CHALMERS | GÖTEBORG UNIVERSITY

MASTER'S THESIS

A method for simulation based optimization using radial basis functions

Johan Rudholm
Adam Wojciechowski

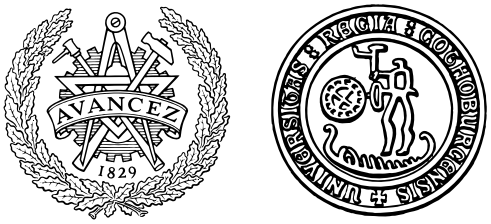
Department of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg Sweden 2007

Thesis for the Degree of Master of Science

A method for simulation based optimization using radial basis functions

Johan Rudholm
Adam Wojciechowski

CHALMERS | GÖTEBORG UNIVERSITY



Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, August 2007

Matematiskt centrum
Göteborg 2007

Abstract

The background of this thesis was the GMMC *Combustion Engine Optimization* project which started in the spring of 2006. Proposed is an algorithm for global optimization of noisy and expensive black box functions using response surfaces based on radial basis functions (RBFs). A method for RBF-based approximation is introduced in order to handle noise. New points are selected to minimize the total model uncertainty weighted against the function value. The algorithm is extended to multiple objective functions by instead weighting against the distance to the Pareto front. Numerical results on analytical test functions show promise in comparison to other (commercial) algorithms, as well as results from simulations. The algorithm also handles noisy multiple objectives, something which is lacking in previous research.

Sammanfattning

Bakgrunden till detta examensarbete är GMMC-projektet *Combustion Engine Optimization* som inleddes under våren 2006. En global optimeringsalgoritm som använder sig av responsytor baserade på radiella basfunktioner (RBF) för brusiga och dyra black-box funktioner presenteras. Hantering av brus sker genom att introducera en metod för RBF-baserad approximation. Nya punkter väljs genom att minimera den totala osäkerheten i modellen viktat mot funktionsvärdet. Algoritmen utökas till flermål genom att istället vikta mot avståndet till Paretofronten. Numeriska resultat från analytiska testfunktioner, såväl som resultat från simuleringar, är lovande i jämförelse med (kommersiella) algoritmer. Algoritmen hanterar även brusiga flermål, något som saknas i tidigare forskning.

Acknowledgments

This work was performed within the framework of the Gothenburg Mathematical Modelling Centre (GMMC), which is a strategic research centre in mathematical modelling funded by the Swedish Foundation for Strategic Research (SSF). All members in GMMC *Combustion Engine Optimization* should be mentioned, Mattias Ljungqvist at Volvo Car Corporation, Johan Wallesten and Dimitri Lortet at Volvo Powertrain. The work was done at the Fraunhofer-Chalmers Research Centre for Industrial Mathematics, FCC.

To test and develop the algorithm further, the company CD-adapco has granted FCC a research license for STARCD for the continuation of the GMMC *Combustion Engine Optimization* project.

We would also like to thank all the people at Fraunhofer-Chalmers Research Centre for Industrial Mathematics, in particular: Stefan Jakobsson (our supervisor, who continuously supported us during the course of the thesis, and from whom the main ideas behind the thesis originated), Michael Patriksson (our examiner), Ann-Brith Strömberg and Erik Höök.

At Volvo Powertrain we would like to thank Mr Dimitri Lortet for his help with the engine simulations.

Contents

1 Introduction

The background of this thesis was the GMMC *Combustion Engine Optimization* project which started in the spring of 2006. Participants in this project are Fraunhofer-Chalmers Research Centre for Industrial Mathematics, FCC, Volvo Car Corporation and Volvo Powertrain. The goal of this project was to develop best practice and a software demonstrator for combustion engine optimization. At both Volvo Cars and Volvo Powertrain there is a lot of experience of this type of complicated engine simulations, where the physical and chemical processes inside a diesel engine is modelled and computed using the software STARCD. Also some attempts to optimize the design parameters of the engine with respect to fuel consumption and emissions of soot and nitrogen oxides were done there. However, these computer simulations are time consuming and together with the multiobjective character of the problem it is necessary to have efficient multiobjective optimization algorithms at hand. Therefore, to meet the goals of the project, the development of a new optimization algorithm based on radial basis functions started at FCC by Stefan Jakobsson in 2006. The basic idea was to focus on the approximation of the objective function and during the course of optimization continuously improve on the approximation; especially in those areas where the approximate objective function has low function values, that is, close to optimum. A preliminary implementation was made in MATLAB and tested on a few two-dimensional problems. Ideas of how to handle multiobjective optimization and noisy functions were also developed.

The purpose of this thesis was to continue the development of the algorithm and to incorporate into it handling of noisy functions as well as multiobjective optimization. A new implementation of the algorithm has been made, which handles up to six design parameters and two objective functions. The main ideas behind this approach, namely the "quality function" and the radial basis functions-based approximation were allready present in the original implementation, developed by Stefan Jakobsson.

In May 2007 we were offered a training period at Volvo Powertrain to test the algorithm on their simulation cases. The simulation is specified by numerous parameters and they all have an impact on the simulated process. Some of the parameters have been assigned different sets of template values to correspond to different load cases. A load case is a certain mode of driving, for instance driving a fully loaded trailer at cruising speed on a flat highway, or driving through a city with no trailer at all. Obviously, these modes give rise to different fuel consumptions and also different degrees of emissions. The decision variables available for controlling the fuel consumption at a given load case are the geometric parameters describing for instance the appearance of the piston head.

To minimize the fuel consumption while at the same time keeping the emissions at an acceptable level is an optimization problem, viewing the output of the simulation as the objective function. Certain aspects of the problem does however set it aside from optimization problems usually considered. There are for instance no analytic derivatives available (usually known as a black box function); a simulation takes 42 hours to complete and the resulting output variables may contain different types of disturbances (numerical noise, for instance). The geometrical parameters found to be optimal in one load case may not be optimal in another case, making the problem a multiobjective optimization problem.

To start at the beginning: research and development departments usually model a new product as a computer simulation before constructing the initial (physical) prototype. In this way resources are conserved since once a feasible design has been created in a simulation environment, design parameters may be altered and new components added and evaluated at limited expense. This is compared to creating and evaluating a new physical prototype every time a new idea has been hatched or some prerequisite has changed.

When creating a product there are often some design goals to be met, may it be a legislative constraint or perhaps the most outstanding feature of the product itself. For instance, when designing an antenna one of the goals may be to maximize the bandwidth at a certain frequency; in the case of a car engine, a goal may be to minimize the fuel consumption at a certain load. This is where optimization comes in. To optimize something means finding a set of input parameters minimizing a certain output parameter. For consistency throughout the thesis the optimum is supposed to be a

minimum, as opposed to a maximum. The terms (global) minimum and (global) optimum are used interchangeably in the text. An *optimum value* is a value which cannot be improved upon. For a more detailed description of the basics of optimization, see Section 2.1.

A common way in the industry to find an optimal set of design variables for a given simulation model is to generate a set of sample points (set of design variables) in the variable space, run the simulation for the set and create a *surrogate model* of the resulting values. The surrogate model may for instance be an interpolation based on Radial Basis Functions (RBFs). This approach is usually called design of experiments (DoE), and is followed by letting an optimization algorithm find the optimum of the surrogate model. The benefit of using a surrogate model is that the evaluation of the surrogate is considerably cheaper. The drawback is that it, to various degrees, may be inaccurate depending on the appearance of the underlying function (simulation) being modeled, the number of points used in creating the surrogate and of course the choice of the surrogate model itself.

Improving on the accuracy of the surrogate model is done by concentrating on relevant parts of the variable space, namely those parts believed to be close to an optimum. In this way no expensive evaluations are "wasted" on parts of less importance, since the ultimate goal is to find an optimum and not to create as accurate a surrogate model as possible. Being able to concentrate on relevant parts does of course demand information on where these parts are located; such information is obtained from the surrogate model itself, since it contains all available information up to this point. The chosen points are then evaluated and used to sequentially update the surrogate model. This kind of approach starts with a small initial number of points which then increases until an acceptable accuracy is obtained, as opposed to selecting all points beforehand. The proposed algorithm follows the scheme just outlined.

This thesis is composed by the following parts. The remainder of this section gives a brief introduction to the common features of the problems considered, an overview of previous research and a rationale behind the proposed algorithm. Section 2 gives the reader a mathematics background to the discussion that follows. Areas including ordinary optimization, multiobjective optimization and scattered data interpolation are covered. The section also contains new theory not mentioned in the existing literature, for instance a novel way of bounding the uncertainty of a radial basis functions based interpolation. Section 3 describes the inner workings of the algorithm proposed, how the interpolation is constructed, how new points for evaluation are chosen and how the algorithm is adapted to multiobjective optimization. Section 4 describes the implementation of the algorithm, simplifications made, numerical "fixes" and so forth. Section 5 reports on numerical results of some standard text-book reference functions, as well as on simulations. Section 6 contains a discussion, suggests improvements to the proposed algorithm and areas of future research. Finally, Section 7 contains the conclusions.

1.1 Simulation-based optimization

The common features of the simulation-based problems considered in the thesis include:

1. function values depend continuously on input parameters
2. expensive to evaluate
3. no analytical derivatives (black box function)
4. subject to box and non-linear constraints in the parameter domain
5. function values may be subject to noise/perturbations
6. a parameter space of low dimension

Items 1–4 imply that the algorithm should be global, keep function evaluations to a minimum, be gradient-free[1] and handle non-linear constraints in the parameter domain. In addition to these features, the parameter domain of the problem must be closed and bounded.

[1] Also with regard to finite-difference approximations of the gradient, since such approximations costs function evaluations.

Item 5 concerns two different types of noise that may arise out of some types of simulations, providing further motivation for the algorithm to be gradient-free. The first type is ordinary numerical noise which may for instance be a product of rounding errors in the simulation process. This type of noise is usually modeled as being independent and identical in distribution (IID). The second type of noise arises from certain types of simulations, particularly Finite Element Method (FEM) and Computational Fluid Dynamics (CFD) based simulations where geometrical structures are described by unions of simpler objects (such as triangles and tetrahedra), forming a mesh. What may happen when some geometrical input parameter is changed, ever so little, is that some of the simple objects may disappear, or new simple objects appear, to accommodate the change in geometry. In a word, the mesh does not depend continuously on the geometry parameters. This gives rise to a disproportionate change in the output which may be regarded as noise, since it is not easily predictable. This noise on the other hand is most likely correlated. Trying to optimize an objective function based on such a simulation without considering the noise can lead to a "false" optimum solution, that may not be optimal in physical reality.

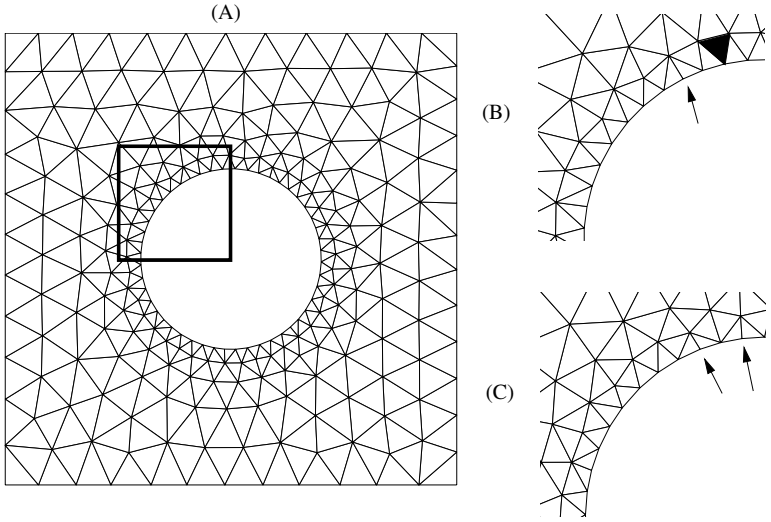


Figure 1: Sample mesh describing a simple geometrical figure. Figure (B) represents an enlargement of the black square in (A). Figure (C) represents the same square, after the radius of the center circle has been increased by 0.25%. Note that the triangles in the two smaller figures may look alike at first glance, but really do differ. For instance, the filled triangle in (B) will disappear when the radius changes and the "tent"-like structure indicated by an arrow will be duplicated in (C).

In Figure 1 a square with a circular hole in the center has been constructed using triangles (a so called mesh)[2]. The radius of the circle has then been been increased by 0.25% and the mesh reconstructed. Figures 1 (B) and (C) are enlargements of the square in the second quadrant of Figure 1 (A). Figure (C) shows an enlargement after the radius has been increased, Figure (B) shows the original radius. The thing to look for is differences in (B) and (C); if one looks closely small changes in the triangles may be detected.

One additional feature, that may or may not appear in the simulations considered, is a *minimum resolution* of input variables. Some simulation software, or rather meshing routines, have a minimum variable resolution $\delta > 0$, which means that for two different sets of input variables that only differ by $\epsilon \in [0, \delta]$ the output will be the same (numerical noise aside). Using an algorithm

[2]The mesh was generated using *DistMesh*, a MATLAB-based script for creating meshes describing arbitrary geometries. For more information, see http://www-math.mit.edu/∼persson/mesh/.

that tends to place points close to each other could possibly waste function evaluations in this way. There may also exist a tolerance in the production line, in which case points lying closer than the tolerance are of no practical interest.

In industrial applications there often exists more than one objective function (for instance, using output of several simulations or several outputs from one simulation) to be minimized. In this scenario, methods for single objective optimization will not work, instead methods for so called multiobjective optimization must be used. Multiobjective methods focus on finding a set of *Pareto* optimal solutions to the objective functions rather than just one single solution. A Pareto optimal set is a set of feasible solutions in which every member is strictly better in at least one objective function.

1.2 Previous research

There is no shortage of (global) optimization algorithms and concepts in the research community, but few of them seem to be suited for the kind of problem this thesis aims to optimize. The absence of analytic derivatives narrows the field to algorithms using no first or second order information about the function; this excludes for instance globalized quasi-Newton methods. Algorithms inspired by physics and natural selection (simulated annealing and genetic algorithms for instance) generally use far too many function evaluations to be practical, although they are able to handle a wide field of problems. Direct methods (pattern search methods and simplex methods[3] for instance, see [LTT00] for a taxonomy of direct search methods) assumes the existence of a gradient, but do not use analytical gradients or approximations thereof, they may however exhibit slow convergence in some cases.

Other algorithms not falling into any of the above categories include Shor's r-algorithm [Sho85], DIRECT (*DIviding RECTangles*) [JPS93] and MCS (*Multilevel Coordinate Search*) [HN99] which uses an approach similar to that of DIRECT but tries to remedy convergence problems that may arise under certain conditions. These algorithms generally converge to a global minimum in a smaller number of function evaluations than those mentioned previously, according to numerical experiments on standard text-book reference functions. This summary is not intended to be complete, its purpose is to give a feeling of the depth of the field.

Response Surface Methods (RSMs) are common to a class of optimization algorithms especially suited for the type of problem considered here. Basically, an RSM provides an approximation, a *surrogate model*, of the expensive function (simulation). This surrogate model may then be used in place of the expensive function when performing analyses, such as visualization and optimization. Examples of response surfaces are linear or quadratic approximations, or some sort of interpolation. A widely used form of interpolation consists of linear combinations of basis functions. There are different choices of these basis functions, the most popular ones being kriging, thin plate splines, cubic splines and multiquadratics. Kriging basis functions also have a statistical property making them especially suitable for estimating uncertainty in the surrogate. In general, the more points used when creating an interpolation, the more accurately the interpolation depicts the underlying function. For a more in-depth study of different choices of response surfaces, see for instance [Jon01]. Common to all surrogate-based optimization methods is the concept of iteratively selecting new points for evaluation using the surrogate, and using the new points to further refine the surrogate model.

The above mentioned article also includes basic categorizations of different types of methods based on response surfaces, the major categorization being that a method is either a *one-stage* or a *two-stage* approach. A two-stage approach can be described by the following steps:

0. Create and evaluate an initial set of points.
1. Create a surrogate model using the evaluated points.
2. Select and evaluate a new point using data from the surrogate model.

[3] This is not the simplex method for linear programs.

3. Go to step 1, unless a stopping criterion is met.

A one-stage approach, on the other hand, does not explicitly create a surrogate model, rather it compounds step 1 and 2 into one single step. The steps of the general outline above, are explained as follows.

The initial sample in step 0 has to be created without any à priori knowledge of the function. Usually the set is created using some design of experiments such as latin hypercubes; it is however not at all obvious how to select an optimal number of points. [JSW98] suggest an initial sample roughly of size $10d$ (where d is the dimension of the problem), whereas [Gut01] uses 2^d points (one point in each corner of the hyperrectangle defined by the box-constraints). The size does also seem to be dependent on what kind of basis function is used in the interpolation. The risk when choosing a small number of initial points is that the initial surrogate model may be unable to give a satisfactory description of the underlying function, and thus will guide the analysis in step 2 poorly. When starting with a large number of points, the risk lies in wasting precious function evaluations.

Step 2 is the most critical step of any algorithm of this class, since this is where function evaluations may be saved. The algorithm must balance local and global searches so that the information in the surrogate model is used, but also so that no part of the domain is left completely unexplored. A local search entails more or less trusting the surrogate model, and selecting points near or at the (global) minima of the model. A global search enables the exploitation of previously unexplored regions of the domain, selecting points where few points previously have been selected and thus reducing the uncertainty of the model in these areas. Both extremes are important; a too local search strategy may make the algorithm converge to a local minimum while too global a search strategy wastes function evaluations by not exploiting the information in the surrogate model. The stopping criterion in step 3 varies from problem to problem. When working with time-consuming simulations, the number of function evaluations is usually bounded by some allotted running time for the entire optimization procedure.

The algorithm proposed in this thesis falls into the two-stage approach category, and uses a novel approach to selecting new points for evaluation. The response surface consists of an approximation using thin plate splines as basis functions. Below follows a short chronological summary of previous research in the area of RSM-based optimization algorithms. Selected strategies for choosing new points are explained in more detail in Section 2.4. The summary is not complete, but rather provides an informal overview. Points highlighted are the different algorithms ability to handle the common features discussed in the previous section.

Jones et al. [JSW98] propose an algorithm based on kriging basis functions, called Efficient Global Optimization (EGO), and uses *expected improvement* of the surrogate to select new points. This algorithm is among the first to use RSMs and is frequently cited in recent publications. It can handle non-linear variable constraints.

Gutmann [Gut01] bases the response surface on Radial Basis Functions (RBFs, such as thin plate splines and cubic splines). This is a one-stage approach where the new points are selected using a *target-value* f^* which is varied in cycles during the course of iterations to achieve a balance between local and global search. In short, the point selected for evaluation is that which maximizes the smoothness of the interpolation, using some external global optimization algorithm. The algorithm handles non-linear constraints by passing them on to the external solver.

Nakayama et al. [NAS02] use a form of trust-region to select the new points and to speed up local convergence. The interpolation is based on RBFs. An external global solver is employed to find the minimizer of the surrogate model.

Regis et al. [RS05] balance local and global searches by weighting the global minimization of the surrogate model against the distance from previously selected points. In this way, basins with small function values but having had many previous visits are not visited as often as areas with few points. This ensures a balance between improving on function values and on decreasing the uncertainty of the model. The algorithm relies on external solvers for the auxiliary minimization problem, and may handle non-linear variable constraints depending on the external solver.

Sòbester et al. [SLK05] use an approach similar to that of Jones called *generalized expected improvement* in selecting new points. This approach lets the user bias the balance between local

and global search in one direction or the other, possibly achieving faster convergence on certain functions than for instance Jones and Gutmann. As Jones, this algorithm has the ability to handle non-linear variable constraints.

Huang et al. [HANZ06] provide the first RSM-based algorithm that considers noisy objective functions. The algorithm is an extension of Jones' EGO algorithm to stochastic systems. The algorithm uses *augmented expected improvement* to select new points. Can be adapted to handle non-linear constraints, according to the article.

Regis et al. [RS07] suggest improvements to algorithms using approaches similar to those of Gutmann and Regis, the improvements being the use of a trust region to speed up local convergence.

All of the above methods are focused on single-objective optimization, although some of them may be extended to multiple objectives. [Kno06] for instance developed ParEGO, an adaptation of Jones' EGO algorithm, for multiobjective optimization. It does not however handle noisy functions.

Some of the algorithms presented tend to "cluster" points, that is, to select new points for evaluation close to already existing points, by emphasizing local search too much. This may all be well in a controlled test environment, when the goal is to pin-point the optimum in as few iterations as possible. When dealing with simulation related problems, however, this is not an especially desired quality since focus lies more on locating a region containing a global minimum than in the ability to pin-point and evaluate it. When the problem is influenced by noise, the situation becomes even more clear, since the function value at the global minimum may be the result of noise. As mentioned earlier, certain types of simulation software, or more accurately, certain types of meshing routines, may also have a lower bound on the resolution of input parameters, in which case it is not beneficial to cluster points close to each other. There may also be a tolerance in the production line involved, limiting the accuracy of the final product. Ideally, one would want to be able to exercise control over the minimum distance between chosen points, when necessary. Other problems, for instance in the algorithm proposed in [Gut01], concerns the over-emphasis of domain boundaries, that is, the algorithm tends to place many points at the boundaries.

1.3 Desired properties of an optimization algorithm

As discussed in the preceding section, the existing algorithms may display certain unwanted behavior in real-life situations: especially in clustering points too close to each other, and in over-emphasizing border regions. The algorithm proposed is designed to solve the type of problems described in Section 1.1, as well as to be able to handle a number of additional features. To summarize, these features are:

1. avoid clustering of points
2. avoid over-emphasis of border regions
3. accept a minimum resolution of variables
4. handle multiple objective functions

In order to achieve these goals, the proposed algorithm, henceforth referred to as *qualSolve*, implements a novel way of selecting new points and a way of handling noise in the objective function. The mechanism for selecting new points is called the *quality function*; basically it measures the quality of the surrogate model. The next point for evaluation is the maximizer of the quality function, that is, the point where the quality of the surrogate model will benefit most by a function evaluation. Quality with respect to a not yet evaluated point consists of two parts: how much the total uncertainty of the model will decrease and the value of the model at this point. This is achieved by integrating over an approximation of the uncertainty in the model multiplied by a weight function depending on the surrogate model's value. In this way a balance between local and global search is attained. The algorithm can also handle multiple objective functions, by creating one surrogate model for each objective function, and letting the weight function use a measure of distance to

the surrogates' Pareto front instead of the surrogate function value, as in the single objective case. Noise is handled by letting the surrogate model approximate the objective function values, rather than interpolating them. In this way, deviations from the noisy objective function values are introduced, allowing the surrogate model to, in a sense, more accurately represent the "true" (noiseless) function. This approach relies on the assumption that the noisy function is an approximation of the "true" function containing no noise.

2 Theory and background

Presented in this section is a comprehensive summary of some of the theory needed to develop the methods of this thesis. Some subsections, notably 2.3.3 and 2.3.4 on error estimation and approximation respectively, contain new theory not found in existing literature. This theory includes a novel way of bounding the uncertainty of a radial basis functions based interpolation.

2.1 Continuous optimization

This master's thesis is centered around optimization, and an understanding of the underlying concepts is beneficial. Here follows a short introduction to the general field of optimization, and continuous optimization in particular. For a more comprehensive text on continuous optimization, see e.g. [AEP05].

In mathematical terms, the most general statement of a global optimization problem is: given a function $f : \Omega \to \mathbb{R}$, $\Omega \subseteq \mathbb{R}^d$, find $f^* = \inf f(\boldsymbol{x})$, $\boldsymbol{x} \in \Omega$, or equivalently

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}), \\ \text{subject to} \quad & \boldsymbol{x} \in \Omega. \end{aligned} \tag{1}$$

The function f is called the objective function, and the point $\boldsymbol{x}^*$ satisfying the above condition is called a global optimum, with the corresponding function value $f^* = f(\boldsymbol{x}^*)$. Depending on the objective function, there may also exist a number of local optima, defined as:

Definition 1 (Local minimum). *Let $f : \Omega \to \mathbb{R}$, $\boldsymbol{x}, \boldsymbol{y} \in \Omega$ and $\boldsymbol{x} \neq \boldsymbol{y}$. $\boldsymbol{y}$ is a local minimum if $f(\boldsymbol{y}) \leq f(\boldsymbol{x})$ and $\boldsymbol{x} \in B_\epsilon(\boldsymbol{y}) \cap \Omega$ for an $\epsilon > 0$ small enough, where $B_\epsilon(\boldsymbol{y})$ defines an open ball centred around $\boldsymbol{y}$ with radius ϵ.*

Qualitatively, a local minimum $(\boldsymbol{x},\ f(\boldsymbol{x}))$ is a point in the graph of the function having a lower function value than any other point in the graph in a small area surrounding the point. By definition, a global optimum is also a local optimum.

The domain (or region) Ω of the problem may be the whole of $\mathbb{R}^d$, but it may just as well be defined only on a (non-connected) subset $\Omega \subseteq \mathbb{R}^d$, called the feasible region (or set). The feasible region is usually defined by so-called constraints, and there are different types of constraints:

$$\boldsymbol{x}_l \leq \boldsymbol{x} \leq \boldsymbol{x}_u, \tag{2}$$
$$h_i(\boldsymbol{x}) = 0, \qquad i = 1, \ldots, k, \tag{3}$$
$$g_j(\boldsymbol{x}) \leq 0, \qquad j = 1, \ldots, l, \tag{4}$$

where $h_i : \Omega \to \mathbb{R}$ and $g_j : \Omega \to \mathbb{R}$. These examples represent the most common constraints, but there are more ways of defining the feasible region. Constraint (2) is usually called a *box-constraint*, since it defines a box-shaped region in two and three dimensions. Constraints (3) and (4) are called equality and inequality constraints, respectively, and may be constructed using arbitrary functions. An optimization problem without any constraints on the feasible region is called an unconstrained problem.

Before the actual solution process can begin the mathematical model has to be properly formulated in an unambiguous and correct way. This is the first stage of the optimization procedure and requires a thorough understanding of the problem. A good understanding of the problem can often reduce the difficulties involved in the solution process, if certain characteristics of the problem for instance can be exploited. Important aspects of the problem includes interpreting and formalizing exactly what the goal of the optimization is, that is, constructing the objective function. What quantity is to be optimized? Is it a minimization or a maximization problem? Are there more than one quantity, possibly in conflict with each other, to be optimized? Are there any constraints involved, and if so, of what kind? Is there any possibility of simplifying the problem beforehand?

2.1.1 Characteristics of the objective function

The objective function can be subjected to a number of different characterizations. For instance, if f is linear and the optimization is subject to linear constraints, problem (1) is referred to as a linear problem. A linear constraint is of the form $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} \leq b_i$, where $\boldsymbol{a}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$. By exploiting the structure of a general linear problem and the fact that any linear function is convex, there are ways of creating very efficient methods and algorithms for solving these problems. Furthermore, if the domain is non-empty and bounded, a global optimum is always located at one of the extreme points (that is, the vertices, since the domain is defined by piecewise linear segments) of the domain.

f may be continuously differentiable ($f \in C^1$), in which case the gradient vector $\nabla f(\boldsymbol{x}) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d})^{\mathrm{T}}$ may be calculated. The gradient can be used by an optimization algorithm to, for instance, find a direction of descent. That is, a direction in the parameter space in which the objective function value decreases.

Another important characteristic is that of convexity:

Definition 2 (Convex function). *Suppose that $S \subseteq \mathbb{R}^d$. A function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is convex at $\bar{\boldsymbol{x}} \in S$ if*

$$\left.\begin{array}{l} \boldsymbol{x} \in S \\ \lambda \in (0,1) \\ \lambda\bar{\boldsymbol{x}} + (1-\lambda)\boldsymbol{x} \in S \end{array}\right\} \implies f(\lambda\bar{\boldsymbol{x}} + (1-\lambda)\boldsymbol{x}) \leq \lambda f(\bar{\boldsymbol{x}}) + (1-\lambda)f(\boldsymbol{x}).$$

The function f is convex on S if it is convex at every $\bar{\boldsymbol{x}} \in S$.

Convexity is a very useful characterization of a function, since it implies that every local minimum is a global minimum, if the domain of optimization is also convex. The definition is well known, this particular definition is however taken from [AEP05]. If the definition of convexity is modified with strict inequality and the extra condition that $\boldsymbol{x} \neq \bar{\boldsymbol{x}}$, convexity is referred to as being *strict*. For a strictly convex function, in a convex domain, the global minimum is also unique. If a function can be characterized as being (strictly) convex, the optimization procedure is much simplified.

The objective function f may also be non-linear but continuous, and hence the optimization procedure may locate a number of local minima. There are several methods and algorithms for finding a local minimum for these type of functions; there are also ways of characterizing a local minimum. The same cannot as easily be said for a global minimum, unless there are some special characteristics of the function that may be exploited.

2.1.2 Characteristics of optima

Before considering different characteristics of the optimal solutions, are there really any? Weierstrass theorem is a very powerful and basic condition for the existence of a global optimum. It enforces some mild conditions on the objective function and the corresponding domain that are fulfilled for the functions considered in this thesis. The theorem states[4]:

[4] This is a somewhat less general formulation than the one found in [AEP05, p. 80] , but it is still relevant for the functions considered in the scope of this thesis.

Theorem 3 (Weierstrass' theorem). *Let $\Omega \subseteq \mathbb{R}^d$ be a nonempty and closed set, and $f : \Omega \to \mathbb{R}$ be a continuous function on Ω. If f is weakly coercive with respect to Ω, then there exists a nonempty, closed and bounded (thus compact) set of globally optimal solutions to the problem (1).*

Basically, that a function is weakly coercive on a non-empty and closed set Ω means that either the set is also bounded, or $f(\boldsymbol{x}) \to \infty$ as $\|\boldsymbol{x}\| \to \infty$, for $\boldsymbol{x} \in \Omega$. All feasible sets in this thesis are defined by box constraints, and consequently bounded.

There are several ways of characterizing the local optima of a function subject to constraints. The simplest case is when considering the unconstrained domain. For the unconstrained optimization problem of a non-linear function in C^1, a local minimum has the necessary property that it is a stationary point of the function. A stationary point has the property of the gradient (or derivative, in the case of one dimensional problems) being equal to zero. This is however not a sufficient criterion, since the stationary point may be a "saddle point". A sufficient criterion for local optimality for functions in C^2, is that the Hessian (second derivative in the one dimensional case) must be positively definite (positive in the one dimensional case). More in-depth descriptions of optimality conditions may be found in literature on continuous optimization.

A criterion for global optimality can only be constructed when the objective function is convex, since every local minimum is also a global minimum. For arbitrary non-convex functions there are no general criteria for a global optimum. Therefore it is in general hard, if not impossible, to determine whether a local optimum found is also a global one.

For constrained problems, not only the stationary points may be local optima, but also points located at the boundaries. One possible way of finding an optimum would be to check all stationary points and all boundary points. In one dimension this method may work, but it quickly becomes impossible as the dimensionality increases. For functions in C^1 and C^2 there exists necessary and sufficient criteria for local optima, the Karush-Kuhn-Tucker (KKT) conditions for instance. The KKT criterion is also sufficient for global optimality if the function f and the domain Ω both are convex.

There are several algorithms for local optimization, quasi-Newton and SQP (Sequential Quadratic Programming) for instance, and these are built on the different characterizations of optimal points. However, since there is no general characterization of global optimal points, it is also hard to develop a global algorithm for general problems. One can never be quite sure that the global optimum has been reached, until every local optimal point has been visited and compared. There are of course exceptions; if the problem can be analyzed and the structure of the problem exploited in some way, several parts of the region may be disregarded. Branch-and-Bound is one such technique, when the structure of the problem is utilized to calculate a lower bound of a subset of feasible solutions and thus being able to speed up the global search by "cutting away" solution branches that would not yield any better solutions. There is however no such general method available, and heuristic methods (without guarantee of "fast" convergence) must be employed.

When considering functions that are expensive to evaluate and without analytical derivatives, a new approach must be considered. One such approach is to create a surrogate model of the objective function that is considerably less expensive to evaluate, and employ the above discussed methods.

2.2 Multiobjective optimization

In engineering applications as well as other real-life applications there often arises the need to optimize not only one objective function, but several. This may for instance be the case when designing an engine for a car or a truck, the goal being to minimize the fuel consumption at some given load while at the same time minimizing the amount of soot in the exhaust. In this scenario there are two objective functions, and most likely their optima are not obtained at some common choice of input variables. Another example may be found in bridge construction, where one may want to minimize the weight of the bridge and maximize the stiffness simultaneously. In a multiobjective optimization (MOO) setting, the different objective functions usually obtain their respective minima at different points in the parameter space.

Mathematically, the MOO problem may be formulated as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^{\mathrm{T}}, \\ \text{s.t.} \quad & \boldsymbol{x} \in \Omega, \end{aligned} \tag{5}$$

where $m \geq 2$, $f_i : \Omega \to \mathbb{R}$ are scalar functions, $\Omega \subseteq \mathbb{R}^d$ and $\boldsymbol{f} : \Omega \to Z$ a vector-valued function with m components. Ω is denoted the *feasible region* to which the vectors $\boldsymbol{x} = (x_1, \ldots, x_d)^{\mathrm{T}}$ belong. $Z = \{\boldsymbol{z} \in \mathbb{R}^m : \boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x}),\ \boldsymbol{x} \in \Omega\}$ is denoted the *feasible objective region*, and contains all possible objective vectors $\boldsymbol{z} = (z_1, \ldots, z_m)^{\mathrm{T}}$ where $z_i = f_i(\boldsymbol{x})$, $\boldsymbol{x} \in \Omega$.

One way of attacking this problem would be to simply construct a weighted sum of the objectives, use this sum as a single objective function and solve it using a suitable method. This however requires a decision of how the objective functions should be weighted, and there may not always exist information on which to base such a decision. The objectives may also be incommensurable (in different units), in which case a weighted sum will have no physical interpretation.

This section intends to give a cursory introduction to the concept of multiobjective optimization; the interested reader may for instance turn to [Mie99] for a more in-depth discussion on the subject. The following definitions belong to the public domain but have been cited from [Mie99]; the concepts introduced below are naturally biased towards what is to come later in the thesis.

2.2.1 Pareto optimality

It is somewhat problematic to strictly define optimality in the MOO case, since there is only partial ordering in the objective space. It it not obvious how to compare two vectors (for instance, is $(1,3)^{\mathrm{T}}$ greater than $(3,1)^{\mathrm{T}}$?). There is however one way of classifying decision vectors, called *Pareto optimality*:

Definition 4 (Pareto optimality). *A vector $\boldsymbol{x}^* \in \Omega$ is Pareto optimal[5] if there does not exist another vector $\boldsymbol{x} \in \Omega$ such that $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{x}^*)$ for all $i = 1, \ldots, m$ and $f_j(\boldsymbol{x}) < f_j(\boldsymbol{x}^*)$ for at least one index j.*

An objective vector $\boldsymbol{z}^* \in Z$ is Pareto optimal if there does not exist another objective vector $\boldsymbol{z} \in Z$ such that $z_i \leq z_i^*$ for all $i = 1, \ldots, m$ and $z_j < z_j^*$ for at least one index j; or equivalently, $\boldsymbol{z}^*$ is Pareto optimal if the vector corresponding to it is Pareto optimal.

