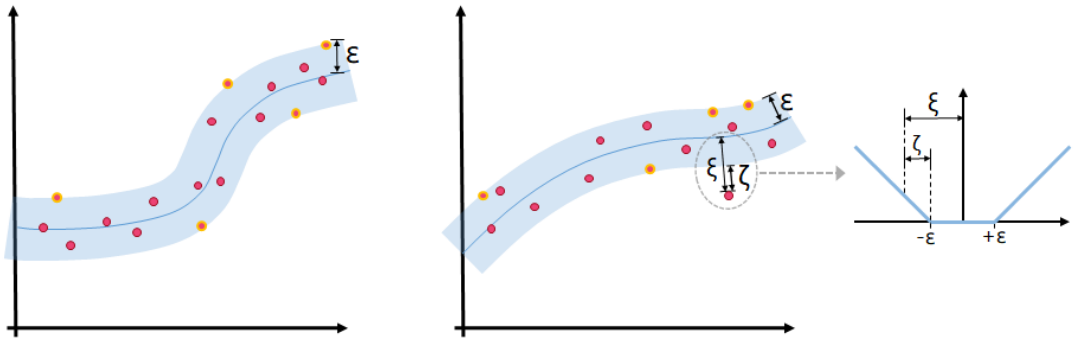
(a) If the data is linearly separable, the SVM fits a hyperplane through the data that separates the two classes with the largest possible distance ε to the nearest points.

(b) The data that has to be classified is not always linearly separable. In such cases, a kernel transformation is applied to the data that maps the data into a higher dimensional feature space, where the data is linearly separable. The points that are marked yellow are the support vectors that lie closest to the separating hyperplane.

Figure 2.7: Classification of two types of data points using a SVM.

However, the input vectors $\mathbf{x}$ are not always linearly separable. If this is the case, they are mapped into a high dimensional feature space Z. This is achieved by using some nonlinear mapping φ, that has been chosen beforehand. In this space, an optimal separating hyperplane is constructed. Figure 2.7b shows such a situation.

In the case of SVMs for regression, a function $f(x)$ is searched for, which approximates the known vectors with the smallest maximum deviation ε for all training vectors from the function $f(x)$ while also being as flat as possible. Here, too, the points that lie on the ε-deviation border are referred to as support vectors. Figure 2.8a illustrates such a function that approximates a set of training vectors with an accepted error of ε.



(a) SVMs for regression aim to find a function that approximates the training data, while minimizing the maximum deviation ε between training data and the function.

(b) Error handling using a soft margin penalty. Points with a deviation smaller than ε don't add to the cost of the function. Points with a larger deviation than ε are penalized linearly with its distance ζ to the ε-border.

Figure 2.8: Modelling data using Support Vector Machines for Regression

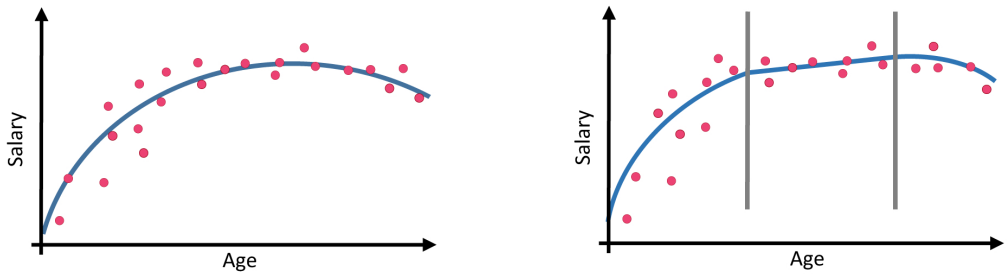
But this procedure is not strict, SVMs also allow for larger deviations in a few points using a soft margin penalty. For that purpose, the trade-off between the flatness of the function and number or cost of errors respectively that is accepted may be adjusted to allow for a few outliers. The outliers are then penalized with a linear cost function that assigns the cost ζ to an outlier that has a deviation of ξ with $\zeta = |\xi| - \varepsilon$. Figure 2.8b displays an example for such a soft margin penalty.

For the experiments carried out in this thesis a regression model based on the R

package `e1071` [41], that builds a support vector machine for regression is also used ('Esvm'). The model is based on the works of Chang et al. [42].

2.1.5 Multivariate Adaptive Regression Spline Models

Multivariate Adaptive Regression Splines (MARS) can be understood as a generalization of the recursive partitioning strategy used by regression trees [43].



