Master of Science Thesis

Assessing and developing optimization methodologies for practical engineering design with high-fidelity CFD simulations

Iliass Azijli

June 10, 2011

Assessing and developing optimization methodologies for practical engineering design with high-fidelity CFD simulations

Iliass Azijli

Submitted in partial fulfillment of the requirements for the degree of Master of Science

June 10, 2011

Faculty of Aerospace Engineering · Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY
FACULTY OF
AEROSPACE ENGINEERING
AERODYNAMICS CHAIR

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance the thesis entitled **"Assessing and developing optimization methodologies for practical engineering design with high-fidelity CFD simulations"** by **Iliass Azijli** in fulfillment of the requirements for the degree of **Master of Science**.

Dated: June 10, 2011

Exam committee:

prof. dr. ir. drs. H. Bijl

dr. ir. P.A. Cusdin

dr. R.P. Dwight

dr. S.J. Hulshoff

Preface

This work, carried out at the CFD department of Lotus Renault GP, finalizes my studies at the Aerospace Engineering faculty of Delft University of Technology. I joined the company in 2009 to do a four months internship at the Aerodynamics department. After this, I did a six months internship at the CFD department, followed by the final thesis research, which is the work you are currently reading.

There have been a number of notable players during my stay here whom I would like to thank explicitly. The whole thing started with me sending a letter to the head of aerodynamics Dirk de Beer, asking whether I could do an internship here. I am truly grateful to him for opening the gates to me. During my stay at the Aerodynamics department I was supervised by Hari Roberts. I really enjoyed the project that he had assigned to me; analyzing wind tunnel data and track data and trying to combine the two into something special. That was a real challenge! Not only did I learn a lot about the aerodynamic development process itself, I also learned a great deal about the dynamics of Formula One cars. I think he was a great supervisor because he also gave me an insight into the way he approaches problems and how he comes up with interesting new ideas to solve them.

When I moved to the CFD department, I was assigned to work in the rear team, which focuses on developing the rear part of the car. I was supervised by Kristofer Midgley, who guided me through the whole CFD process. I have learned a great deal about aerodynamics from him. The main project I was assigned to work on was the initial development of the F-duct. This project has truly been a unique experience where I witnessed the whole development process: from the initial development of the concept in CFD to the point of testing the part in the wind tunnel; from seeing it being produced at the manufacturing department and fitted at the race bay to watching the end result on TV.

With regards to the present work, I would like to thank Paul Cusdin, my supervisor here, and Richard Dwight, my supervisor at Delft. They both have been very supportive and enthusiastic throughout this process. I would also like to thank Steve Fenwick, who helped me to get to grips with the complete iSIGHT framework and helped me improve KERS. I would also like to thank my college mate Marius Birsanu for supplying his LaTeX thesis template and answering all my LaTeX related questions.

Last but certainly not least, I would like to thank Jarrod Murphy, the head of this department. He allowed me to do the internship and my MSc thesis here. Furthermore, I worked closely with him on the F-duct project, which I really enjoyed.

Iliass Azijli
Enstone, June 10, 2011

Abstract

In modern day Formula One, aerodynamics has become the key performance differentiator. The two tools used for the aerodynamic design are the wind tunnel and Computational Fluid Dynamics (CFD). For a rapid development rate, it is crucial to use these tools as efficiently as possible. The goal of this thesis is to improve the optimization methodology at the CFD department of Lotus Renault GP (LRGP). This is a challenge because CFD simulations are expensive; they require a relatively large amount of wall clock time to complete and in addition there is a restriction on the number of runs that can be carried out. This restriction was agreed upon between the Formula One teams to reduce the cost of the sport.
The pre-existing optimization methodology makes extensive use of the design of experiments (DOE) approach, in which the design space is filled with a set of points that are run in CFD. The results are then analyzed and the best design is chosen, possibly followed by tweaking it manually to improve the performance.
Two approaches are investigated to improve upon this methodology and two CFD test cases are set up to assess them. The idea of the first approach is to make use of local optimization algorithms. Gradient-free and gradient-based algorithms are investigated. SQP (Sequential Quadratic Programming) methods are found to be the most efficient solver.
The second approach is a surrogate modeling framework. Given the DOE results, the idea is to build a cheap surrogate model of the expensive function. The optimization is then carried out on this model while the original function is evaluated at each iteration to update the surrogate. The surrogate modeling technique used is Kriging, which is an interpolating method. Since CFD data can contain a level of numerical noise, there will be cases where it would be preferred to regress the data instead. The original method is modified to be able to do this. Additionally, a method which only regresses points within a user-defined threshold is introduced and investigated. The test cases show that they indeed improve the traditional interpolating method. Furthermore, an alternative technique to building a Kriging surface is introduced. It is based on minimizing the cross-validation error rather than maximizing the likelihood function. For sparse data sets, it is found to produce more accurate response surfaces. To avoid getting stuck in a local minimum, the algorithm EGO is implemented.
Comparing the local solvers to surrogate modeling, the latter can provide the user with a better understanding of the design problem. It also gives the user more control over the optimization and can supply better insights if it fails.
The surrogate modeling approach is applied to a couple of actual projects carried out at LRGP's CFD department and proves to be a good extension to the current methodology. Not only does it manage to find improved designs, it also helps in providing a better understanding of the design problems.

Nomenclature

Abbreviations

AMGA	archive-based micro genetic algorithm
AoA	angle of attack
BFGS	Broyden-Fletcher-Goldfarb-Shanno
bsf	best so far
CAD	computer aided design
CFD	computational fluid dynamics
CG	conjugate gradient
DACE	design and analysis of computer experiments
DIRECT	DIviding RECTangles
DOE	design of experiments
DS	downhill simplex
EGO	efficient global optimization
F1	Formula One
FIA	Fédération Internationale de l'Automobile
FOTA	Formula One Teams Association
HJ	Hooke-Jeeves
ib	in-board
KERS	Kriging and Exploring Response Surfaces / Kinetic Energy Recovery System
LRGP	Lotus Renault GP
LSGRG	large scale generalized reduced gradients
MLE	maximum likelihood estimator
MMFD	modified method of feasible directions
MOST	multi-functional optimization system tool

NLP	nonlinear programming
ob	out-board
QP	quadratic programming
RAAE	relative average absolute error
RBF	radial basis function
RMAE	relative maximum absolute error
RWMP	rear wing main plane
Sp	starting point
SQP	sequential quadratic programming
TRW	top rear wing
Xval	cross-validation

Greek Symbols

α	angle of attack
β	polynomial coefficient
Δx	finite difference step size
ϵ	modeling/correlation error
γ	airfoil camber
$\hat{y}$	prediction of y
λ	lagrange multiplier
μ	penalty parameter (constraint handling) / regression term (Kriging)
Φ	normal cumulative distribution function
ϕ	normal probability distribution function
ϕ_1	ℓ_1 exact penalty function
ψ	concentrated log likelihood function (dacefit)
ρ	regularization constant (Kriging)
σ^2	variance
θ	correlation parameter (Kriging)

Latin Symbols

$\mathbf{p}$	search direction vector / smoothing parameter
$\mathbf{r}$	correlation vector
$\mathbf{x}$	design vector
C	lower triangular square matrix (Cholesky factorization)
c	constraints / chord length
C_D	drag coefficient
C_L	lift coefficient
C_M	pitching moment coefficient
$E\left[I\left(\mathbf{x}\right)\right]$	expected improvement
F	$m \times t$ regression matrix (Kriging)
f	objective function
h	number of polynomial coefficients
K	lipschitz constant
L	Lagrangian function
m	number of sample points
n	number of design variables
R	$m \times m$ correlation matrix
S	$m \times n$ matrix containing the sample data variables
s	root mean squared error (Kriging)
t	number of regression terms (Kriging) / airfoil thickness
u	unit round-off error
X	horizontal position
Y	m-sized vector containing the sample data output / vertical position
y	real response

Contents

Appendices

'Most experts suggest that one should open with a joke. Obviously, they've never heard me tell a joke.'

Anonymous

1 Introduction

Formula One is widely considered to be the pinnacle of motorsport. Each competing team is required to build its own chassis, which should satisfy a set of rules specified by the FIA, the sport's regulating body. The fascinating aspect of the sport is that there are two levels of competition. In the foreground, there are the twenty-odd races held across the world and watched by millions of people. In the background, there is the technical development race between the teams. The astute observer will notice that during the course of a season, the cars are constantly evolving.
Of all the components on the car, the aerodynamic surfaces receive the most attention. It is often stated that the four main ingredients that make a modern day Formula One car fast are the driver, engine, tires and the aerodynamics. Nowadays, since the engines are homologated[1] and because there is only one tire manufacturer, it should be no surprise why the aerodynamics have become so important. The two tools used in developing the aerodynamics of the car are the wind tunnel and Computational Fluid Dynamics (CFD). Each team invests heavily in these tools and nowadays, virtually every team has its own wind tunnel and CFD resources.
The situation at Lotus Renault GP[2] (LRGP) is no different. A major step forwards was the construction of the 'Computational Aerodynamics Research Center' in 2008. Prior to this, the wind tunnel and CFD aerodynamicists worked in the same building, which houses the wind tunnel. The construction of the center enabled the team to increase its number of CFD aerodynamicists and also expand its computational power. To carry out the demanding CFD computations the team owns an Xtreme-X2 supercomputer which has in excess of 4000 cores and 8Tbytes of memory. At its peak, it can provide a performance of 38TFlops. Next to all these hardware updates, the software used is frequently updated too. Not only are the commercial packages regularly updated to their latest versions, the in-house developed codes are constantly refined and improved.

[1] This basically means that the engine development that competing engine manufacturers are allowed to carry out has been reduced to a minimum.

[2] The Enstone based team has been re-branded to *Lotus Renault GP* as of the 2011 season. Since 2002, the team had competed as *Renault F1 Team*, winning back-to-back Constructor's and Driver's titles in 2005 and 2006. From 1986 to 2001, the team had raced as *Benetton Formula 1 Racing Team*, winning the Driver's titles in 1994 and 1995 and the Constructor's title in 1995.

1.1 The CFD development process

While developing the aerodynamics of the car, the focus is both on creating *new* concepts and improving *existing* concepts. When CFD was introduced into Formula One in the 90's, it was primarily used for *concept exploration*[3]. This is because CFD allows one to visualize the flow field around the car. To be more specific, it is possible to visualize the scalar and vector fields of the different flow parameters. This provides an *understanding* as to why a certain concept works (or not). The wind tunnel does not lend itself very well for gaining understanding, even though it is possible to visualize surface lines by using UV paint. The wind tunnel is more appropriate for optimizing known concepts, i.e. *concept exploitation*. Nowadays however, CFD is being used more and more as an optimization tool as well.

1.1.1 Concept exploration

Let's take as the starting point a certain aerodynamic concept. It could either be an innovation or a well known and tried idea. The first step is to generate the necessary surfaces in CAD. The commercial package used for this is CATIA.
The next step is to integrate the surfaces into an existing baseline model of the car. The commercial package ANSA is used for this. ANSA is also used for generating the surface mesh. Once the new surfaces have been integrated with the car, an unstructured surface mesh consisting of triangular elements is generated.
The discretized surfaces of the car are then imported into Star-CCM+. This commercial package is used as a volume mesher, flow solver and post-processor. It can also be used as a surface mesher, but it is customary within LRGP to generate the surface mesh in ANSA. To simulate the boundary layer, a prism layer mesh composed of orthogonal prismatic cells is grown from the car surfaces.
The next step in the process is to solve the flow equations. The flow field parameters and loads extracted from the CFD results are averaged over a final set of iterations (for example, the last 100) instead of simply taking the value of the last iteration. When plotting these parameters as a function of the iterations, they may show an oscillatory behavior. Even though the solver used is the steady state solver, the flow field around the car is quite unsteady, primarily due to the wake shed by the tires. The reason why it is important to average the parameters over a number of final iterations instead of just taking the final value is because the results will be compared to other runs. Now, if for example the configuration of a run 1 actually generates a higher load than run 2 but the final iteration of run 1 is at the minimum of an oscillation while run 2 is at the maximum of an oscillation, then simply taking the final value will give the opposite results. Figure 1.1 gives a simple example of this, in which the blue line represents run 1 while the red line represents run 2.

[3] Benetton F1 was one of the first F1 teams to use CFD.

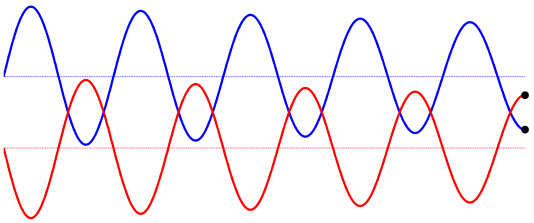


Figure 1.1 Simply extracting the values of the final iteration can obscure the comparison between different runs.

Once the whole process has finished, the CFD aerodynamicist takes over again and analyzes the results. The process from geometry generation to post-processing is summarized in Figure 1.2.

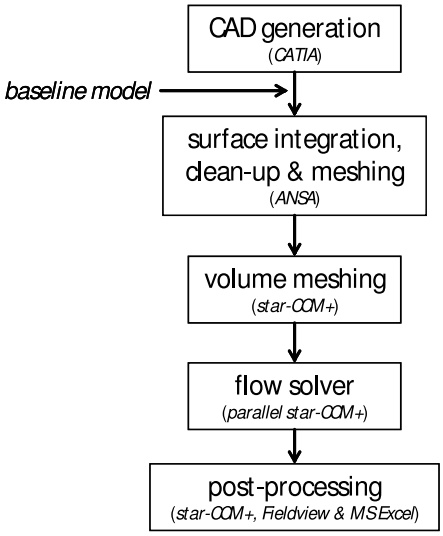


Figure 1.2 The CFD process at LRGP.

1.1.2 Concept exploitation

The CAD generation, surface cleanup and post-processing blocks are labor intensive processes. This is not a real problem if only a couple of runs are carried out and the main goal is to determine whether the concept works or not. However, once the concept is believed to have potential and the focus shifts to optimizing it, it is anticipated that a relatively large number of runs will be carried out. In that case, it becomes very important to fully automate the process summarized in Figure 1.2. To this end, the commercial package iSIGHT is used. iSIGHT is basically a program that allows the user to link different simulation codes and monitor and analyze these

runs in a clear and graphical manner. To use iSIGHT, the user should write control files for each simulation code. Also, the CAD geometry should be parameterized. In addition to this, the car model is defeatured to reduce the runtime of each function evaluation. This has become particularly important due to recent regulation changes concerning the aero resources a team can use (the 'FOTA[4] restriction'). The pre-existing optimization methodology works as follows: Given a number of runs to be carried out, a Design Of Experiments (DOE) is used to seed the design space using a suitable criterion, depending on the type of DOE. Based on these results, the user should pick the best design point. If this configuration is infeasible, it will be tweaked manually to make it feasible. If required, the next step is then to carry out a new DOE around this point and investigate whether an improvement has been found. If this is the case but the configuration is infeasible, then some manual tweaking is again required to make it feasible. Figure 1.3 summarizes the approach.

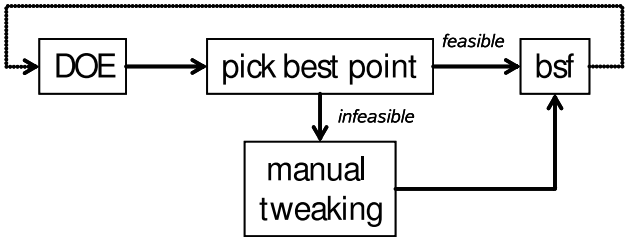


Figure 1.3 The pre-existing optimization process at LRGP. 'bsf' stands for 'best so far'.

The optimization process described is not really an optimization process in the traditional sense where a numerical algorithm iteratively searches for an optimum. Instead, in the process described the experienced CFD aerodynamicist is the optimizer. The advantage of this approach is that the user retains more control over the process and directions taken, which is especially desired when it is difficult to express the objective mathematically and the aerodynamic knowledge of the user is paramount in determining what a good design is.
For example, when developing the rear wing for a high downforce circuit, the primary goal is to maximize downforce, which is easy to express mathematically. However, it is not uncommon for highly loaded wings to be close to separation and in fact, as the trailing edge region of the flap starts to separate and an off-body stall is observed, they can still be generating high downforce. On top of this, the trailing edge region of the main plane can also show signs of separation. So, if the objective is simply expressed to be maximizing the rear wing's load, then it is very likely that the resulting wing will be an unstable one. But, there is no one objective way of quantifying this stability. Skin friction, measured at the back of the flap and main plane, can be used or the total pressure behind the wing. But the question remains how they should be weighted in the optimization problem. To make the problem

[4]The Formula One Teams Association (FOTA) is a group consisting of the competing F1 teams. Its goal is to give the teams a united voice in their ongoing discussion with the sport's regulating body (FIA) regarding the future of Formula One.

even more difficult, it should be noted that the aerodynamic characteristics of an F1 car are highly sensitive to its attitude[5] with respect to the incoming flow. When carrying out an optimization in CFD, the car is commonly set to a single attitude. So, an optimum configuration at that particular point may turn out to be very poor at other attitudes. Additionally, to reduce the CFD runtime, some features may be removed from the model. This will change the flow field around the configuration and possibly the optimum configuration.
The example described above highlights the fact that applying optimization methods to an F1 car is far from straightforward. Simply encapsulating a CFD process with an optimization solver could easily lead to poor configurations. This is one of the reasons why the optimization methodology as summarized in Figure 1.3 is the accepted one. However, this does not mean that other methods should be discarded.

1.2 Goal of the present research

The optimization methodology presented in the previous section has some drawbacks, which are summarized below:

- apart from the DOE itself, the optimization methodology is a manual process, which makes it time consuming;
- the DOE returns a discrete set of configurations. As a result of this, the user may overlook better designs;
- no full use is made of the DOE results to gain a good understanding of the design problem.

Since the CFD simulations are expensive, calls made to them should be kept at a minimum. A popular method that takes this into account, while at the same time addresses the above mentioned drawbacks, is known as surrogate modeling. Using the DOE results, a surrogate model is built, which is a cheap replacement to the expensive function. This surrogate can then be used for a variety of goals. The two most important uses in the present context are to find the optimum on the surrogate and to explore it in order to gain a better understanding of the design problem.
The main goal of the present research is to develop a surrogate modeling framework and eventually apply it to actual aerodynamic design projects at LRGP. The surrogate modeling technique chosen is known as Kriging[6]. The reason why Kriging is chosen is because it has the following desirable features:

[5] Pitch, yaw, roll, heave and front tire steer.
[6] In the context of deterministic computer experiments it is known as DACE, which is short for *Design and Analysis of Computer Experiments*.

- it is an interpolating method. This is desirable in the context of deterministic computer simulations;
- it can capture highly nonlinear responses;
- its statistical interpretation makes it particularly efficient in the context of global optimization.

Next to developing a surrogate modeling framework, the present research also assesses the idea of applying *existing* optimization algorithms directly to the expensive functions. In other words, the step of carrying out a DOE followed by building a cheap surrogate is circumvented. Although this approach does not lend itself very well for gaining an understanding of the design problem, it may find a better optimum, potentially using fewer function evaluations. Two CFD test cases are set up to compare the existing optimization algorithms and the developed surrogate modeling framework.
So, the goal of the present research is threefold:

1. develop a surrogate modeling framework;
2. compare the performance of the surrogate modeling framework with existing optimization algorithms;
3. apply the developed surrogate modeling framework to real life projects.

1.3 Key contributions

The following contributions are made in the present research:

- To potentially improve the accuracy of the surrogate, an alternative objective function is introduced for determining the optimum correlation parameters that comprise a Kriging model. In the literature, the maximum likelihood function is used to build the optimum Kriging model. The alternative objective function introduced in this research is based on leave-one-out cross-validation.
- Kriging is modified to handle noisy functions by smoothing data points rather than interpolating them. The two ideas introduced in this research are to either only smooth clustered points within a user-defined threshold or apply a level of smoothing to each point which is proportional to its corresponding uncertainty.
- The global optimization algorithm DIRECT[7], which can only handle box constraints, is modified to handle nonlinear constraints using an *exact* penalty function.

[7] DIRECT is used to find the global optimum on the surrogate model.

The above contributions, together with the original Kriging method, are implemented into a Matlab based standalone executable and successfully applied to a number of real-life rear wing development projects carried out for the 2011 car.

1.4 Thesis outline

Chapter 2 reviews the existing optimization algorithms used in the present research. Rather than mentioning the actual implementations used, it focuses on the concepts behind these methods. Chapter 3 reviews the surrogate modeling framework. The three building blocks (setting up the DOE, building the surrogate and finally using it) are discussed in detail. Chapter 4 mentions the two contributions made to Kriging, i.e. the alternative objective function introduced for finding the optimum correlation parameters and a method to fit noisy functions. The resulting Kriging based optimization tool is presented in Chapter 5. Chapter 6 presents two test applications that are used to compare the existing optimization algorithms with the newly developed tool. Four real-life problems are tackled by the Kriging based optimization tool in Chapter 7. Chapter 8 ends this thesis with conclusions and recommendations regarding the existing optimization algorithms and the surrogate modeling framework.

'This downhill path is easy, but there's no turning back.'

Christina Rossetti, Amor Mundi

2 Review of local solvers

This chapter reviews a number of optimization methods that only look for a *local* optimum. This is because the objective functions are obtained from expensive CFD simulations and therefore it would not make sense to apply global algorithms like simulated annealing, genetic algorithms or branch-and-bound as they require a significant number of iterations to find the global optimum. Before presenting the local solvers, it is worthwhile to mathematically express the optimization problem:

$$\begin{gathered}\min_{\mathbf{x}\in\mathbb{R}^n,\mathbf{u}\in\mathbb{R}^p} f(\mathbf{x},\mathbf{u}) \\ \text{subject to} \\ \mathbf{R}(\mathbf{x},\mathbf{u}) = \mathbf{0} \\ c_i(\mathbf{x},\mathbf{u}) = 0, i = 1,...,m_e \\ c_j(\mathbf{x},\mathbf{u}) \leq 0, j = 1,...,m_i\end{gathered} \tag{2.1}$$

f is the objective function. It is obtained by solving the set of partial differential equations $\mathbf{R}(\mathbf{x},\mathbf{u}) = \mathbf{0}$, also known as the flow solver. f is a function of n design variables[1] and p state (/flow) variables[2]. The flow solver essentially enters the optimization problem in the form of an equality constraint. In addition to this constraint, there are m_e equality constraints and m_i inequality constraints. (2.1) is known as a *PDE-constrained optimization problem*. There are two main strategies that one can use to solve this type of problem:

1. Simultaneously solve the PDE and optimization problem (one-step method).
2. Eliminate the state variables from the optimization problem and solve the resulting two systems (flow and optimization) separately at each iteration (two-step method).

[1] Examples of design variables in the present context are wing chord length, angle of attack and wing thickness.

[2] Examples of state variables in the present context are flow velocities and pressures.

The advantage of the two-step method over the one-stage method is that it allows one to use *existing* optimizers and PDE solvers to solve the problem. In that case, one can reformulate (2.1) as follows:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \\ &\text{subject to} \\ &c_i(\mathbf{x}) = 0, i = 1, ..., m_e \\ &c_j(\mathbf{x}) \leq 0, j = 1, ..., m_i \end{aligned} \tag{2.2}$$

The flow system is now embedded in f. (2.2) is known as a *non-linear programming problem* (NLP) since at least one of the functions f, c_i or c_j may be nonlinear.
The local solvers can be subdivided into methods that make use of the gradients of the considered functions and gradient-free methods. Section 2.1 discusses gradient-free methods while Section 2.2 discusses the gradient-based methods. To keep the discussion clear, these two sections assume that no constraints are present. The handling of constraints is reserved to Section 2.3. Finally, Section 2.4 discusses how gradients can be calculated.

2.1 Gradient-free methods

Gradient-free methods do not make use of the objective function's gradient to determine where to sample the design space. Instead, they evaluate the function at a set of points, compare the function values with each other and subsequently decide where to sample next.

2.1.1 The simplex method

The most well-known gradient-free method is the simplex method, proposed by John Nelder and Roger Mead [37]. Given a starting point in an n-dimensional space, a simplex with $(n+1)$ vertices is formed[3]. Its n edges can be considered to be vectors that can span the *whole* space. Once the initial simplex has been formed, the function values at the vertices are ordered from best to worst. A new function evaluation is carried out at the point where the worst point reflects through the opposing edge. Depending on the function values found, the simplex can either expand or contract. In the subsequent iterations, *reflection*, *expansion* and *contraction* are the

[3]In a two dimensional space, a triangle is formed and in a three dimensional space, a tetrahedron is formed.

three basic moves that the simplex can make as it marches towards the minimum. Figure 2.1 illustrates the method in two dimensions.

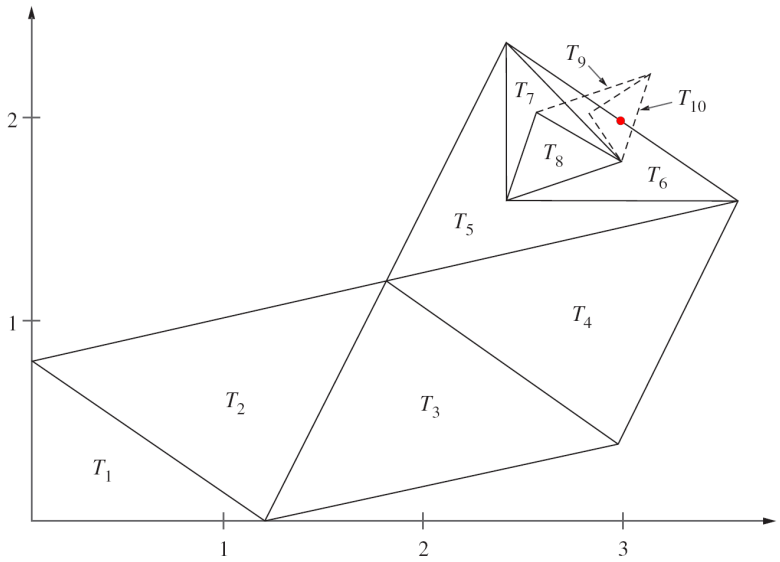


Figure 2.1 The sequence of triangles T_k converging to the point (3,2) for the simplex method.

2.1.2 The alternating variables method

Another well-known method that does not make use of gradients is the alternating variables method. As the name suggests, at each iteration k ($k = 1, 2, ..., n$) the variable x_k is changed in order to minimize the objective function value while the other variables are kept fixed. When all variables have been changed (independently from each other), the whole cycle is repeated until convergence occurs. The method is usually very inefficient and unreliable. Since the search directions are independent of each other and aligned with the coordinate axes, the method ignores the possibility of correlation between the variables. Various modifications have been proposed to this basic method. The most prevalent is based on the observation that the points at the start and end of a cycle determine a line along which more progress might be made towards the optimum than along the coordinate directions independently from each other. The most efficient of these modified methods is probably the method proposed by Hooke and Jeeves [19].

Despite their slow convergence compared to gradient-based methods, gradient-free methods remain popular within the engineering community. It often occurs in engineering practice that the designer is looking for a *quick* improvement over a baseline design and frequently a highly accurate solution is neither possible nor desired. An optimization algorithm will therefore often be stopped long before it formally con-

verges. So, even though the mathematical theory predicts that gradient algorithms are faster than the gradient-free methods, this may not always be observed in practice. Furthermore, if the objective functions contain noise or discontinuities, then gradient algorithms will generally perform worse, whereas gradient-free methods are better able to deal with these types of functions. For a further discussion on gradient-free methods, the reader is referred to Torczon [50] and Wright [56].

2.2 Gradient-based methods

Instead of comparing function values, gradient-based methods make us of the function's gradient to determine where to go. The gradient is a vector pointing into the direction of maximum ascent. Gradient-based methods can be subdivided into methods that only use the gradient (first-order methods) and methods that use (an approximation of) second-order derivatives as well. Using second-order derivatives to determine the search direction can increase the rate of convergence.

2.2.1 First-order methods

The steepest descent method is the most basic of the gradient methods; the search direction is simply the opposite direction of the gradient. Compared to the more advanced gradient methods discussed below, it is theoretically the slowest (linear convergence). To appreciate why this method is slow, consider a two-dimensional function with an ellipse shaped minimum, as given by Figure 2.2. Since the direction of steepest descent will be orthogonal to the contour lines, it will generally not point to the minimum, except of course at the extremes of the major and minor axes. For this reason, the method will zigzag its way to the solution; a behavior reminiscent to the alternating variables method discussed in the previous section. However, whereas the alternating variables method would simply choose its directions based on the coordinate axes used, the steepest descent method clearly makes a better choice in that respect.
An improvement on the steepest descent method is the conjugate gradient (CG) method. Instead of the local gradient, it uses the conjugate gradient to determine the search direction. The conjugate gradient method was originally developed by Hestenes and Stiefel [17] as an iterative method for solving large linear systems with positive-definite matrices. Fletcher and Reeves [11] later adapted it for solving large-scale nonlinear optimization problems. The details of how the directions are chosen are outside the scope of this thesis. The interested reader is referred to Shewchuk [46], who describes the workings of the CG method in a clear and illustrative manner.

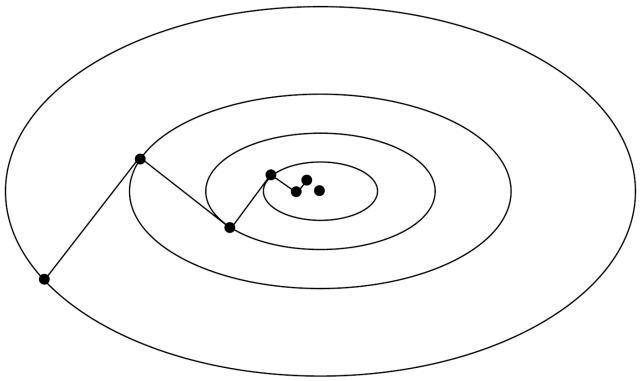


Figure 2.2 Zigzagging behavior of the steepest descent method [39].

2.2.2 Second-order methods

Second-order methods locally model the objective function at the iterate k through the use of a quadratic function:

$$f(\mathbf{x}_k + \mathbf{p}) \approx f(\mathbf{x}_k) + \mathbf{p}^T \nabla f(\mathbf{x}_k) + \frac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_k)\,\mathbf{p} \tag{2.3}$$

They then (approximately) minimize this model to find the next iterate $k+1$. $\nabla f(\mathbf{x}_k)$ and $\nabla^2 f(\mathbf{x}_k)$ are the gradient and Hessian of the function at the current iterate, respectively. To have a model with a strict local minimum (a bowl shape in a two-dimensional problem), the Hessian has to be positive definite. Assuming that the gradient and Hessian are known, the direction and magnitude to be taken in order to find the next iterate can be determined by setting the derivative of (2.3) to zero. The result is the following explicit formula:

$$\mathbf{p}_k = -\nabla^2 f_k^{-1} \nabla f_k \tag{2.4}$$

The method described above is known as Newton's method, and it is a significant improvement over the steepest descent method as it possesses quadratic instead of linear convergence behavior. Even though the method is simple and appears ready to be used in practice, there are two main difficulties associated with it:

- obtaining the Hessian efficiently; contrary to the gradient, which is an n-sized vector containing the first order partial derivatives of the objective function, the Hessian is an n^2-sized matrix containing the function's second order partial derivatives. So, one can imagine that determining the Hessian directly may prove extremely costly when the objective functions are themselves expensive to calculate and/or the number of variables is high;

- uphill search direction; far from a minimum, it is possible that the Hessian is not positive-definite. Taking the actual Newton step can then move the search uphill rather than downhill.

Quasi-Newton methods are able to deal with the above two difficulties. First of all, they *approximate* the Hessian using gradient information alone, which enables them to obtain the second-order information efficiently. They start with an initial guess of the Hessian (generally just the identity matrix) and update this guess using only gradient information. Secondly, by starting with a positive-definite matrix and updating it such that it stays positive-definite, quasi-Newton method will guarantee that a downhill move is always made. The first quasi-Newton method was formulated by Davidon [7]. Various updating schemes have been devised but the one presently considered to be the most effective is the BFGS method[4]. The disadvantage of quasi-Newton methods compared to the Newton method is that theoretically they are slower. They exhibit super-linear convergence rather than quadratic converge near a minimum.

2.2.3 Stepping to the next iterate

Up till now, the focus has been on describing different ways of choosing a *direction* using gradients. The next matter is the determination of the *step size* to be taken in this direction. Indeed, this is an important consideration since certain step sizes may in fact produce an *uphill* movement, even though the direction chosen was a descent direction. Figure 2.3 illustrates this. The current iterate is represented by the black dot. The actual function is represented by the blue line whereas the red line represents the local quadratic model. If the step size is chosen to be the minimizer of this quadratic model (step length of 1 for (quasi)-Newton methods), then in reality this would result in an uphill move.

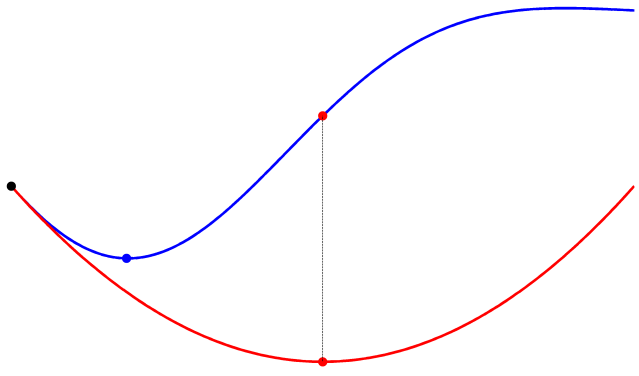


Figure 2.3 Stepping to the next iterate.

[4]BFGS stands for Broyden [5], Fletcher [9] Goldfarb [15] and Shanno [45], the names of the researchers who developed the method independently from each other.

To stabilize the gradient-based methods, line search strategies can be used. For efficiency, practical methods will only look for an approximate minimizer. All these methods start out with an initial estimate and as long as certain conditions are not met, new step lengths are estimated provided an upper bound on the number of function evaluations has not been reached. The most popular of these conditions are the Wolfe conditions [55]:

- the first condition states that there should be a sufficient decrease in the objective function. However, this condition alone could hinder the algorithm from making reasonable progress, as a sufficiently small step would already give a decrease;

- the second condition (curvature condition) ensures that the steps cannot be too small by forcing the gradient at the next iterate to be a certain factor higher than at the current iterate.

Having specified the termination conditions, line search algorithms proceed by bracketing the search interval. For Newton and quasi-Newton methods, the end point is usually chosen to be the point resulting from the step length of 1. If this point does not satisfy the conditions, the step is reduced until either the conditions are satisfied or a maximum number of line search function evaluations is reached. To make this so-called backtracking more efficient, it is possible to interpolate a quadratic or cubic to choose the next step length. Steepest descent and conjugate gradient methods do not have a natural step length so they have to use some heuristic to determine the initial guess. Optimal step sizes from previous iterations may be used to this end.
An alternative to line search methods are trust-region methods. Where line search methods first choose a direction and then a step size, trust-region methods choose a direction and step size simultaneously. They do this by defining a region around the current iterate within which the local model is assumed to be accurate and they then try to find the minimizer within this region. Once the position of this minimizer has been found, they calculate its actual value. The following three scenarios are possible:

1. if the prediction was particularly poor or the new iterative has a worse function value than the current iterate, the step is rejected and the trust-region shrinks;

2. if the prediction was very good then the step is accepted and the trust-region is expanded;

3. if the prediction was not particularly bad but also not especially good, the step is accepted but the trust-region is left unchanged.

There is no consensus in the literature on which of the two methods is better. More information on gradient-based algorithms can be found in the book by Nocedal and Wright [39].

2.3 Constraint handling

This section discusses how constraints can be handled. In the present context, constraint functions could for example be aerodynamic loads or functions representing distances between (complex) shapes. For this reason, all the methods discussed are capable of handling *nonlinear* constraints. The literature on constraint handling is vast and by no means is it intended here to provide a complete description. Instead, the most prevalent methods are presented. The discussion of these methods was inspired by the books by Fletcher [10] and Nocedal and Wright [39].
There is no standard taxonomy for nonlinear constrained optimization algorithms. Nevertheless, the most basic subdivision one can make is to categorize the methods into those that directly handle constraints and those that indirectly take the constraints into account.

2.3.1 Indirect methods

Instead of handling the constraints directly, indirect methods recast the constrained problem into an unconstrained problem. The resulting unconstrained problem may then be solved by any of the methods discussed in Sections 2.1 and 2.2. The original constrained problem can be recasted by defining the new objective function as the sum of the original objective function and the constraint violations. If none of the constraints are violated, then the latter term should naturally be zero. If the current iterate is infeasible on the other hand, this term should be positive. The newly defined function is therefore referred to as a penalty function.

2.3.2 Direct methods

Instead of recasting a constrained problem into an unconstrained one and optimizing the result, direct methods deal with the constraints directly.

Feasible direction methods

The most intuitive way of dealing with constraints is to choose the search direction that will produce a decrease in the objective function while ensuring that the current iterate does not become infeasible. This is the basic idea behind the so-called *feasible direction methods*, introduced by Zoutendijk [57]. If none of the constraints are active or violated, then a search direction is chosen using one of the gradient methods

described in Section 2.2. If a constraint is active, i.e. the current iterate is on the constraint boundary, then Figure 2.4 illustrates how the direction is chosen. The blue and red curves are part of the function and constraint contours, respectively, at the current iterate. The region under the blue dotted line, which is normal to the gradient of the objective function $\nabla F(\mathbf{x})$, consists of *usable* search directions. However, they are not all *feasible*. The region to the right of the red dotted line, which is normal to the constraint gradient $\nabla c(\mathbf{x})$, comprises the feasible directions, not all of which are usable. The intersection of the usable and feasible regions is called the *usable-feasible* region. The method of feasible directions uses a control parameter to determine the search direction $\mathbf{p}$ within this region. The parameter is referred to as the push-off factor. A value of zero will result in a search direction tangent to the active constraint while large values will produce search directions that tend to follow the contour of the objective function. This means that small values will produce a large decrease of the objective function but the same constraint boundary may quickly be re-encountered. On the other hand, large values will reduce the risk of hitting (or puncturing) the constraint boundary but the decrease in the objective function will not be very large. Instead of randomly choosing a value, the push-off factor can be made a function of the constraint value.