There may be (infinitely) many Pareto optimal vectors, denoted a *Pareto optimal set* or *Pareto front*. For clarity and future reference, the *Pareto optimal set* and *Pareto optimal objective set* are defined as:

Definition 5 (Pareto optimal (objective) set). *The Pareto optimal set $\Omega^* \subseteq \Omega$ is defined as the set consisting of all Pareto optimal vectors $\boldsymbol{x}^* \in \Omega$ with respect to a given vector-valued function $\boldsymbol{f}$.*

The Pareto optimal objective set $Z^* \subseteq Z$ is defined as the set consisting of all Pareto optimal objective vectors $\boldsymbol{z}^* \in Z$ with respect to a given vector-valued function $\boldsymbol{f}$.

The Pareto optimal (objective) sets may be non convex and non connected. Another characteristic of points in the Pareto optimal set following from the definition of Pareto optimality is *non dominance*:

Definition 6 (Non dominance). *An objective vector $\boldsymbol{z}^* \in Z$ is non dominated if there exists no other vectors $\boldsymbol{z} \in Z$ in the negative orthant of $\boldsymbol{z}^*$. That is, $\left(\{\boldsymbol{z}^*\} - \mathbb{R}_+^m \setminus \{0\}\right) \cap Z = \varnothing$.*

In the two dimensional Cartesian coordinate system, the negative orthant, with respect to the origin, is the third quadrant. By the same coin, an objective vector $\boldsymbol{z}^*$ can be said to dominate all vectors in its positive orthant. The optimal solutions of the individual scalar objective functions f_i are of course also members of the Pareto front; they are located at the extremes of the front (when such a term is applicable). The (often infeasible) objective vector where all objectives reach their

[5] The name Pareto comes from Vilfredo Pareto, a French-Italian economist and sociologist in the late nineteenth century.

optima simultaneously is called the *ideal objective vector*, denoted $\boldsymbol{z}^\star = (z_1^\star, \ldots, z_m^\star)^{\mathrm{T}}$. $z_i^\star$ is defined as the optimal value of the optimization problem to $\min_{\boldsymbol{x}\in\Omega}\ f_i(\boldsymbol{x})$.

When moving along a Pareto front, at least one of the objectives must deteriorate in value (if all objectives had improved in value, the point(s) in question would not be on the Pareto front).

One additional concept of Pareto optimality has to be defined: the concept of *weak Pareto optimality*.

Definition 7 (Weak Pareto optimality). *A decision vector $\boldsymbol{x}^* \in \Omega$ is weakly Pareto optimal if there does not exist another decision vector $\boldsymbol{x} \in \Omega$ such that $f_i(\boldsymbol{x}) < f_i(\boldsymbol{x}^*)$ for all $i = 1, \ldots, m$.*

An objective vector $\boldsymbol{z}^* \in Z$ is weakly Pareto optimal if there does not exist another objective vector $\boldsymbol{z} \in Z$ such that $z_i < z_i^*$ for all $i = 1, \ldots, m$.

The Pareto optimal set is a subset of the weakly Pareto optimal set. From a mathematical point of view, all Pareto optimal solutions are equally acceptable but in a real-life optimization problem only one solution is required. If the MOO problem for instance is a model of a bridge, then the ultimate goal of the project may be to build only one good bridge. The mathematical formulation of the problem contains no further data aiding in preferring one particular solution over an other, so an individual, capable of deciding whether $\boldsymbol{f}(\boldsymbol{x}_1^*)$ is preferable to $\boldsymbol{f}(\boldsymbol{x}_2^*)$, has to make the choice. This individual is usually called the *decision maker*. Compare to single objective optimization, where given the optimal solution, no subjective choice is involved.

Situations may also arise when the decision maker from the beginning has made clear that not all points on the Pareto front are of interest. This may be the case when the optimal vector for one of the objectives results in far too poor values in some other objective. In the engine example at the beginning of Section 2.2, there may for instance exist legislative upper bounds on the level of soot in the exhaust, so that input parameters causing the soot objective to exceed its bound may be disregarded in the optimization stage. Knowing such bounds à priori may speed up the optimization process, since some areas of the feasible objective region may be cut off.

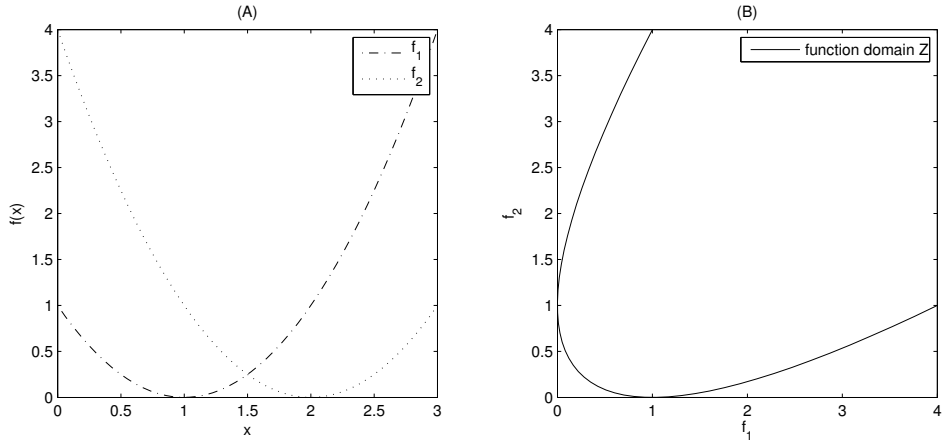


Figure 2: The figure in (A) depicts two functions, $f_1(x) := (x-1)^2$ and $f_2(x) := (x-2)^2$, plotted on the interval $[0, 3]$. The figure in (B) depicts the corresponding feasible objective function domain, as a parabolic curve.

Figure 2 (A) depicts the two simple one-dimensional functions $f_1(x) := (x-1)^2$ and $f_2(x) := (x-2)^2$, with globally minimal solutions over $\mathbb{R}$ at $x = 1$ and $x = 2$ respectively. Figure 2 (B) depicts the feasible objective space of the two functions. Figure 3 illustrates the concept of non dominance, where the point marked by an asterisk in Figure 3 (B) is non dominated. Finally, Figure 4 (B) illustrates the Pareto front Z^* of the two objective functions, that is, all non dominated points in the feasible objective region. Correspondingly, the Pareto optimal set Ω^* is shown in Figure 4 (A), lying in the interval $[1, 2]$, between the two optima.

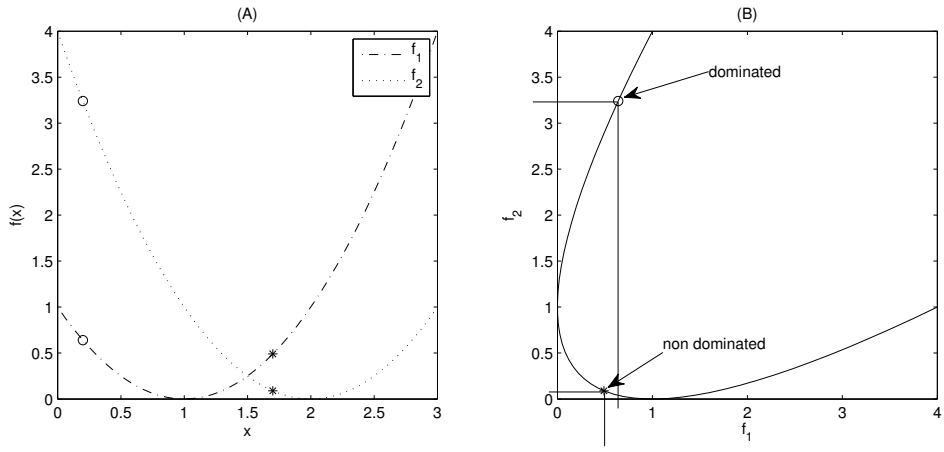


Figure 3: Two points in the parameter space and their corresponding points in the objective space, illustrating the concept of (non) dominance.

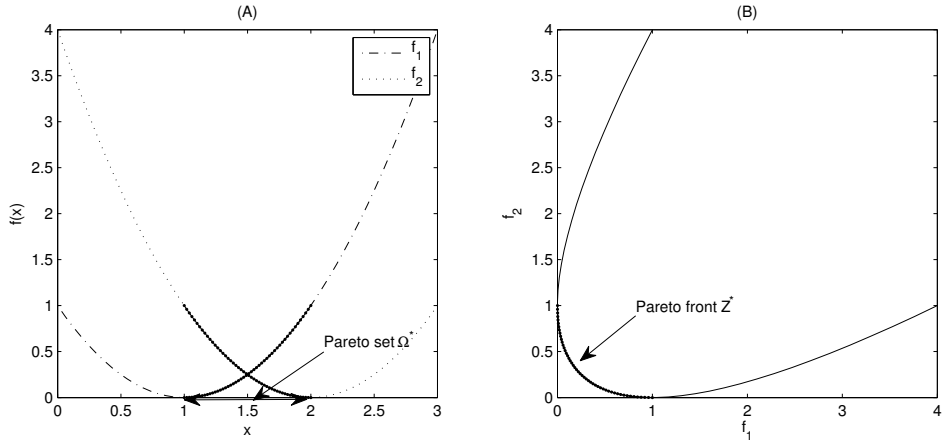


Figure 4: The Pareto optimal set and the Pareto front of the two functions.

2.2.2 Methods for finding the Pareto front

The easiest and most straight forward way of solving an MOO problem is to create a convex combination of the objective functions and solve the resulting single objective optimization problem. The problem (5) is then replaced by

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \sum_{i=1}^{m} \lambda_i f_i(\boldsymbol{x}), \\ \text{s.t.} \quad & \boldsymbol{x} \in \Omega, \end{aligned}$$

where $\lambda_i > 0$ and $\sum_{i=1}^{m} \lambda_i = 1$. This approach would not yield all points on the Pareto front, and solutions in non-convex parts of the front would be missed out entirely. Furthermore, it requires à priori knowledge from the decision-maker about how to weight the objectives, knowledge that may not be readily available.

When dealing with MOO problems arising out of, for instance, simulations, information about derivatives may not be available. In these cases, an Evolutionary Algorithm (EA) may be applied.

What makes MOO problems especially suited for EAs is the structure of the problem, the points on the Pareto front may be viewed as individuals in a population. The individuals also have to be coupled with some measure of quality, how likely a particular individual is to "survive" compared to other individuals. An example of a quality measure may be the number of points being dominated by the point in question.

In applications where evaluation time of the objectives is an issue, EAs are however at a disadvantage since the number of evaluated points may be quite large.

2.3 Interpolation and approximation using radial basis functions

In the task of optimizing an expensive black box function it is helpful to create a surrogate model and utilize it in order to find new points for evaluation (a decision that has to be taken with care). Hence now follows a discussion of interpolation and approximation with radial basis functions; the aim of which is to motivate the radial basis function interpolation made. Also, error estimates for it are introduced and insight into the choice of particular basis functions is given. Most of the material in this section is taken from [Wen05] (which is recommended for a thorough investigation of the concepts presented here). The exceptions are the subsection dealing with approximation (Section 2.3.4) and Proposition 26 which are new contributions.

2.3.1 Interpolating scattered data

Interpolation is the task of finding a continuous function that corresponds to the data at some evaluated points, i.e., a function $S \in C^0$, such that given a set of data points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and corresponding function values $\boldsymbol{f} = (f_1, \ldots, f_n)^T$ it holds that:

$$S(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, n.$$

Naturally this problem has no unique solution, as C^0 is an infinitely dimensional space, but by considering specific finite dimensional linear spaces uniquely solvable problems can be formulated. An intuitive space to use for interpolation of n data points in one dimension is the space of $n-1$ degree polynomials (π_{n-1}). The problem can be formulated as finding the coefficients α_i in order to solve

$$\sum_{j=1}^{n} x_i^{j-1} \alpha_j = f_i, \qquad i = 1, \ldots, n.$$

This is equivalent to solving the linear system

$$\begin{aligned} &A\boldsymbol{\alpha} = \boldsymbol{b}, \\ &A_{ij} = x_i^{j-1},\ b_i = f_i, \qquad i, j = 1, \ldots, n. \end{aligned}$$

The existence of a solution to such a problem can be guaranteed, and the interpolation space has the appealing property of being independent of data point location and data values (it depends only on the amount of data points). In spite of these excellent properties, the use of polynomials even for one dimensional problems is limited, due to the fact that the degree of the polynomial interpolant increases with the number of evaluated points. This often results in a very strongly oscillating interpolation (see Figure 5), which is undesirable when interpolating most functions. In order to avoid this, the one dimensional space is split into intervals between the data points, and different polynomial interpolations (often cubic, that is of 3:rd degree) are used in each interval. The piecewise polynomials are called *splines* and the problem can be formulated as follows: let the data points be ordered as $a < x_1 < \ldots < x_n < b$, define $x_0 = a$, $x_{n+1} = b$ and the following function space:

$$S_3(X) = \{S \in C^2([a, b]) : S|_{[x_i, x_{i+1}]} \in \pi_3(\mathbb{R}), i = 0, \ldots, n\},$$

the task is to find $S \in S_3(X)$ so that S interpolates the data points

$$S(x_i) = f_i, \qquad i = 1, \ldots, n.$$

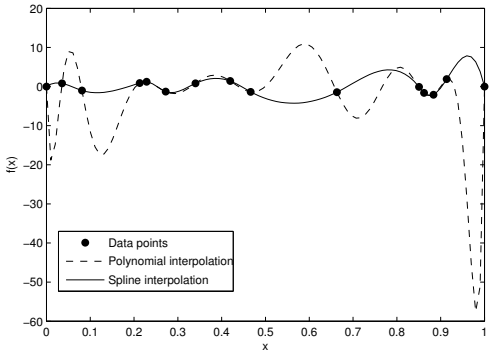


Figure 5: An illustration of different interpolations with polynomials and splines.

As S_3 is an $n+4$ dimensional space[6], a unique solution cannot be found (only n degrees of freedom are covered by the data). A uniquely solvable problem can be formulated by replacing S_3 with the natural spline space ($\mathcal{N}S_3$), which reduces the dimensionality by using only first order polynomials in the first and last intervals:

$$\mathcal{N}S_3(X) = \{S \in S_3(X) : S|_{[a,x_1]}, S|_{[x_n,b]} \in \pi_1(\mathbb{R})\}.$$

Using the natural spline space, a unique solution can be obtained for all data points and values. Compared to the case of using π_{n-1} as the interpolation space, the oscillating property is now avoided (see Figure 5), however at the cost of the data point location independent space π_{n-1} being replaced by $\mathcal{N}S_3(X)$, which is highly location dependent.

Unfortunately, interpolating multivariate functions in a higher dimension than one is more difficult. It can be shown that interpolating arbitrary data sites with polynomials in higher dimensions is not possible[7]. Although interpolation is still possible under some restrictions[8], the problem with oscillating interpolations, when many data points are used, remains. Generalizing splines into higher dimensions also proves to be difficult. The domain has to be divided into areas corresponding to the intervals in one dimension. This can be done by triangles in two dimensions but even in this simple case, the dimensionality of the spline space is in general unknown, hence the method is not suited for higher dimensions (see [Sch91]). Fortunately there are other ways to generalize the one dimensional spline interpolation into a multivariate setting. This is done by reformulating the natural cubic spline interpolation. According to [Wen05, Ch. 1] every natural cubic spline S has a representation of the form:

$$S(x) = \sum_{j=1}^{n} \alpha_j \phi(|x - x_j|) + p(x), \qquad x \in \mathbb{R} \tag{6}$$
$$\sum_{j=1}^{n} \alpha_j = \sum_{j=1}^{n} \alpha_j x_j = 0,$$

where the basis function $\phi(r) = r^3, r \geq 0$, and $p \in \pi_1(\mathbb{R})$. This provides a motivation for proceeding into multivariate interpolation and introducing *radial functions*:

Definition 8 (Radial function). *A function $f : \mathbb{R}^d \to \mathbb{R}$ is called a radial function if there exists a*

[6] 4 degrees of freedom at each of the $n+1$ intervals, 3 degrees fixed at each of the n inner nodes, as the function and its derivatives of first and second order must be continuous.

[7] No Haar spaces exist, i.e. no interpolating linear spaces of dimension N exist to interpolate N arbitrary data sites in a unique way according to the Mairhuber-Curtis theorem [Wen05, p. 19].

[8] The points have to be π_n unisolvent (Definition 13), see [Wen05, Ch. 2].

univariate function $\phi : [0, \infty) \to \mathbb{R}$ *such that*

$$f(\boldsymbol{x}) = \phi(\|\boldsymbol{x}\|), \qquad \boldsymbol{x} \in \mathbb{R}^d.$$

In the definition above, $\|\cdot\|$ denotes the Euclidean norm. Identifying ϕ in (6) as a radial function in one dimension and temporarily ignoring the polynomial p, the RBF-based interpolation for multivariate problems can be introduced: find the interpolation coefficients α_i such that

$$\begin{aligned} S(\boldsymbol{x}) &= \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|), \\ S(\boldsymbol{x}_i) &= f_i, \qquad i = 1, \ldots, n, \end{aligned} \tag{7}$$

or equivalently solve the linear system

$$\begin{aligned} &A\boldsymbol{\alpha} = \boldsymbol{b}, \\ &A_{ij} = \phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|),\ b_i = f_i, \qquad i, j = 1, \ldots, n. \end{aligned} \tag{8}$$

In order for the system (8) to be uniquely solvable, the matrix A has to be invertible. A way to ensure this, is to use *positive definite* basis functions.

Definition 9 (Positive definite function). *A continuous function* Φ*:* $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *is called positive definite if, for all* $n \in \mathbb{N}$*, any n pairwise different points* $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ *and all* $\mathbf{c} \in \mathbb{R}^n \backslash \{\mathbf{0}^n\}$

$$\sum_{j=1}^{n} \sum_{i=1}^{n} c_j c_i \Phi(\boldsymbol{x}_j, \boldsymbol{x}_i) > 0.$$

The function ϕ*:* $[0, \infty) \to \mathbb{R}$ *is called positive definite in* $\mathbb{R}^d$ *if* $\Phi(\boldsymbol{x}, \boldsymbol{y}) := \phi(\|\boldsymbol{x} - \boldsymbol{y}\|)$ *is positive definite.*

The definition of semidefiniteness is obtained by changing the the strict inequality $>$ to $\geq$. The use of positive definite radial basis function ensures that the inequality

$$\boldsymbol{c}^{\mathrm{T}} A \boldsymbol{c} = \sum_{j=1}^{n} \sum_{i=1}^{n} c_j c_i \phi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|) > 0, \quad \boldsymbol{c} \in \mathbb{R}^n \backslash \{\mathbf{0^n}\}$$

holds for all possible data points. This means that A is positve definite and hence invertible. It is sufficient to consider positive definite functions since for all negative definite functions Φ, there exists a positive definite function $-\Phi$. It should also be pointed out that positive definite functions that are not radial could be used for interpolating the data sites, but the radial functions have the attractive property of being invariant under all Euclidean transformations (translations, rotations and reflections, see Section 2.3.2 for implications of this). Further, a simple criterion can be shown for deciding if a radial functions is positive definite:

Definition 10 (Completely monotone). *A function* ϕ *is called completely monotone on* $[0, \infty)$ *if* $\phi \in C^\infty([0, \infty))$ *and*

$$(-1)^l \phi^{(l)}(r) \geq 0$$

$\forall l \in \mathbb{N} \cup \{0\}$ *and all* $r > 0$.

The following theorem is proven in Wendland [Wen05, Thm. 7.14].

Theorem 11. ϕ*:* $[0, \infty) \to \mathbb{R}$ *is positive definite in* $\mathbb{R}^d$ *in all dimensions d if and only if* $\phi(\sqrt{\cdot})$ *is completely monotone on* $[0, \infty)$ *and not constant.*

There are radial functions that are positive definite only in some specific dimensions, while others are positive definite in arbitrary dimensions, these latter ones will be focused upon here. An example of such a function is the Gaussian,

$$\phi(r) = e^{-r^2}.$$

That ϕ is positive definite can easily be checked by a simple differentiation and use of Theorem 11. Although these functions are perfectly legitimate as basis functions for the RBF-based interpolation, there is a possibility to relax the condition of positive definiteness in order to allow a wider range of basis functions. For instance, the previously mentioned spline basis function, $\phi(r) = r^3$, is not a positive definite function. Therefore it is necessary to introduce the notion of *conditionally* positive definite functions.

Definition 12 (Conditionally positive definite function). *A continuous function Φ: $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is called conditionally positive definite of order m if, for all $n \in \mathbb{N}$ any n pairwise different points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and all $\boldsymbol{c} \in V_m \backslash \{\mathbf{0}^n\}$ it holds that*

$$\sum_{j=1}^{n} \sum_{k=1}^{n} c_j c_i \Phi(\boldsymbol{x}_j, \boldsymbol{x}_i) > 0,$$

where

$$V_m = \left\{ \boldsymbol{c} \in \mathbb{R}^n : \sum_{j=1}^{n} c_j p(\boldsymbol{x}_j) = 0, \quad \forall p \in \pi_{m-1}(\mathbb{R}^d) \right\}.$$

The function ϕ: $[0, \infty) \to \mathbb{R}$ is called conditionally positive definite of order m in $\mathbb{R}^d$ if $\Phi(x) := \phi(\|x\|)$ is conditionally positive definite of order m.

The definition of conditional semidefiniteness can be obtained by exchanging the strict inequality $>$ with $\geq$. By using conditionally positive definite functions for the interpolation problem (7) the matrix A in (8) may become singular as $\boldsymbol{c}^{\mathrm{T}} A \boldsymbol{c} > 0$ is only guaranteed for $\boldsymbol{c} \in V_m$. Hence the problem has to be reformulated in order to guarantee the existence of a unique solution to (7): given the points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and function values $f_1, \ldots, f_n$, find $\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^Q$ such that

$$S(\boldsymbol{x}) = \sum_{j=1}^{n} \alpha_j \phi(\|\boldsymbol{x} - \boldsymbol{x}_j\|) + \sum_{k=1}^{Q} \beta_k p_k(\boldsymbol{x}), \tag{9}$$

$$S(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, n,$$

$$\sum_{j=1}^{n} \alpha_j p_k(\boldsymbol{x}_j) = 0, \qquad k = 1, \ldots, Q, \tag{10}$$

where $Q = \dim \pi_{m-1}(\mathbb{R}^d)$, p_k is the basis for this space and therefore (10) is equivalent to $\boldsymbol{\alpha} \in V_m$. This formulation of the interpolation problem is equivalent to solving the linear system:

$$\begin{pmatrix} A & P \\ P^{\mathrm{T}} & \mathbf{0}^{Q \times Q} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \mathbf{0}^Q \end{pmatrix}, \tag{11}$$

where $A_{ij} = \phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$ and $P_{ij} = p_i(\boldsymbol{x}_j)$. For future reference the following notation is used:

$$\widetilde{A} = \begin{pmatrix} A & P \\ P^T & \mathbf{0}^{Q \times Q} \end{pmatrix}.$$

In order to investigate if $\widetilde{A}$ is invertible the notion of *polynomial unisolvence* is introduced:

Definition 13. *The points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^d$ with $n \geq Q = \dim \pi_m(\mathbb{R}^d)$ are called $\pi_m(\mathbb{R}^d)$ unisolvent if the zero polynomial is the only polynomial from $\pi_m(\mathbb{R}^d)$ that vanishes at all of the points in X.*

To exemplify the above definition, π_1 unisolvence is equivalent to demanding that not all points in X belong to a common hyperplane. This can now be used to show the existence of a unique solution:

Proposition 14. *Suppose ϕ is conditionally positive definite of order m and X is $\pi_{m-1}(\mathbb{R}^d)$ unisolvent. Then (11) is uniquely solvable.*

Proof. Suppose $(\boldsymbol{\alpha}^{\mathrm{T}}, \boldsymbol{\beta}^{\mathrm{T}})^{\mathrm{T}}$ lies in the null space of the matrix $\widetilde{A}$; then,

$$\begin{aligned} A\boldsymbol{\alpha} + P\boldsymbol{\beta} &= \mathbf{0}^n, \\ P^{\mathrm{T}}\boldsymbol{\alpha} &= \mathbf{0}^Q. \end{aligned}$$

The second expression means that $\boldsymbol{\alpha} \in V_m$ (V_m as in Definition 12). Multiplying the first equation by $\boldsymbol{\alpha}^{\mathrm{T}}$ results in $0 = \boldsymbol{\alpha}^{\mathrm{T}} A\boldsymbol{\alpha} + (P^{\mathrm{T}}\boldsymbol{\alpha})^{\mathrm{T}}\boldsymbol{\beta} = \boldsymbol{\alpha}^{\mathrm{T}} A\boldsymbol{\alpha}$. Since ϕ is conditionally positive definite $\boldsymbol{\alpha} = \mathbf{0}^n$, and hence $P\boldsymbol{\beta} = \mathbf{0}^n$. This means that $\sum_{j=1}^{Q} \beta_j p_j(\boldsymbol{x}_i) = 0$ for $i = 1, \ldots, n$. As $\sum_{j=1}^{Q} \beta_j p_j(\boldsymbol{x}) \in \pi_{m-1}(\mathbb{R}^d)$, X is $\pi_{m-1}(\mathbb{R}^d)$ unisolvent, and $p_k(\boldsymbol{x})$ are not linearly dependent, it follows that $\boldsymbol{\beta} = \mathbf{0}^Q$. Hence, the null space of $\widetilde{A}$ consists of the null vector, which implies that $\widetilde{A}$ is invertible. □

So, for a conditionally positive definite function ϕ of order m a unique solution to problem (9) is guaranteed if X is $\pi_{m-1}(\mathbb{R}^d)$ unisolvent. Further, an analogue to Theorem 11 can be shown (for a proof, see [Wen05, Thm. 8.19]):

Theorem 15 (Micchelli). *Suppose that $\phi \in C([0,\infty)) \cap C^\infty((0,\infty))$ is given. Then $\Phi = \phi(\|\cdot\|^2)$ is conditionally positive semidefinite of order $m \in \mathbb{N}$ on $\mathbb{R}^d$ in every dimension d, if and only if $(-1)^m \phi^{(m)}$ is completely monotone on $(0,\infty)$.*

Corollary 16. *Suppose, in addition to the conditions of Theorem 15, that $\phi \notin \pi_{m-1}$. Then, ϕ is conditionally positive definite of order m on $\mathbb{R}^d$, for every dimension d.*

Theorem 15 constitutes a powerful tool in investigating the nature of potential radial basis functions. A quick calculation shows that $r \mapsto \phi(r) := r^3$ is a conditionally positive definite function of order two, which explains the presence of the polynomial term in (6). Table 1 summarizes commonly used (conditionally) positive definite radial basis functions, and they can be seen as 1D plots in Figure 6. Their definiteness can easily be verified by simple calculations. All these functions can be used in arbitrary dimensions in order to produce interpolations of scattered data points (fulfilling the unisolvence condition).

Name	$\phi(r)$	Order
linear	$-r$	1
cubic	r^3	2
thin plate spline	$r^2 \log r$	2
multiquadratic	$-\sqrt{r^2+1}$	1
inverse multiquadratic	$1/\sqrt{r^2+1}$	0
Gaussian	$\exp(-r^2)$	0

Table 1: List of common RBFs

The shape of functions in Table 1 can be modified, without effecting their positive definiteness, with the exception of the linear and cubic splice, which are invariant of scaling. This can be done by introducing a *shape parameter* κ:

$$r \mapsto \phi_\kappa(r) = \phi(\kappa r), \quad \kappa > 0.$$

Considering for instance the Gaussian, changing the shape parameter κ is equivalent to changing the width of the function, as seen in Figure 7. Using a basis function with a large width (small shape parameter) will produce an interpolation that is very smooth but will have difficulties to interpolate fast oscillating data, a small width on the other hand will produce peaks at the data sites and miss the smoothness inbetween. An equivalent way of accomplishing the change of basis function width is to scale the parameter domain, and this is the approach used in this thesis (see Section 3.4.1).

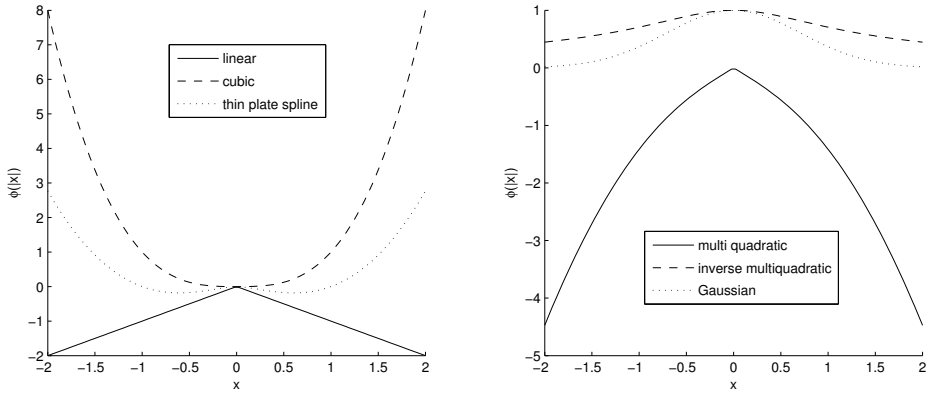


Figure 6: Different RBFs in 1D.

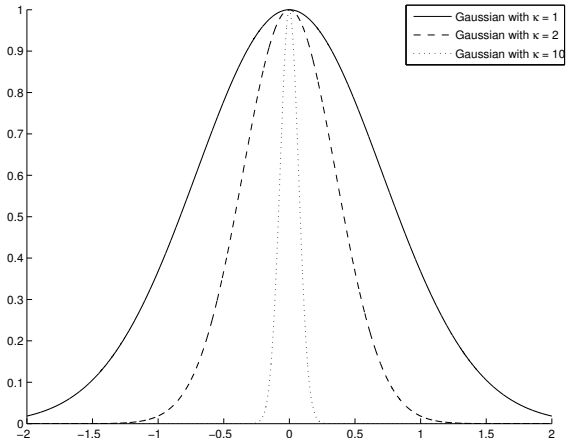


Figure 7: Illustrating the impact of different values of the shape parameter κ for the Gaussian.

If information is known about the objective function, the scaling could be used to improve the interpolation[9].

The main difference between different radial basis functions concerns error estimation, which will be covered in Section 2.3.3. Already in this section a difference concerning the nature of the data points X that are possible to interpolate has been shown: the difference is that, when interpolating using a conditionally positive definite function of order m, the data points X have to be π_{m-1} unisolvent.

2.3.2 A Hilbert space interpretation of interpolation with radial basis functions

In order to discuss error estimation of an interpolation, a Hilbert space interpretation is necessary. This subsection begins by introducing *reproducing kernels* and the spaces generated by them. Further, the positive definite radial basis functions are identified as reproducing kernels. The spaces generated by the kernels, called *native spaces*, are derived and some of their properties are discussed. These spaces are important, since the error estimates in the next subsection assume that the function being interpolated belongs to the native space. Furthermore, a norm in this space

[9] If, for instance, a function $f(x, y)$ varies much more when changing y than x, the domain should be scaled to a rectangle that is longer in the y-direction than in the x-direction.

is introduced in the current subsection, which will be very useful when taking the step from the interpolation to the *approximation* of a function.

The starting point is the definition of the reproducing kernels:

Definition 17 (Reproducing kernel). *Let $\mathcal{F}$ be a real Hilbert space of functions $f : \Omega \to \mathbb{R}$. A function $\Phi : \Omega \times \Omega \to \mathbb{R}$ is called a reproducing kernel for $\mathcal{F}$ if:*

1. $\Phi(\cdot, y) \in \mathcal{F}, \quad \forall y \in \Omega,$
2. $f(y) = \langle f, \Phi(\cdot, y)\rangle_{\mathcal{F}}, \quad \forall f \in \mathcal{F} \quad$ *and all* $y \in \Omega,$

where $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ is the inner product of the Hilbert space $\mathcal{F}$.

The kernel of a Hilbert space is unique. This can be seen by assuming the existence of two kernels Φ_1 and Φ_2. Property 2 above then gives $0 = f(y) - f(y) = \langle f, \Phi_1(\cdot, y) - \Phi_2(\cdot, y)\rangle$ for all $f \in \mathcal{F}$ and $y \in \Omega$. By letting $g = \Phi_1(\cdot, y) - \Phi_2(\cdot, y)$ for a fixed y, the relation $\langle f, g\rangle = 0$ for all $f \in \mathcal{F}$ is obtained. This implies that g is a null element in $\mathcal{F}$. Hence, Φ_1 and Φ_2 must be identical.
Further, there exists a strong connection between the *point evaluating functional* and the reproducing kernel:

Definition 18 (Point evaluating functional). *A functional $\delta_y : \mathcal{F} \to \mathbb{R}$ is called the point evaluating functional for $y \in \Omega$ on $\mathcal{F}$ if $\delta_y(f) = f(y)$ for all $f \in \mathcal{F}$.*

For Hilbert spaces $\mathcal{F}$ containing functions $f : \Omega \to \mathbb{R}$ the following statements are then equivalent:

1. all point evaluating functionals are continuous, i.e., $\delta_y \in \mathcal{F}^*$, $y \in \Omega$;
2. $\mathcal{F}$ has a reproducing kernel,

where $\mathcal{F}^*$ is the dual space to $\mathcal{F}$. To show this, suppose that the point evaluating functionals are continuous. By the Riesz representation theorem, for all $y \in \Omega$ there exists a unique element Φ_y in $\mathcal{F}$ such that $\delta_y(f) = \langle f, \Phi_y\rangle$ for all $f \in \mathcal{F}$. This implies that the function $\Phi(x, y) := \Phi_y(x)$ is the reproducing kernel of $\mathcal{F}$. If on the other hand $\Phi(x, y)$ is the reproducing kernel, then $\delta_y(f) = \langle f, \Phi(\cdot, y)\rangle$ for $y \in \Omega$, and since the inner product is continuous so is δ_y. An example of a space with no reproducing kernel is L^2, since the point evaluating functional δ_x, commonly known as "Dirac's δ-pulse", is not continuous.

The following two properties for reproducing kernels will also prove useful:

$$\begin{aligned} &\Phi(x, y) = \langle \Phi(\cdot, x), \Phi(\cdot, y)\rangle_{\mathcal{F}} = \langle \delta_x, \delta_y\rangle_{\mathcal{F}^*}, \\ &\Phi(x, y) = \Phi(y, x), \qquad \forall x, y \in \Omega. \end{aligned} \tag{12}$$

To show these properties, observe that $\delta_y(f) = \langle f, \Phi(\cdot, y)\rangle$ and consider it as the mapping $F : \mathcal{F}^* \to \mathcal{F}$ obtained by Riesz representation theorem, $F(\delta_y) = \Phi(\cdot, y)$:

$$\langle \delta_x, \delta_y\rangle_{\mathcal{F}^*} = \langle F(\delta_x), F(\delta_y)\rangle_{\mathcal{F}} = \langle \Phi(\cdot, x), \Phi(\cdot, y)\rangle_{\mathcal{F}}.$$

Furthermore

$$\Phi(x, y) = \delta_x(\Phi(\cdot, y)) = \langle \Phi(\cdot, y), \Phi(\cdot, x)\rangle_{\mathcal{F}} = \langle \Phi(\cdot, x), \Phi(\cdot, y)\rangle_{\mathcal{F}}.$$

The following theorem connects the concept of reproducing Hilbert spaces to the previous section by identifying the reproducing kernels as positive definite functions:

Theorem 19. *Suppose $\mathcal{F}$ is a reproducing-kernel Hilbert function space with reproducing kernel $\Phi : \Omega \times \Omega \to \mathbb{R}$, then Φ is positive semidefinite. Φ is positive definite if and only if the point evaluating functionals are linearly independent in $\mathcal{F}^*$.*

Proof. Using the property (12),

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j \Phi(x_i, x_j) = \left\langle \sum_{i=1}^{n} \alpha_i \delta_{x_i}, \sum_{j=1}^{n} \alpha_j \delta_{x_j} \right\rangle_{\mathcal{F}^*} = \left\| \sum_{i=1}^{n} \alpha_i \delta_{x_i} \right\|_{\mathcal{F}^*}^2 \geq 0,$$

the sum will only equal zero if the point evaluating functionals are linearly dependent. □

Hence the reproducing kernel Hilbert function spaces are characterized by (semi) definite kernels. As was mentioned in the previous section, the introduction of radial functions is motivated by their transformation (translation and rotation) invariant properties. In order to investigate this more formally, the following definition is introduced:

Definition 20. *Let $\mathcal{T}$ be a group of transformations $T : \Omega \to \Omega$. $\mathcal{F}$ is invariant under $\mathcal{T}$ if*

1. $f \circ T \in \mathcal{F},\ f \in \mathcal{F}$ *and* $T \in \mathcal{T}$.
2. $\langle f \circ T, g \circ T \rangle_{\mathcal{F}} = \langle f, g \rangle_{\mathcal{F}},\ f,g \in \mathcal{F}$ *and* $T \in \mathcal{T}$.

