# Engineering Design via Surrogate Modelling

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

# Engineering Design via Surrogate Modelling

## A Practical Guide

**Alexander I. J. Forrester, András Sóbester and Andy J. Keane**

*University of Southampton, UK*

WILEY

A John Wiley and Sons, Ltd., Publication

This edition first published 2008


*Registered office*
John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex,
PO19 8SQ, United Kingdom



***Library of Congress Cataloging in Publication Data***

Forrester, Alexander I. J.
Engineering design via surrogate modelling : a practical guide / Alexander I.J. Forrester, András Sóbester, and Andy J. Keane.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-06068-1 (cloth : alk. paper) 1. Engineering design—Mathematical models. 2. Engineering design—Statistical methods. I. Sóbester, András. II. Keane, A. J. III. Title.
TA174.F675 2008
620′.0042015118—dc22

2008017093

A catalogue record for this book is available from the British Library

ISBN 978-0-470-06068-1

Set in 10/12pt Times by Integra Software Services Pvt. Ltd. Pondicherry, India
Printed and bound in Great Britain by Antony Rowe Ltd, Chippenham, Wiltshire

# Contents

# Preface

Think of a well-known public personality whom you could easily identify from a photograph. Consider now whether you would still recognize them if most of the photograph was obscured, except for the corner of an eye, a small part of their chin and, perhaps, a half of their mouth. This is a game often played on television quiz shows and some contestants (and viewers at home) often display an uncanny ability to come up with the correct name after only a few small sections of the picture are revealed.

This is a demonstration of the brain's astounding ability to fill in blanks by subconsciously constructing a *surrogate model* of the full photograph, based on a few samples of it. The key to such apparently impressive feats is that we actually know a great deal about the obscured parts. We know that the photograph represents a human face, that is the image is likely to be roughly symmetrical, and we know that somewhere in the middle there must be a pattern we usually refer to as a 'nose', etc. Moreover, we know that it is a famous face. The 'search space' thus reduced, the task seems a lot easier.

The surrogate models that form the subject of this book are educated guesses as to what an engineering function might look like, based on a few points in space where we can afford to measure the function values. While these glimpses alone would not tell us much, they become very useful if we build a number of assumptions into the surrogate based on our experience of what such functions tend to look like. For example, they tend to be continuous. We may also assume that their derivatives are continuous too. With such assumptions built into the learner, the surrogate model becomes a very effective low cost replacement of the original function for a wide variety of purposes.

Surrogate modelling has had a great impact on the way the authors think about design and, after many years of combined experience in the subject, it has become a fundamental element of our engineering thought processes. We wrote this book as a means of sharing some of this experience on a practical level. While a lot has been written about the deeper theoretical aspects of surrogate modelling (indeed, references are included throughout this text to the landmarks of this literature that have informed our own thinking), what we strove to offer here is a manual for the practitioner wishing to get started quickly on solving their own engineering problems. Of course, like any sharp tool, surrogate modelling can only be used in a scientifically rigorous way if the user is constantly aware of its dangers, pitfalls, potential false promises and limitations – the present text goes to great lengths to point these out at the appropriate times.

To emphasize the practical dimension of this guide, we accompany it with our own *MATLAB*® implementation of the techniques described therein. Snippets of this code are included in the text wherever we felt that, through the 'maths-like' and compact nature of *MATLAB*, they contribute to the explanations. These, as well as all the rest of the code, can be found on the book website at `www.wiley.com/go/forrester`. Template scripts are also provided, ready for the user to replace our objective function modules with his or her own. It is worth noting here that our own example functions, while mostly representing 'real life' engineering problems, were designed for easy experimentation; that is they take only fractions of a second to run. We expect, however, that most of the applications the codes will be used for 'in anger' will be several orders of magnitude more time-consuming.

This is a self-contained text, though we assumed a basic familiarity with calculus, linear algebra and probability. Additional 'mathematical notes' are included wherever we had to refer to more advanced topics within these subjects. We therefore hope that this book will be useful to graduate students, researchers and professional engineers alike.

While numerous colleagues have assisted us in the writing of this volume, a few names stand out in particular. We would like to thank Prasanth Nair and David Toal of the University of Southampton, Max Morris of Iowa State University, Donald Jones of the General Motors Co., Natalia Alexandrov of NASA, Tom Etheridge of Astrium, Lucian Tudose of the Technical University of Cluj Napoca, Danie G. Krige and Stephen J. Leary of BAE Systems for their suggestions and for reading various versions of this manuscript.

Finally, a disclaimer. Surrogate modelling is a vast subject and this text does not claim nor, indeed, can hope to cover it all. The selection of techniques we have chosen to include reflect, to some extent, our personal biases. In other words, this is the combination of tools that works for us and we earnestly hope that it will for the reader too.

**Alexander Forrester, András Sóbester and Andy Keane**
Southampton, UK

## *Disclaimer*

# About the Authors

Dr Alexander I. J. Forrester is Lecturer in Engineering Design at the University of Southampton. His main area of research focuses on improving the efficiency with which expensive analysis (particularly computational fluid dynamics) is used in design. His techniques have been applied to wing aerodynamics, satellite structures, sports equipment design and Formula One.

Dr András Sóbester is a Lecturer and EPSRC/ Royal Academy of Engineering Research Fellow in the School of Engineering Sciences at the University of Southampton. His research interests include aircraft design, aerodynamic shape parameterization and optimization, as well as engineering design technology in general.

Professor Andy J. Keane currently holds the Chair of Computational Engineering at the University of Southampton. He leads the University's Computational Engineering and Design Research Group and directs the Rolls-Royce University Technology Centre for Computational Engineering. His interests lie primarily in the aerospace sciences, with a focus on the design of aerospace systems using computational methods. He has published over two hundred papers and three books in this area, many of which deal with surrogate modelling concepts.

# Foreword

Over the last two decades, there has been an explosion in the ability of engineers to build finite-element models to simulate how a complex product will perform. In the automotive industry, for example, we can now simulate the injury level of passengers in a crash, the vibration and noise experienced when driving on different road surfaces, and the vehicle's life when subjected to repeated stressful conditions such as pot holes. Moreover, our ability to quickly modify these simulation models to reflect design changes has greatly increased. The net result is that the potential for using optimization to improve an engineering design is now higher than ever before.

One of the major obstacles to the use of optimization, however, is the long running time of the simulations (often overnight) and the lack of gradient information in some of the most complicated simulations (especially crashworthiness). Due to the long running times and the lack of analytic gradients, almost any optimization algorithm applied directly to the simulation will be slow.

Despite this slowness, one could still bite the bullet and invest one's computational budget in applying an optimization algorithm directly to the simulations. But this is unlikely to be satisfying, because rarely does a single optimization result settle any design issue. For example, if the result is *not* satisfactory, one may want to gain insight into what is going on by performing parameter sweeps and plotting input-output relationships. Or one might want to repeat the optimization with a modified formulation (different starting point, different constraints). All this, of course, requires doing more simulations. On the other hand, if the result *is* satisfactory, one still want might to do further investigations to see if a better tradeoff can be struck between competing objectives. Again, this requires more simulations. Clearly, if one uses up all the available resources solving the first optimization problem, all these follow-up studies would not be possible, or at least lead to missed deadlines.

The basic idea in the 'surrogate model' approach is to avoid the temptation to invest one's computational budget in answering the question at hand and, instead, invest in developing fast mathematical approximations to the long running computer codes. Given these approximations, many questions can be posed and answered, many graphs can be made, many tradeoffs explored, and many insights gained. One can then return to the long running computer code to test the ideas so generated and, if necessary, update the approximations and iterate.

While the basic idea of the surrogate model approach sounds simple, the devil is in the details. What points do you sample to use in building the approximation? What approximation method do you employ? How do you use the approximation to suggest new, improved designs? How do you use the approximations to explore tradeoffs between objectives? What do you do if your simulation has numerical noise in it? And, equally important: Where do I get the computer code to do all these things?

In *Engineering Design via Surrogate Modelling: A Practical Guide*, the authors answer all of these questions. They are like cooks giving you a recipe for an entire meal: appetizer, salad, entrée, wine, dessert, and coffee. It is not an isolated recipe for bread rolls such as you might find in the cooking section of the Sunday paper. The authors start at the very beginning, with variable screening to determine which variables to include in the study. One then learns how to develop a sampling plan for developing the initial approximations. Several approximation methods are then discussed, but the authors' preference for Kriging is clear. They then show how to use Kriging approximations to do unconstrained optimization, constrained optimization, and tradeoff studies. At each step, sample code is provided in *MATLAB*, which is also available in electronic form on an associated website.

No different than any cook, the authors have their biases: they like particular ways of sampling, particular ways to use Kriging for optimization, etc. To their credit, however, in several sections the authors go out of their way to mention other possible approaches and to provide references for you to follow up if you are interested.

In my view, the book can appeal to two audiences. For those experienced in the field of surrogate models, the book provides a glimpse at what the authors, as experienced practitioners, consider to be the state of the art. For those just beginning, the book provides a self-contained introduction to the field.

Like any cookbook, the book is a place to start, not to finish. I suspect that people reading this book will take some recipes as they are, will modify others to suit their taste, and will ignore still others in favor of their own recipes. But I am convinced that even the most experienced persons in the field will find new things that pique their interest (this was certainly true for myself). So, to all those beginning this book, may I say, *Bon Appetit*!

Donald R. Jones
General Motors Co.

# Prologue

Engineering design is concerned with the making of decisions based on analysis, which directly impact the product or service being designed. To accomplish this, engineers typically engage in a great deal of analysis to understand the background to their decisions. It is often necessary for months of analysis by dedicated teams to be undertaken to inform key product decisions. It is against this backdrop that the current book has been written. For example, in modern aerospace design offices the computational power needed to support advanced decision making can be prodigious and, even with the latest and most powerful computers, designers still wish for greater understanding than can be gained by straightforward use of the familiar analysis tools, such as those coming from the fields of computational fluid dynamics or computational structural mechanics.

One way of gaining this desirable increased insight into the problems being studied is via the use of surrogate (or meta) models. Such models seek to provide answers in the gaps between the necessarily limited analysis runs that can be afforded with the available computing power. They can also be used to bridge between various levels of sophistication afforded by varying fidelity physics based simulation codes, or between predictions and experiments. Their role is to aid understanding and decision taking by wringing every last drop of information from the analysis and data sources available to the design team and making it available in a useful and powerful way. This book aims to discuss the application of such surrogate models using some of the most recent results stemming from the academic and industrial research communities. To place these ideas in context we begin with a (far from exhaustive) summary of where surrogate models typically find use in engineering design.

The simplest, and currently most common, use of surrogate models is to augment the results coming from a single, expensive simulation code that needs to be run for a range of possible inputs dictated by some design strategy (perhaps a planned series of runs or those suggested by some search process). The basic idea is for the surrogate to act as a ‘curve fit’ to the available data so that results may be predicted without recourse to use of the primary source (the expensive simulation code). The approach is based on the assumption that, once built, the surrogate will be many orders of magnitude faster than the primary source while still being usefully accurate when predicting away from known data points. Note that there

are two key requirements here: (1) a significant speed increase in use and (2) useful accuracy. Clearly, these factors are often in tension with each other and the user will often have to balance these competing needs carefully.

Another increasingly common use for surrogates is to act as calibration mechanisms for predictive codes of limited accuracy. It is quite common when producing a software model of some physical process to have to simplify the approach taken so as to gain acceptable run times. For example, in computational fluid dynamics there are a whole raft of different simulation approaches that run from simple but very rapid potential flow solvers, through Euler codes to Reynolds averaged Navier–Stokes methods to large eddy simulations and on to direct numerical simulation of the full equations. A surrogate may well be trained to bridge between two such codes by being set up to represent the differences between a simple but somewhat inaccurate code and a more accurate but slower approach, the idea being to gain the accuracy of the expensive code without the full expense. Such 'multi-fidelity' or 'multi-level' approaches can be extended to dealing with data coming from physical experiments and their correlation with computational predictions – indeed, much early work in this field stems from long term agricultural experiments where data coming from crop trials had to be interpreted.

A third use of surrogate models is to deal with noisy or missing data. It is a commonplace experience that results coming from physical experiments are subject to small random errors. These need to be dealt with when the data are used, often by some process of averaging. It will also often occur in physical experimentation that some experiments fail to yield usable results at all. It is less well known that the results of computational codes also suffer from such problems, though in this case any noise is generally not random. Computational 'noise' stems from the schemes used to set up computational models, notably discretized and iterative approaches where solutions are not fully independent of the discretization or the number of iterations used. Similarly, most numerical schemes are rarely completely foolproof and will sometimes fail in unexpected ways. In these circumstances surrogate models can be used as filters and fillers to smooth data, revealing overall trends free of extraneous fine detail and spanning any gaps.

Finally, surrogate models may be used in a form of data mining where the aim is to gain insight into the functional relationships between variables open to the design team and results of interest. If appropriate methods are selected and applied to sets of data, surrogates can be used to demonstrate which variables have most impact and what the forms of such effects appear to be. This can allow engineers to focus on those quantities that have most importance and also to understand such quantities with greater clarity. Sometimes such understanding comes directly from the equations resulting from surrogate construction; alternatively surrogates may be used in visualization schemes to map and graph different projections of the data more rapidly than would be possible by repeated runs of the available analysis codes.