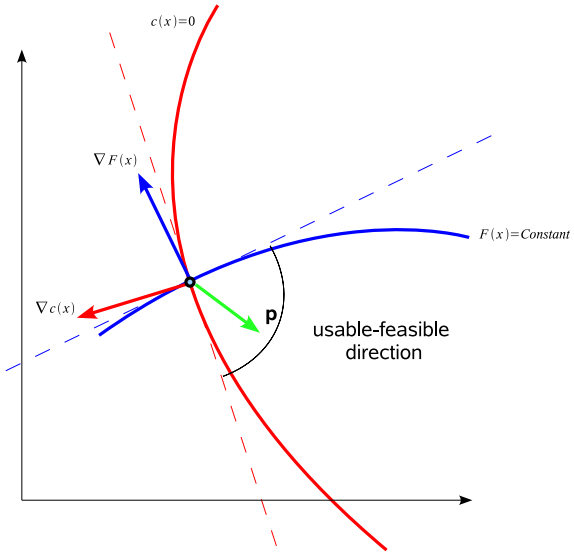


Figure 2.4 The usable-feasible direction.

Elimination methods

To introduce the next method of directly handling constraints, assume temporarily that there are only equality constraints. Also, the nonlinear constraints were

linearized. In that case, the following system can be set up:

$$A^T\mathbf{x} = \mathbf{b} \tag{2.5}$$

If m is the number of equality constraints and n is the number of variables, then $m \leq n$ must hold for the problem to be solvable. In that case, it is possible to split the variables up, where the first set will contain m elements while the second will contain the rest; $\mathbf{x}_1 \in \mathbb{R}^m$ and $\mathbf{x}_2 \in \mathbb{R}^{n-m}$. It is easy to show that (2.5) can be rewritten to:

$$A_1^T\mathbf{x}_1 + A_2^T\mathbf{x}_2 = \mathbf{b} \tag{2.6}$$

where $A^T = \begin{pmatrix} A_1^T & A_2^T \end{pmatrix}$. (2.6) can readily be solved such that it expresses $\mathbf{x}_1$ in terms of $\mathbf{x}_2$:

$$\mathbf{x}_1 = A_1^{-T}\left(\mathbf{b} - A_2^T\mathbf{x}_2\right) \tag{2.7}$$

The *unconstrained* problem can now be solved by expressing the objective function in terms of the reduced set of variables $\mathbf{x}_2$. $\mathbf{x}_1$ follows from $\mathbf{x}_2$ through (2.7) and the equality constraints are automatically satisfied. Inequality constraints can be taken into account by adding slack variables.

Both the feasible direction method and the elimination method are feasible methods, meaning that they try to keep all iterates feasible. This is an advantage if the objective function is undefined in the infeasible space. Another advantage is that the actual objective function is minimized at each step. In other words, one does not have to modify the problem by introducing a penalty or merit function (this will be described later). A disadvantage of these methods is that they cannot take shortcuts through infeasible domains to get to the solution quicker. So, when comparing them with methods that do allow the intermediate iterates to be infeasible, the feasible methods may require more function evaluations. As a final note, it is interesting to mention that the feasible direction and elimination methods have opposite philosophies. Where elimination methods try to follow the constraint boundary, feasible direction methods try to push themselves away from it.

Sequential Quadratic Programming (SQP) methods

The most advanced of the direct methods are known as Sequential Quadratic Programming methods. As the name suggests, the idea is to solve a quadratic programming (QP) sub problem at each iteration. A QP problem is defined to have a quadratic objective function with linear constraints. However, instead of setting up

a quadratic approximation of the objective function, as given in (2.3), a quadratic approximation of the Lagrangian is used instead. The Lagrangian is defined as follows:

$$L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \sum_i \lambda_i c_i(\mathbf{x}) \tag{2.8}$$

where the summation is taken over all the active constraints. Whereas one tries to simply minimize the objective function f in an unconstrained problem, as discussed in Section 2.2, the plan is now to minimize the Lagrangian, as its minimum will correspond to a point that is the minimum of f while at the same time satisfying the constraints. A wide range of methods exist to solve the QP problem. The interested reader is referred to Chapter 16 of Nocedal and Wright [39].
An SQP method can either be implemented as a line search method or a trust-region method. When it is implemented as a line search method, the next iterate is chosen along the direction found by the QP problem. When it is implemented as a trust-region method, the trust-region is added as a constraint to the QP problem. In both cases, use is made of a so-called merit function to ensure that the SQP method converges from remote starting points. The merit function is essentially a penalty function. In the case of a line search method, the merit function replaces the actual objective function to determine the step size. In the case of a trust-region method, the merit function is used to determine whether a step should be accepted or not and to determine whether the trust-region should be expanded, shrunk or kept unchanged for the next iterate. Not only does the merit function make it possible for the SQP method to converge from remote starting points, it also makes it possible for the algorithm to accept an infeasible iterate, even though it should eventually converge to a feasible point. This may be seen as a disadvantage if the objective function is undefined in feasible regions but it can be an advantage as the method will be able to take shortcuts to the optimum point. This property is contrary to the two feasible methods discussed earlier.

2.4 Calculating derivatives

Since the gradient methods require the derivatives of the objective function and constraints to be calculated, it is important to know how to obtain them. There are three approaches that can be used (from least to most accurate):

1. the finite difference method
2. the complex variables method
3. exact differentiation

2.4.1 Finite difference method

The finite difference method makes use of Taylor's theorem to *approximate* the derivative at a certain point. For example, the first derivative at a point x can be calculated as follows:

$$\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{2.9}$$

(2.9), also known as forward differencing, is a first order approximation to the derivative. A second order approximation, known as central differencing, can be obtained as follows:

$$\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \tag{2.10}$$

Note that even though the latter should be more accurate (in theory), it requires twice as many function evaluations. The advantage of calculating derivatives with the finite difference method is its simplicity. Given a black box function, it can be calculated 'around it', i.e. the source code is not required. This is in contrast to the complex variables method and exact differentiation. A disadvantage of the finite difference method is that one has to choose a suitable value for the gradient step size, Δx. The step size cannot be too big because truncation errors will spoil the results. Theoretically, it has to be chosen as small as possible, i.e. it has to approach zero. In practice however, when using a computer to calculate the functions, round-off errors prevent the step size from becoming too small. One can therefore expect that there will be an optimal step size. When the objective function is a simple evaluation on a computer, the step size can be chosen quite small. If u is the round-off error (typically about 1E-16 in double-precision arithmetic), a fairly optimal choice of the step size is $\sqrt{u}$ [41]. This rule is followed by many practical optimizers when they use finite differencing to determine the functions' gradients.
However, the objective functions may contain a certain degree of noise and for this reason, it is not practical to take a gradient step size as small as $\sqrt{u}$. Noise in this context is not the same as in physical experiments. On the contrary, the noise is repeatable. In the present context, the noise in the CFD functions can have various sources:

- the first is attributed to the discretization error, in particular, perturbations in the mesh as small changes in the geometry occur. Especially when remeshing the geometry for each design point, this can be a substantial contributor to the noise. The method of remeshing from scratch is the current method used in the LRGP CFD department;
- another possible source is due to incomplete convergence. In practice the simulation is stopped long before the residuals reach machine precision. To

decrease the influence of this source it is therefore wise to average the objective function over the last X number of iterations instead of taking the actual final value, a method that is already implemented in the current LRGP CFD methodology;

- yet another source may be due to the inaccurate application of boundary conditions (e.g., iterating during the solution towards a fixed value of lift).

If no prior knowledge is available and the optimal gradient step size for a CFD function is to be determined, a parameter sweep will have to be carried out at a certain design point and the step should be chosen such that one is outside the noise level of the function. As the optimal gradient step size may be different at various design points and from problem to problem, it should be evident that even though the method of finite differencing is very simple, it has a serious practical drawback.

2.4.2 Complex variables method

An alternative, more accurate method of calculating derivatives makes use of a *complex* Taylor series expansion instead. By introducing a small perturbation parameter ϵ to the imaginary component of a function f, the derivative may be approximated as follows:

$$\frac{\partial f}{\partial x} \approx \frac{\Im\left[f\left(x+\epsilon\mathbf{i}\right)\right]}{\epsilon} \tag{2.11}$$

As a result, the gradients are obtained without subtracting similarly valued functions [33]. However, this method is still in principle an approximation method for calculating derivatives and in addition requires the function's source code.

2.4.3 Exact differentiation

The final method discussed is an exact method for obtaining gradients, which boils down to the well-known method of symbolic differentiation. Even though one might expect that exact differentiation would only be possible for simple analytic functions and not for a sophisticated computer code, the idea of applying exact differentiation to a computer code is that at the end of the day, any function is calculated by carrying out a *sequence* of simple elementary operations involving one or two arguments at a time. Examples of one-argument operations are the trigonometric, exponential and logarithmic functions. Two-argument operations include addition, multiplication, division and the power operation. Extensive use is made of the chain rule, which states that if h is a function of the vector $\mathbf{y} \in \mathbb{R}^m$, which in turn is a

function of the vector $\mathbf{x} \in \mathbb{R}^n$, the derivative of h with respect to $\mathbf{x}$ can be written as follows:

$$\nabla_x h\left(\mathbf{y}\left(\mathbf{x}\right)\right) = \sum_{i=1}^{m} \frac{\partial h}{\partial y_i} \nabla y_i\left(\mathbf{x}\right) \tag{2.12}$$

In the same way that the original code is written to calculate the objective function at a certain design point, it is possible to write a code that can calculate the derivative of the objective function at the same point. However, realizing that writing a good computer code can already be labor intensive, one can imagine that writing the differentiated code will be labor intensive as well, especially if the original program is updated regularly. Fortunately, there are computer programs available that can automatically generate code capable of computing the derivative of the original function. These Automatic Differentiation (AD) programs work by breaking the original code into the most elementary operations mentioned above. They then apply the chain rule to build the exact derivative. For a review on recent developments in automatic differentiation the reader is referred to Bischof *et al.* [2].
Even though exact differentiation will provide an exact value for the derivative of a certain function f with respect to x_i, one still has to carry out this procedure n times to obtain the function's gradient, required by the gradient-based algorithms. So, even though exact differentiation is superior to the two previously mentioned methods in terms of accuracy, in terms of cost it is the same. Luckily, it is possible to obtain the gradient of f at the cost of (approximately) one function evaluation by solving the adjoint (dual) problem instead of the original (primal) problem. The concept of duality is a well known concept in mathematics. In the context of design, the first application of adjoint methods was made by Pironneau [40]. In aerodynamic shape optimization, where the number of variables can become very high and the function values are obtained from expensive CFD simulations, adjoint methods have become very popular, as reflected in the large amount of articles that have been published about the subject in recent years ([20], [42], [38], [36], [35]).
Still, one should not forget that the source code is needed in order to carry out the exact differentiation and on top of this, if the adjoint method is to be used, an adjoint solver should be developed. In industry, extensive use is made of commercial codes, for which the source code is not available (i.e. the program is a black box). This has therefore limited the use of these methods in industrial applications.

'All that we see or seem, is but a dream within a dream...'

Edgard Allan Poe, A Dream within a Dream

3 Review of surrogate modeling

The previous chapter reviewed local optimizers. They have two main drawbacks, which especially become evident when the objective and/or constraint functions are expensive.

First of all, if they fail to find an improvement, it is not straightforward to determine why this happened. There are a number of potential causes. For example, it is conceivable that there is indeed no improvement possible. Another cause could be the solver settings. For the gradient algorithms, inaccurate gradients can easily cause the methods to stall or choose suboptimal search directions. Finally, failure to find an improvement could have been caused by a failed run that occurred somewhere in the process. In previous projects carried out in the LRGP CFD department, cases have been reported where an optimizer would suddenly start searching in a completely different part of the design space after an analysis failure occurred. If the function to be optimized is cheap and the solver had not found an improvement, it is easy to change some settings or change the initial starting point and restart the algorithm. However, when the functions are particularly expensive, this may not be possible.

The second drawback to directly linking an optimizer with an expensive function is that they do not provide an understanding of the design problem. When successful they do return an improvement compared to the initial point and if they are particularly efficient, a small number of design points in between. This may not be a problem if the improved design is indeed the last stage in the process but in practice this is not the case. Models used in optimization projects are de-featured at certain regions to reduce the mesh count. To confirm whether the design does indeed improve the baseline, it is consequently evaluated on a more refined model, possibly at a number of ride heights or even in yaw and steer, which then requires a full-car simulation. Once the concept has been confirmed to work, it is tested in the wind tunnel and of course the last stage is track testing. At any of these stages it could be found that the concept is actually not an improvement. In that case, it is important to determine *why* this is the case and above all *how* this can be solved. Since an optimizer simply yields an (apparent) optimum, they are obviously not a great tool for gaining understanding.

This chapter reviews an alternative optimization methodology considered suitable when the relevant functions are expensive. Indeed, it is not one algorithm but rather

a set of separate ideas that when combined appropriately can provide a valuable enhancement to the current optimization methodology. It will be referred to as the *surrogate modeling framework*.
As the name suggests, the concept of surrogate modeling is to replace an expensive function with a cheaper function that should serve as its surrogate. So, instead of thoroughly exploring the expensive function or using it for (global) optimization, these operations are carried out on the cheaper surrogate. The framework can be roughly subdivided into three steps:

1. choose a set of points where the original, expensive function should be evaluated. This is the so-called design of experiments (DOE) or 'calibration' phase;
2. using the DOE results, build a surrogate model;
3. use the surrogate for the intended purpose(s).

The following three sections describe each step in detail.

3.1 The design of experiments

The first step in the surrogate modeling framework is to seed the design space with a number of sample points where the actual function is to be evaluated. To obtain a surrogate model with sufficient accuracy throughout the design space it is important that the sample points cover the whole space. The easiest way of achieving this is to divide the space into a rectangular mesh and make each vertex a design point. This approach is known as a *full factorial*. The main disadvantage of this approach is that the number of points required rises exponentially with the number of variables. An alternative to the full factorial approach is to only use a subset of it. Instead of choosing the subset randomly, it can be chosen such that it maintains orthogonality (independence) among the different variables and interactions. These designs are also known as *orthogonal arrays*.
However, when it is essential to use a minimum number of sample points, a very popular method is to make use of *Latin Hypercubes*. If it is decided to use m samples, then each dimension is subdivided into m levels. There is no unique way of obtaining such a design. For this reason, Latin Hypercubes are generated randomly. The advantage of Latin Hypercubes is that each variable instance is visited only once. The disadvantage is that since they are generated at random, it is possible to come up with a sample set that does not cover the space sufficiently.
To obtain a so-called space filling design, an optimization routine can be set up with the ordering of the sample points in the space as the variables and the minimum distance between two points as the objective function to be maximized. These optimization problems can be solved with various combinatorial optimization methods, like genetic algorithms or simulated annealing. The *Optimal Latin Hypercube* is the

DOE technique of choice within the optimization methodology at the CFD department.
Another DOE technique is known as *Central Composite Designs*. The design is generated by augmenting a 2-level full factorial with a center point and two additional star points for each of the n variables. This amounts to $2^n + 2n + 1$ design points in total. The disadvantage of this type of DOE is that the number of design points rises quickly with the number of dimensions and in addition to this the user cannot specify the number of design points to be used. This is one of the major advantages of (Optimal) Latin Hypercubes, as the user can explicitly specify the number of sample points to be used for constructing the surrogate model.
In the rest of this thesis, Optimal Latin Hypercube sampling is used as the DOE technique.

3.2 Building the surrogate model

Using the DOE results, the next step is to build the surrogate model[1]. The model can either be built by fitting a mathematical function through the data points or one can take into account the underlying physics. The former method is considered in this thesis. A discussion of the latter approach can be found in Beran and Silva [1].
A number of surrogate modeling techniques exist. Polynomial fitting is the simplest and most well-known technique. The surrogate model can be represented as follows:

$$\hat{y}\left(\mathbf{x}\right) = \sum_{h} \beta_h f_h\left(\mathbf{x}\right) + \epsilon \tag{3.1}$$

where $\hat{y}$ is the prediction of the real response y and where each $f_h\left(\mathbf{x}\right)$ is a polynomial term. The β_h's are unknown coefficients. ϵ is a normally distributed, independent error term with mean zero and variance σ^2. The unknown coefficients are determined by solving the following system:

$$\begin{bmatrix} f_1\left(\mathbf{x}^{(1)}\right) & \cdots & f_h\left(\mathbf{x}^{(1)}\right) \\ \vdots & \ddots & \vdots \\ f_1\left(\mathbf{x}^{(m)}\right) & \cdots & f_h\left(\mathbf{x}^{(m)}\right) \end{bmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_h \end{pmatrix} \approxeq \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix} \tag{3.2}$$

The system has m equations and h unknowns, since there are m sample or DOE points and h coefficients. If $h < m$, then the surrogate model will regress the data; the system of equations is solved in the least squares sense. If $h = m$, then the surrogate model will interpolate the data. Even though the technique of polynomial

[1] Another word for surrogate model is metamodel, which means 'model of a model'. Indeed, the expensive simulation code is a model of physical reality and the surrogate is the model of the expensive code.

fitting is very simple, it is not particularly suitable when the actual function is highly nonlinear. Of course, one can use a polynomial of higher order (increase h). However, high order polynomials will tend to show excessive oscillations between data points. So, even though one can easily force the method to interpolate a highly nonlinear function by simply choosing $h = m$, the danger of overfitting (i.e. poor generalization of the surrogate model to unseen data points) is quite high.
A popular fitting method is to make use of radial basis functions (RBFs). As their name suggests, an approximation consists of a sum of weighted basis functions that take as input the Euclidean distance between a trial point and the sample points used to build the surrogate model. A large number of basis functions exist. The most popular are the thin plate spline, multiquadratics and the Gaussian. Given a set of sample points with their corresponding function values, the weights are determined by solving a matrix-vector equation.
Having introduced RBFs, the next technique that is presented has been the focus of the present research. The technique is known as *Kriging* and fundamentally it is an RBF method. However, it has some crucial extensions to the standard method that will become evident shortly.

3.2.1 Kriging: the theory

Kriging was originally used in the 1950s by D.G. Krige [27] to analyze mining data. The method was later refined by Matheron [34]. Nowadays, Kriging is a fundamental technique in the field of geostatistics. The idea of using Kriging to build surrogate models of *computer experiments* was introduced by a group of statisticians in the late 1980s. A paper by Sacks *et al.* [43], entitled 'Design and Analysis of Computer Experiments', deserves special mention here. Following the title of this paper, the use of Kriging in the context of computer experiments has become widely known as DACE. In the rest of this thesis, when mentioning Kriging it will be thought of as its usage in the context of DACE.
The presentation of Kriging in this section follows the explanation outlined by Jones *et al.* [26]. They motivate the idea behind Kriging as a modification to the previously described method of polynomial fitting that addresses some of its shortcomings in the context of fitting responses from deterministic computer experiments. Recall that in polynomial fitting, it is assumed that the errors are independent. For a physical experiment, this is an acceptable assumption. However, for a computer experiment a set of inputs will always return the same outputs. In other words, the assumption of *independent* errors is incorrect. Any error in the metamodel will be due to modeling errors. The error term is therefore really a function of the inputs: $\epsilon(\mathbf{x})$. Additionally, if two points i and j are close to each other, then the errors $\epsilon\left(\mathbf{x}^{(i)}\right)$ and $\epsilon\left(\mathbf{x}^{(j)}\right)$ should be close too. So, instead of assuming that the errors are independent, for computer experiments it makes more sense to assume that they are related somehow; in other words, they are *correlated*. In fact, it makes sense to assume that the correlation is high when points are close to each other and low when

they are far apart. For each sample point, its correlation with the rest of the space can be represented as a function of the distance. This reminds one of the radial basis function method and in fact, Kriging itself is a type of radial basis function method. However, it should be considered a special case since the distance is not defined as the well-known Eucledian distance. Instead, it is defined as follows:

$$d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \sum_{h=1}^{n} \theta_h \left| x_h^{(i)} - x_h^{(j)} \right|^{p_h} \tag{3.3}$$

with $\theta_h \geq 0$ and $p_h \in [1, 2]$. With θ_h fixed and the same for all variables and p_h set to 2, the method would reduce to the standard RBFs. However, Kriging allows each variable to have an individual contribution to the approximation, therefore allowing a more flexible response surface. Kriging can therefore be referred to as a *tuned* RBF method. With the distance function described by (3.3), the correlation between errors at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is:

$$Corr\left[\epsilon\left(\mathbf{x}^{(i)}\right), \epsilon\left(\mathbf{x}^{(j)}\right)\right] = \exp\left[-d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)\right] \tag{3.4}$$

The choice of the exponential was not arbitrary. The reason it was chosen is because it has the required correlation properties:

- for distantly spaced points, the correlation approaches 0;
- for closely spaced points, the correlation approaches 1.

In fact, if the distance between two points is zero then the correlation *is* 1. This makes sense as it was stated that for a computer experiment, a given input will always return the same output. By modeling the correlation in this way, it is acceptable to defer from using a complicated regression term (first term in (3.1)) and just use a simple constant instead. One can state that polynomial regression and Kriging have contradictory philosophies. Where polynomial regression focuses entirely on the regression term and simply assumes independent errors, Kriging focuses on modeling the errors, which are not assumed to be independent but correlated. For Kriging, (3.1) can therefore be written as:

$$\hat{y}\left(\mathbf{x}\right) = \mu + \epsilon\left(\mathbf{x}\right) \tag{3.5}$$

where μ is the constant regression term and $\epsilon\left(\mathbf{x}\right)$ is $Normal\left(0, \sigma^2\right)$. Contrary to (3.1), the correlation between errors is given by (3.3) and (3.4) and is not zero. (3.3)-(3.5) form the so-called 'DACE stochastic process model', From these equations it can be deduced that there are $2n + 2$ parameters that should be determined: μ, σ^2, $\theta_1, ..., \theta_n$ and $p_1, ..., p_n$. Notice that μ is equivalent to the β_h's in (3.1). In the case of

Kriging, it would be found by solving the equation in the *generalized* least-squares sense:

$$\mu = \left(\mathbf{1}^T R^{-1} \mathbf{1}\right)^{-1} \mathbf{1}^T R^{-1} \mathbf{y} \tag{3.6}$$

R is the so-called correlation matrix. If m sample points were used to build the response surface, it is an $m \times m$ matrix whose (i, j) entry is given by (3.4). $\mathbf{1}$ is an m-sized vector of ones. $\mathbf{y}$ is the vector containing the m responses. Having calculated μ, the variance σ^2 can be found as follows:

$$\sigma^2 = \frac{1}{m} \left(\mathbf{y} - \mathbf{1}\mu\right)^T R^{-1} \left(\mathbf{y} - \mathbf{1}\mu\right) \tag{3.7}$$

To be able to calculate μ and σ^2, the parameters $\theta_1, ..., \theta_n$ and $p_1, ..., p_n$ have to be determined, since the correlation matrix is a function of these parameters. They are determined by maximizing a function known as the likelihood function, which is a common function in statistics. The idea of the maximum likelihood function is that it gives the parameters most likely to have resulted in the data observed. The variables $\theta_1, ..., \theta_n$ are referred to as the correlation parameters or simply 'the θ's'. The variables $p_1, ..., p_n$ are known as the smoothing parameters. In geostatistical models they are often set to values between 0 and 1 to allow for erratic responses. In engineering however, where the objective functions are frequently smooth and do not have singularities, the values are often set to 2. In the current thesis, the smoothing parameters were all fixed to 2, leaving the θ's as the tuning parameters. To evaluate the Kriging surface at a point $\mathbf{x}^*$, the following expression is used (the derivation of this predictor can be found in Sacks *et al.* [43]:

$$\hat{y}\left(\mathbf{x}^*\right) = \mu + \mathbf{r}^T R^{-1} \left(\mathbf{y} - \mathbf{1}\mu\right) \tag{3.8}$$

where $\mathbf{r}$ is an m-sized vector representing the correlations between the trial point $\mathbf{x}^*$ and the sample points used to build the surface. To verify that the method indeed interpolates the sample data, assume that $\mathbf{x}^*$ is the i-th sample point ($\mathbf{x}^{(i)}$). In that case, $\mathbf{r}$ will simply be the i-th column of the correlation matrix. So, the term $\mathbf{r}^T R^{-1}$ can be expressed as follows:

$$\mathbf{r}^T R^{-1} = \left(R^{-1}\mathbf{r}\right)^T = \left(R^{-1} R_i\right)^T = \mathbf{e}_i^T \tag{3.9}$$

where $\mathbf{e}_i$ is the i-th unit vector. (3.8) then reduces to:

$$\hat{y}\left(\mathbf{x}^{(i)}\right) = \mu + \mathbf{e}_i^T \left(\mathbf{y} - \mathbf{1}\mu\right) = \mu + \left(y^{(i)} - \mu\right) = y^{(i)} \tag{3.10}$$

So, the prediction at the i-th sample point indeed returns the actual observed value $y^{(i)}$.

Kriging is an attractive method since it is capable of approximating highly nonlinear functions, much more so than the traditional RBFs, as it allows each direction to be weighted differently when calculating distances. In addition, the description of Kriging from the statistical viewpoint, i.e. assuming that a prediction is a realization of a stochastic process, allows one to quantify the uncertainty of a prediction. This feature will be of benefit when discussing the use of Kriging in the context of *global* optimization, which is the subject of the next section. The mean squared error of the predictor at a point $\mathbf{x}^*$ is denoted by the following equation (again, the reader is referred to Sacks *et al.* [43] for a derivation of this formula):

$$s^2\left(\mathbf{x}^*\right) = \sigma^2\left[1 - \mathbf{r}^T R^{-1}\mathbf{r} + \frac{\left(1 - \mathbf{1}^T R^{-1}\mathbf{r}\right)^2}{\mathbf{1}^T R^{-1}\mathbf{1}}\right] \tag{3.11}$$

The term $-\mathbf{r}^T R^{-1}\mathbf{r}$ represents the reduction in the prediction error due to the fact that $\mathbf{x}^*$ is correlated with the sampled points. The term $\left(1 - \mathbf{1}^T R^{-1}\mathbf{r}\right)^2 / \mathbf{1}^T R^{-1}\mathbf{1}$ represents the uncertainty that comes from the fact that μ is not known exactly, but instead is estimated from the data. Since the method interpolates this data, the error at a sampled point *must* be zero, i.e. $s^2\left(\mathbf{x}^{(i)}\right) = 0$. Indeed, since $R^{-1}\mathbf{r} = \mathbf{e}_i$ as given by (3.9), $\mathbf{r}^T R^{-1}\mathbf{r}$ becomes:

$$\mathbf{r}^T R^{-1}\mathbf{r} = \mathbf{r}^T\mathbf{e}_i = r_i\left(\mathbf{x}^*\right) \equiv Corr\left(\mathbf{x}^*, \mathbf{x}^{(i)}\right) = Corr\left(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}\right) = 1 \tag{3.12}$$

Additionally,

$$\mathbf{1}^T R^{-1}\mathbf{r} = \mathbf{1}^T\mathbf{e}_i = 1 \tag{3.13}$$

Substituting (3.12) and (3.13) into (3.11) therefore gives that $s^2\left(\mathbf{x}^{(i)}\right) = 0$.

3.2.2 Kriging: the implementation

A Matlab implementation of Kriging, developed by Lophaven *et al.* [32], was used in the present work[2]. In the following, it will be referred to as *dacefit*, which is the name of the Matlab script that generates the Kriging surface, given a set of sample data. Once a Kriging surface has been generated, predictions can be made using the script called *predictor*. The program comes with three different types of regression functions; a constant, a linear function and a quadratic. The user however has total freedom in supplying a custom made regression function. Recall that in the previous section, the correlation between two points was represented with an exponential function (see (3.4)). Like the traditional RBF method, it is possible to define other types of correlation functions or (basis functions). *Dacefit* comes with 7

[2]The source code is freely available at URL: `http://www2.imm.dtu.dk/~hbn/dace/`

Table 3.1 Correlation models available in *dacefit* ($d_j = w_j - x_j$).

Name	$\mathbf{R_j}(\boldsymbol{\theta_j}, \mathbf{d_j})$
EXP	$\exp(-\theta_j \lvert d_j \rvert)$
EXPG	$\exp\left(-\theta_j \lvert d_j \rvert^{\theta_{k+1}}\right), 0 < \theta_{k+1} \leq 2$
GAUSS	$\exp\left(-\theta_j d_j^2\right)$
LIN	$\max[0, 1 - \theta_j \lvert d_j \rvert]$
SPHERICAL	$1 - 1.5\xi_j + 0.5\xi_j^3, \xi_j = \min[1, \theta_j \lvert d_j \rvert]$
CUBIC	$1 - 3\xi_j^2 + 2\xi_j^3, \xi_j = \min[1, \theta_j \lvert d_j \rvert]$
SPLINE	$\zeta(\xi_j)$

different choices, as summarized in Table 3.1. (3.14) shows how the spline function ζ is defined, where $\xi_j = \theta_j \lvert d_j \rvert$.

$$\zeta(\xi_j) = \begin{cases} 1 - 15\xi_j^2 + 30\xi_j^3, & 0 \leq \xi_j \leq 0.2 \\ 1.25(1 - \xi)^3, & 0.2 < \xi_j < 1 \\ 0, & \xi \geq 1 \end{cases} \tag{3.14}$$

If there are n dimensions, the correlation R between two points is the product of n one-dimensional correlations:

$$R(\theta, \mathbf{w}, \mathbf{x}) = \prod_{j=1}^{n} R_j(\theta_j, w_j - x_j) \tag{3.15}$$

The correlation functions can be separated into two groups. The first group consists of the functions that have a parabolic behavior near the origin (GAUSS, CUBIC and SPLINE) and the second group consists of the functions that have a linear behavior near the origin (EXP, LIN and SPHERICAL). Depending on the last parameter θ_{k+1}, the general exponential (EXPG) can either belong to the first group ($\theta_{k+1} = 2$) or the second group ($\theta_{k+1} = 1$), where it becomes GAUSS or EXP, respectively. The functions in the first group have a continuous gradient at the origin and are smoother than the functions in the second group, which have a discontinuous gradient at the origin. Figure 3.1 illustrates the correlation functions from the two groups.
As discussed in the previous section, the correlation functions should be 1 when the distance is zero (correlation of a point with itself) and decrease towards zero with increasing distance. The correlation functions of *dacefit* indeed have these

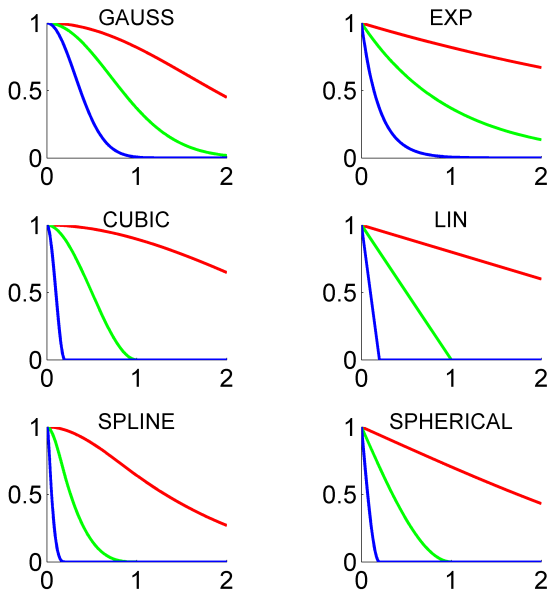


Figure 3.1 Correlation functions for $0 \leq d_j \leq 2$ with $\theta_j = 0.2$ (red), $\theta_j = 1$ (green) and $\theta_j = 5$ (blue).

properties. Additionally, it can be observed that large values for θ_i lead to a faster decrease. θ_i basically represents how far a sample point's influence extends. A low θ_i means that the sample points have a high correlation in the i-direction, while a high θ_i means that there is a considerable difference between the sample points in this direction. For this reason, θ_i can be considered to be a measure of how 'active' a certain variable is. It is helpful to look at the correlation parameters in this way, especially in high-dimensional problems where it is difficult to visualize the design landscape and the effect of the variables is unknown. By examining the elements of $\boldsymbol{\theta}$ one can determine which variables are the most important and maybe even eliminate the unimportant ones in future considerations.
The rest of this section describes how *dacefit* calculates the model parameters $\boldsymbol{\mu}$, σ^2 and $\theta_1, ..., \theta_n$ when a set of sample points has been given. The discussion follows the description by Lophaven *et al.* [31]. Given m sample points in an n dimensional space ($\bar{S} \in \mathbb{R}^{m \times n}$) with their corresponding function values $\bar{Y} \in \mathbb{R}^{m \times 1}$, the first step is to normalize the data:

$$\begin{aligned} S_{:,j} &= \frac{\bar{S}_{:,j} - S_{j,mean}}{S_{j,std}} \\ Y &= \frac{\bar{Y} - Y_{mean}}{Y_{std}} \end{aligned} \tag{3.16}$$

where $S_{j,mean}$ and $S_{j,std}$ are the mean and standard deviation of the j-th variable, respectively and Y_{mean} and Y_{std} are the mean and the standard deviation of the output, respectively. All subsequent computations are carried out with the normalized data which will have a mean of zero and a variance of one in each coordinate. The next step is to calculate the regression matrix $F \in \mathbb{R}^{m \times t}$. The i-th row contains the t elements of the regression function evaluated at the i-th sample point. The i, j-th element of the correlation matrix $R \in \mathbb{R}^{m \times m}$ was defined in (3.15). Note that the correlation matrix is a function of $\boldsymbol{\theta}$. The following regression problem is formed:

$$F\boldsymbol{\mu} \approxeq Y \tag{3.17}$$

where $m \geq t$ to ensure that the system is not underdetermined. The generalized least squares solution was already given by (3.6) in the case that a constant regression term was used and therefore F was a vector of ones and μ was a scalar. In the general case, the solution will be:

$$\boldsymbol{\mu} = \left(F^T R^{-1} F\right)^{-1} F^T R^{-1} Y \tag{3.18}$$

(3.7) then becomes:

$$\sigma^2 = \frac{1}{m} \left(Y - F\boldsymbol{\mu}\right)^T R^{-1} \left(Y - F\boldsymbol{\mu}\right) \tag{3.19}$$

Even though (3.18) and (3.19) contain the inverse of the correlation matrix and the inverse of $F^T R^{-1} F$, they should *not* be calculated explicitly to obtain the solutions. Calculating the inverse of a matrix in order to solve a system of linear equations can easily lead to inaccurate results if the matrix to be inverted is ill-conditioned. Instead, *dacefit* uses a better procedure. The first step consists of calculating the Cholesky factorization of the correlation matrix:

$$R = CC^T \tag{3.20}$$

where C is a lower triangular matrix with positive diagonal entries. The Cholesky factorization is only possible if the correlation matrix is symmetric and positive-definite. The first condition is a given because of the way the correlation matrix is defined; the correlation between sample points i and j is of course equal to the correlation between sample points j and i. Positive-definiteness however is not a guarantee. For example, if there are n variables and two of them are perfectly correlated, then the correlation matrix will not have n independent columns and therefore the matrix is singular, i.e. not positive-definite. If the correlation matrix is found not to be positive-definite, *dacefit* aborts. Writing R in its factorized form in (3.18) produces the following expression:

$$\boldsymbol{\mu} = F^{-1}CC^{T}F^{-T}F^{T}C^{-T}C^{-1}Y \tag{3.21}$$

which reduces to:

$$\boldsymbol{\mu} = \left(C^{-1}F\right)^{-1}C^{-1}Y \tag{3.22}$$

The following two matrices are defined:

$$\begin{array}{rcl} \tilde{F} & = & C^{-1}F \\ \tilde{Y} & = & C^{-1}Y \end{array} \tag{3.23}$$

Since C is a lower triangular matrix, $\tilde{F}$ and $\tilde{Y}$ can simply be found by forward substitution. (3.22) then reduces to a regression problem which can be solved by solving the normal equations:

$$\tilde{F}^{T}\tilde{F}\boldsymbol{\mu} = \tilde{F}^{T}\tilde{Y} \tag{3.24}$$

To reduce effects of rounding errors when the correlation matrix is very ill-conditioned, the QR factorization of $\tilde{F}$ is computed:

$$\tilde{F} = QG^{T} \tag{3.25}$$

where Q has orthonormal columns and G^{T} is upper triangular. Substituting (3.25) into (3.24) gives, after some cancellations:

$$G^{T}\boldsymbol{\mu} = Q^{T}\tilde{Y} \tag{3.26}$$

$\boldsymbol{\mu}$ can then be found by simple back substitution. Because of the Cholesky factorization and the definition of the matrices in (3.23), σ^2 can simply be calculated as follows:

$$\sigma^{2} = \frac{1}{m}\left|\tilde{Y} - \tilde{F}\boldsymbol{\mu}\right|^{2} \tag{3.27}$$

Where $|\cdot|$ represents the determinant.
Of all operations carried out by *dacefit*, the Cholesky factorization dominates the computational effort. As the correlation matrix is an $m \times m$ matrix, the factorization uses $\mathcal{O}\left(m^{3}\right)$ flops. The backward and forward substitutions carried out to solve the systems in (3.23) on the other hand take $\mathcal{O}\left(m^{2}\right)$ flops. The QR factorization is not so expensive. If t regression terms are used, it is an $\mathcal{O}\left(tm^{2}\right)$ process.
To find the optimum correlation parameters, *dacefit* carries out a box constrained

non-linear optimization. The objective function optimized is the maximum likelihood estimator. For every distribution a likelihood function can be set up. For the distribution considered here (a multivariate normal distribution) the likelihood function can be expressed as follows:

$$\frac{1}{(2\pi)^{\frac{m}{2}}\left(\sigma^2\right)^{\frac{m}{2}}\left|R^{\frac{1}{2}}\right|}\exp\left(\frac{-\left(Y-F\boldsymbol{\mu}\right)^T R^{-1}\left(Y-F\boldsymbol{\mu}\right)}{2\sigma^2}\right) \tag{3.28}$$

By taking the natural logarithm of (3.28), the maximization process can be simplified without modifying the resulting parameters. Of course, constant terms can be ignored in the optimization process;

$$-\frac{m}{2}\ln\left(\sigma^2\right)-\frac{1}{2}\ln\left(|R|\right)-\frac{\left(Y-F\boldsymbol{\mu}\right)^T R^{-1}\left(Y-F\boldsymbol{\mu}\right)}{2\sigma^2}+const \tag{3.29}$$

Inserting the values of $\boldsymbol{\mu}$ and σ^2 (from (3.18) and (3.19)) into (3.29) gives the so-called concentrated log likelihood function:

$$-\frac{m}{2}\ln\left(\sigma^2\right)-\frac{1}{2}\ln\left(|R|\right) \tag{3.30}$$

Instead of maximizing (3.30), *dacefit* minimizes the following function:

$$\psi=|R|^{\frac{1}{m}}\sigma^2 \tag{3.31}$$

It can be verified that (3.30) and (3.31) will return the same optimum set of θ's by taking the natural logarithm of (3.31) and multiplying the result with $-m$. *Dacefit* uses a modified version of the Hooke-Jeeves method for the optimization.