(a) The figure shows an example for polynomial regression on salary data using a single polynomial

(b) Depicted is an example of piecewise polynomial regression on salary data using three independent polynomials. The vertical lines mark the so-called 'knots' that define the borders of the partitions.

Figure 2.9: The Figures show two approaches to approximating a function that describes best the relation between age and salary. The red points mark the observed points, and the blue line represents the approximated function.

Supposed that a function is to be learned that estimates the salary of a person with respect to the age of this person best. A simple method would be to fit a polynomial to the data using least squares regression. Figure 2.9a depicts an example of this approach. The red points mark the observed data concerning this relation.

In many cases, this may lead to good solutions, but there will also be cases where the data cannot be adequately approximated using a single polynomial. Then, it may be beneficial to divide the parameter range into disjoint subsets and fit an independent polynomial for each of the ranges. To obtain K ranges, $K-1$ so-called 'knots' have to be specified, that define these ranges. Furthermore, it

has to be ensured, that the resulting piecewise approximation function obtained is continuous or even smooth. For this purpose, additional constraints are introduced for each knot. Figure 2.9b shows an example of such a piecewise regression. The obtained function is continuous at the knots, but not smooth.

In opposition to polynomial regression, which must use polynomials of a higher degree to fit functions of higher complexity, regression splines allow keeping the degree of the polynomials fixed and induce higher flexibility through a higher number of knots. An exhaustive introduction to Multivariate Adaptive Regression Splines is given by Friedman et al. [43].

For the experiments carried out in this thesis also a regression model ('Earth') that uses the model building techniques of Friedman et al. for Multivariate Adaptive Regression Splines [44] and 'Fast MARS' [45] based on the R package `earth` [46] is added to the set of base models.

2.1.6 Kriging Based Models

The Kriging method (also known as Gaussian processes [47]) was first introduced by D.G. Krige [48] to improve mine valuation methods and with it the estimation of the concentration of gold in ore bodies. In 1963 Matheron extended the theory and formalized the technique [49]. Sacks et al. later applied the method to the approximation of computer experiments [50]. Forrester et al. give a very comprehensive introduction to optimization using the Kriging method [40]. The following remarks and equations are also based on these works.
The Kriging method uses a model of the form

$$\hat{y}(\mathbf{x}^*) = \hat{\mu} + \mathbf{k}^T\mathbf{K}^{-1}(y - \mathbf{1}\hat{\mu}). \tag{2.1}$$

as predictor. If μ is an a priori given constant, the Kriging variant is called 'simple' Kriging. If μ is estimated from the data, but then used as a constant, the variant is called 'ordinary' Kriging. Moreover, if μ is estimated from the data and depends on $\mathbf{x}$ then the variant is called 'universal' Kriging. In the following 'ordinary' Kriging is used since it has an excellent prediction quality and not too many parameters.
To derive this model it has to be started from a set of known data, $\mathbf{X} = \{\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^n\}^T$, with related known function values $\mathbf{y} = \{\mathbf{y}^1, \mathbf{y}^2, \cdots \mathbf{y}^n\}^T$ and it is searched for an expression to predict a value at an unknown point $\mathbf{x}$. Kriging views the known function values $\mathbf{y}$ as if they are realizations of a stochastic process, with errors ϵ spatially correlated. Between two points that are close

to one another the errors are considered positively correlated. With increasing distance between these points the correlation converges to zero. This correlation between the distance of the points and their errors are modeled using a correlation function, also referred to as kernel. An example for such a kernel is given by Equation (2.2).

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\sum_{i=1}^{m} \theta_i |x_i - x_i'|^{p_i}\right) \tag{2.2}$$

Here, two points x and x' are considered, with $x_i \in \mathbb{R}$ denoting the i-th element of the vector x. This kernel meets the requirements to yield a function value of one if $\mathbf{x} = \mathbf{x}'$, converges to zero for larger distances and is positive semi-definite. Some examples of different kernel functions are shown in Figure 2.10. The kernel function is utilized to create a correlation matrix $\mathbf{K}$ containing all pairwise correlations between errors of all known function values $\mathbf{y}$ at locations $\mathbf{X}$.