In all the above cases the basic steps of the surrogate modelling process remain essentially the same, and are illustrated in the flowchart in Figure P.1, where each stage is related to the chapter that describes it.

Firstly, some form of data set relating a series of inputs and outputs is obtained, typically by *sampling* the design decision space, making use of the available, and often expensive, analysis codes. In other words, a number of possible candidate designs are generated and analysed, using whatever computational or experimental means are at hand.

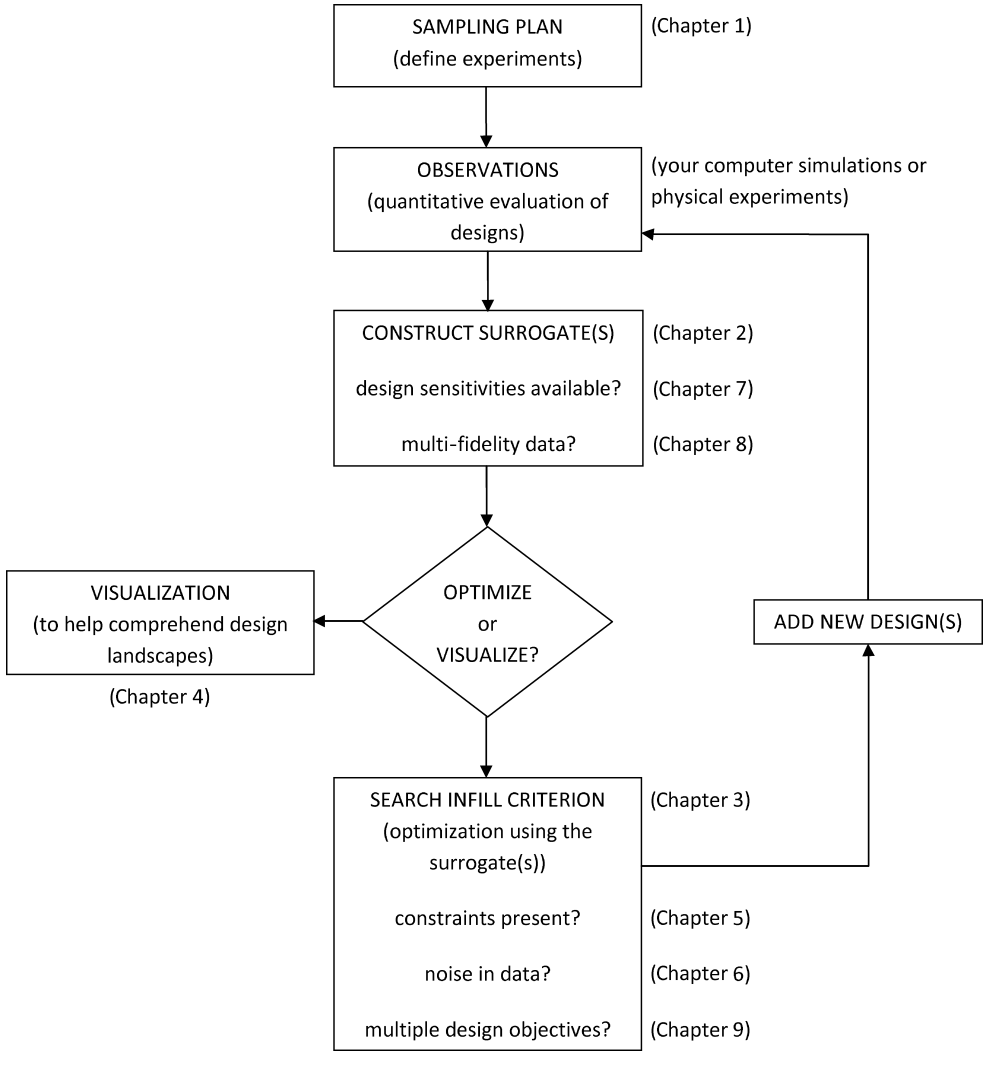



**Figure P.1.** The surrogate modelling process

Following this, a suitable surrogate model form must be selected and fitted to the available data – this process lies at the heart of this book. Its parameters must be estimated, it must be assessed for accuracy and a number of schemes can be used to do this. Note a key limitation of the surrogate approach at this point: if the problem being dealt with has many dimensions the number of points needed to give reasonably uniform coverage rises exponentially – the so-called *curse of dimensionality*. Currently the only way around this problem is either to limit the ranges of the variables so that the shape being modelled by the surrogate is sufficiently simple to be approximated from very sparse data or, alternatively, to freeze many of the design values at hopefully sensible values and work with just a few at a time, iterating

around those being made active as the design process progresses (for example, in aircraft design, dealing with aerodynamic quantities at one stage with structural variables fixed and then swapping these around).

Since the initial design selections made to produce the first set of data will almost inevitably miss certain features of the landscape, the construction of a useful surrogate often requires further, judiciously selected calls to the analysis codes. These additional calls are termed *infill points* and the process of applying them is known as *updating*. The selection of new points is usually made either in areas where the surrogate is thought to be inaccurate or, alternatively, where the surrogate model suggests that particularly interesting combinations of design variables lie. The selection of such points is often made using an optimization-based search over the surrogate. The updating of the surrogate with infill points may be carried out a number of times until the surrogate is fit for purpose (or perhaps the available budget of computing effort has been exhausted).

Having constructed (and hopefully tested) a suitably accurate model, it is then finally exploited or explored in some fashion, perhaps being embedded in a modified solver or as a subject for use along with optimization or visualization tools. The processes of exploration, exploitation and updating may well be closely interlinked so that the surrogate remains usefully accurate as the design process evolves. Moreover, data coming from previous design processes may well also be melded into the system if appropriate.

It turns out that it is rare for a completely fixed approach to be appropriate in all cases of interest, since the data itself may well influence the directions taken. This will call for knowledge, care and experience from those constructing and using the surrogates – hopefully the following sections and chapters will help support this process. A good understanding of the capabilities and limitations of the various techniques presented will be the hallmark of the knowledgeable designer.

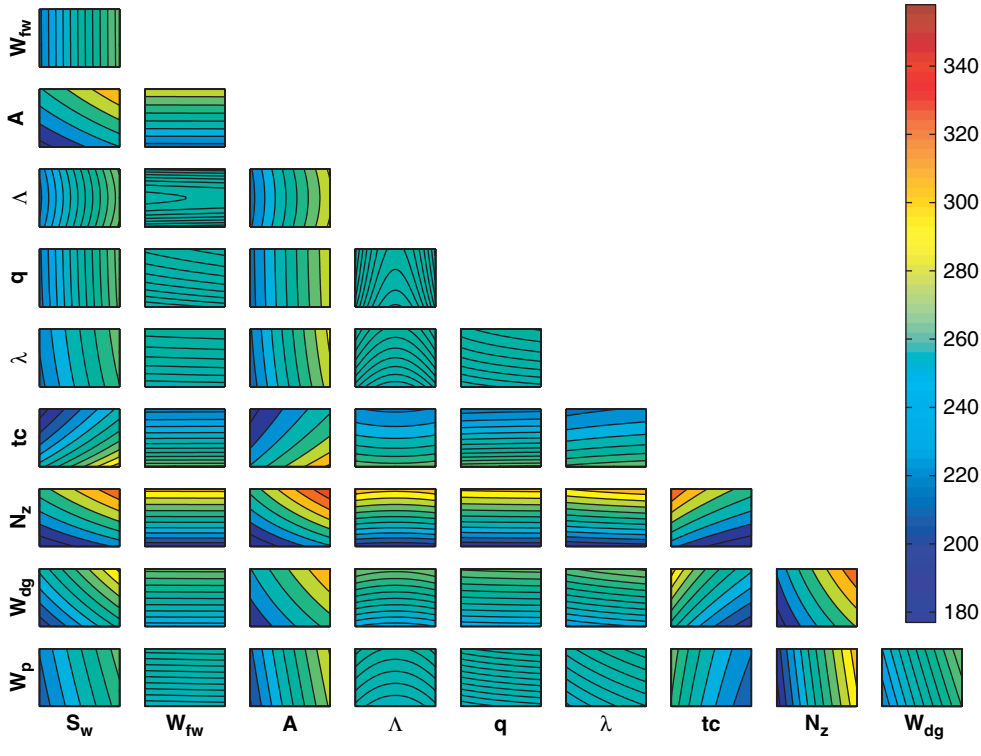



**Plate I.** (colour version of Figure 1.1) Light aircraft wing weight (W) landscape. Each tile shows a contour of the weight function (Equation (1.4)) versus two of the ten variables, with the remaining eight variables held at the baseline value.

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

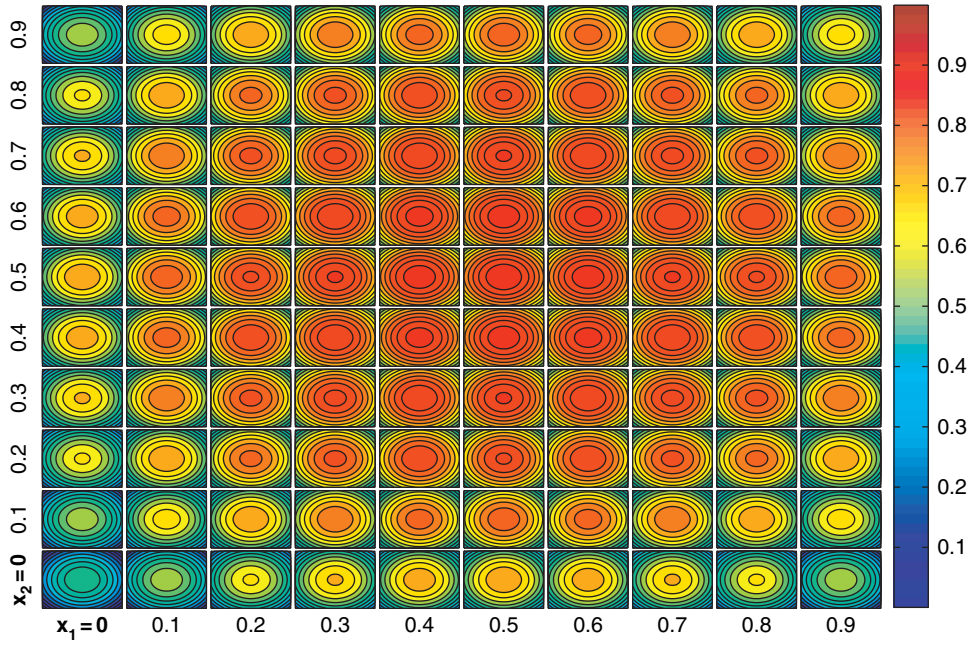



**Plate II.** (colour version of Figure 4.2) Four-variable nested plot of the surrogate of the function $f(\mathrm{x}) = 1/4 \sum_{i=1}^{4} 1 - (2x_i - 1)^2$, $\mathrm{x} \in [0,1]^4$, generated using `nested4.m`. Here $x_3$ varies along the horizontal axis of each tile, $x_4$ along the vertical axes, while the values of $x_1$ and $x_2$ can be read off the bottom of each column of tiles and the beginning of each row respectively.

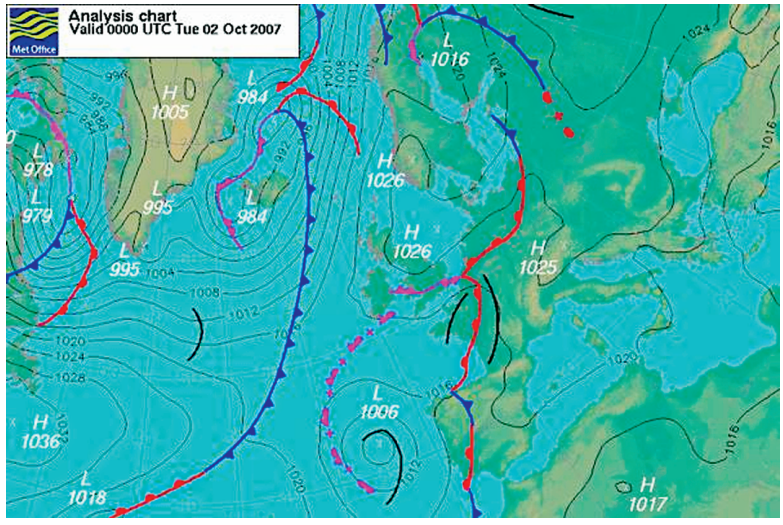



**Plate III.** (colour version of Figure 4.1) A flat depiction of two functions of the same two variables – a colour-coded topographic map (of height above mean sea level) and labelled isobars representing surface pressure (This is Crown copyright material which is reproduced with the permission of the Controller of HMSO and the Queen's Printer for Scotland).