By using the second property of the above definition and property 2 of the reproducing Hilbert kernels the following can established:

$$f(\boldsymbol{y}) = f \circ T^{-1}(T\boldsymbol{y}) = \langle f \circ T^{-1}, \Phi(\cdot, T\boldsymbol{y}) \rangle_{\mathcal{F}} = \langle f, \Phi(T\cdot, T\boldsymbol{y}) \rangle_{\mathcal{F}}.$$

Using $f(\boldsymbol{y}) = \langle f, \Phi(\cdot, \boldsymbol{y}) \rangle_{\mathcal{F}}$ and the uniqueness of the Hilbert kernel gives $\Phi(\cdot, \boldsymbol{y}) = \Phi(T\cdot, T\boldsymbol{y})$. This proves that the invariance under transformations of Hilbert spaces induces an invariance of their reproducing kernels. Consider a translation $T_{\boldsymbol{\xi}} : \mathbb{R}^d \to \mathbb{R}^d$ and an orthogonal transformation $A_{\boldsymbol{\nu}} : \mathbb{R}^d \to \mathbb{R}^d$:

$$\begin{aligned} T_{\boldsymbol{\xi}}\boldsymbol{x} &= \boldsymbol{x} - \boldsymbol{\xi}, \qquad \boldsymbol{\xi} \in \mathbb{R}^d, \\ A_{\boldsymbol{\nu}}\boldsymbol{x} &= \|\boldsymbol{x}\|_2 \frac{\boldsymbol{\nu}}{\|\boldsymbol{\nu}\|_2}, \qquad \boldsymbol{\nu} \in \mathbb{R}^d, \end{aligned}$$

and the group consisting of all possible transformations above:

$$\mathcal{T}_{TO} = \{T_{\boldsymbol{\xi}} : \boldsymbol{\xi} \in \mathbb{R}^d\} \cup \{A_{\boldsymbol{\nu}} : \boldsymbol{\nu} \in \mathbb{R}^d\}.$$

Assume that $\mathcal{F}$ is invariant under $\mathcal{T}_{TO}$. Choose $\boldsymbol{\xi} = \boldsymbol{x}$ and let $\boldsymbol{\nu}$ be fixed. By using the invariance of the kernel the following is then obtained:

$$\begin{aligned} &\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(T_{\boldsymbol{x}}\boldsymbol{x}, T_{\boldsymbol{x}}\boldsymbol{y}) = \Phi(0, \boldsymbol{y} - \boldsymbol{x}) =: \phi_0(\boldsymbol{y} - \boldsymbol{x}), \\ &\Phi(\boldsymbol{x}, \boldsymbol{y}) = \Phi(A_{\boldsymbol{\nu}}\boldsymbol{x}, A_{\boldsymbol{\nu}}\boldsymbol{y}) = \phi_0(A_{\boldsymbol{\nu}}(\boldsymbol{y} - \boldsymbol{x})) = \phi_0(\|\boldsymbol{y} - \boldsymbol{x}\|_2 \boldsymbol{\nu}/\|\boldsymbol{\nu}\|_2) =: \phi(\|\boldsymbol{y} - \boldsymbol{x}\|_2). \end{aligned}$$

The conclusion is that reproducing Hilbert spaces that are invariant with respect to translation and orthogonal transformation have radial functions as their reproducing kernels. Therefore the use of radial basis functions produces an interpolation belonging to a rotation and translation invariant space. It ensures that these transformations do not produce a function outside of this space (property 1 in Definition 20) and does not change the norm of it (consequence of property 2). Therefore no direction in the parameter space Ω is prioritized (in this sense). This is a desirable property since, in most cases, no information about the parameter space and its effect on the function is known. Hence a further motivation for using radial functions as basis functions for the interpolation is provided.

It has now been established that a rotation and translation invariant function space will have positive definite radial functions as their reproducing kernels. But the question remaining to be answered is what spaces correspond to some of the radial basis functions presented in the previous section, or more generally, given a positive definite symmetric function Φ, which space does it induce? In order to answer this question, the following linear space is defined:

$$F_{\Phi}(\Omega) := \operatorname{span}\{\Phi(\cdot, \boldsymbol{y}) : \boldsymbol{y} \in \Omega\},$$

as well as the bilinear form

$$\left\langle \sum_{j=1}^{n} \alpha_j \Phi(\cdot, \boldsymbol{x}_j), \sum_{i=1}^{m} \beta_i \Phi(\cdot, \boldsymbol{y}_i) \right\rangle_{\Phi} = \sum_{j=1}^{n} \sum_{i=1}^{m} \alpha_j \beta_i \Phi(\boldsymbol{x}_j, \boldsymbol{y}_i). \tag{13}$$

This bilinear form is an inner product on $F_\Phi(\Omega)$, since it is symmetric (as Φ is symmetric) and for arbitrary $f = \sum_{i=1}^{n} \alpha_i \Phi(\cdot, x_i) \not\equiv 0$, the bilinear form is positive (that is $\langle f, f \rangle_\Phi > 0$ which follows from Φ being positive definite). A completion of the linear space $F_\Phi(\Omega)$ with respect to the norm $\|\cdot\|_\Phi = \sqrt{\langle \cdot, \cdot \rangle_\Phi}$ can therefore be made; this completed space is denoted $\mathcal{F}_\Phi$. The elements of this space are abstract entities obtained by completing Cauchy sequences in an infinitely dimensional space. In order to interpret these abstract elements as continuous functions, an extension of the point evaluating functionals from F_Φ, which are $\langle \cdot, \Phi(\cdot, x) \rangle_\Phi$, has to be made. Since the functionals are continuous in F_Φ, their extension to $\mathcal{F}_\Phi$ remains continuous and the following linear operator can be defined:

$$R : \mathcal{F}_\Phi(\Omega) \to C(\Omega), \quad R(f)(x) := \langle f, \Phi(\cdot, x) \rangle_\Phi.$$

Using this, the Hilbert space induced by the positive definite function Φ can in turn be defined as:

Definition 21. *The native space corresponding to the symmetric positive definite function $\Phi : \Omega \times \Omega \to \mathbb{R}$ is defined by*

$$\mathcal{N}_\Phi(\Omega) := R(\mathcal{F}_\Phi(\Omega)),$$

with the inner product

$$\langle f, g \rangle_{\mathcal{N}_\Phi} = \langle R^{-1} f, R^{-1} g \rangle_\Phi.$$

Note that for the native space $\mathcal{N}_\Phi$, where Φ is positive definite, Φ itself is the reproducing kernel. The previous procedure can be extended to conditionally positive definite functions. Assuming a conditionally definite function of order m, consider the following linear space:

$$F_\Phi := \left\{ \sum_{j=1}^{N} \alpha_j \Phi(\cdot, \boldsymbol{x}_j) : N \in \mathbb{N}, \boldsymbol{\alpha} \in \mathbb{R}^N, \quad \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \Omega \text{ and } \sum_{j=1}^{N} \alpha_j p(\boldsymbol{x}_j) = 0, \quad \forall p \in \pi_{m-1} \right\}.$$

The same bilinear form (13) as previously can be used as inner product, where the additional constraint on the coefficients in the definition of F_Φ guarantee that the bilinear form is positive for all nonzero elements of F_Φ. Again a completion with respect to the norm $\|\cdot\|_\Phi$ is made and the completed Hilbert space is denoted by $\mathcal{F}_\Phi$. Unfortunately the construction of an operator $R : \mathcal{F}_\Phi \to C$, connecting the abstract completed space to the continuous function space, cannot be made in the same way as with the positive definite functions. The reason is that $\Phi(\cdot, \boldsymbol{x})$ is not a member of F_Φ (for $\Phi(\cdot, \boldsymbol{x})$ to be a member of F_Φ, there can only exist one coefficient $\alpha_j \neq 0$, but the additional constraint $\alpha_j p(\boldsymbol{x}) = 0$ would not be fulfilled for $p \in \pi_0$). The construction of the R operator is nevertheless still possible, although it becomes more technical (see [Wen05, Ch. 10.3]). Eventually a native space can be defined on the form:

Definition 22. *The native space corresponding to the symmetric conditionally positive definite function $\Phi : \Omega \times \Omega \to \mathbb{R}$ of order m is defined by*

$$\mathcal{N}_\Phi(\Omega) := R(\mathcal{F}_\Phi(\Omega)) + \pi_{m-1},$$

with a semi-inner product

$$\langle f, g \rangle_{\mathcal{N}_\Phi} = \langle R^{-1}(f - \Pi f), R^{-1}(g - \Pi g) \rangle_\Phi,$$

where Π is a projection operator:

$$\Pi : C(\Omega) \to \pi_{m-1}, \qquad \Pi(f) = \sum_{k=1}^{Q} f(\xi_k) p_k.$$

The set $\{\xi_1, \ldots, \xi_Q\}$ is π_{m-1} unisolvent and used to construct the R operator, $Q = \dim \pi_{m-1}$.

Note that the conditionally positive definite function Φ no longer is a reproducing kernel for the native space $\mathcal{N}_\Phi$, and that the bilinear form no longer is an inner product for the space, rather

it is a semi inner product. This is so since the contribution of any polynomial terms has a zero semi norm.

In other words, the space $\mathcal{N}_\Phi$ consists of one polynomial part and one part coming from the basis function, which is in agreement with equation (9). It can also be noted that the semi norm is not altered by changing the polynomial contribution, and all polynomials of order $m-1$ obtain zero semi norm. Furthermore for functions $f, g \in F_\Phi$ the (semi) inner product naturally reduces to

$$\langle f, g \rangle_{\mathcal{N}_\Phi} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \tilde{\alpha}_j \Phi(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_j), \tag{14}$$

which will be used frequently, as the interpolation belongs to F_Φ and the coefficients $\boldsymbol{\alpha}$ are known. As was pointed out previously, the native spaces and their properties are important when dealing with RBF-based interpolations. This is so since all interpolations S arising when solving the problems (7) and (9) lie in the native space, and hence the ability to interpolate a function is dependent on it belonging to this space. Fortunately the native spaces prove to be comparably large, although they vary with the RBF used, for some of the RBFs they may even be identified with previously known spaces (for instance, the native space of thin plate splines can be shown to be the Beppo-Levi space). Further it can be shown that smoothness of the RBF is inherited by the native space, i.e. if $\Phi \in C^{2k}(\Omega \times \Omega)$ then $\mathcal{N}_\Phi(\Omega) \subset C^k(\Omega)$.

Finally the section is sealed by the following theorem, showing that the interpolation S obtained by solving (8) or (11) is the function from $\mathcal{N}_\Phi$ that interpolates the data $\{X, \boldsymbol{f}\}$ with the smallest (semi) norm $|\cdot|_{\mathcal{N}_\Phi}$ (for a proof, see [Wen05, Thm. 13.2]):

Theorem 23. *Suppose $\Phi \in C(\Omega \times \Omega)$ is a conditionally positive kernel of order m. Suppose further that X is π_{m-1} unisolvent. Let S be the interpolant obtained by solving (8) or (11). Then*

$$|S|_{\mathcal{N}_\Phi} = \min\{|\tilde{S}|_{\mathcal{N}_\Phi} : \tilde{S} \in \mathcal{N}_\Phi \text{ with } \tilde{S}(\boldsymbol{x}_j) = f_j,\ j = 1, \ldots, n\}.$$

More loosely, this property guarantees that S is the "simplest possible" function in $\mathcal{N}_\Phi$ that interpolates the data. This will be important when constructing the approximation in Section 2.3.4.

2.3.3 Error estimation of an interpolation

The purpose of this section is to investigate the possible deviation of the interpolation when interpolating a function f at some discrete data points X, that is to find an estimate of $|f(\boldsymbol{x}) - S(\boldsymbol{x})|$. For this purpose, and using notation from Section 2.3.1, the following definition is made:

Definition 24 (Power function). *Given a (conditionally) positive definite function Φ and a data set X, the power function $P_{\Phi,X}$ is defined as:*

$$P_{\Phi,X}(\boldsymbol{x})^2 = \Phi(\boldsymbol{x}, \boldsymbol{x}) - 2\sum_{i=1}^{n} u_i^*(\boldsymbol{x})\Phi(\boldsymbol{x}, \boldsymbol{x}_i) + \sum_{i,j=1}^{n} u_i^*(\boldsymbol{x}) u_j^*(\boldsymbol{x}) \Phi(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

where $\boldsymbol{u}^*(\boldsymbol{x})$ is a partial solution to the linear system

$$\begin{pmatrix} A & P \\ P^{\mathrm{T}} & 0^{Q\times Q} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}^* \\ \boldsymbol{v}^* \end{pmatrix} = \begin{pmatrix} \boldsymbol{V}(\boldsymbol{x}) \\ \boldsymbol{W}(\boldsymbol{x}) \end{pmatrix}, \tag{15}$$

where $\boldsymbol{V}(\boldsymbol{x}) = (\Phi(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \Phi(\boldsymbol{x}, \boldsymbol{x}_n))^{\mathrm{T}}$, $\boldsymbol{W}(\boldsymbol{x}) = (p_1(\boldsymbol{x}), \ldots, p_Q(\boldsymbol{x}))^{\mathrm{T}}$.

The deviation can now be divided into two parts, one depending on the spread of data points and the RBF used to interpolate them and the other depending on the function (or its native space (semi) norm). The following error estimation is proven in [Wen05, Thm. 11.4]:

Theorem 25. *Let $\Omega \subseteq \mathbb{R}^d$ be open, suppose $\Phi \subseteq C^2(\Omega \times \Omega)$ is a conditionally positive definite function on Ω of order m with native space $\mathcal{N}_\Phi$. Suppose further that X is π_{m-1} unisolvent. Denote as S the interpolation of f at data points X using basis function Φ. Then,*

$$|f(\boldsymbol{x}) - S(\boldsymbol{x})| \leq P_{\Phi,X}(\boldsymbol{x}) |f|_{\mathcal{N}_\Phi}, \qquad \boldsymbol{x} \in \Omega. \tag{16}$$

An improvement of this estimate is achieved by the following proposition, which is a new contribution here[10].

Proposition 26. *Let $\Phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a positive definite function with native space $\mathcal{N}_\Phi$. Suppose that S is the interpolation of a function $f \in \mathcal{N}_\Phi$ at points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Denote the native space norm by $\|\cdot\|$, then*

$$|f(\boldsymbol{x}) - S(\boldsymbol{x})| \leq P_{X,\Phi}(\boldsymbol{x})\sqrt{\|f\|^2 - \|S\|^2}, \qquad \boldsymbol{x} \in \Omega.$$

Proof. Let $G = f - S$. Since Φ is positive definite, it is the reproducing kernel of $\mathcal{N}_\Phi$. As $S \in \mathcal{N}_\Phi$ and $f \in \mathcal{N}_\Phi$

$$\langle G, \Phi(\cdot, \boldsymbol{x}_j)\rangle = \langle f, \Phi(\cdot, \boldsymbol{x}_j)\rangle - \langle S, \Phi(\cdot, \boldsymbol{x}_j)\rangle = f(\boldsymbol{x}_j) - S(\boldsymbol{x}_j) = 0, \qquad j = 1, \ldots, n,$$

which implies that $G \in \operatorname{span}\{\Phi(\cdot, \boldsymbol{x}_1), \ldots, \Phi(\cdot, \boldsymbol{x}_n)\}^\perp$. Since $S \in \operatorname{span}\{\Phi(\cdot, \boldsymbol{x}_1) \ldots, \Phi(\cdot, \boldsymbol{x}_n)\}$, using the Pythagorean theorem the relation

$$\|G\| = \sqrt{\|f\|^2 - \|S\|^2} \tag{17}$$

is obtained. Now let $S_{\Phi(\cdot,\boldsymbol{x})}$ be the interpolation of $\Phi(\cdot, \boldsymbol{x})$ at X. Using notation from Definition 24,

$$S_{\Phi(\cdot,\boldsymbol{x})}(\boldsymbol{y}) = \sum_{j=1}^{n} u_j^*(\boldsymbol{x})\Phi(\boldsymbol{y}, \boldsymbol{x}_j),$$

since the solution of the linear system (15) for a positive definite kernel gives the coefficients of the above interpolation. Now consider that:

$$\langle G, \Phi(\cdot, \boldsymbol{x}) - S_{\Phi(\cdot,\boldsymbol{x})}\rangle = \langle G, \Phi(\cdot, \boldsymbol{x})\rangle - \sum_{j=1}^{n} u_j^*(\boldsymbol{x})\langle G, \Phi(\cdot, \boldsymbol{x}_j)\rangle = G(\boldsymbol{x}), \tag{18}$$

and further that

$$\begin{aligned}
\|\Phi(\cdot, \boldsymbol{x}) - S_{\Phi(\cdot,\boldsymbol{x})}\|^2 &= \left\langle \Phi(\cdot, \boldsymbol{x}) - \sum_{j=1}^{n} u_j^*(\boldsymbol{x})\Phi(\cdot, \boldsymbol{x}), \Phi(\cdot, \boldsymbol{x}) - \sum_{j=1}^{n} u_j^*(\boldsymbol{x})\Phi(\cdot, \boldsymbol{x}) \right\rangle \\
&= \langle \Phi(\cdot, \boldsymbol{x}), \Phi(\cdot, \boldsymbol{x})\rangle - 2\sum_{j=1}^{n} u_j^*(\boldsymbol{x})\langle \Phi(\cdot, \boldsymbol{x}_j), \Phi(\cdot, \boldsymbol{x})\rangle \\
&\quad + \sum_{i,j=1}^{n} u_i^*(\boldsymbol{x})u_j^*(\boldsymbol{x})\langle \Phi(\cdot, \boldsymbol{x}_i), \Phi(\cdot, \boldsymbol{x}_j)\rangle \\
&= P_{X,\Phi}^2(\boldsymbol{x}),
\end{aligned}$$

where the last equality uses the definition of $P_{X,\Phi}$ and the property of reproducing kernels given by (12). Finally, by using the Cauchy-Schwarz inequality, the above result and equations (18) and (17) the following is obtained:

$$\begin{aligned}
G(\boldsymbol{x}) &= \langle G, \Phi(\cdot, \boldsymbol{x}) - P_{X,\Phi}(\Phi(\cdot, \boldsymbol{x}))\rangle \\
&\leq \|G\|\|\Phi(\cdot, \boldsymbol{x}) - P_{X,\Phi}(\Phi(\cdot, \boldsymbol{x}))\| \\
&= P_{X,\Phi}(\boldsymbol{x})\sqrt{\|f\|^2 - \|S\|^2}.
\end{aligned}$$

□

[10] This is a reformulation of the theorem originally derived in an unpublished paper (A minimax optimization algorithm), 2007, by Stefan Jakobsson.

Name	$B(h)$
linear	$h^{1/2}$
cubic	$h^{3/2}$
thin plate spline	h
multiquadratic	$e^{-c/h}$
inverse multiquadratic	$e^{-c/h}$
Gaussian	$e^{-c\lvert\log(h)\rvert/h}$

Table 2: Error bounds, $B(h)$, on power functions for common RBFs, from [Wen05, Table 11.1].

The extension of this theorem to conditionally positive definite functions Φ is not trivial since Φ is not the reproducing kernel of the native space. It should nevertheless be possible by constructing a kernel for this space (as done in [Wen05, Thm. 10.20]). It is however left as future research.

Proceeding, bounds on the power function can be obtained in terms of the fill distance h, defined as:

$$h = \sup_{\boldsymbol{x}\in\Omega} \min_{\boldsymbol{x}_j\in X} \|\boldsymbol{x} - \boldsymbol{x}_j\|,$$

which can be thought of as the radius of the largest sphere that can be placed in Ω without including any data points X. Table 2 presents a bound of the type $P_{\Phi,X} \leq CB(h)$ on the power function giving $B(h)$ for different RBFs, with a constant C independent of X. A quick glance at this table shows that there is a large difference in convergence rates between the RBFs; the Gaussians have the best asymptotic convergence, whereas the linear RBF show a rather poor behavior. This would of course be a motivation for choosing Gaussians as basis. On the other hand, the error estimates are asymptotic and do not necessarily imply a better interpolation with a finite number of points. Further, as previously discussed, it is crucial that the function f belongs to the native space of the RBF, which for Gaussians is fairly limited. In the algorithm presented in this thesis, the choice of RBF can be any function in Table 1, but during test runs the thin plate spline was used. The motivation for this choice was that thin plate spline combines a good error bound with nice stability properties and a large native space.

2.3.4 Approximation of scattered data

Approximation here refers to the task of finding a continuous function S such that for given data points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and function values $\boldsymbol{f} = (f_1, \ldots, f_n)^T$,

$$S(\boldsymbol{x}_i) = f_i + e_i, \qquad i = 1, \ldots, n, \tag{19}$$

where $\boldsymbol{e}$ are, in some sense, small errors in the data $\boldsymbol{f}$. The reason for introducing these errors is that by doing so a less complex function, $S(\boldsymbol{x})$, can be obtained. The motivation for using an approximation as a surrogate model is that when dealing with noisy functions, the function values $\boldsymbol{f}$ can not be entirely trusted. As was mentioned previously, the underlying "true" function to be optimized is often smooth, whereas an interpolation of the values obtained from simulations, since noise is present, is more bumpy (more about this in Section 3.1.2) than is reasonable.

As stated in Theorem 23 the interpolation obtained by solving equation (11) is the function in the native space that interpolates the data with the least norm. Hence the interpolation problem can be reformulated as:

$$\begin{aligned} \min_{S\in\mathcal{N}_\Phi} \quad & |S|_{\mathcal{N}_\Phi}, \\ \text{s.t.} \quad & S(\boldsymbol{x}_i) = f_i, \qquad i = 1, \ldots, n. \end{aligned}$$

In order to obtain a function with an even smaller norm (that is, even more smooth), the following

approximation problem can be solved instead, with $\eta \in (0,1)$:

$$\begin{aligned} \min_{\substack{S\in\mathcal{N}_\Phi \\ \boldsymbol{e}\in\mathbb{R}^n}} \quad & \eta |S|^2_{\mathcal{N}_\Phi} + (1-\eta)\|\boldsymbol{e}\|^2_{l^2}, \\ \text{s.t.} \quad & S(\boldsymbol{x}_i) = f_i + e_i, \qquad i = 1,\dots,n. \end{aligned} \tag{20}$$

This means that some error is allowed, if it leads to a smaller (semi) norm of S. The parameter η controls the balance between reducing the error and reducing the approximation norm. One extreme case is $\eta \to 1$, where minimizing the norm is prioritized much higher than minimizing the error, leading to an approximation with zero norm (which is a polynomial in the case of conditionally positive definite RBFs). The other extreme is $\eta \to 0$ which in turn prioritizes the error much higher than the norm leading to an interpolation. By choosing η somewhere in between a balance is obtained where some error is allowed, but the main trend of the data points is still present. Figure 8 illustrates different approximations of the same data for different choices of η.

The question still remains how to solve the approximation problem (20), that is, how to obtain the function S. In order to do so, first consider the fact that given an error vector $\boldsymbol{e}$, coefficients for the function $S \in \mathcal{N}_\Phi$ satisfying equation (19) with least norm are obtained by solving:

$$\begin{pmatrix} A & P \\ P^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} + \boldsymbol{e} \\ \mathbf{0} \end{pmatrix}. \tag{21}$$

Hence, $\boldsymbol{\alpha} = B(\boldsymbol{f} + \boldsymbol{e})$ and $\boldsymbol{\beta} = C(\boldsymbol{f} + \boldsymbol{e})$, where $B_{ij} = (\widetilde{A}^{-1})_{ij}$ for $i,j = 1,\dots,n$ and $C_{ij} = (\widetilde{A}^{-1})_{ij}$ for $i = n+1,\dots,n+d$ and $j = 1,\dots,n$. Using equation (14), the relation $|S|^2_{\mathcal{N}_\Phi} = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|) = \boldsymbol{\alpha}^T A \boldsymbol{\alpha}$ is obtained, and the whole problem (20) reduces to a quadratic programming problem:

$$\min_{\boldsymbol{e}\in\mathbb{R}^n} q(\boldsymbol{e}) = \eta(\boldsymbol{f} + \boldsymbol{e})^{\mathrm{T}} B^{\mathrm{T}} A B^{\mathrm{T}} (\boldsymbol{f} + \boldsymbol{e})^{\mathrm{T}} + (1-\eta)\boldsymbol{e}^{\mathrm{T}}\boldsymbol{e}.$$

For functions $g \in C^2$ the convexity of g is equivalent to it having a positive semidefinite Hessian (see [AEP05]). Since A is a symmetric matrix,

$$\nabla^2 q(\boldsymbol{e}) = 2\eta B^{\mathrm{T}} A B^{\mathrm{T}} + 2(1-\eta) I^{n\times n}. \tag{22}$$

Next the positive definiteness of this Hessian is proven. Let $\boldsymbol{x} \in \mathbb{R}^n$ be arbitrary, by using the definitions of B and C,

$$\begin{pmatrix} A & P \\ P^{\mathrm{T}} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \boldsymbol{x} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} B\boldsymbol{x} \\ C\boldsymbol{x} \end{pmatrix}.$$

Multiplying this relation by $\begin{pmatrix} A & P \\ P^{\mathrm{T}} & \mathbf{0} \end{pmatrix}$ from the left the following is obtained:

$$\begin{pmatrix} \boldsymbol{x} \\ \mathbf{0} \end{pmatrix} = \begin{pmatrix} A & P \\ P^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} B\boldsymbol{x} \\ C\boldsymbol{x} \end{pmatrix}.$$

Considering the lower row of this equation yields:

$$P^{\mathrm{T}} B\boldsymbol{x} = \mathbf{0}.$$

By letting $\boldsymbol{y} = B\boldsymbol{x}$, and because $P^{\mathrm{T}}\boldsymbol{y} = \mathbf{0}$ and Φ is conditionally positive definite, this implies:

$$\boldsymbol{x}^{\mathrm{T}} B^{\mathrm{T}} A B \boldsymbol{x} = \boldsymbol{y}^{\mathrm{T}} A \boldsymbol{y} > 0, \qquad \forall \boldsymbol{x} \in \mathbb{R}^n \setminus \{\mathbf{0}^n\}.$$

This proves that $B^{\mathrm{T}} A B$ is a positive definite matrix, and hence the Hessian (22) is positive definite, which in turn implies that q is a convex function. For convex functions in unconstrained domains the minimum is given by the solution to the linear equation

$$\nabla q(\boldsymbol{e}) = \mathbf{0}^n.$$

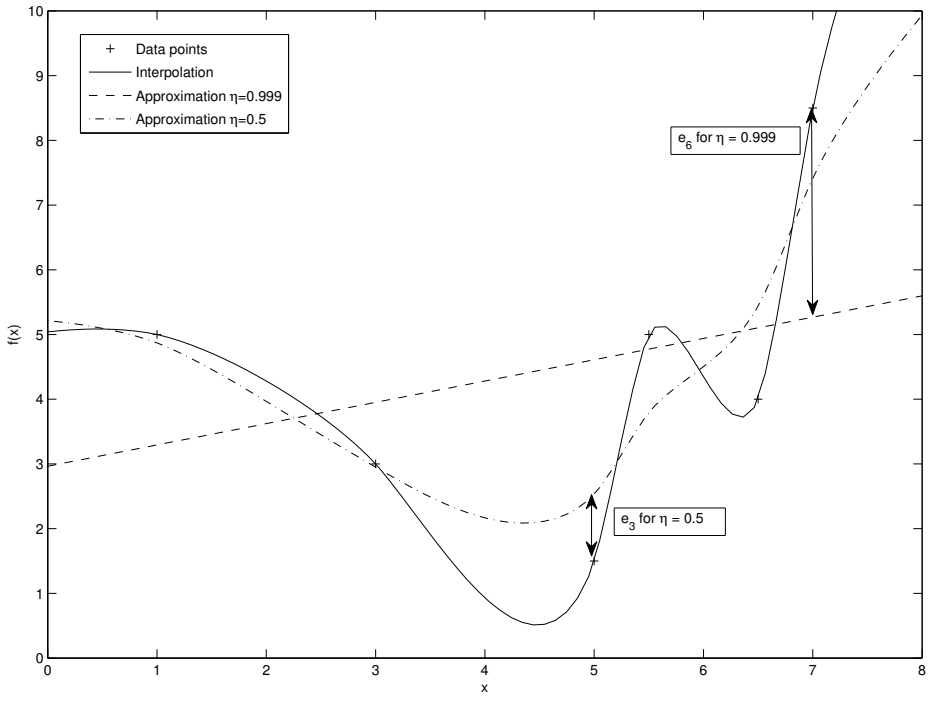


Figure 8: A comparison of an interpolation and two different approximations given the same data.

Inserting, differentiating, and using the symmetry of A gives:

$$\eta(2B^{\mathrm{T}}AB\boldsymbol{e} + 2B^{\mathrm{T}}AB\boldsymbol{f}) + 2(1-\eta)\boldsymbol{e} = \boldsymbol{0}^n,$$

and the vector $\boldsymbol{e}$ is obtained by solving the linear system:

$$\left(\frac{\eta - 1}{\eta} I^{n\times n} - B^{\mathrm{T}}AB\right)\boldsymbol{e} = B^{\mathrm{T}}AB\boldsymbol{f}. \tag{23}$$

The approximation function S can now be obtained by solving the linear system (11) using the errors $\boldsymbol{e}$.

2.4 Strategies for selecting new points to evaluate

This subsection aims to describe some of the strategies for selecting new points in the RSM-based algorithms presented in Section 1.2. This is important for understanding the inner workings of the new algorithm to be introduced presently, and the strategies presented represents different views of and solutions to the problem of achieving a good balance between local and global search.

2.4.1 Gutmann's strategy

As stated in Section 1.2, Gutmann uses RBFs as basis functions for the interpolation in [Gut01]. His strategy for selecting new points relies on one basic assumption about the function being explored and the interpolation of the function. The assumption is this: *The more smooth an interpolation, the more likely it is to accurately depict the original function*. Given a set of points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \in \Omega$ and correspondingly a set of evaluated values $(f_1, \ldots, f_n)^{\mathrm{T}}$, and an RBF-based interpolation S

interpolating the values, the strategy is as follows. Given a *target value* f^*, select the point $\boldsymbol{x}_{n+1} \in \Omega \setminus \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ which maximizes the smoothness (or equivalently minimizes the "bumpiness") of the interpolation. For RBF-based interpolations, the semi norm of the interpolation is a suitable measure of bumpiness.

The target value must lie in $f^* \in [-\infty, \min_{\boldsymbol{y}\in\Omega} S(\boldsymbol{y})]$. Choosing $f^* = \min_{\boldsymbol{y}\in\Omega} S(\boldsymbol{y})$ means selecting the global minimizer of the (current) interpolation, corresponding to trusting the model's accuracy. This choice is only admissible if none of the $\boldsymbol{x}_i$, $i = 1, \ldots, n$, is the global minimizer, because all points in an interpolation must be unique (it makes no sense evaluating the same point twice). Choosing $f^* = -\infty$ would approximately correspond to choosing the point with maximum minimum distance to other points, corresponding to decreasing the uncertainty in previously uncharted territories of the parameter space. Figure 9 illustrates the strategy used by Gutmann. Figure 9 (A) represents using $f^* = \min_{\boldsymbol{y}\in\Omega} S(\boldsymbol{y})$ as target value, and the corresponding point that is chosen as next point. Figure 9 (B) represents setting $f^* = -\infty$ and the corresponding global search that is the consequence. The next point chosen is the one that will decrease the "bumpiness" the most, and this point lies as far away for any other points as possible in this case.

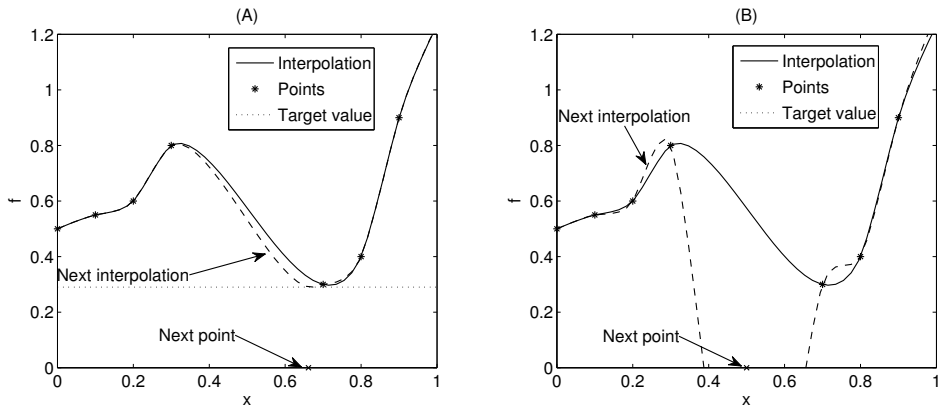


Figure 9: Illustration of Gutmann's strategy for selecting new points in the two extremes of the balance, local and global. The function values were chosen arbitrarily.

During the course of iterations, the algorithm performs a cycle of length 5 in which the target value is varied between these two extremes to get the desired balance between local and global search. During experiments with this algorithm, some potentially unwanted behaviors have been observed:

- The algorithm places a strong emphasis (chooses many points) at the box boundaries of the domain.
- The algorithm tends to cluster points around what is suspected to be a promising basin of low interpolation values.

The first point may be explained by the fact that it is easy to minimize the bumpiness of an interpolation by selecting points at the boundaries. The second point may be due to the cycle not presenting a smooth enough transition between local and global search.

2.4.2 Jones' strategy

The approach in [JSW98] is to model the objective function with stochastic processes, an approached called "kriging" in mathematical geology literature. This summary does not go into much detail about the stochastic machinery involved, but rather presents the results of it in order to give

the reader some idea of what is going on. In essence, an interpolation based on kriging basis functions utilizes a predictor to interpolate and extrapolate data between and beyond sampled points. The predictor also comes with a measure of standard deviation, which may be used when estimating the accuracy of certain areas in the interpolation. Basically, the stochastic process model employed may be written as $y(\boldsymbol{x}_i) = \mu + \epsilon(\boldsymbol{x}_i)$, $i = 1, \ldots, n$, where μ is the mean of the stochastic process and $\epsilon(\boldsymbol{x}_i) \sim N(0, \sigma^2)$. The correlation of the errors $\epsilon(\boldsymbol{x}_i)$ is not zero, but depends among other things on the distance between the sampled points. The error term $\epsilon(\boldsymbol{x}_i)$ is a stochastic process, that is, a set of correlated random variables. For a more detailed description of the derivation and interpretation, see [JSW98].