3.3 Exploring and exploiting the surrogate model

Once the surrogate model has been built and its accuracy has been deemed satisfactory, it is ready to do its duty, namely to act as a cheap replacement to the expensive code. The two uses of interest in the present context are *gaining an understanding* of the problem and using the response surface to *find an optimum design*. One can gain an understanding of the problem by creating suitable plots like parameter sweeps and contour curves. The rest of this section describes the use of surrogates in an optimization framework

Starting with an initial response surface, built using results obtained from a DOE, the optimum on the response surface is searched. The expensive code is then evaluated at this point, the response surface is updated and the optimum is found on the new surface. The expensive code is then evaluated at the new point and the response

surface is updated again. This process is repeated until a maximum number of expensive function evaluations is reached, or some user specified convergence criterion has been met. As the surrogate model is cheap to evaluate, global optimization algorithms like branch-and-bound, genetic algorithms and simulated annealing can be used to find the optimum. The algorithm used in this research for finding the global optimum on the response surface is the global optimizer DIRECT. A description of how it works can be found in Appendix A. The next sub section describes a number of criteria that can be used to choose the next sample point to be evaluated, i.e. the 'infill criteria'.

3.3.1 Infill criteria

Kind of Response Surface			Method for selecting search points					
			Two-stage approach: first fit a surface, then find the next iterate by optimizing an auxiliary function based on the surface				One-stage approach: evaluate hypotheses about optimum based on implications for the response surface	
			Minimize the Response Surface	Minimize a Lower Bounding Function	Maximize the Probability of Improvement	Maximize Expected Improvement	Goal seeking: find point that achieves a given target	Optimization: find point that minimizes an objective
Not interpolating (smoothing)	Quadratic polynomials and other regression models		1					
Interpolating	Fixed basis functions. NO statistics.	Thin-plate splines, Hardy multiquadratics	2				6	7
	Tuned basis functions. Statistical interpretation	Kriging		3	4	5		

Figure 3.2 A taxonomy of response-surface-based global optimization methods. (From Jones [24])

The most obvious and intuitive way of choosing the next sample point is to determine the actual best function value on the response surface. However, Jones [24] shows that it can cause the method to get stuck in a local optimum. Figure 3.6 illustrates this using a one-dimensional function. It has two minima: one at $x \approx 0.14$ and the other at $x \approx 0.76$. The latter is the global minimum. To avoid getting stuck in a local minimum, a number of infill criteria have been proposed in the open literature. The surrogate modeling technique of choice in the present research is Kriging and Jones discusses a set of infill criteria based on using this particular modeling technique. Figure 3.2 summarizes the various methods available, as extracted from the mentioned article. For a discussion of a global optimization framework using the standard RBFs, the interested reader is referred to Gutman [16]. From the figure it can be determined that the method illustrated in Figure 3.6 is Method 2. The various infill criteria are grouped according to the metamodeling technique used and the method used for selecting infill points.
Regarding the method to select infill points, a subdivision is made between the

two-stage approach and the *one-stage approach*. The two-stage approach first fits a surface through the sample data (by determining the optimum model parameters), followed by choosing the next iterate by optimizing an auxiliary function based on the surface (in the case of Method 2, the auxiliary function is the function value on the response surface itself). Jones mentions that the fundamental flaw of this approach is that it assumes that the response surface obtained from the first stage is accurate. As a result of this it can be deceived when the initial sample is sparse and gives a highly misleading view of the function.

One-stage methods (methods 6 and 7 in Figure 3.2) on the other hand avoid estimating the model parameters based only on the observed sample set. The basic idea is to first define a hypothesis about the optimum: 'the optimum function value is f^* and obtained at $\mathbf{x}^*$'. Then, the surface most likely to go through the sample points *and* this optimum is sought by optimizing the model parameters with the point $\mathbf{x}^*$. So, the number of optimization parameters is extended with $\mathbf{x}^*$. Therefore, the response surface is built in one step. The method described here is method 6 from Figure 3.2. Method 7 is similar to method 6 with the modification that instead of defining one f^*, a range of these values are evaluated. Figure 3.3 gives an example of Method 6. In principle, one-stage methods are better than two-stage methods. The disadvantage of one-stage methods is that they are more difficult to optimize.

In the present framework, the response surfaces will not only be used for optimization. Rather, once a DOE has been carried out the first focus will be to understand the design problem. To this end, a response surface will be built and suitable plots will be made to explore the relevant functions. Once the user is confident that the problem was set up correctly and a fair understanding of it has been achieved, the next step will be to use the response surface for optimization purposes, i.e. to find the best configuration. This type of process naturally lends itself to the two-stage process rather than the one-stage process, as the response surface will already have been built.

The metamodeling techniques are subdivided into two groups. The first consists of techniques that *regress* data points and the second consists of *interpolating* techniques. According to the table, method 1 is the only suitable method for regressing techniques. It is the intuitive method of choosing the infill point to be the optimum on the response surface. So, clearly these techniques are not suitable when it is crucial to find the global optimum as they can easily get stuck in a local optimum.

The second group of metamodeling techniques consists of the interpolating methods. This group is subdivided into the fixed basis functions (the traditional RBF method) and the *tuned* basis function method Kriging.

The final three methods given in Figure 3.2, methods 3, 4 and 5, are reserved for Kriging. The reason is that these methods rely on the metamodeling technique's ability to quantify a prediction error at a given point, in addition to the prediction itself. As explained in the previous section, a prediction made by Kriging can be thought of as being a realization of a stochastic process. This does not mean that the output is a random process. On the contrary, they are obtained from *deterministic* computer experiments.

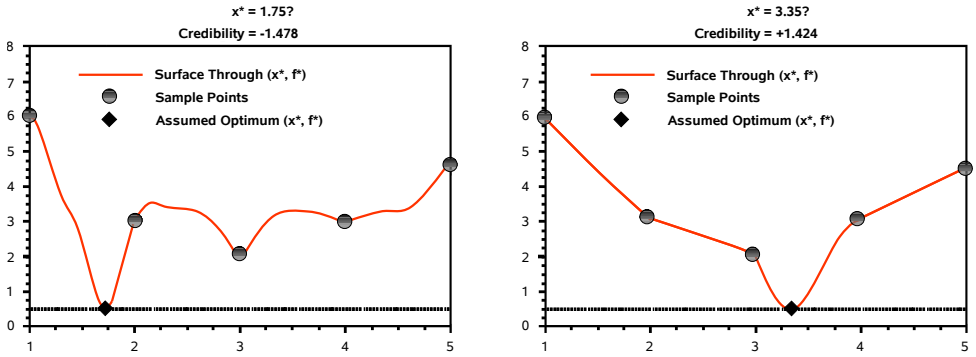


Figure 3.3 In Method 6, the credibility of the hypothesis that the surface passes through (x^*, f^*) is based on the likelihood of the data conditional upon the surface passing through this point. The figure compares hypotheses for two possible values of x^*, both with the same value of f^*. (From Jones [24])

The statistical view simply states something about the uncertainty in the predictions themselves. So, next to the predicted value at a certain design point, given by (3.8), one can also calculate the uncertainty about this prediction; recall the mean squared error, given in (3.11). Note that methods 1 and 2 assume that the prediction is correct and therefore they simply use the predicted value as the auxiliary function for choosing the next sample points. If this assumption is true, then if an appropriate global optimization algorithm is applied on the response surface, the actual global optimum will eventually be found. However, if the assumption is not true, then there is a chance that the method will get stuck in a local minimum. Instead of stating that the prediction is correct, one can acknowledge the fact that there is some uncertainty present. So, instead of using the predicted value as the objective function, one can combine it with the uncertainty.
The first method discussed that makes use of the prediction error is method 3, proposed by Cox and John [6]. The auxiliary function is the so-called 'statistical lower bound' and is defined as follows:

$$\hat{y}(\mathbf{x}) - \kappa \cdot s(\mathbf{x}) \tag{3.32}$$

where $\hat{y}(\mathbf{x})$ is the predicted value and $s(\mathbf{x})$ is the root mean squared error of the prediction. For strictly positive values of κ the method will put some emphasis on searching in relatively un-sampled regions of the design space where the prediction error will be high. The higher the value of κ, the more global the search will be. The disadvantage of this method is that *in the limit*, the iterates will not be dense. In other words, not all possible points in the design space will eventually be sampled. A theorem by Torn and Žilinskas [51] states that in order to converge to a global optimum for a general continuous function, the sequence of iterates must be dense.

A more popular method, originally proposed by Kushner [28], defines the probability of improving the function beyond some target T. The infill point will be the point that maximizes this utility function. As a prediction made by Kriging is assumed to be a realization of a stochastic process $Y(\mathbf{x})$ with the mean given by (3.8) and the error given by (3.11), the probability of improvement can be calculated as follows:

$$\Phi\left(\frac{T-\hat{y}(\mathbf{x})}{s(\mathbf{x})}\right) \tag{3.33}$$

where $\Phi(\cdot)$ is the Normal cumulative distribution function. Contrary to method 3, the iterates generated by method 4 will be dense, as proved by Gutman [16]. Indeed, as sample points will start to cluster around a point, the error in that region will become small meaning that the probability of finding an improvement there will become small too. The algorithm will therefore be driven to search elsewhere in the design space.
The last method described aims at maximizing a utility function known as the expected improvement. This is method 5 in Figure 3.2 and it is the method implemented in this thesis.

3.3.2 EGO: Efficient Global Optimization

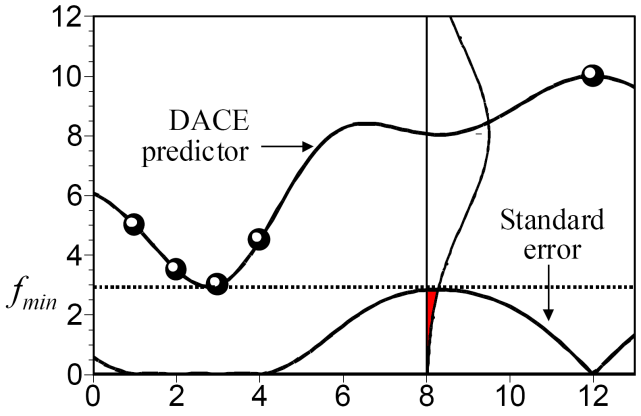


Figure 3.4 The uncertainty about the function's value at a point (such as $x = 8$ above) can be treated as if the value there was a realization of a normal random variable with mean and standard deviation given by the DACE predictor and its standard error. (From Jones *et al.* [26])

Maximizing the expected improvement has been made popular by Jones *et al.* [26]. They called the algorithm EGO, which is short for *Efficient Global Optimization*. Locatelli [30] proved that under mild assumptions the iterates generated by the method are dense, meaning that eventually all points in the design space will be sampled. This means that the method *will* find the global optimum (eventually). To calculate the expected improvement, the best point among the sample points

has to be identified first. In case of minimization: $f_{\min} = \min(y_1, ..., y_m)$ (refer to Figure 3.4). Again, interpret Kriging as if it is modeling the uncertainty at $\hat{y}(\mathbf{x})$ by treating it as the realization of a normally distributed random variable Y with mean and standard deviation given by (3.8) and (3.11) respectively. For example, at the point $x = 8$, a normal density function is shown. Its mean is at the value predicted by the response surface and its width depends on the error at that point. A wider function corresponds to higher uncertainty. By treating the prediction at $x = 8$ as a realization of the random variable Y with the density function shown, there will be a chance that the function $x = 8$ is an improvement over the current best value $f_{\min}$. This is conceivable since the density function extends below the line $y = f_{\min}$ (the area colored in red). If the improvement is defined as follows:

$$I = \max(f_{\min} - Y, 0) \tag{3.34}$$

then the expected improvement is obtained by taking the expected value of (3.34):

$$E[I(\mathbf{x})] = E[\max(f_{\min} - Y, 0)] \tag{3.35}$$

It turns out that (3.35) can be expressed in closed form:

$$E[I(\mathbf{x})] = (f_{\min} - \hat{y})\,\Phi\left(\frac{f_{\min} - \hat{y}}{s}\right) + s\phi\left(\frac{f_{\min} - \hat{y}}{s}\right) \tag{3.36}$$

where $\Phi(\cdot)$ and $\phi(\cdot)$ are the standard normal *cumulative* and *probability* distribution functions, respectively. Looking at (3.36), the following three interesting observations are made:

1. the expected improvement will be 0 at a sample point. As the technique interpolates the data, the error at a sample point i is 0. This makes the second term in (3.36) equal to 0. The associated function value at i is y_i, with $f_{\min} \leq y_i$. The argument in $\Phi(\cdot)$ is therefore $-\infty$, which renders it equal to 0. So, the first term in (3.36) will be equal to 0 too. The fact that the expected improvement will be 0 at a sample point guarantees that no re-sampling will occur. In the limit, the iterates will therefore be dense. For this reason, the method is guaranteed to find the global optimum;

2. the first term in (3.36) is the difference between the current minimum and the predicted value multiplied by the probability that the function value will be smaller than $f_{\min}$. It is therefore large where the prediction is likely to be smaller than $f_{\min}$. The second term tends to be large where there is high uncertainty about whether or not the prediction will be better than $f_{\min}$, i.e. unexplored regions. The utility function can therefore be thought of as providing a trade-off between local and global search;

3. contrary to the 'statistical lower bound' method given in (3.32) and the probability of improvement in (3.33), no user-supplied parameter is needed.

Figure 3.7 shows the method implemented on the same 1D function as was given in Figure 3.6. In addition to the true function and the Kriging surface, the plots also show the expected improvement. This is the objective function maximized in the background. At the first iteration, the infill point is the same as for method 2, i.e. the minimum on the response surface. This is because the expected improvement there is at its maximum. Note that this is the only maximum at the considered interval. At the second iteration, the expected improvement has three maxima. Two of these are in the vicinity of the global minimum of the response surface. However, the global maximum of the expected improvement occurs at a different point, where the design space has not been explored extensively yet. For this reason, the infill point is at that position. When the point is evaluated on the real function, it is found that the function value found is considerably better than the already sampled points. The new function value found becomes the new $f_{\min}$. In the next three iterations, the infill points are all in the vicinity of the global minimum of the response surface (around $x = 0.76$). In the sixth iteration however, the method samples at a different point as the response surface has been sampled extensively around $x = 0.76$ and the expected improvement there has therefore diminished. So, when examining the infill points chosen it can clearly be seen that the method tries to strike a balance between local and global search, which confirms point 2 stated above. Point 1 can be confirmed by examining the 'double bumps' at Figures 3.7(b) and 3.7(c). They can be thought of as being one bump 'pinched' to zero in the middle by the presence of the sample point there. Also, note how the maximum expected improvement starts out around 6E-1 and in the sixth iteration has dropped to 1E-11.

Since EGO tries to balance local and global search, the sampling of the space can be quite erratic. To define a stopping criterion for the method it is therefore undesirable to monitor the convergence of the objective function value or the parameter values. The most intuitive stopping criterion in that case would be the expected improvement value. If it drops below a certain threshold, the method can be stopped. To illustrate this, a two-dimensional test function is considered, as given in Figure 3.8(a). The test function has four minima. The global minimum is denoted by the cross. Figure 3.8(b) gives the initial response surface built using 10 sample points distributed by an Optimal Latin Hypercube algorithm. 20 iterations are carried out using Method 2 and EGO. Figures 3.8(c) and 3.8(e) give the iterates at iteration 10 and 20 for Method 2, respectively. Note that the figures are identical. This is because the method converged after 9 iterations. Subsequent iterates were duplicate points so the method stalled; it converged to one of the *local* minima. Figures 3.8(d) and 3.8(f) give the iterates at iterations 10 and 20 for EGO, respectively. After 10 iterations it seems to be converging to the same local minimum as Method 2 but it clearly escapes getting stuck as at iteration 20 it is seen to have located the global minimum. Figure 3.5 plots the function values and maximum improvement

as functions of the iteration number. Whereas the function value is not converging to any value, the maximum expected improvement is clearly converging to 0. So, when using EGO it makes more sense to use the maximum expected improvement for assessing convergence.

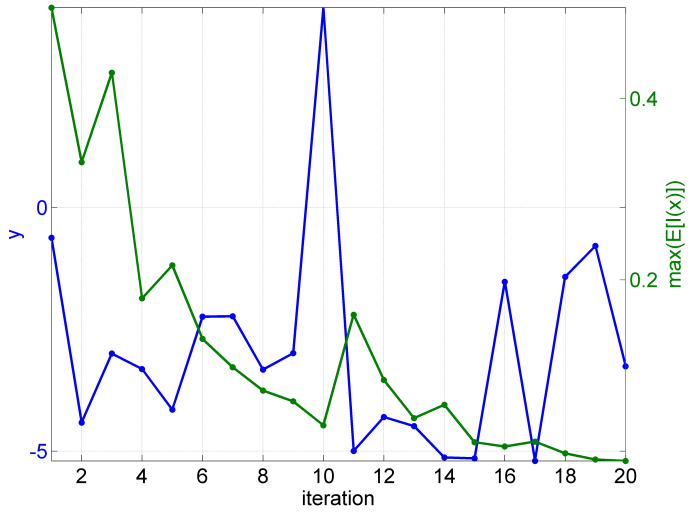


Figure 3.5 Convergence of EGO in terms of sample function value (blue line) and the maximum expected improvement (green line), applied to the function from Figure 3.8.

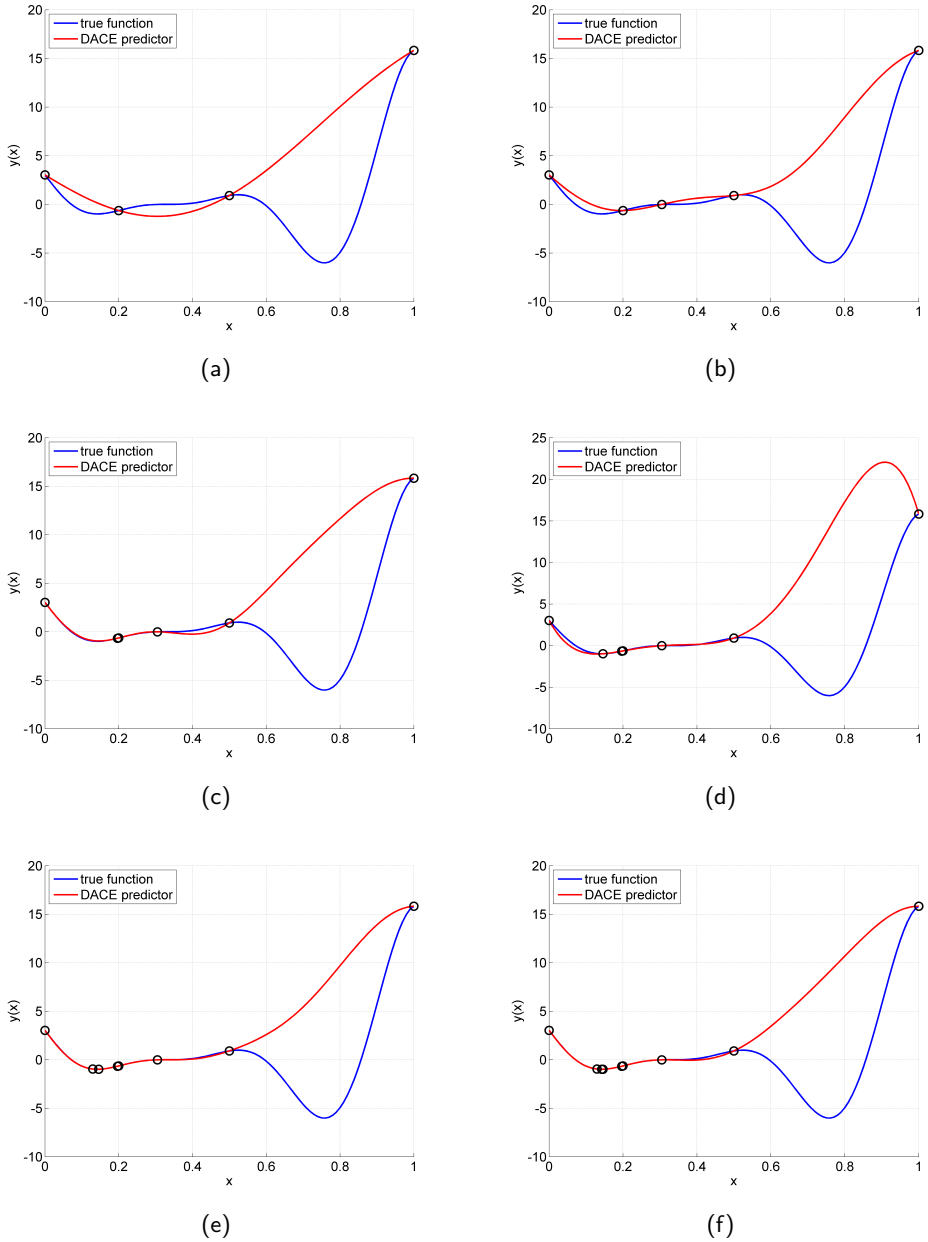


Figure 3.6 Optimization of the function $y(x) = (6x-2)^2 \sin(12x-4)$ using the global minimum on the response surface as the infill point.

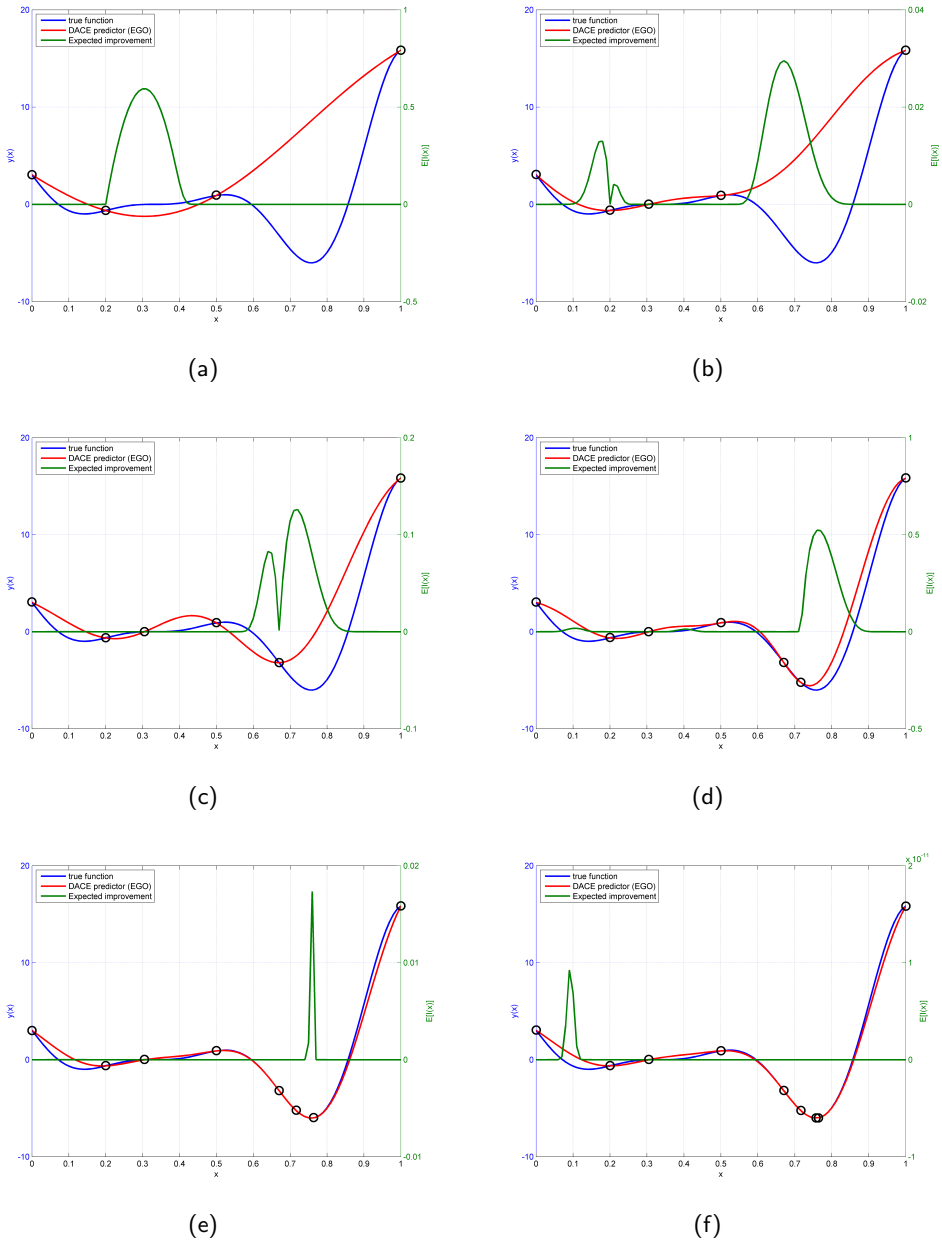


Figure 3.7 Optimization of the function $y(x) = (6x-2)^2 \sin(12x-4)$ using the maximum of the expected improvement as the infill point.

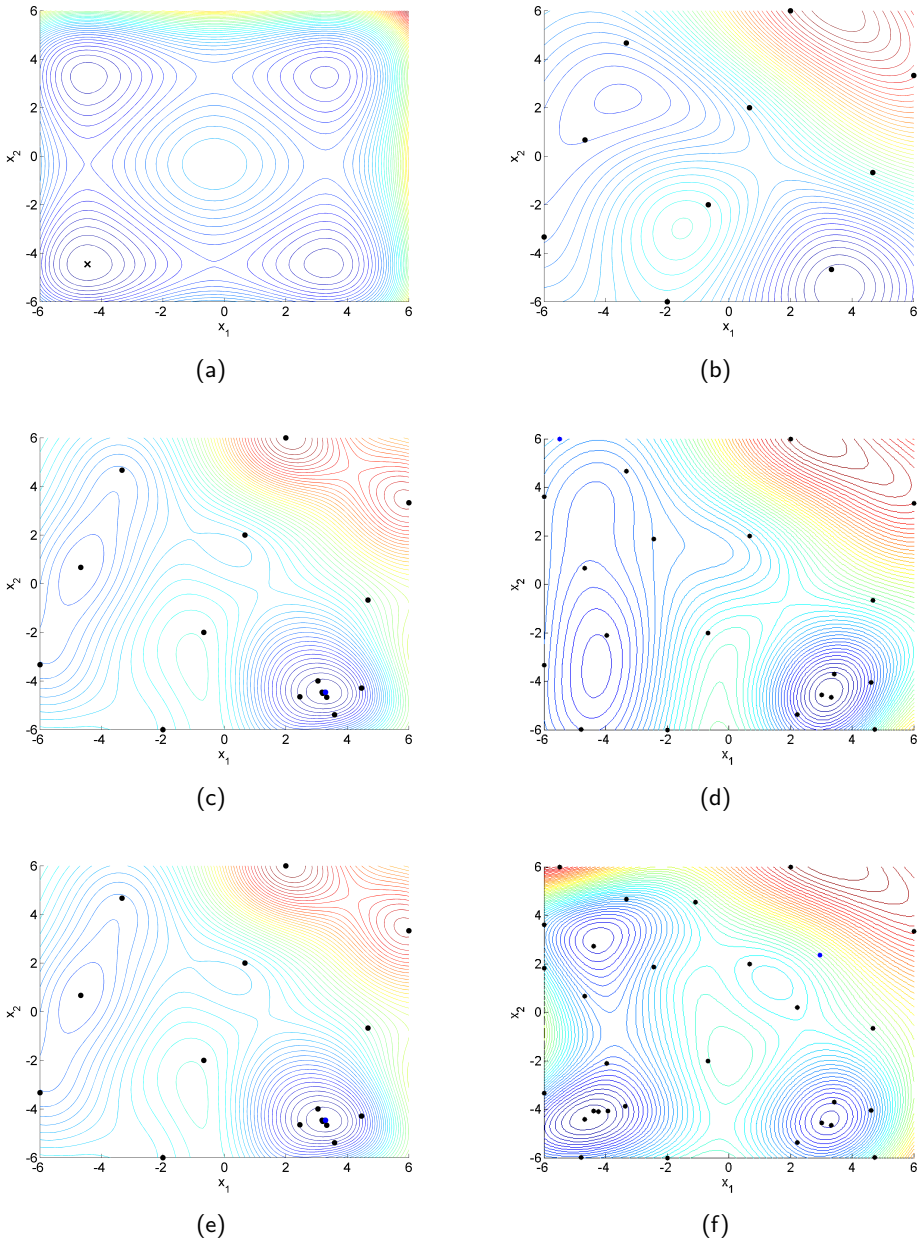


Figure 3.8 The two dimensional test function $y(\mathbf{x}) = \sum_{j=1}^{2} 0.01\left[(x_j + 0.5)^4 - 30x_j^2 - 20x_j\right]$ with the global minimum at the black cross (3.8(a)); the initial response surface using 10 sample points (3.8(b)); response surfaces after 10 (3.8(c)) and 20 (3.8(c)) iterations for Method 2; response surfaces after 10 (3.8(d)) and 20 (3.8(f)) iterations for EGO.

'Do not follow where the path may lead. Go instead where there is no path and leave a trail.'
George Bernard Shaw

4 Improvements to DACE

This chapter presents two contributions made to DACE. They are both based on improving the quality of the response surface. It should be clear why obtaining an accurate response surface is crucial. Not only will it speed up convergence in an optimization loop, it will also give a more accurate global representation of the expensive function, which is particularly important when the surrogate is used for visualization purposes. The first contribution is the use of leave-one-out cross-validation instead of the widely accepted maximum likelihood function when determining the optimum correlation parameters. The second contribution boils down to smoothing the DOE points rather than interpolating them, which is a useful technique when the function of interest is noisy.

4.1 How to determine the optimum correlation parameters

To find the optimum correlation parameters, the open literature suggests using the maximum likelihood estimator (MLE) as the objective function. This is indeed the route followed by *dacefit*. As the optimizer, it uses a modified version of the Hooke-Jeeves (HJ) method. Since the algorithm will only find a local optimum, it was thought that if one uses a global algorithm, the response surfaces acquired should naturally be of better quality. To this end, the global algorithm DIRECT[1] was used in the optimization process. To verify whether this modification would indeed provide better response surfaces, the interpolated version of CFD test case 1 (discussed in Chapter 6) was used. For a number of different sample sizes, response surfaces were built using HJ and DIRECT. The quality of the response surfaces obtained with these two optimizers was then compared by calculating their errors on a fine 100×100 mesh of the actual function. Figure 4.1 shows the results. The blue line, corresponding to the left vertical axis, gives the difference between the minimum ψ found by DIRECT and HJ. As expected, DIRECT always finds a better optimum. The green line, corresponding to the right axis, gives the difference between the average error of the response surfaces obtained with DIRECT and HJ. Interestingly, even though DIRECT always finds a better optimum for ψ, this does *not* always translate into a better response surface. Another interesting observation is that

[1] see Appendix A for a description of the algorithm.

for the scarcest data set of 10 sample points, the difference in accuracy between the two response surfaces is the largest, in HJ's favor. These observations raised the question of whether there may be another, more suitable objective function that could be used for tuning the correlation parameters. The rest of this section introduces a new concept for tuning a Kriging surface and compares it with the traditional approach.

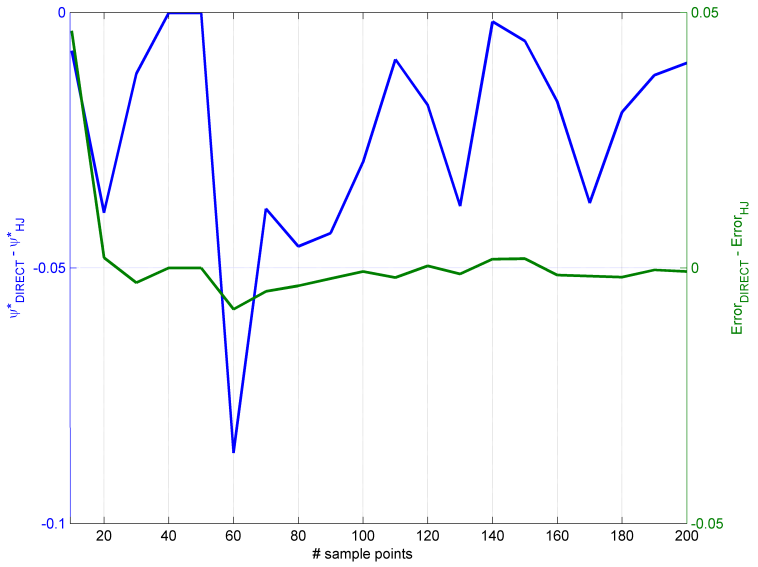


Figure 4.1 Comparison between DIRECT and HJ in terms of the optimum MLE (blue line) and response surface quality (green line).

4.1.1 Cross-validation based parameter tuning

Once a Kriging surface is constructed, it is ready to predict responses at un-sampled points. However, it is good practice to check the quality of the model first. For regression techniques, a commonly used method is to calculate a metric representing the deviation between the sample outputs and their predicted values. The so-called R^2 is a well known metric for this. The closer it is to 1, the better the fit. However, since DACE *interpolates* the sample data, it would not make sense to compare the predicted values at sample points with their corresponding output values. For an interpolation technique, it would make more sense to fit the sampled points and then test the predictions on an additional set of points. This process is known as holdout validation. However, when the computer experiments are particularly expensive, one may be reluctant to carry out additional simulations for validation purposes. In that case, cross-validation is a good alternative. Given a set of m sample points, the basic idea of cross-validation is to fit the model to $m - v$ of these points (the

sample set) and subsequently measure the prediction error for the v remaining points (the validation set). In total, there are $\begin{pmatrix} n \\ v \end{pmatrix}$ possibilities for doing this. For example, if there are 100 sample points and the validation set is chosen to contain 5 sample points, then there are 75,287,520 distinct cases. Since building a Kriging model is relatively expensive when compared to say linear regression, this is not an efficient way of validating the model. Another possibility is to use a *subset* of the combinations. In that case, it is best to randomly select which points to include in the validation set. Of course, care should be taken such that no identical cases are generated or only a certain part of the design space is used for validation. If one chooses v to be equal to 1, then there are m distinct scenarios. This is the case where $m-1$ points are used to build the model and the remaining point is used for validation. This special case is known as leave-one-out cross-validation. Indeed, this is the alternative to the MLE considered in this research. To use the cross-validation method as the objective function for finding the optimum correlation parameters, the following steps are carried out:

- Given $\theta_1, ..., \theta_n$
- For $i = 1...m$

  Build the Kriging model by excluding sample point i;

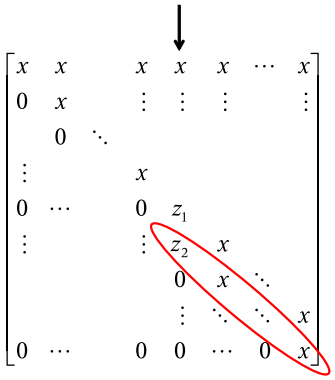
  Predict the output value of i;

  Calculate the error between the actual and predicted value;
- From the m error values, build a suitable metric (M), like the root-mean square error or the average error. This metric is the objective function.