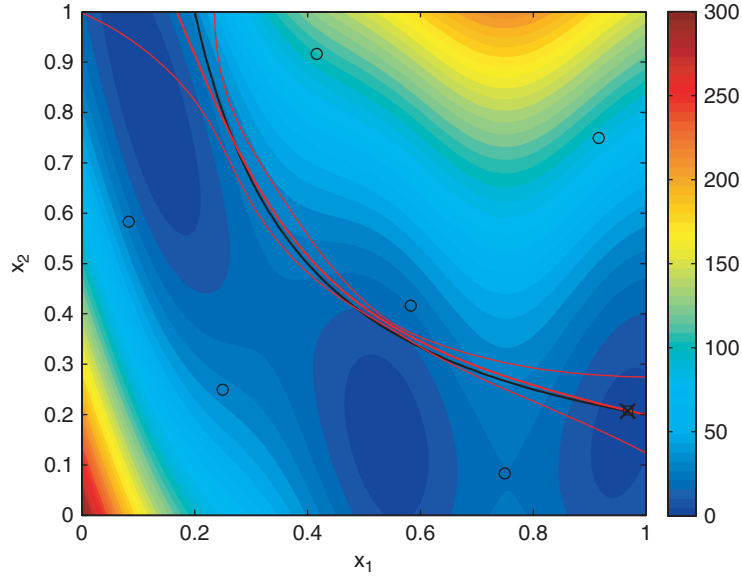



**Plate IV.** (colour version of Figure 5.3) Contours of the Branin function, the product constraint level curve (black) and Kriging estimate (bold red), together with ± one standard error (fine red). The figure also shows the location of the data points used to sample the constraint (circles), the minimum of the function, subject to the Kriging model of the constraint (square) and the true optimum (cross).

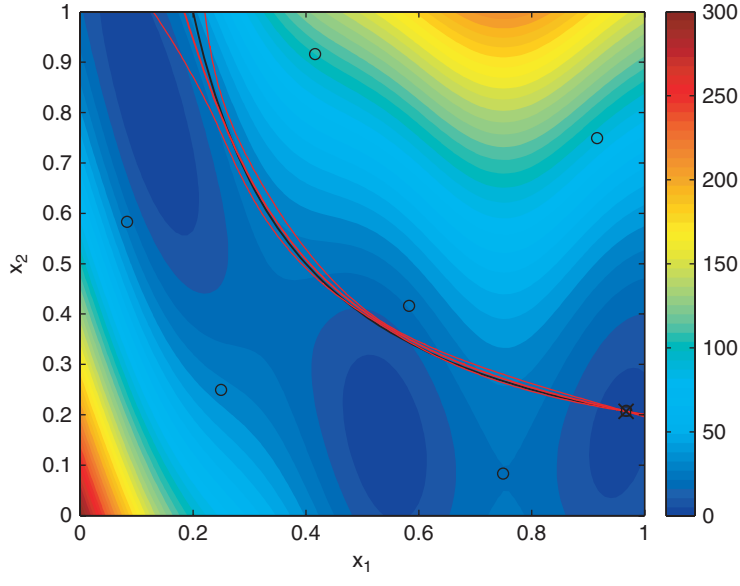



**Plate V.** (colour version of Figure 5.4) Contours of the Branin function and the product constraint level curve and Kriging estimate after one infill point. Key as per Plate IV.

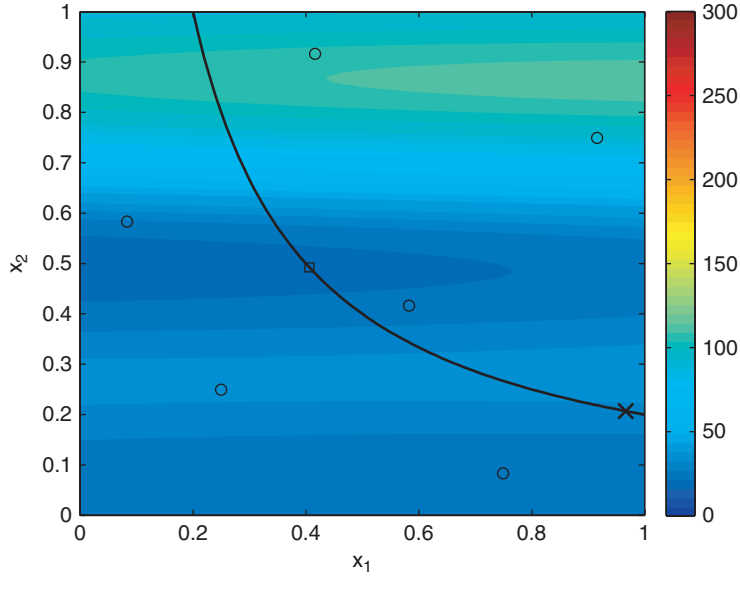



**Plate VI.** (colour version of Figure 5.5) Contours of a Kriging prediction of the Branin function, showing the true product constraint level curve, sample points (circles), the optimum of the Kriging prediction subject to the constraint (square) and the true constrained optimum (cross).

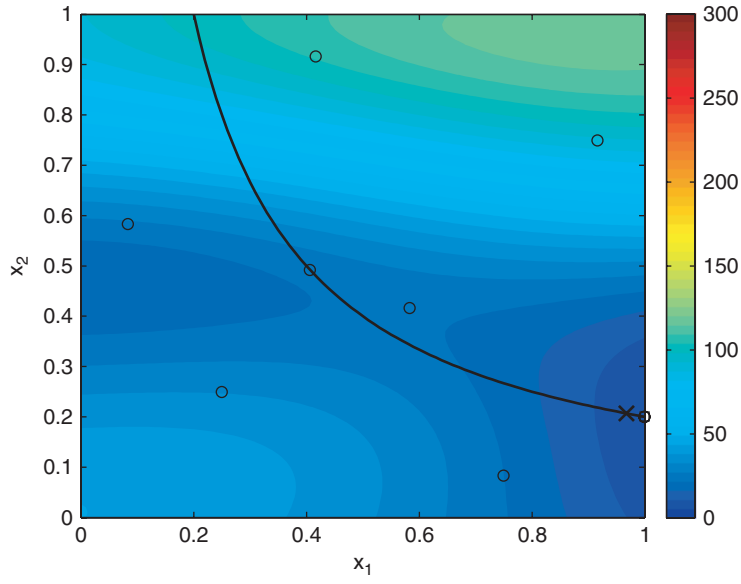



**Plate VII.** (colour version of Figure 5.6) Contours of the Kriging prediction of the Branin function and the product constraint level after five infill points.

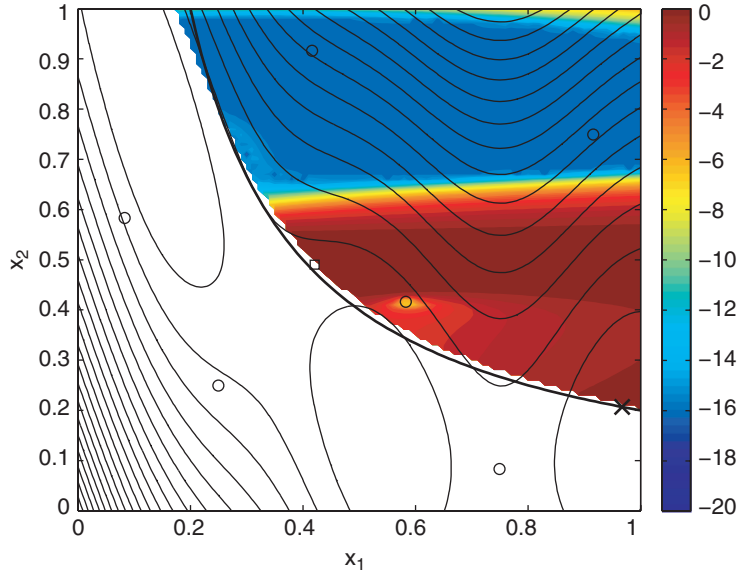



**Plate VIII.** (colour version of Figure 5.7) The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of predicted constraint satisfaction (filled contours) shown together with contours of the true Branin function and the sample points. The true constraint limit is also shown (bold contour), along with the locations of the maximum $E[I(\mathbf{x})]$ (with penalty applied, square) and the actual optimum (cross). The irregular contours at low $\log(E[I(\mathbf{x})])$ are where the $E[I(\mathbf{x})]$ calculation is encountering problems with floating point underflow. See the mathematical note in Section 6.2.1 for more information and a way to avoid this.

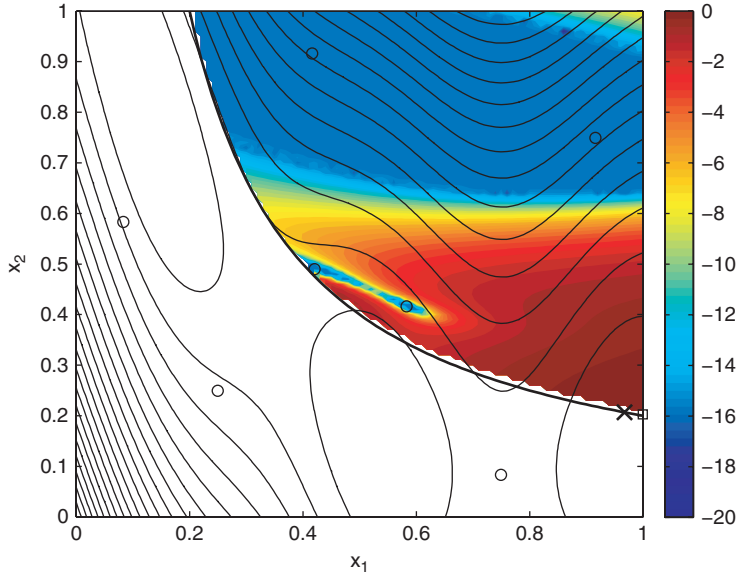



**Plate IX.** (colour version of Figure 5.8) The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of predicted constraint satisfaction after one infill point (key as Plate VIII). The maximum $E[I(\mathbf{x})]$ (with penalty applied) has located the region of the global optimum.

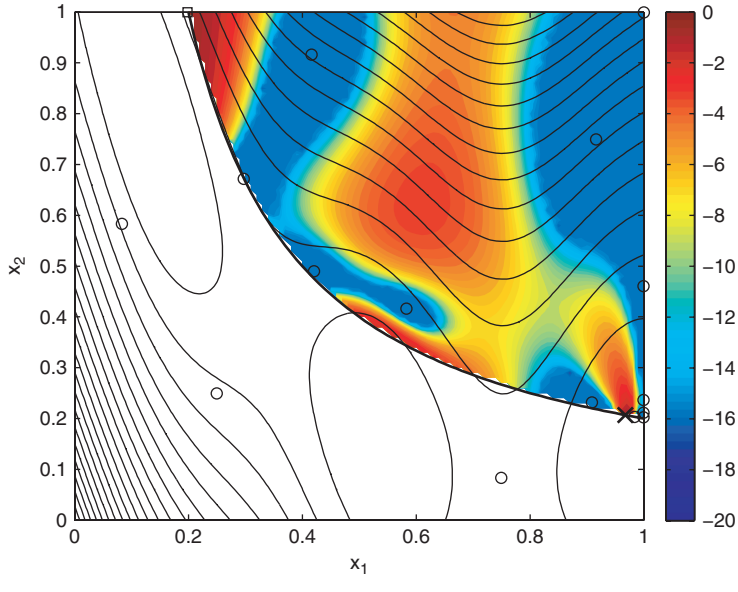



**Plate X.** (colour version of Figure 5.9) The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of constraint satisfaction after nine infill points (key as Plate VIII).

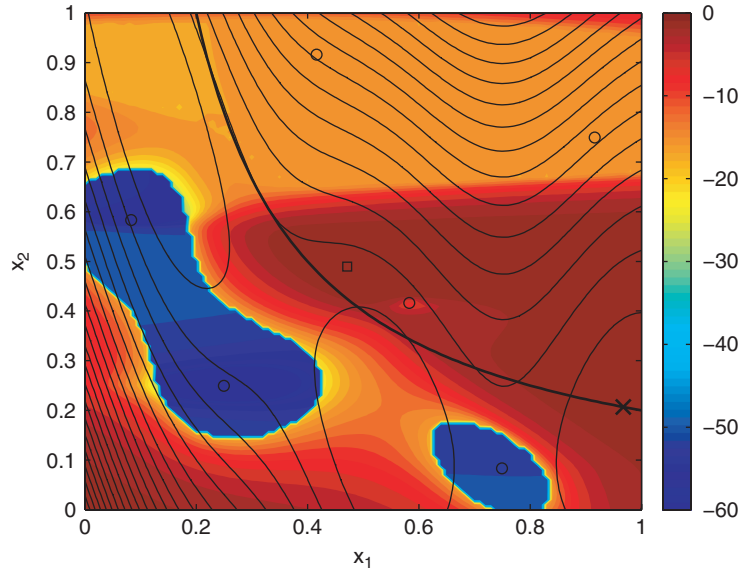



**Plate XI.** (colour version of Figure 5.10) The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ based on the initial sample of six points (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross).