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}^1, \mathbf{x}^1) & \cdots & k(\mathbf{x}^1, \mathbf{x}^n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^n, \mathbf{x}^1) & \cdots & k(\mathbf{x}^n, \mathbf{x}^n) \end{pmatrix}$$

These correlations depend on the absolute distance between the known points $|\mathbf{x} - \mathbf{x}'|$ and the parameters θ_i and p_i. These parameters, θ and p, are in general estimated using Maximum Likelihood Estimation [40, 51], such that for the given model, the known data points have the largest likelihood. The likelihood function of the Kriging model is based on the probability density function of a multivariate normal distribution,

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}|\mathbf{C}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{y} - \mathbf{1}\mu)^T \mathbf{C}^{-1} (\mathbf{y} - \mathbf{1}\mu)\right)$$

with $\mathbf{1}$ being the identity vector and $\mathbf{C}$ the stationary covariance matrix. The covariance matrix $\mathbf{C}$ is related to the correlation matrix $\mathbf{K}$ by $\mathbf{C} = \sigma^2 \mathbf{K}$. With this the Kriging likelihood function can be expressed as

$$f(\epsilon(\mathbf{X})|\mu, \sigma, \theta, p) = \frac{1}{(2\pi\sigma^2)^{n/2}|\mathbf{K}|^{1/2}} \exp\left(-\frac{(\mathbf{y} - \mathbf{1}\mu)^T \mathbf{K}^{-1} (\mathbf{y} - \mathbf{1}\mu)}{2\sigma^2}\right) \tag{2.3}$$

This equation has to be simplified by taking the natural logarithm. The partial derivatives of this function then have to be set to zero in order to obtain the Maximum Likelihood Estimates for μ and σ^2:

$$\hat{\mu} = \frac{\mathbf{1}^T \mathbf{K}^{-1} \mathbf{y}}{\mathbf{1}^T \mathbf{K}^{-1} \mathbf{1}}$$

and

$$\hat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\hat{\mu})^T \mathbf{K}^{-1} (\mathbf{y} - \mathbf{1}\hat{\mu})}{n}.$$

These terms for $\hat{\mu}$ and $\hat{\sigma}^2$ can now be substituted back into the logarithmized form of Equation 2.3. Removing constant terms yields the so-called concentrated ln-likelihood function:

$$con(ln(L)) = -\frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2}\ln(|\mathbf{K}|). \tag{2.4}$$

Here, the parameter θ and p are still unknown. In order to find values for these parameters that maximize the function, numerical optimization has to be applied, since the function cannot be differentiated. However, as the function is quick to compute, as long as the search space does not get too large, the function can be searched directly, so that a classical global optimization method can be applied.

Introductory it was said that it is searched for an expression to predict a value $\hat{y}$ at an unknown point $\mathbf{x}$. The main idea to do so is to augment the known data $\mathbf{y}$ with the new prediction $\hat{y}$, which yields the vector $\mathbf{y}_{aug} = \mathbf{y}^T, \hat{y}^T$. Treating $\hat{y}$ as a model parameter, the likelihood function is to be maximized with respect to $\hat{y}$, given the already known correlation parameters.
Defining the vector of correlations between the set of known data $\mathbf{X}$ and the new data point x^* as $\mathbf{k} = (k(x^1, x^*), \cdots, k(x^n, x^*))^T$, this vector can be substituted into the prediction function (2.1) together with the already known model parameters to make a prediction.

For a next iteration, the augmented correlation matrix can be defined as $\mathbf{K}_{aug} = \begin{pmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & 1 \end{pmatrix}$. Together with the already known model parameters, $\mathbf{K}_{aug}$ and $\mathbf{y}_{aug}$ are substituted into the likelihood function (2.3). Maximizing the resulting term with respect to $\hat{y}$ yields the predictor (2.1).

In the experiments carried out in this thesis, three Kriging models with different correlation functions are used. These are Kriging with exponential correlation function ('correxp'), gaussian correlation function ('corrgauss'), and spline correlation function ('corrspline'). Figure 2.10 depicts the kernels used. The Kriging implementation is part of the `SPOT` package and follows the implementation of Lophaven et al. as described in [52].