From the model it is possible to derive an unbiased predictor of $y(\boldsymbol{x})$, denoted $\hat{y}(\boldsymbol{x})$. The predictor at a previously evaluated point $\boldsymbol{x}_i$ with corresponding value f_i gives $\hat{y}(\boldsymbol{x}_i) = f_i$, that is the predictor interpolates the data. Furthermore, associated with the predictor is an estimate of the mean squared error, denoted $s^2(\boldsymbol{x})$ and a corresponding root mean squared error $s(\boldsymbol{x}) = \sqrt{s^2(\boldsymbol{x})}$. The root mean squared error is zero at a sampled point, communicating that there is no uncertainty at this point. As previously discussed, the most critical part of any RSM-based algorithm is how to achieve a good balance between local and global search. In this model, sampling where the predictor maximizes the uncertainty would be equivalent to a global search, and sampling at the minimizer of the predictor would yield a local search. A figure of merit balancing these two extremes is needed, one such merit for this particular model is called *expected improvement*. Consider the following case, where $f_{\min} = \min(f_1, \ldots, f_n)$ denotes the current best function value. Now, consider the point $\boldsymbol{x}$ which has not previously been sampled as the next candidate for function evaluation. The value of $y(\boldsymbol{x})$ is unknown and may be treated as the realization of a normally distributed random variable Y with mean μ and standard deviation s taken from above. Then, there is as certain probability that $Y < f_{\min}$ and the improvement at the point $\boldsymbol{x}$ is $I = \max(f_{\min} - Y, 0)$. This expression is random since Y is a random variable, and the expected improvement at the point $\boldsymbol{x}$ is obtained by taking the expected value of the improvement: $\mathrm{E}[I(\boldsymbol{x})] = \mathrm{E}[\max(f_{\min} - Y, 0)]$. The expected improvement is zero at the sampled points and positive in between. The next point for evaluation is selected as the point maximizing the expected improvement. This auxiliary optimization problem is solved by exploiting certain characteristics of the expected improvement, facilitating an approach based on Branch-and-Bound.

3 The optimization algorithm

The algorithm ("qualSolve") presented in this thesis can be characterized as a two-stage approach (using the terminology from [Jon01], see Section 1.2). This means that sampled data is used to create a surrogate model (an approximation or interpolation of the real objective function) and in a second step this model is used to find the best point for the next expensive function evaluation. The motivation for such an approach is that when dealing with noisy functions, the values at each point can not be trusted, hence the approximation contains more information than the set of evaluated points alone. Further, since a two-stage approach separates the choice of surrogate model from the decision of selecting a new point to evaluate, it enables independent approaches to solving each of these problems.

The section begins by showing how the surrogate models are constructed, followed by a discussion on how to select points for an initial surrogate model. Thereafter, the strategy for choosing new points, by minimizing total uncertainty of the surrogate in areas relevant for the optimization, is presented. This is done by introducing the *quality function*, which is an integral of weighted uncertainty over the parameter domain; the next point to evaluate is obtained by maximizing the quality function. The definition of relevance of an area differs between single and multiobjective optimization: in the single objective case, the interest lies in finding low function values; in the multiobjective case, all non dominated points are of interest. To accomodate this, the weight is constructed differently depending on the type of optimization. Two types of transformations are also introduced: domain transformation and transformation of function values. The purpose behind the domain transformation is to scale all parameters equally, whereas the transformation of function

values is necessary in order to obtain a satisfactory surrogate model for functions that have a very large span. The section is concluded by a summary of the algorithm in pseudocode.

3.1 Creating a surrogate model of the objective function

3.1.1 Interpolation as a surrogate model

If no noise is present in the objective function, an interpolation is the best choice for a surrogate model. For this purpose, an RBF-based interpolation is created using one of the RBFs from Table 1. The construction of the interpolation follows the discussion in Section 2.3. In order to handle RBFs of order up to 2, and since all conditionally positive definite functions of order l are also conditionally positive definite of order $m > l$, the choice made was to use 1:st degree polynomials. Hence the interpolation problem solved can be formulated as follows: given n points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ in a space $\Omega \subset \mathbb{R}^d$, such that $n \geq d+1$, that are π_1 unisolvent (i.e. they do not belong to a common hyperplane) and their function values $(f_1, \ldots, f_n)^{\mathrm{T}} = \boldsymbol{f}$, the interpolation is found by solving the linear system

$$\begin{pmatrix} A & P \\ P^T & \mathbf{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \mathbf{0} \end{pmatrix}, \tag{24}$$

where $A_{ij} = \phi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|)$ and $P_{k\cdot} = (1, x_{1,k}, \ldots, x_{j,k})$, $k = 1, \ldots, d$. The conditions on the points together with the conditionally positive definiteness of the RBFs guarantee a unique solution to (24). The explicit form of the interpolation is:

$$S(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) + \beta_1 + \sum_{j=1}^{d} \beta_{j+1} x_j \tag{25}$$

This is the surrogate model to be used in the next step of the algorithm, which is to decide where the next function evaluation is going to take place.

3.1.2 Approximation as a surrogate model

When dealing with noisy functions, i.e., functions whose response contain some random error, interpolation may lead to a strongly oscillating surrogate model that corresponds poorly to the true objective function (see Figure 10). In order to prevent this type of behavior an approximation is used, thus allowing for a moderate amount of deviation from the sampled function values in exchange for a more smooth function. The assumption is that the "bumpy" behavior is the result of random noise and not the function itself. As was discussed in Section 2.3.4, the approximation is the solution to the following problem:

$$\begin{aligned} \min_{\substack{S \in \mathcal{N}_\Phi \\ \boldsymbol{e} \in \mathbb{R}^n}} \quad & \eta |S|^2 + (1-\eta)\|\boldsymbol{e}\|^2, \\ \text{s.t.} \quad & S(\boldsymbol{x}_i) = f(\boldsymbol{x}_i) + e_i, \qquad i = 1, \ldots, n. \end{aligned}$$

The optimal errors are obtained by solving the linear system (23), using the same set of RBFs and a polynomial space as in the preceding section. Given that the data points X are a π_1 unisolvent set, the problem has a unique solution. The surrogate model can then be obtained by interpolating the data values $\boldsymbol{f} + \boldsymbol{e}$, giving a function on the form (25).

In Figure 11 a surrogate model is constructed for the same set of points as in Figure 10 but by using the approximative method. It is clear that the approximation resembles the "true"[11] function better than the interpolation does. Furthermore, locating the minimum of the approximation is much easier than doing so for the interpolation, and it corresponds better to the "true" function minimum. The issue becomes even more important when using several surrogate models for multiobjective optimization, as a Pareto front of bumpy functions becomes very discontinuous. This

[11] "True" function denotes the hypothetical function that would be obtained if no noise was present.

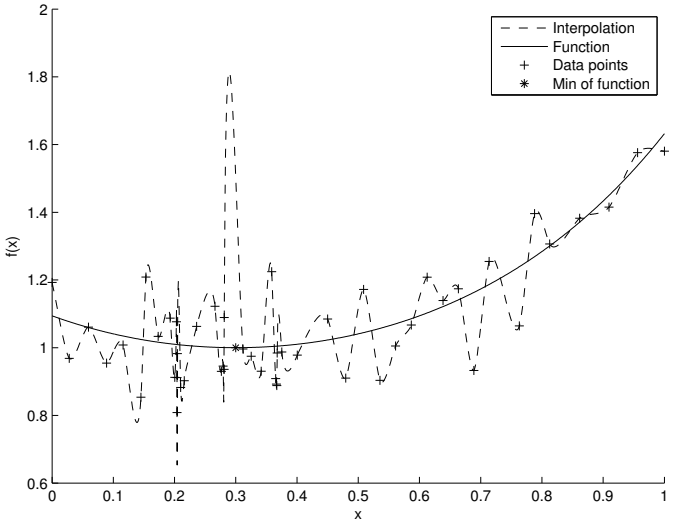


Figure 10: Interpolation of a noisy function.

makes the location of such a front difficult to find and even if this problem is solved, it would still correspond poorly to the "true" front. The "true" functions would be more smooth and have a correspondingly smooth Pareto front. Hence, the use of an approximation instead of an interpolation can be motivated both by obtaining a more correct and a simpler representation of the "true" problem.

The price for using this method to obtain a better surrogate model is the introduction of a new parameter that has to be chosen with care: the approximation parameter η. As was discussed in Section 2.3.4, this parameter controls the balance between minimizing the error and obtaining a smooth approximation, the setting of which depends on the amount of noise expected in the objective function. In order to handle cases where the amount of noise in the problem is unknown, a method called *cross-validation* is utilized; it attempts to give an unbiased estimate of η from the data. The method is described next.

3.1.3 Estimating the approximation parameter

Using the approximation as described in the previous section introduces a parameter η that needs to be estimated. The parameter can be chosen by plotting the surrogate in some subspace of the parameter space or by investigating the maximal or standard deviation of the error for different choices of η. This, however, requires some initial data to work with and knowledge about what to expect in terms of smoothness or noise magnitude. An alternative method, that automates the choice of η and does not have the these requirements, is an estimation procedure called cross-validation. It is a common method for parameter estimation and has its roots in statistics, see e.g. [Koh99]. The implementation in this thesis uses the so-called Leave-one-out cross-validation, the idea of which is the following. Given a set of points and values, $(\boldsymbol{x}_i, f_i)$, $i = 1, \ldots, n$, the best choice of η should be that which allows the approximation to predict the value of a missing point with as small an error as possible. This is done by removing one of the data points $\boldsymbol{x}_i$, using η to construct a surrogate model $S_{\eta\{X \setminus x_i\}}$ of all the remaining points, and evaluating it at the removed point. Formally, the objective is to

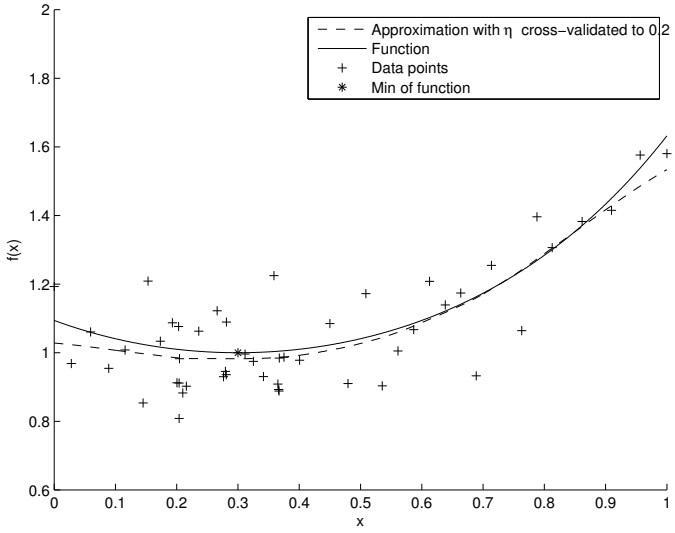


Figure 11: Approximation of a noisy function.

$$\min_{\eta \in (0,1)} \quad \sum_{i=1}^{n} (S_{\eta\{X \setminus x_i\}}(\boldsymbol{x}_i) - f_i)^2, \tag{26}$$

where $f_i = f(\boldsymbol{x}_i) + \varepsilon_i$, f is the "true" objective function and ε_i the noise, assumed to be independent and identical in distribution. Figure 12 illustrates the approach by removing one point and using the others to create a surrogate, and doing so for two different points. In Figure 12 (A) the surrogate with $\eta = 0.1$ produces the best prediction of the removed point, whereas in Figure 12 (B) both $\eta = 0.5$ and $\eta = 0.1$ are equally good. The assumption behind this approach is that a surrogate with an η that can perform good predictions for previously chosen points will also provide good predictions for yet unsampled points. Since the error is assumed to be a random and independent quantity, it can not be predicted and hence the surrogate $S_{\eta\{X \setminus x_i\}}$, where η has been obtained by cross-validation, should at best be able to predict the "true" function f without the noise. Furthermore, if an estimation of the standard deviation or a bound on the error is known, this can be used to assist the cross-validation (see the implementation details in Section 4.1.2).

3.2 Choosing initial points

In order to be able to solve the approximation or interpolation problem uniquely, the conditions in Proposition 14 have to be satisfied. The main issue, aside from the requirement of conditionally positive definite basis functions, is to guarantee points that are unisolvent. Since the decision was made to use conditionally positive functions of maximum order 2 (and treating the ones with lower conditionality as order 2 functions as well; see Section 3.1.1 on surrogate models) the necessary condition is that the data points $X \subset \Omega \subseteq \mathbb{R}^d$ are π_1 unisolvent. The first observation is that the number of points in X must be at least $d + 1$, since given d points a polynomial $p \in \pi_1$ can be constructed that vanishes at all these points. This polynomial is simply the hyperplane that passes all the points in X; hence the condition for X being π_1 unisolvent is that the points in X do not belong to a hyperplane in $\mathbb{R}^d$. Adding more points to an already unisolvent set does not alter its unisolvence. Therefore the fulfillment of the unisolvent condition for the initial points guarantees a unisolvent set for the progression of the algorithm.

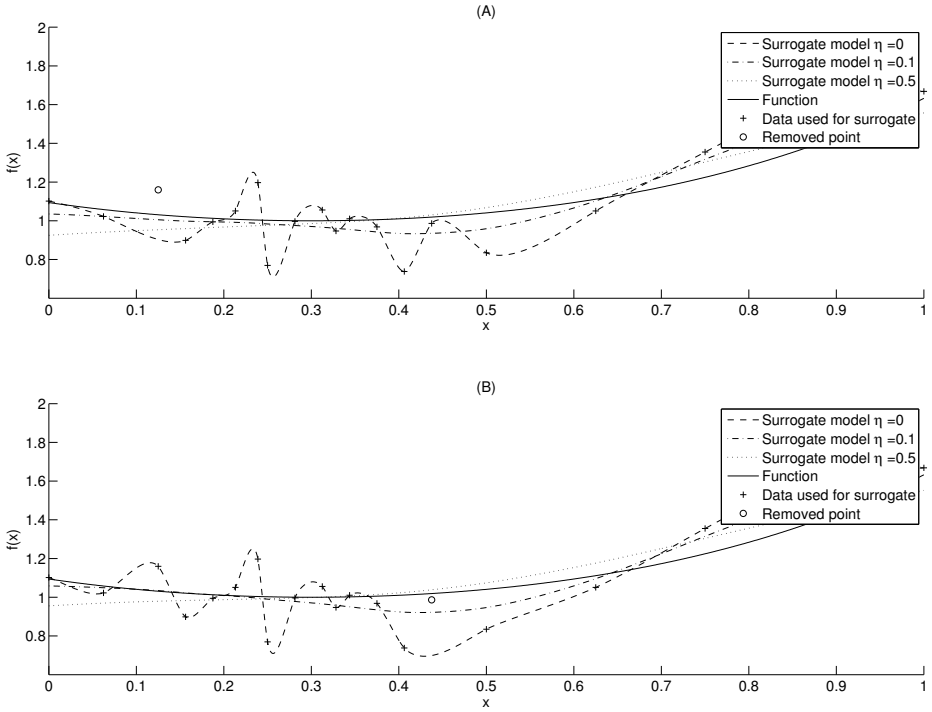


Figure 12: Surrogate models obtained by removing one data point; in (A) the point at 0.12 was remove wheras in (B) it was the point at 0.43. The figure illustrates the ability for surrogate models with different η:s to predict removed points. The data used was produced by adding normal distributed noise to the function.

Besides the necessary demand on X being π_1 unisolvent, good initial points should also provide the maximal amount of information about the function in the domain being investigated, that is they should cover as large a part of the domain as possible. One approach to this end (used in [Gut01]) is to use all 2^d corners of the domain that is defined by the box constraints (2):

$$x_{L,j} \leq x_j \leq x_{U,j}, \qquad j = 1, \ldots, d.$$

Let $\boldsymbol{x}_{C_1,C_2,..,C_d} = (x_{C_1,1}, x_{C_2,2}, \ldots, x_{C_d,d})$ where $C_j = L$ or $C_j = U$, and choose all possible combinations of $C_1, \ldots, C_d$:s (see Figure 13 (D) for an illustration in three dimensions). This guarantees that the points are π_1 unisolvent; further it gives a spread of the points over the entire domain and all its dimensions. The drawback of this strategy is that the number of initial points grows exponentially with the dimension (2^d), which already in six dimensions corresponds to 64 initial points. Since only $d + 1$ initial points are necessary and given that the functions evaluated are expensive, so that the total number of function evaluations for common problems may be around 100-200, as many initial points as 64 seems somewhat high.

An alternative strategy is to choose only $d + 1$ corners of the domain such that the chosen points still are π_1 unisolvent. An example of this would be to choose all the lower corners, i.e. choosing the first point as $C_k = L$, $k \in \{1, \ldots, d\}$ and the remaining d points as $C_j = U$, $C_k = L$ $k \neq j, j = 1, \ldots, d$. This strategy would produce $d+1$ π_1 unisolvent initial points but also prioritizes the lower part of the domain. For instance the distance from an initial point to the point $\boldsymbol{x}_{U,\ldots,U}$ would be considerable.

A completely different initial point strategy that guarantees a large spread and can be used to produce an arbitrary number of initial points is the so called Latin Hypercube Design (LHD, see

[YLS00] for further information). The approach is as follows: in order to obtain n points, construct an $n \times d$ matrix M by assigning to each column a randomly perturbed combination of the sequence $\{1, \ldots, n\}$. Let each row in this matrix correspond to an initial point according to:

$$x_{k,j} = x_{L,j} + M_{k,j}(x_{U,j} - x_{L,j}), \qquad k = 1, \ldots, n, \quad j = 1, \ldots, d.$$

This guarantees an arbitrary number of initial points that are reasonably spread (for instance all constraint boundaries are reached once) and are feasible with respect to the box constraints. Examples of different LHDs are shown in Figure 13. For instance, Figure 13 (A) presents an LHD in two dimensions for three initial points, generated by the matrix

$$M = \begin{pmatrix} 1 & 3 \\ 2 & 1 \\ 3 & 2 \end{pmatrix}.$$

The LHD construction does not by itself satisfy π_1 unisolvence (an outcome could for instance be a matrix with identical columns producing initial points that all lie on a line), and hence the unisolvence condition has to be explicitly checked. Furthermore, additional conditions can be set on the points produced by the LHD. The EGO algorithm (see [JSW98]) and its extension to noisy functions (the SKO algorithm, see [HANZ06]) both use around $11d$ initial points by choosing an LHD that maximizes the spread of the points. For details on the specific LHD implementation used in this thesis, see Section 4.4.

The main difference between LHD and the previously described simpler methods (aside from the ability to generate an arbitrary amount of points) is the randomness introduced. This is a feature that can be problematic, since when solving two equivalent problems the outcome can be highly dependent on a "lucky"[12] choice of initial points. Hence when solving a specific problem or problem class a choice of initial points should be decided upon (either obtained from the LHDs, one of the deterministic strategies or by further knowledge about the objective function) and not altered when solving a similar problem, since this influences the robustness[13] of the algorithm. A similar problem may for instance be two simulation optimizations done using different simulation software or with other parameters or goals for one of the simulations. On the other hand, when dealing with test functions, randomness can be an advantage in order to show that convergence is not dependent on a specific choice of initial points.

Finally, when additional constraints other than the box constraints are present, the "corner points" or points produced by LHD may be infeasible. This can be handled by replacing the infeasible corners by finding feasible points that are well spread in the domain and satisfy the unisolvence condition. Finding such points can, for instance, be done by solving an optimization problem using a software for constrained optimization or, if the domain is not too complicated, simply by random attempts.

3.3 A Strategy for choosing the next evaluation point

All algorithms that handle expensive functions have to keep the number of function evaluations at a minimum. The decision of where to choose the next point is therefore crucial and a considerable amount of CPU time may be used in the process leading to a "good" choice. The key issue for algorithms based on surrogate models is that while there always are features of the underlying objective function that are not captured by the surrogate model, the model still contains all of the information currently possessed. Hence the algorithm has to balance trusting the model (restricting its search to regions where the surrogate model values are low) against exploring areas where no function evaluations have been performed (areas where the surrogate model has more uncertainty and could potentially contain lower function values). This balance is referred to as the balance between local and global search.

[12] Refers to initial points close to the minimum, or initial points yielding a fast convergence that can be dramatically altered by, for instance, rotating the parameter domain.

[13] Refers here to the ability of the algorithm to perform equally well when run on the same or a similar problem.

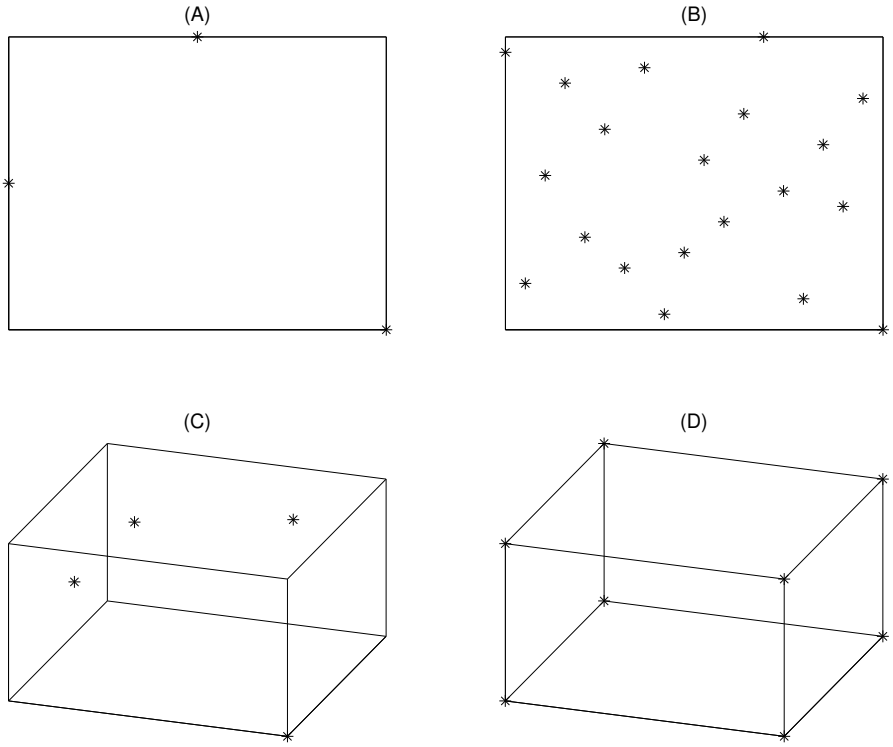


Figure 13: Some initial point configurations, (A) three initial points from LHD in two dimensions, (B) 20 initial points from LHD in two dimensions, (C) four initial points from LHD in three dimensions, (D) initial points generated by choosing all corners in three dimensions.

A common way of achieving this balance is to cycle some parameter that indirectly controls the amount of trust that is given the interpolation (for instance the parameter f^* in [Gut01], see Section 2.4.1). The approach developed here handles this issue in a more direct way. It is done by maximizing an auxiliary function that balances the improvement in certainty gained in an area against the probability that the information obtained there is of interest for the optimization.

As stated in Proposition 26, the error of an interpolant is bounded by:

$$|f(\boldsymbol{x}) - S(\boldsymbol{x})| \leq P_{\Phi,X}(\boldsymbol{x}) \sqrt{\|f\|^2_{\mathcal{N}_\Phi} - \|S\|^2_{\mathcal{N}_\Phi}}. \tag{27}$$

The norm of the function is not known in advance, but, since the power function $P_{\Phi,X}(\boldsymbol{x})$ is the only part of the bound in (27) that depends on $\boldsymbol{x}$, it could be used as a measure of relative uncertainty at different points. As was described in Section 2.3.3, the power function can be obtained by solving a linear system. In order to obtain a simpler measure of relative uncertainty, the bounds of $P_{\Phi,X}(\boldsymbol{x})$ in Table 2 can be considered, given as functions of fill distance h over Ω (see Section 2.3.3). As seen in the table, for most basis functions and specifically the thin plate splines, the power function could be approximated by Ch for some constant C. Inspired by this the following approximation of the power function is presented as a measure of uncertainty:

$$P_{\Phi,X}(\boldsymbol{y}) \approx U_X(\boldsymbol{y}) = \min_{\boldsymbol{x} \in X} \|\boldsymbol{x} - \boldsymbol{y}\|, \tag{28}$$

where X is the set of previously evaluated points, and therefore has zero uncertainty (disregarding the noise which, if present, is addressed during the surrogate model construction phase of the

algorithm). Figure 14 illustrates how the power function $P_{\Phi,X}$ and the uncertainty function U_X predict the uncertainty for an interpolation using the upper bound (27), demonstrating that the approximation performs well for most parts of the domain.

The goal of selecting new points is not to minimize the overall uncertainty, since the certainty of the surrogate in areas uninteresting from an optimization point of view is not important (high function values in the single objective case and far from the Pareto front in the multiobjective case). Therefore a measure of total uncertainty in relevant areas is introduced:

$$\int_\Omega U_X(\boldsymbol{x})\omega(S(\boldsymbol{x}))dV(\boldsymbol{x}),$$

where the weight function $\omega(S(\boldsymbol{x}))$ calculates the relevance of a point by using the surrogate model S (multiple surrogate models in the multiobjective case) in order to make predictions. The choice of the function ω differs between single objective and multiobjective optimization (details on constructing ω will be the subject of the two following subsections), but the purpose is the same: to prioritise areas that are of interest to the optimization.The choice of ω also decides the amount of overall trust that is attached to the surrogate model. When choosing a new point to evaluate, the point is chosen such that the *overall decrease* in uncertainty weighted againts relevance is maximized.

Definition 27 (Quality function). *Let m be the number of objective functions, $\Omega \subseteq \mathbb{R}^d$ the parameter space, $\omega : \mathbb{R}^m \to \mathbb{R}^+$ the weight function that prioritizes regions of interest for the optimization and $U_X : \mathbb{R}^d \to \mathbb{R}^+$ a measure of uncertainty given by (28). The quality function is then given by:*

$$Q(\boldsymbol{y}) = \int_\Omega (U_X(\boldsymbol{x}) - U_{X\cup\boldsymbol{y}}(\boldsymbol{x}))\omega(S(\boldsymbol{x}))dV(\boldsymbol{x}). \tag{29}$$

The choice of which point to evaluate next is obtained by solving the following optimization problem:

$$\arg\max_{\boldsymbol{y}\in\Omega} Q(\boldsymbol{y}). \tag{30}$$

Figure 15 illustrates the process of choosing the next point to evaluate. For illustration purposes the weight function $\omega \equiv 1$ was used, hence treating all parts of the domain as equally important. It is clear that the algorithm chooses points in order to minimize the uncertainty area (i.e., overall uncertainty) remaining after the next point has been evaluated. This prevents the algorithm from overemphasizing edges or any other regions where the *point-wise* largest uncertainty is located.

Finally, since the quality function Q only contains surrogate models, no expensive evaluations are necessary. Therefore the solution to the problem (30) can be obtained by using some external solver for global non-linear optimization (see Section 4.5).

3.3.1 Single objective optimization

The goal of single objective optimization[14] is simply to find the minimal objective function value. Therefore areas where the surrogate model has low function values are of a larger interest than areas where the function values are high. But when treating black box functions no additional information about the function is given; therefore all parts of the parameter domain have to be regarded as potentially containing a minimum. This is the motivation for placing the following demands on the weight function ω:

- No areas are completely disregarded: $\omega(z) > 0, \forall z \in \mathbb{R}$.
- Areas with low surrogate function values are weighted higher than areas with high surrogate function values, i.e.: ω is strictly decreasing.

[14] See Section 2.1 for a brief introduction to single objective optimization.

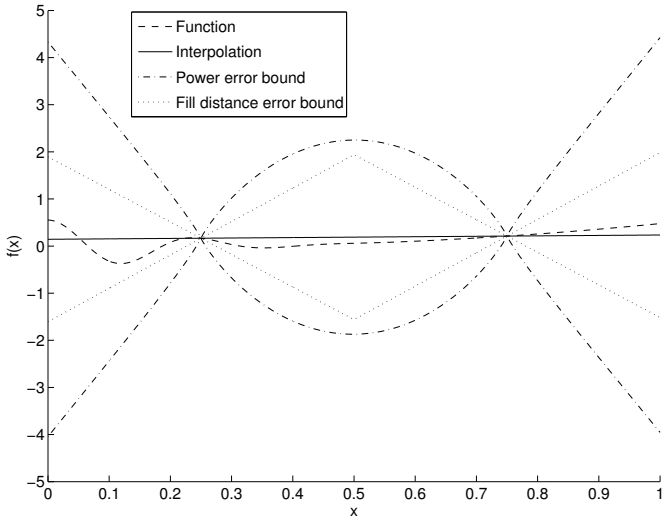


Figure 14: Uncertainty obtained by using the power function (27) or the fill distance ($U_X(x)$ instead of $P_{\Phi,X}(x)$) on a one dimensional function with thin plate splines as basis functions for the interpolation. The norm of the function was obtained by evaluating and interpolating the function at 50 equally spaced points. The function is interpolated at two sample points, $x_1 = 0.25$ and $x_2 = 0.75$.

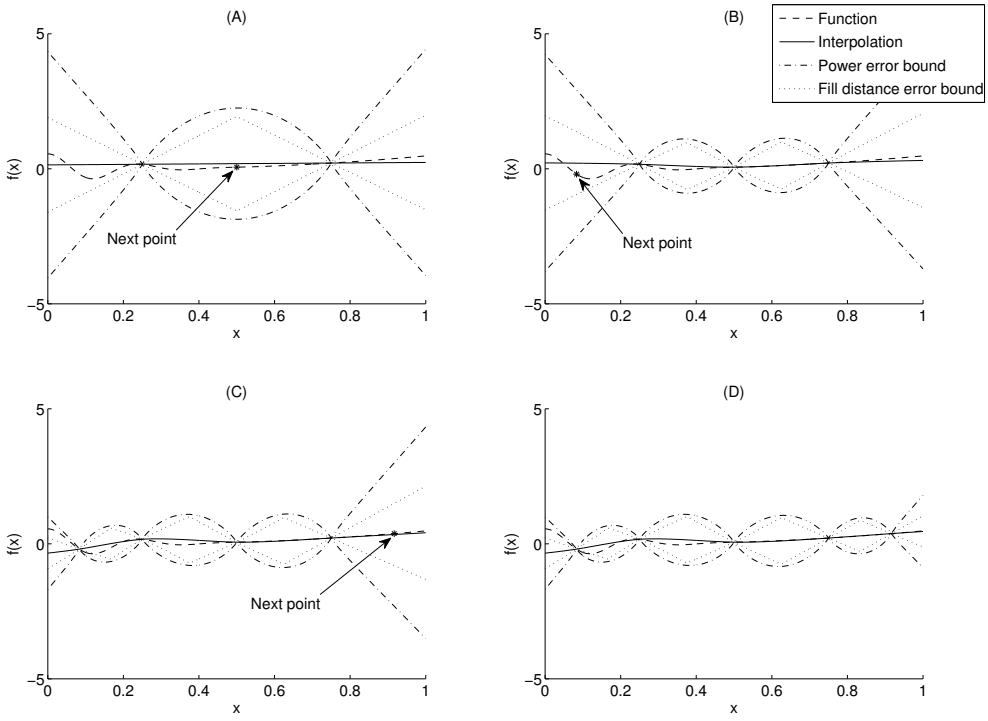


Figure 15: A sequence of points chosen by maximizing the quality function, treating the whole domain as equally important.

A choice of weight function that satisfies these conditions is

$$\omega(z) = \exp\left(-\sigma \frac{z - \min_{\boldsymbol{x}\in\Omega} S_X(\boldsymbol{x})}{\max_{\boldsymbol{x}\in\Omega} S_X(\boldsymbol{x}) - \min_{\boldsymbol{x}\in\Omega} S_X(\boldsymbol{x})}\right). \tag{31}$$

The function is constructed so that $\omega(\min S_X(\boldsymbol{x})) = 1$ and $\omega(\max S_X(\boldsymbol{x})) = e^{-\sigma}$. The factor $\sigma \geq 0$ decides how much areas with lower surrogate values are prioritized over areas with higher values, i.e., controls the balance between trusting the surrogate model and minimizing global uncertainty. The extremes are $\sigma = 0$ which completely ignores the surrogate model and aims to minimize the total uncertainty (i.e. space filling), whereas $\sigma \to \infty$ corresponds to simply choosing $\arg\min S_X(x)$ as the new point to evaluate, hence trusting the model completely.

Although theoretically one choice of this parameter is sufficient to induce both local and global searches, in practice we have, however, experienced that cycling its value provides the best results. In this way an additional mix between local and global searches is added. A way of further improving the choice of the parameter σ is to limit the choices by resetting the cycle to the beginning, should the next candidate point already have some previously evaluated neighbour point at distance closer than δ. It is a useful feature since in many practical situations the interest in densely evaluated points is small (see the end of Section 4.2 for details on this).

3.3.2 Multiobjective optimization

When dealing with multiple objective functions, surrogate models have to be created for each objective function and the weight function becomes a multivariate function of all the surrogate models. As was discussed in Section 2.2, the multiobjective optimization problem,

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x}))^{\mathrm{T}}, \\ \text{s.t.} \quad & \boldsymbol{x} \in \Omega, \end{aligned}$$

can be solved using different approaches, depending on information known about the preferences of the decision maker. Concerning multiobjective optimization, the focus of this thesis is to find the Pareto front Z^* (or Pareto optimal objective set, see Definition 5). Therefore minimizing the uncertainty at points close to (in objective function space) or at the surrogate models' Pareto front is of importance, since these are the most probable areas for points belonging to the Pareto front of the underlying functions. Furthermore, if a point $\boldsymbol{x}_1$ dominates another point $\boldsymbol{x}_2$, implying that the corresponding objective vector $\boldsymbol{z}_1$ is better than $\boldsymbol{z}_2$ (where $\boldsymbol{z} = (S_X^1(\boldsymbol{x}), \ldots, S_X^m(\boldsymbol{x}))^{\mathrm{T}}$, for m surrogate models) in every objective function and hence also closer to the Pareto front, point $\boldsymbol{x}_1$ must be considered more important than point $\boldsymbol{x}_2$. As in the case of single objective optimization, no areas should be completely disregarded in order for the algorithm to converge for all types of functions. Again, let Z denote the space of all feasible objective vectors. The demands on the weight function ω are the following:

- No areas are completely disregarded: $\omega(\boldsymbol{z}) > 0, \forall \boldsymbol{z} \in Z$.
- If $\boldsymbol{x}_1$ dominates $\boldsymbol{x}_2$ in terms of the surrogate functions, then $\omega(\boldsymbol{z_1}) > \omega(\boldsymbol{z_2})$ should hold.