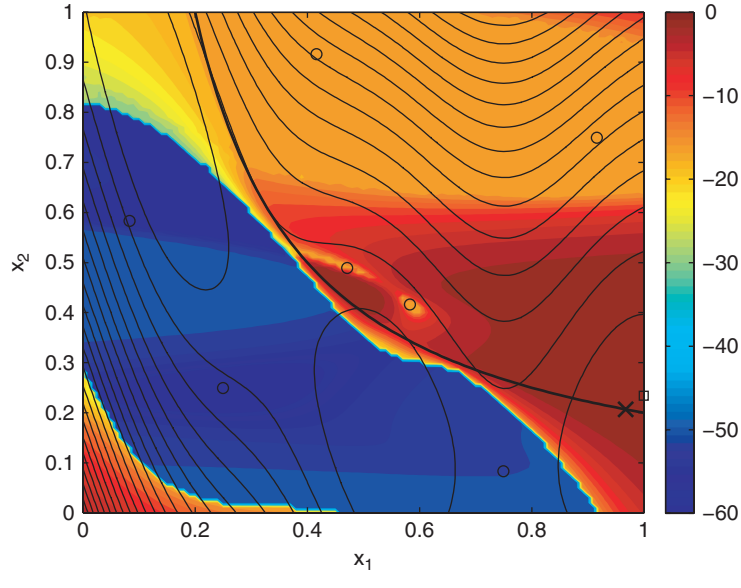



**Plate XII.** (colour version of Figure 5.11) The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ after one infill point (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross).

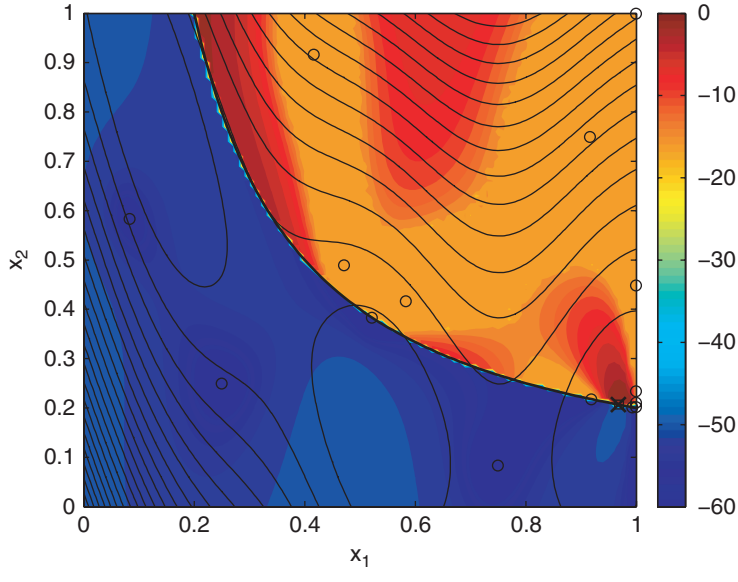



**Plate XIII.** (colour version of Figure 5.12) The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ after nine infill points (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross).

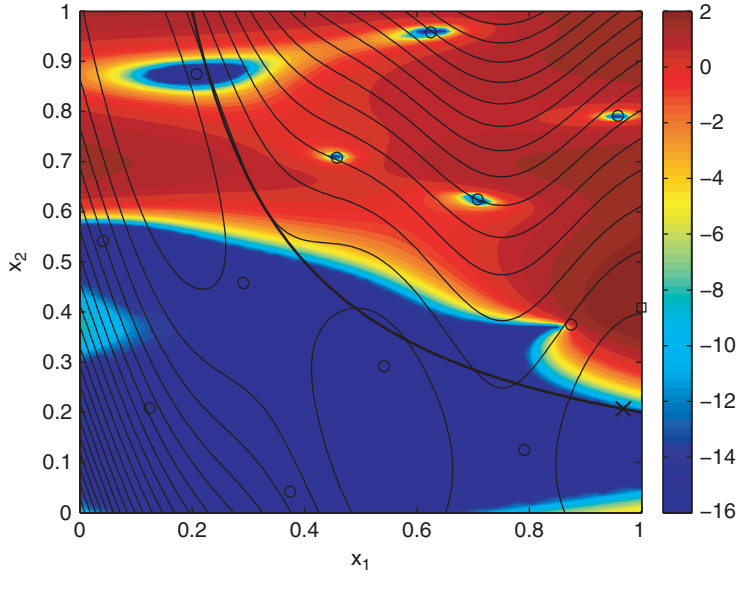



**Plate XIV.** (colour version of Figure 5.13) The $\log(E[I(\mathbf{x})])$ of a Kriging model through the combined successful and imputed data set (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross).

# Part I
# Fundamentals

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

# 1

# Sampling Plans

Engineering design problems requiring the construction of a cheap-to-evaluate 'surrogate' model $\widehat{f}$ that emulates the expensive response of some black box $f$ come in a variety of forms, but they can generally be distilled down to the following template.

Here $f(\mathbf{x})$ is some continuous quality, cost or performance metric of a product or process defined by a $k$-vector of design variables $\mathbf{x} \in D \subset \mathbb{R}^k$. In what follows we shall refer to $D$ as the *design space* or *design domain*. Beyond the assumption of continuity, the only insight we can gain into $f$ is through discrete *observations* or *samples* $\left\{\mathbf{x}^{(i)} \rightarrow y^{(i)} = f(\mathbf{x}^{(i)}) | i = 1, \ldots, n\right\}$. These are expensive to obtain and therefore must be used sparingly. The task is to use this sparse set of samples to construct an approximation $\widehat{f}$, which can then be used to make a cheap performance prediction for any design $\mathbf{x} \in D$.

Much of this book is made up of recipes for constructing $\widehat{f}$, given a set of samples. Excepting a few pathological cases, the mathematical formulations of these modelling approaches are well-posed, regardless of how the *sampling plan* $\mathbf{X} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\right\}$ determines the spatial arrangement of the observations we have built them upon. Some models do require a minimum number $n$ of data points but, once we have passed this threshold, we can use them to build an unequivocally defined surrogate.

However, a well-posed model does not necessarily *generalize* well, that is it may still be poor at predicting unseen data, and this feature *does* depend on the sampling plan $\mathbf{X}$. For example, measuring the performance of a design at the extreme values of its parameters may leave a great deal of interesting behaviour undiscovered, say, in the centre of the design space. Equally, spraying points liberally in certain parts of the inside of the domain, forcing the surrogate model to make far-reaching extrapolations elsewhere, may lead us to (false) global conclusions based on patchy, local knowledge of the objective landscape.

Of course, we do not always have a choice in the matter. We may be using data obtained by someone else for some other purpose or the available observations may come from a variety of external sources and we may not be able to add to them. The latter situation often occurs in conceptual design, where we wish to fit a model to performance data relating to existing, similar products. If the reader is only ever concerned with this type of modelling problem, he or she may skip the remainder of this chapter. However, if you have the possibility of

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

selecting your own objective function sampling locations, please read on, as in what follows we discuss a number of systematic techniques for building sampling plans that will enable the surrogate model to be built subsequently to generalize well.

## 1.1 The 'Curse of Dimensionality' and How to Avoid It

It is intuitively obvious that the higher the number of design variables in a modelling problem, the more objective function measuring locations we need if we are to build a reasonably accurate predictor. What is more striking is just *how many more*: if a certain level of prediction accuracy is achieved by sampling a one-variable space in $n$ locations, to achieve the same sample density in a $k$-dimensional space, $n^k$ observations are required. To get a better feel for why this is often referred to as the *curse of dimensionality*, consider the following example.

Let us imagine that we would like to model the cost of a car tyre and we have a complex computational tyre design software that, given a set of geometrical variables, can, through a range of simulations, design a tyre and plan a manufacturing process for it, the latter model resulting in a cost estimate. For the sake of this example, let us assume that the analysis and design process takes one hour of computation per design. If we need a model of wheel diameter versus cost and have a computational budget of, say, ten hours, we can thus compute ten cost values at diameter values ranging from the smallest to the largest car in the manufacturer's range. Ten simulations should give us a reasonably accurate predictor, even considering that the response can be highly nonlinear (for example due to different types of tools being needed for different sizes, nonlinearity of performance requirements, etc.). What happens, however, if we decide to refine the model by including other variables, say, tread width, groove spacing, sidewall height, flexing area thickness, shoulder thickness, bead seat diameter and liner thickness? We now have eight parameters, which, assuming the same sampling density as on the wheel diameter, means that our computational budget requirement jumps to $10^8$ runs. This will take almost 11 416 years!

There are two important conclusions here. Firstly, evaluating the objective function for every possible combination of every possible design variable value can become a very expensive undertaking. Statisticians refer to this type of scenario as a *full factorial experiment*.

The second conclusion we can draw is that the number of design variables has a massive impact on the number of experiments required. It is therefore imperative that we minimize this at the outset. The question is, how can we tell which variables can be left out of a design study, that is, which variables do not have a significant effect on the objective function? More to the point, how can we answer the above question with a minimum number of runs of the (usually expensive) simulation? We will discuss this shortly, but first we need to make a few general points about physical and computational experiments, the two sources that may be used to obtain the objective function.

## 1.2 Physical versus Computational Experiments

The results of physical experiments are almost always subject to experimental error. These departures from the 'true' result come from three main sources:

- **human error**, that is error introduced simply by the experimenter making a mistake;
- **systematic error**, due to a flaw in the philosophy of the experiment that adds a consistent bias to the result;
- **random error**, which is due to measurement inaccuracies inherent to the instruments being used.

The key concept that differentiates between the last two items in this list is *repeatability*. If there is a systematic component in the experimental error, this will have the same value each time we repeat the experiment. The random error, however, will be different every time and, given enough experiments, it will take both positive and negative values.

Computational experiments are also subject to experimental error, resulting from:

- **human error**, 'bugs' in the analysis code, incorrectly entered boundary conditions in the solution of a partial differential equation, etc., and
- **systematic error**. For example, an inviscid mathematical model of the viscous flow around a body (an approximation sometimes made for computing time saving purposes) will consistently underestimate the drag forces acting on the body. Another example is the error caused by the inherently finite resolution of the numerical modelling process (e.g. errors caused by insufficient mesh resolution in a finite element solve). While this type of error can lead to underestimates or overestimates, it will do so in *exactly the same way* if we repeat the experiment.

The difference, therefore, compared to physical experiments is that computational experiments are not affected by random error – they are *deterministic*.

We dwell on this seemingly academic point here for three good reasons. Firstly, it is germane to a question of terminology. Physical experimentalists often use the term 'noise', referring to the *random error* that corrupts their experiments. Somewhat confusingly, though, 'noise' often crops up in the computational experiments literature as well, referring to *systematic error* (hardly ever stating this explicitly, but it must do, as computers do not make random errors!). This is not an especially pernicious usage, as long as both author and reader understand what it refers to. To that end, we shall, throughout this book, differentiate between the two meanings by putting a pair of inverted commas around 'noise' when it refers to the systematic errors of computer experiments and leaving them out when we are talking about the random noise of physical experiments or about both types.

Beyond the semantics, it will also be important for the reader to be aware of the differences between the various types of error when, later on, we tackle Gaussian process based approximation techniques. The reason is that the statistical apparatus behind these methods requires a fictional 'physicalization' of computer experiments: we will view the outputs (results) of computer experiments, known to be *deterministic* values, as *realizations of a stochastic process*. This is merely to facilitate the mathematical process and one of the purposes of this section is to dispel, well in advance, any confusion this artifice may cause.

The final reason for insisting on the differences between the two types of experiments is to explain why their respective experimental design techniques are so different. A vast literature has been written on devising screening strategies and sampling plans for physical experiments, which are aimed, among other things, at mitigating the effects of the random error that affects the responses. In principle, this is done by replicating experiments – a pointless exercise, of course, when a deterministic computer code provides the data upon which the approximation will be built.

## 1.3 Designing Preliminary Experiments (Screening)

We saw earlier just how important it is to minimize the number of design variables $x_1,\ x_2, \ldots,\ x_k$ before we attempt to model the objective function $f$. But how do we achieve this screening, as we shall call this process in what follows, without compromising the relevance of the analysis?

If $f$ is at least once differentiable over the design domain $D$ with respect to each $x$, $\partial f/\partial x_i|_{\mathbf{x}}$ is a useful criterion for establishing a taxonomy of design variables. Namely:

- if $\partial f/\partial x_i|_{\mathbf{x}} = 0, \forall \mathbf{x} \in D$, the variable $x_i$ can safely be neglected,
- if $\partial f/\partial x_i|_{\mathbf{x}} = \text{constant} \neq 0, \forall \mathbf{x} \in D$, the effect of the variable $x_i$ is linear and additive,
- if $\partial f/\partial x_i|_{\mathbf{x}} = g(x_i), \forall \mathbf{x} \in D$, where $g(x_i) \neq \text{constant}$, $f$ is nonlinear in $x_i$,
- if $\partial f/\partial x_i|_{\mathbf{x}} = g(x_i), \forall \mathbf{x} \in D$, where $g(x_i, x_j, \ldots) \neq \text{constant}$, $f$ is nonlinear in $x_i$ and involved in interactions with $x_j, \ldots$,

The above classification is merely a statement of terminology, as in practice we have no way of measuring $\partial f/\partial x_i|_{\mathbf{x}}$ across the entire design space. Even a reasonable estimate is a tall order considering that the budget available for the screening study is generally very limited. Incidentally, there is no hard and fast rule as to what percentage of the available time should be spent on screening the variables, as this is largely problem-dependent. If we expect many variables to be inactive, a thorough screening study has the potential of enhancing the accuracy of the subsequent model considerably (due to its reduced dimensionality). If, however, there is an (engineering) reason to believe that most variables have a considerable impact on the objective, it is advisable to focus efforts on the modelling itself.