Following the definitions from Lophaven et al., the correlation models can be described as follows. We consider stationary correlations of the form $\mathcal{R}(\theta, x, x') = \prod_{j=1}^{n} \mathcal{R}(\theta_j, x_j - x'_j)$. The first model uses the *exponential* kernel $\mathcal{R}(\theta, x_j, x'_j) =$

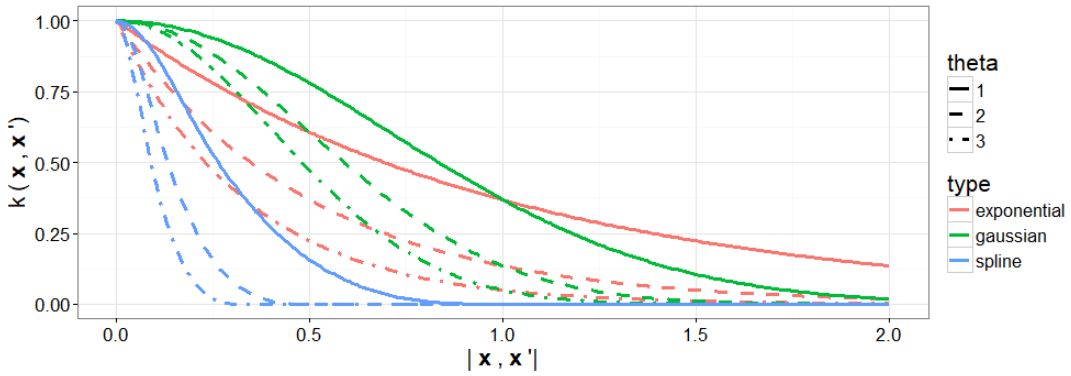


Figure 2.10: The figure shows an example of different correlation functions. The functions 'exponential' and 'gaussian' implement the correlation function specified in Equation 2.2 with $p = 1$ for 'exponential' and $p = 2$ for 'gaussian'. The 'spline' function follows the definitions from [52] and only takes the θ-parameter. The function values then specify the assumed correlation of two known points, based on their distance.

$\exp(-\theta_j|x_j - x'_j|)$ the second model uses a *gaussian* kernel $\mathcal{R}(\theta, x_j, x'_j) = \exp(-\theta_j|x_j - x'_j|^2)$, whereas the third model is based on the *spline correlation* function $\mathcal{R}(\theta, x_j, x'_j) = \zeta(\theta_j|x_j - x'_j|)$ with

$$\zeta(\epsilon_j) = \begin{cases} 1 - 15\epsilon_j^2 + 30\epsilon_j^3 & \text{for} \quad 0 \leq \epsilon_j \leq 0.2 \\ 1.25(1 - \epsilon_j)^3 & \text{for} \quad 0.2 < \epsilon_j < 1 \\ 0 & \text{for} \quad \epsilon_j \geq 1. \end{cases}$$

Here, ϵ and θ are hyperparameters estimated by likelihood maximization.

Additionally, a treed Gaussian process model with jumps to the limiting linear model ('tgp') that is based on the `tgp` package [53] is used in the experiments.

Finally, a simple ensemble model ('Rfmlegp') is also used for the experiments. The ensemble internally builds a random forest model using the `randomForest` package [28] and a Gaussian process model using the `mlegp` package [54]. The mean of these models' predictions is returned as the ensemble prediction.

2.2 Surrogate Model-Based Optimization

In most real-world optimization problems, the number of function evaluations that can be carried out is massively limited by time or cost. At the same time, not seldom the objective function is expensive to evaluate. Direct search methods, in general, require more function evaluations than the number that can actually be spent, which makes such real-world problems a special challenge for global optimization.