The specific choice of weight function made, that satisfies the above requirements, was:

$$\omega(\boldsymbol{z}) = \exp\left(-\sigma \frac{\mathrm{dist}_{Z_S^*}(\boldsymbol{z})}{\mathrm{dist}_{Z_S^*}(\tilde{\boldsymbol{z}})}\right), \tag{32}$$

where the function $\mathrm{dist}_{Z_S^*}(\boldsymbol{z}) : \mathbb{R}^m \to \mathbb{R}$ is the smallest Euclidean distance between the point $\boldsymbol{z}$ and the Pareto front Z_S^* of the surrogate functions in a domain scaled to unity (see Figure 16):

$$\mathrm{dist}_{Z_S^*}(\boldsymbol{z}) = \min_{\boldsymbol{z}^*\in Z_S^*} \sqrt{\sum_{j=1}^{m} \left(\frac{z_j^* - z_j}{z_{j,\max} - z_{j,\min}}\right)^2},$$

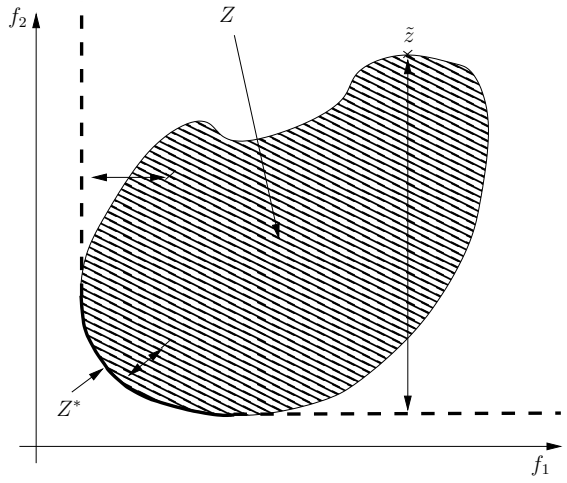


Figure 16: Illustration of the distance function to the Pareto front for different points belonging to the set $Z_{\boldsymbol{S}}$. The dashed line symbolizes the "extended" Pareto front, lying outside of the feasible objective function space. The purpose of the extended front is to be consequent (measure the all distances in the same way) in the measure of distance.

where

$$z_{j,\max} = \max_{\boldsymbol{x}\in\Omega} S_X^j(\boldsymbol{x}), \tag{33}$$

$$z_{j,\min} = \min_{\boldsymbol{x}\in\Omega} S_X^j(\boldsymbol{x}). \tag{34}$$

The objective vector $\tilde{\boldsymbol{z}}$ in (32) defined as:

$$\tilde{\boldsymbol{z}} = \max_{\boldsymbol{z}\in Z} \operatorname{dist}_{Z_{\boldsymbol{S}}^*}(\boldsymbol{z}).$$

Analogous to the case of single objective optimization the weight is constructed such that it varies between 1 and $e^{-\sigma}$. For objective vectors belonging to the Pareto front of the surrogate, $\boldsymbol{z} \in Z_{\boldsymbol{S}}^*$,

$$\omega(\boldsymbol{z}) = 1.$$

The feasible point farthest away, $\tilde{\boldsymbol{z}}$, obtains the weight

$$\omega(\tilde{\boldsymbol{z}}) = e^{-\sigma}.$$

The parameter $\sigma \geq 0$, again as in single objective optimization, balances global and local searches. The extremal choice $\sigma = 0$ corresponds to minimizing total uncertainty without considering the surrogate models (global search) and $\sigma \to \infty$ corresponds to trusting the model completely and minimizing the uncertainty at the Pareto front (local search). The Pareto front of the surrogate models can be found by using any multiobjective solver for non-linear problems (for instance a genetic algorithm, see Section 4.5 for details), since no expensive function evaluations are involved.

To solve the optimization problems (33) and (34) the same external solver for single objective optimization as in the previous section is used. The main difficulty is to construct the distance function $\operatorname{dist}_{Z_{\boldsymbol{S}}^*}$. It cannot be obtained by solving an optimization problem each time, since the integrand in (29) is evaluated thousands of times for every candidate point. Instead the approach is to create an interpolation of the distance function using RBFs and the points on the Pareto front obtained from the multiobjective solver (see Section 4.3 for details).

3.4 Transformations

Transformations enable the algorithm to treat problems that are differently scaled and contain different entities. Two types of transformations are used, transformation of the parameter space Ω and transformation of the objective function f.

3.4.1 Transformation of the parameter space

When dealing with real life optimization problems, the parameters making up the parameter space can well be of different scales or corresponding to incommensurable units. In order to obtain a parameter space that is not affected by the choice of scaling and to avoid comparing entities, such as length and time, the box constraints (2) are used to unit scale the parameter space Ω to $\hat{\Omega}$ according to:

$$T_{\Omega,j}(\boldsymbol{x}) = \frac{x_j - x_{L,j}}{x_{U,j} - x_{L,j}},$$
$$\hat{\Omega} = \{\boldsymbol{x} : \boldsymbol{x} = \boldsymbol{T}_\Omega(\boldsymbol{y}),\ \boldsymbol{y} \in \Omega\}.$$

The set $\hat{\Omega}$ is now a subset of the unit hypercube (or equal to, if only box constraints are present) and ensures that all parameters are treated equally. This linear transformation possesses a well defined inverse:

$$T^{-1}_{\Omega,j}(\boldsymbol{x}) = (x_{U,j} - x_{L,j})y + x_{L,j}.$$

The transformation is introduced at the beginning of the algorithm, which thereafter only handles the transformed parameters from $\hat{\Omega}$, and only transforms back in order to evaluate the objective function.

3.4.2 Transformation of the objective function

The motivation for transforming the objective function values differs from the motivation for transforming the parameter domain. The transformation is performed in order to obtain a better surrogate model for functions that have a very large span compared to the function span locally around the minimum. In Figure 17 an example is shown, where a steep function is interpolated without any transformation of the function values. Although the interpolation seems pretty decent when considering the overall picture, in the closeup it is clear that an oscillating behavior is present that would prevent the accurate location of the minimum. In order to prevent this oscillating behavior when interpolating steep functions, a transformation of the function is used. The transformation $T_{\boldsymbol{f}}$ should fulfill the following conditions:

- The minimization of the transformed function should be equivalent to minimizing the function itself: if $f_1 > f_2$ then $T_{\boldsymbol{f}}(f_1) \geq T_{\boldsymbol{f}}(f_2)$.
- The transformation should prevent overshooting in regions close to minima.

There are different ways to accomodate these demands. One is to simply replace function values above the median of all function values by the median value itself (as done in [Gut01]). This does indeed prevent overshooting in regions close to minimum, but it disregards all information obtained for function values above the median. A logarithmic transformation of the function values has a similar effect, but does not disregard any information completely. The following construction is used:

$$T_{\boldsymbol{f}}(z) = \log\left(z - \left(\min_{i=1,\ldots,n}\{f_i\} - 1\right)\right),$$

where f_i are previously evaluated function values. This function has an inverse:

$$T^{-1}_{\boldsymbol{f}}(z) = e^z + \min_{i=1,\ldots,n}\{f_i\} - 1.$$

By construction,

$$T_{\boldsymbol{f}}\left(\min_{i=1,\ldots,n}\{f_i\}\right) = 0.$$

The derivative of the transformation is:

$$\frac{\partial}{\partial x_j} T_{\boldsymbol{f}}(f(\boldsymbol{x})) = \frac{\frac{\partial}{\partial x_j} f(\boldsymbol{x})}{f(\boldsymbol{x}) - (\min_{i=1,\ldots,n}\{f_i\} - 1)}, \qquad j = 1, \ldots, d, \quad \boldsymbol{x} \in \Omega.$$

This shows that the transformation is strictly increasing, since if $\frac{\partial}{\partial x_j} f(\boldsymbol{x}) > 0$ then

$$\frac{\partial}{\partial x_j} T_{\boldsymbol{f}}(f(\boldsymbol{x})) > 0.$$

The transformation also makes the function less steep (hence preventing overshooting), since

$$\begin{aligned} &\frac{1}{f(\boldsymbol{x}) - (\min_{i=1,\ldots,n}\{f_i\} - 1)} \leq 1, \\ &\qquad \Longrightarrow \quad \frac{\partial}{\partial x_j} T_{\boldsymbol{f}}(f(\boldsymbol{x})) \leq \frac{\partial}{\partial x_j} f(\boldsymbol{x}), \qquad j = 1, \ldots, d, \; \boldsymbol{x} \in \Omega. \end{aligned} \tag{35}$$

Further, the left hand side of (35) decreases as f increases, making the transformation less and less steep as function values increase.

As was mentioned before, the transformation is used to create a better surrogate model for the purpose of optimization. Transformation is not always benificial, but is so when dealing with steep functions. By introducing a surrogate model $\hat{S}$ constructed to interpolate (or approximate) $T_{\boldsymbol{f}}(f_1), \ldots, T_{\boldsymbol{f}}(f_n)$ at X, the surrogate model becomes:

$$S(\boldsymbol{x}) = T_{\boldsymbol{f}}^{-1}(\hat{S}(\boldsymbol{x})). \tag{36}$$

This transformation is applied each time a new point is added to a surrogate function. Figure 17 shows that the interpolation of the transformed values using (36) corresponds better to the function when considering the local picture, as well as in the global picture. As argument in the weight function ω (see Definition 27) $\hat{S}$ and not S from (36) is used. This gives a better "resolution"[15] of the weight function for steep functions.

3.5 Description of the algorithm in pseudocode

The complete single objective algorithm is presented in Algorithm 1, using notation from previous parts of this section. Furthermore the procedure for the cross-validation, the choice of σ, the construction of Q and the software used for solving the comparatively inexpensive optimization of the surrogate and auxiliary functions are described in detail in Section 4. The goal of the single objective algorithm is, given an expensive function f, to find the minimum of it, that is $f_{\min}$, and the point where it is obtained, that is $\boldsymbol{x}_{\min}$. It should be pointed out that, when dealing with deterministic functions, the lowest evaluated function value is returned as it can be given with certainty. When noisy functions are considered, the values themselves can not be trusted, instead the surrogate model is used to provide the minimum.

In Algorithm 2, the algorithm for multiobjective optimization is described. The method for creating the surrogate models is the same as in the single objective case, and is hence not described in detail. The main difference is the distance function, whose construction is described in Section 4.3. The output of the multiobjective algorithm are the Pareto optimal objective set of the final surrogate models ($Z_{\boldsymbol{S}}^*$) and the corresponding Pareto set ($\Omega_{\boldsymbol{S}}^*$).

[15] In Figure 17, using S from (36) as surrogate in ω would give all points $x \in (-1.5, 0.5)$ about the same weight, keeps the algorithm from prioritizing the minimum at -1. When using $\hat{S}$ on the other hand, the total function span will be smaller while the function span around the minimum will not be affected, yielding a larger weight difference between the points in (-1.5 , 0.5).

Input: Objective function f and domain Ω (given by constraints)
Output: $f_{\min} = \min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x})$ and $\boldsymbol{x}_{\min} = \arg\min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x})$
transform box constraints to unit hypercube $\hat{\Omega}$;
generate initial points $\hat{X}$ in $\hat{\Omega}$;
inverse transform $\hat{X}$ to Ω giving X;
evaluate objective function at X to obtain $\boldsymbol{f}$;
let $S(\boldsymbol{x}) = T_{\boldsymbol{f}}^{-1}(\hat{S}(\boldsymbol{x}))$;
while *running algorithm* **do**
  `// create surrogate model`
  if *create interpolation* **then**
    $\hat{\boldsymbol{f}} = T_{\boldsymbol{f}}(\boldsymbol{f})$;
  else *create approximation*
    cross-validate η;
    obtain $\boldsymbol{e}$ by solving linear system (23);
    $\hat{\boldsymbol{f}} = T_{\boldsymbol{f}}(\boldsymbol{f} + \boldsymbol{e})$;
  end
  obtain $\boldsymbol{\alpha}, \boldsymbol{\beta}$ by solving linear system (24);
  $\hat{S}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \phi(\|\boldsymbol{x} - \boldsymbol{x}_i\|) + \beta_1 + \sum_{j=1}^{d} \beta_{j+1} x_j$;
  solve $\min_{\boldsymbol{x}} \hat{S}(\boldsymbol{x})$;
  solve $\max_{\boldsymbol{x}} \hat{S}(\boldsymbol{x})$;
  `// evaluate new point`
  choose σ;
  solve $\max_{\hat{\boldsymbol{y}} \in \hat{\Omega}} Q(\hat{\boldsymbol{y}})$;
  add point $\hat{\boldsymbol{y}}$ to $\hat{X}$;
  $\boldsymbol{y} = \boldsymbol{T}_{\Omega}^{-1}(\hat{\boldsymbol{y}})$;
  evaluate function f at $\boldsymbol{y}$;
  add value to $\boldsymbol{f}$;
  if *stopping criteria met* **then**
    break
  end
end
transform $\hat{X}$ to obtain X ;
if *interpolation* **then**
  $k = \arg\min_{i=1,\ldots,n}\{f_i\}$;
  $f_{\min} = f_k$;
  $\boldsymbol{x}_{\min} = \boldsymbol{x}_k$;
else *approximation*
  $f_{\min} = \min_{\boldsymbol{x} \in \hat{\Omega}} S(\boldsymbol{x})$;
  $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x} \in \hat{\Omega}} S(\boldsymbol{x})$;
  $\boldsymbol{x}_{\min} = \boldsymbol{T}_{\Omega}^{-1}(\boldsymbol{x}^*)$;
end
return $\boldsymbol{x}_{\min}, f_{\min}$

Algorithm 1: Single objective optimization.

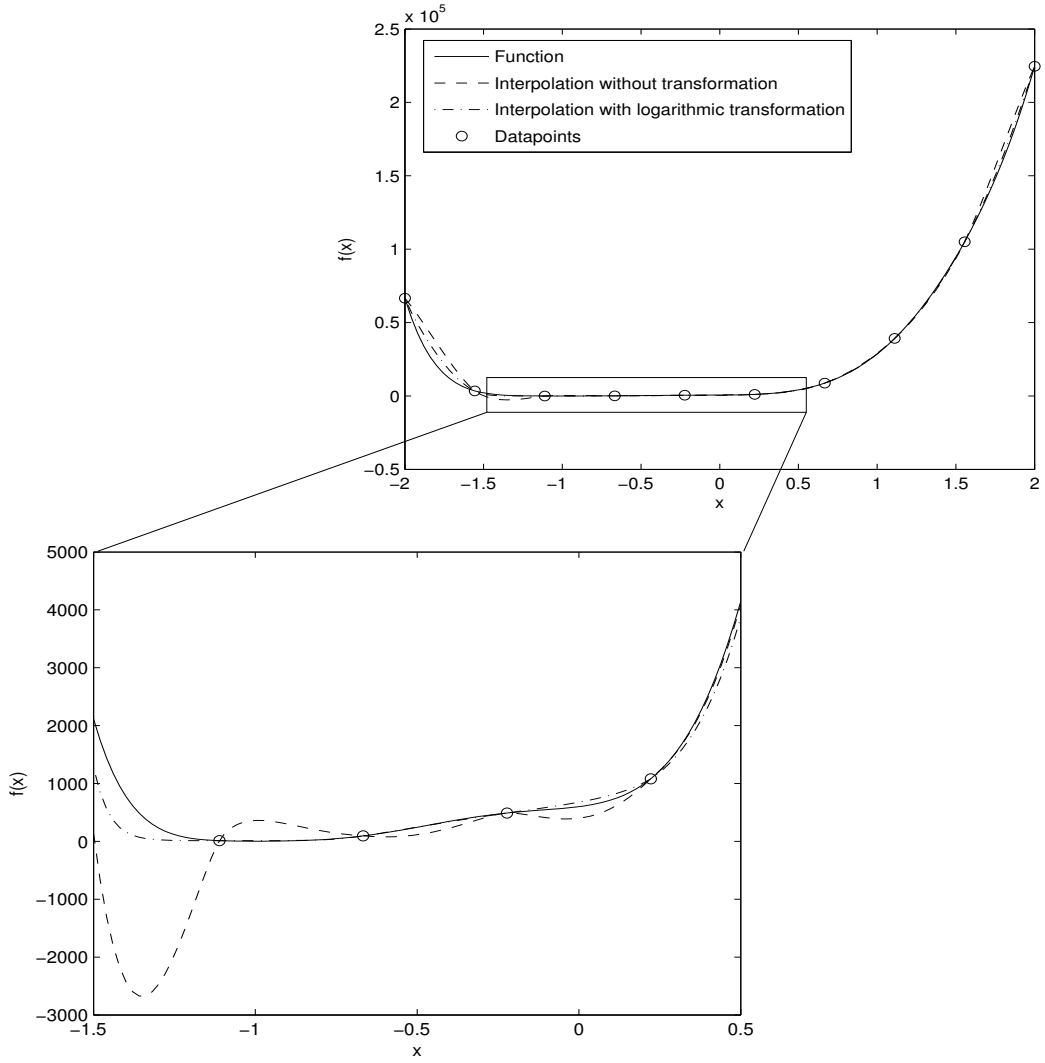


Figure 17: Interpolating a steep function with or without a logarithmic transformation.

4 Implementation

The purpose of this section is to discuss and describe in detail some of the problems arising when implementing the algorithm in practice. Some of it concerns practical solutions to problems made in order to increase computational speed and stability, other parts are numerical fixes that were used in order to prevent computational difficulties. Furthermore strategies for facilitating the setting of various parameters are described. The major part of the algorithm was implemented in MATLAB with exception of the most computationally heavy parts which were written in C and connected to MATLAB through the MEX interface, which allows the user to call C functions from MATLAB. In this way, computationally heavy and specialized routines may be converted to C, gaining efficiency but still retaining the flexibility of MATLAB. The external optimization softwares used were TOMLAB and MultiOb, they are described more thoroughly below.

Input: Objective functions $f_1, \ldots, f_m$ and domain Ω (given by constraints)
Output: Z^*, Ω^* and surrogate models $\boldsymbol{S} = (S_X^1, \ldots, S_X^m)^{\mathrm{T}}$
transform box constraints to unit hypercube $\hat{\Omega}$;
generate initial points $\hat{X}$ in $\hat{\Omega}$;
inverse transform $\hat{X}$ to Ω giving X;
for $j = 1$ **to** m **do**
  evaluate objective function j at X to obtain $\boldsymbol{f}_j$;
end
while *running algorithm* **do**
  for $j = 1$ **to** m **do**
    create surrogate model j;
  end
  `// evaluate new point`
  solve multiobjective problem for surrogate functions;
  construct distance function;
  chose σ;
  solve $\min_{\hat{\boldsymbol{y}} \in \hat{\Omega}} Q(\hat{\boldsymbol{y}})$;
  add $\hat{\boldsymbol{y}}$ point to $\hat{X}$;
  $\boldsymbol{y} = \boldsymbol{T}_{\Omega}^{-1}(\hat{\boldsymbol{y}})$;
  for $j = 1$ **to** m **do**
    evaluate function f_j at $\boldsymbol{y}$;
    add value to $\boldsymbol{f}$;
  end
  if *stopping criteria met* **then**
    break
  end
end
solve multiobjective problem for surrogate functions;
return $Z_{\boldsymbol{S}}^*, \Omega_{\boldsymbol{S}}^*, \boldsymbol{S}$;

Algorithm 2: Multiobjective optimization.

4.1 Constructing the surrogate models

4.1.1 Implementing the surrogate model in C

One of the algorithmic parts that was implemented in C was the surrogate model and its gradient, the reason for which is the following. As was described in Section 3.1.1, the interpolation used for the surrogate model is obtained by solving a linear system and then using the solution to explicitly construct the surrogate through the sum in equation (25). The solution of the linear system is computed in MATLAB and needs only be solved once (or a couple of times when cross-validating) in each iteration. The RBF expansion on the other hand has to be computed thousands of times and, although in itself computationally inexpensive, the frequent use motivates the implementation in C. The implementation of the gradient was motivated by the use of gradient-based local solvers. The implementation is independent of whether an approximation or an interpolation is used as surrogate model, since both result in coefficients that have to be expanded as a sum of RBFs.

4.1.2 Finding the approximation parameter through cross-validation

As was described in Section 3.1.3, the parameter η can be estimated by considering the surrogate's ability to predict already existing points, when excluding them one at a time, and creating a surrogate model of the remaining points. The argument was that errors cannot be predicted, and hence the technique will yield an approximation that predicts the function values well but removes the errors. A problem that was discovered during the development of the algorithm was that the technique often yielded too high an η (that is, the deviation was much larger than the true error of the

function). This seems to depend on the fact, that when trying to predict a function value at a point far away from other points the uncertainty is very high. In practice all choices of η yield large errors, but models that are very insensitive to data (large η) may on average yield better predictions. This is a consequence of the fact that far away from other points, function values are unpredictable and unseparable from noise. This results in choices of η that treat the whole function as noise, and, as a consequence, removes all information. In order to focus on compensating for noise and not for true function variation, an approach was developed that weighs the possibility for a removed point to be predicted by other points. That is, a good approximation is one that can predict points well, where such a prediction is possible; this would be where there are many close neighbouring points. Hence the cross-validation was altered from (26) to

$$\min_{\eta} \sum_{i=1}^{n} W_{X\setminus\{\boldsymbol{x}_i\}}(S_{\eta X\setminus\{\boldsymbol{x}_i\}}(\boldsymbol{x}_i) - f_i)^2,$$

where

$$W_{X\setminus\{\boldsymbol{x}_i\}} = \sum_{j\neq i} \exp\left(-\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{d}\right)^2\right), \tag{37}$$

and d is the cross distance length, a length at which prediction is still possible. The weight $W_{X\setminus\{\boldsymbol{x}_i\}}$ is constructed to appreciate the amount of points at distance around d from $\boldsymbol{x}_i$, since the terms in the sum are about 1 when the distance is close to d, and approaching 0 as the distance is much greater than d. The choice of d is naturally problematic, but since it has a highly indirect impact on the algorithm the choice of it is not particularly crucial; still a proper choice improves on the behavior of the cross-validation immensely.

Finally, if an upper bound on the noise is known, this can be used in order to bound the cross-validation from choosing values η that give rise to error vectors ($\boldsymbol{e}$) exceeding this bound. Since the size of the errors grow with η, the cross-validation can simply be terminated when the optimal errors obtained exceed a given value. This approach was not used on the test functions, but becomes useful when dealing with real problems where information of the error is known or a study of it has been carried through, as a prequel to the optimization process. The whole procedure for cross-validation is described in Algorithm 3; the creation of a surrogate model is identical to that in Algorithm 1.

Input: Vector of η:s to cross-validate ($\boldsymbol{\eta}$), data X and $\boldsymbol{f}$, maximal error e_{max}
Output: η with best prediction
for $i = 1$ **to** *length($\boldsymbol{\eta}$)* **do**
  obtain optimal $\boldsymbol{e}$ by solving linear system (23);
  if $e_{max} \geq \max_{i=1,..,n} |e_i|$ **then**
    `// ` η_i `provides an error within bound`
    add i to D ;
    for $j = 1$ **to** *amount Points in* X **do**
      remove $\boldsymbol{x_j}$ and f_j;
      create surrogate model $S_{\{X\setminus\boldsymbol{x}_j\}}$ of the remaining data using η_i;
      $W_{\{X\setminus\boldsymbol{x}_j\}}j = \sum_{j\neq i} \exp\left(-\left(\frac{\|\boldsymbol{x}_i - \boldsymbol{x}\|}{d}\right)^2\right)$;
    end
    `// error of the prediction`
    $p_i = \sum_{j=1}^{n} W_{\{X\setminus\boldsymbol{x}_j\}}(S_{\{X\setminus\boldsymbol{x}_j\}}(\boldsymbol{x}_j) - f_j)^2$;
  end
end
$k = \arg\min_{i\in D}\{p_i\}$;
return η_k;

Algorithm 3: Computing the approximation parameter using cross-validation.

4.1.3 Approximation of transformed function values

When the objective function values have a large span, a transformation of the values may be beneficial in order to decrease oscillations of the approximation at small function values. In this case, a logarithmic transformation is used:

$$T_{\boldsymbol{f}}(z) = \log\left(z - \left(\min_{i=1,\dots,n}\{f_i\} - 1\right)\right),$$

with the inverse

$$T_{\boldsymbol{f}}^{-1}(z) = e^z + \min_{i=1,\dots,n}\{f_i\} - 1,$$

as defined in Section 3.4.2. When using an approximation as surrogate model together with the transformation of the function values an issue arises when problem (20) is solved: should it be solved for transformed or untransformed function values. On the one hand solving the problem for transformed values is more correct, since an interpolation using them will be constructed and first after that is done, inverse transformation takes place. On the other hand, the purpose of using the approximation is to compensate for noise that does not change in magnitude with function values. When solving the approximation problem (20) for the transformed function, the transformed function errors will be spread about equally and independently of the transformed function values. But as the surrogate $\hat{S}$ is transformed back to become the surrogate for the true function ($S = T_{\boldsymbol{f}}^{-1}(\hat{S})$) the error becomes magnified. This can be easily seen by:

$$\frac{\mathrm{d}S}{\mathrm{d}\hat{S}} = e^{\hat{S}} \implies \Delta S \approx e^{\hat{S}}\Delta\hat{S},$$

which means that the deviation is magnified exponentially. In practice, the result is a surrogate model that deviates a couple orders of magnitude more at higher function values than at lower, and this has no correspondence in the random error that this deviation should compensate for. In order to prevent this behavior, problem (20) is solved for untransformed function values. Thereafter the vector $\boldsymbol{e} + \boldsymbol{f}$ is used for interpolation, i.e., it is transformed and used to obtain coefficients for $\hat{S}$. Hence, indirectly, an untransformed approximation is used in order to calculate the error, and this error is used for a transformed interpolation.

4.2 Constructing the quality function

The implementation of the quality function (see Definition 27) was done in C, since this is where the algorithm spends most of its time. The integral was calculated using trapezoidal numerical integration, with a user-defined partitioning of the domain. To recapitulate, the quality function is defined as:

$$Q(\boldsymbol{y}) = \int_{\Omega} (U_X(\boldsymbol{x}) - U_{X\cup\boldsymbol{y}}(\boldsymbol{x}))\omega(S(\boldsymbol{x}))dV(\boldsymbol{x}),$$

with the measure of uncertainty:

$$U_X(\boldsymbol{y}) = \min_{\boldsymbol{x}\in X} \|\boldsymbol{x} - \boldsymbol{y}\|.$$

4.2.1 Trapezoidal numerical integration of the quality function

Given a function $f : \mathbb{R} \to \mathbb{R}$ and an interval $[x_l, x_u] \in \mathbb{R}$ over which f is to be integrated, the trapezoidal rule works as follows. First, partition the interval into N parts, with $x_l = x_1 < x_2 < \dots < x_{N+1} = x_u$. Secondly, calculate the function values at each of the divisors x_i, $i = 1, \dots, N+1$. The integral is now approximated as

$$\int_{x_l}^{x_u} f(x)dx \approx \sum_{i=1}^{N} \frac{f(x_{i+1}) + f(x_i)}{2}(x_{i+1} - x_i),$$

that is, approximating the function linearly in each interval, and calculating the integral of this approximation. There are more accurate ways of numerically approximating an integral, such as using a quadrature; this is however left for future improvements. In higher dimensions, the trapezoidal approximation is generalized using a recursive method, as described in Algorithm 4. In d dimensions, the integration interval will now be $[\boldsymbol{x}_l, \boldsymbol{x}_u]$, with $\boldsymbol{x}_l, \boldsymbol{x}_u \in \mathbb{R}^d$. The partitioning will be generalized to $x_{l,k} = x_{k,1} < x_{k,2} < \ldots < x_{k,N+1} = x_{u,k}$ for the k:th dimension, $k \in \{1, \ldots, d\}$.

Function: TrapzInt, Input: $\boldsymbol{y} \in \mathbb{R}^k$ $(k = 0, \ldots, d)$, $\boldsymbol{x}_l \in \mathbb{R}^d$, $\boldsymbol{x}_u \in \mathbb{R}^d$, $N \in \mathbb{N}$.
Output: $r \in \mathbb{R}$, approximation of integral
Let empty $\boldsymbol{y}$ imply $k = 0$;
`// Reached bottom of recursion`
if $k = d$ **then**
  $r = f(\boldsymbol{y})$;
  return r;
end
`// Perform trapezoidal integration in dimension` $k+1$
for $i = 1$ **to** N **do**
  `// Current subinterval:` $[x_{k+1,i}, x_{k+1,i+1}]$
  $\tilde{\boldsymbol{y}}_l = (\boldsymbol{y}, x_{k+1,i})$;
  $\tilde{\boldsymbol{y}}_u = (\boldsymbol{y}, x_{k+1,i+1})$;
  `// Recursive callback to dimension` $k+1$
  $r = r + \big(\text{TrapzInt}(\tilde{\boldsymbol{y}}_l, \boldsymbol{x}_l, \boldsymbol{x}_u, N) + \text{TrapzInt}(\tilde{\boldsymbol{y}}_u, \boldsymbol{x}_l, \boldsymbol{x}_u, N)\big)/2$;
end
return r;

Algorithm 4: Function "TrapzInt" for repeated trapezoidal numerical integration of a function f using N partition intervals in each dimension. The function must initially be called with an empty vector $\boldsymbol{y}$.

The fact that the algorithm uses an integral limits its usability to lower dimensions, since the time spent integrating rises exponentially with the dimension of the domain. The C-implementation of the quality function does not explicitly calculate the value of $S_X(\boldsymbol{x})$, but rather calls a MATLAB-function to do so. The approach results in a large amount of callbacks to MATLAB, but also increases the flexibility of the implementation. This will be of use later when considering multiobjective optimization.

4.2.2 Limiting the domain of integration

There is a connected and closed subset $\Omega_{\boldsymbol{y}} \subseteq \Omega$ depending on $\boldsymbol{y}$ outside which the integral vanishes. This is so because the terms approximating the uncertainty, U_X and $U_{X \cup \boldsymbol{y}}$, will be equal whenever $\boldsymbol{y}$ is not the closest point to the current value of the integration variable $\boldsymbol{x}$. In other words, whenever $\|\boldsymbol{x} - \boldsymbol{y}\| \geq \|\boldsymbol{x} - \boldsymbol{w}\|$ for $\boldsymbol{w} \in X$, $U_{X \cup \boldsymbol{y}}(\boldsymbol{x})$ simplifies to:

$$U_{X \cup \boldsymbol{y}}(\boldsymbol{x}) = U_X(\boldsymbol{x}).$$

The subset may consequently be defined as

$$\Omega_{\boldsymbol{y}} = \big\{\boldsymbol{x} \in \Omega : \|\boldsymbol{x} - \boldsymbol{y}\| \leq \min_{\boldsymbol{w} \in X} \|\boldsymbol{x} - \boldsymbol{w}\|\big\}.$$

When inside $\Omega_{\boldsymbol{y}}$, the terms are reduced to $(U_X(\boldsymbol{x}) - U_{X \cup \boldsymbol{y}}(\boldsymbol{x})) = \min_{\boldsymbol{w} \in X} \|\boldsymbol{w} - \boldsymbol{x}\| - \|\boldsymbol{x} - \boldsymbol{y}\|$. As the integral vanishes outside $\Omega_{\boldsymbol{y}}$, the integral only needs to be approximated inside the subset, potentially decreasing the computational time required. However, the definition of $\Omega_{\boldsymbol{y}}$ is not constructive, in order to use it an explicit expression is needed. The subset is defined by a polytope, see Figure 18, and as such can be described using an expression of the form $A\boldsymbol{x} \leq \boldsymbol{b}$, where

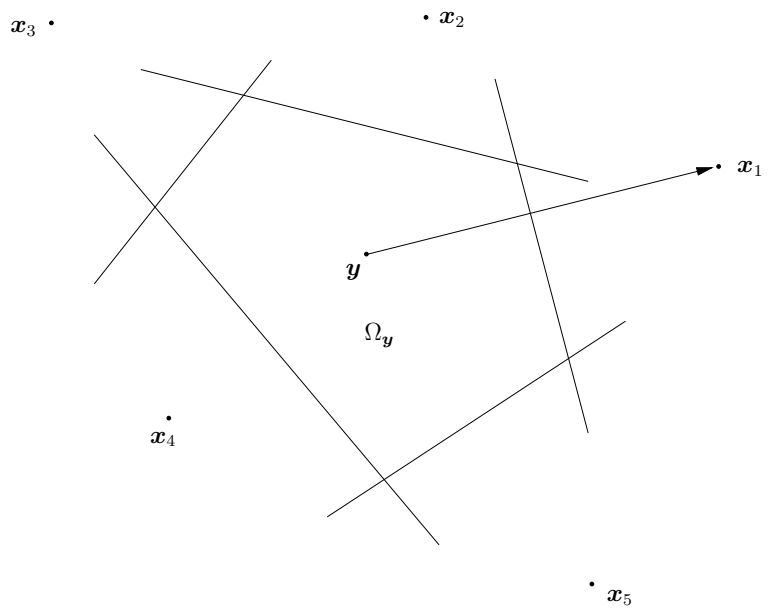


Figure 18: Example of the subset $\Omega_{\boldsymbol{y}} \subseteq \Omega$ defined by a piecewise linear curve.

$A \in \mathbb{R}^{n \times d}$ and $\boldsymbol{b} \in \mathbb{R}^d$. Qualitatively, the domain is constructed by finding the hyperplane orthogonal to the vector $\boldsymbol{x}_i - \boldsymbol{y}$ intersecting it halfway along the vector, and doing so for $i = 1, \ldots, n$. These hyperplanes together with the box constraints will then define a bounded polyhedron.

Formally, given a set of evaluated points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ and a point $\boldsymbol{y}$ for which to calculate $Q(\boldsymbol{y})$, the subset $\Omega_{\boldsymbol{y}}$ is constructed as:

$$\begin{aligned} A_{\cdot j} &= (\boldsymbol{x}_j - \boldsymbol{y})^{\mathrm{T}}, \qquad j = 1, \ldots, n, \\ b_j &= (\boldsymbol{x}_j - \boldsymbol{y})^{\mathrm{T}} \frac{(\boldsymbol{x}_j + \boldsymbol{y})}{2}, \\ \Omega_{\boldsymbol{y}} &= \{\boldsymbol{x} : A\boldsymbol{x} \leq \boldsymbol{b}, \quad \boldsymbol{x} \in \Omega\}. \end{aligned}$$

The expression for $Q(\boldsymbol{y})$ may now be equivalently stated as:

$$Q(\boldsymbol{y}) = \int_{\Omega_{\boldsymbol{y}}} (U_X(\boldsymbol{x}) - \|\boldsymbol{x} - \boldsymbol{y}\|)\, \omega(S_X(\boldsymbol{x}))\, dV(\boldsymbol{x}), \tag{38}$$

As the number of points in X increases, the subset generally decreases in size and, as a result, the approximation of the integral actually becomes faster when more points are added. However, the domain of integration must be rectangular for a trapezoidal approximation to be used, and $\Omega_{\boldsymbol{y}}$ is not rectangular in general. For this reason, the subset was extended somewhat to form a rectangular domain; which unfortunately implies a slight increase in computational time, since an area larger than needed is integrated over.

To be able to find the smallest possible rectangle $\Omega_{\boldsymbol{y}R}$ so that $\Omega_{\boldsymbol{y}} \subseteq \Omega_{\boldsymbol{y}R}$, the extremal vertices of the polytope in the (positive and negative) directions of the Cartesian basis vectors must be identified. This can be formulated as a linear optimization problem, and solved using an ordinary LP-solver:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & (0, \ldots, 0, a, 0, \ldots, 0)\boldsymbol{x}, \\ \text{s.t.} \quad & A\boldsymbol{x} \leq \boldsymbol{b}, \end{aligned}$$

where $a \in \{-1, 1\}$ at the jth position, $j = 1, \ldots, d$, where d is the number of dimensions. The sign of a denotes which "end" of the interval in the specific dimension to locate. In total there are $2d$ LP-problems to be solved for each $\boldsymbol{y}$ considered, this may seem a lot but the resulting gain in computational efficiency outweighs the time needed to solve these LP-problems.