A great deal has been written about sampling plans and modelling methods specifically aimed at input variable screening (Jones *et al.* (1998)). Their working principles vary according to the assumptions they make about the objective function and the variables. Here we concentrate on an algorithm described by Morris (1991) because the only assumption it makes is that the objective function is deterministic (a feature shared by most computational models).

### *1.3.1 Estimating the Distribution of Elementary Effects*

In order to simplify the presentation of what follows we make, without loss of generality, the assumption that the design space $D = [0, 1]^k$; that is we normalize all variable into the unit cube. We shall adhere to this convention for the rest of the book and we strongly urge the reader to do likewise in implementing any algorithms described here, as, in addition to yielding clearer mathematics in some cases, this step safeguards against scaling issues.

Before proceeding with the description of the Morris algorithm we need to define an important statistical concept. Let us restrict our design space $D$ to a $k$-dimensional, $p$-level full factorial grid, that is $x_i \in \{0, 1/(p-1), 2/(p-1), \ldots, 1\}$, for $i = 1, \ldots, k$. For a given baseline value $\mathbf{x} \in D$, let $d_i(\mathbf{x})$ denote the *elementary effect* of $x_i$, where

$$d_i(\mathbf{x}) = \frac{y(x_1, x_2, \ldots, x_{i-1}, x_i + \Delta, x_{i+1}, \ldots, x_k) - y(\mathbf{x})}{\Delta}, \tag{1.1}$$

where $\Delta = \xi/(p-1)$, $\xi \in \mathbb{N}^*$ and $\mathbf{x} \in D$ such that its components $x_i \leq 1 - \Delta$.

Morris's method aims to estimate the parameters of the distribution of elementary effects associated with each variable, the principle being that a large measure of central tendency indicates a variable with an important influence on the objective function across the design space and a large measure of spread indicates a variable involved in interactions and/or in terms of which $f$ is nonlinear. In practice, we estimate the sample mean and the sample standard deviation of a set of $d_i(\mathbf{x})$ values calculated in different parts of the design space.

Clearly, it is desirable to generate the preliminary sampling plan $\mathbf{X}$ in such a way that each evaluation of the objective function $f$ will participate in the calculation of two elementary effects (instead of just one, if we were to pick, naively, a random spread of baseline $\mathbf{x}$'s and then to add $\Delta$ to one of their variables). Also, the sampling plan should give us a certain number (say, $r$) elementary effects for each variable, independently drawn with replacement. The reader interested in a thorough discussion of how to obtain such an $\mathbf{X}$ is invited to read Morris's original paper (Morris, 1991) – here we limit ourselves to a description of the process itself.

Let $\mathbf{B}$ denote a $k+1 \times k$ *sampling matrix* of 0s and 1s with the property that for every column $i = 1, 2, \ldots, k$ there are two rows of $\mathbf{B}$ that differ only in their $i$th entries (we shall give an example of such a matrix in the *MATLAB*® implementation of the method). We then compute a *random orientation* of $\mathbf{B}$, denoted by $\mathbf{B}^*$:

$$\mathbf{B}^* = \left(\mathbf{1}_{k+1,1}\mathbf{x}^* + (\Delta/2)\left[(2\mathbf{B} - \mathbf{1}_{k+1,k})\mathbf{D}^* + \mathbf{1}_{k+1,k}\right]\right)\mathbf{P}^*, \tag{1.2}$$

where $\mathbf{D}^*$ is a $k$-dimensional diagonal matrix, where each element on the diagonal is either $+1$ or $-1$ with equal probability, $\mathbf{1}$ is a matrix of 1s, $\mathbf{x}^*$ is a randomly chosen point in our discretized, $p$-level design space (limited around the edges by $\Delta$, as discussed previously) and $\mathbf{P}^*$ is a $k \times k$ random permutation matrix in which each column contains one element equal to 1 and all others equal to 0 and no two columns have 1s in the same position. Here is a *MATLAB* implementation of Equation (1.2):

```
function Bstar=randorient (k, p, xi)
% Generates a random orientation for a screening matrix
%
% Inputs:
%       k – number of design variables
%       p – number of discrete levels along each dimension
%       xi – elementary effect step length factor
%
% Output:
%       Bstar – random orientation matrix

% Step length
Delta=xi/(p-1);
m=k+1;

% A truncated p – level grid in one dimension
xs=(0:1/(p-1):1-Delta);
xsl=length(xs);
```

(*continued*)

```
% Basic sampling matrix
B=[zeros(1,k); tril(ones(k))];

% Randomization

% Matrix with +1s and -1s on the diagonal with equal probability
Dstar=diag(2*round(rand(1,k))-1);

% Random base value
xstar=xs(floor(rand(1,k)*xsl)+1);

% Permutation matrix
Pstar=zeros(k);
rp=randperm(k);
for i=1:k, Pstar(i,rp(i))=1; end

% A random orientation of the sampling matrix
Bstar=(ones(m,1)*xstar+(Delta/2)*...
    ((2*B-ones(m,k))*Dstar+ones(m,k)))*Pstar;
```

To obtain $r$ elementary effects for each variable, the screening plan is built from $r$ random orientations:

$$\mathbf{X} = \begin{bmatrix} \mathbf{B}_1^* \\ \mathbf{B}_2^* \\ \dots \\ \mathbf{B}_r^* \end{bmatrix}, \tag{1.3}$$

or in *MATLAB*:

```
function X=screeningplan(k, p, xi, r)
% Generates a Morris screening plan with a specified number of
% elementary effects for each variable.
%
% Inputs:
%       k – number of design variables
%       p – number of discrete levels along each dimension
%       xi – elementary effect step length factor
%       r – number of random orientations (=number of elementary
%           effects per variable).
%
% Output:
%       X – screening plan built within a [0,1]^k box

X=[];
for i=1:r
   X=[X; randorient(k,p,xi)];
end
```

We then compute the value of $f$ for each row of $\mathbf{X}$; in what follows we shall store these objective function values in the $r(k+1) \times 1$ column vector $\mathbf{t}$. Taking one random orientation at a time, the adjacent rows of the screening plan and the corresponding function values from $\mathbf{t}$ can be inserted into Equation (1.1) to obtain $k$ elementary effects (one for each variable).

Once a sample of $r$ elementary effects has been collected for each variable, the sample means and sample standard deviations of these can be computed and represented on the same chart for comparison purposes. Here is how `screening plot.m` accomplishes this:

```
function screeningplot(X, Objhandle, Range, xi, p, Labels)
% Generates a variable elementary effect screening plot
%
% Inputs:
%       X – screening plan built within a [0,1]^k box (e.g. with
%          screeningplan.m)
%       Objhandle – name of the objective function
%       Range – 2xk matrix (k – number of design variables) of lower
%              bounds (first row) and upper bounds (second row) on
%              each variable.
%       xi – elementary effect step length factor
%       p – number of discrete levels along each dimension
%       Labels – 1xk cell array containing the names of the variables

k=size(X,2);
r=size(X,1)/k-1;

for i=1:size(X,1)
   X(i,:)=Range(1,:)+X(i,:).*(Range(2,:)-Range(1,:));
   t(i)=feval(Objhandle,X(i,:));
end

for i=1:r
   for j=(i-1)*(k+1)+1:(i-1)*(k+1)+k
      F(find(X(j,:)-X(j+1,:)~=0),i)=(t(j+1)-t(j))/(xi/(p-1));
   end
end

% Compute statistical measures
for i=1:k
   ssd(i)=std(F(i,:));
   sm(i)=mean(F(i,:));
end

figure, hold on

for i=1:k
   text(sm(i),ssd(i),Labels(i),'FontSize',25)
end

axis([min(sm) max(sm) min(ssd) max(ssd)]);
xlabel('Sample␣means')
ylabel('Sample␣standard␣deviations')
```

Before illustrating the process by means of an engineering design example, it is worth mentioning two scenarios, where the deployment of the algorithm described above requires special care.

Firstly, if the dimensionality $k$ of the space is relatively low and we can afford a large $r$, one should keep in mind the increased probability of the same design cropping up twice in $\mathbf{X}$. If the responses at the sample points are deterministic, there is, of course, no point in repeating the evaluation. This issue does not occur especially often, as large numbers of elementary effects are generally needed when screening spaces with *high* dimensionalities.

Secondly, numerical simulation codes sometimes fail to return a sensible (or, indeed, any) result, due to meshing errors, the failure of a partial differential equation solution process to converge, etc. From a screening point of view this is significant because an entire random orientation $\mathbf{B}^*$ is compromised if the objective function computation fails for one of the points therein.

### A ten-variable weight function

Let us consider the following analytical expression (implemented in `liftsurfw.m`) used as a conceptual level estimate[1] of the weight of a light aircraft wing:

$$W = 0.036 S_w^{0.758} W_{fw}^{0.0035} \left(\frac{A}{\cos^2 \Lambda}\right)^{0.6} q^{0.006} \lambda^{0.04} \left(\frac{100tc}{\cos \Lambda}\right)^{-0.3} (N_z W_{dg})^{0.49} + S_w W_p. \qquad (1.4)$$

Table 1.1 contains a nomenclature of the symbols used in Equation (1.4), as well as a baseline set of values, roughly representative of a Cessna C172 Skyhawk aircraft and a somewhat arbitrarily chosen range for each variable. These baseline values and the ranges were used to generate a filled contour plot of the weight function (see Figure 1.1) by varying the inputs pairwise and keeping the remaining variables at the baseline value.

**Table 1.1.** Nomenclature of the ten-variable screening example problem

| Symbol | Parameter | Baseline | Minimum value | Maximum value |
|---|---|---|---|---|
| $S_w$ | Wing area ($ft^2$) | 174 | 150 | 200 |
| $W_{fw}$ | Weight of fuel in the wing (lb) | 252 | 220 | 300 |
| $A$ | Aspect ratio | 7.52 | 6 | 10 |
| $\Lambda$ | Quarter-chord sweep (deg) | 0 | –10 | 10 |
| $q$ | Dynamic pressure at cruise ($lb/ft^2$) | 34 | 16 | 45 |
| $\lambda$ | Taper ratio | 0.672 | 0.5 | 1 |
| $tc$ | Aerofoil thickness to chord ratio | 0.12 | 0.08 | 0.18 |
| $N_z$ | Ultimate load factor | 3.8 | 2.5 | 6 |
| $W_{dg}$ | Flight design gross weight (lb) | 2000 | 1700 | 2500 |
| $W_p$ | Paint weight ($lb/ft^2$) | 0.064 | 0.025 | 0.08 |

[1] Such equations are generally derived by fitting curves to existing aircraft data. This particular formula has been adapted from Raymer's excellent aircraft conceptual design text (Raymer, 2006).

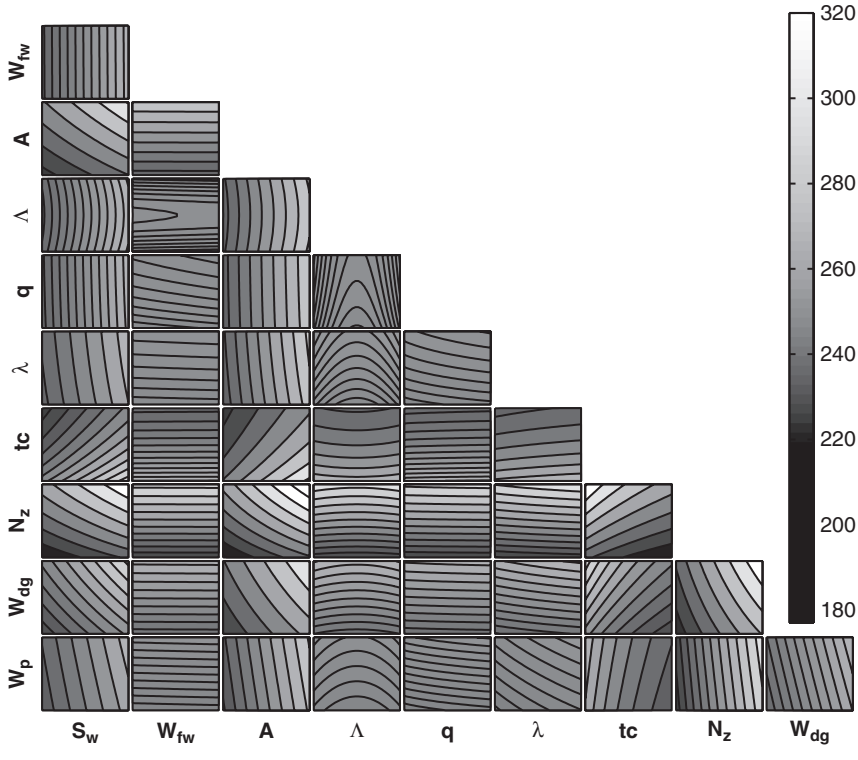



**Figure 1.1.** Light aircraft wing weight (W) landscape. Each tile shows a contour of the weight function (Equation (1.4)) versus two of the ten variables, with the remaining eight variables held at the baseline value (See Plate I for colour version).