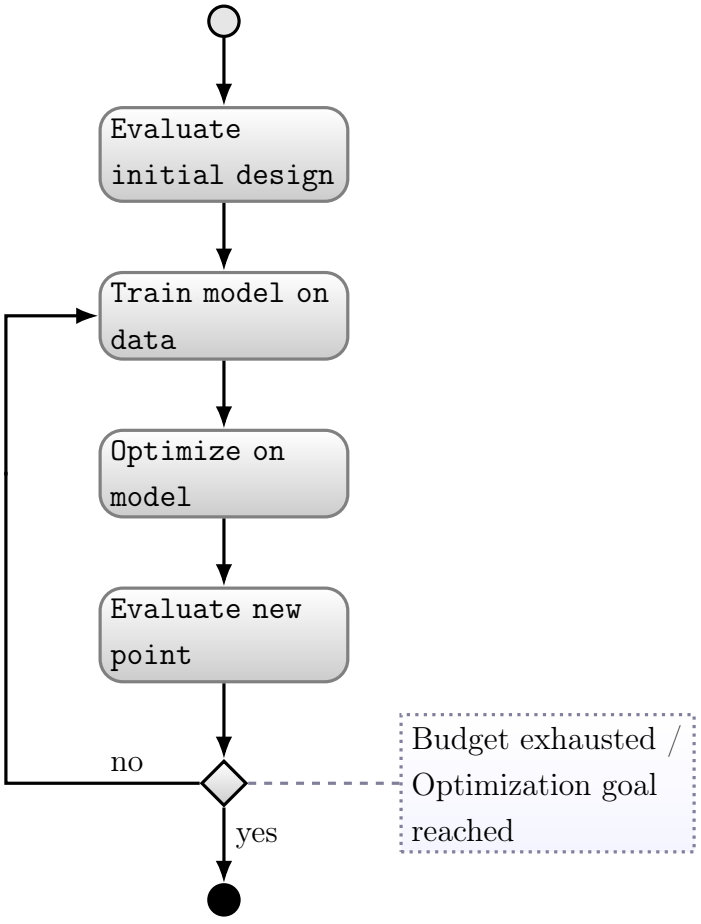


Figure 2.11: Shown is a schematic representation of a surrogate model based optimization process. Initially, a first set of data points is evaluated on the function; in general, these points are chosen using a design of experiment. Then, a surrogate model is fitted to the data. On the fitted model, a separate optimization process is started, and the best points found are evaluated on the function. The process ends when the predefined stopping criterium is reached.

Jones et al. [17] and others [14, 55, 56] addressed this challenge by using surrogate models as a fast and cheap approximation of the real objective function. Instead of performing a direct search on the objective function, in every step a model is trained to the available data, and an optimization process is carried out on this model. The best n solutions found during this intermediate optimization step are then evaluated on the real objective function. Figure 2.11 depicts the general procedure of such an SBO process. Evolutionary surrogate model based optimization was introduced by Emmerich et al. [57].

Like in any optimization process, if initially no information about the function is available, some data points have to be evaluated to gain some base knowledge D about the function. Usually, a design of experiment (DOE) is used to obtain this knowledge.

Then, the main optimization loop is started by fitting the model to the observed data D. On the fitted model that is now approximating the objective function a separate optimization step is carried out.

From the set of points that were evaluated on the model during this optimization, a choice of n points is selected to be evaluated on the objective function. This choice is not limited to merely choosing the point that has the best-predicted value on the model, but can also be based on other criteria like the expected improvement[1] if the model provides confidence intervals for its predictions. The main optimization loop is stopped if the budget of available function evaluations is exhausted or a predefined optimization goal is reached.

Software packages like SPOT provide a complete optimization framework featuring heaps of statistical tweaks to optimize this process even more, as well as parameters for additional fine-tuning [59].

Nonetheless, for the experiments carried out in this work a very basic SPO framework is developed to allow for complete control over the behavior of the sequential optimization process and to gain more insight and interpretable results. Algorithm 1 depicts the main steps of this framework which, up to some minor details, corresponds to standard SPO processes.

In a first step the initial design is evaluated (cf. Algorithm 1 line 2). To allow for reproducible results and equal experimental preconditions across several experimental setups, the framework takes predefined experimental designs so that

[1] Expected improvement is a criterion that helps to balance between exploitation and exploration. It was introduced by Mockus et al. in 1978 [58]. Jones et al. [17] give a comprehensive introduction to the topic.

Algorithm 1: The SPO Framework that was used for the experiments carried out in this thesis

Data: objective function $f : \mathbb{R}^d \to \mathbb{R}$
initial design $\{x_1, \cdots, x_n\}$, $x \in \mathbb{R}^d$,
surrogate model m
sequential step size n_{eval}

Result: Resultset containing all evaluated points

1: **begin**
2: $D \leftarrow$ Evaluate initial design on objective function f;
3: **while** *function evaluations available* **do**
4: $m^* \leftarrow$ Fit model m to known datapoints D ;
5: Evaluate sequential design on model m^*;
6: Choose $\lceil \frac{1}{2} n_{\text{eval}} \rceil$ points x^* according to exploitation;
7: Choose $\lfloor \frac{1}{2} n_{\text{eval}} \rfloor$ points x^{**} according to exploration;
8: Evaluate chosen points x^* and x^{**} on objective function f;
9: Add new information to known data D ;

all experiments use the same initial configuration (cf. Algorithm 1, data input). The resulting dataset $D = \{(x_1, y(x_1)), \ldots, (x_n, y(x_n))\}$ builds the foundation for the subsequent optimization process.