The goal is then to minimize $M = M(\boldsymbol{\theta})$. As for the commonly used maximum likelihood estimation objective function, this is a non-linear optimization problem with box constraints.
As was mentioned at the end of the previous chapter, the Cholesky factorization dominates the computational effort when building a Kriging surface. If there are m sample points, then it uses $\mathcal{O}\left(m^3\right)$ flops. So, if the maximum likelihood estimator is used as the objective function, then getting it will cost $\mathcal{O}\left(m^3\right)$ flops in total. However, if leave-one-out cross-validation is used, then calculating the cross-validation error for a given set of correlation parameters will be an $\mathcal{O}\left(m^4\right)$ process, since a Kriging surface is built for each left out data point. For this reason, ways to speed up the process were investigated. In the present context, the computational cost of this step is insignificant compared to the CFD computations carried out and for this reason, one can simply follow the $\mathcal{O}\left(m^4\right)$ route. However, in other cases, where the objective function takes less wall clock time, reducing the time of the Kriging building process may become more important. In that case, it becomes crucial to obtain the cross-validation objective function in a more efficient way. This is described in the next sub section.

4.1.2 An efficient cross-validation scheme

Since the Cholesky factorization is the most expensive operation carried out, it is only natural to try and reduce its overhead first. Let's start out with the correlation matrix R. Its Cholesky factorization was given in (3.20). Say that observation i is removed from the training set. The correlation matrix of the training set can be obtained by removing the i-th column and row from R, resulting in the correlation matrix denoted by $R_{[-i]}$. Instead of recalculating the Cholesky factorization of this new matrix from scratch, it is possible to simply update the original factorization, as described by Gill *et al.* [14]. The first step is to remove column i from C^T:



Having done that, one can see that the resulting matrix is neither upper triangular nor square. To make it upper triangular, the entries circled have to become zero somehow[2]. This can be achieved by applying a series of Givens rotations to the matrix. Given a vector $\mathbf{x}$, the Givens rotation $J_k = J(k, k-1, \theta)$ zeroes out the k^{th} element of $\mathbf{x}$ and is given by:

$$\begin{array}{c} \begin{bmatrix} 1 & \vdots & \vdots & \\ \cdots & c & s & \cdots \\ \cdots & -s & c & \cdots \\ & \vdots & \vdots & 1 \end{bmatrix} \begin{matrix} \\ k-1 \\ k \\ \\ \end{matrix} \\ \begin{matrix} \\ k-1 & k & \\ \end{matrix} \end{array}$$

To convert the off-diagonal term z_2 to zero, c and s can be calculated as follows:

[2] Once this has been achieved, the last row must be removed to make the matrix square again.

$$\begin{array}{rcl} \rho^2 & = & z_1^2 + z_2^2 \\ c & = & z_1/\rho \\ s & = & z_2/\rho \end{array} \tag{4.1}$$

The entry z_1 then becomes ρ. Downdating an *existing* Cholesky factorization with the discussed sequence of Givens rotations is an $\mathcal{O}\left(m^2\right)$ process. This process is repeated m times, so the resulting algorithm is an $\mathcal{O}\left(m^3\right)$ process. The algorithm described here is therefore a clear improvement over the simple method of building a Kriging surface from scratch for each left out point.
If the problem is solved by means of a Cholesky factorization, which is the route followed in *dacefit*, then downdating the Cholesky factorization by means of a sequence of Givens rotations, as described earlier, is an efficient method when using cross-validation. If the generalized least squares problem is instead obtained by means of the matrix inversion route, then for this method there is also an efficient way of implementing cross-validation. Again, instead of calculating the inverse of the reduced correlation matrix from scratch for each left out sample point, it is possible to update the original inverted correlation matrix. Following Jolly *et al.* [23], the procedure works as follows. Starting out with the correlation matrix R of the full m sample points, the inverse R^{-1} is calculated and expressed as follows:

$$R^{-1} = \begin{bmatrix} A_{(i-1)\times(i-1)} & \mathbf{f}_{(i-1)\times 1} & B_{(i-1)\times(m-i)} \\ \mathbf{f}'_{1\times(i-1)} & \mathbf{e}_{1\times 1} & \mathbf{g}'_{1\times(m-i)} \\ B'_{(m-i)\times(i-1)} & \mathbf{g}_{(m-i)\times 1} & D_{(m-i)\times(m-i)} \end{bmatrix} \tag{4.2}$$

A new matrix H is constructed by removing the i-th row and column from R^{-1};

$$H = \begin{bmatrix} A & B \\ B' & D \end{bmatrix} \tag{4.3}$$

and a new vector is formed with $\mathbf{f}$ and $\mathbf{g}$:

$$\mathbf{k} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix} \tag{4.4}$$

The inverse of the correlation matrix with the i-th column and row removed can then be calculated as follows:

$$R^{-1}_{[-i]} = H - \frac{\mathbf{k}\mathbf{k}'}{\mathbf{e}} \tag{4.5}$$

Like the Cholesky factorization, calculating the inverse from scratch is an $\mathcal{O}\left(m^3\right)$ process, whereas updating the existing inverse as given in (4.5) is an $\mathcal{O}\left(m^2\right)$ process. Both routes (i.e. the Cholesky route and the matrix inversion route) were implemented in Matlab. Testing showed that updating the inverse is *much* faster than updating the Cholesky factorization. However, the disadvantage of the matrix inversion route is that in cases when there are sample points close to each other, the correlation matrix becomes ill-conditioned and therefore the results obtained are inaccurate. Indeed, as the correlation matrix becomes more ill-conditioned, it was found that the results returned by the two methods started to diverge. The Cholesky route is therefore the more robust implementation. When speed is key, it is possible to use the matrix inversion route as the default, but revert to the Cholesky route if the correlation matrix is ill-conditioned (this can be determined by calculating the condition number of the matrix).
As a side note, it is interesting to state that even though arithmetic operations are often used to compare algorithms with each other to determine which one is faster, in a practical implementation there are other factors that should be taken into account too. For example, operations like storing or accessing entries of a matrix or the presence of loops. The latter is an important factor when an interpretative language is used, like Matlab. Matlab is notorious for being slow when loops are used, as the arguments inside have to be interpreted each time the loop is entered. If possible, loops should be replaced by vectorization. However, when downdating the Cholesky factorization, a sequence of Givens rotations has to be applied and unfortunately this cannot be vectorized, as it is a recursive process. For this reason, the Cholesky route contains a loop. The matrix inversion route on the other hand does not contain a loop and this is probably the reason why its Matlab implementation was found to be so much faster. Genz *et al.* [13] give an interesting, detailed description of how an algorithm that in theory should be faster than another algorithm (Fast Givens QR factorization vs. ordinary Givens QR factorization) actually turns out to be slower when implemented in the interpretative environment Matlab. When they implemented the two methods in compiled FORTRAN however, they found that the program speeds were similar to the theoretically expected ones.

4.1.3 Numerical experiments

This sub section compares the quality of the Kriging surfaces obtained using the maximum likelihood estimate (MLE) and cross-validation (Xval). Eleven test functions were used to compare the two approaches. Table 4.1 gives the names of the test functions used with their dimensions and bounds. The problem 'branin' is the branin test function [4]. The problem 'camel' is the so-called six hump camel back test function [8]. Both these functions are often used to test global optimization algorithms. The test functions denoted with 'P' followed by a number were obtained from the Hock and Schittkowski [18] set which offers 180 problems for testing nonlinear optimization algorithms.

Table 4.1 Test functions to compare the MLE and Xval ideas for building Kriging surfaces.

No.	Name	Dim.	Bounds
1	branin	2	$0 \leq x_i \leq 10$
2	camel	2	$-2 \leq x_i \leq 2$
3	P5	2	$-3 \leq x_i \leq 3$
4	P26	3	$-1 \leq x_i \leq 1$
5	P42	4	$-5 \leq x_i \leq 5$
6	P44	4	$-5 \leq x_i \leq 5$
7	P48	5	$-5 \leq x_i \leq 5$
8	P93	6	$-1 \leq x_i \leq 1$
9	P100	7	$-1 \leq x_i \leq 1$
10	P108	9	$-1 \leq x_i \leq 1$
11	P113	10	$-1 \leq x_i \leq 1$

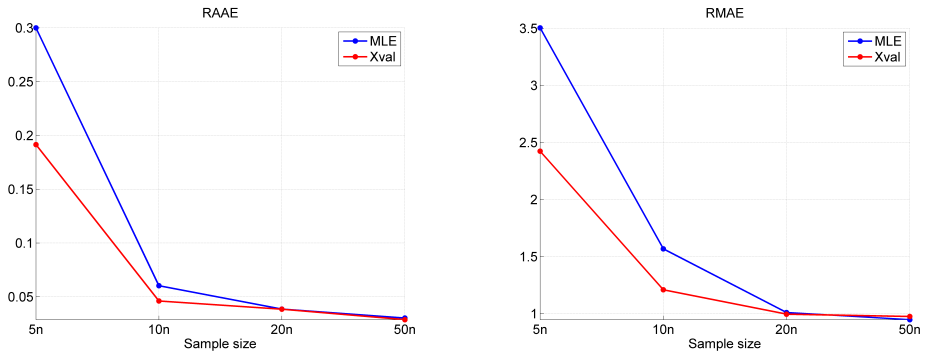
The DOE technique used to build the Kriging surfaces was iSIGHT's Optimal Latin Hypercube. Four different sample sizes were used for each test function: $5n$, $10n$, $20n$ and $50n$, where n is the number of dimensions. To find the optimum correlation parameters, the SQP algorithm of Matlab's fmincon routine was used. This algorithm was not designed to find global optima. For this reason, it was started at multiple points. Specifically, it was started at the $50n$ points obtained from the Optimal Latin Hypercube algorithm.
The quality of the surfaces is expressed using the following two metrics: the Relative Average Absolute Error (RAAE) and the Relative Maximum Absolute Error (RMAE). They are a measure for the global and local accuracy of the response surface, respectively and are defined as follows [21]:

$$RAAE = \frac{\sum_{i=1}^{k} |y_i - \hat{y}_i|}{k \cdot STD} \tag{4.6}$$

$$RMAE = \frac{\max\left(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, ..., |y_k - \hat{y}_k|\right)}{STD} \tag{4.7}$$

where $\hat{y}_i$ is the predicted value at a point, y_i is the actual value and STD is the standard deviation. Since Kriging interpolates the data, the errors at the sample points will be zero. For this reason, the errors were calculated at k other points in the space, distributed by means of a full factorial. The value of k was around $1E6$. The particular error used in the cross-validation method was the root mean squared error.
Figure 4.2 plots the RAAE and RMAE averaged over the test functions. The figure shows that for $5n$ and $10n$ sample points, the Xval objective function gives a more accurate response surface, especially for the $5n$ case. For $20n$ and $50n$ sample points, both criteria seem to give very similar surfaces. It should not come as a surprise that for larger sample sizes, the two objective functions give similarly

accurate surfaces as the choice of the values for the correlation parameters becomes less crucial. In fact, the observation that the advantage of Xval over MLE for large sample sizes drastically diminishes should be considered a good sign since the former is a more expensive method. The cost becomes more evident for larger sample sizes. Interestingly, recall that in Figure 4.1 it was observed that for the scarcest sample set, the globally optimal MLE value found by DIRECT actually gave a much larger error than the local optimum found by HJ. Again, in Figure 4.2 it is seen that for the scarcest sample set of $5n$ the correlation parameters corresponding to the best MLE value are clearly not as good as the ones at a different part of the correlation parameter space (the ones found by Xval). This is an indication that for scarce sample sets, the MLE may not be a very good objective function for tuning the correlation parameters.



(a) (b)

Figure 4.2 RAAE and RMAE as a function of the sample size for the MLE and Xval criteria, averaged over the 11 test functions.

4.2 How to fit noisy functions

A deterministic computer simulation will always return the same output if the input is left unchanged. In that case, interpolating surrogate modeling techniques are the conceptually correct approach to data fitting, since the surrogate model will exactly go through the sample points. In contrast, a surrogate model built using a regressing (or smoothing) method will not necessarily go exactly through all sample points. Regression techniques are therefore popular in approximating functions obtained from physical experiments, where measurement errors and other factors outside the influence of the experimenter may produce different output values at different times, even though the inputs are (thought to be) the same. These factors are referred to as experimental noise.

DACE, the focus of this thesis, is an interpolating method. In the present context, the deterministic computer simulations are high-fidelity CFD computations.

As mentioned in Chapter 2, these simulations may contain a certain degree of noise as well. Discretization errors and incomplete convergence are the two main sources. Contrary to the random nature of experimental noise, this *numerical* noise is reproducible. The next subsection shows how using an interpolating technique as DACE can be undesirable when the underlying function is noisy.

4.2.1 The downside of interpolation

Figure 4.3(a) shows the downforce generated by a two element airfoil[3] as a function of the flap angle. A sixth order polynomial fit of the sample points, shown by the black dotted line, reveals the familiar trend in which an increasing flap angle results in an increase in downforce up to a point where the flap stalls. Beyond that angle, the downforce created by the assembly starts to decrease with increasing flap angle. One can observe that the CFD results are positioned quite irregularly around this trend. It is these irregularities that are referred to as noise. Figure 4.3(b) shows two interpolating response surfaces. The green line was obtained using five sample points, namely at 30°, 32°, 36°, 39° and 40°. The red line was obtained using an additional sample point at 36.25°, which is very close to the sample point at 36°. Whereas the green line captures the trend of the underlying function well, the red line clearly does not. This is the downside of interpolation methods when the underlying functions are noisy since they will not only capture the actual trend but also the noise. When the sample points are sufficiently far from each other, which they will be when they are obtained from a DOE[4], then it is safe to use an interpolation technique. However, if the surrogate model is then used in an optimization loop and the sample points start to cluster as the method converges to an optimum, then it becomes undesirable to use such a technique. Figure 4.8 illustrates how an optimization can break down when an interpolating surrogate model is applied onto a noisy surface.

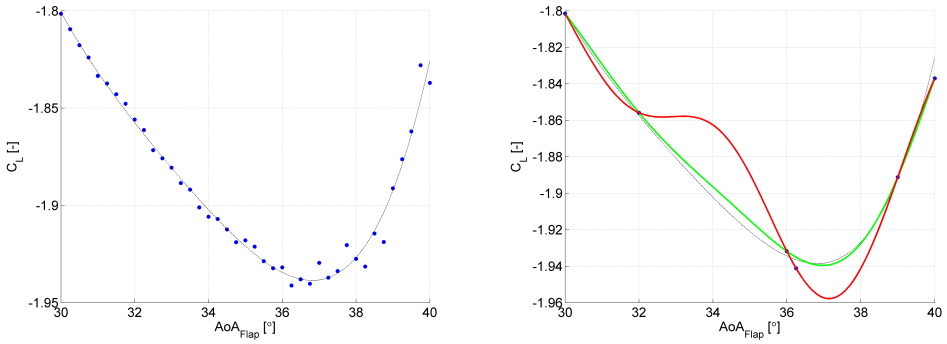
4.2.2 Regressing noisy functions

When the response surface is expected to be noisy and an optimization is pursued, it is still desirable to use Kriging rather than revert to polynomial regression, since Kriging is capable of accurately capturing highly nonlinear functions. This sub section describes how *dacefit* was modified such that it could regress data. The technique used is quite simple:

- define a positive constant ρ and add it to the leading diagonal of the correlation matrix: $R + \rho I$, where I is the identity matrix;

[3] This is the second CFD test case described in Chapter 6.

[4] When the Optimal Latin Hypercube sampling method is used, the optimization actually tries to maximize the smallest distance between the sample points.



(a) (b)

Figure 4.3 (4.3(a)) CFD data points for test case 2 (see Chapter 6). The dotted line shows the trend; 4.3(b) DACE surface fitted with sample points having a sufficient inter-distance (green line) and a DACE surface fitted with two clustered points at $36°$ and $36.25°$ (red line).

- the set of optimization variables used to find the best DACE surface now consists of the correlation parameters (the θ's) *and* this ρ.

Interestingly, the seed of the idea came from a method already used by *dacefit* to improve its robustness. Problems may occur when the correlation matrix is ill-conditioned. This can for example occur when there are closely spaced sample points. The program therefore uses a modified Cholesky factorization, where (3.20) is replaced by:

$$CC^T = R + \rho I \tag{4.8}$$

with $\rho = (10 + m)\,\epsilon_M$. The parameter ϵ_M is the machine accuracy. The process of replacing an ill-conditioned matrix with a better conditioned matrix by adding a small constant to the diagonal entries is known as regularization and it is a popular method in a range of disciplines[5]. The result of regularization is that *the response surface will not exactly go through the sample points*. Of course, since the regularization constant used by *dacefit* is so small, this will not be visible.
Realizing what regularization leads to (regression or smoothing) and knowing how it can be achieved (add a constant to the diagonal entries of the correlation matrix) the technique of using it to regress noisy functions was conceived. Contrary to the original implementation used by *dacefit*, where the regularization constant ρ is small and independent of the data values, the idea introduced here is to use it as a parameter in the optimization process along with the correlation parameters and impose larger bounds on it. Figure 4.4 compares Kriging interpolation with

[5]In general statistics, regularization is known as *ridge regression*. In inverse problems it is referred to as *Tikhonov regularization* and in optimization it is called *damped Newton*.

Kriging regression for the case illustrated in Figure 4.3, i.e. the one with the two clustered data points at 36° and 36.25°. Whereas the interpolating surface goes through the clustered points, the regressing surface clearly smoothes these points out. The lower and upper bounds imposed on the regularization constant were 1E-6 and 1E-1, respectively. Figure 4.9 illustrates the use of this regressing surface in an optimization. As opposed to the interpolating case, the optimization does not break down for the regressing case.

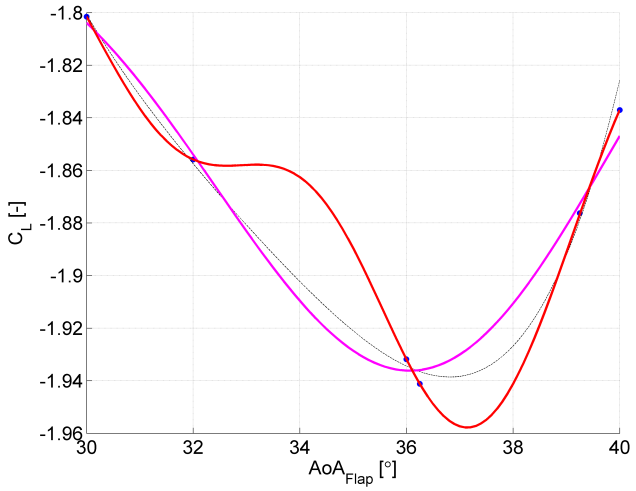


Figure 4.4 Kriging interpolation (red line) vs. Kriging regression (magenta line)

4.2.3 A Bayesian point of view to regression

In the previous section, the solution to fitting noisy functions using Kriging was introduced in a pragmatic fashion; a problem was detected, namely that interpolating a noisy function is not desired when there are closely spaced sampled points. Then, by stating that in such cases it is preferred to regress data and realizing that regularization has that effect, the solution was introduced. When one approaches Kriging from a Bayesian point of view, it is possible to provide a more convincing motivation for the technique used.
In the Bayesian approach, one states a prior belief of a process of interest, and then collects data, followed by updating that belief given this new data. Central to this approach is Bayes Theorem:

$$p(\mathbf{g}|\mathbf{y}) \propto p(\mathbf{y}|\mathbf{g})\, p(\mathbf{g}) \tag{4.9}$$

It has the following components:

- **Data distribution**, $p(\mathbf{y}|\mathbf{g})$: the distribution of the data, given the unobserved

data. If $\mathbf{y}$ represents imperfect observations of the true values $\mathbf{g}$, then it is a quantification of the measurement errors;

- **Prior distribution**, $p(\mathbf{g})$: a quantification of the a-priori belief of the distribution of the data;

- **Posterior distribution**, $p(\mathbf{g}|\mathbf{y})$: the distribution of the unobserved data given the data.

Wikle and Berliner [54] give a Bayesian derivation of Kriging. Say that one is interested in the $m+1$ discrete values $\mathbf{g}$ and that they have the following normal prior distribution:

$$\mathbf{g} \sim N(\boldsymbol{\mu}, R) \tag{4.10}$$

Then, m of these values are observed. The observations have the following distribution:

$$\mathbf{y} \sim N(H\mathbf{g}, D) \tag{4.11}$$

where H is the observation matrix and D is the observation error covariance matrix. Applying Bayes Theorem, they state that the posterior mean is given by:

$$E(\mathbf{g}|\mathbf{y}) = \boldsymbol{\mu} + K(\mathbf{y} - H\boldsymbol{\mu}) \tag{4.12}$$

where K is the gain matrix, defined as:

$$K = RH^T \left(D + HRH^T\right)^{-1} \tag{4.13}$$

It can easily be shown that (4.12) is equivalent to the DACE predictor expressed by (3.8) in the previous chapter, with the exception that the Bayesian approach defines R as a sum between the standard correlation matrix *and* the observation error matrix. If one defines the observation error matrix as ρI, the same result is obtained as regularization (see (4.8)). Now however, the regularization constant ρ has an intuitive meaning, as it represents the variance in the function values returned by the computer simulation. As discussed earlier, even though the computer simulation returns a deterministic value, there is an uncertainty in this value due to for example incomplete convergence. So, in that sense it makes sense to state that there is an observation error.

4.2.4 Adaptive regression

The Bayesian point of view is more useful than the regularization point of view in that it provides a more convincing foundation to the notion of *adaptive* regression introduced here. The discussion above mentioned adding one constant to the diagonal terms, meaning that all sample points receive the same amount of smoothing, or alternatively, the variance of the function values returned is the same for all sample points. One can take the method a step further and try to apply the correct amount of smoothing to each sample point. In other words, if a certain part of the design space is known to be considerably 'noisier' than another part, it should be possible to add more smoothing in the noisy part.
For example, recall the CFD data points shown in Figure 4.3(a). Between a flap angle of 30° and 36°, the flow on the flap suction surface is still attached. However, beyond this angle it starts to separate. So, the figure can be split into two parts; the un-stalled part and the stalled part. The data points in the first part clearly follow the trend, given by the dotted line, better than the data points in the second part. So, one would naturally be inclined to introduce more smoothing in the second part. It was already mentioned in Chapter 1 that the flow field parameters and loads extracted from the CFD results are averaged over the final set of iterations. Using these averaged values instead of the values of the last iteration should make comparisons with other configurations more reliable. Next to the average values, the standard deviations are also calculated. For this example, these values will serve as a metric for quantifying noise. In a way, they represent the noise due to incomplete convergence and possibly flow unsteadiness. However, the other source of noise, namely the one due to the discretization error, in particular due to the remeshing of geometries, should also be taken into consideration to define a good metric for the noise level of each configuration. This could be an interesting subject for future research. In the mean time, the standard deviation is taken as the metric to quantify noise. The blue line in Figure 4.5 shows the CFD data points from Figure 4.3(a), but now the standard deviation of the C_L, shown by the green line, is plotted too. It was obtained from the last 100 iterations. As expected, the standard deviation increases with flap angle. Interestingly, looking at the slope of the standard deviation, one can roughly divide it into two parts: the first part lies between 30° and 36°, i.e. in the un-stalled region; the second part lies in the stalled region between 36° and 40° and is clearly steeper than the first part. To get a feel of how adaptive smoothing would work in practice, nine sample points were taken from the CFD data in order to build a response surface. Three of these points were in the un-stalled region while the rest was in the stalled region. Figure 4.6 shows the results. Three response surfaces were built. The red response surface interpolates all the sample points and is clearly a poor representation of the CFD function. The magenta line was obtained by regressing the sample points by using an equal regression constant for all sample points. The response surface is a clear improvement. Finally, the green line was obtained by taking into account the standard deviation of each sample point. The regression constant for each diagonal

entry was calculated as follows:

$$\rho_i = \rho \cdot \sigma_i^2 \tag{4.14}$$

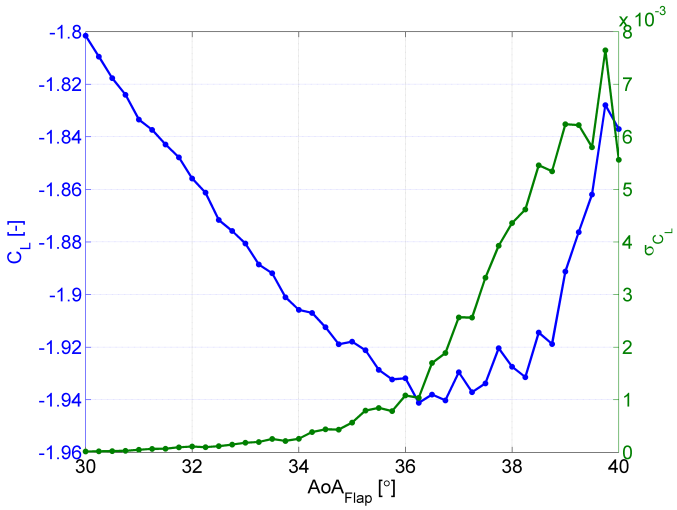


Figure 4.5 C_L as a function of the flap angle, calculated from CFD simulations (blue line), and the C_L standard deviation obtained from the last 100 iterations (green line).

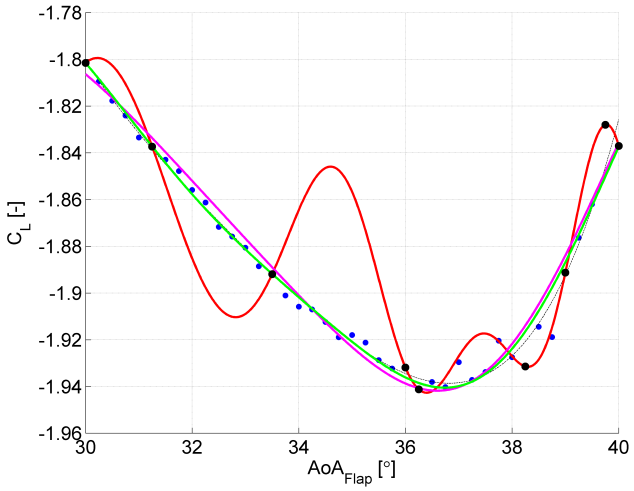


Figure 4.6 Interpolating (red line), regressing (magenta line) and 'adaptively' regressing (green line) response surfaces. The latter used the standard deviation of C_L to weigh the smoothing among the sample points, given by the black dots.

where σ_i is the standard deviation of sample point i. The two regressing lines are very similar but one can observe that the latter respects the sample data in the un-stalled region more.

The second interesting implementation of adaptive smoothing is essentially the extreme case of the first; set the regularization constant to 0 for a certain set of the sample data and set it to ρ for the rest. In other words, a part of the sample points will be interpolated while the rest will be regressed. The reason for proposing this application is that it was found that when using a regressing surface in an optimization loop, new sample points (the 'infill points') may not change the shape of the new response surface sufficiently. Of course, this is the downside of smoothing. The consequence of this behavior may be that the optimization will progress slowly. To appreciate that this can indeed occur, compare Figure 4.8(b) with 4.9(b). For clarity, they are shown side by side in Figure 4.7, with their initial surfaces. The new sample point in the interpolated case causes the new surface to change more radically than the regressed case. This enables the former to get closer to the actual minimum than the latter case. As was shown in Figure 4.9, the latter does eventually reach the actual minimum and in fact does not break down like the interpolated case. However, this example only serves to give the reader an idea of what could happen in a more complex case. The extreme case bears some resemblance with the process of determining a good finite difference step for gradient algorithms, as discussed in Chapter 2. A good step size ensures that one is outside the CFD noise. Sample points within the threshold will be regressed while the rest of the points will be interpolated. In this way, it is possible to avoid slowing down the optimization process as described above while keeping it stable as the sample points start to cluster when the process converges to an optimum.

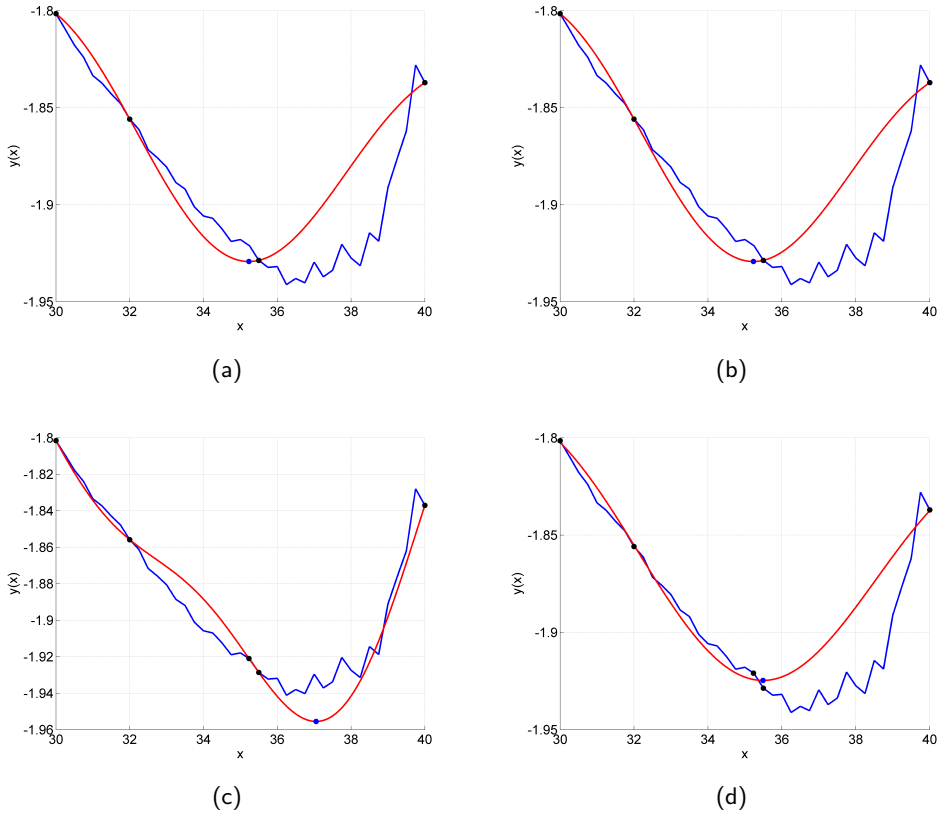


Figure 4.7 Initial interpolated (4.7(a)) and regressed (4.7(b)) response surfaces; interpolated (4.7(c)) and regressed (4.7(d)) surfaces after the first optimization iteration.

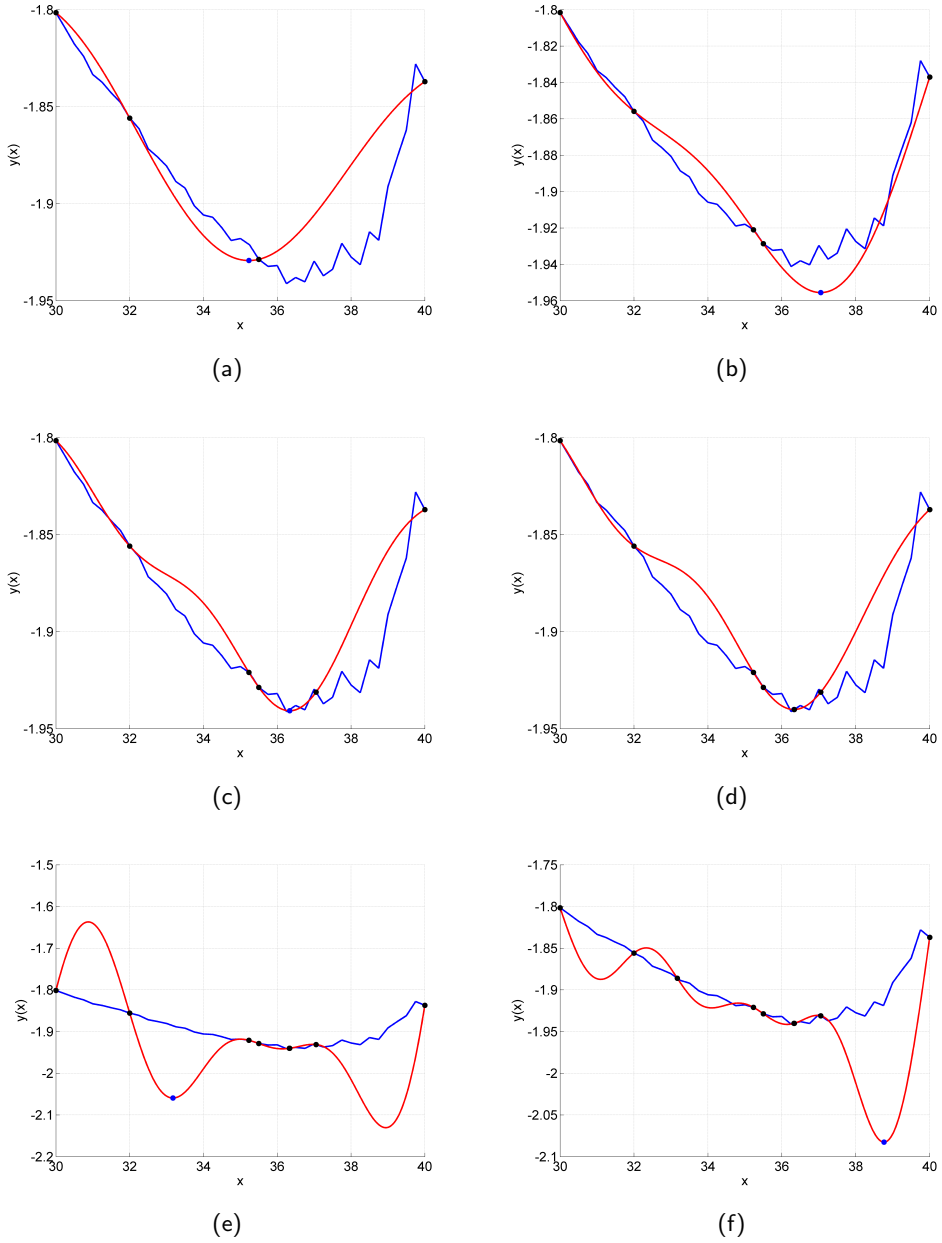


Figure 4.8 Optimization carried out with an interpolating Kriging surface (red) on a noisy function (blue). 4.8(a) is the starting point and 4.8(b) to 4.8(f) give the iterations 1 to 5, respectively. The black points are the sample points and the blue point is the minimum on the response surface, i.e. the new sample point.

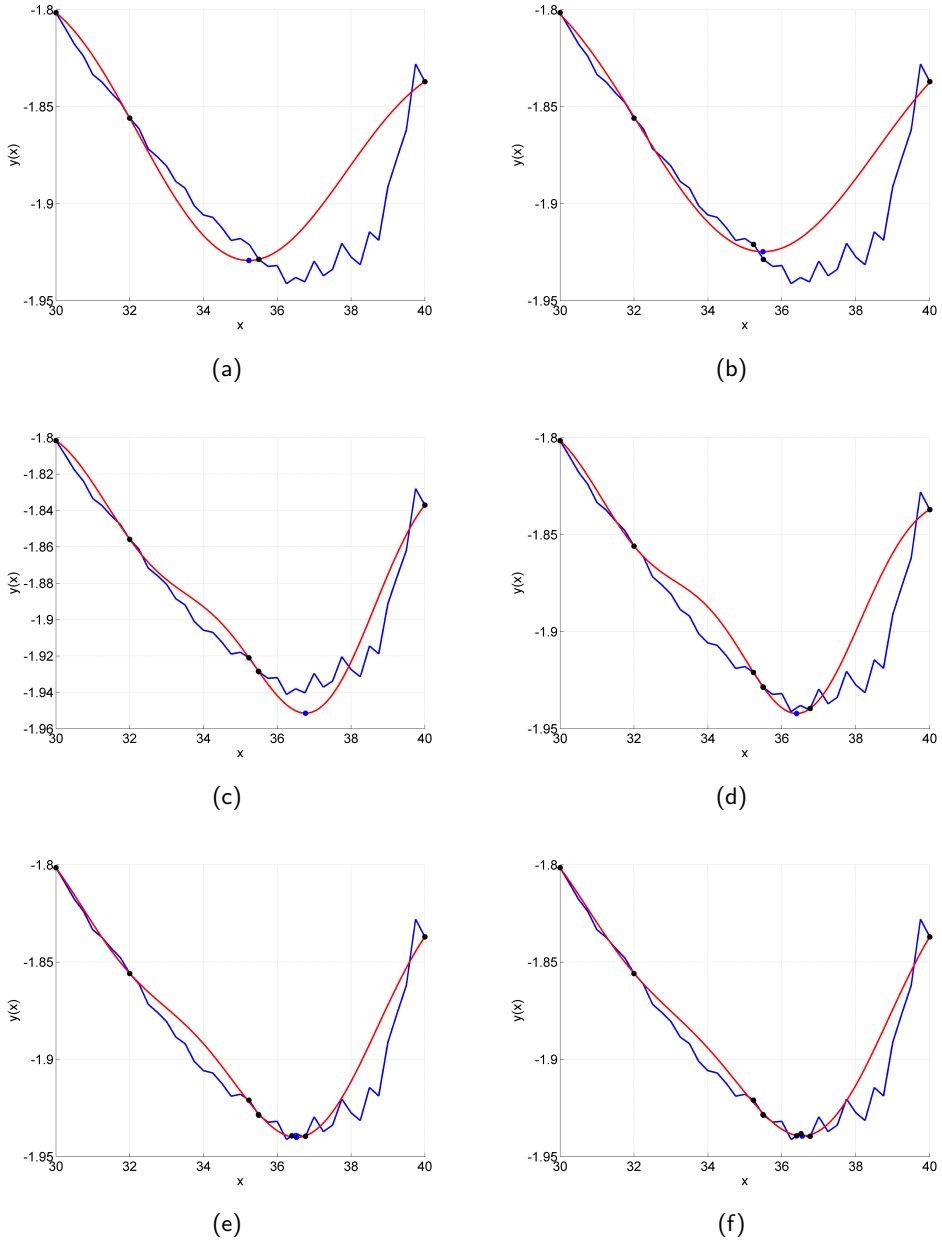


Figure 4.9 Optimization carried out with a regressing Kriging surface (red) on a noisy function (blue). 4.9(a) is the starting point and 4.9(b) to 4.9(f) give the iterations 1 to 5, respectively. The black points are the sample points and the blue point is the minimum on the response surface, i.e. the new sample point.