So what does the plot reveal from the point of view of variable activity? As expected, for example, the weight per unit surface area of the paint ($W_p$) does not make much of an impact on the shape of the surface, whereas the load factor $N_z$ (which determines the magnitude of the maximum aerodynamic load on the wing) is clearly very active and it is involved in interactions with other variables. A classic example is the interaction with the aspect ratio $A$: the red area in the top right-hand corner of the weight versus $A$ and $N_z$ indicates a heavy wing for high aspect ratios and large $g$-forces (this is the reason why highly manoeuvrable fighter jets cannot have very efficient, glider-like wings).

Of interest to us here, however, is how much of all this would we have guessed simply from a cheap screening study, without an understanding of the engineering significance of the variables involved (which is quite often the case in engineering design) and without the ability to compute such a tile plot (which is almost always the case in engineering design – after all, if the objective $f$ was so cheap to compute, we would not be thinking about surrogate modelling anyway).

So what does Figure 1.2, depicting the results of an $r = 5$ screening study, reveal? The first observation we can make is that there is a clearly defined group of variables clustered around the origin – recall that a small measure of central tendency is a feature of inputs with little impact on the objective function. Indeed, we find the weight of the paint here, as

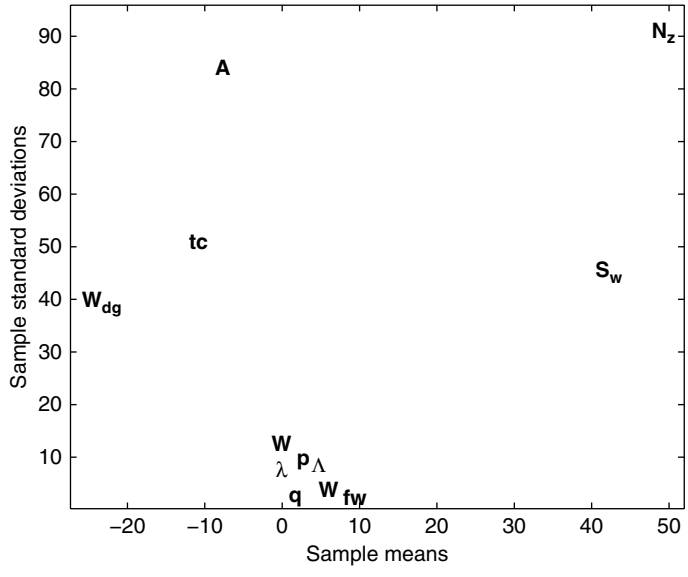



**Figure 1.2.** Estimated means and standard deviations of the elementary effect distributions of each of the 10 variables of the wing weight example.

expected, as well as the dynamic pressure (meaning that it does not make a big difference in wing weight where we are in our chosen range of dynamic pressures – with the cruising speed specified, this can be viewed in terms of a cruising altitude range). The same reasoning applies (and is confirmed by Plate I) to the taper ratio and the quarter-chord sweep.[2]

Although still close to the zero mean, the variable with the largest central tendency within this group is the fuel weight $W_{fw}$. Its sample of elementary effects has a very low standard deviation and a mean slightly larger than the rest of the group, indicating that it is more important than them but is not involved in interactions. The plot indicates *A* and *tc* having similar importance, but having a nonlinear/interactive effect (as seen from their high standard deviation values).

Finally and unsurprisingly, a large (absolute) central tendency and large measure of spread point to $W_{dg}$, $S_w$ and $N_z$ having the most significant impact on wing weight. Of course, aircraft designers know that the overall weight of the aircraft and the wing area must be kept to a minimum (the latter usually dictated by constraints such as required stall speed, landing distance, turn rate, etc.) and that a requirement for high $N_z$ will translate into a need for sturdy, heavy wings. In fact, this is precisely why we have used such a well-understood function here to illustrate the workings of the screening algorithm. Screening will be done in anger, however, when no such prior knowledge is available and the identification of the important variables can merely rely on the objective function as a black box.

The script `wing.m` will run the example discussed above and will produce Plate I and a scatter plot similar to Figure 1.2 (*MATLAB* will generate a slightly different screening plan each time, as this comprises random orientations of the sampling matrix **B**).

[2] Large variations in the sweep angle would make a significant difference. Here, however, we are looking at a small range of values (−10 to +10 degrees) typical of light, low speed aircraft.

We shall return briefly to the issue of establishing the order of importance (or activity) of the inputs of the objective function in the section about Kriging models. For now, let us look at the next step of the modelling process. With the active variables identified (either through engineering judgement or through a systematic screening study) we can now design the main sampling plan in the space defined by these variables. This will form the basis of the data that the model of the objective will be built upon.[3]

## 1.4 Designing a Sampling Plan

### *1.4.1 Stratification*

A feature shared by all of the approximation models discussed in this book is that they are more accurate in the vicinity of the points where we have evaluated the objective function. In later chapters we will delve into the laws that quantify our decaying trust in the model as we move away from a known, sampled point, but for the purposes of the present discussion we shall merely draw the intuitive conclusion that a uniform level of model accuracy throughout the design space requires a uniform spread of points. A sampling plan possessing this feature is said to be *space-filling*.

The most straightforward way of sampling a design space in a uniform fashion is by means of a rectangular grid of points. This is the full factorial sampling technique referred to in the section about the curse of dimensionality.

Here is the simplified version of a *MATLAB* function that will sample the unit hypercube at all levels in all dimensions, with the $k$-vector $q$ containing the number of points required along each dimension. The variable `Edges` specifies whether we want the points to be equally spaced from edge to edge (`Edges=1`) or we want them to be in the centres of $n = q_1 \times q_2 \times \cdots \times q_k$ bins filling the unit hypercube (for any other value of `Edges`).

```
function X=fullfactorial(q, Edges)
% Generates a full factorial sampling plan in the unit cube
%
% Inputs:
%       q – k – vector containing the number of points along each
%           dimension
%       Edges – if Edges=1 the points will be equally spaced from
%               edge to edge (default), otherwise they will be in
%               the centres of n=q(1)* q(2)*... q(k) bins filling
%               the unit cube.
%
```

(*continued*)

[3] There may be some cases where the runs performed for screening purposes may be recycled at the actual model fitting step, in particular when the objective is very expensive. For example, if some of the variables turn out not to have any impact at all, these runs can form part of the reduced dimensionality sampling plan. Of course, life is rarely as black and white as this and the ignored variables will have had *some* effect. In this case, a judgement will have to be made regarding the trade-off between losing some expensive runs versus introducing some additional noise into the model fitting data.

```
% Output:
%       X – full factorial sampling plan

if nargin < 2, Edges = 1; end

if min(q) < 2
       error('You must have at least two points per dimension.');
end

% Total number of points in the sampling plan
n=prod(q);

% Number of dimensions
k=length(q);

%Pre–allocate memory for the sampling plan
X=zeros(n,k);

%Additional phantom element
q(k+1)=1;

for j=1:k
         if Edges==1
                 one_d_slice=(0:1/(q(j)-1):1);
         else
                 one_d_slice=(1/q(j)/2:1/q(j):1);
         end

         column=[];
         while length(column) <n
                 for l=1:q(j)
                        column=[column; ones (prod(q(j+1:k)),1)*···
                                one_d_slice(l)]
                 end
         end
         X(:,j)=column;
end
```

For example, `fullfactorial([3 4 5],1)` will produce the three-dimensional sampling plan shown in Figure 1.3. Clearly, such a design will satisfy the uniformity criterion, but it has two major flaws.

Firstly, it is only defined for designs of certain sizes, those that can be written as products of the numbers of levels for each dimension, that is $n = q_1 \times q_2 \times \cdots \times q_k$.[4] Secondly, when projected on to the axes, sets of points will overlap and it can be argued that the sampling

[4] Random sampling plans provide an alternative, which clearly can be created for any number of design points and they do make sense in some cases. For example, if we sample the design variables according to some distribution, we can assess how the objective values are distributed (Santner *et al*., 2003), a scenario typical of *robustness studies*. In most cases, however, the spread of a random plan can be quite uneven and therefore not space-filling (especially for small values of $n$).

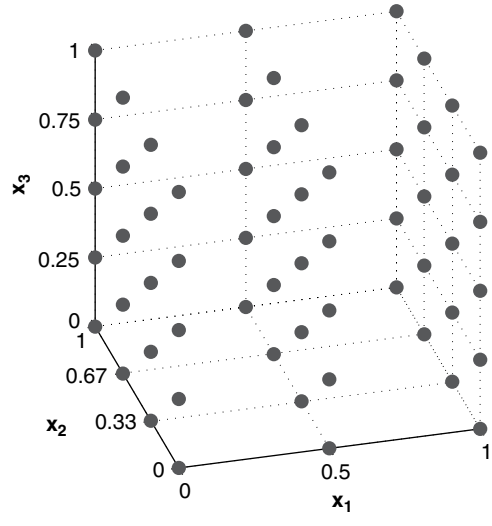



**Figure 1.3.** Example of a three-dimensional full factorial sampling plan.

of any individual variable could be improved by making sure that these projections are as uniform as possible.

This can be done by splitting the range of that variable into a relatively large number of equal sized bins and generating equal sized random subsamples within these bins. This approach is known as *stratified random sampling*. A natural development of this idea is to generate a sampling plan that is stratified on all of its dimensions. The technique commonly used to achieve this is known as *Latin hypercube sampling*.

## *1.4.2 Latin Squares and Random Latin Hypercubes*

As we have seen, stratification of sampling plans aims to generate points whose projections onto the variable axes are uniform. Before we look at generic techniques for building such plans, it is worth considering the case of discrete valued variables in two dimensions. Such uniform projection plans can be generated quite readily: if $n$ designs are required, an $n \times n$ square is built by filling every column and every line with a permutation of $\{1, 2, \ldots, n\}$, that is every number must only appear once in every column and every line. For example, for $n = 4$, a *Latin square* (for this is what such plans are usually known as) might look like this:

| | | | |
|---|---|---|---|
| 2 | **1** | 3 | 4 |
| 3 | 2 | 4 | **1** |
| **1** | 4 | 2 | 3 |
| 4 | 3 | **1** | 2 |

We have highlighted the **1**'s to illustrate the point about the uniform projection idea but, of course, we could have chosen 2, 3 or 4 just as well. Also, this is just one, arbitrarily chosen four-point Latin square – we could equally have picked any of the other 575 possible arrangements. Incidentally, the number of distinct Latin squares increases rather sharply with $n$; for example, there are 108 776 032 459 082 956 800 Latin squares of order eight! (It is left as an exercise for the reader to check this.)

Building a *Latin hypercube*, that is the multidimensional extension of this, can be done in a similar way, by splitting the design space into equal sized hypercubes (bins) and placing points in the bins (one in each), making sure that from each occupied bin we could exit the design space along any direction parallel with any of the axes without encountering any other occupied bins. This is illustrated for three dimensions in Figure 1.4.

We achieve this using the following technique. If $\mathbf{X}$ denotes the $n \times k$ matrix in which we wish to build our sampling plan of $n$ points in $k$ dimensions (each row represents a point) we begin by filling up $\mathbf{X}$ with random permutations of $\{1, 2, \ldots, n\}$ in each column and we normalize our plan into the $[0, 1]^k$ box. The following code performs this in *MATLAB*.

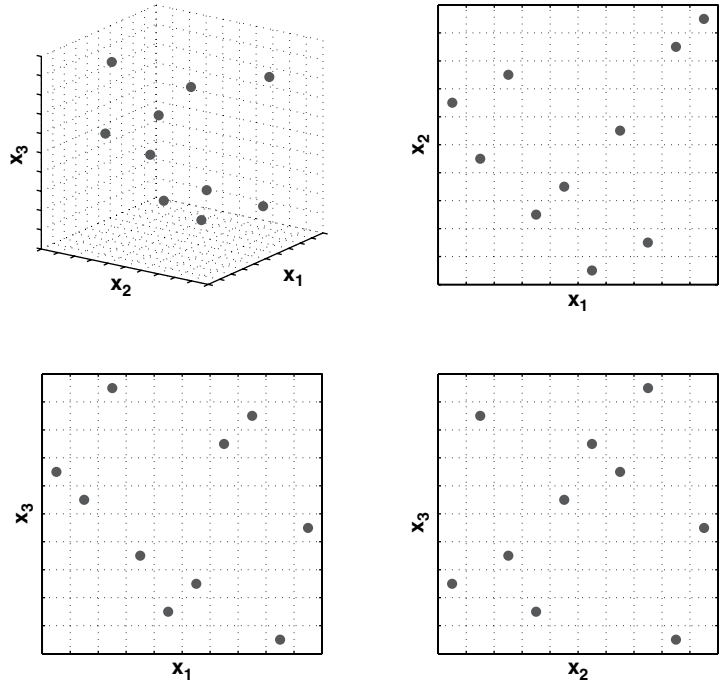



**Figure 1.4.** Three-variable, ten-point Latin hypercube sampling plan shown in three dimensions (top left), along with its two-dimensional projections. All ten points are visible on each of the latter, while each row and column of bins contains exactly one point.

```
function X=rlh(n, k, Edges)
% Generates a random Latin hypercube within the [0,1]^k hypercube.
%
% Inputs:
%       n – desired number of points
%       k – number of design variables (dimensions)
%       Edges – if Edges=1 the extreme bins will have their centres
%               on the edges of the domain, otherwise the bins will
%               be entirely contained within the domain (default
%               setting).
%
% Output:
%       X – Latin hypercube sampling plan of n points in k
%           dimensions.

if nargin < 3
   Edges=0;
end

% Pre – allocate memory
X=zeros (n,k);

for i=1:k
   X(:,i)=randperm(n)';
end

if Edges==1
   X=(X-1)/(n-1);
else
   X=(X-0.5)/n;
end
```

The above recipe will thus yield a randomized sampling plan, whose projections on to the axes are uniformly spread (multidimensional stratification). This, however, does not guarantee that the plan will be space-filling. After all, placing all of the points on the main diagonal of the design space will fulfil the multidimensional stratification criterion, but, intuitively, will not fill the available space uniformly. We therefore need some measure of uniformity that will allow us to distinguish between 'good' and 'bad' Latin hypercubes, even in cases that are not as clear-cut as the diagonal example given above.