For the external optimization routine to find the optimum of Q efficiently the gradient $\nabla Q(\boldsymbol{y})$ is needed. The derivation of the gradient is quite straight-forward as most terms of the integral are independent of $\boldsymbol{y}$; the reduced domain $\Omega_{\boldsymbol{y}}$ however is not. In order to handle the problem of a $\boldsymbol{y}$-dependent domain, the following function is defined:

$$f(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} (U_X(\boldsymbol{x}) - \|\boldsymbol{x} - \boldsymbol{y}\|)\omega(S_X(\boldsymbol{x})) & \boldsymbol{x} \in \Omega_{\boldsymbol{y}}, \\ 0 & \boldsymbol{x} \notin \Omega_{\boldsymbol{y}}. \end{cases}$$

Using this, the quality function can be written as

$$Q(\boldsymbol{y}) = \int_{\Omega} f(\boldsymbol{x}, \boldsymbol{y}) dV(\boldsymbol{x}).$$

Since the derivative of f with respect to both $\boldsymbol{x}$ and $\boldsymbol{y}$ is bounded, the function is Lipschitz continuous. Therefore the gradient can be obtained by exchanging integration and differentiation:

$$\begin{aligned} \nabla Q(\boldsymbol{y}) &= \int_{\Omega} \nabla f(\boldsymbol{x}, \boldsymbol{y}) dV(\boldsymbol{x}) \\ &= -\int_{\Omega_{\boldsymbol{y}}} \frac{\boldsymbol{y} - \boldsymbol{x}}{\|\boldsymbol{y} - \boldsymbol{x}\|} \omega(S_X(\boldsymbol{x})) dV(\boldsymbol{x}). \end{aligned}$$

$\nabla Q(\boldsymbol{y}) = 0^d$, whenever $\boldsymbol{y} \in X$, since $Q(\boldsymbol{y}) = 0$ and $Q(\boldsymbol{x}) \geq 0$ for any $\boldsymbol{x} \in \Omega$, making the point $\boldsymbol{y} \in X$ a stationary point. In the implementation, $Q(\boldsymbol{y})$ is not maximized to find the next point for evaluation, rather $-\log(Q(\boldsymbol{y}) + 1)$ is minimized. The logarithmic transformation is used for numerical stability, and the constant 1 is to ensure that the logarithm does not approach infinity.

4.2.3 Balancing local and global search with the weight function

In Section 3.3.1 the weight function $\omega : \mathbb{R} \to \mathbb{R}^+$ was introduced and chosen as

$$\omega(z) = \exp\left(-\sigma \frac{z - \min_{\boldsymbol{x} \in \Omega} S_X(\boldsymbol{x})}{\max_{\boldsymbol{x} \in \Omega} S_X(\boldsymbol{x}) - \min_{\boldsymbol{x} \in \Omega} S_X(\boldsymbol{x})}\right),$$

where the factor σ controls the balance between local and global search. Empirically, letting the value of σ vary in a cyclic manner during the course of iterations leads to the best results, as opposed to letting the value be constant. All choices of σ in the interval $[0, \infty]$ are valid, though in practice setting $\sigma = \infty$ is of course not possible. However, the behavior of the quality function when $\sigma \to \infty$ is desirable, since it means selecting the global minimizer of the surrogate model as the next point. In order to achieve this, MATLAB's representation of ∞, Inf, is used to indicate that the quality function should be bypassed entirely, and

$$\boldsymbol{x}_{\min} = \arg\min_{\boldsymbol{x} \in \Omega} S_X(\boldsymbol{x})$$

be returned as the next point for evaluation. If, however, $\boldsymbol{x}_{\min}$ is too close to another point in X, $\boldsymbol{x}_{\min}$ is not returned; if $\boldsymbol{\sigma}$ is a vector of values, the cycle will be reset and the quality function will be called as usual. The intention of doing so is that the beginning of the $\boldsymbol{\sigma}$-vector will hopefully contain a value lower than Inf, and a point different from $\boldsymbol{x}_{\min}$ will be chosen. If on the other hand σ is not a vector, or $\boldsymbol{x}_{\min}$ is yet again returned, the algorithm will terminate.

4.2.4 Enforcing a minimum distance between chosen points

As mentioned before, some simulation software may have a built-in minimum resolution δ_i for each variable (or rather, a minimum resolution for each unit of measure). If the differences between all d variables of two points is smaller than δ_i, the resulting output will be the same. In order to handle this, the implementation relies on the existence of a vector $\boldsymbol{\sigma}$ with several σ-values. Given is a set of previously evaluated points $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id})^{\mathrm{T}}$, a not yet evaluated candidate point $\boldsymbol{y} = (y_1, \ldots, y_d)^{\mathrm{T}}$ and a vector of minimum resolutions for each variable $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_d)^{\mathrm{T}}$. If $|y_j - x_{ij}| < \delta_j$ for all of the components $j = 1, \ldots, d$ of the candidate point, for some $i \in \{1, \ldots, n\}$, then the candidate point is too close to a previously evaluated point. Since the algorithm cycles through all the values in $\boldsymbol{\sigma}$ in order to bias the search in either a global or a local way, the cycle is reset to the beginning where (hopefully) a global search will ensure. This does of course depend on how the $\boldsymbol{\sigma}$-vector has been constructed.

4.3 Adapting the algorithm to multiobjective optimization

The main difference between the single and multiobjective algorithm lies in the weight function ω, which no longer takes values from the surrogate model $S_X(\boldsymbol{x})$ as an argument, but rather the Euclidean distance from the point $\boldsymbol{S}(\boldsymbol{x}) = (S_X^1(\boldsymbol{x}), \ldots, S_X^m(\boldsymbol{x}))^{\mathrm{T}}$ in objective function space to the surrogate models' Pareto front.

This subsection mostly describes the machinery around the quality function leading up to calculating the distance to the approximate Pareto front. First of all, the implementation relies on an external solver to calculate a finite set of Pareto optimal points given a number of objective functions. The inner workings of that solver is of no concern to the implementation, although the correctness of the resulting points is important for the quality of the distance function. The distance function is what the quality function calls in place of the actual surrogate model, the design of which is to be described presently. The current implementation of the distance function is limited to two objective functions only, the description is however generalizable to several objective functions.

It is assumed that the surrogate models $S_X^1(\boldsymbol{x}), \ldots, S_X^m(\boldsymbol{x})$ have been normalized to yield values in the range $[0, 1]$. This is to give the different objectives equal value in case of incommensurability; compare with Section 3.4.1 on the transformation of the parameter space. The distance function seen from the point of view of the quality function is denoted $\mathrm{dist}_{Z_{\boldsymbol{S}}^*} : Z \to \mathbb{R}^+$ (see Section 3.3.2), returning the shortest distance to the approximate Pareto front. In order to construct the distance function, the set of Pareto points generated by the external solver has to be used to approximate the continuous Pareto front $Z_{\boldsymbol{S}}^*$. This can be done through the following procedure: given a set of points $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k\} \in Z_{\boldsymbol{S}}^*$ all lying on the Pareto front $Z_{\boldsymbol{S}}^*$ of the surrogate models, create an RBF-based interpolation of the front. At a first glance, it is not apparent how to interpolate a *curve* rather than a function, since a curve cannot be described as the bijective input-output relation of an ordinary function. The trick is to add an additional dimension to the problem and assign "fake" values in the added dimension as follows. Given points $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ on a simple curve ∂C, give these points the value 0. For each point $\boldsymbol{z}_i$ on the curve, add one point inside the curve and one point outside the curve and give these two points the values $+1$ and -1 respectively. Each added point should lie in the (negative) normal direction of their respective point on the curve, see Figure 19. Proceed to interpolate the set of points and values as usual, and denote the interpolation P. The interpolation of the curve will now be given by the solution to $P(\boldsymbol{z}) = 0$. This enables $Z_{\boldsymbol{S}}^*$ to be approximated as:

$$Z_{\boldsymbol{S}}^* \approx \{\boldsymbol{z} \in Z_{\boldsymbol{S}} : P(\boldsymbol{z}) = 0\}.$$

With the Pareto front interpolated, the Euclidean distance from any point in the feasible objective region to the Pareto front may now be calculated by solving the following auxiliary optimization problem:

$$\begin{aligned} \mathrm{dist}_{Z_{\boldsymbol{S}}^*}(\boldsymbol{z}) = \min_{\boldsymbol{y}} \quad & \|\boldsymbol{y} - \boldsymbol{z}\|, \\ \text{s.t.} \quad & P(\boldsymbol{y}) = 0. \end{aligned} \tag{39}$$

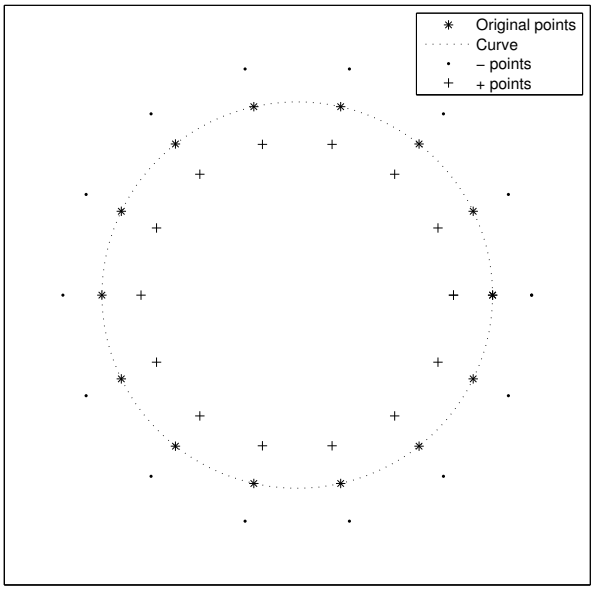


Figure 19: Example of how to place help points when interpolating a general curve in two dimensions. The z-axis would be in the direction normal to the paper surface and the resulting interpolation would look something like a cone, with its maximum value at the center.

The auxiliary problem (39) may be solved every time the quality function calls the dist-function, but this can take considerable time (for example, in six dimensions and with 9 intervals in the trapezoidal integral approximation, the auxiliary problem will in the worst case be solved 10^6 times). The approach chosen in this implementation was instead to create an interpolation of the distance function as well. This was done by generating a number of training points, solve the auxiliary problem (39) for each of these points and then to interpolate them accordingly. In two dimensions a simple 20×20 grid of the feasible objective region was used; in higher dimensions this approach is not practical, instead some random sample strategy has to be employed in order to limit the number of training points but still retain some desired space-filling property.

The Pareto front of the surrogate models may be disconnected and may not "extend" throughout the extremes of the feasible objective region (which is still assumed to be $[0, 1]^m$), and if these problems are not addressed properly, the resulting intermediate interpolation P may exhibit a strong unwanted curvature, not in line with what is expected. To remedy this, additional "filling" points have to be added where needed. Bear in mind that these filling points have no correspondence to the Pareto front, they are simply an aid to the interpolation. In reality, there is no way of knowing how the Pareto front would look where the filling points have been added (if there had been any Pareto front there), at least in this way the distance measure becomes consistent. This is relatively easy in two dimensions but becomes harder as the dimensionality increases. There are two types of filling points; points extending the Pareto front parallel to the Cartesian axes (in essence creating a set of weakly Pareto optimal points, refer to definition (7) on page 15), and points helping to bridge discontinuities in the front, see Figure 20. The filling points may or may not lie in the feasible objective region $Z_{\boldsymbol{S}}$ of the surrogate models. The general outline of steps needed in creating the distance function are described in Algorithm 5.

The weight function ω in the multiobjective case has the parameter σ available for adjusting the balance between local and global search. Just as in the single objective case, see Section 4.2.3,

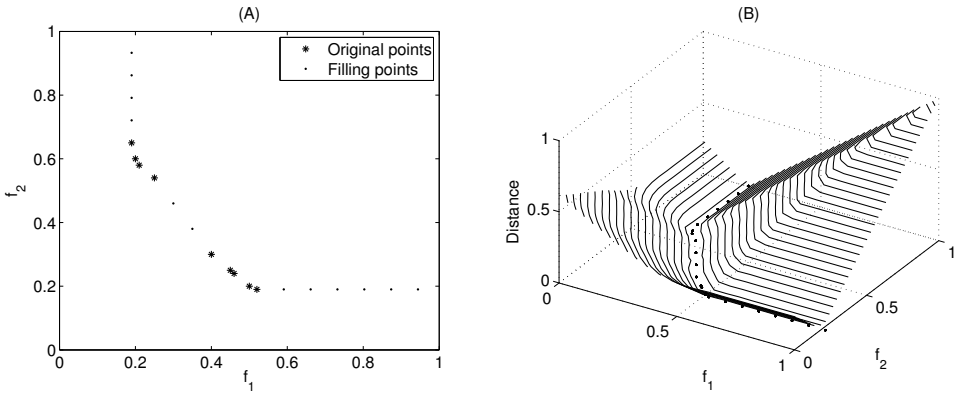


Figure 20: Figure (A) depicts a sample two-dimensional Pareto front (with very few points) and the filling points used to bridge discontinuities and extend the Pareto front. Figure (B) depicts a three dimensional contour plot of the interpolated distance function $\mathrm{dist}_{Z_S^*}$ (not the intermediate interpolation P).

Input: Surrogate models for each objective function $\boldsymbol{S}(\boldsymbol{x}) = (S_X^1(\boldsymbol{x}), \ldots, S_X^m(\boldsymbol{x}))^{\mathrm{T}}$
Output: Interpolated distance function $\mathrm{dist}_{Z_S^*} : Z_S \to [0, 1]$
scale surrogate models to unity;
generate a finite set of points on the surrogates' Pareto front;
bridge larger gaps in the set; extend front where needed;
add help points inside and outside of the front;
add fake values to the points and create interpolation $P(\boldsymbol{z})$;
generate a set $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N$ of training points in the feasible objective region of the surrogates;
for *i=1* **to** N **do**

$$d_i = \min_{\boldsymbol{x}} \quad \|\boldsymbol{z}_i - \boldsymbol{x}\| \quad \text{s.t.} \quad P(\boldsymbol{x}) = 0 \;;$$

end
normalize the distances d_i to unity;
create interpolation $\mathrm{dist}_{Z_S^*}$ of points $\boldsymbol{z}_i$ and distances d_i, for all $i = 1, \ldots, N$;
return $\mathrm{dist}_{Z_S^*}$;

Algorithm 5: Creating the distance function.

σ can be set to Inf. This is handled in a similar way, by bypassing the quality function entirely and choosing the next point for evaluation directly from the surrogate models' Pareto front Z_S^*. As stated earlier, an external solver is used to calculate a finite set of Pareto optimal points. From this set the point with most uncertainty, that is, the point farthest away from any other points in X, is chosen and returned.

4.4 Using Latin Hypercube Design for selecting initial points

The purpose of this initial point strategy was to create a π_1 unisolvent set X with a comparably large spread of points. This was solved in a very simple and straightforward way, and is described in Algorithm 6. A large amount of LHDs are simply generated, then the points are checked for unisolvence and feasibility and thereafter the configuration with largest minimal distance between points is chosen.

Input: Box constraints, $\boldsymbol{x}_L$, $\boldsymbol{x}_U$, number of initial points to generate = n
Output: Set of initial points X
d = dimension($\boldsymbol{x}_L$);
for *i=1* **to** *amount tries* **do**
  `// create LHD matrix`
  for *j=1* **to** d **do**
    $M_{.j}$ = random perturbation of $(1, \ldots, n)$;
  end
  `// generate points from matrix`
  for *j=1* **to** n **do**
    for *k=1* **to** d **do**
      $x_{j,k} = \frac{M_{j,k}-1}{n-1}(x_{U,k} - x_{L,k}) + x_{L,k}$;
    end
  end
  $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$;
  `// pick best point configuration`
  if X *is feasible and* π_1 *unisolvent* **then**
    $h = \min_{i \neq j} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$;
    if $h < h_{\text{keep}}$ **then**
      $X_{\text{keep}} = X$;
      $h_{\text{keep}} = h$;
    end
  end
end
return X_{keep};

Algorithm 6: Latin hypercube design.

4.5 Global/local optimization solvers

At various stages of the implementation the need to solve auxiliary optimization problems arises, such as finding the minimum of the current surrogate model(s) or finding the parameter which maximizes the quality function. These problems have been solved using 3rd party commercial implementations, but any kind of solver up to the task may be used.

There are typically three types of optimization problems that need to be addressed in the implementation. (1) Non-linear objective function with non-linear constraints (NLP), (2) linear objective function with linear constraints (LP) and (3) non-linear vector-valued objective function (Pareto). All of these problems are also subject to box constraints. For optimization problems of the first type, a global solver was used to generate a suitable feasible starting point for the subsequent call to a local solver. In some cases, a known previously best point was used as alternative starting point for the local solver, to ensure proper convergence to the global minimum. For minimizing the surrogate function S, the alternative starting point $\boldsymbol{x}_j$ was chosen such that

$$j = \arg \min_{i=1,\ldots,n} \{f_i\},$$

and for the quality function

$$\arg\min_{\boldsymbol{x} \in \Omega} S(\boldsymbol{x})$$

was chosen. For the problem of finding the minimum distance to interpolated Pareto front, a global solver was not used to generate a starting point. Instead, the closest known point (either a point generated by the external multiobjective algorithm, or a point extending the front or bridging gaps) was used as starting point. For the second type a general LP-solver was used. For the third and last type a multiobjective solver was used to produce a set of Pareto optimal solutions.

The commercial optimization toolbox TOMLAB was used to supply all the above mentioned solvers, except that for multiobjective optimization. For multiobjective optimization, *MultiOb* (see

[Han06]) from ITWM was used; this solver is based on a genetic algorithm. As global solver, *glcSolve* or *glcFast* were used. These are implementations of the DIRECT algorithm (see [JPS93]) in C and Fortran respectively. As local solver *snopt* was used, which is an implementation of SNOPT, an algorithm based on sequential quadratic programming. For more information on SNOPT, see for instance [GMS05]. As solver for the linear problem *lp-minos* was used, which is an implementation of the classic (primal) Simplex algorithm. lp-minos is part of a larger optimization package called MINOS, originally developed at the Systems Optimization Laboratory (SOL) at Stanford University.[16]

5 Results and comparisons

This section intends to evaluate the performance of the algorithm on a few analytical test functions found in the literature. The algorithm is tested on deterministic functions, functions disturbed by noise and deterministic functions with multiple objectives; the latter are used to evaluate the performance of a version of the algorithm applicable to multiobjective optimization. The impact on the final result of the initial point strategy is also examined, since accidentally choosing "good" initial points may lead to fast convergence, but with little help from the algorithm itself. An open question is how to compare different algorithms, and what features to reward. Some algorithms may produce good results on one class of test functions, while failing on another class. Robustness is an important aspect when testing an algorithm; the rate of convergence should not be diametrically changed, for instance, when changing the set of initial points somewhat. The rate of convergence itself is naturally one of the most important aspects of any optimization algorithm, since this is where resources may be saved. One may always argue about the representativeness of the "classical" test functions, such as the Dixon–Szegö suite. Do they represent the class of problems that the algorithm is supposed to be good at? If an algorithm performs well on a test bed of problems, is this any guarantee that it will perform well in other, less controlled, situations as well? Our standpoint is that good results on test functions may at least point in the right direction.

Finally, the algorithm was tested on real life simulation problems. The progress of optimizing the simulation of combustion engines in cooperation with Volvo Powertrain is discussed. Although the algorithm was not fully adapted to handle this problem (new issues arose during problem formulation), an iteration of the algorithm was performed and new ideas for addressing some of the problems were developed. To test the algorithm on a less complicated problem, a multiobjective optimization of the simulation of a heater element was performed as well.

5.1 Deterministic functions

The suite of deterministic functions is part of the well-known test bed of Dixon–Szegö (see [DS78]), summarized in Table 3. The test functions are subject to box constraints only.

Function	Dimension	# local minima	Domain
Branin	2	3	$[-5, 10] \times [0, 15]$
Goldstein–Price	2	4	$[-2, 2]^2$
Hartman 3	3	4	$[0, 1]^3$
Shekel 5	4	5	$[0, 10]^4$
Shekel 7	4	7	$[0, 10]^4$
Shekel 10	4	10	$[0, 10]^4$
Hartman 6	6	4	$[0, 1]^6$

Table 3: The Dixon–Szegö test bed. All functions have one global minimum, except for Branin, which has three.

[16] http://www.stanford.edu/group/SOL/

The algorithm qualSolve was compared with EGO (originally presented in [JSW98], using TOMLAB's implementation), Gutmann's algorithm (originally presented in [Gut01]), and rbfSolve (TOMLAB's improvement of Gutmann's algorithm) on the seven test functions. All algorithms are designed to handle expensive objective functions and hence are suited for comparison. The qualSolve algorithm was run with $\boldsymbol{\sigma} = (0, 10, 20, 20, 50, 50, \infty)^{\mathrm{T}}$ and using interpolation as surrogate model. As stopping criterion, following [Gut01], the relative error was used in order to enable a comparison with reported results. The relative error is defined as

$$e_i = \frac{|f_i - f^*|}{|f^*|},$$

where f_i is the current evaluated function value and f^* the global minimum objective value. The global minimum has to be non-zero, limiting the usefulness of this stopping criterion. However, the functions belonging to the Dixon–Szegö test bed all have non-zero optimal values. The algorithms were terminated when the relative error reached 1%, or when the number of function evaluations exceeded 150, whichever came first.

Two different choices of initial points were used. The first choice was to use all corner points of the domain, resulting in 2^d initial points (see Section 3.2; this strategy was also used in [Gut01]). To see how much of an impact the initial points have on the outcome, a second run was performed using $d + 1$ initial points randomly generated by LHDs. 20 runs were performed for each test function and algorithm, in order to get a statistically valid sample. The results are shown in Tables 4 and 5; it is apparent that initial points do play a big role in the speed of convergence. Note further that some runs did not finish in less than 150 iterations. To show that this is not a result of changing the initial points for the algorithms, Table 6 presents the results for EGO and rbfSolve run with the standard initial points as implemented in TOMLAB.

Function	qualSolve	Gutmann	rbfSolve	EGO
Branin	32	44	59	**21**
Goldstein–Price	**60**	63	84	125
Hartman 3	46	43	18	**17**
Shekel 5	**70**	76	–	–
Shekel 7	85	**76**	–	–
Shekel 10	71	**51**	–	–
Hartman 6	99	112	109	**92**

Table 4: Dixon–Szegö test functions evaluated using the 2^d corners of the domain as initial points. Shown is the number of function evaluations until the relative error goes below 1%. The best values are in bold. '–' denotes that the relative error never went below 1% during the 150 functions evaluations. The results presented for Gutmann's algorithm are obtained from [Gut01], whereas the data for rbfSolve and EGO was produced by running TOMLAB's implementation.

To give a graphical illustration of the differences between compared algorithms, qualSolve, EGO, Gutmann's algorithm (as implemented by TOMLAB) and rbfSolve were run on the 'Tilted' Branin[17] test function (Figure 21) and the results plotted and analysed in Figures 22–26. In Figure 22 the interpolation and points evaluated by qualSolve with the usual sigma vector $\boldsymbol{\sigma} = (0,\ 10,\ 20,\ 20,\ 50,\ 50,\ \infty)^{\mathrm{T}}$ together with $\boldsymbol{\delta} = (0.1, 0.1)$ are shown. Figure 24 shows the points evaluated by the other algorithms and interpolations created from them using thin plate spline as RBF. It should be pointed out that EGO does not use RBFs to construct the response surface (it uses kriging) and rbfSolve uses a cubic RBF. Still, interpolating using thin plate splines gives a hint on the information that can be obtained from the evaluated points. Comparing Figures 22 and 24 shows that qualSolve increases the density of the evaluated points in a continuous manner when approaching the minimum, whereas Gutmann's algorithm tends to mix very global searches with

[17]The Branin test function was tilted to obtain only one global minimum instead of three; see [HANZ06].

Function	qualSolve	rbfSolve	EGO
Branin	100%, 26.85 (4.52)	100%, 62.70 (1.34)	100%, 23.00 (0.00)
Goldstein–Price	100%, 30.35 (10.98)	100%, 63.00 (0.00)	0%
Hartman 3	100%, 38.80 (16.94)	100%, 28.00 (0.00)	100%, 31.00 (0.00)
Shekel 5	30%, 61.00 (10.90)	0%	0%
Shekel 7	60%, 66.00 (7.62)	0%	0%
Shekel 10	60%, 78.00 (39.82)	0%	0%
Hartman 6	95%, 50.74 (14.69)	100%, 122.00 (0.00)	0%

Table 5: Dixon–Szegö test functions evaluated by qualSolve, rbfSolve and EGO using LHD of size $d+1$ as initial points. Shown for each algorithm are the percentage of runs that reached 1% relative error in less than 150 function evaluations, the mean number of evaluations to reach the goal and the standard deviation. Runs that did not reach the target within 150 function evaluations are not included in the mean. The total number of runs for each function and algorithm was 20. The data for rbfSolve and EGO was obtained by running TOMLAB's implementation.

Function	rbfSolve	EGO
Branin	100%, 56.00 (0.00)	100%, 30.00 (0.00)
Goldstein–Price	100%, 111.00 (0.00)	0%
Hartman 3	100%, 52.00 (0.00)	100%, 36.00 (0.00)
Shekel 5	0%	0%
Shekel 7	0%	0%
Shekel 10	0%	0%
Hartman 6	100%, 115.00 (0.00)	100%, 73.00 (0.00)

Table 6: rbfSolve and EGO run with their default initial points, as implemented in TOMLAB.

a very local clustering of evaluated points (an example are the points inside the box in Figure 24, where the algorithm has placed evaluations so dense that it is impossible to distinguish individual points). This behavior is not shared by rbfSolve. Instead, many evaluated points seem to be placed in areas with high function values without any focus on local search. For both algorithms the result is a very low number of points in the area around the global minimum of the function. EGO, on the other hand, succeeds in locating the global minimum. Still, it could be argued, that the transition between the globally spread points and the locally spread points around the minimum is worse than that of qualSolve. There is a large area around the global minimum in which no points have been evaluated. Finally the result of qualSolve, without the δ parameter set but using the same σ-vector, is shown. This illustrates the impact of values of δ; the mechanism prevents points from being evaluated denser than δ apart and further spreads the points. In Figures 23 and 25 the power functions (see Definition 24) corresponding to the interpolations in Figures 22 and 24 are plotted. A comparison between the power function obtained from qualSolve and the power function obtained from the other algorithms is made in Figure 26. Considering the power function as a measure of uncertainty, this shows that whereas EGO and qualSolve have low uncertainty around the minima, rbfSolve and Gutmann have high uncertainty there. Furthermore, although EGO has low uncertainty at the minimum, qualSolve has a better coverage in the basin around it.

5.2 Non-deterministic functions

In order to investigate the performance of qualSolve on non-deterministic test functions (test functions containing noise), the test bed presented in Table 7 was used. The noise was created by adding a normally distributed random value. For details on the test functions see [HANZ06].

The qualSolve algorithm was compared with SKO (presented in [HANZ06]), which is an extension of EGO designed to handle expensive and noisy objective functions. Comparisons with

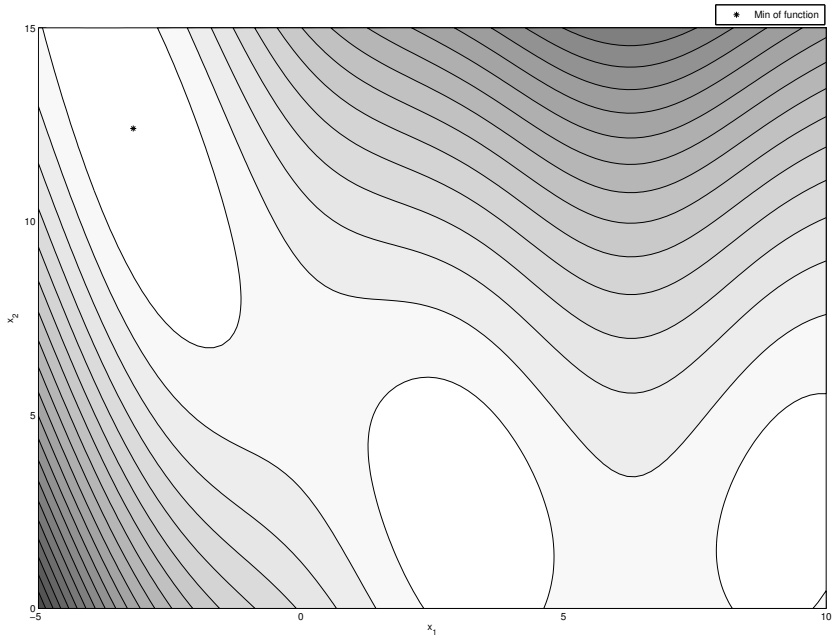


Figure 21: 'Tilted' Branin test function.

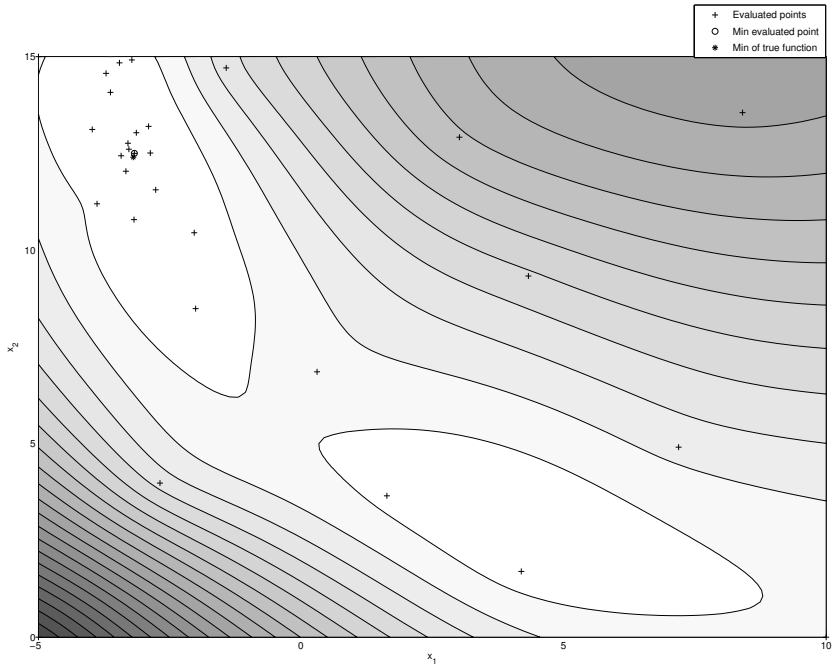


Figure 22: The figure shows the interpolation created from points evaluated by qualSolve when solving the 'Tilted' Branin test function, using $\boldsymbol{\delta} = (0.1, 0.1)$.

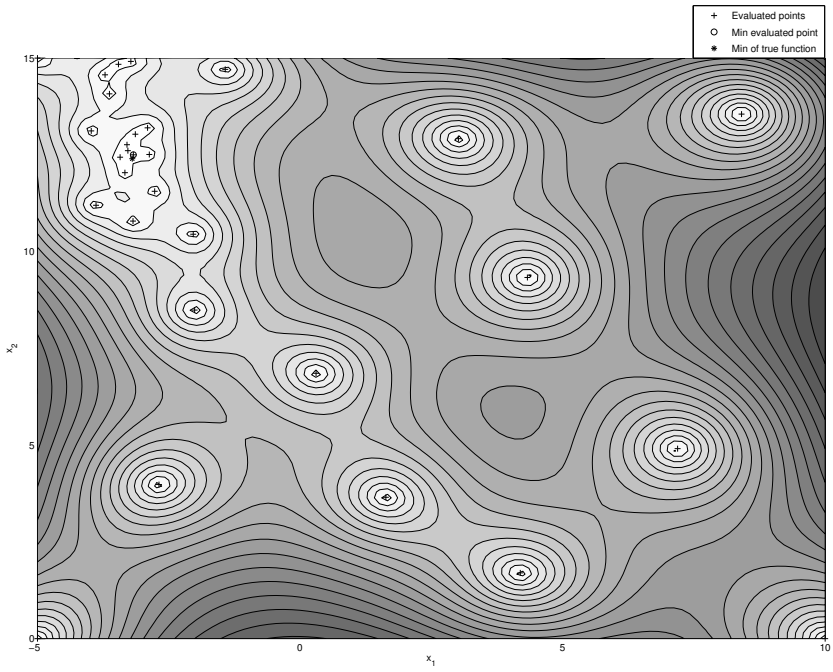


Figure 23: Uncertainty for the interpolation created from points evaluated by qualSolve with $\boldsymbol{\delta} = (0.1, 0.1)$ when solving the 'Tilted' Branin test function. Gray indicate high uncertainty, whereas white indicate low.

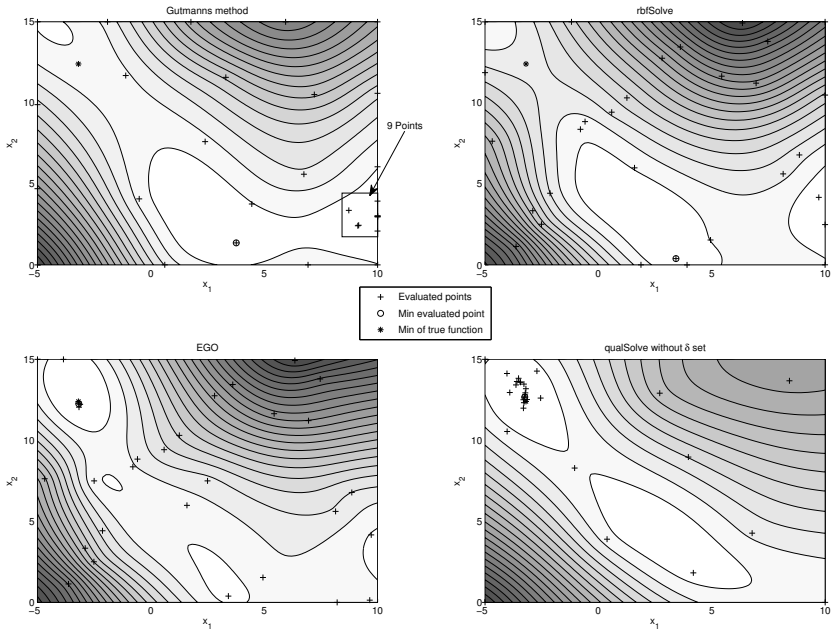


Figure 24: Interpolations created from points evaluated by different methods, when solving the 'Tilted' Branin test function.