'The solving of a problem lies in finding the solvers.'

Van Herpen's Law

5

KERS: a Kriging based optimization tool

The work carried out in the context of the surrogate modeling framework has culminated in a computer program called *KERS*, which is short for *Kriging and Exploring Response Surfaces*. In Formula One, *KERS* is a system that recovers and stores a part of the kinetic energy that would otherwise be wasted into heat when an F1 car breaks. When activated by the driver, it can provide a *boost* in speed[1]. Since the program developed in this thesis is intended to extend (or boost) the current optimization methodology and it tries to make full use (i.e. not waste) of the DOE points, *KERS* seemed an appropriate choice of name.
KERS was written in the Matlab environment as it is widely considered to be the preferred language in the development stage of programs. This is because it comes with a large range of functions, toolboxes (like fmincon in the Optimization Toolbox) and libraries (like LAPACK and BLAS) which allows the programmer to focus on developing the actual concepts rather than having to constantly write additional codes to carry out elementary operations. An additional reason for using Matlab was because the original Kriging implementation and the global optimizer DIRECT were written in this language. However, the number of Matlab licenses available within the company is limited and the goal is that everyone within the CFD group should be able to use *KERS*. Additionally, not everyone within the group has experience with Matlab. For these reasons, the Matlab code was converted into a Windows standalone executable[2], which does not require the user to have either a Matlab license[3] or experience. To control the program, the user has to write a control file in .txt format and subsequently call the executable. This feature also makes it easy to use the program within the iSIGHT (or similar) framework(s). A detailed description of the program is given in Appendix B.
KERS can be run in two modes, namely the *build* mode and the *optimize* mode.

[1] In F1 circles, *KERS* stands for *Kinetic Energy Recovery System*. The system was introduced in the 2009 season but quickly abandoned by a number of teams. Even though it was still legal in 2010, all teams had agreed not to use it. The 2011 season however has seen the reintroduction of the system.
[2] Fortunately, it was not necessary to rewrite the whole Matlab code into for example C to generate an .exe file. Instead, a Matlab toolbox called Matlab compiler was used for this. For more information on the toolbox, visit URL: `http://www.mathworks.com/help/pdf_doc/compiler/compiler.pdf`
[3] To be able to run the executable, the end-user has to install the Matlab Compiler Runtime (MCR) which should be supplied by the developer of the Matlab code. These are a standalone set of shared libraries that enable the execution of Matlab files on computers without an installed version of Matlab.

They are described in the next two sections.

5.1 The build mode

- The surrogate modeling technique used to build the response surfaces is Kriging.
- Three different regression terms are available for building the Kriging surface; a constant, a linear function or a quadratic.
- Seven different correlation models are available for building the Kriging surface (see Table 3.1).
- The response surface can either interpolate or smooth the data. An additional option is to only smooth data points that are clustered within a user-defined threshold.
- To tune the model parameters, the user can either specify the maximum likelihood estimator as the objective function or use the cross-validation concept.
- Three different optimization algorithms can be used to tune the model parameters: a gradient-free local algorithm (Hooke-Jeeves based), a gradient-based local algorithm (Matlab's fmincon) and the gradient-free global algorithm (DIRECT).
- Warm starting can be used to provide a good initial guess for the model parameter optimization process. These have to be .mat files from previous builds.
- The response surfaces built can be validated. If the sample points were interpolated, leave-one-out cross-validation is used. If the sample points were regressed, then the customary validation technique of comparing the actual sample function values with the response surface built using the *whole* sample set is used. In both cases, a plot is made of the predicted values versus the actual sample value and in addition to this, error metrics[4] are calculated.

5.2 The optimize mode

- The type of optimization problem that can be solved is the NLP problem, defined in (2.2).

[4] The error metrics calculated are the root mean squared error, the average absolute error, the maximum absolute error and the R^2 (coefficient of determination).

- The global solution of the NLP problem is found using DIRECT, followed by fmincon.
- By supplying warm start files of previously built response surfaces, the building step can be skipped.
- The auxiliary function used to find the optimum on the response surface can either be the actual objective function or the expected improvement (EGO). In the program they are called *exploit* and *explore*, respectively. Exploit may get stuck in a local optimum if the underlying function is multimodal and/or a scarce data set was used to build the initial response surface. Explore on the other hand will eventually find the global optimum. Whichever way the response surface was built (interpolation, regression or a mix) both methods can be used to find infill points.

'A goal is not always meant to be reached, it often serves simply as something to aim at.'
Bruce Lee

6 Test applications

Using two CFD test cases, this chapter assesses the performance of a number of existing local optimizers and the newly developed surrogate modeling based optimization tool KERS. The existing local optimizers of iSIGHT were used.

6.1 Global settings

This section describes the settings used for *both* test cases.

6.1.1 iSIGHT

Version 5.2, Release 3.5-1 of iSIGHT was used. This version comes with six local optimizers (see Table 6.1). It has two gradient-free methods in DS and HJ (simplex and Hooke-Jeeves method, respectively) and four gradient-based methods. The maximum number of function evaluations these optimizers are allowed to carry out is set to $20n$, where n is the number of variables. All other optimizer settings were kept at their default values, shown in Table 6.2.

Table 6.1 iSIGHT's local optimizers

Algorithm	Search direction	Search step	Constraint handling	Reference
DS	-	-	Penalty function	[37]
HJ	-	-	Penalty function	[19]
LSGRG	BFGS / CG	line search	Elimination method	[29]
MMFD	Steepest descent	line search	Feasible direction method	[53]
MOST	BFGS	line search	SQP	[52]
NLPQL	BFGS	line search	SQP	[44]

Some comments with regards to the various settings are given:

Table 6.2 iSIGHT optimizers settings.

Algorithm	Setting	Value
DS	Intial Simplex Size	0.1
HJ	Relative Step Size	0.5
	Step Size Reduction Factor	0.5
LSGRG	Binding Constraint Epsilon	1.0E-4
	Phase 1 Objective Ratio	1.0
MMFD	*None specified*	-
MOST	Constraint Tolerance	1.0E-04
NLPQL	Forward Differencing	-

- **DS**: a value of 0.1 for the initial simplex size means that in each coordinate direction, the initial simplex will cover 10% of the variable range.
- **HJ**: the Relative Step Size is the initial step size in each coordinate direction. Like the initial simplex size parameter in DS, a larger value will result into a more global search. The other control parameter is the Step Size Reduction Factor, which enables the method to actually converge to an optimum.
- **LSGRG**: the Binding Constraint Epsilon represents the threshold for binding constraints. If a constraint is within this epsilon, it is assumed to be feasible. Increasing it can sometimes speed up convergence, while decreasing it occasionally yields a more accurate solution. The other parameter is the Phase 1 Objective Ratio. This parameter sets the ratio of the true objective value to the sum of constraint violations to be used as the objective function during the so-called Phase 1 of optimization. In this phase, the algorithm tries to make an infeasible point feasible before it can proceed.
- **MMFD**: this algorithm does not have any specific settings.
- **MOST**: the Constraint Tolerance specifies the acceptable constraint violation for feasible designs.
- **NLPQL**: this is the only iSIGHT algorithm that provides central differencing in addition to forward differencing. Since the other gradient algorithms only use forward differencing, the same was set for NLPQL.

6.1.2 KERS

The following settings were used for KERS:

- **Kriging model**: for all the Kriging surfaces built, a constant term was used for the regression term μ and the Gaussian function was used for the correlation model, as these are the most common settings in the context of engineering functions ([43], [48], [12]).
- **Tuning criterion**: it was found in Chapter 4 that the cross-validation route can give a more accurate response surface than the MLE route when the data set is scarce, but when the data set is larger, both routes will return similarly accurate surfaces. Also, it was found that the cross-validation route is computationally more demanding, especially when the number of sample points is large. More sample points can be used to build response surface for geometric functions as they are considerably cheaper to evaluate than the CFD functions. Therefore, it was decided to build the expensive CFD functions using cross-validation while the geometric functions were built using the MLE criterion.
- **Sampling strategy**: iSIGHT's Optimal Latin Hypercube sampling technique was used. The algorithm is based on the work by [22].
- **Number of sample points**: it was decided to follow the '$10n$-rule', where n is the number of variables. In the current LRGP methodology, $20n$ points are used for DOEs. Since the goal is to use the surrogate model for optimization, it makes sense to use fewer points for the DOE to allow for the optimization iterations. For the two dimensional cases, 10 initial points were used instead of 20 as it was found that in these cases, accurate surfaces could already be built with as little as 10 points.
- **Smoothing threshold**: for the adaptively smoothing Kriging surface, the threshold in each variable direction is taken to be equal to the gradient step sizes. So, sample points within this threshold will be regressed while the other points will be interpolated.

6.2 Test case 1 - the double airfoil

6.2.1 Test case description

The main goal of the first test case is to gain an *understanding* of how the optimization algorithms work. To visualize the paths that these algorithms take towards the optimum, the geometry was parameterized with two variables. The configuration of test case 1 consists of two identical airfoils, as shown by Figure 6.1. The center airfoil is fixed while the outer airfoil can move along the dotted circle. On top of this, its angle of attack can vary from -10° to $+10^\circ$. The free stream flow comes from the left at 40m/s. The mesh for test case 1 consisted of approximately 27,000 polyhedral faces.

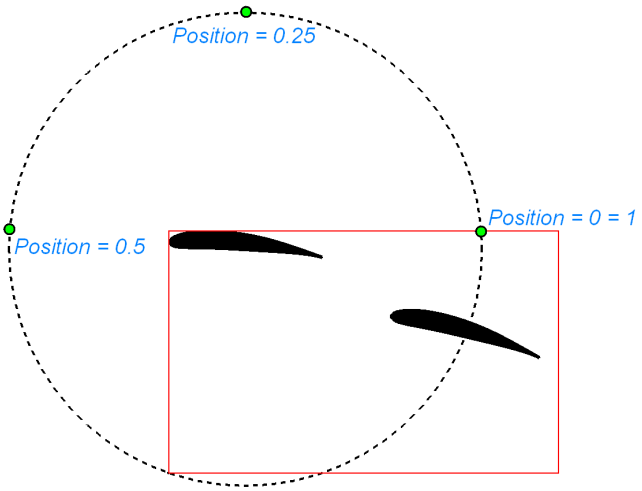


Figure 6.1 Test case 1 - geometry

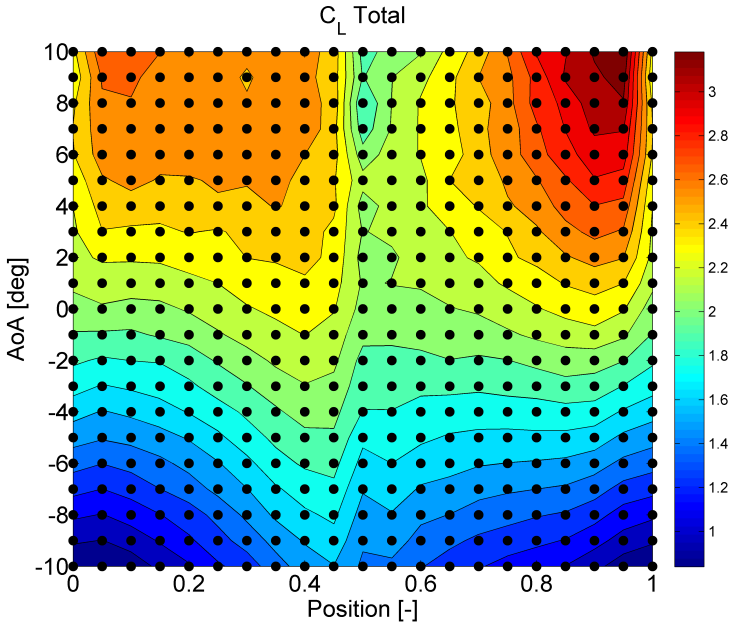


Figure 6.2 Test case 1 - full factorial CFD results.

To locate the global optimum and to determine how nonlinear the problem is, a full-factorial DOE was carried out. Both variables were divided into 21 points, resulting in a total of 441 design points. Figure 6.2 shows the results. The surface is highly nonlinear, which should pose a challenge for the iSIGHT optimizers *and* for KERS. To understand to what extent the CFD noise can compromise the performance of an optimizer, it would be interesting to have a smooth reference function to the CFD results. To this end, the full factorial results for test case 1 were interpolated. To

allow continuous gradients throughout the domain, bi-cubic interpolation was used instead of linear interpolation. Figure 6.3 shows the smooth interpolated design space.
Test case 1 is subdivided into the following optimization problems:

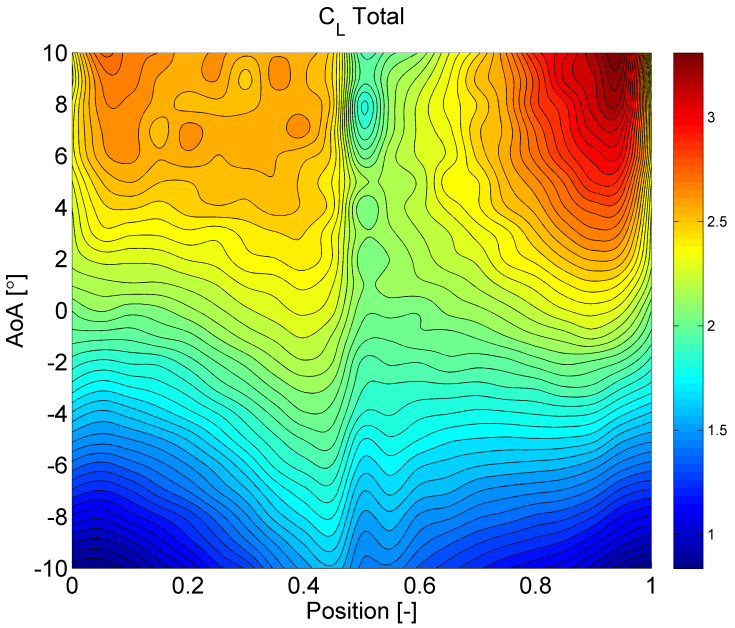


Figure 6.3 Test case 1 - bicubic interpolation of the full factorial CFD results.

1. maximize the lift produced by the assembly. This case only imposes bounds on the variables; the outer airfoil can freely move along the entire circle and its angle of attack can take any value between $-10°$ and $+10°$;

2. maximize the lift produced by the assembly while ensuring that the outer airfoil stays inside the red rectangle shown in Figure 6.1;

3. maximize the lift produced by the assembly while ensuring that the pitching moment coefficient around the leading edge of the center airfoil is between -0.5 and 1.5;

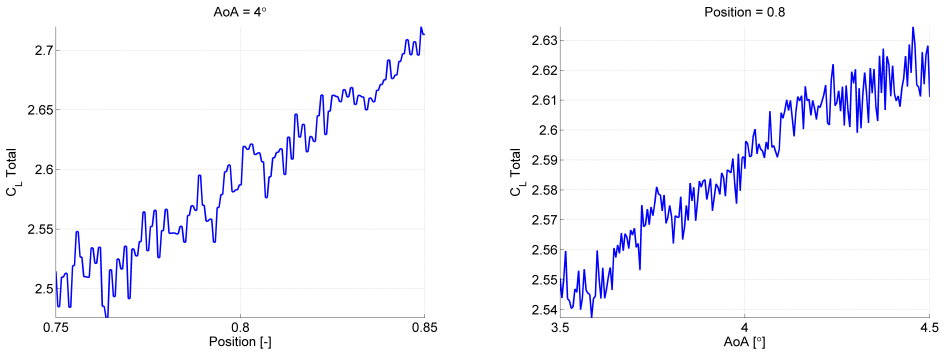
4. maximize the lift produced by the assembly while ensuring that the pitching moment coefficient around the leading edge of the center airfoil is 0.

By defining the four sub cases, it can be investigated how the optimization algorithms will cope in the presence of non-linear equality and inequality constraints. They are described in detail in Sections 6.2.3 to 6.2.6, together with the results obtained. For a summary of the results, the reader is referred to Section 6.2.7.

6.2.2 The optimum gradient step size

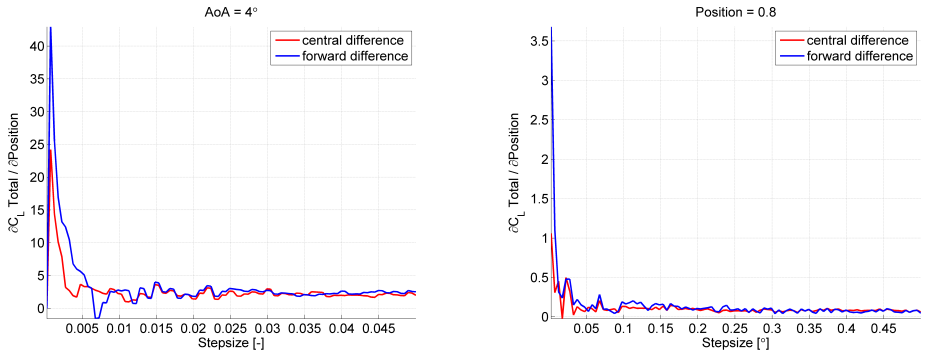
For the gradient-based algorithms, the gradient step sizes have to be determined. To determine the optimum step size for test case 1, a parameter sweep was carried out at $[Position, AoA] = [0.8, 4°]$. Figure 6.4 shows the results. The presence of the noise is evident. Taking the gradient step size as small as $\sqrt{u}$ would render the calculated gradients useless. Figure 6.5 shows the finite difference gradients as a function of the step size. Based on the figures, the following step sizes were chosen:

- $0.25°$ for AoA;
- 0.04 for $Position$.



(a) (b)

Figure 6.4 Parameter sweeps for test case 1.



(a) (b)

Figure 6.5 Finite difference gradients as a function of the step size for test case 1.

6.2.3 Box constraints

The first sub problem is the simplest of the four, as it only has box constraints. This case was started at three different points;

- Starting Point 1 (Sp1); [Position, AoA] = [0.5, 0]
- Starting Point 2 (Sp2); [Position, AoA] = [0, -10]
- Starting Point 3 (Sp3); [Position, AoA] = [0.8, 6]

Figure 6.6 shows the starting points, together with the global optimum, indicated by the red cross. It lies at [Position, AoA] $\approx$ [0.938,10.0] and the C_L there is 3.360.

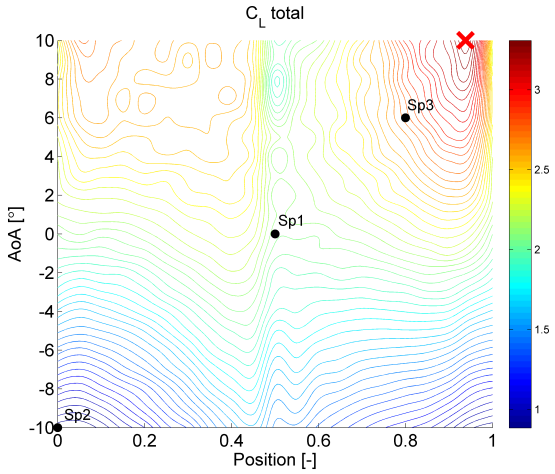


Figure 6.6 Global optimum and starting points for the box constrained sub problem of test case 1.

Interpolated surface

For the interpolated cases, the Kriging surfaces would interpolate the sample points. The center plot of Figure 6.7 shows the initial response surface produced by KERS, together with the true function it is approximating at the left. Although not perfect, the response surface does show the global trend of the function. The infill points were found using Method 2 from Figure 3.2, i.e. the global optimum on the response surface is chosen to be the next sample point. Table 6.3 shows the results. KERS finds the solution in 8 iterations, since the initial response surface was built with 10 sample points. Three different starting points were used for the iSIGHT algorithms. The table shows the results obtained when they were started closest to the global optimum (Sp3). All of the iSIGHT optimizers found the global maximum in this case.

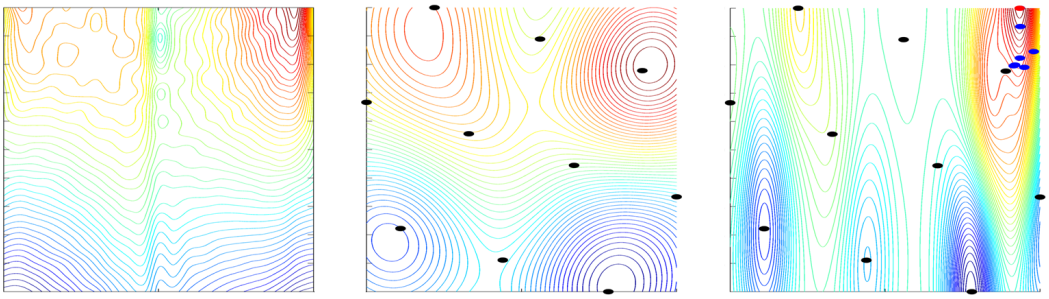


Figure 6.7 Contour curves of the total lift coefficient C_L for test case 1; (left) actual surface, (center) initial Kriging surface built with the 10 black sample points, (right) Kriging surface after 8 iterations; the blue points are the infill points and the red point is the global optimum found.

MMFD did not find the global maximum in the specified 40 function evaluations (using the $20n$ rule). However, by increasing the number of function evaluations it did eventually find it after 124 function evaluations. The reason it was so slow was because it started to zigzag its way to the optimum. As mentioned in Chapter 2, steepest descent methods are prone to this behavior if the optimum is ellipse shaped, which it indeed is when examining the actual function in Figure 6.7. The best iSIGHT optimizer for this particular case was MOST, which required 24 function evaluations, i.e. 6 more than KERS. The right-hand picture of Figure 6.7 shows the state of the response surface after the 8 iterations. The accuracy of the response surface in the vicinity of the global maximum has increased, but at the expense of the rest of the surface. One may argue however that when one is trying to use the response surface for the purpose of optimization, the response surface is merely a 'vehicle' for reaching that goal. As long as the final surface is not used for visualization purposes or gaining understanding, the deterioration in overall surface quality may not be considered a problem.

Table 6.3 Results obtained for the box constrained case of test case 1 on the interpolated CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.938	10.0	3.360	36
HJ	0.938	10.0	3.360	27
NLPQL	0.938	10.0	3.360	34
MOST	0.937	10.0	3.360	24
MMFD	0.939	8.58	3.262	40
LSGRG	0.938	10.0	3.360	26
KERS	0.938	10.0	3.360	18

CFD surface

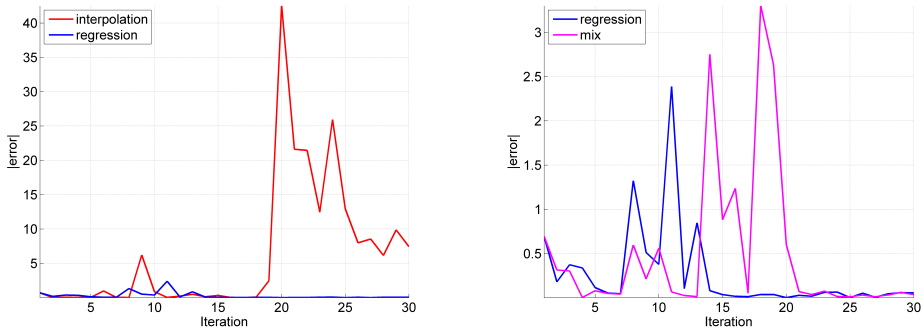
On the actual (noisy) CFD surface, a smoothing (regr) and adaptively smoothing (mix) response surface was used, in addition to the interpolating response surface. Table 6.4 shows the results. Comparing the three response surface methods with each other, the fully regressing surface finds the highest C_L, closely followed by the mixed surface. Note that the number of runs is the sum of 10 (used to build the initial response surface) with the iteration in which the best C_L value is found. Figure 6.8(a) compares the prediction error of the interpolating and regressing surfaces. The interpolating surface clearly breaks down after 20 iterations. Indeed, the optimum value was found by the interpolating method after 27 runs, which is the 17^{th} iteration (i.e. before the surface breaks down). The regressing surface is clearly more stable and looking at Figure 6.8(b), one can observe that the error does indeed converge to 0 (after some glitches), contrary to the interpolating method. The mixed surface is also more stable than the interpolating surface and also sees its prediction errors converge to zero. Comparing KERS with the iSIGHT optimizers, it seems that KERS does not find the best value. However, it does reach the same region where the global optimum lies. In the response surface method's defense, it should be stated that the results shown for the iSIGHT optimizers are the results obtained when they were started closest to the known global optimum. For the response surface method however, no such prior knowledge was required.

Table 6.4 Results obtained for the box constrained case of test case 1 on the actual CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.937	9.36	3.325	36
HJ	0.939	10.0	3.338	40
NLPQL	0.932	10.0	3.326	12
MOST	0.906	10.0	3.314	32
MMFD	0.926	9.60	3.280	35
LSGRG	0.932	10.0	3.326	32
KERS (interp)	0.932	9.46	3.287	27
KERS (regr)	0.908	10.0	3.323	33
KERS (mix)	0.937	10.0	3.318	32

To get an idea of how the interpolating surface deteriorates due to the noisy CFD function, Figure 6.9 compares the surface at the 20^{th} iteration with the regressing surface.

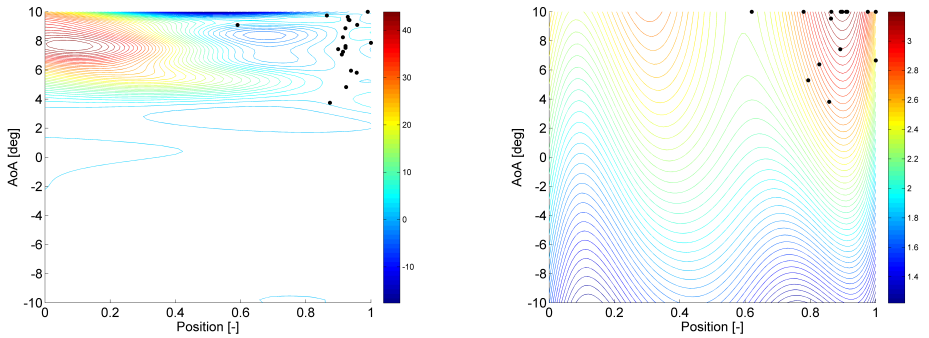
The gradient-free iSIGHT optimizers are also influenced by the presence of the noise. Figure 6.10 uses DS to illustrate this. The black dots show the sampled points on the interpolated surface and the blue stars show the points on the actual CFD surface. Two observations are made. First of all, the blue points cluster (converge) *close* to the actual optimum which explains why the optimum value found on the CFD surface is lower than the one found on the interpolated surface. Secondly, the *initial*



(a) (b)

Figure 6.8 Absolute error of the C_L prediction as a function of the iteration number for the box constrained case of test case 1, applied on the actual CFD surface.

sampled points are the same for both cases when the simplex is still large and the noise has no effect on the process. However, as the simplex becomes smaller the noise kicks in and the routes start to deviate from each other.



(a) (b)

Figure 6.9 Interpolated (6.9(a)) and regressed (6.9(b)) response surfaces after 20 iterations, for the box constrained case of test case 1, applied on the actual CFD surface. The black dots are the infill points.

6.2.4 Geometric constraints

Figure 6.11 shows the constraint boundaries of the geometrically constrained sub case, together with the three starting points and the global optimum. The gray area is the infeasible space. The constraints significantly decrease the size of the feasible space such that only *Positions* between 0.8 and 1 are feasible. The starting points were:

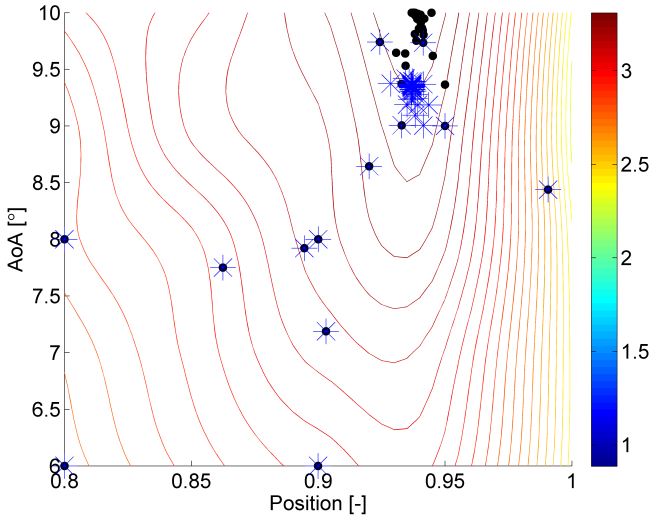


Figure 6.10 Points sampled by DS on the interpolated surface (black dots) and the actual CFD surface (blue stars).

- Starting Point 1 (Sp1); [Position, AoA] = [0.5, 0]
- Starting Point 4 (Sp4); [Position, AoA] = [0.8, -9]
- Starting Point 5 (Sp5); [Position, AoA] = [0.9, -9]

From Figure 6.11 it can be seen that Sp4 is an infeasible point, but close to the bottom left constraint boundary. Sp5 is on the feasible side of this boundary. The global optimum lies at [Position, AoA] $\approx$ [0.938,10.0], where C_L is 3.360.

Interpolated surface

To deal with the geometric constraints, response surfaces were built for these functions too. Since the evaluation of these functions would be considerably cheaper than the CFD functions, it is possible to use more sample points when building them. Therefore, 80 sample points were used instead of 10. Note that for the geometric constraints, interpolating surfaces were used. This is because these values, obtained from the CAD package CATIA, do not contain the noise that the CFD functions exhibit. The resulting response surfaces, corresponding to the four sides of the legality boxes, can be found in Figure 6.12. Notice the periodicity in the *Position* variable, due to the fact that the outer airfoil moves along a circle. For this sub case, the global maximum was the same as the one for sub case 1. Table 6.5 shows the results. For this particular case, MOST, NLPQL and KERS used the same number of function evaluations to find the global maximum.

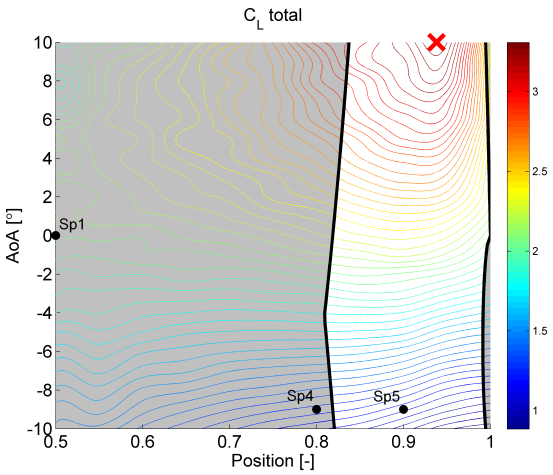


Figure 6.11 Global optimum and starting points for the geometrically constrained sub problem of test case 1.

Table 6.5 Results obtained for test case 1, sub case 2 on the interpolated CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.938	10.0	3.360	38
HJ	0.938	10.0	3.360	27
NLPQL	0.938	10.0	3.360	18
MOST	0.938	10.0	3.360	18
MMFD	0.938	10.0	3.360	26
LSGRG	0.938	10.0	3.360	26
KERS	0.938	10.0	3.360	18

CFD surface

Table 6.6 shows the results for the geometrically constrained case. Again, the regressing surface outperforms the interpolating surface. Figure 6.13 compares the actual C_L values for the three response surface methods as the optimization progressed. Starting with the interpolating method, one can observe that it behaves well *until* it reaches the 9^{th} iteration. Before that, it steadily finds better C_L values compared to the first guess. After this, the method is clearly struggling to find improved designs. The regressing surface on the other hand is clearly more stable. After a *slow* start, it does eventually converge to an optimum, without breaking down as the sample points start to cluster. The reason why a regressing method could be slower than an interpolating method is because it smoothes the data, so added points may not change the response surface as much as an interpolating method would. Depending on the total number of infill points chosen, this implementation could have returned a design less optimal than the interpolating implementation.

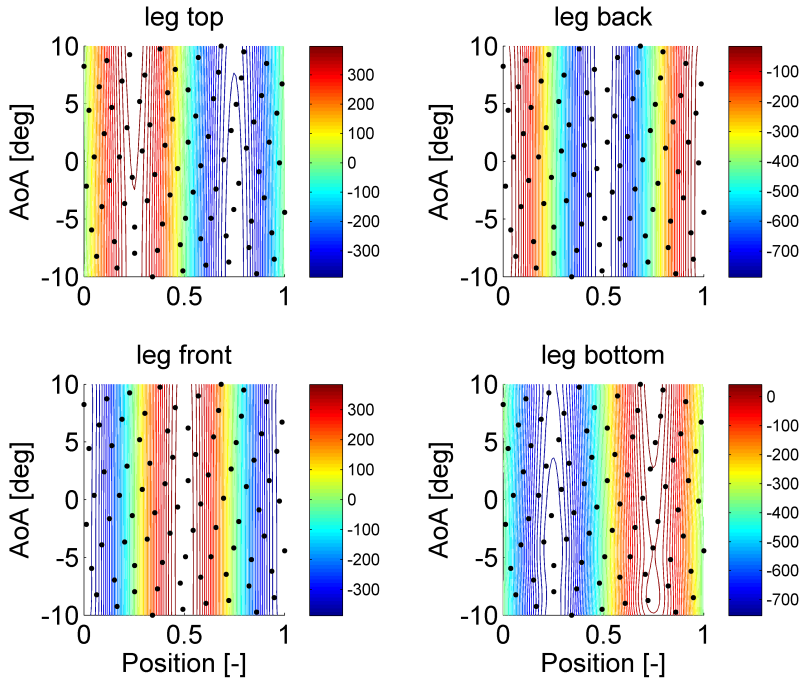


Figure 6.12 Response surfaces built for the geometric constraints of test case 1 using the 80 sample points shown in black.

Table 6.6 Results obtained for test case 1, sub case 2 on the actual CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.927	9.91	3.326	35
HJ	0.938	10.0	3.309	27
NLPQL	0.932	9.87	3.350	40
MOST	0.908	10.0	3.308	35
MMFD	0.905	9.76	3.271	24
LSGRG	0.891	8.00	3.126	30
KERS (interp)	0.894	10.0	3.296	18
KERS (regr)	0.917	10.0	3.318	34
KERS (mix)	0.895	9.64	3.287	34

This would have been the case if the method was stopped after 9 iterations. That is the reason why the mixed method of surface generation was introduced. In the first couple of iterations, it simply chooses the same infill points as the interpolating method. However, where the interpolating method starts to break down due to sampling closely spaced points, the mixed method smoothes these points and as a result can progress without breaking down. It is this robustness, combined with the fact that it will not be slowed down as much as the regressing method that could

make the mixed method an interesting option for Kriging based optimization.

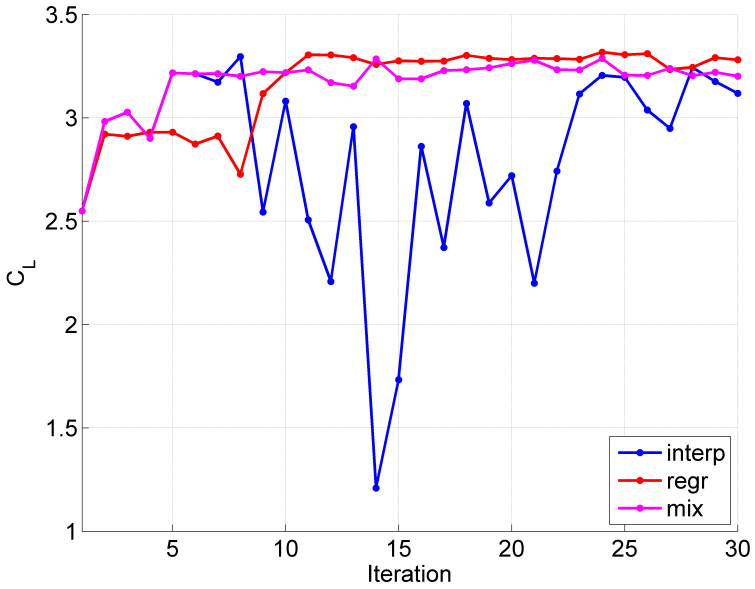


Figure 6.13 Actual C_L vs. iterations for the geometrically constrained case of test case 1, applied on the actual CFD function.

6.2.5 Pitching moment bounds

Figure 6.14 shows the inequality constraints. The dotted curve in the center is the pitching moment coefficient contour line at -0.5. The two dotted curves at the left and right sides are the pitching moment coefficient contour lines at 1.5. The global maximum is $C_L \approx 2.837$ and lies at [Position, AoA] $\approx$ [0.779, 10.0], that is on the upper constraint boundary. This sub case was started at two different points:

1. Starting Point 1 (Sp1); [Position, AoA] = [0.5, 0]
2. Starting Point 6 (Sp6); [Position, AoA] = [0.2, -10]

Sp6 is a feasible point but to get to the maximum lift, the optimizers will have to face the center constraint.