The main optimization steps are carried out in a loop that ends when the number of allowed function evaluations has been reached (cf. Algorithm 1 line 3).

First, the model M is fitted to the observed data D (cf. Algorithm 1 line 4). On the fitted model M^* a sequential design is evaluated (cf. Algorithm 1 line 5). Based on the resulting set of model predictions, points are selected for evaluation on the objective function. For this choice, the available budget of function evaluations per sequential step n_{eval} is shared equally on points x^* that correspond to the criteria of exploitation[1] and on points x^{**} that correspond to the criteria of exploration[2].

[1] Exploitation refers to the strategy of searching a restricted search space in the area of the best-known solution in order to improve this solution. This can also be referred to as local search.

[2] Exploration refers to a strategy of searching the entire region of interest in the search space in order to find new promising solutions. This strategy helps to diversify the search and to

The selection of these points is carried out as follows:
For exploitation from the predictions of the model on the large sequential design, the k best performing points are chosen and added to the candidate set C^*. A typical value of k is 20. Since these points are restricted to points from the design, from each of these points a local search is initiated on the model. The points that are obtained with this search are also added to the candidate set C^*. If these points violate constraints of the search space, they are repaired by setting those violated parameters to the allowed limit. From this set of candidates C^* the $\lceil \frac{1}{2} n_{\text{eval}} \rceil$ points x^* that perform best on the model are chosen for evaluation on the objective function (cf. Algorithm 1 line 6).
For exploration, those k points from the sequential design are chosen and added to the candidate set C^{**}, that have the largest distance to their nearest neighbors. From these candidates those $\lfloor \frac{1}{2} n_{\text{eval}} \rfloor$ points x^{**} that perform best on the model are chosen to be evaluated on the objective function (cf. Algorithm 1 line 7).

The points x^* and x^{**} chosen before are evaluated on the objective function. And the new information gained in this step is added to the dataset D (cf. Algorithm 1 line 8-9).

After finishing the main loop, the dataset D is returned as the result.

The default settings for the experiments performed in this study were chosen as follows: for the sequential design, a size of 200 points and a sequential step size of $fe_{step} = 2$ was chosen.

This optimization is intentionally kept simple to allow for better insight into the performance of the models used for optimization.

2.3 Objective Functions

The experiments carried out for this work are run on a set of objective functions. These objective functions can be roughly divided into three groups.

1. Generic objective functions generated to fit the requirements,
2. standard optimization test problems,
3. and test functions based on physical models.

prevent it from getting trapped in a local optimum. This can also be referred to as global search.

Since most of the black-box real-world problems considered to be difficult are multimodal, the focus for this work is also on multimodal function approximation (cf. [60, 61, 62]). In the following, these functions are introduced in more detail.

2.3.1 Gaussian Landscape Generator

To allow for the flexible generation of test functions that meet the requirements in certain features the *Max-Set of Gaussian Landscape Generator* (GLG) is applied. It computes the upper envelope of m weighted Gaussian process realizations and can be used to generate continuous, bound-constrained optimization problems [63].

Thus, a GLG objective function is defined as

$$G(x) = \max_{i \in 1,2,\ldots,m} (w_i g(x)),$$

where $g : \mathbb{R}^n \to \mathbb{R}$ denotes an n-dimensional Gaussian function

$$g(x) = \left(\frac{\exp\left(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^T\right)}{(2\pi)^{n/2}|\Sigma|^{1/2}} \right)^{1/n},$$

μ is an n-dimensional vector of means, and Σ is an $(n \times n)$ covariance matrix. Implementation details are presented in [64]. For the generation of the objective functions the spotGlgCreate method of the SPOT package is used.

The options used for our experiments are shown in Table 2.1. With the parameter d the dimension of the objective function is specified. The lower and upper bounds (l and u, respectively) specify the region where the peaks are generated. The value max specifies the function value of the global optimum, while the maximum function value of all other peaks is limited by t, the ratio between the global and the local optima.

2.3.2 Optimization Test Problems

Two classical mathematical test problems are chosen. Both, the Ackley function [65] as well as the Rosenbrock function [66] are widely used for testing optimization algorithms.