### *1.4.3 Space-filling Latin Hypercubes*

One of the most widely-used measures to evaluate the uniformity ('space-fillingness') of a sampling plan is the *maximin* metric introduced by Johnson *et al*. (1990). The criterion based on this may be defined as follows.

Let $d_1, d_2, \ldots, d_m$ be the list of the unique values of distances between all possible pairs of points in a sampling plan $\mathbf{X}$, sorted in ascending order. Further, let $J_1, J_2, \ldots, J_m$ be defined such that $J_j$ is the number of pairs of points in $\mathbf{X}$ separated by the distance $d_j$.

***Definition 1.1.*** *We will call* $\mathbf{X}$ *a maximin plan among all available plans if it maximizes* $d_1$ *and, among plans for which this is true, minimizes* $J_1$.

Clearly, this definition could be applied to any set of sampling plans, but, since we would like to keep the appealing stratification properties of Latin hypercubes, we restrict our scope to that class of designs. Nonetheless, even across this narrower domain, Definition 1.1 might still yield several maximin designs. Therefore we shall use the more complete 'tie-breaker' definition of Morris and Mitchell (1995).

***Definition 1.2.*** *We call* $\mathbf{X}$ *the maximin plan among all available plans if it maximizes* $d_1$*, among plans for which this is true, minimizes* $J_1$*, among plans for which this is true, maximizes* $d_2$*, among plans for which this is true, minimizes* $J_2, \ldots,$ *minimizes* $J_m$.[5]

Before proceeding further, we need to clarify what we mean by 'distance' in the above definitions. The metric most widely used is the *p-norm* of the space:

$$d_p\left(\mathbf{x}^{(i_1)}, \mathbf{x}^{(i_2)}\right) = \left(\sum_{j=1}^{k} |x_j^{(i_1)} - x_j^{(i_2)}|^p\right)^{1/p}. \tag{1.5}$$

For $p = 1$ this is the rectangular distance (sometimes also referred to as the *Manhattan norm* in reference to the district's grid-like layout) and $p = 2$ yields the *Euclidean norm*. There is little evidence in the literature of one being more suitable than the other for sampling plan evaluation if no assumptions are made regarding the structure of the model to be fitted, though it must be noted that the rectangular distance is considerably cheaper to compute. This can be quite significant, especially if large sampling plans are to be evaluated.

And now, onto the practicalities of working with Definition 1.2 in a computational context. First, we need to build the vectors $d_1, d_2, \ldots, d_m$ and $J_1, J_2, \ldots, J_m$. Here is a *MATLAB* implementation of a function that accomplishes this task:

```
function [J,distinct_d]=jd(X,p)
% Computes the distances between all pairs of points in a sampling
% plan X using the p—norm, sorts them in ascending order and
% removes multiple occurrences.
%
% Inputs:
%       X — sampling plan being evaluated
%       p — distance norm (p=1 rectangular — default, p=2 Euclidean)
%
% Outputs:
%       J — multiplicity array (that is, the number of pairs
%           separated by each distance value).
%       distinct_d — list of distinct distance values
```

(*continued*)

[5] To be completely rigorous, Definition 1.1 has been shown by Johnson *et al*. (1990) to be equivalent to the so-called *D-optimality* criterion used in linear regression, whereas Definition 1.2 is simply intuitively appealing and practically more useful. As we are considering sampling plans that do not make any assumptions relating to model structure, we shall use the latter.

```
if ~exist('p','var')
    p=1;
end

% Number of points in the sampling plan
n=size(X,1);

% Compute the distances between all pairs of points
d=zeros(1,n*(n-1)/2);

for i=1:n-1
    for j=i+1:n
        % Distance metric: p-norm
        d((i-1)*n-(i-1)*i/2+j-i)=norm(X(i,:)-X(j,:),p);
    end
end

% Remove multiple occurrences
distinct_d=unique(d);

% Pre-allocate memory for J
J =zeros(size(distinct_d));

% Generate multiplicity array
for i=1:length(distinct_d)
    % J(i) will contain the number of pairs separated
    % by the distance distinct_d(i)
    J(i)=sum(ismember(d,distinct_d(i)));
end
```

A very time-consuming part of this calculation is the creation of the vector that contains the distances between all possible pairs of points. This becomes especially significant for large sampling plans (for example, in the case of a 1000-point plan almost half a million calculations are required). Therefore pre-allocation of the memory is essential, which leaves us with the somewhat roundabout way of computing the indices of d (as opposed to appending each new element to d, which would require the use of expensive dynamic memory allocation).

We now need to implement Definition 1.2 itself. Since finding the most space-filling design will require pairwise comparisons, we 'divide and conquer' the problem by simplifying it to the task of choosing the better of two sampling plans. The function `mm(X1,X2,p)` accomplishes this, returning the index of the more space-filling of the two designs and 0 if they are equal (the $p$-norm is used to compute the distances):

```
function Mmplan=mm(X1,X2,p)
% Given two sampling plans chooses the one with the better
% space-filling properties (as per the Morris-Mitchell criterion).
%
% Inputs:
```

(*continued*)

```
%       X1, X2 — the two sampling plans
%       p — the distance metric to be used (p=1 rectangular —
%           default, p=2 Euclidean)
%
% Outputs:
%       Mmplan — if Mmplan=0, identical plans or equally space —
%       filling, if Mmplan=1, X1 is more space-filling,
%       if Mmplan=2, X2 is more space—filling.

if ~exist('p','var')
   p=1;
end

if sortrows(X1)==sortrows(X2)
   % If the two matrices contain the same points
   Mmplan=0;
else
   % Calculate the distance and multiplicity arrays
   [J1,d1]=jd(X1,p); m1=length(d1);
   [J2,d2]=jd(X2,p); m2=length(d2);

   % Blend the distance and multiplicity arrays together for
   % comparison according to Definition 1.2B. Note the different
   % signs — we are maximizing the d's and minimizing the J's.
   V1(1:2:2*m1-1)=d1;
   V1(2:2:2*m1)=-J1;

   V2(1:2:2*m2-1)=d2;
   V2(2:2:2*m2)=-J2;

   % The longer vector can be trimmed down to the length of the
   % shorter one
   m =min(m1,m2);
   V1=V1(1:m); V2=V2(1:m);

   % Generate vector c such that c(i)=1 if V1(i)>V2(i), c(i)=2 if
   % V1(i)<V2(i) and c(i)=0 otherwise
   c=(V1>V2)+2*(V1<V2);

   % If the plans are not identical but have the same space—filling
   % properties
   if sum(c) ==0
      Mmplan=0;
   else
      % The more space—filling design (mmplan)
      % is the first non—zero element of c
      i=1;
      while c(i)==0
        i=i+1;
      end
   Mmplan =c(i);
   end
end
```

As we stated above, searching across a space of potential sampling plans can be accomplished by pairwise comparisons. We could, therefore, in theory, write an optimization algorithm with `mm` as the comparative objective. However, there is some experimental evidence (Morris and Mitchell, 1995) that the resulting landscape will be quite deceptive from an optimization point of view and therefore difficult to search reliably. The reason is that the comparison process will stop as soon as we find a nonzero element in the comparison array c and therefore the remaining values in $d_1, d_2, \ldots, d_m$ and $J_1, J_2, \ldots, J_m$ will be lost. These, however, could provide the optimization process with potentially useful 'slope' information on the global landscape.

Morris and Mitchell (1995) define the following scalar-valued criterion function used to rank competing sampling plans. This, while based on the logic of Definition 1.2, includes the vectors $d_1, d_2, \ldots, d_m$ and $J_1, J_2, \ldots, J_m$ in their entirety:

$$\Phi_q(\mathbf{X}) = \left( \sum_{j=1}^{m} J_j d_j^{-q} \right)^{1/q}. \tag{1.6}$$

The smaller the value of $\Phi_q$, the better the space-filling properties of $\mathbf{X}$ will be. Here is Equation (1.6) in *MATLAB*-speak:

```
function Phiq=mmphi(X, q, p)
% Calculates the sampling plan quality criterion of Morris and
% Mitchell.
%
% Inputs:
%       X - sampling plan
%       q - exponent used in the calculation of the metric
%       p - the distance metric to be used (p=1 rectangular -
%           (default, p=2 Euclidean)
%
% Output:
%       Phiq - sampling plan 'space-fillingness' metric

% Assume defaults if arguments list incomplete
if ~exist('p','var')
   p=1;
end

if ~exist('q','var')
   q=2;
end

% Calculate the distances between all pairs of
% points (using the p-norm) and build multiplicity array J
%
[J,d]=jd(X,p);

% The sampling plan quality criterion
Phiq=sum(J.*(d.^(-q)))^(1/q);
```

This equation distills the cumbersome definition of the maximin criterion into a rather neat and compact form, but it raises the question of how to choose the value of $q$. Large $q$'s will ensure that each term in the sum dominates all subsequent terms. Thus, because the distances $d_j$ are arranged in ascending order, $\Phi_q$ will rank sampling plans in a way that matches the original definition of the criterion quite closely and therefore we are back to the original problem. Lower values of $q$ yield a $\Phi_q$ landscape that, while it may not match the definition exactly, is more amenable to optimization.

To illustrate the relationship between Equation (1.6) and the maximin criterion of Definition 1.2, let us consider sets of 50 random Latin hypercubes of different sizes and dimensionalities. Let us then compute the ranking of each plan within its set according to Definition 1.2, as well as according to $\Phi_q$ (using $p = 1$ in each case) for a range of values of $q$.

Figure 1.5 depicts the results of this small investigation. It is unwise to draw far-reaching conclusions from only a few arbitrarily chosen experiments and we neither attempt this here, nor is it really necessary for practical purposes. Nonetheless, the correlation plots suggest that the larger the sampling plan, the smaller the $q$ required to produce a ranking based on $\Phi_q$ that matches that of Definition 1.2 almost exactly. Taking the case of the set of 50 100-point 15-variable hypercubes, the bottom right-hand tile of Figure 1.5 indicates that the $\Phi_{250}$-based ranking only differs from that of the definition in three places and even these plans are only mis-ranked by one place. At the other end of the scale, it can be seen that for $q = 1$, there is virtually no correlation, except for the smallest sampling plans considered.

Should the reader wish to conduct their own investigation for different families of sampling plans, here are the tools required to do it. The rankings according to `mm` and `mmphi` using a simple bubble sort algorithm are implemented in `mmsort.m` and `phisort.m` respectively:

```
function Index=mmsort(X3D,p)
% Ranks sampling plans according to the Morris–Mitchell criterion
% definition. Note: similar to phisort, which uses the numerical
% quality criterion Phiq as a basis for the ranking.
%
% Inputs:
%       X3D – three–dimensional array containing the sampling plans
%             to be ranked.
%       p – the distance metric to be used (p=1 rectangular –
%           default, p=2 Euclidean)
%
% Output:
%       Index – index array containing the ranking

if ~exist('p','var')
   p=1;
end

% Pre-allocate memory
Index=(1:size(X3D,3));
```

(*continued*)

```
% Bubble-sort
swap_flag=1;

while swap_flag==1
  swap_flag=0;
  i=1;
  while i<=length(Index)-1
    if mm(X3D(:,:,Index(i)),X3D(:,:,Index(i+1)),p)==2
      buffer=Index(i);
      Index(i)=Index(i+1);
      Index(i+1)=buffer;
      swap_flag=1;
    end
    i=i+1;
  end
end
```

`phisort.m` only differs in having `q` as the third argument, as well as the comparison line being: `if mmphi(X3D(:,:,index(i)),q,p)` > `mmphi(X3D(:,:,index(i+1)),q,p)`.

So how does one find the best Latin hypercube for a given application? Morris and Mitchell (1995) recommend minimizing $\boldsymbol{\Phi}_q$ for $q = 1, 2, 5, 10, 20, 50$ and 100 (Figure 1.5 confirms these to be reasonable values) and then choosing the best of the resulting plans according to the actual maximin definition. This is one more possible use for `mmsort.m`; one can create a matrix `X3D` with each two-dimensional slice being the best sampling plan found according the various $\Phi_q$'s and `mmsort(X3D,1)` will then rank them according to Definition 1.2 using the rectangular distance metric. The only remaining question then is, how to find these optimized $\Phi_q$ designs? We discuss this next.