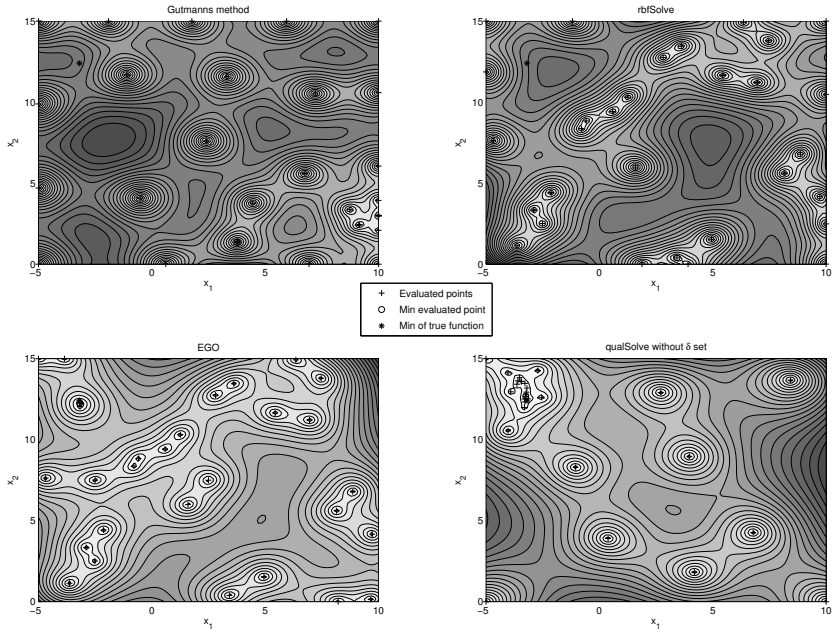


Figure 25: Uncertainty for the interpolations created from points evaluated by different methods, when solving the 'Tilted' Branin test function. Gray indicate high uncertainty, whereas white indicate low.

Function	Dimension	# local minima	Domain
Six-hump camel back	2	6	$[-1.6, 2.4] \times [-0.8, 1.2]$
'Tilted' Branin	2	3	$[-5, 10] \times [0, 15]$
Hartman 3	3	4	$[0, 1]^3$
Ackley 5	5	–	$[-1.6, 2.4]^5$

Table 7: Test functions used for non-deterministic algorithm evaluation. The Branin function has been tilted so that only one local minimum is global. The test function Ackley 5 contains an oscillating term, and is therefore littered with local minima.

the TOMLAB implementation rbfSolve was also performed, although the algorithm was not developed to handle noisy functions. In order to compare the algorithms, following [HANZ06], a measure of convergence is defined:

$$G_i = \frac{f(\boldsymbol{x}_{\text{in}}) - f(\boldsymbol{x}_i)}{f(\boldsymbol{x}_{\text{in}}) - f^*}, \tag{40}$$

where $\boldsymbol{x}_i$ is the minimum proposed by the algorithm at the i:th iteration and f is the "true" function. The point $\boldsymbol{x}_{\text{in}}$ is the median initial point:

$$\boldsymbol{x}_{\text{in}} = \arg\text{median}_{\boldsymbol{x} \in X_{\text{in}}}(f(\boldsymbol{x})),$$

where X_{in} is the set of initial points. The measure compares the deviation from the optimum in the current iteration with the deviation from the optimum at the initial points. It is constructed such that $G \leq 1$. Reaching $G = 1$ means that the global minimum has been found.

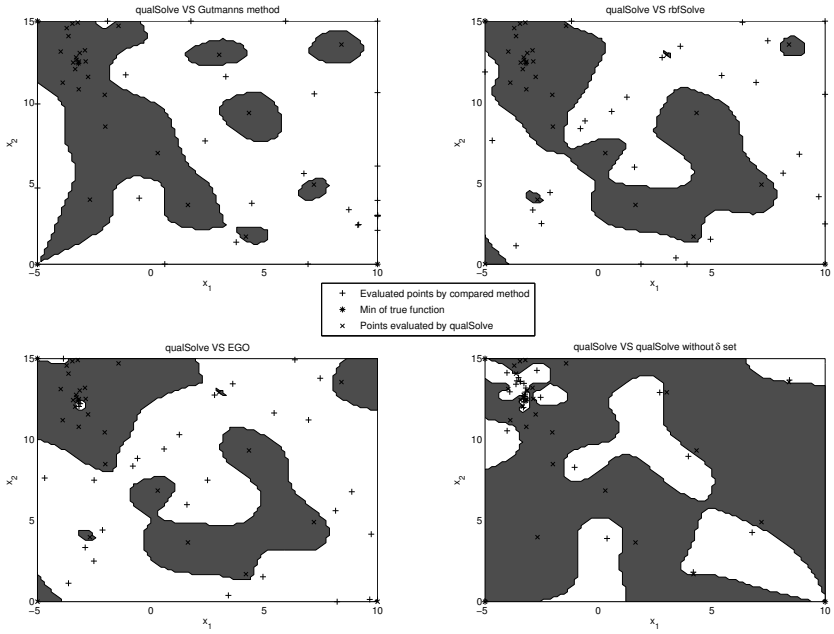


Figure 26: Comparison of uncertainty between qualSolve with $\boldsymbol{\delta} = (0.1, 0.1)$ and other methods when solving the 'Tilted' Branin test function, gray areas indicates lower uncertainty for qualSolve and white indicates lower uncertainty for the compared method.

The $\boldsymbol{\sigma}$ vector used for the non-deterministic test functions was slightly modified from the deterministic case, to $\boldsymbol{\sigma} = (0, 10, 20, 20, 50, 50)^{\mathrm{T}}$. The reason for removing the ∞-term is that, when optimizing deterministic functions, the smallest evaluated function value is used as a measure of success. When dealing with noisy functions, on the other hand, the goal is to obtain the minimum of the "true" function, not the smallest noisy function value. Hence, instead of the smallest evaluated value, the minimum of the surrogate ($\arg\min_{\boldsymbol{x}\in\Omega} S(\boldsymbol{x})$) is used as output from qualSolve (that is as $\boldsymbol{x}_i$ in (40)). There is no reason to evaluate this point using the noisy function (which would otherwise have been done by the ∞-term). The surrogate model was constructed by choosing an approximation parameter η, by cross-validation, from the vector

$$\boldsymbol{\eta} = (10^{-8}, 10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 0.2)^{\mathrm{T}}.$$

Two different choices of initial points were used. The first was $11d$ LHD initial points, as done in [HANZ06], in order to compare with the results from SKO. This strategy yields many initial points, way above the $d + 1$ number of points required by qualSolve. Hence, the second choice was to use $d + 1$ LHD initial points. The algorithms were run until $G \geq 0.99$ or a maximal amount of function evaluations was reached. If G did not reach 0.99 the run was considered as unconverged. Tables 8 and 9 show the results obtained. In the case when using $11d$ initial points SKO attains the lowest mean value, but qualSolve has a somewhat higher convergence percentage on all test functions except the Six-hump camel back. The poor performance of rbfSolve motivates the necessity of handling noise explicitly. By decreasing the number of initial points to $d + 1$, the performance of qualSolve was considerably improved and outperformed the SKO algorithm on all test functions except the 'Tilted' Branin.

To give a graphical illustration of the importance of handling noise when constructing an RBF-based surrogate model, the interpolation and approximation of noisy data obtained from 'Tilted' Branin is presented. Figure 27 shows the test function itself, without noise. Figure 28 shows the

Function	Noise	qualSolve	SKO	rbfSolve
Six-hump camel back	$N(0, 0.12^2)$	100%, 31.40 (11.23)	100%, 29.2 (5.7)	94%, 30.06 (6.37)
Six-hump camel back	$N(0, 0.24^2)$	92%, 34.70 (12.25)	94%, 29.4 (6.6)	86%, 37.37 (15.35)
'Tilted' Branin	$N(0, 2.0^2)$	100%, 58.40 (25.09)	98%, 28.4 (5.3)	80%, 60.70 (19.17)
Hartman 3	$N(0, 0.08^2)$	100%, 61.22 (22.85)	96%, 45.4 (7.9)	100%, 71.30 (11.99)
Ackley 5	$N(0, 0.06^2)$	100%, 100.54 (16.85)	94%, 98.9 (5.6)	12%, 148.33 (31.09)

Table 8: Various test functions with LHD of size $11d$ (where d is the dimension) as initial points. Shown are the means and standard deviations (in parentheses) of the number of function evaluations necessary for reaching $G \geq 0.99$. The percentage is the number of runs that reached this target. The maximal amount of evaluations was 200 for all functions, except for the two Six-hump camel back ones which were only alotted 100 evaluations. The total number of runs for each function and algorithm was 50. Data for SKO was obtained from [HANZ06], whereas data for rbfSolve was obtained by performing runs with TOMLAB's implementation.

Function	Noise	qualSolve	rbfSolve
Six-hump camel back	$N(0, 0.12^2)$	100%, 17.14 (10.85)	96%, 33.04 (13.06)
Six-hump camel back	$N(0, 0.24^2)$	98%, 21.88 (16.04)	84%, 35.19 (18.24)
'Tilted' Branin	$N(0, 2.0^2)$	98%, 47.29 (22.51)	42%, 71.81 (26.31)
Hartman 3	$N(0, 0.08^2)$	100%, 37.12 (25.66)	100%, 32.86 (15.50)
Ackley 5 SD	$N(0, 0.06^2)$	100%, 59.58 (18.37)	10%, 103.00 (23.87)

Table 9: Various test functions with Latin Hypercube Design of size $d+1$ (where d is the dimension) as initial points. Shown are the means and standard deviations (in parentheses) for reaching $G \geq 0.99$. Data for rbfSolve was obtained by performing runs with TOMLAB's implementation.

interpolation of noisy data and Figure 29 shows the same data approximated. The approximation corresponds better to the "true" function, and is therefore more suited for use as a surrogate. It should be noted that the standard deviation of the noise used to create these illustrations was 10 times higher than during the runs presented in Tables 8 and 9. This was done in order to illustrate the difference more clearly.

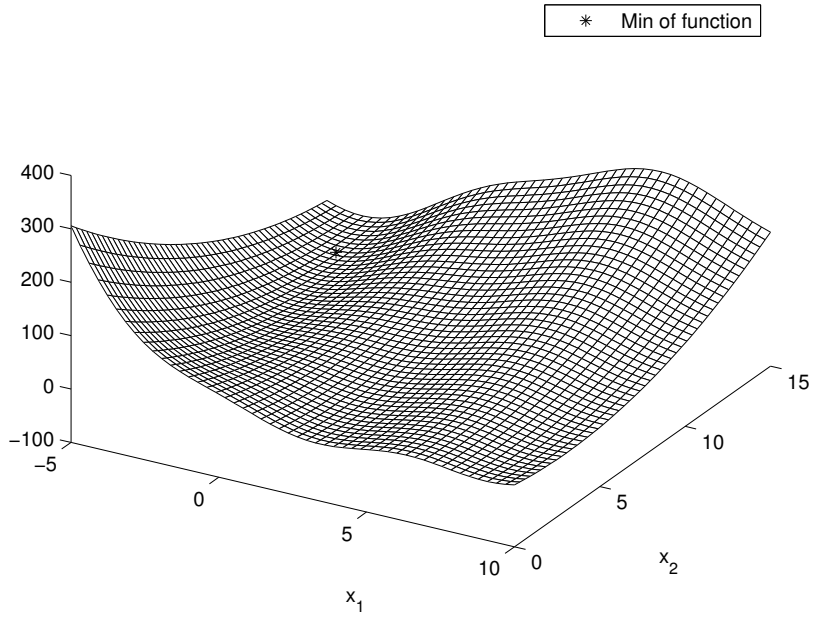


Figure 27: The 'Tilted' Branin test function.

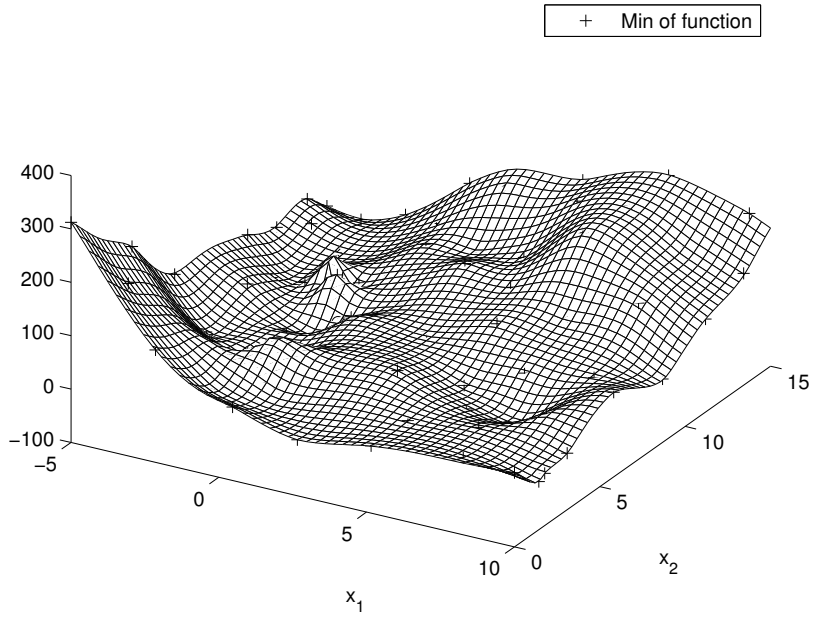


Figure 28: The surrogate model created by interpolating noisy data from the 'Tilted' Branin test function. The noise used was obtained from a $\mathrm{N}(0, 20^2)$ distribution.

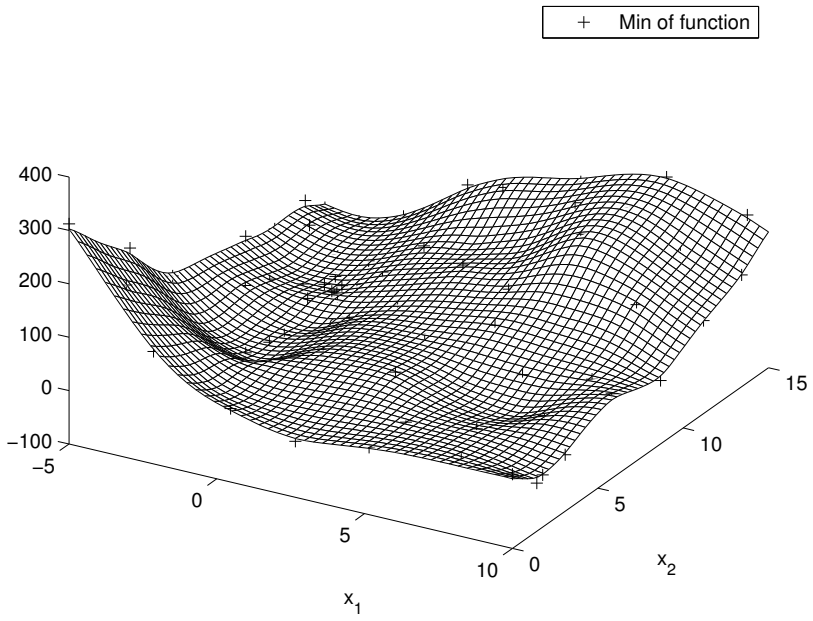


Figure 29: The surrogate model created by approximating noisy data from the 'Tilted' Branin test function. The noise used was obtained from a $\mathrm{N}(0, 20^2)$ distribution.

5.3 Vector-valued functions

The success of a multiobjective optimization algorithm is somewhat harder to quantify than that of a single objective algorithm. This is due to the fact that a multiobjective algorithm returns a set of points rather than a single point, and the figures of merit used for single objective optimization can not easily be adapted to the multiobjective case. First off, there seem to be no *absolute* figures of merit in the literature, at least not general enough to be applied to any kind of algorithm; all figures are relative. This means that there is no easy way of comparing the proposed algorithm to other existing algorithms without actually running them. Secondly, there appears to exist almost no algorithms in the community that aims to keep the number of function evaluations to a minimum; this makes a fair comparison hard to make. One of the few algorithms designed with this in mind is *ParEGO* [Kno06]. In this article, the performance of ParEGO is compared to that of NSGA-II, a genetic algorithm, showing promising results. Two figures of merit were used, the *Hypervolume Indicator* and the *Additive Binary Epsilon Indicator* (see [Zit99] and [ZTL$^+$03] respectively), both relative measures. Unfortunately, no implementation of ParEGO was available for comparison with the proposed algorithm. The only multiobjective algorithm available was MultiOb, used internally by the implementation of qualSolve, in order to obtain a set of Pareto optimal solutions to the surrogate models, see Section 4.3. There would however be no point in comparing qualSolve with MultiOb since: a) the algorithm uses MultiOb (and consequently could never outperform MultiOb, at least with respect to the quality of solutions if MultiOb is allowed enough function evaluations); and b) the algorithms are of different kinds and a comparison based on a small number of evaluations would be unfair. 100 function evaluations is not enough for MultiOb to produce any kind of near-optimal solution (if such an expression is applicable); being a genetic algorithm it rather needs 10,000 function evaluations.

In light of this, what follows is a simple qualitative evaluation of the implementation of qualSolve. This evaluation is only present to indicate that qualSolve also has promise in the area of multiobjective optimization. In Table 10, the vector-valued test functions chosen are presented.

These have not been subjected to any noise. The algorithm was run for 100 and 250 function evaluations on each function respectively, with the following $\boldsymbol{\sigma}$-vector:

$$\boldsymbol{\sigma} = (0, 10, 10, 10, 20, 20, 20, 20, 20, 20, 50, 50, 50, 50, 50, 50, 50, \infty, \infty, \infty)^{\mathrm{T}}.$$

The vector may be recognized as a prolonged version of the vectors used in the previous numerical experiments, this is so because a larger region is relevant, hence requiring more function evaluations for decent certainty. The test functions selected all supply two objective functions, and have varying complexity. The functions were originally intended to evaluate the performance of algorithms designed for inexpensive objective functions (such as genetic multiobjective algorithms), and as such may not be completely suited for the category of solvers into which the proposed algorithm falls. In a word, 100–200 function evaluations may not be enough to properly explore the functions; for this reason one of the functions has been slightly modified from its original definition. Here follows the analytical expressions of the test functions used; 3D plots of the three two-dimensional functions can be seen in Figures 30–32.

Function	Dimension
OKA1	2
OKA2	3
Kursawe	3
Deb bimodal	2
Modified Deb bimodal	2

Table 10: Vector-valued test functions used to evaluate qualSolve. All functions define two objective functions, which are to be minimized.

OKA1 [Kno06] (Figure 30):

$$\begin{aligned}
f_1(\boldsymbol{x}) &= x_1', \\
f_2(\boldsymbol{x}) &= \sqrt{2\pi} - \sqrt{|x_1'|} + 2|x_2' - 3\cos(x_1') - 3|^{\frac{1}{3}}, \\
x_1' &= \cos\left(\frac{\pi}{12}\right) x_1 - \sin\left(\frac{\pi}{12}\right) x_2, \\
x_2' &= \sin\left(\frac{\pi}{12}\right) x_1 + \cos\left(\frac{\pi}{12}\right) x_2, \\
x_1 &\in [6\sin\left(\frac{\pi}{12}\right), 6\sin\left(\frac{\pi}{12}\right) + 2\pi\cos\left(\frac{\pi}{12}\right)], \\
x_2 &\in [-2\pi\sin\left(\frac{\pi}{12}\right), 6\cos\left(\frac{\pi}{12}\right)].
\end{aligned}$$

OKA2 [Kno06]:

$$\begin{aligned}
f_1(\boldsymbol{x}) &= x_1, \\
f_2(\boldsymbol{x}) &= 1 - \frac{1}{4\pi^2}(x_1 + \pi)^2 + |x_2 - 5\cos(x_1)|^{\frac{1}{3}} + |x_3 - 5\sin(x_1)|^{\frac{1}{3}}, \\
x_1 &\in [-\pi, \pi], \quad x_2, x_3 \in [-5, 5].
\end{aligned}$$

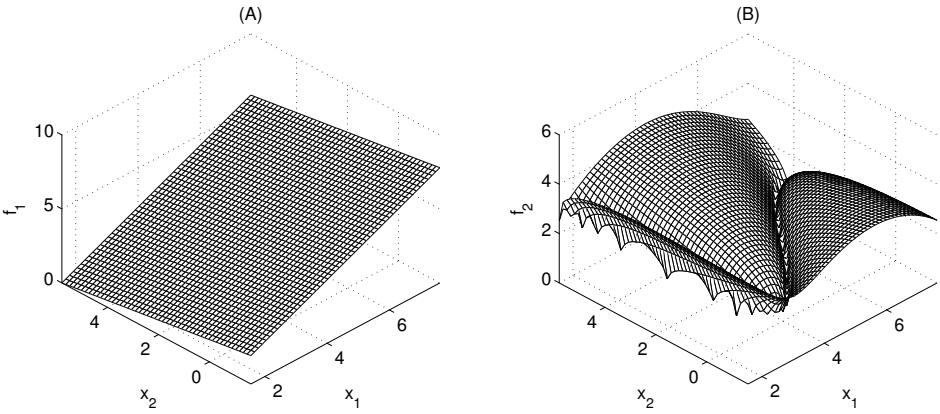


Figure 30: The objective functions of the OKA1 test function.

Kursawe:

$$
\begin{aligned}
f_1(\boldsymbol{x}) &= \sum_{i=1}^{n-1} \left(-10 \exp\left(-0.2 \sqrt{x_i^2 + x_{i+1}^2} \right) \right), \\
f_2(\boldsymbol{x}) &= \sum_{i=1}^{n} \left(|x_i|^{0.8} + 5 \sin(x_i)^3 \right), \\
\boldsymbol{x} &\in [-5, 5]^3.
\end{aligned}
$$

Deb bimodal (Figures 31 and 32):

$$
\begin{aligned}
f_1(\boldsymbol{x}) &= x_1, \\
f_2(\boldsymbol{x}) &= \frac{g(\boldsymbol{x})}{x_1}, \\
g(\boldsymbol{x}) &= 2 - \exp\left(-\left(\frac{x_2 - 0.2}{a} \right)^2 \right) - 0.8 \exp\left(-\left(\frac{x_2 - 0.6}{0.4} \right)^2 \right), \\
\boldsymbol{x} &\in [0.1, 1]^2;
\end{aligned}
$$

where the original definition has $a = 0.004$, the modified version has $a = 0.1$ in order to widen the "valley" created by the exponential function. This definition and the one for Kursawe can be found at http://www.lania.mx/∼ccoello/EMOO/testfuncs/. Figures 33–37 depict the resulting non-dominated points returned by the algorithm after 100 and 250 function evaluations respectively. MultiOb was used to produce a reference Pareto front; in order to get as smooth a front as possible multiple runs of MultiOb were performed on each test function and the results merged into a single Pareto front. This procedure amounted to a total of 250,000 function evaluations for each test function. In each figure the number of non-dominated points returned by qualSolve is reported.

5.4 Functions defined by simulations

This subsection presents results based on runs performed on simulations rather than on test functions. The problem when using simulations, compared to when using analytical test functions, is that optimality is hard to check for. On the other hand, simulations are the intended use of the algorithm, so this is an important part of the numerical results.

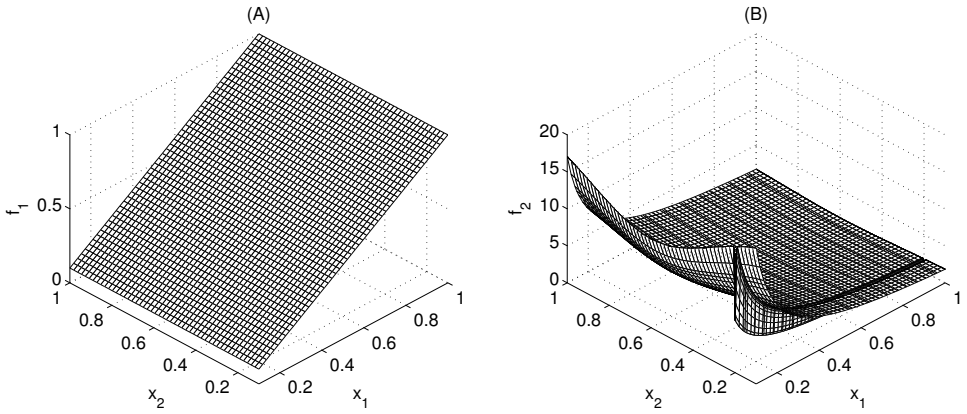


Figure 31: The objective functions of the Deb bimodal test function.

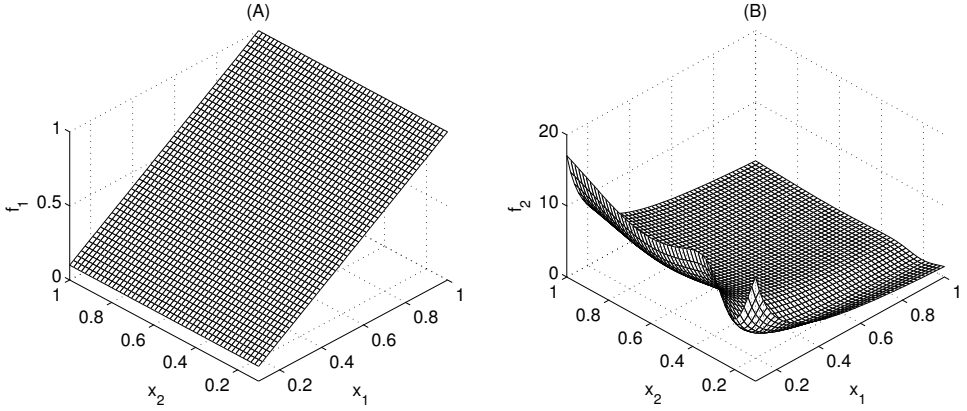


Figure 32: The objective functions of the modified Deb bimodal test function.

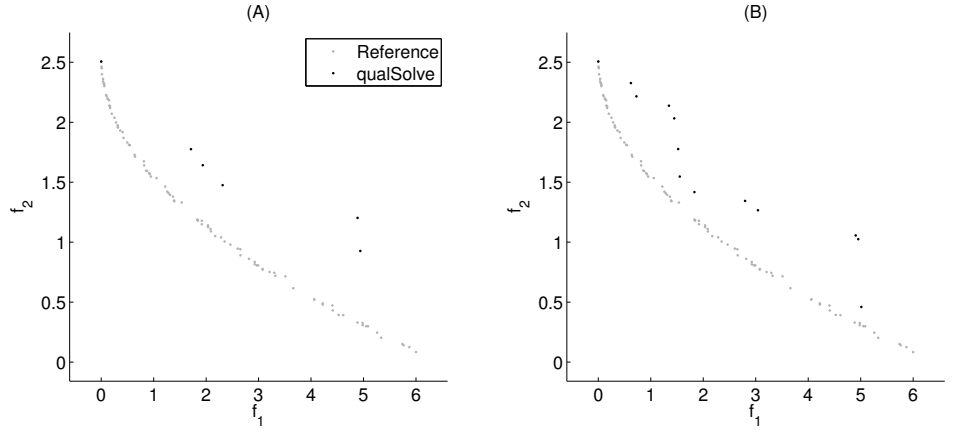


Figure 33: The Pareto fronts of the test function OKA1. The gray dots represent the reference Pareto front, the black dots the non-dominated points returned by qualSolve. Figures (A) and (B) represents the results after 100 and 250 function evaluations respectively, with 6 and 13 non-dominated points respectively.

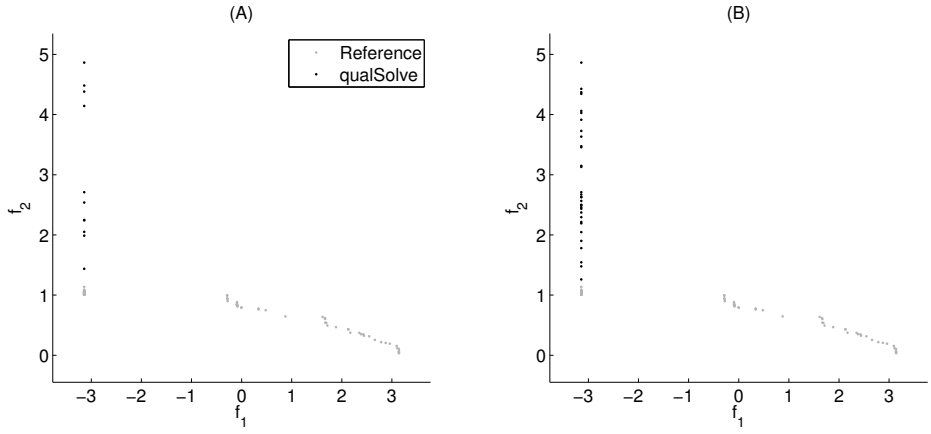


Figure 34: The Pareto fronts of the test function OKA2. The gray dots represent the reference Pareto front, the black dots the non-dominated points returned by qualSolve. Figures (A) and (B) represents the results after 100 and 250 function evaluations respectively, with 13 and 34 non-dominated points respectively.

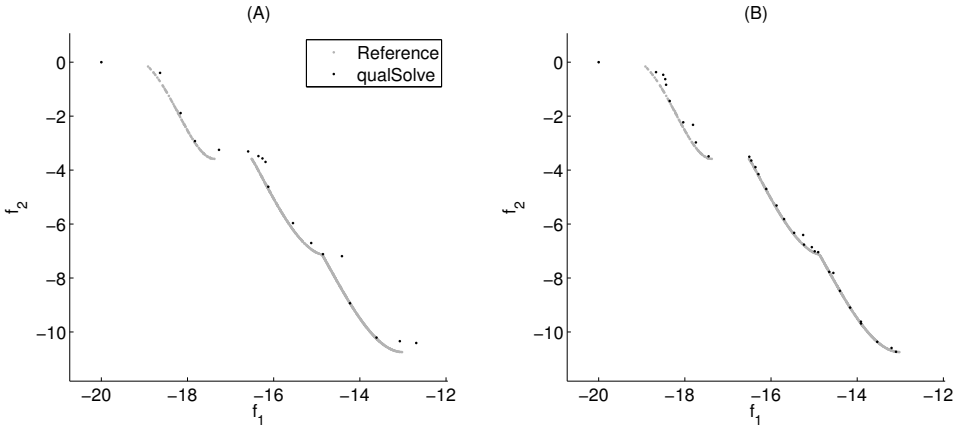


Figure 35: The Pareto fronts of the test function Kursawe. The gray dots represent the reference Pareto front, the black dots the non-dominated points returned by qualSolve. Figures (A) and (B) represents the results after 100 and 250 function evaluations respectively, with 18 and 32 non-dominated points respectively.

5.4.1 Engine optimization at Volvo Powertrain

As mentioned in Section 1, the motivation behind this thesis was a project at GMMC in cooperation with Volvo Powertrain (VPT), Volvo Cars and FCC. Access to engine simulations at VPT was provided in order to try out the algorithm, however the algorithm was actually never used to propose new design parameters for these simulations, since the time ran out. Instead an iteration of the algorithm was run manually resulting in points proposed for simulation (at the time of writing yet to be simulated). To recapitulate, an engine for the next generation of trucks is in the design and the problem to be addressed is that of minimizing the fuel consumption of the diesel engine. To this end a simulation of a sector of the combustion chamber has been created at VPT using the simulation software STARCD. By running the simulation, which takes around 40 hours,

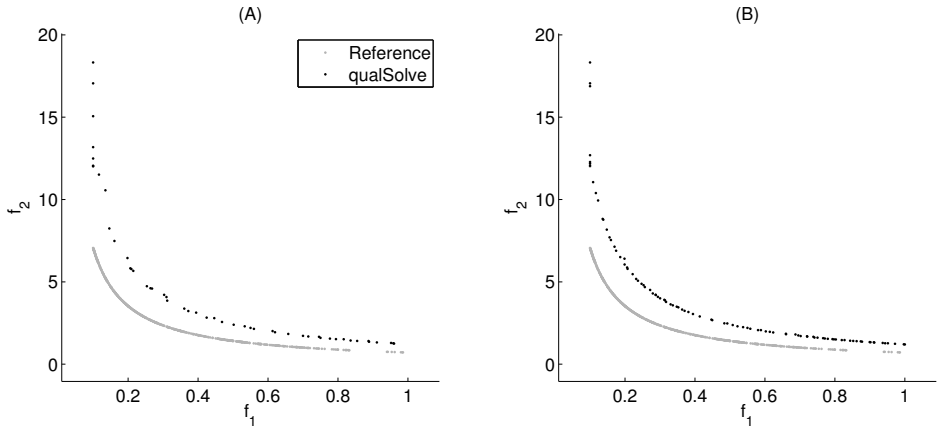


Figure 36: The Pareto fronts of the test function Deb bimodal. The gray dots represent the reference Pareto front, the black dots the non-dominated points returned by qualSolve. Figures (A) and (B) represents the results after 100 and 250 function evaluations respectively, with 49 and 93 non-dominated points respectively.

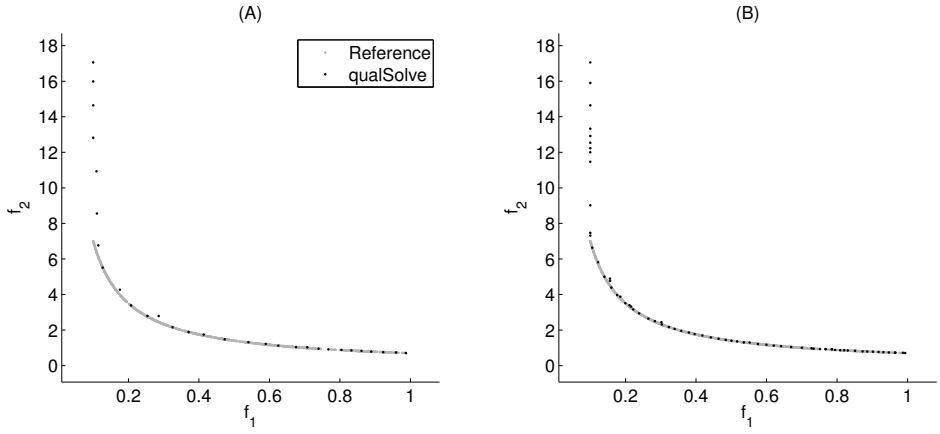


Figure 37: The Pareto fronts of the test function modified Deb bimodal. The gray dots represent the reference Pareto front, the black dots the non-dominated points returned by qualSolve. Figures (A) and (B) represents the results after 100 and 250 function evaluations respectively, with 30 and 70 non-dominated points respectively.

fuel consumption, soot and NOx[18] emissions may be calculated for different load cases. Two out of 13 different load cases are considered, the chosen two being representative for the other 11. A load case is a typical mode of driving; a specific case may for instance represent driving with a fully loaded trailer at cruising speed on a flat highway, or scaling a steep climb with a fully loaded trailer. Naturally these cases give rise to different amounts of emissions and fuel consumptions and their respective optima will possibly not coincide, so the algorithm needs to be able to handle multiple objective functions. There are six design variables available for tuning, four describing the shape of the piston head and two describing the fuel injector nozzle. All four variables describing the piston head are continuous and subject to a geometrical constraint (the design has to be feasible); the volume of the cylinder at top dead center, when the piston is as close to the top as possible, must correspond to the volume at bottom dead center in such a way that a specified compression ratio

[18]NOx is a compound name for NO_1 and NO_2.

is achieved. If the geometrical constraint is not fulfilled, the compression ratio specified will not be attainable. One of the two variables describing the nozzle is binary, the other one is continuous, and both of them together with the four other variables are subject to constraints on the emissions of soot and NOx. These constraints are legislative, so that the trucks may be sold at the American market.

When creating the mesh for the simulation, there is a minimum resolution involved. If, for instance, one of the variables is measured in the unit of millimeter and the variable changes one micron, the mesh and the resulting output of the simulation will remain unchanged (aside from numerical noise). This minimum resolution also comes into play when the design is to finally be implemented, since the production line itself will have some minimum tolerance.