Table 2.1: Gaussian landscape generator options

Param.	Description	Value
d	Dimension	$1-8$
m	Number of peaks	$10-320$
l	Lower bounds of the area, where peaks are generated	$\{0_1,\ldots,0_d\}$
u	Upper bounds of the area, where peaks are generated	$\{5_1,\ldots,5_d\}$
max	Max function value	100
t	Ratio between global and local optima	0.8

The Ackley function
was proposed by David Ackley in 1987 [65]. Later, in 1993 the function was generalized by Bäck and Schwefel [67]. In its generalized form it is defined by

$$f(x) = -a\,\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

with $a = 20$, $b = 0.2$ and $c = 2\pi$.

It is a non-convex function featuring many local optima. In its two-dimensional form the function is almost flat in its outer regions with the global minimum $(f(x^*) = 0, \text{at } x^* = (0,\ldots,0))$ being a large peak at the center. With its highly multi-modal characteristics, it poses a risk to optimization algorithms to get stuck in local minima.

As region of interest we regarded the hypercube $x_i \in [-32.768, 32.768],\ \forall x \in [1, d]$.

The Rosenbrock function
is also known as Valley- or Banana function and has been introduced by Howard H. Rosenbrock in 1960 [66]. It is defined by

$$f(x) = \sum_{i=1}^{d-1}\left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

The function is non-convex, but in contrast to the Ackley function, the Rosenbrock function is unimodal. Its global minimum $(f(x^*) = 0)$, lies at $x^* = (1,\ldots,1)$. In its two dimensional form it is located in a narrow, parabolic valley that is following the parabola $x_2 = x_1^2$. Though the valley is easy to find, convergence to the minimum is difficult.

As region of interest we regarded the hypercube $x_i \in [-2.048, 2.048],\ \forall x \in [1, d]$.

2.3.3 Physical Functions

In order to gain a set of test functions that also contains functions with relation to real-world optimization problems, four test functions that model a physical model are added to the set. As a source of these test functions, the 'Virtual Library of Simulation Experiments' [68] is used. The website provides a collection of well-structured test problems. From the choice of emulation/prediction test problems based on physical models, all functions, that are not stochastic and given in the form of a function instead of a dataset, are added to the system. Under this premise we get a set of four real-world objective functions as follows.

The OTL Circuit Function models an output transformerless push-pull circuit [69]. The response variable of the function is V_m, the midpoint voltage, that is affected by six parameters. The parameters, with its units and ranges, are specified in Table 2.2.

Parameter	Description	Value range
R_{b1}	resistance b1 (K-Ohms)	$[50, 150]$
R_{b2}	resistance b2 (K-Ohms)	$[25, 70]$
R_f	resistance f (K-Ohms)	$[0.5, 3]$
R_{c1}	resistance c1 (K-Ohms)	$[1.2, 2.5]$
R_{c2}	resistance c2 (K-Ohms)	$[0.25, 1.2]$
β	current gain (Amperes)	$[50, 300]$

Table 2.2: Search Parameters of the otl-circuit function

The midpoint voltage V_m can be derived from the parameters as follows:

$$V_m(\mathbf{x}) = \frac{(V_{b1} + 0.74)\beta(R_{c2} + 9)}{\beta(R_{c2} + 9) + R_f} + \frac{11.35R_f}{\beta(R_{c2} + 9) + R_f} + \frac{0.74R_f\beta(R_{c2} + 9)}{(\beta(R_{c2} + 9) + R_f)R_{c1}}$$

where

$$V_{b1} = \frac{12R_{b2}}{R_{b1} + R_{b2}} \qquad \text{and} \qquad \mathbf{x} = (R_{b1}, R_{b2}, R_f, R_{c1}, R_{c2}, \beta)$$

The Piston Function is a simulator, that models the movement of a piston within a cylinder. The piston consists of a linear rod that is connected to a disk. By this connection, the linear movement of the rod is transformed into a circular motion. The faster the piston moves within the cylinder the faster rotates also the disk. The performance of the piston is measured by its cycle time, the time it takes to perform one rotation of the disk, in seconds.

The performance of the piston is affected by a set of parameters, given in Table 2.3. These parameters affect the cycle time T_c via a chain of nonlinear equations [69]:

$$T_c(\mathbf{x}) = 2\pi\sqrt{\frac{M}{k + S^2\frac{P_0V_0}{T_0}\frac{T_0}{V^2}}}$$

where

$$V = \frac{S}{2k}\left(\sqrt{A^2 + 4k\frac{P_0V_0}{T_0}T_a} - A\right) \ , \qquad A = P_0S + 19.62M - \frac{kV_0}{S}$$

and

$$\mathbf{x} = (M, S, V_0, k, P_0, T_a, T_0).$$

The function was developed by Kenett and Zacks in 1998 [70].