Interpolated surface

Figure 6.15 shows the actual lift and pitching moment coefficients on the left, together with their initial (interpolated) surrogates on the right. Table 6.7 shows the

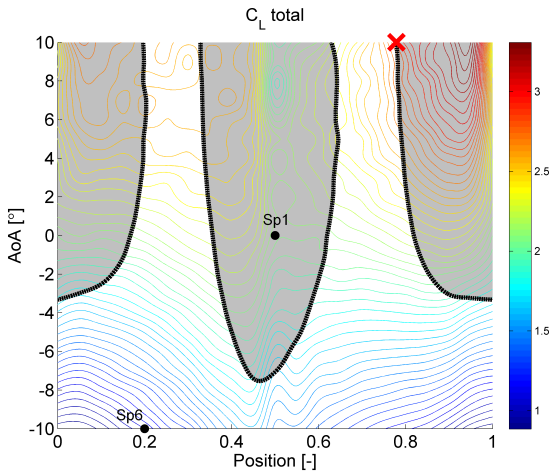


Figure 6.14 Global optimum and starting points for the sub problem of test case 1 with the pitching moment bounds.

results of the optimizers. For this particular case, NLPQL was the fastest iSIGHT algorithm as it found the solution in 24 iterations. Recall that when constraints are present, SQP methods allow intermediate iterates to be infeasible. This enables them to find 'short-cuts' as they progress towards the feasible optimum. MMFD and LSGRG however are feasible methods, meaning that they try to keep all iterates feasible. This was confirmed by inspecting the optimization routes taken by the solvers. Figure 6.16 shows the optimizer routes for Sp6, which was a feasible point. Whereas the global maximum lies on the constraint boundary at $[Position, AoA] \approx [0.8, 10]$, there are some local maxima in the region $0.2 < Position < 0.3$ and $6 < AoA < 10$. MOST was the only optimizer that converged to the global maximum. The rest converged to local maxima. As discussed above, the SQP method MOST allows its iterates to enter the infeasible domain. In this way, it was able to simply cross the 'obstacle' in the center formed by the lower bound of the constraint in order to reach the optimum. One if its iterates is in the infeasible region in the center of the domain. MOST still accepts this step when it carries out its line search. Recall that it uses a merit function for this, which is the sum of the objective function and the weighted constraint violations. Even though the iterate is infeasible, it is still accepted because it produces a very good objective function value. MMFD on the other hand ricochets between the constraint boundaries (recall the push-off factor mentioned in Chapter 2). This happens around $[Position, AoA] = [0.4, -5]$ and $[0.2, 9]$. LSGRG shows a poorer behavior. It closely follows the constraint boundary as it converges to the same optimum that NLPQL reaches. Again, this is a result of the fact that it is a feasible method. It wants to cross the boundary since the function value is high there but it cannot do that as it will produce an infeasible iterate. The result is therefore that it closely follows the constraint boundary as it converges to its optimum. Note that DS and HJ, which make use of the penalty

method, can produce infeasible iterates too.

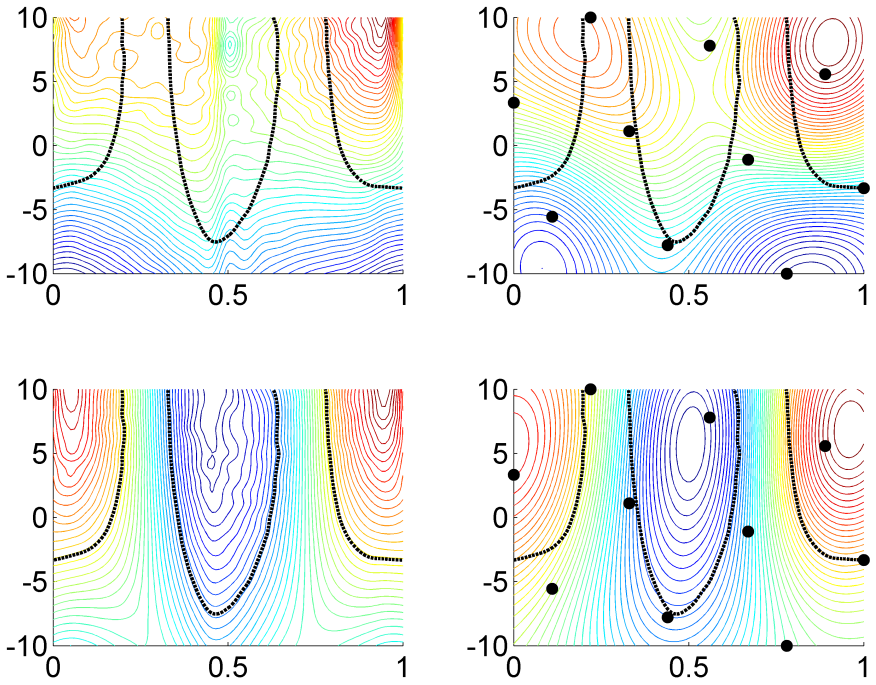


Figure 6.15 Contour curves of the total lift (top) and pitching moment (bottom) coefficients for test case 1; (left) actual surfaces, (right) Kriging surfaces built with the 10 black sample points. The black lines represent the C_M curves at -0.5 and 1.5.

Table 6.7 Results obtained for test case 1, sub case 3 on the interpolated CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.781	9.06	2.749	18
HJ	0.777	10.0	2.827	38
NLPQL	0.779	10.0	2.837	24
MOST	0.778	10.0	2.834	36
MMFD	0.779	10.0	2.837	29
LSGRG	0.779	10.0	2.837	30
KERS	0.779	10.0	2.837	15

CFD surface

Table 6.8 shows the results. All the response surface methods locate the global optimum. The mixed method finds the highest C_L value.

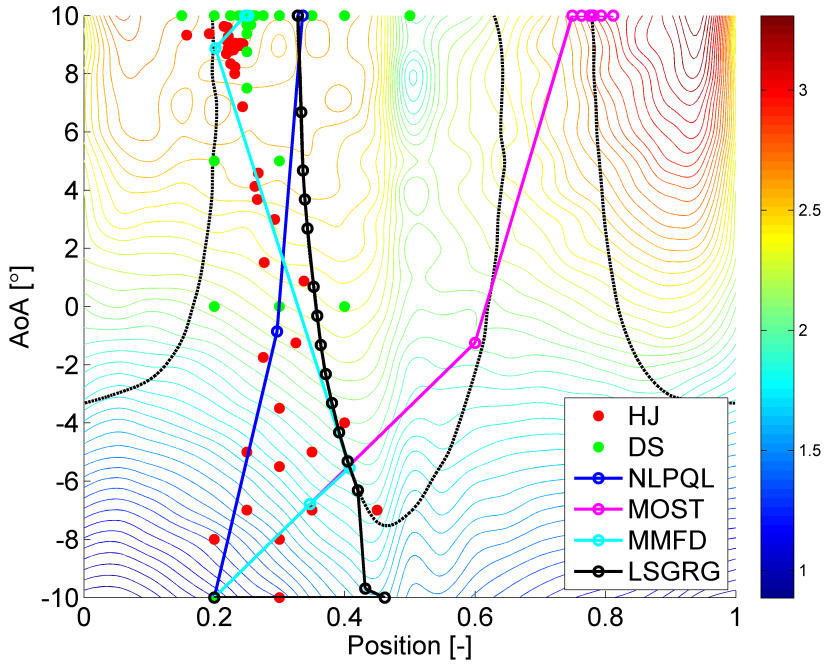


Figure 6.16 Routes followed by the iSIGHT optimizers for sub case 3 of test case 1.

Table 6.8 Results obtained pitching moment bounded problem of test case 1 on the actual CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	0.778	9.07	2.766	32
HJ	0.781	8.75	2.775	28
NLPQL	0.782	8.43	2.784	31
MOST	0.781	9.16	2.784	31
MMFD	0.781	9.20	2.749	34
LSGRG	0.784	5.88	2.714	19
KERs (interp)	0.779	10.0	2.784	20
KERS (regr)	0.782	10.0	2.783	32
KERS (mix)	0.784	10.0	2.787	25

6.2.6 Pitching moment target

The final sub case imposed a C_M target of 0 as the equality constraint. Figure 6.17 shows the constraint curves, the starting point and the global optimum ([Position, AoA] = [0.300, 6.30]), where $C_L = 2.591$. The starting point was in the center of the domain:

- Starting Point 1 (Sp1); [Position, AoA] = [0.5, 0]

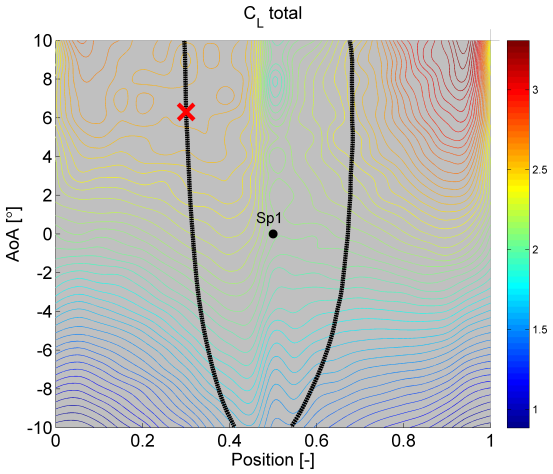


Figure 6.17 Global optimum and starting point for the sub problem of test case 1 with the pitching moment target.

Interpolated surface

Table 6.9 shows the results. For this particular case, none of the iSIGHT optimizers were able to find the global maximum. DS, HJ, MOST and MMFD were unable to follow the equality constraint. Among the successful iSIGHT optimizers, NLPQL found a local optimum, which lies on the right hand constraint boundary. KERS was able to find the global maximum.

Table 6.9 Results obtained for the test case 1, constrained with the pitching moment target, on the interpolated CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	×	×	×	×
HJ	×	×	×	×
NLPQL	0.682	7.04	2.456	30
MOST	×	×	×	×
MMFD	×	×	×	×
LSGRG	0.589	-8.00	1.542	39
KERS	0.300	6.30	2.591	17

CFD surface

Table 6.10 shows the results for this sub case when applied on the actual CFD function. The three implementations of KERS all converge to the left hand curve

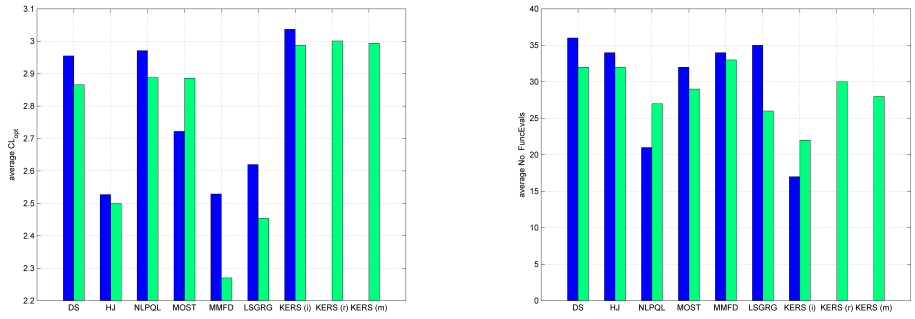
(although they do not all converge to the same point) where the *global* optimum lies. Among the iSIGHT optimizers, only the SQP methods NLPQL and MOST managed to find a point that satisfied the equality constraint, although they converged to the right hand curve, which contain *local* optima.

Table 6.10 Results obtained for test case 1, sub case 4 on the actual CFD surface.

Algorithm	Position	AoA	C_L	Runs
DS	×	×	×	×
HJ	×	×	×	×
NLPQL	0.681	3.33	2.349	23
MOST	0.683	6.58	2.459	22
MMFD	×	×	×	×
LSGRG	×	×	×	×
KERS (interp)	0.302	5.36	2.583	24
KERS (regr)	0.297	9.01	2.581	20
KERS (mix)	0.298	7.92	2.585	20

6.2.7 Summary

The goal of this test case was to gain an understanding of how the optimizers work. This sub section summarizes the results obtained. Figure 6.18(a) gives the average optimum C_L found by the different optimizers and Figure 6.18(b) shows how many function evaluations were needed to reach the optimum. The figures contain quite a bit of information so a break down in terms of the following aspects is made:



(a) (b)

Figure 6.18 Optimizer results for test case 1.

- **best iSIGHT optimizer**: Among the iSIGHT optimizers, it seems like NLPQL performs the best. It finds the best optima in the least amount of function evaluations.

- **comparison between iSIGHT and KERS**: KERS finds better optima in the least amount of function evaluations. In all cases, it was able to find the *global* optimum, whereas the optima found by the iSIGHT optimizers depended on the starting point chosen (and they therefore rely on the prior knowledge of the user regarding the design problem).

- **influence of the CFD noise**: when comparing the optima found by the optimizers on the interpolated objective function and the actual (noisy) CFD function, one can observe that lower function values are found when the objective contains noise[1]. KERS (in its interpolating mode) shows the same behavior. For this reason, it was investigated how a(n) (adaptively) regressing response surface would cope. One can see that it *does* provide a slight improvement. From the figure it is not clearly visible, but the averages were 2.988, 3.001 and 2.994 for the interpolating, regressing and adaptively regressing surfaces, respectively. In terms of the number of function evaluations required, the iSIGHT optimizers tend to stop earlier, explaining why the average number of function evaluations is lower. KERS on the other hand requires more function evaluations. The interpolating mode shows this behavior because it 'breaks' down at a certain point. The regressing and adaptively regressing surfaces show this behavior because when they smooth the data they tend to converge at a slower rate.

6.3 Test case 2 - the two-element airfoil

6.3.1 Test case description

In reality, it is common to use more than two variables in an optimization process. Test case 2 is used to represent problems with a greater number of design variables. It is a two-element airfoil as shown in Figure 6.19. The mesh for test case 2 consisted of approximately 22,000 polyhedral faces.

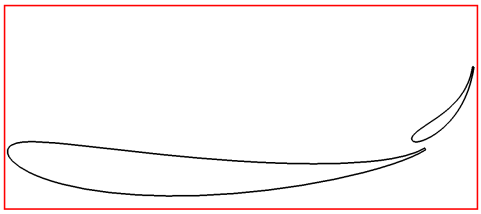


Figure 6.19 Test case 2 - geometry

[1] MOST does not show this behavior. On closer inspection, it was found that for sub case 4 (the equality constraint case) the algorithm stalled when applied on the interpolated surface but not when applied on the CFD surface. So, this has skewed the results.

The flap and main plane both have six variables, so there are twelve variables in total:

- horizontal position (X)
- vertical position (Y)
- angle of attack (α)
- airfoil thickness (t)
- airfoil camber (γ)
- chord length (c)

As in test case 1, test case 2 is divided into a set of sub cases. In total there are six cases. For all cases the downforce has to be maximized (minimization of the lift coefficient C_L) while ensuring that the airfoils stay inside the red rectangular box. The sub cases differ in the number of variables. Table 6.11 shows the breakdown.

Table 6.11 Sub cases for test case 2.

	Flap						Main plane					
sub case	**X**	**Y**	**α**	**t**	**γ**	**c**	**X**	**Y**	**α**	**t**	**γ**	**c**
1		×	×									
2		×	×		×						×	
3		×	×	×	×					×	×	
4		×	×	×	×			×	×	×	×	
5		×	×	×	×		×	×	×	×	×	×
6	×	×	×	×	×	×	×	×	×	×	×	×

Table 6.12 gives the bounds imposed on the variables, together with the *feasible* starting point. At this point, the assembly generated a C_L of **-1.239**.

Table 6.12 Starting point and bounds imposed on the twelve variables of test case 2.

	Main plane			Flap			
Variable	**Lower**	**Start**	**Upper**	**Lower**	**Start**	**Upper**	**Unit**
X	50	**151**	300	200	**315**	350	mm
Y	0	**35**	100	50	**60**	150	mm
α	-5	**0**	20	0	**15**	50	◦
t	1	**30**	70	1	**7**	30	mm
γ	0.1	**0.35**	1.5	0.1	**0.2**	1.0	-
c	100	**300**	350	50	**70**	200	mm

The bounds were chosen such that there will be configurations where the two airfoils clash. This can indeed occur in practice for certain parameterizations. When a geometry clashes, then an error will occur in one of the simulation codes. For example, Star-CCM+ will not be able to generate a volume mesh for such a case. iSIGHT will interpret this as a failed run so it would be informative to see how the optimization algorithms deal with these occurrences. If the geometry is parameterized in such a way that no clashing can occur, then it is still possible for a run to fail. For example, a failed run can be caused by an error in the cluster or a floating point error in the solver. Since it was possible for the airfoils to clash, an additional inequality constraint was defined; the minimum distance between the elements was not allowed to be smaller than 2mm.
To start KERS, a DOE was carried out for each sub case. The best C_L value found for each sub case can be found in Table 6.13. The table also shows the initial number of sample points used to build the response surfaces of the geometric constraints and the objective function. To build the constraint surfaces, more sample points were used since they are considerably cheaper to evaluate. Note that for C_L, the number of sample points is not exactly $10n$. This is because some DOE points produced clashed geometries, where the C_L function could therefore not be evaluated.

Table 6.13 Test case 2; KERS starting points.

	Samples		
Dimensions	**geo const.**	$\mathbf{C_L}$	**bsf**
2	80	19	-1.767
4	200	38	-2.269
6	300	55	-1.955
8	500	70	-2.219
10	750	89	-1.793
12	1000	188	-1.372

6.3.2 The optimum gradient step size

A parameter sweep around the starting point was carried out to determine the optimal gradient step size (see Figures 6.20 and 6.21). The following step sizes were chosen:

- for the angles: 0.25°;
- for the cambers; 0.05;
- for all the parameters with units length (in mm); 1mm.

The choices were not solely based on the parameter sweep results. iSIGHT does not allow the user to explicitly define different gradient steps for each variable. Instead,

there is one gradient step parameter that can be set and this value is then used for all variables. Next to this setting, it is possible to scale each variable differently. This setting is intended to ensure that all variables are in the same order of magnitude. However, since it is not possible to choose both settings (scaling and gradient step) for each variable, a compromise had to be found to partly use the scaling to ensure that the variables would be in the same order of magnitude but at the same time the gradient step sizes were chosen appropriately.

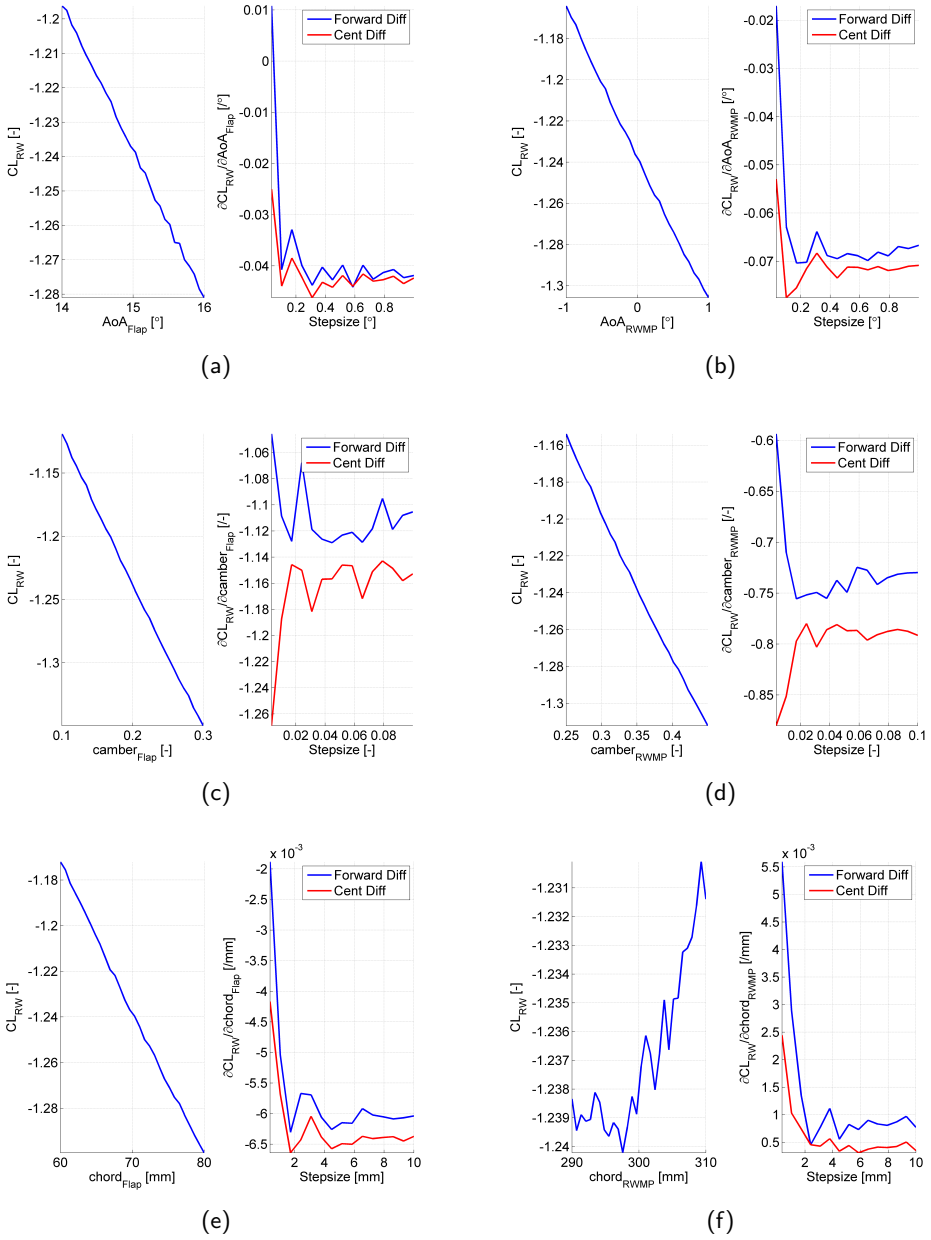


Figure 6.20 Parameter sweeps for test case 2 and the accompanying finite difference gradients as a function of the step size.

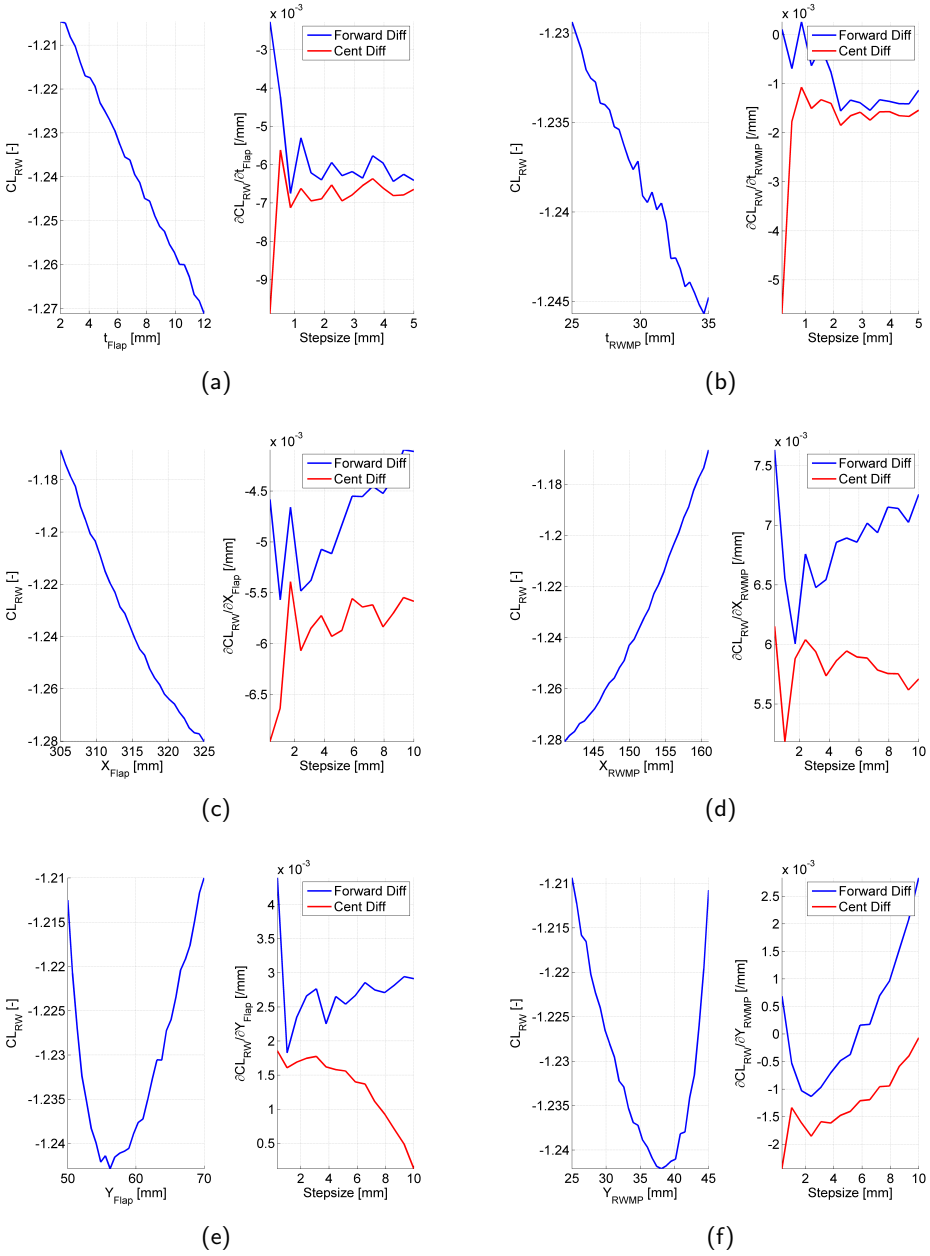
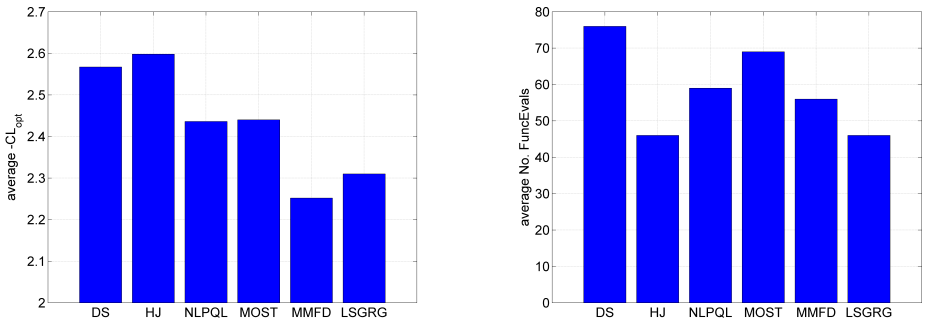


Figure 6.21 Parameter sweeps for test case 2 and the accompanying finite difference gradients as a function of the step size (cont.).

6.3.3 Results

This sub section presents the results for test case 2. First, the results obtained with the iSIGHT optimizers are presented and elaborated. Figure 6.22 shows the optimum C_L values found and the number of function evaluations required, averaged over the six sub cases.



(a) (b)

Figure 6.22 iSIGHT results averaged over the 6 sub problems of test case 2.

In terms of the optimum C_L and number of function evaluations, the following observations are made:

Optimum C_L

- There is a clear distinction between the two gradient-free algorithms on one side and the four gradient algorithms on the other side. The gradient-free algorithms clearly find better optimum values. Even though these two algorithms are local algorithms like the gradient algorithms, they *are* able to search the design space more globally. For DS, the setting that controls this is the initial simplex size. A larger initial simplex size increases the probability of finding the global optimum. As mentioned earlier, the value used was the default value, which is equal to 10% of the design space. For HJ, the particular setting is the so-called *Relative Step Size*. This is the step size the algorithm takes in each direction at the start of the optimization. The default value is 0.5, which means that in each direction it will span half the design space. For the gradient algorithms however, it is not possible to control the extent of the global search. If the starting point is chosen far away from the global optimum, they can easily get stuck in a local minimum or require more function evaluations to reach the optimum. DS and HJ are less sensitive to the starting point.
- Among the gradient algorithms, it is again apparent that the SQP methods find better optima than the feasible methods. Because of the parameterization

chosen, it was possible that the two airfoils would clash. Especially when greater numbers of design variables are used, the design space can consist of 'infeasible patches'. SQP algorithms can deal with these types of spaces more efficiently. Even though they could therefore sample at a point where the configuration clashes and therefore the objective function is undefined, they did not break down but simply decreased the step size along the line search.

Function evaluations

- Examining the number of function evaluations carried out by the optimizers, DS uses the most number of function evaluations whereas HJ uses the least. Upon closer inspection, it was found that HJ often stopped prematurely because too many runs failed. This was due to the fact that the geometries clashed, so somewhere in the process (CAD, surface/volume meshing) a failure would occur. This should not come as a surprise as HJ is not very delicate in choosing its samples. It simply chooses them based on the coordinate system used and on top of this, at the beginning the step sizes are quite large so for this particular problem it is bound to come across a clashed geometry.
- Another interesting observation was made for LSGRG. It was already found in test case 1 that this algorithm tends to follow constraint boundaries when it comes across them. This same behavior was observed in the current test case. For a couple of problems, the solver would try to follow a constraint boundary but because the gradient steps chosen were quite large (and the constraint boundaries were probably highly nonlinear), it was not able to follow them closely. For this reason it often found itself at the infeasible side of the boundary. When this happens and the solver is unable to recover itself back into feasibility, it stalls. This may explain why LSGRG uses the least number of function evaluations among the gradient algorithms.

Tables 6.14 and 6.15 shows the results obtained with KERS. For each subcase, the iSIGHT optimizer that found the best C_L is shown too. This would either be DS or HJ, as it was stated above that for this particular test case, the gradient-free methods found better function values than the gradient-based algorithms. *exploit* means that the infill points chosen is the best point on the response surface, while *explore* means that the infill point is the point with the maximum expected improvement (EGO).

A couple of observations are made:

- KERS finds better configurations than the best iSIGHT optimizer and generally in fewer function evaluations;
- The exploit mode finds better optima than the explore mode. This may be an indication that the response surface is not as multimodal as was expected. When the response is not multimodal, EGO is of course expected to be slower, since it tries to balance local (exploiting) search with global (exploring) search;

Table 6.14 KERS results for test case 2 (optimum C_L).

		KERS (exploit)			KERS (explore)		
Dim.	**iSIGHT**	**interp**	**regr**	**mix**	**interp**	**regr**	**mix**
2	-1.946	-1.940	-1.730	-1.949	-1.935	-1.627	-1.935
4	-2.665	-2.737	-2.712	-2.683	-2.680	-2.455	-2.674
6	-2.540	-2.824	-2.804	-2.833	-2.806	-2.803	-2.761
8	-2.535	-2.819	-2.771	-2.673	-2.547	-2.820	-2.592
10	-2.257	-2.909	-2.847	-2.929	-2.872	-2.761	-2.744
12	-2.387	-2.406	-2.709	-2.822	-2.321	-2.436	-2.631
	-2.388	-2.606	-2.596	-2.648	-2.527	-2.484	-2.556

Table 6.15 KERS results for test case 2 (Number of function evaluations).

		KERS (exploit)			KERS (explore)		
Dim.	**iSIGHT**	**interp**	**regr**	**mix**	**interp**	**regr**	**mix**
2	23	27	34	30	20	25	37
4	76	76	60	57	41	47	73
6	110	92	111	86	63	88	102
8	151	147	155	122	135	156	157
10	162	167	154	176	91	98	200
12	200	229	214	201	226	203	229
	120	123	121	112	96	103	133

- the interpolating response surface is better than the regressing surface. It was already found in the two-dimensional test case 1 that a regressing surface may converge slower because the response surface changes less radically when a new sample point is added. Since higher dimensional problems are considered in this test case and therefore the sample points are spaced further apart, the noise will not immediately become a hindrance;
- The adaptively regressing surface (mix) performs better than the purely interpolating surface.

'You talk the talk. Do you walk the walk?'
Animal Mother, Full Metal Jacket

7 Real-life problems

The two test cases presented in the previous chapter were very useful for comparing iSIGHT's local optimizers and KERS and contributed to the development and understanding of the developed tool. To determine whether KERS will indeed give a boost to the current LRGP optimization methodology, it was decided to test it out on a couple of actual optimization problems investigated in the CFD department. They were all part of a project focusing on the development of the R31's top rear wing (TRW)[1].
The 2011 season has seen the introduction of moveable rear wings as part of the Drag Reduction System. This regulation change was born out of the desire to increase overtaking in Formula One. By allowing the following car to rotate the leading edge of its rear wing flap upwards, the TRW will potentially stall. This loss in downforce is accompanied by a loss in induced drag, which should make it easier for the following car to overtake. Additionally, the leading car will not be allowed to stall its rear wing when the following car is within a specified distance. Because of the introduction of these moveable rear wings, a significant amount of effort has gone into developing effective rear wings, i.e. rear wings that will produce sufficient downforce in their normal state (the 'off-condition') and at the same time be able to shed as much drag as possible when the flap is activated (the 'on-condition').
To reduce the mesh count, the CFD model was defeatured (see Figure 7.1). Regarding the response surfaces built, they all used a constant regression term and a Gaussian correlation model. To tune the model parameters, the cross-validation method was used. Additionally, all response surfaces were interpolated. Not only were the CFD functions found to be much 'cleaner' than those of the test cases, the number of variables used was 4 or higher and the number of infill points was chosen to be small.

7.1 Twisted wing

In the DOE carried out for this part of the project, the TRW was parameterized with 8 variables:

[1]The R31 is LRGP's 2011 contender.

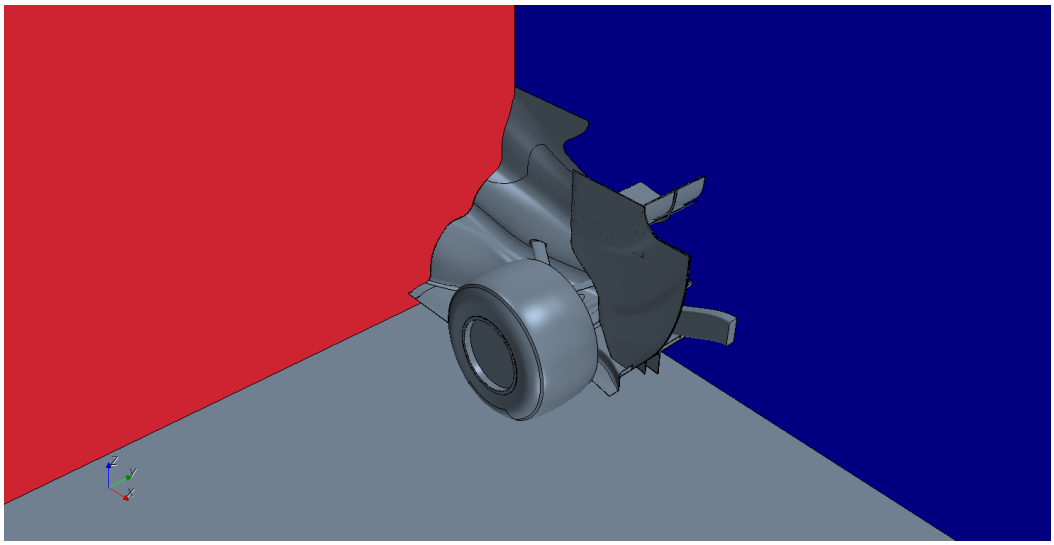


Figure 7.1 Optimization model used. The car was cut along the red surface, which imposes an inlet condition. The flow values at this surface were obtained from the corresponding half-car model. The blue surface is the symmetry surface.

- angle of attack (or twist) of the in-board (ib) and out-board (ob) flap sections;
- scale of the ib and ob flap sections;
- angle of attack of the ib and ob main plane (RWMP) sections;
- scale of the ib and ob RWMP sections.

The parameters were specified relative to a baseline case (*bs*). The lower and upper bounds for the angles were $-5°$ and $+5°$, respectively. The lower and upper bounds for the scales were -5% and $+5\%$, respectively. The DOE consisted of 180 design points, spread out using iSIGHT's Optimal Latin Hypercube algorithm. The focus of this particular problem was to maximize the downforce of the TRW in the off-condition while at the same time maximize the drag reduction when the flap is opened. Of course, the resulting geometry must also fit inside the legality box. Table 7.1 shows the results of the baseline run (*bs*), the best[2] design from the DOE (*bs_DOE*) and a shrunken version of it that would fit inside the box (*bs_DOE_s*). All the aerodynamic loads are expressed with respect to the baseline. Observe that even though *bs_DOE* is generating 1pt[3] more downforce and shedding 4units more drag than baseline, it is sticking 7.7mm out of the bottom legality box. The shrunken, legal version generates 1.3pts less downforce than *bs_DOE*. Using the DOE results, response surfaces were built for the following functions:

[2] This is the DOE point that produced a large amount of downforce without being 'too illegal'.

[3] A point (pt) is 0.01 of force coefficient while a unit is 0.001. The unit is the smallest force coefficient value that is considered to be of significance.

Table 7.1 The baseline design (*bs*), the best DOE points (*bs_DOE*) and a legalized version (*bs_DOE_s*) of it.

Run	ΔCL_{RW} [-]	ΔCD_{RW} [-]	legal_bottom [mm]	legal_front [mm]
bs	0	0	0.69	-0.024
bs_DOE	-0.010	-0.004	7.7	0.33
bs_DOE_s	+0.003	-0.001	-0.40	-0.79

s

- CL_{RW}; CFD function representing the C_L of the rear wing assembly in the off-condition.
- CD_{RW}; CFD function representing the amount of C_D shed by the rear wing assembly when switching from the off- to on-conditions.
- legal_bottom; geometric function representing the distance between the bottom surface of the TRW legality box and the lowest point of the RWMP. Negative values indicate a legal configuration, i.e. the RWMP is not puncturing the bottom surface of the legality box.
- legal_front; geometric function representing the distance between the front surface of the TRW legality box and the front most point of the RWMP. Again, negative values indicate a legal configuration.

The values for the CFD functions were averaged over the last 100 iterations. The values for the geometric functions were obtained from CATIA. With the parameterization used and the imposed bounds on the variables, the top and back surfaces of the legality box could not be crossed, so these functions were not included in the surrogate model. Figure 7.2 shows one of the DOE designs, including the TRW legality box. This particular design was sticking out of the front and bottom surfaces of the legality box.