To summarize, the features of this particular simulation are:

1. non-convex but (probably) continuous
2. no analytical derivatives
3. expensive to evaluate
4. noise in function data
5. minimum resolution of variables
6. multiple objectives (two load cases)
7. subject to box and non-linear constraints
8. low order of dimension in the parameter space

The decision maker has been one of the representatives at Volvo Powertrain, and has indicated several ways of making the optimization procedure more efficient. As for the fuel consumption, not all points on the Pareto front are of interest: areas exceeding a certain value in one of the objectives is eligible to be cut away and thus saving computational time by not treating them as "relevant" areas of the Pareto front. All levels of emissions are acceptable as long as they are below the legislative constraints. However, there may be value in a solution that combines both a low fuel consumption and low emissions, since this leaves room for improvements elsewhere in the design (such as in the aftertreatment of the exhaust). Hence, there are two possible (and not equivalent) ways of formulating the optimization problem. The first is

$$
\begin{aligned}
\min_{\boldsymbol{x}} \quad & \left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\right)^{\mathrm{T}}, \\
\text{s.t.} \quad & g_{11}(\boldsymbol{x}) \leq c_{11}, \\
& g_{12}(\boldsymbol{x}) \leq c_{12}, \\
& g_{21}(\boldsymbol{x}) \leq c_{21}, \\
& g_{22}(\boldsymbol{x}) \leq c_{22}, \\
& h((x_1, \ldots, x_4)^{\mathrm{T}}) \leq d, \\
& (x_1, \ldots, x_5)^{\mathrm{T}} \in \Omega, \\
& x_6 \in \Omega_b,
\end{aligned}
\tag{41}
$$

where $g_{i1}(\boldsymbol{x})$ and $g_{i2}(\boldsymbol{x})$ are the amount of soot and NOx in the exhaust of simulation i; constants c_{i1} and c_{i2} define the legislative bounds on soot and NOx; Ω is the domain of the first five variables; Ω_b is the domain of the binary variable x_6; and $h : \Omega \to \mathbb{R}^+$ is a function calculating the minimum volume, with d as the maximum minimum cylinder volume at top dead center. The non-linear constraints described by the function g_{ij} are in themselves expensive to evaluate as, they are outputs from the simulations. As such they also share the above mentioned features, making them hard to handle. This optimization problem would have two objective functions, six decision variables and

five non-linear constraints, four of which are "soft" (that is, the simulation does not crash if violated) and one is "hard". Furthermore, being expensive, the four legislative constraints will have to be interpolated just as the objective functions. This means that if the interpolations of the constraints are inaccurate some possibly important areas of the decision space may be cut away. These constraints cannot just be passed on to the external optimization routines for this reason, instead they will have to be handled through a set of penalty functions.

The alternative formulation is to regard also the legislative constraints in the form of objective functions (of course, subject to the same constraints as before, by cutting away parts of the Pareto front as in the case of fuel consumption), since there is also a value in solutions where both fuel consumption and emissions are low. The alternative formulation follows:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), g_{11}(\boldsymbol{x}), \ldots, g_{22}(\boldsymbol{x}))^{\mathrm{T}}, \\ \text{s.t.} \quad & h((x_1, \ldots, x_4)^{\mathrm{T}}) \leq d, \\ & (x_1, \ldots, x_5)^{\mathrm{T}} \in \Omega, \\ & x_6 \in \Omega_b. \end{aligned} \tag{42}$$

This formulation instead has six objective functions, but no expensive constraints that need to be explicitly interpolated and handled.

Common to both formulations is the binary constraint on x_6. A possiblity would be to handle it as a nonlinear constraint, but would result in problems when applying a global solver to the auxilary optimization problems such as (30). The solver will have a rough time getting feasible starting points, since the domain Ω has one dimension less than the box constraints making it difficult to locate feasible points without furhter information. But, since there is only one binary variable, each auxiliary problem is instead solved twice, once for each value of the binary variable. The resulting values are then compared and the best one selected.

Formulation (41) was chosen partly because with six objective functions, a larger number of function evaluations is needed, and partly because the current implementation of qualSolve does not support more than two objective functions. As was mentioned, the algorithm was never used directly. Instead a manual procedure was employed in order to find interesting points, by running the quality function manually. Interpolations were created for all functions f_i and g_{ij}, $i, j \in \{1, 2\}$, and a set of penalized functions $\tilde{f}_i$ were created:

$$\tilde{f}_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) + \Gamma_h(\boldsymbol{x}) + \ldots + \Gamma_{g_{22}}(\boldsymbol{x}), \qquad i = 1, 2,$$

where

$$\Gamma_p(\boldsymbol{x}) = \mu_p \max\{p(\boldsymbol{x}) - c, 0\}^2,$$

and $p : \Omega \times \Omega_b \to \mathbb{R}$ is the constraint function, c its upper bound and $\mu_p > 0$ a constant depending on the constraint function. Both objective functions f_1 and f_2 were penalized by all constraint functions in this way, and then subjected to MultiOb, once correct values of the μ-parameter were found. In this way, a set of feasible points that could possibly improve upon current results were found. This set is indicated as the cluster of points in Figure 38. The points in the upper right part of the figure gives a feeling of how the feasible objective region may look. As can be seen, the region is almost arrow-shaped, leaving very little space for the Pareto front. At the time of writing the results from the simulations at Volvo were yet to be reported.

5.4.2 Optimization of a heater element

Finally, a comparably fast and simple simulation in COMSOL Multiphysics was used to evaluate the algorithm; one simulation run took about two minutes. The simulation models a straight pipe through which a fluid is forced, and in the pipe a heater element is mounted, see Figure 39. The heater element has the form of an ellipsoid whose equatorial and polar radii are variable. When a fluid is forced through the pipe, two things occur: a) the pressure will be lower at the end of the

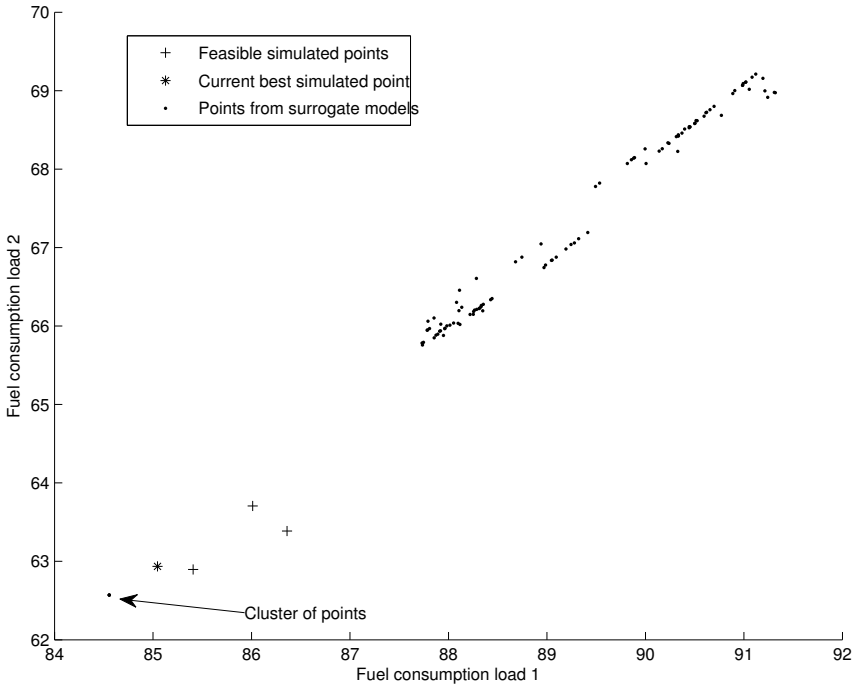


Figure 38: The feasible function space of the fuel consumption for the two load cases in the Volvo project. The crosses mark feasible, previously simulated points, and the asterisk is the best feasible simulated point. The dots indicate points obtained from surrogate models (created from about 50 simulated points). The cluster of points indicates possibility for improvements. To comply with requests from Volvo, the values of fuel consumption have been altered in the figure.

pipe than at the beginning; and b) the temperature will be higher at the end of the pipe than at the beginning. The direction of the fluid determines the meaning of the words “end” and “beginning” as used in this context.

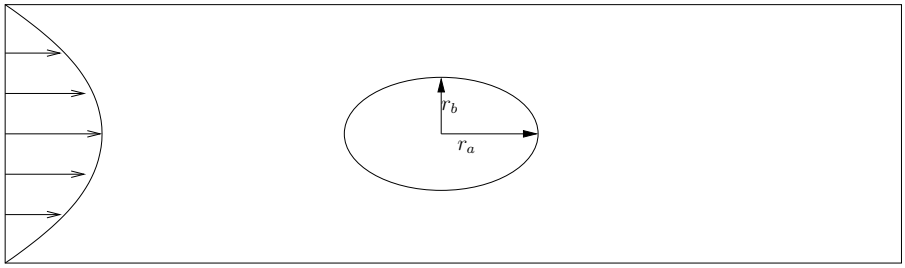


Figure 39: The pipe with the heater element. Fluid is forced from the left to the right, the temperature of the fluid rises and the pressure drops as it passes the heater.

Two objective functions are defined from the output of the simulation, and the following opti-

mization problem is formulated:

$$\begin{aligned} \min_{(r_a, r_b)^{\mathrm{T}}} \quad & (\Delta p, -\Delta T)^{\mathrm{T}}, \\ \text{s.t.} \quad & 0.003 \leq r_a \leq 0.020, \\ & 0.003 \leq r_b \leq 0.020, \end{aligned}$$

where Δp is the pressure drop and ΔT is the increase in temperature. In a word, the objective is to minimize the pressure drop and to maximize the increase in temperature. In this setting, the performance of qualSolve was compared to that of MultiOb. Both algorithms were run for 200 function evaluations, and the results can be seen in Figure 40. What is notable is that qualSolve covers a larger part of the Pareto front than MultiOb, while they perform more or less equally well in the enlarged part (to the right in Figure 40). In all fairness it should also be stated that 200 function evaluations is far to small a number for MultiOb. For qualSolve, 200 evaluations are more than necessary in this case.

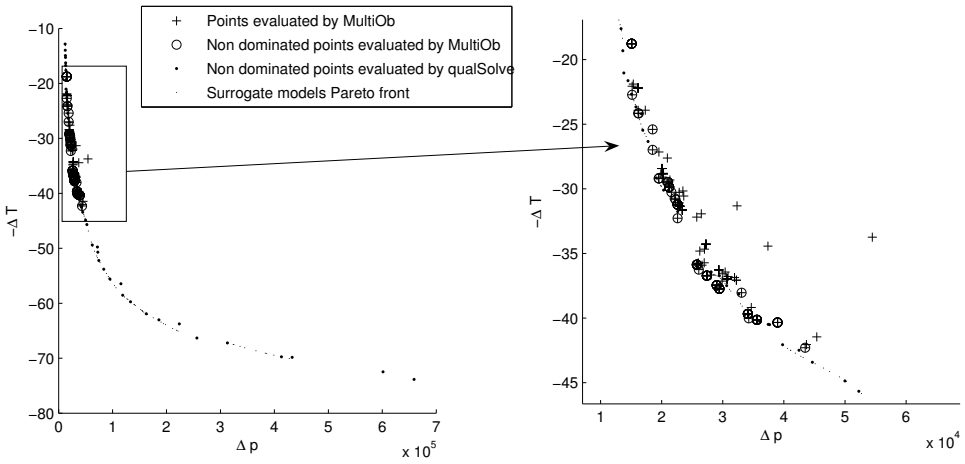


Figure 40: Result of the heater simulation. 200 evaluations performed by MultiOb and qualSolve each.

6 Discussion and future research

This section will present a discussion on the pros and cons of the proposed algorithm. Possible improvements as well as more rigorous tests are suggested.

6.1 Empirical comparisons of test functions and simulations

From the numerical results in the previous section, it is clear that qualSolve shows promise. Compared to commercial solvers in the same category, it performs quite well. Speed of convergence is however not the only thing to look for in an algorithm; even more important is the concept of *robustness*. Robustness in this sense means that the algorithm should be able to perform well even though the initial points are changed. To have the outcome being totally dependent on the choice of initial points bad, especially when considering real-life applications, when you perhaps only have one shot at optimizing before your time runs out. Robustness by itself, without any particular speed of convergence, is however equally bad.

Further comparisons of the algorithm should be performed, particularly in the field of multiobjective optimization. There are however few other algorithms in the field of expensive multiobjective optimization, we have been able to find only one, called ParEGO [Kno06], an adaptation of EGO (see [JSW98]) to multiple objectives. When considering expensive and noisy multiobjective optimization, no previous research has been found. Naturally, there is also a lack of tools for comparisons. The field is interesting as well as relevant, since there is no shortage of expensive and noisy multiobjective optimization problems in the industry. Regarding single objective optimization of expensive and noisy functions, we also had trouble finding algorithms for comparison. SKO, see [HANZ06], seems to be the only one developed to handle this particular type of problem. It would be interesting to perform more rigorous comparisons to this algorithm.

A more ambitious study of different simulation software and their features would also be beneficial. It is likely that there exists more features than the ones considered in this thesis that need to be addressed in order to model and optimize the simulation correctly. Also, such a study could possibly lay the ground for some general guidelines for parameter settings of the algorithm. Such a guideline would be of particular interest for the $\boldsymbol{\sigma}$-vector, controlling the balance between global and local search. When the stopping criterion is a number of function evaluations, say N, and no further function evaluations can be afforded, the best idea may be to construct a strictly increasing $\boldsymbol{\sigma}$-vector of length $N - N_i$, where N_i is the number of initial points. In this way the algorithm will start out with a bias towards global search and, as the iterations proceed, swing the bias towards local search.

6.2 The connection with simulations

One may always discuss how well the surrogate models correspond to the objective functions considered, and how they may be further improved. An important issue that has not been as thoroughly addressed as it demands, is the issue of correlated noise in the objective function. The current implementation only handles independent noise. The assumption has been that this kind of noise arises out of numerical errors or discontinuities in the mesh of the simulation and is not possible to predict. However, noise arising from mesh discontinuities is probably not independent, but has some correlation since a remeshing only occurs when some design parameter shift becomes large enough. If the noise becomes correlated, there is a possibility that the approximation may predict a value that is actually noise and not a true function value, giving a poor estimate of η when cross-validating (see Section 3.1.3). This may be remedied by removing neighbouring points to the point currently subject to prediction as well, removing the chance that correlated noise predicts the removed point. Formally, it means replacing problem (26) by

$$\begin{aligned} \min_{\beta} \quad & \sum_{i=1}^{n} (S_{\{X \setminus B(\boldsymbol{x}_i, d)\}}(\boldsymbol{x}_i) - f_i)^2, \\ \text{s.t.} \quad & \beta \in (0, 1], \end{aligned}$$

where

$$B(\boldsymbol{x}_i, d) = \{\boldsymbol{y} : \|\boldsymbol{y} - \boldsymbol{x}_i\| < d, \quad \boldsymbol{y} \in \Omega\},$$

and d is the maximum distance of correlation. Unfortunately, while removing the need to manually estimate η, this method introduces the need to estimate d instead. However, this parameter may be estimated through a simple parameter-study of the objective function or knowledge about the mesh size used in the simulation.

Common to most computer simulations may be the possibility to trade accuracy versus speed (for instance, by varying the mesh size). This could be seen as having one low-fidelity model and one high-fidelity model, and perhaps some model in between. These models give rise to different uncertainties in their evaluated points, the high-fidelity model having the lowest uncertainty of course. An interesting idea would be to allow the algorithm to controll the level of fidelity for the simulation performed. The reason is that while performing global searches less accuracy is required than during local searches, since most global searches are performed in areas of low intrest to the

optimization procedure. While this could enable faster global searches, it would require a surrogate model constructed of evaluated points containing different amount of noise. Furthermore, the uncertainty function (28) would have to be modified to take this into consideration.

Another issue that arose out of the Volvo Powertrain engine simulation was the necessity to "cut" away parts of the Pareto front. When considering the Pareto front made up of objective functions describing for instance the fuel consumption of the different load cases, some previously best values were known, and consequently any values higher than these best values were not of interest. This is in contrast to the higher values still being on the Pareto front; from this point of view all points on the Pareto front are *not* equally good. Hence, something has to be done. One way would for instance be to filter the resulting points coming from MultiOb, before creating the distance interpolation for $\mathrm{dist}_{Z_S^*}$, and in some way making the unwanted parts of the front appear to be further away than they really are.

6.3 Improvements of the current implementation

There are of course many things in the current implementation that can be improved upon. Here follow a few that for different reasons have not been included.

Quality function improvements

During the construction of the quality function, the measure of uncertainty defined in (28),

$$U_X(\boldsymbol{y}) = \min_{\boldsymbol{x} \in X} \|\boldsymbol{x} - \boldsymbol{y}\|,$$

was used. A more satisfactory measure of uncertainty would be to use the power function (Definition 24). The downside to the latter approach is that the speed of the algorithm would suffer since the power function is more complex to calculate than the above expression and the possibility of obtaining a smaller integration domain (see Section 4.2.2) would disappear.

The numerical integration of $Q(\boldsymbol{y})$ could also be improved upon in a variety of ways. The current implementation is probably the most straight-forward one, but may not be optimal with regard to speed and accuracy. This is also where most time is spent as dimensionality increases, currently limiting the implementation to at most a six dimensional parameter space. One strategy is simply to use a better approximation technique than that of trapezoidal numerical integration, for instance a quadratic approximation. In this way fewer subintervals can be used while maintaining accuracy, consequently increasing the speed of the algorithm. One may also consider Monte-Carlo integration, a numerical integration method which does not suffer from an exponential increase in computational time. A third way may be to exploit the structure of the integrand since everything about it, such as first and second derivatives, is known. This could, for instance, be used to create a Taylor expansion of the integrand, potentially leading to simplifications. The domain of integration may also be decreased, since the current implementation extends it to be rectangular, when it really only needs to be the smaller polytope contained in the rectangle.

Extending the distance function to more than two objectives

The $\mathrm{dist}_{Z_S^*}$-function, used in the weight function ω for multiobjective optimization (32), is currently limited to only two objective functions. This limitation is strictly an implementation issue, the theory is independent of the number of objectives. The $\mathrm{dist}_{Z_S^*}$-function is currently merely an interpolation trained on the result of repeated solutions to a specific auxiliary distance optimization problem. These training points, as they are called, are currently generated quite oblivious of the actual appearance of the objective function domain. The training points are generated as an equally spaced grid, with 20 points in every dimension, generating 400 points for two objective functions, 8000 points for three and so on. Six objective functions would require 64 million points, using this method. The rationale for choosing such an approach from the beginning was that 400 points seemed to create a stable enough interpolation. Too few points generated a "bumpy" interpolation

which sometimes gave negative values (something which is not possible), while too many points were: a) not practically possible to interpolate[19] and b) not practically possible to solve the auxiliary distance problem for. Clearly, the quality of the underlying interpolation for the distance function has to be preserved when increasing the number of objective functions. This may be achieved in several different ways.

One way is to exploit the domain of the objective functions in a better way, by not selecting training points outside the feasible objective region Z, since the distance function will not be used there anyway. Depending on the objective functions, this may limit the space necessary to interpolate and fewer points may be used. These points may for instance be generated by some space-filling technique, such as a latin hypercube design. It should also be possible to exploit the general appearance of a Pareto front to rationalize the generation of training points, since the distance varies linearly in different parts of the objective function domain, particularly at points far away from the front itself.

The question of what basis function to use is also relevant. The current implementation uses thin plate splines for all interpolations, but a simple linear interpolation could perhaps be more beneficial, especially when it comes to extending the Pareto front. The extension of the Pareto front is a linear curve made up by additional points not actually on the Pareto front.

One additional problem when extending the implementation to more than two objective functions is the problem of bridging larger gaps in the front. Large discontinuities have to be filled using additional points, similar to those extending the Pareto front, in order to get a stable interpolation of the curve. This is trivial with two objective functions, but becomes more involved as the number of objectives increase.

Additional radial basis functions and transformations

As was just mentioned, the only RBF implemented is the thin plate spline. There is no reason not to implement more basis functions, and to investigate their strengths and weaknesses. Some may work better than others in higher dimensions, for instance. Along these lines, work may also be done in the area of transformations, both with respect to the parameter and the objective spaces. For instance, a logarithmic transformation of the objective function values is not the only transformation possible; other transformations include $T_{\boldsymbol{f}}(y) = \frac{1}{y}$. As in the EGO algorithm, see [JSW98], the choice of transformation can be made by cross-validating the surrogate model. In the parameter domain, normalizing the domain to the unit hypercube may not always be the best idea. Sometimes the objective function may be more active (display more oscillatory behavior) in one dimension than the others, this dimension could then be "stretched", so as to make the function more equal in the different dimensions.

Choosing the initial point strategy

The subject of initial points is important and there seems to be different opinions about the best choice of these points. Some articles propose quite a large number of initial points that rise quickly with the number of dimensions, for instance $11d$ or 2^d. Since the objective functions are expensive, 100–200 evaluations can be performed at most, a large number of initial points means that a large amount of simulation time is spent on evaluating these. When using radial basis functions as bases for the interpolation, there is a minimum number of initial points needed to create the interpolation. This number is $d + 1$ when using RBFs of order 2, the number of points needed to create a simplex. Starting with such a comparatively low number of points seems to work quite well for qualSolve, but not so well for other algorithms. Number of points aside, how should the points be selected? Should they for instance be as far away from each other as possible?

[19] The major matrix A of an interpolation requires $8n^2$ bytes of memory, where n is the number of interpolated points.

Evaluating the objective function in parallel

Often a cluster of computers is available for performing simulations, capable of running multiple simulations in parallel. The current implementation of the algorithm evaluates the objective function sequentially, and needs to be adapted to exploit the benefits of parallel evaluation. This can be done by using the surrogate model to provide "fake" objective function values and then later replacing these values with real ones. This implies trusting the surrogate model more than usual, and can result in sub-optimal choices of points. This is however still more beneficial, with respect to time, than not utilizing the cluster's capacity at all. If the cluster is able to run N simulations in parallel, run the algorithm for $N-1$ iterations, and evaluate the surrogate model S instead of the real objective function. At the N:th iteration, send the N chosen points to the cluster, wait for the results and replace the fake values with the real function values.

Non-linear constraints

The handling of non-linear constraints in the implemented algorithm is done by simply passing them on to the external solvers used for solving the "cheap" optimization problems. Considering the quality function (Definition 27),

$$Q(\boldsymbol{y}) = \int_{\Omega} (U_X(\boldsymbol{x}) - U_{X\cup\boldsymbol{y}}(\boldsymbol{x}))\omega(S(\boldsymbol{x}))dV(\boldsymbol{x}), \tag{43}$$

and the implementation of the integral described in Section 4.2.1 (which is to integrate over the domain inside the box constraints) leads to integrating over regions not belonging to Ω. In practice this means that minimization of uncertainty in regions outside the feasible domain is considered. Although passing the constraints onto the external solvers will prevent the choice of infeasible points for evaluation, this will also make it very attractive to choose points on the border of the non-linear constraints. This is an unsatisfactory behavior. The non-linear constraints should be handled in a more direct way, so that the integration in (43) is truly performed over Ω and not the whole hypercube befined by the box constraints. One way to accomodate this demand would be to add a penalty function $\psi(c(\boldsymbol{x}))$, directly to the objective function. Considering a constraint on the form $c(\boldsymbol{x}) \leq 0$, a choice of external penalty function can, for instance, be:

$$\boldsymbol{x} \mapsto \psi(\boldsymbol{x}) := \begin{cases} 0, & \boldsymbol{x} \leq 0 \\ \boldsymbol{x}^2, & \boldsymbol{x} > 0 \end{cases}$$

Adding a penalty to the objective function can be done without changing the algorithm, but will lead to different penalties depending on the choice of σ in equations (31) and (32). A more satisfactory method would be to modify the weight function to include a contribution from the non-linear constraints. For instance, by defining:

$$\tilde{\omega}(\boldsymbol{S}(\boldsymbol{x}), c(\boldsymbol{x})) := \omega(\boldsymbol{S}(\boldsymbol{x})) \exp(-\nu\psi(c(\boldsymbol{x}))),$$

where c is the non-linear constraint, ψ a penalty function and $\nu \geq 0$ a penalty parameter. The new weight function, $\tilde{\omega}$, would handle non-linear constraints by making the integrand vanish in infeasible areas.

Finally, a comment on a type of constraints that was discovered during the modelling of the Volvo problem (see Section 5.4.1). A type of *expensive* non-linear constraint arose, as the legislative constraints on soot and NOx emissions. The simulation had to be run in order to evaluate the constraints, hence the term "expensive". These were handled by creating surrogate models of them, and then using penalty methods based on these surrogates. Although it seemed to work, the approach could lead to treating feasible areas as infeasible, since uncertainty is present in the constraints. Therefore, a better way of handling them could be to connect the amount of penalty to the certainty in the constraint surrogate model. That is, to let the ν parameter decrease with increasing $U_X(\boldsymbol{x})$, hence decreasing the probability of penalizing areas where the surrogates have high uncertainty and adding more penalty where uncertainty is low.

6.4 Convergence

When dealing with global optimization of non-linear black box functions, proving meaningful convergence results is very difficult. Little information is available about the problem, mostly limited to continuity and (occasionally) smoothness of the objective function. One type of convergence, that to some extent can be seen as necessary for an algorithm, is the requirement that it should produce a dense sequence of points (see [Gut01, Thm. 4]). The theorem states that an algorithm will converge for every continuous function, if and only if it generates a sequence of points that is dense in Ω. The requirement of a dense set guarantees that the algorithm does not miss a minimum if it is allowed to run for an infinite amount of time. It is fulfilled in qualSolve by the demand on the weight function: $\omega(\boldsymbol{z}) > 0$ for all $\boldsymbol{z} \in Z$. It implies (a more stringent proof is however required, to ensure this) that there cannot exist any parts of Ω without any evaluated points if the algorithm is allowed to run for an infinite amount of time. The reason is that if $D \subset \Omega$ would be such a part, then the evaluated point density in $\Omega \backslash D$ would increase, and after enough iterations Q (see (29)) would obtain its maximum in D.

Although it is important that an algorithm satisfies the requirement of producing a dense set, it is far from sufficient for a good algorithm to do so; for instance an algorithm that randomly distributes points in space would fulfill it. Unfortunately, when dealing with global optimization of black box functions not much more can be proven; if there exists a region that does not contain any evaluated points, there exists a possibility for the function to obtain its minimum there. Hence, there is no real way of knowing if convergence has occurred after a finite amount of steps, if not some sort of restriction is applied to the function. This could for instance be done by setting an upper limit M on the function norm, so that $|f|_{\mathcal{N}} \leq M$. Theorem 25 would then imply that the existence of $\boldsymbol{x}_{\min}$ could be ruled out in a domain $D \subset \Omega$, such that

$$D = \left\{ \boldsymbol{x} \in \Omega : S_X(\boldsymbol{x}) - P_{\Phi,X}(\boldsymbol{x})M > \min_{\boldsymbol{y} \in \Omega} (S_X(\boldsymbol{y}) + P_{\Phi,X}(\boldsymbol{y})M) \right\}. \tag{44}$$

If all parts of the domain except for one point, or a small section, are eliminated in this way convergence is guaranteed. There are two problems with this approach: one is that the algorithm will not produce a dense set and hence violate the condition of the above mentioned theorem; the other is that, for an arbitary black box function, the norm is not known. Too high a norm would yield a very slow algorithm and nothing is gained, whereas too low a norm could yield an algorithm that fails. In order to get around this problem, an approach could be to use some sort of probability techniques in order to estimate, for instance, the norm. An interpretation of the "qualSolve" algorithm in such probability terms could be made by constructing a simple model of the probability density function for $P(\boldsymbol{x}_{min} \in D)$ (or in the multiobjective case, $P(\Omega^* \subseteq D)$):

$$f(\boldsymbol{x}) = \frac{\omega(\boldsymbol{S}(\boldsymbol{x}))}{\int_\Omega \omega(\boldsymbol{S}(\boldsymbol{x}))d\boldsymbol{x}}. \tag{45}$$

The probability of finding the minima inside some region $D \subseteq \Omega$ would then be:

$$P(\boldsymbol{x}_{min} \in D) = \int_D f(\boldsymbol{x})d\boldsymbol{x}.$$

By considering the expected uncertainty at $\boldsymbol{x}_{\min}$ $\boldsymbol{x}_{\min}$ is:

$$\mathrm{E}[U(\boldsymbol{x}_{\min})] = \int_\Omega U(\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x},$$

and using the probability density (45) the maximization of the quality function is interpreted as minimizing the expected uncertainty at $\boldsymbol{x}_{\min}$. This interpretation is somewhat speculative, as the assumption on the probability density function f for $\boldsymbol{x}_{\min}$ is pretty ad hoc. Nevertheless, a possible improvement of the algorithm could be to derive a more theoretically satisfactory density function f (for instance, by including the uncertainty U). Another possibility would be to start from a

probability distribution of the norm and transform this into a probability density for $\boldsymbol{x}_{\min}$ using (44). Still some sort of assumption has to be made and the question is if the distribution of the norm is more easily specified than the above assumption.

As presented in this section, there are many possibilities to show some convergence results for the algorithm. Nevertheless, obtaining a decent result most of the time after not too many function evaluations is probably a more desirable feature than theoretical convergence results, since mostly about 100–200 function evaluations can be afforded. Hence the best motivation for a good simulation-based optimization algorithm is practical use and results on representative test function cases.

7 Conclusion

The proposed algorithm is designed to handle both single and multiple objective functions generated by simulations. It addresses problems in existing single objective algorithms, such as the clustering of points and overemphasizing the border regions (see Section 1.3), and handles single objective functions with or without noise. This is an area in which there is no shortage of previous contributions. Numerical results and comparisons show that the proposed algorithm performs as well or better than existing (commercial) algorithms. The algorithm can also handle multiobjective optimization and has shown good performance on analytical test functions and simulations, although more rigorous tests and comparisons of the multiobjective performance should be carried through. Further, the algorithm can also be applied to noisy multiobjective problems, which is a new research area with no existing algorithms. This is relevant since there is no lack of noisy multiobjective problems arising from simulations in the industry.

The main motivation for this thesis was the project at GMMC resulting in optimization of engine simulations performed at Volvo PowerTrain and Volvo Car Corporation. During the formulation of the optimization problem, features of the simulation were discovered that had not initially been considered (such as expensive constraints, binary variables, the need for "cutting" the Pareto front and discrepancies in the data). The further adaptation of the algorithm to these features required more time than was available, and as a consequence the algorithm was never run on the Volvo simulations. Some progress was however made by manually performing one iteration of the algorithm suggesting a new set of parameters to simulate. At the time of writing, the results of the simulation were not known.

The current implementation should be seen as a prototype, limited to two objective functions and a parameter space of up to six dimensions. Suggested future research includes: handling of more objective functions and larger parameter spaces, improved handling of non-linear constraints and the ability to model correlated noise.

In summary, the proposed algorithm fills a previously empty niche in the world of simulation based optimization, accompanied by promising numerical results. It also constitutes a platform for future research in the area of expensive and noisy multiobjective optimization.

References

[AEP05] N. Andréasson, A. Evgrafov, and M. Patriksson, *An Introduction to Continuous Optimization*, Studentlitteratur, Lund, 2005.

[DS78] L. C. W. Dixon and G. Szegö, *The global optimization problem: an introduction*, North-Holland, 1978.

[GMS05] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: an SQP algorithm for large-scale constrained optimization*, SIAM Review **47** (2005), pp. 99–131.

[Gut01] H.-M. Gutmann, *A radial basis function method for global optimization*, Journal of Global Optimization **19** (2001), pp. 201–227.

[Han06] T. Hanne, *Applying multiobjective evolutionary algorithms in industrial projects*, Multicriteria Decision Making and Fuzzy Systems. Theory, Methods and Applications. (K.-H. Küfer, H. Rommelfanger, C. Tammer, and K. Winkler, eds.), Shaker Verlag, Aachen, 2006, pp. pp. 125–142.

[HANZ06] D. Huang, T. T. Allen, W. I. Notz, and N. Zeng, *Global optimization of stochastic black-box systems via sequential kriging meta-models*, Journal of Global Optimization **34** (2006), pp. 441–466.

[HN99] W. Huyer and A. Neumaier, *Global optimization by multilevel coordinate search*, Journal of Global Optimization **14** (1999), pp. 331–355.

[Jon01] D. R. Jones, *A taxonomy of global optimization methods based on response surfaces*, Journal of Global Optimization **21** (2001), pp. 345–383.

[JPS93] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, *Lipschitzian optimization without the Lipschitz constant*, Journal of Optimization Theory and Applications **79** (1993), pp. 157–181.

[JSW98] D. R. Jones, M. Schonlau, and W. J. Welch, *Efficient global optimization of expensive black-box functions*, Journal of Global Optimization **13** (1998), pp. 455–492.

[Kno06] J. Knowles, *ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems*, IEEE Transactions on Evolutionary Computation **10** (2006), pp. 50–66.

[Koh99] R. Kohavi, *A study of cross-validation and bootstrap for accuracy estimation and model selection*, Proceedings of the 14:th International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann: San Francisco, 1999, pp. 1137–1145.

[LTT00] R. M. Lewis, V. Torczon, and M. W. Trosset, *Direct search methods: then and now*, Journal of Computational and Applied Mathematics **124** (2000), pp. 191–207.

[Mie99] K. Miettinen, *Nonlinear Multiobjective Optimization*, International Series in Operations Research & Management Science, vol. 12, Kluwer Academic Publishers, Boston, MA, 1999.

[NAS02] H. Nakayama, M. Arakawa, and R. Sasaki, *Simulation-based optimization using computational intelligence*, Optimization and Engineering **3** (2002), pp. 201–214.

[RS05] R. G. Regis and C. A. Shoemaker, *Contrained global optimization of expensive black box functions using radial basis functions*, Journal of Global Optimization **31** (2005), pp. 153–171.

[RS07] ———, *Improved strategies for radial basis function methods for global optimization*, Journal of Global Optimization **37** (2007), pp. 113–135.

[Sch91] L. L. Schumaker, *Recent progress on multivariate splines*, The Mathematics of Finite Elements and Applications, VII (Uxbridge, 1990), Academic Press, London, 1991, pp. 535–562.

[Sho85] N. Z. Shor, *Minimization Methods for Nondifferentiable Functions*, Springer Series in Computational Mathematics, vol. 3, Springer-Verlag, Berlin, 1985.

[SLK05] A. Sóbester, S. J. Leary, and A. J. Keane, *On the design of optimization strategies based on global response surface approximation models*, Journal of Global Optimization **33** (2005), pp. 31–59.

[Wen05] H. Wendland, *Scattered Data Approximation*, Cambridge Monographs on Applied and Computational Mathematics, vol. 17, Cambridge University Press, Cambridge, 2005.

[YLS00] K. Q. Ye, W. Li, and A. Sudjianto, *Algorithmic construction of optimal symmetric Latin hypercube designs*, Journal of Statistical Planning and Inference **90** (2000), pp. 149–159.

[Zit99] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: methods and applications*, Ph.D. thesis, ETH Zurich, Switzerland, 1999.

[ZTL+03] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, *Performance assessment of multiobjective optimizers: an analysis and review*, IEEE Transactions on Evolutionary Computation **7** (2003), pp. 117–132.