Parameter	Description	Value range
M	piston weight (kg)	$[30, 60]$,
S	piston surface area (m^2)	$[0.005, 0.020]$
V_0	initial gas volume (m^3)	$[0.002, 0.010]$
k	spring coefficient (N/m)	$[1000, 5000]$
P_0	atmospheric pressure (N/m^2)	$[90000, 110000]$
T_a	ambient temperature (K)	$[290, 296]$
T_0	filling gas temperature (K)	$[340, 360]$

Table 2.3: Search Parameters of the Piston Function

The Robot Arm Function is commonly used in neural network literature. It models a four-segment robot arm with the shoulder of the arm fixed at the origin in the (u, v)-plane [71]. Each segment of the arm has a specific length L_i and an angle θ_i, $i = 1, \ldots, 4$. The angle of the first segment, with respect to the horizontal coordinate axis of the plane, is given by θ_1. The angles θ_2, θ_3, θ_4

describe the rotation of the corresponding arm segment in relation to the previous arm segment.

The response variable of the function D, models the distance of the end of the robot arm to the origin, on the (u, v)-plane. This distance can be directly derived from the eight parameters by:

$$D(\mathbf{x}) = \sqrt{u^2 + v^2}$$

where

$$u = \sum_{i=1}^{4} L_i \cos\left(\sum_{j=1}^{i} \theta_j\right), \quad v = \sum_{i=1}^{4} L_i \sin\left(\sum_{j=1}^{i} \theta_j\right) \text{ and } \quad \mathbf{x} = (L_1, L_2, L_3, L_4, \theta_1, \theta_2, \theta_3, \theta_4).$$

The input parameters, with its ranges, are specified in Table 2.4

Parameter	Description	Value range
$L_i,\ i = 1, \ldots, 4$	length of the i-th arm segment	$[0, 1]$
$\theta_i,\ i = 1, \ldots, 4$	angle of the i-th arm segment	$[0, 2\pi]$

Table 2.4: Search Parameters of the Robot Arm Function

The Wing Weight Function models the weight W of the wing of a light aircraft [40]. The analytical expression is adapted from the work of Raymer (2006) on conceptual aircraft design [72]:

$$W = 0.036\, S_W^{0.758}\, W_{fw}^{0.0035} \left(\frac{A}{\cos^2\Lambda}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100tc}{\cos\Lambda}\right)^{-0.3} (N_Z W_{dg})^{0.49} + S_W W_P$$

The ten parameters, that affect the weight W are given in Table 2.5, along with its baseline and range.

While the baseline values roughly represent the values of a Cessna C172 Skyhawk aircraft, the given ranges where specified by Forrester et al. [40].

Parameter	Description	Baseline	Value range
S_w	wing area (ft^2)	174	$[150, 200]$
W_{fw}	weight of fuel in the wing (lb)	252	$[220, 300]$
A	aspect ratio	7.52	$[6, 10]$
Λ	quarter-chord sweep (degrees)	0	$[-10, 10]$
q	dynamic pressure at cruise (lb/ft^2)	34	$[16, 45]$
λ	taper ratio	0.672	$[0.5, 1]$
t_c	aerofoil thickness to chord ratio	0.12	$[0.08, 0.18]$
N_z	ultimate load factor	3.8	$[2.5, 6]$
W_{dg}	flight design gross weight (lb)	2000	$[1700, 2500]$
W_p	paint weight (lb/ft^2)	0.064	$[0.025, 0.08]$

Table 2.5: Search Parameters of the Wing Weight Function

Some passages in this chapter are based on descriptions that were already published in [73, 74, 75].
The composition of base models and the set of objective functions used for the experiments carried out in this thesis were already described in [75]. The specification of the Kriging kernels was already given in [73]. Occasionally, text elements have been adopted verbatim from these publications. However, the text was significantly extended, and adapted to fit the notation of this thesis.
The SPO Framework introduced in this Chapter has already been described in [74, 75]. Some of these descriptions, as well as Algorithm 1, are adopted verbatim or with minor changes to adapt them to the notation and structure of this thesis.