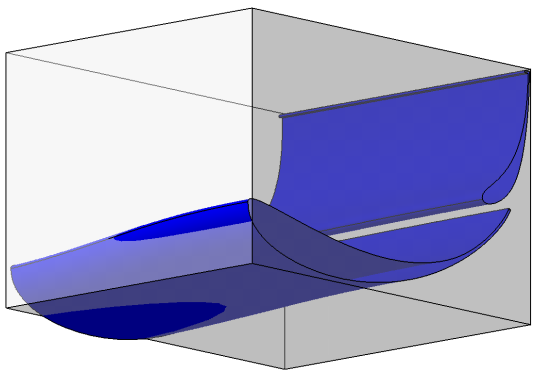


Figure 7.2 TRW with legality box.

Figure 7.3 shows the cross-validation plots for the four functions while Table 7.2 gives the average and maximum absolute errors for each fitted functions. Looking at the cross-validation plots and the error metrics, the fit appears to be satisfactory.

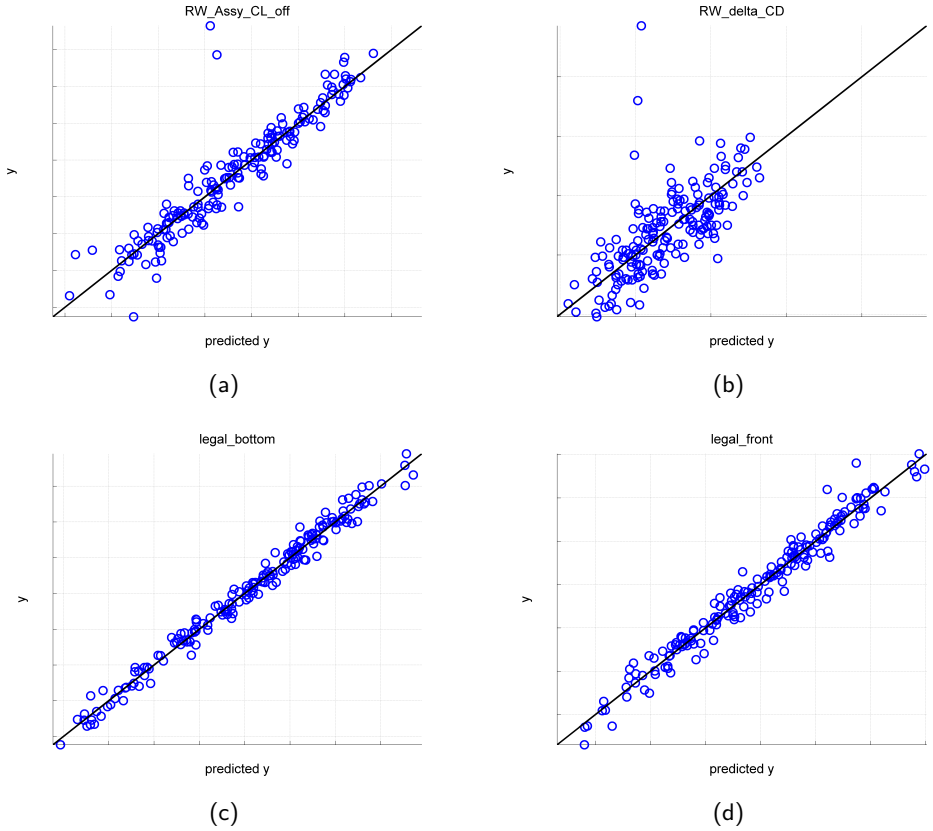


Figure 7.3 Cross-validation plots for the four fitted functions.

Table 7.2 Average and maximum absolute errors calculated by means of cross-validation.

	Absolute error		
Parameter	**Average**	**Maximum**	**Unit**
CL_{RW}	0.3	4.5	pts
CD_{RW}	0.2	1.9	pts
legal_bottom	0.87	3.2	mm
legal_front	0.93	5.3	mm

The response surfaces were then used for optimization purposes. Four different cases were run:

1. $C_L - exploit$: minimize CL_{RW} by searching the optimum on the response

surface;

2. $C_D - exploit$: minimize CD_{RW} by searching the optimum on the response surface;

3. $C_L - explore$: minimize CL_{RW} using EGO;

4. $C_D - explore$: minimize CD_{RW} using EGO.

Table 7.3 shows the results. Interestingly, the configurations all lie on the bottom surface of the legality box, indicating that this constraint is the main limiter in achieving a maximum downforce and drag switch.

Table 7.3 Optima obtained from the surrogate model.

	Exploit		Explore		
Parameter	$\mathbf{C_L}$	$\mathbf{C_D}$	$\mathbf{C_L}$	$\mathbf{C_D}$	**Unit**
ΔCL_{RW}	-0.004	+0.003	-0.001	+0.004	-
ΔCD_{RW}	-0.002	-0.005	-0.001	-0.005	-
legal_bottom	0.0	0.0	0.0	0.0	mm
legal_front	0.0	-3.3	-0.50	-3.9	mm

The proposed configurations were subsequently run in CFD. Table 7.4 shows the prediction errors. The predictions appear to be quite accurate. The largest error for the geometric constraints is 1.9mm. The largest error for CL_{RW} is 3units, while the largest error for CD_{RW} is 5units. Comparing the $C_L - exploit$ configuration with baseline, gains of 3 units in downforce and 3 units in drag delta are achieved.

Table 7.4 Prediction errors.

	Exploit		Explore		
Parameter	$\mathbf{C_L}$	$\mathbf{C_D}$	$\mathbf{C_L}$	$\mathbf{C_D}$	**Unit**
CL_{RW}	+0.001	+0.000	+0.003	+0.001	-
CD_{RW}	-0.001	+0.005	+0.000	+0.004	-
legal_bottom	+1.0	-1.9	+0.13	-1.8	mm
legal_front	+1.1	-0.3	+0.00	-0.5	mm

To understand how the CL_{RW} and CD_{RW} are related, the surrogate model was used to construct a Pareto front. The algorithm used for this was one of iSIGHT's genetic algorithms called AMGA, based on the work of Tiwari *et al.* [49]. Figure 7.4(a) shows all the points, legal and illegal, that the algorithm sampled. The positive horizontal direction corresponds to a decrease in downforce and the positive vertical direction corresponds to a decrease in drag switch. As the rear wing generates more downforce, it can shed more drag when the flap is actuated. This is of course what

one would expect as more downforce is accompanied with more (induced) drag. However, it seems that the legality box is limiting this trend; observe how the red points reach deeper into the bottom left corner of the plot. Figure 7.4(b) zooms into this limiting region and has removed the illegal points for clarity. A Pareto front is visible indicating that at a certain point, if one wants to increase downforce while keeping the configuration legal, the amount of drag that one can shed will naturally be reduced.

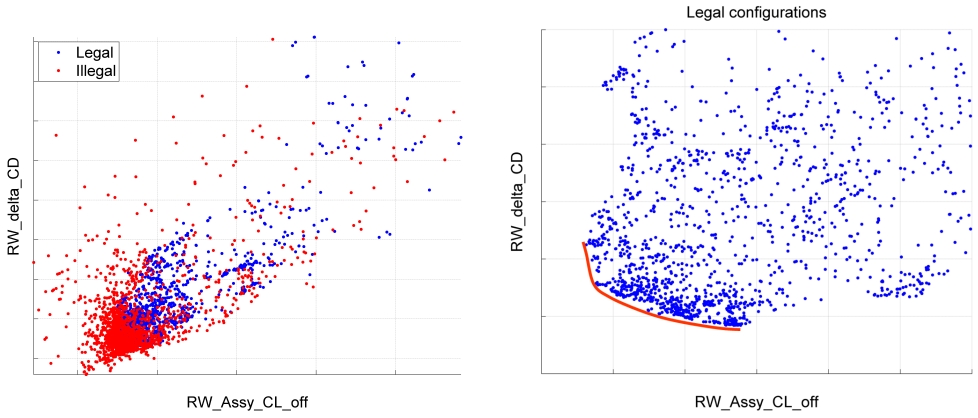


Figure 7.4 Legal design points generated by AMGA. The red curve is the Pareto front.

7.2 Extruded wing

In this new DOE, 4 variables were used:

- angle of attack of the flap;
- scale of the flap;
- angle of attack of the RWMP;
- scale of the RWMP.

Contrary to the twisted wing, in this DOE the ib and ob sections were the same (i.e. the wing was a 2D extrusion). The parameters were specified relative to a baseline case (bs). The lower and upper bounds for the angles were $-5°$ and $+5°$, respectively. The lower and upper bounds for the scales were -10% and $+10\%$, respectively. A DOE of 80 points was set up. 3 runs failed, leaving 77 points that could be used for building the surrogate model. The focus of this particular problem was to maximize the downforce of the TRW in the off-condition and at the same

time maximize the drag switch when the flap is rotated. Table 7.5 shows the results of the baseline run, together with two interesting DOE points (*DOE_1* and *DOE_2*) with their accompanying scaled versions that should fill the box better (*DOE_1_s* and *DOE_2_s*). *DOE_2_s* generates the highest downforce while *DOE_1_s* produces the largest drag switch.

Table 7.5 The baseline design (*bs*), two interesting DOE points and two manually tweaked versions.

Run	ΔCL_{RW} [-]	ΔCD_{RW} [-]	legal_bottom [mm]	legal_front [mm]
bs	0	0	0.67	-0.21
DOE_1	+0.007	-0.002	-5.15	-13.04
DOE_1_s	+0.001	-0.004	-1.35	-0.86
DOE_2	-0.001	+0.003	0.01	-3.78
DOE_2_s	-0.003	+0.003	0.89	-0.64

Again, response surfaces were built for the four functions. Table 7.6 gives the average and maximum absolute errors for each fitted function. Looking at dation plots and the error metrics, the fit appears to be satisfactory.

Table 7.6 Average and maximum absolute errors calculated by means of cross-validation.

	Absolute error		
Parameter	Average	Maximum	Unit
CL_{RW}	0.1	0.3	pts
CD_{RW}	0.1	0.2	pts
legal_bottom	1.9	7.7	mm
legal_front	0.14	0.49	mm

Before using the surrogate model for optimization, it was decided to gain an understanding of the problem first. To this end, a couple of plots were generated. As there were four variables, the nested plot shown in Figure 7.5 was made. The represented function is the CL_{RW}. The horizontal axis represents the angles of attack while the vertical axis represents the scales. The axes on the mini tiles belong to the flap while the main tile belongs to the RWMP. The RWMP angle and scale were subdivided into 13 intervals, explaining why there are 13 mini tiles in both directions. The most downforce is generated by the blue regions. The figure indicates that for both the RWMP and flap, an increase in scale will increase the downforce generated by the RW. Indeed, an increased scale means a bigger surface area, so more load can be generated. Regarding the angle of attack; lower values (right to left) mean that the leading edge is moved down, while the trailing edge is kept constant. So, lower values should see an increase in downforce, which is indeed what can be observed from the figure.

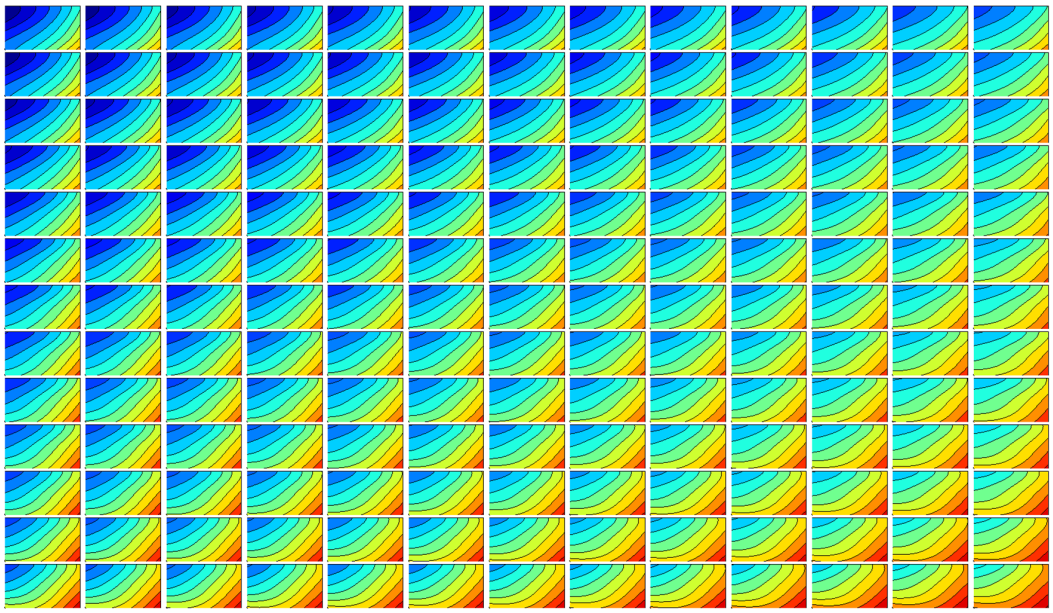


Figure 7.5 Nested plot of the CL_{RW} for the extruded wing. The horizontal axis represents angles of attack while the vertical axis represents the scales. The axes on the mini tiles belong to the flap while the main tile belongs to the RWMP.

Figure 7.6 gives the nested plot of the CD_{RW}. This function shows a more complex behavior with the four variables. Blue regions represent configurations that generate a large drag switch. The following observations can be made:

- an increasing RWMP scale appears to result in a decreasing drag delta;
- an increasing flap scale appears to result in an increasing drag delta;

The next step was to optimize the response surface. As for the twisted wing, four different cases were set up: $C_L{-}exploit$, $C_D{-}exploit$, $C_L{-}explore$ and $C_D{-}explore$. Table 7.7 shows the results (the $C_L\&C_D$ column is discussed later).
The proposed configurations were subsequently run in CFD. Table 7.8 shows the prediction error. The predictions appear to be quite accurate. The largest error for the geometric constraints is 2.1mm. The largest error for CL_{RW} is 3units, while the largest error for CD_{RW} is 1unit. Comparing the $C_L - exploit$ design with the baseline run, a gain of 4 units in CL_{RW} and 3 units in CD_{RW} has been achieved. Comparing with the best design obtained by manually scaling a DOE point (DOE_2_s), a gain of 1 unit in CL_{RW} has been achieved with a 6units gain in CD_{RW}. As in the previous problem, the surrogate model was linked with AMGA to construct a Pareto front for the downforce and drag switch. Figure 7.7 shows the results. The red curve shows the Pareto front indicating the limit where an increase in downforce will cause a decrease in drag switch. From the Pareto front, a design point was chosen and subsequently run.

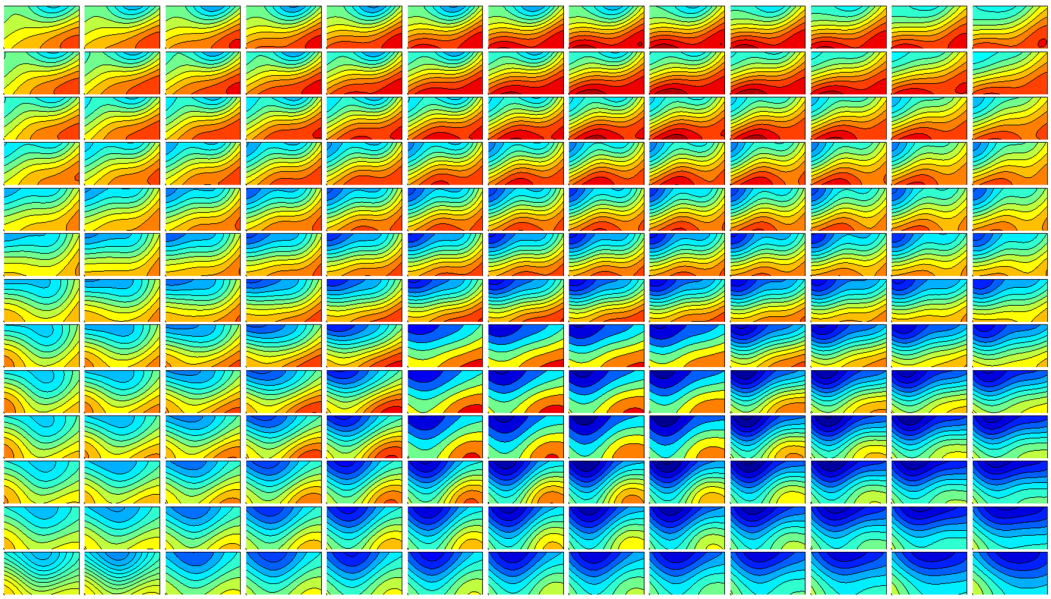


Figure 7.6 Nested plot of the CD_{RW}. The horizontal axis represents angles of attack while the vertical axis represents the scales. The axes on the mini tiles belong to the flap while the main tile belongs to the RWMP.

Table 7.7 Optima obtained from the surrogate model.

	Exploit			Explore		
Parameter	$\mathbf{C_L}$	$\mathbf{C_D}$	$\mathbf{C_L}$&$\mathbf{C_D}$	$\mathbf{C_L}$	$\mathbf{C_D}$	**Unit**
ΔCL_{RW}	-0.005	+0.003	-0.005	-0.004	+0.003	-
ΔCD_{RW}	-0.002	-0.004	-0.002	-0.000	-0.004	-
legal_bottom	0.0	0.0	0.0	0.0	0.0	mm
legal_front	0.0	0.0	-0.13	-19	0.0	mm

This point corresponds to the C_L&C_D column of Table 7.7. It turned out that the actual CL_{RW} was the same as predicted (within the unit accuracy), while the drag delta was 2 units more than predicted. In addition, legal_bottom and legal_front were -0.59mm and -0.015mm, respectively, meaning that the configuration was fully legal.

7.3 Airfoil optimization

This DOE focused on maximizing downforce. For these design points, only the off-condition was therefore run. The baseline for this case was the point chosen on the Pareto front in the previous problem. Again, the TRW was chosen to be a 2D extrusion. Both the RWMP and flap were parameterized with 6 variables. In

Table 7.8 Prediction errors

	Exploit		Explore		
Parameter	$\mathbf{C_L}$	$\mathbf{C_D}$	$\mathbf{C_L}$	$\mathbf{C_D}$	**Unit**
ΔCL_{RW}	+0.001	+0.000	+0.003	+0.001	-
ΔCD_{RW}	-0.001	-0.001	-0.001	+0.000	-
legal_bottom	-0.59	0.40	+2.1	+0.29	mm
legal_front	+0.22	+0.0	-19	+0.032	mm

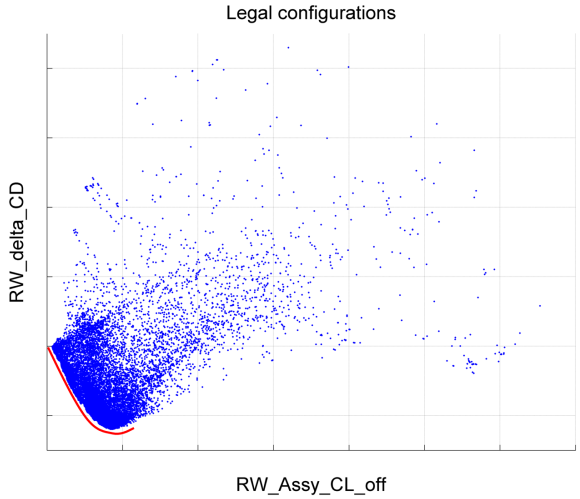


Figure 7.7 Legal design points generated by AMGA. The red curve is the Pareto front.

addition to the angle of attack and scale, four camber variables were defined on the main plane and flap:

- camber_frt
- camber_mid
- camber_rr
- camber_rr_angle

The total number of variables was therefore 12. A DOE of 400 points was set up. Since the parameterization and the imposed variable bounds allowed for certain configurations to cross the back of the legality box, a response surface for this constraint was also built, in addition to the CL_{RW} and the bottom and front constraints. Table 7.9 gives the average and maximum absolute errors for each fitted function. From the DOE, the design point producing the highest load was outside the legality box; 5mm at the back, 3mm at the front and a considerable 21mm at the bottom.

Table 7.9 Average and maximum absolute errors calculated by means of cross-validation.

	Absolute error		
Parameter	**Average**	**Maximum**	**Unit**
CL_{RW}	0.3	2.6	pts
legal_back	0.07	0.90	mm
legal_bottom	0.13	0.66	mm
legal_front	0.12	0.49	mm

The configuration that produces a good amount of load but that did not stick out the legality box considerably (8mm and 9mm at the bottom and front) produced 1.5pts more downforce. This configuration was shrunk to fit the box, which reduced the downforce with 1.1pts. Using the surrogate model, an optimization was carried out to find a legal configuration that could generate more RW downforce than this bsf configuration. To determine the infill points, it was decided to alternate the auxiliary function between the CL_{RW} and the expected improvement. Table 7.10 shows the results relative to the bsf. The best design point was found in iteration 7. Even though it generates the same load as the design point found in the 11^{th} iteration, the former was completely inside the box. Note that the prediction error indeed goes down and on top of this the expected improvement clearly diminished. This gives one confidence that a configuration close to the global optimum was found. All in all, the response surface methodology has given an improvement of 6 units over the previously re-scaled DOE point.

Table 7.10 Progress of the optimization using the surrogate model built.

	$\Delta \mathbf{CL_{RW}}$		$\lvert\mathbf{Error}\rvert$	
Iteration	**predicted**	**actual**	**[pts]**	$\mathbf{E}\left[\mathbf{I}\left(\mathbf{x}\right)\right]$
1	-0.012	-0.001	1.1	1.5E-02
2	-0.009	-0.002	0.7	2.0E-02
3	-0.013	-0.001	1.2	1.3E-02
4	-0.010	+0.005	1.5	1.0E-02
5	-0.008	-0.003	0.5	5.7E-03
6	-0.003	-0.004	0.1	5.9E-03
7	-0.004	-0.006	0.2	2.1E-04
8	-0.004	-0.005	0.2	2.4E-04
9	-0.007	-0.005	0.2	3.5E-04
10	-0.007	-0.005	0.2	8.2E-05
11	-0.006	-0.006	0.0	9.9E-06

7.4 Rib integration

The three design problems described in the previous sections did not have a legality rib[4]. The present section describes a DOE where the rib was included, as shown in Figure 7.8. Note that the TRW shown in Figure 7.8 is the left half of the complete TRW (i.e. the full TRW has two legality ribs). The presence of the rib can cause the flow around it to separate from the suction surfaces of the flap and RWMP, leading to a loss in downforce. To try and regain this loss, a DOE with four variables was set up. The variables were scale and angle of attack of the flap and RWMP sections at the position of the legality rib. The blue surfaces were kept unchanged, while the green surfaces were set up such that they would provide a smooth blend. The scales were bounded between -10% and $+10\%$ of the baseline, while the angles were bounded between -10° and $+10^\circ$. Figure 7.9 shows the velocity magnitude at a vertical plane behind the TRW for the baseline, which was an extruded wing. One can clearly identify the stall coming from the region where the rib resides (the blue region). A DOE of 70 points was set up. The configuration producing the highest load (4pts more than baseline) was well outside the legality box (31mm outside the bottom). The configuration that produced the highest load while being the closest to legality (1mm outside the back) produced 2.2pts more downforce than baseline. This was the best so far (bsf) from the DOE. To understand how the parameters chosen influence the C_L, a response surface was built for it. In addition, response surfaces were built for the back, bottom and front legality constraints. Table 7.11 gives the average and maximum absolute errors for each fitted function.

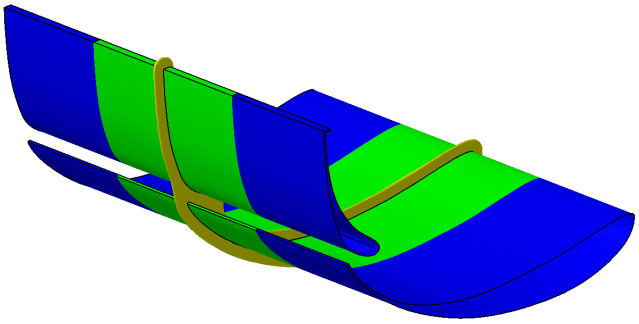


Figure 7.8 TRW with legality rib.

As there were four variables, it was possible to visualize the whole design space in one plot, using the concept of the nested plot introduced in Section 7.2. Figure 7.10 shows the nested plot for the present problem. The horizontal axis represents angles of attack. Going from right to left is equivalent to drooping the leading edge of the section at the position of the legality rib. The vertical axis represents the scales of the sections. Going from the bottom to the top is equivalent to increasing the scale

[4] A stiff spacer separating the RWMP and flap in order to maintain a constant slot gap. It is required for legality.

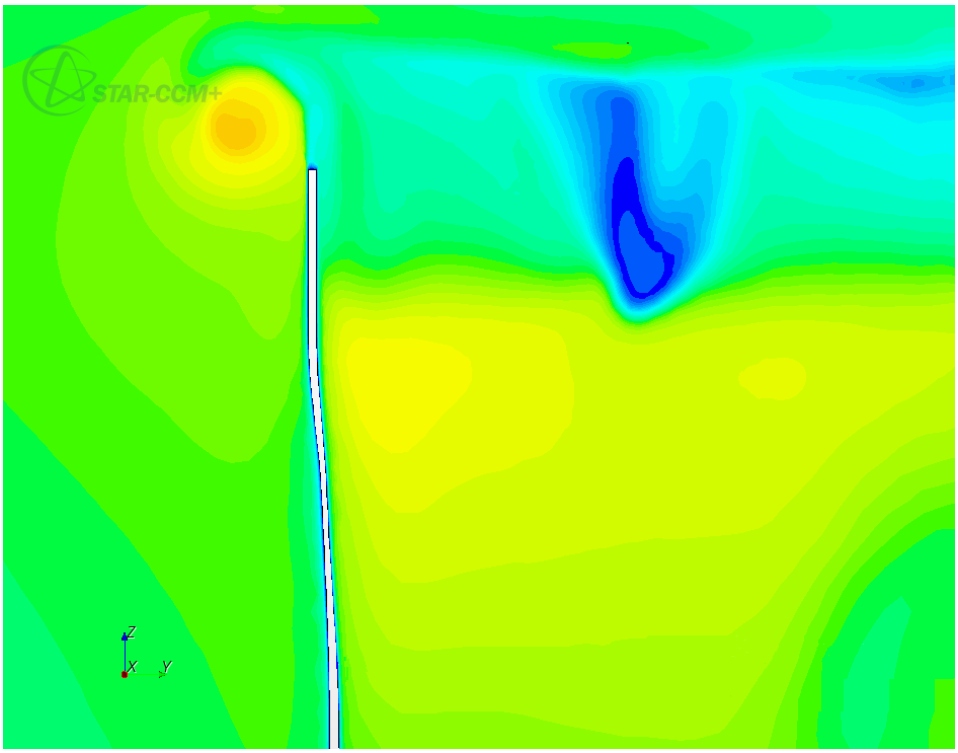


Figure 7.9 Velocity magnitude behind the baseline TRW with legality rib.

Table 7.11 Average and maximum absolute errors calculated by means of cross-validation.

	Absolute error		
Parameter	**Average**	**Maximum**	**Unit**
CL_{RW}	0.8	2.7	pts
legal_back	0.002	0.02	mm
legal_bottom	0.89	2.8	mm
legal_front	0.47	3.2	mm

of the sections. It is therefore clear from the figure that increasing the scale of the section and drooping it will increase the downforce. Using the surrogate model, a couple of iterations were carried out. For all iterations, the auxiliary function used was the actual C_L on the response surface ('exploit'). The reason for not using the expected improvement is because prior to carrying out the DOE, the response surface was not expected to be multimodal. Examining Figure 7.10, one can observe that this is indeed the case. An additional reason is that the 'exploit' approach tends to converge faster, which was already confirmed in the previous chapter for test case 2. Table 7.12 shows the results of the iterations, relative to the bsf. The best legal configuration was found in the first iteration. Comparing it with the bsf, the gain

achieved is 1pt. Figure 7.11 shows the configuration, together with the TRW legality box.

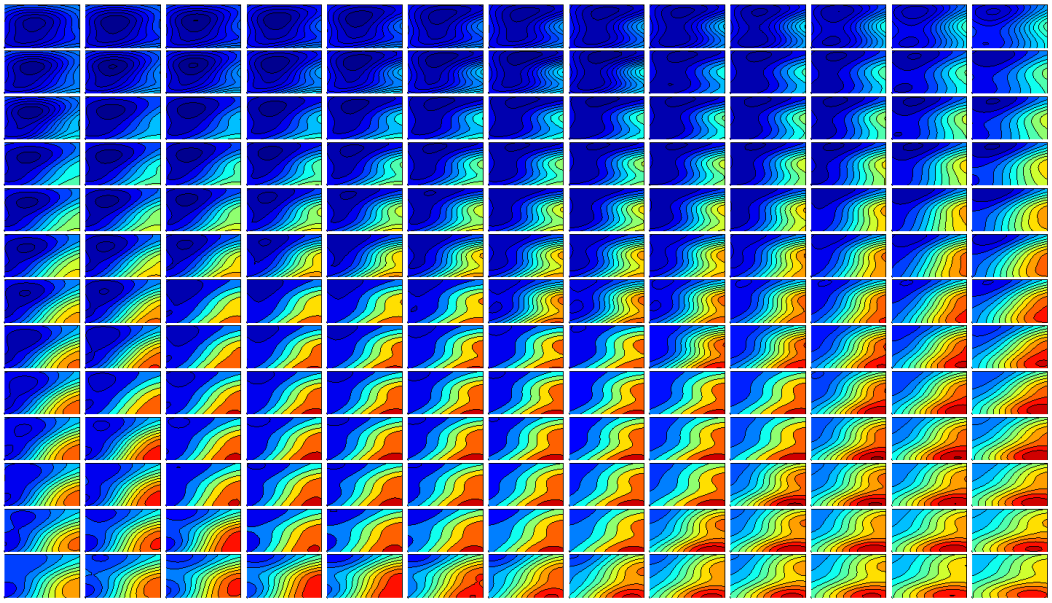


Figure 7.10 Nested plot of CL_{RW}. The horizontal axis represents angles of attack while the vertical axis represents the scales. The axes on the mini tiles belong to the flap while the main tile belongs to the RWMP.

Table 7.12 Progress of the optimization using the surrogate model built.

	Flap		Main plane			ΔCL_{RW}		|Error|
iteration	angle	scale	angle	scale	Max_legal	predicted	actual	[pts]
1	-4.74	0.1	2.39	-0.06	+0.0	-0.021	-0.010	1.1
2	-6.00	0.1	3.55	-0.02	+0.0	-0.014	-0.010	0.4
3	0.36	0.1	2.45	-0.07	+0.0	-0.012	-0.000	1.2
4	-5.99	0.1	2.52	-0.06	+0.0	-0.013	-0.008	0.5
5	-5.99	0.1	4.27	0.02	+0.3	-0.020	-0.012	0.8
6	-3.62	0.08	0.70	-0.07	+1.9	-0.012	-0.004	0.8

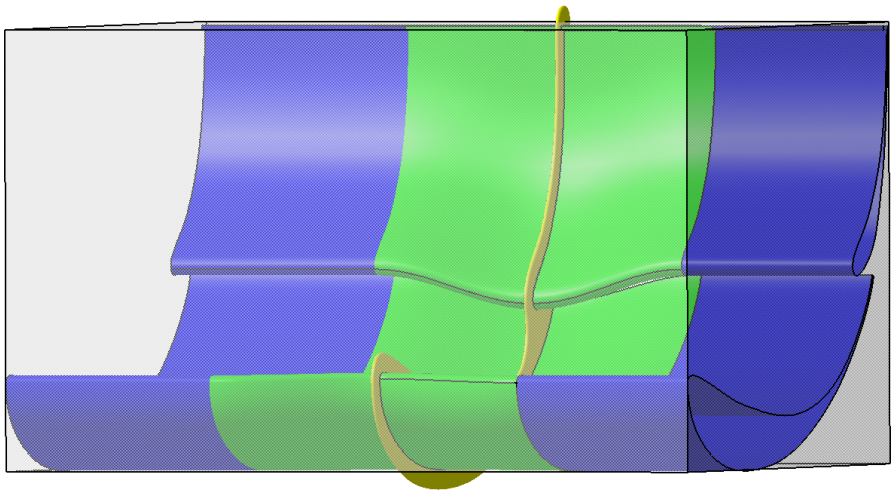


Figure 7.11 Best legal configuration found by the response surface optimization.

'A conclusion is the place where you got tired thinking.'

Martin Henry Fischer

8 Conclusions & recommendations

The next section describes how the present research has contributed to enhancing the current optimization methodology, followed by suggestions for possible future work.

8.1 Conclusions

Two approaches to optimization were investigated in this research, and assessed using two CFD test cases.

8.1.1 Local solvers

The first optimization approach investigated is the traditional approach, in which a numerical algorithm is used to solve the optimization problem. The following conclusions are drawn:

- *SQP methods showed the best performance*: they found the best optimum in the least number of function evaluations.
- *noise can decrease the performance of the algorithms*; when applied on noisy functions, the actual values these solvers found were lower than for the noise-free versions.

8.1.2 The surrogate modeling framework

The surrogate modeling technique used in this research was *Kriging*. The following modifications were introduced and investigated in order to improve the quality of the surrogate model:

- *use a cross-validation error metric instead of the maximum likelihood estimator to find the optimum response surface*; The latter is the widely accepted methodology in the open literature. It was found that for scarce sample sets, the cross-validation idea can give more accurate response surfaces;

- *allow the Kriging surface to regress sample points*; this is particularly desirable when the underlying function is noisy and there are clustered sample points. It was found that for noisy functions, the smoothing indeed stabilizes the optimization procedure. This was evident in the two dimensional test case 1 but less so in the higher dimensional problems in test case 2. Indeed, as the number of dimensions increases, sample points will be spaced further apart so noise will be less of a hindrance. Since an interpolating surface will change more considerably as a result of an infill point it can converge more quickly to the actual optimum;
- *adaptive regression*; a method was introduced which will apply the level of smoothing to each sample point that is proportional to its standard deviation. The extreme case of this was to only regress sample points that are clustered within a user specified threshold. It was concluded from the test cases that it indeed provides better results than the fully interpolating and fully regressing methods.

To find the global optimum on the response surface, the global optimizer DIRECT was used. Originally, the algorithm could only handle box-constraints. So, DIRECT was extended to handle nonlinear equality and inequality constraints. This was achieved by using an exact penalty function.
The development of the surrogate modeling framework culminated in the program *KERS*. For the two test cases, KERS was found to compare favorably with the iSIGHT optimizers.
Finally, KERS was used on a couple of real-life CFD design projects. For all these projects, it showed to be a good improvement over the current optimization methodology as it was able to enhance the understanding of a design problem while at the same time find better designs when used in an optimization loop. These two features clearly make surrogate modeling attractive in the context of using expensive codes in the design process, contrary to the more standard approach of using black-box optimizers.

8.2 Recommendations

8.2.1 Local solvers

Instead of using gradients obtained from finite differencing, it is possible to calculate them with more accurate methods like the complex variables method or exact differentiation. More accurate gradients should lead to a better performing gradient-based algorithm. The adjoint approach can be very useful in this respect, as it is capable of obtaining accurate gradients at the cost of a CFD function evaluation. In the present context, the challenges will be to obtain (a part of) the source code of an *accurate* CFD solver. Indeed, if the primal is not accurate enough in reliably

modeling the complex flow around an F1 car, then the gradients will not be as useful. Then, one needs to develop the actual adjoint solver.

8.2.2 The surrogate modeling framework

Regarding the surrogate modeling framework, since KERS has been a first attempt to using this concept within the CFD optimization methodology at LRGP, there is still significant room for improvement:

- *make use of gradient-enhanced Kriging*; the Kriging surface will therefore not only go through a sampled point, it will also have the same gradient there. Compared to the standard Kriging approach, one can therefore either obtain a more accurate response surface for the same number of points or the same accuracy by using fewer points. Note that gradient-enhanced Kriging is only useful when the gradients can be obtained cheaply. Recall that the adjoint method is capable of obtaining the gradient at the cost of one function evaluation.

- *use a surrogate model for other purposes than optimization*; In the present context, where the car is continuously being developed both in CFD and in the wind tunnel, an interesting application would be to combine the results from these two methods into one model. Another application may be to use results from a high-fidelity code to calibrate results from a low-fidelity code. The surrogate model would then serve as a transfer function which maps the discrepancy between the two stages across the design space. This will enable one to carry out a large number of runs with the low-fidelity code, using the surrogate model to 'correct' the results. In the present context, the high-fidelity code would be the solution obtained from Star-CCM+ applied on an accurate half-car (or even full-car) model. The low-fidelity code can be a variety of things. For example, it could be a defeatured model. Another possibility would be to use results from a panel method like NEWPAN as the low-fidelity code.

- *when using adaptive regression, try to quantify the error at each sample point more accurately.* It was briefly mentioned and shown that using the standard deviation of the function as an error metric can provide better results than full regression. If a better error estimate can be found, for example by using the adjoint solution, then it may be possible to obtain an even better response surface.

A

A description of the global optimizer DIRECT

This appendix describes the workings of the global optimization algorithm DIRECT[1], which is used to find the global optimum on the surrogate response surface. It was proposed by Jones *et al.* [25]. The Matlab source code of the implementation used in this research can be found in the article by Björkman and Holmström [3]. DIRECT is a Lipschitzian optimization method. The Lipschitz constant K gives an upper bound on the rate of change of a function. Knowledge of this constant is a powerful tool for global optimization. An algorithm that makes use of this constant comes with a number of advantages:

1. convergence theorems can easily be proved;
2. these methods are deterministic so there is no need for multiple runs;
3. besides the Lipschitzian constant itself, one does not have to define a large number of parameters to set up the algorithm, therefore minimizing the need for parameter fine-tuning;
4. Lipschitzian methods can define bounds on how far they are from the optimum. They can therefore use a stopping criterion which is more meaningful than simply specifying an iteration limit.

However, specifying the Lipschitz constant is not easy. In fact, determining the correct value is as difficult as finding the global optimum since it boils down to finding the maximum of the gradient's magnitude. For this reason, Lipschitzian methods will usually specify a large value for this constant such that it is at least higher than the actual value. As will be explained shortly, in the context of global optimization the Lipschitzian constant (or, to be more specific, its *approximation*) can be interpreted to be a weighting parameter specifying how local and global search should be balanced. High values put more emphasis on global search. It

[1] The name DIRECT not only refers to the fact that the algorithm is a direct technique, i.e. it does not makes use of function gradients, it is also an abbreviation for DIviding RECTangles, a reference to the way the method works.

can therefore be appreciated why the standard Lipschitzian methods can exhibit slow convergence speeds. To get a feel for how a Lipschitzian optimizer works, Figure A.1 shows Shubert's [47] algorithm for a one-dimensional function at three subsequent iterations. The black dots are the points where the objective function has been evaluated. The magnitude of the slope of the blue lines is equal to the (approximation of the) Lipschitz constant K. Since it represents the upper bound of the rate of change of the function, the function will never be below the V shapes. At each iteration, the function is evaluated at the lowest V-notch in the interval. It can easily be verified that the value of a V-notch is equal to:

$$\frac{f(a)+f(b)}{2}-K(b-a) \tag{A.1}$$

where a and b are the left- and right-hand sides of an interval, respectively and $f(\cdot)$ is the one-dimensional function. The first term is low when the function values at the endpoints are low. The second term is low when the interval is big (relatively unexplored region). In other words, the first term emphasizes local search while the second term emphasizes global search.

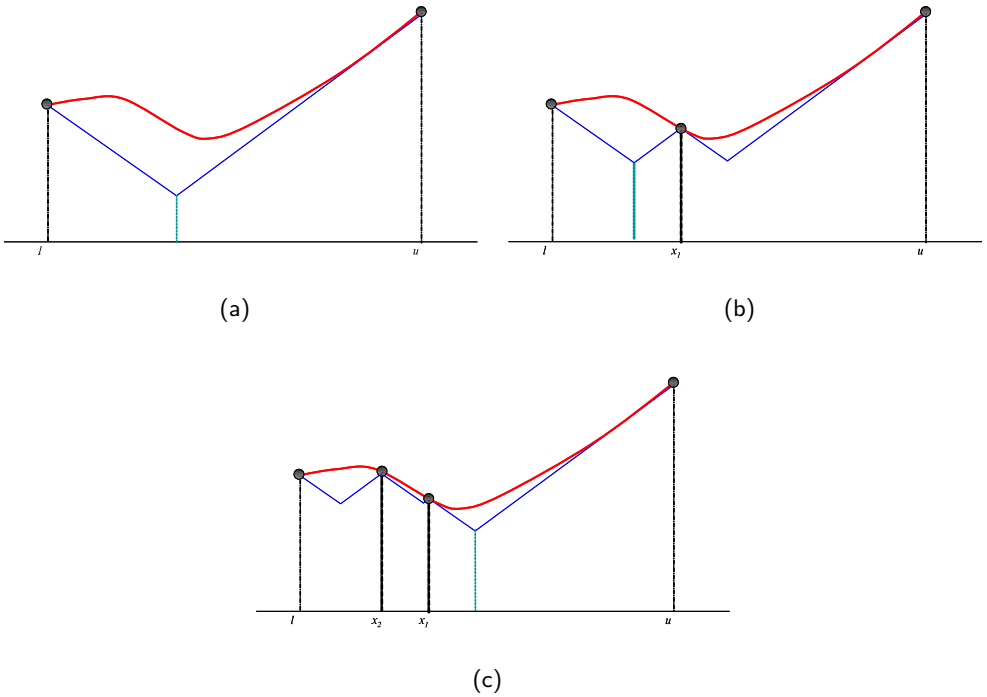


Figure A.1 Shubert's algorithm.

DIRECT makes the following modifications to Shubert's algorithm:

1. instead of evaluating an interval at its endpoints (vertices), it is evaluated at its center point. Shubert's algorithm becomes impractical in higher dimensions, since an interval becomes a hypercube with 2^n vertices, where n is the number of dimensions. The algorithm DIRECT on the other hand will always start with one point at the center of the hypercube. Shifting to center-point sampling requires two additional modifications:

   - the 'V-shape' constructed using the Lipschitzian constant becomes an 'inverted V', with the notch located at the sampled center. So, instead of the notch being the hypothetical minimum, its two endpoints become the hypothetical minima;
   - an interval is subdivided into three to ensure that center sampling can still be carried out in subsequent iterations. Figure A.2 illustrates how these modifications would play out in practice.

2. assume that the space has already been subdivided into m intervals $[a_i, b_i]$, $i = 1, ..., m$ and the next step is to choose the interval that should be sampled. Figure A.3(a) plots these intervals where the horizontal axis represents the distance from the interval's center and the vertical axis represents the function value at this center. The blue line has slope K. Therefore, if the line is taken through one of these points, its intersection with the vertical axis will represent the hypothetical minimum for that particular interval. Shubert's algorithm will choose the interval that produces the lowest hypothetical minimum (the red dot). From the figure it can be derived that the larger K is, the more global the search will become since large intervals will be chosen. Instead of choosing one value for K, DIRECT will essentially evaluate *all* possible K values by choosing the sample points that form the bottom right edge of the convex hull, as shown in Figure A.3(b). The green dots are the intervals that will be subdivided. For this reason, DIRECT will not put all emphasis on global search. Rather, at each iteration it will carry out global *and* local search. In other words, the user does not have to specify a Lipschitzian constant.

To understand how DIRECT would work in more than 1 dimension, Figure A.4 shows three iterations of the algorithm in two dimensions.

A.1 Constraint handling

DIRECT can only handle bounded problems. In the present context, it is very important for an optimizer to be able to handle nonlinear equality and inequality constraints. This section therefore describes how DIRECT was extended to handle nonlinear equality and inequality constraints.
As described in Chapter 2, the easiest way of handling constraints is to use an indirect method (penalty function), which is the method chosen here. An additional

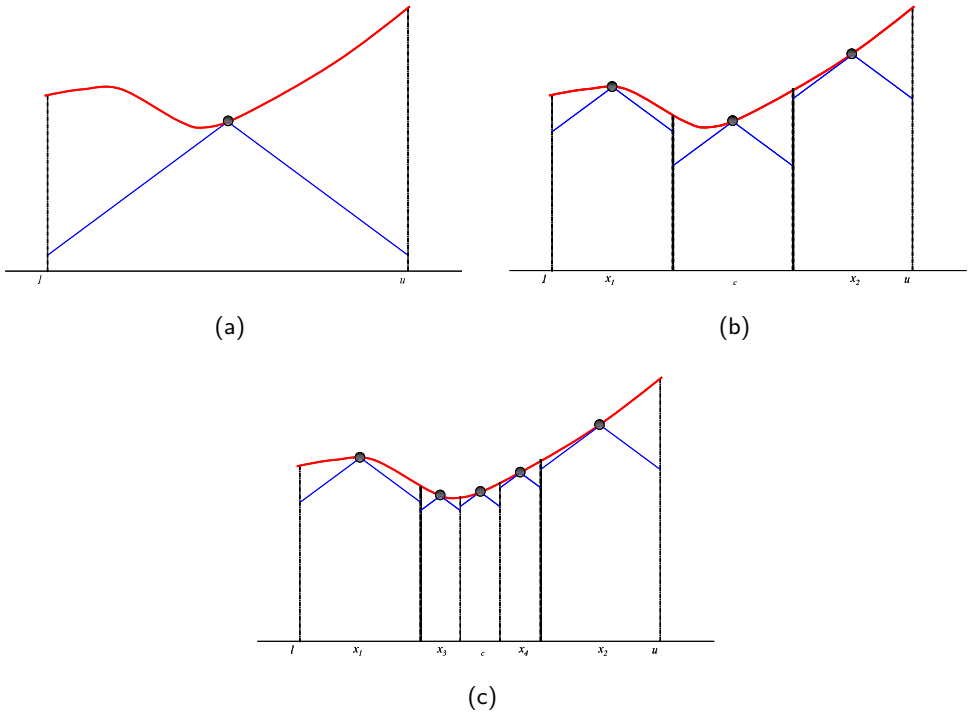
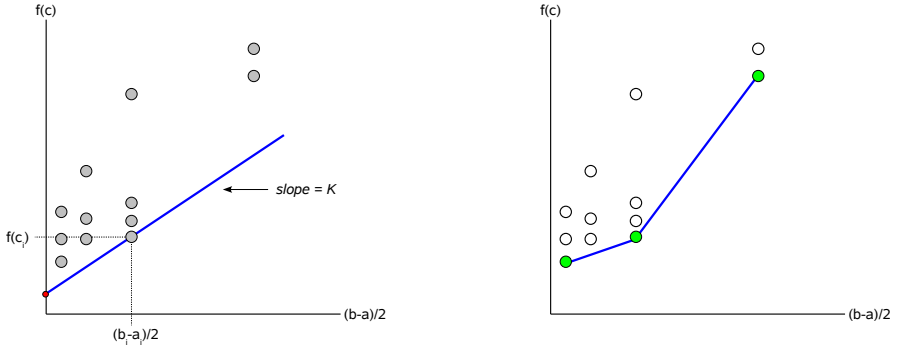


Figure A.2 DIRECT algorithm in 1D.

reason for choosing an indirect method is because the goal is to extend an *existing* algorithm so it can handle constraints. So, instead of taking the difficult route of changing the source code of the algorithm, an indirect method allows one to build the extension *around* the original code. Furthermore, it was decided to follow up DIRECT with the SQP algorithm of Matlab's fmincon solver. In other words, DIRECT is used to locate the global minimum in a possibly multimodal design space. Then, using the optimum found by DIRECT, fmincon is used to provide a more accurate solution. This will be especially useful if the solution lies on a constraint boundary as indirect methods are known to have difficulty finding these kinds of solutions. A number of penalty functions exist, like the quadratic penalty function, the logarithmic barrier function, the augmented Lagrangian function and the exact penalty function. The first three are iterative. Starting with an initial guess for the penalty parameter, at each iteration the unconstrained problem is solved, followed by increasing the penalty of constraint violations. In the context of extending a local algorithm, this is not a problem. However, when using a global algorithm, which does not make use of a starting point and does not follow a path to the optimum, it is not a good idea to restart the method each time with a different penalty value. This is the reason why the exact penalty function, shown by Eq.A.2, was chosen.



(a) (b)

Figure A.3 Choosing the next interval for sampling according to standard Lipschitzian methods (A.3(a)) and DIRECT (A.3(b)). The green points are the potentially optimal intervals.

$$\phi_1(\mathbf{x};\mu) = f(\mathbf{x}) + \frac{1}{\mu}\sum_{i\in E}|c_i(\mathbf{x})| + \frac{1}{\mu}\sum_{i\in I}[c_i(\mathbf{x})]^- \tag{A.2}$$

where $[c]^- = \max(0,c)$. For all sufficiently small, positive values of the penalty parameter μ, *one* minimization of the unconstrained function will yield the solution of the originally constrained problem. Note that on the constraint boundary, the first derivative is not defined. For this reason, the exact penalty method is not desired when the original optimizer is gradient-based. However, since DIRECT does not make use of gradients, this is not an issue.

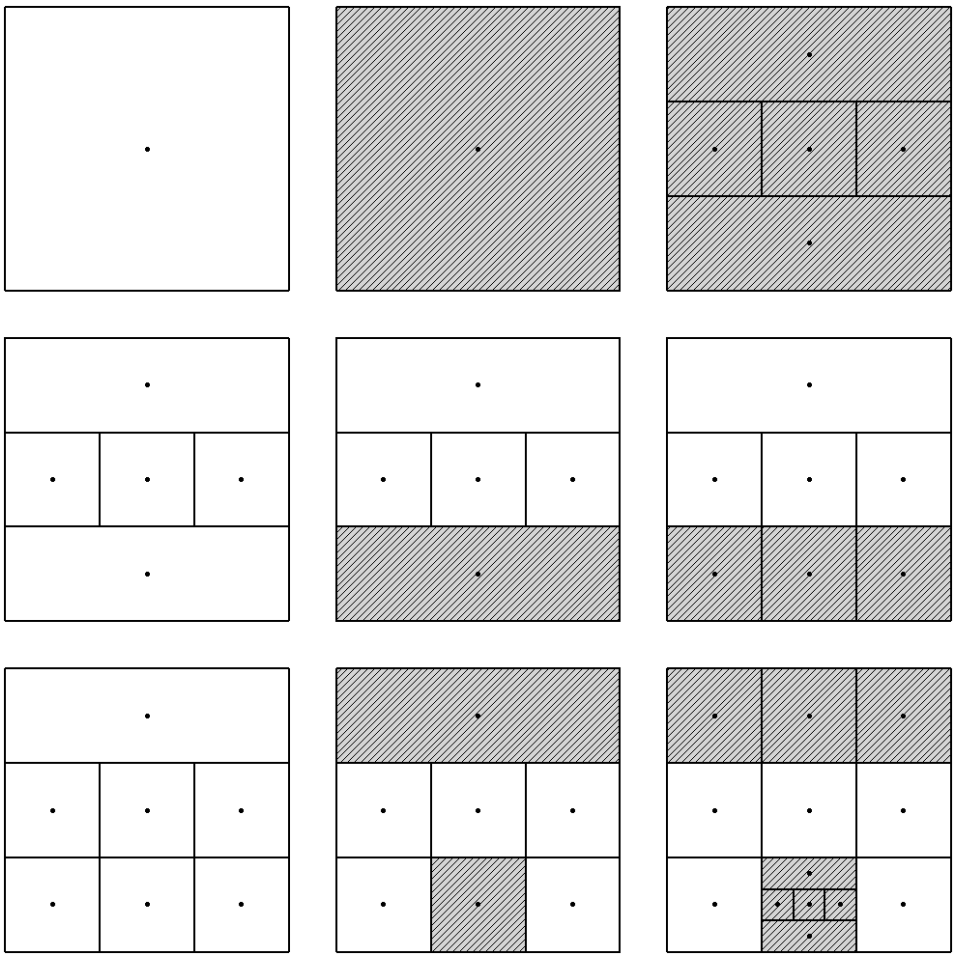


Figure A.4 DIRECT applied in two dimensions. The rows represent the iterations. The first column is the start of each iteration. In the second column, the potentially optimal rectangles are identified as illustrated in Figure A.3(b). In the third column, these rectangles are divided and sampled.

B

Detailed description of KERS

This appendix explains in detail how the program KERS works. The end of a DOE is taken as the starting point (i.e. the design points with their accompanying function values are available). Section B.1 explains the program structure and workings, while Section B.2 lists all the control settings that the program accepts. With all the features implemented in the program, the number of settings totals 37. However, they do not all have to be specified as it will depend on the mode in which the program is run. Furthermore, certain settings are dependent upon choices made earlier in the decision process. To help the user decide which settings should be specified, the reader is referred to the decision tree in Section B.3.

B.1 Structure and workings

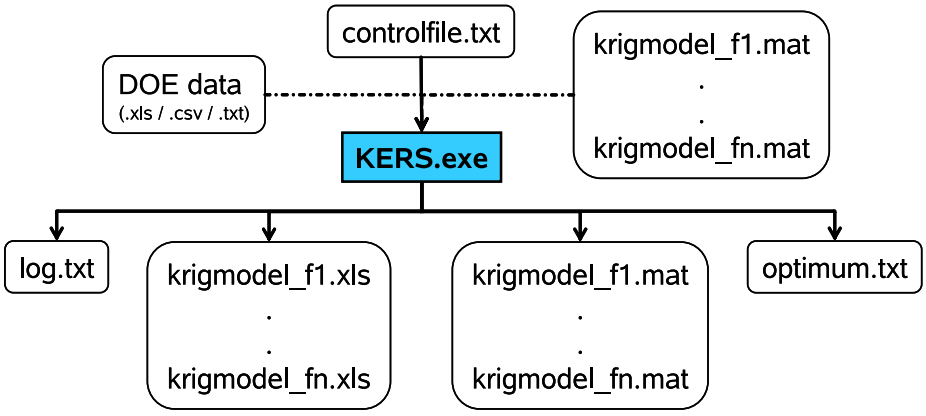


Figure B.1 Input / Output structure of *KERS*.

Figure B.1 gives a schematic of the input / output structure. To call the program and supply it with all the necessary settings, the user has to construct a control file. For a full description of all the settings, the reader is referred to Appendix B.2. If the goal is to build a Kriging model, then a file containing the DOE results has to

be supplied. It can either be an Excel, CSV or text file. For each fitted function, an Excel and Mat file will be created, where fn is the name of the n-th function fitted. Both files contain the same data but the latter can only be read by Matlab. There are two reasons why a Mat file is created too. First of all, the program can read Mat files much faster than Excel files. Secondly, it is a safety issue so the user cannot (accidently) modify the parameters of the created model when warm starting. The Excel file not only contains the model data, it also contains a macro which enables the user to evaluate the fitted functions wherever he/she wants. If the user tries to evaluate the response surface outside the bounds used to build it, the cell containing the prediction will turn orange and the specific variable which was outside the bound will turn orange too. The Excel file contains four sheets:

1. *Predict*: this is the sheet where the input values should be supplied and where the output will be returned.
2. *Summary*: this sheet contains the following information:
   - the regression term and correlation model used;
   - the variable names and the name of the function fitted;
   - the number of variables and the number of sample points used to make the fit;
   - whether the data points were interpolated, regressed or the mix option was used;
   - the model parameters;
   - in case the mix option was used, the threshold values in each dimension;
   - the validation error metrics;
   - the lower and upper bounds of the data set.
3. *Sample data*: all the originally supplied DOE data.
4. *Xtra model data*: additional data like the normalized sample data, the $\boldsymbol{\mu}$ calculated in (3.6) and the cross-validation predictions.

If the goal is optimization and the relevant functions have already been fitted, they can be supplied to the program so it only goes through the optimization stage (warm starting). In this case, no DOE data file is required. The results of the optimization, i.e. the optimum objective function, variables and constraint values are written to *optimum.txt*. Of course, if some required functions are not present in the warm start files, they will be built first before the optimization can start. In that case, a DOE data file must be supplied. All relevant operations carried out by the program are displayed in the DOS command window and written to a log file called *log.txt*.

B.2 Control file settings for KERS

These are all the settings that KERS accepts.

bound_max
The upper bound of the *optimization* problem. If a scalar value is specified, all variables will have this upper bound. To specify a vector, use ';' to separate the different entries. If nothing is specified, then the DOE data is used to determine the maximum for each variable. In other words, you should specify a data file in that case.

bound_min
The lower bound of the *optimization* problem.

bsf
The best so far value of the objective function for calculating the expected improvement (only required when the optimization strategy is *explore*).

close_wndws
When the program makes a picture of the validated response surface, the Matlab window will open, enabling the user to zoom, pan, and read the data points from the plot. This may not be desired if the program is used in a loop, as multiple windows will stay open and the command prompt will not be released until you close the windows.

Options:

- **yes**; plotting windows will automatically close
- **no**; plotting windows will stay open

If nothing is specified, then the plotting window will stay open.

correlation_model
A Kriging model is defined as the sum between a regression term and deviations (the 'correlations') from it. KERS comes with seven different correlation models:

- **exp**; exponential correlation model
- **expg**; generalized exponential correlation model

- **gauss**; Gaussian correlation model
- **lin**; linear correlation model
- **spherical**; spherical correlation model
- **cubic**; cubic correlation model
- **spline**; spline correlation model

datafile
This entry should contain the *full* name of a .txt, .csv or.xls file containing the data for building the response surface. Each column should contain data of a different parameter, with the first row specifying the parameter's name.

equality_constraints
The names of the functions that will be the equality constraints. Separate constraints with a '/'. Equality constraints were set to 0.

functions
The names of the functions for which response surfaces should be built. To define more than 1 function, use the '/' sign to separate the various function names.

globallocal_search_krig
A (positive) parameter specifying the weight between the amount of global and local search that DIRECT should carry out. In this context, DIRECT searches the optimum correlation parameters to build the Kriging surface.

A value of 1E-4 is a good value. Larger values will increase the amount of local search. Jones *et al.* [25] mention that one can vary this parameter between 1E-2 to 1E-7 without observing a significant change in algorithm performance. So, in a sense it's best to keep the value at 1E-4 and just play around with the number of iterations.

globallocal_search_opt
A (positive) parameter specifying the weight between the amount of global and local search that DIRECT should carry out. In this context, DIRECT searches the existing Kriging surfaces for the best design.

A value of 1E-4 is a good value.

goal
The program can be used for two purposes. The first is to gain an understanding of

the design problem. Using the results from a DOE, the program builds a cheap surrogate of the expensive function. As mentioned earlier, the model data is exported to an Excel file containing a macro that allows the user to evaluate the response surface at any point. The second goal that the program can achieve is solving a constrained, nonlinear optimization problem on the response surface. So, the goal specified can either be **build** or **optimize**.

hyper_ellipse
The threshold value in each variable. Data points within this so-called hyper ellipse will be regressed. If one value is specified, all variables will get the same value. If one wants to specify a different value for each variable, they have to be separated with ';'. This parameter only has to be specified if mix is used for smoothing.

inequality_constraints
The names of the functions that will be the inequality constraints. Separate constraints with a '/'. Inequality constraints are ≤ 0.

iterations_direct_krig
The number of iterations that DIRECT should use. In this context, DIRECT searches the optimum correlation parameters to build the Kriging surface.

iterations_direct_opt
The number of iterations that DIRECT should use. In this context, DIRECT searches the existing Kriging surfaces for the best design.

iterations_fmincon_krig
The maximum number of iterations that FMINCON is allowed to use. In this context, DIRECT searches the optimum correlation parameters to build the Kriging surface.

iterations_fmincon_opt
The maximum number of iterations that FMINCON is allowed to use. In this context, DIRECT searches the existing Kriging surfaces for the best design.

minmax
The objective function can either be maximized (**max**) or minimized (**min**). If nothing is specified, then by default the objective function will be minimized.

objective
The name of the function to be optimized.

outputfolder
Full name of the folder where all the results will be written to, including the log file. If nothing is specified, the current directory will be the output folder.

penalty_term_obj_opt
DIRECT handles constraints using the penalty approach. Higher values will punish constraint violations more. In this context, DIRECT searches the existing Kriging surfaces for the best design. A good value is 200.

!!!This parameter must be positive!!!

regression_term
There are three options that can be used: **0**, **1** and **2**. They represent a constant, linear and quadratic function, respectively.

sheetname
If the data file is an Excel file, then the name of the sheet containing the data should be specified here.

smooth_0
If no interpolation is used, this is the initial guess for the smoothing parameter. Larger values result in more smoothing. As opposed to theta_0 which is a vector with as many elements as there are variables, smooth_0 is a scalar.

Values between 1E-6 and 1E-1. Larger values lead to more smoothing

!!!This value must be positive!!!

smoothing
To interpolate the data points, choose **interp**. To regress the data points, choose **smooth**. The **mix** option will only regress data points within a user-defined threshold. The size of the threshold should be defined in 'hyper_ellipse'.

smooth_max
The upper bound of the smoothing parameter.

smooth_min
The lower bound of the smoothing parameter.

strategy
When searching the optimum design using the surrogate models built, two methods can be used to define an optimum:

- **exploit**; The first option is the most straightforward option. Given a response surface, the optimizer will define the optimum as the *globally best point on this particular surface* (which satisfies the constraints present, if any). This method can get stuck in a local optimum.
- **explore**; The maximum expected improvement is used as the auxiliary function to find the next iterate (EGO).

theta_0
The initial guess for the correlation parameters of the Kriging model. Each variable will have its own value. The value given here overwrites values given in a warm start. If you want to use the values from the warm start, don't specify a value for theta_0. It is possible to specify a different value for each parameter. In that case, the different entries of the vector have to be separated using a ';'. If only one value is specified, all correlation parameters will be set to this value.

Good initial guesses should be between 0.1 and 1.

!!!This value must be strictly positive!!!

theta_criterion
Kriging is also referred to as a tuned radial basis function method, i.e. the model parameters are tuned using a certain criterion. This results in a bound optimization problem. There are three options:

- **none**; in this case, no optimization will be carried out to find the optimum model parameters. The model parameters given directly by the user or through a warm start will be used to build the model. Note that warm start data is overwritten by values given directly by the user.
- **MLE**; the optimum model parameters are found using the maximum likelihood criterion. This is the widely accepted criterion in the open literature.
- **Xval**; the optimum model parameters are found using cross-validation. In this case, the user has to specify the error metric that should be used for this (check the 'Xval_error' header).

theta_max
The upper bounds of the correlation parameters. As described in 'theta_0' a vector can be specified. Note that a bound only has to be given when the theta_criterion isn't *none*.

!!! if the theta_optimizer returns a value on the upper bound, try to increase this bound, rerun the algorithm (preferably using warmstart) and check whether a better surface is produced !!!

theta_min
The lower bounds of the correlation parameters. As described in 'theta_0' a vector can be specified. Note that a bound only has to be given when the theta_criterion isn't *none*.

!!! if the theta_optimizer returns a value on the lower bound, try to decrease this bound, rerun the algorithm (preferably using warmstart) and check whether a better surface is produced !!!

theta_optimizer
The optimization algorithm to be used for finding the optimum model parameters.

Options:

- **HJ**; a local solver that doesn't use gradients
- **FMINCON**; a local solver that uses gradients
- **DIRECT**; a global solver that doesn't use gradients

All algorithms require a bound on the parameters. Additionally, the local solvers require a starting point.

validation
It is good practice to validate each response surface built before using it for visualization or optimization purposes.

Options:

- **yes**: validation will be carried out. It consists of calculating error metrics and plotting the actual data vs. the predicted data. Interpolated data points will be cross-validated using the leave-one out method.
- **no**: no validation will be carried out.

If nothing is specified, no validation will be carried out.

variables
The names of the variables that define the functions to be fit. The '/' sign should be used as the delimiter.

warmstart
Full name to a .mat file containing previously built response surfaces. When building a response surface, warm starting is handy for supplying a good initial guess for the optimum model parameters. Multiple warm start files can be given. Each file should be given in a separate row, preceded by the 'warmstart' identifier column.

Xval_error
The error metric to be used when Xval is used to find the optimum model parameters.

Options:

- **RMSE**; Root Mean Square Error
- **ARE**; Average Relative Error
- **AAE**; Average Absolute Error
- **MRE**; Maximum Relative Error
- **MAE**; Maximum Absolute Error

B.3 KERS settings decision tree

The tree in Figure B.2 is intended to help the user decide which settings to specify in the control file.

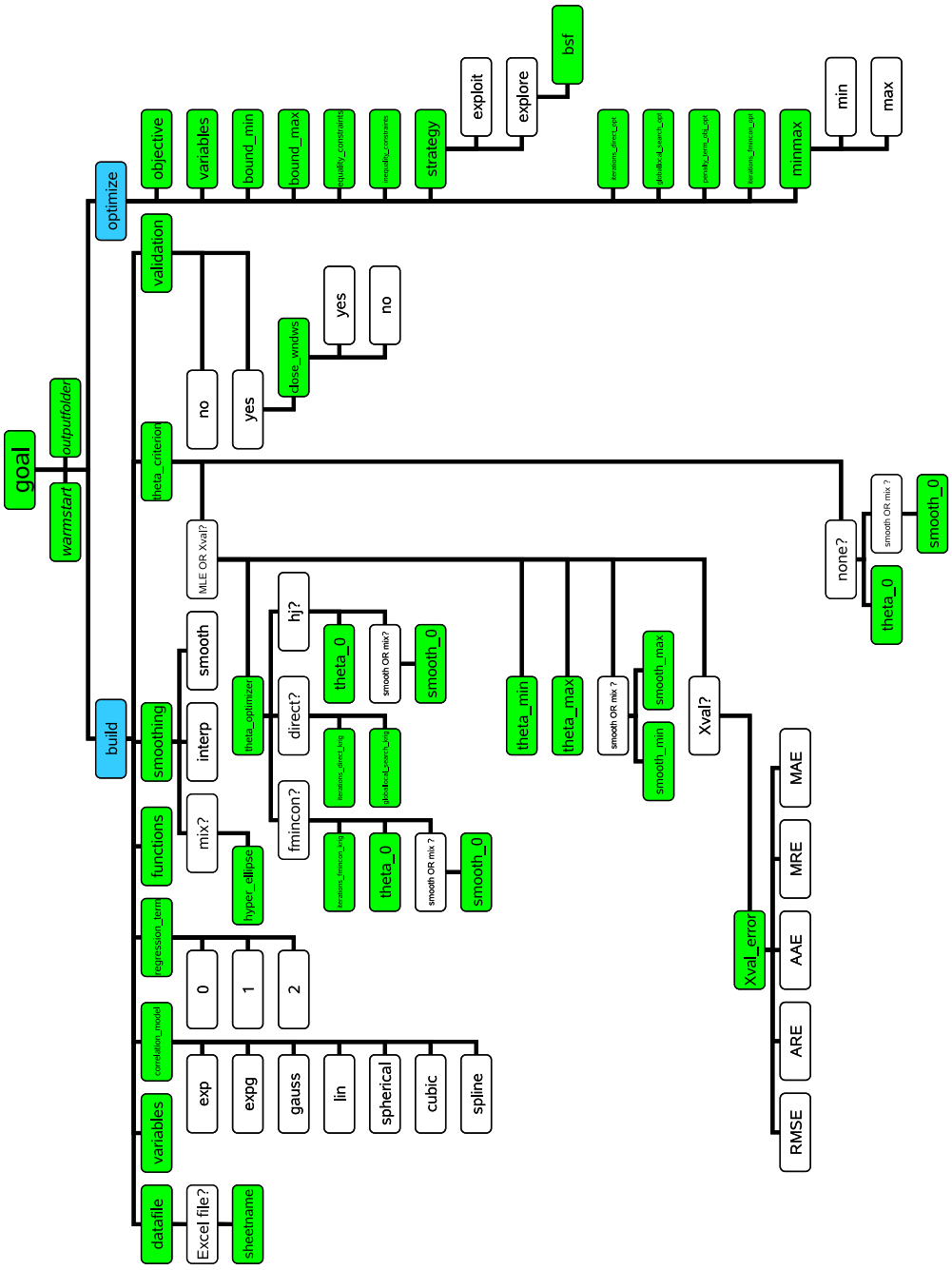


Figure B.2 KERS settings decision tree

References

[1] P.S. Beran and W.A. Silva. Reduced-order modeling - new approaches for computational physics. In *AIAA-2001-853,* 39^{th} *Aerospace Sciences Meeting and Exhibit*, Reno, NV, Jan. 8 - 11 2001.

[2] C.H. Bischof, H.M. Bucker, P. Hovland, U. Naumann, and J. Utke. Advances in automatic differentiation. *Lecture Notes in Computational Science and Engineering*, 64, 2008.

[3] M. Björkman and K. Holmström. Global optimization using the direct algorithm in matlab. *Advanced Modeling and Optimization*, 1(2):17–37, 1999.

[4] F.H. Branin. Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research and Development*, 16:504–522, 1972.

[5] C.G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90, 1970.

[6] D.D. Cox and S. John. Sdo: A statistical method for global optimization. In *Multidisciplinary Design Optimization: State of the Art*, pages 315–329. SIAM, Philadelphia, 1997.

[7] W.C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1:1–17, 1991.

[8] L. Dixon and G. Szegő. The global optimization problem: an introduction. In L. Dixon and G. Szegő, editors, *Toward Global Optimization 2*, pages 1–15. New York, 1978.

[9] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.

[10] R. Fletcher. *Practical methods of Optimization.* John Wiley & Sons, New York, 2^{nd} edition, 1987.

[11] R. Fletcher and C.M. Reeves. Function minimization by conjugate gradients. *Computer Journal*, 7:149–154, 1964.

[12] A.I.J. Forrester, A. Sobéster, and A.J. Keane. *Engineering design via surrogate modelling, a practical guide.* John Wiley & Sons Ltd., 2008.

[13] A. Genz, Z. Lin, C. Jones, D. Luo, and T. Prenzel. Fast givens goes slow in matlab. *ACM SIGNUM Newsletter*, 26:11–16, 1991.

[14] P.E. Gill, G.H. Golub, W. Murray, and M.A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 28(126):505–535, 1974.

[15] D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[16] H.M. Gutman. A radial basis function method for global optimization. *Journal of Global Optimization*, 19:201–227, 2001.

[17] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–436, 1952.

[18] W. Hock and K. Schittkowski. *Test examples for nonlinear programming codes.* Spinger-Verlag, New York, 1981.

[19] R. Hooke and T.A. Jeeves. 'direct search' solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8:212–229, 1961.

[20] A. Jameson. Optimum aerodynamic design using cfd and control theory. *AIAA Paper 95-1729-CP*, 1995.

[21] R. Jin, W. Chen, and T.W. Simpson. Comparative studies of metamodeling techniques under multiple modeling criteria. In *AIAA-2000-4801,* 8^{th} *AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA, Sept. 6 - 8 2000.

[22] R. Jin, W. Chen, and A. Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. In *ASME Design Automation Conference*, Chicago, IL, Sept. 2 - 6 2003.

[23] W.M. Jolly, J.M. Graham, A. Michaelis, R. Nemani, and S.W. Running. A flexible, integrated system for generating meteorological surfaces derived from point sources across multiple geographic scales. *Environmental Modelling & Software*, 20:873–882, 2005.

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

[25] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application*, 79(1):157–181, 1993.

[26] D.R. Jones and M. Schönlau W.J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[27] D.G. Krige. A statistical approach to some mine valuations and allied problems at the witwatersrand. Master's thesis, University of Witwatersrand, 1951.

[28] H.J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[29] L.S. Lasdon, A.D. Waren, A. Jain, and M. Ratner. Design and testing of a generalized reduced gradient code for nonlinear programming. *ACM Transactions on Mathematical Software*, 4(1):34–50, 1978.

[30] M. Locatelli. Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization*, 10:57–76, 1997.

[31] S.N. Lophaven, H.B. Nielsen, and J. Sondergaard. Aspects of the matlab toolbox dace. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2002.

[32] S.N. Lophaven, H.B. Nielsen, and J. Sondergaard. Dace - a matlab kriging toolbox. Technical report, Informatics and Mathematical Modelling, Technical University of Denmark, 2002.

[33] J. Lyness. Numerical algorithms based on the theory of complex variables. In *Proceedings of the ACM* 22^{nd} *National Conference*, pages 124–134, Washington DC, 1967.

[34] G. Matheron. The theory of regionalized variables and its applications. Technical Report 5, Paris School of Mines, 1971. Les Cahiers du Centre de Morphologie Mathematiques de Fontainebleau.

[35] D.J. Mavriplis. Discrete adjoint-based approach for optimization problems on three-dimensional unstructured meshes. *AIAA Journal*, 45(4):740–750, 2007.

[36] B. Mohammadi. A new optimal shape design procedure for inviscid and viscous turbulent flows. *International Journal for Numerical Methods in Fluids*, 25(2):183–203, 1997.

[37] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[38] E. Nielsen and W. Anderson. Aerodynamic design optimization on unstructured meshes using the navier-stokes equations. *AIAA Journal*, 37:1411–1419, 1999.

[39] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Series in Operations Research, New York, 2^{nd} edition, 2006.

[40] O. Pironneau. On optimum design in fluid mechanics. *Journal of Fluid Mechanics*, 64:97–110, 1974.

[41] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3^{rd} edition, 2007.

[42] J. Reuther, A. Jameson, A. Farmer, J. Martinelli, and D. Saunders. Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation. *AIAA Paper 96-0094*, 1996.

[43] J. Sacks, W.J. Welch, T.J. Mitchell, and H. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–423, 1989.

[44] K. Schittkowski. Nlpql: A fortran subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1985.

[45] D.F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656, 1970.

[46] J.R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, School of Computer Science, Carnegie Mellon University, 1994.

[47] B. Shubert. A sequential method seeking the global maximum of a function. *SIAM Journal on Numerical Analysis*, 9:379–388, 1972.

[48] T.W. Simpson. *A concept exploration method for product family design.* PhD thesis, Georgia Institute of Technology, 1998.

[49] S. Tiwari, P. Koch, G. Fadel, and K. Deb. Amga: An archive-based micro genetic algorithm for fast and reliable convergence. In *Genetic and Evolutionary Computation Conference GECCO*, Atlanta, Georgia, July 12 - 16 2008.

[50] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7:1–25, 1997.

[51] A. Torn and A. Žilinskas. *Global optimization.* Springer, Berlin, 1987.

[52] C.H. Tseng. Most 1.1 manual. Technical report, National Chiao Tung University, 1996.

[53] G.N. Vanderplaats. An efficient feasible directions algorithm for design synthesis. *AIAA Journal*, 22(11):1633–1640, 1984.

[54] C.K. Wikle and L.M. Berliner. A bayesian tutorial for data assimilation. *Physica D: Nonlinear Phenomena*, 230(1-2):1–16, 2007.

[55] P. Wolfe. Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235, 1969.

[56] M.H. Wright. Direct search methods: Once scorned, now respectable. In *Numerical Analysis 1995: Proceedings of the 1995 Dundee Biennial Conference in Numerical Analysis*, pages 191–208, Harlow, UK, 1996.

[57] G. Zoutendijk. *Methods of Feasible Directions.* Elsevier, Amsterdam, 2^{nd} edition, 1960.