### Optimizing $\Phi_q$

Having established a criterion that we can use to assess the quality of a Latin hypercube sampling plan, we now need a systematic means of optimizing this metric across the space of Latin hypercubes. This is *not* a trivial task – the reader will recall from the earlier discussion of Latin squares that this space is vast. Indeed, it is so vast that for many practical applications we have little hope of locating the globally optimal solution and we therefore must aim to find the best possible sampling plan within a given computing time budget.

This budget depends on the computational cost of obtaining an objective function value. The optimum division of the total computational effort between generating the sampling plan and actually computing objective function values at the sites therein is an open research question, though typically one would rarely allocate more than about 5% of the total wall time to the former task.

Parallels can be drawn with choosing how much time to spend devising a timetable for revision before an exam. A better timetable will make the revision more effective, but one doesn't want to take too much of the revision time itself!

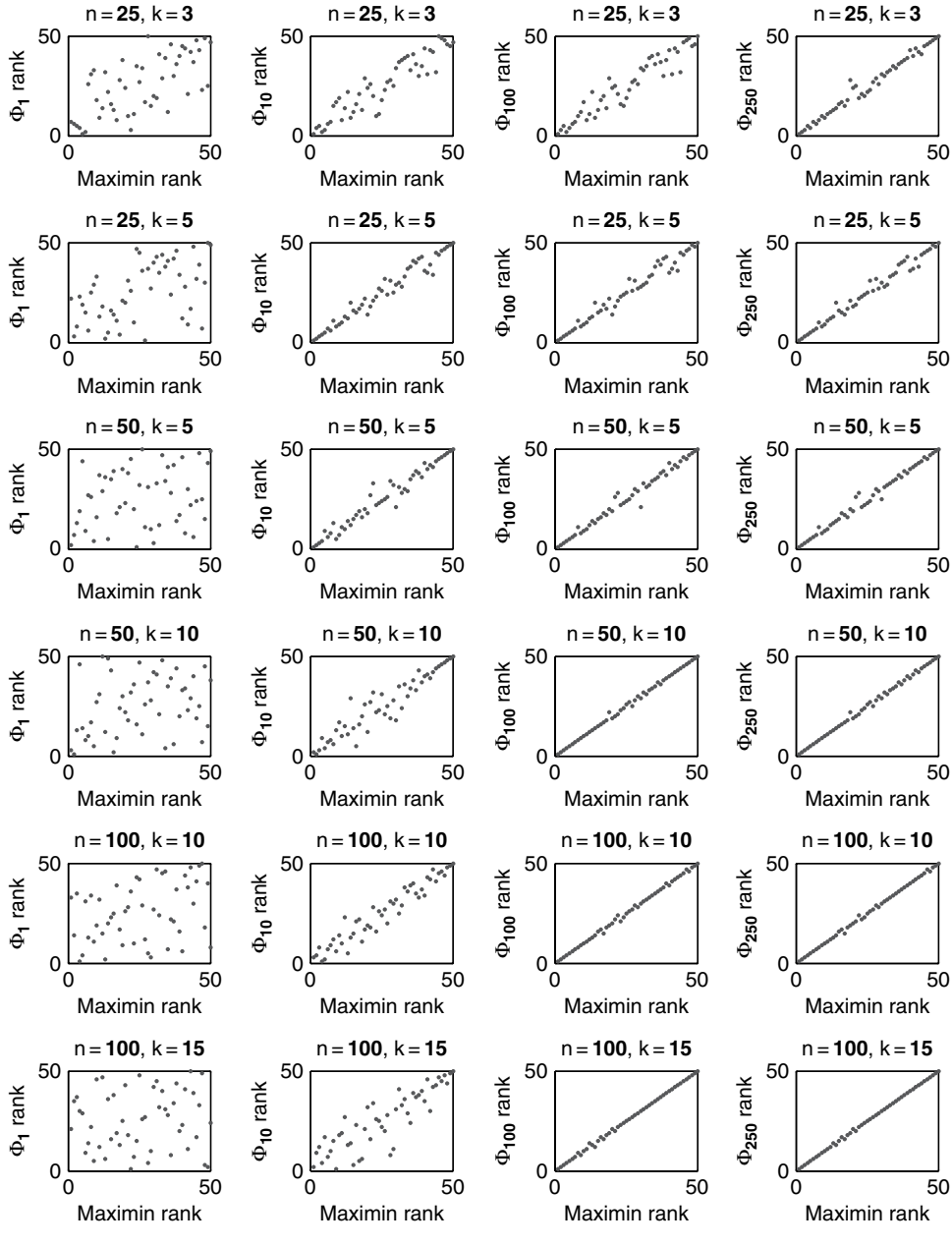



**Figure 1.5.** Scatter plots of maximin rankings versus rankings according to $\Phi_q$, different values of $q$ for sets of 50 random Latin hypercubes of different sizes and dimensionalities (the rectangular distance metric was used).

One of the challenges of devising a sampling plan optimizer is to make sure that the search process always stays within the space of Latin hypercubes. We have seen that the defining feature of a Latin hypercube **X** is that each column is a permutation of the list of the possible levels of the corresponding variable. The smallest alteration we can therefore make to a Latin hypercube without spoiling its key multidimensional stratification property is to swap two of the elements within any of the columns of **X**. Here is a *MATLAB* implementation of 'mutating' a Latin hypercube in this way, generalized to random changes applied to multiple sites:

```
function X=perturb(X, PertNum)
% Interchanges pairs of randomly chosen elements within randomly
% chosen columns of a sampling plan a number of times. If the plan is
% a Latin hypercube, the result of this operation will also be a
% Latin hypercube.
%
% Inputs:
%       X – sampling plan
%       PertNum – the number of changes (perturbations) to be made
%                 to X.
%
% Output:
%       X – perturbed sampling plan

if ~exist('PertNum','var')
        PertNum=1;
end

[n,k]=size(X);

for pert_count=1:PertNum
        col=floor(rand*k)+1;

        % Choosing two distinct random points
        el1=1; el2=1;
        while el1==el2
           el1=floor(rand*n)+1;
           el2=floor(rand*n)+1;
        end

        % Swap the two chosen elements
        buffer=X(el1,col);
        X(el1,col)=X(el2,col);
        X(el2,col)=buffer;
end
```

We use the term 'mutation' because this is a problem that lends itself to nature-inspired computation. Morris and Mitchell (1995) use a simulated annealing (SA) algorithm, the detailed pseudocode of which can be found in their paper. As an alternative here we offer a method based on evolutionary operation (EVOP).

### Evolutionary operation

As introduced by Box (1957), evolutionary operation was designed to optimize chemical processes. The current parameters of the reaction would be recorded in a box at the centre of a board, with a series of 'offspring' boxes along the edges containing values of the parameters slightly altered with respect to the central, 'parent' values. Once the reaction was completed for all of these sets of variable values and the corresponding yields recorded, the contents of the central box would be replaced with that of the setup with the highest yield and this would then become the parent of a new set of peripheral boxes.

This is generally viewed as a local search procedure, though this depends on the mutation step sizes, that is on the differences between the parent box and its offspring. The longer these steps, the more global is the scope of the search.

For the purposes of the Latin hypercube search, we apply a variable scope strategy. We start with a long step length (that is a relatively large number of swaps within the columns) and, as the search progresses, we gradually home in on the current best basin of attraction by reducing the step length to a single change.

In each generation we mutate (randomly, using `perturb.m`) the parent a `pertnum` number of times. We then pick the sampling plan that yields the smallest $\Phi_p$ value (as per the Morris–Mitchell criterion, calculated using `mmphi.m`) among all offspring *and* the parent; in evolutionary computation parlance this selection philosophy is referred to as *elitism*. Should the reader wish to opt for a nonelitist approach (for example to encourage a more global search), the EVOP code can be modified fairly easily to exclude the parent from the selection step.

The EVOP based search for space-filling Latin hypercubes is thus a truly evolutionary process: the optimized sampling plan results from the nonrandom survival of random variations.

### Putting it all together

We now have all the pieces of the optimum Latin hypercube sampling process puzzle: the random hypercube generator as a starting point for the optimization process, the 'space-fillingness' metric that we need to optimize, the optimization engine that performs this task and the comparison function that selects the best of the optima found for the various $q$'s. We just need to put this all into a sequence. Here is the *MATLAB* embodiment of the completed puzzle (with some of the frills omitted):

```
function X=bestlh(n,k, Population, Iterations)
% Generates an optimized Latin hypercube by optimizing the Morris–
% Mitchell criterion for a range of exponents and plots the first two
% dimensions of the current hypercube throughout the optimization
% process. %
% Inputs:
%       n – number of points required
%       k – number of design variables
%       Population – number of individuals in the evolutionary
%                    operation optimizer
%       Iterations – number of generations the evolutionary
```

(*continued*)

```
%                    operation optimizer is run for
%       Note: high values for the two inputs above will ensure high
%       quality hypercubes, but the search will take longer.
%
% Output:
%       X - optimized Latin hypercube

if k<2
      error('Latin hypercubes are not defined for k<2');
end

% List of qs to optimize Phi_q for
q=[1 2 5 10 20 50 100];

% Set the distance norm to rectangular for a faster search. This can
% be changed to p=2 if the Euclidean norm is required.
p=1;

% We start with a random Latin hypercube
XStart=rlh(n,k);

% For each q optimize Phi_q
for i=1: length(q)
   fprintf('Now optimizing for q=%d...\n', q(i));
   X3D(1:n,1:k,i)=mmlhs(XStart, Population, Iterations, q(i));
end

% Sort according to the Morris-Mitchell criterion
Index=mmsort(X3D,p);
fprintf ('Best lh found using q=%d...\n', q(Index(1)));

% And the Latin hypercube with the best space-filling properties is...

X=X3D(:,:,Index(1));

% Plot the projections of the points onto the first two dimensions
plot (X(:,1),X(:,2),'ro');drawnow;

title (strcat('Morris-Mitchell optimum plan found using q=',...
   num2str(q(Index(1)))));

xlabel('x 1'); ylabel('x 2');
```

It is worth noting that we need not necessarily have sorted all the candidate plans in ascending order – after all, it is the best one we are really interested in. Nonetheless, the added computational complexity is minimal (the vector is only ever going to contain as many elements as there are candidate $q$ values and only an index array is sorted, not the actual repository of plans) and this gives the reader the opportunity to compare, if desired, the plans different choices of $q$ lead to.

### *1.4.4 Space-filling Subsets*

We have, so far, looked at the problem of minimizing $\Phi_q$ over the space of all Latin hypercubes of a certain size $n$ and dimensionality $k$. Another question might be: how do we find the best space-filling plan across a more restricted space, say, that of $n_s$ element subsets $\mathbf{X}_s$ of an $n$ element plan $\mathbf{X}$? This is not merely an academic exercise, as this problem will arise later on when we discuss improving the quality of a predictor by running a space-filling subset of its training data through higher fidelity analysis.

Since selecting the $n_s$ element subset that minimizes $\Phi_q$ is an *NP-complete* problem and an exhaustive search would have to examine ${}_{n_s}C_n = n_s!/n!(n_s - n)!$ subsets (infeasible for all but very moderate cardinalities), an alternative strategy could be the following greedy algorithm aimed at locating at least a local optimum.

**Strategy one: greedy local search**

We start from a sample point $\mathbf{x}^{(i)},\ i \leq n$, as the first member of $\mathbf{X}_s$, we then loop through the remaining candidates $\mathbf{x}^{(j)},\ j = 1, \ldots, i-1$ and $j = i+1, \ldots,\ n$, and add the point that minimizes the Morris–Mitchell criterion. This loop is then repeated (always leaving out the points we have already included), choosing each time the point that, when added to $\mathbf{X}_s$, minimizes the optimality criterion over the points we have so far. This is akin to a local optimization of the citerion – better results can be achieved by repeating the process from all $n_c$ possible starting points, keeping the best $\mathbf{X}_s$ overall (multistart local search).

While not exhaustive, even this approach can prove computationally prohibitive beyond plans greater than a few dozen elements. Here is an even cheaper alternative.

**Strategy two: exchange algorithm**

We start from a randomly selected subset $\mathbf{X}_s$ and calculate the $\Phi_q$ criterion. We then exchange the first point $\mathbf{x}_\mathbf{s}^{(\mathbf{1})}$ with each of the remaining points in $\mathbf{X}\backslash\mathbf{X}_s$ and retain the exchange that gives the best $\Phi_q$. Here is the *MATLAB* implementation of this technique:

```
function [Xs,Xr]=subset(X,ns)
% Given a sampling plan, returns a subset of a given size with
% optimized space-filling properties (as per the Morris-Mitchell
% criterion).
%
% Inputs:
%       X - full sampling plan
%       ns - size of the desired subset
%
% Outputs:
%       Xs - subset with optimized space-filling properties
%       Xr - remainder X\Xs

n=size(X,1);

% Norm and quality metric exponent - different values can be used if
% required
p=1; q=5;
```

(*continued*)

```
r=randperm(n);

Xs=X(r(1:ns),:);
Xr=X(r(ns+1:end),:);

for j=1:ns
    orig_crit=mmphi(Xs,q,p);
    orig_point=Xs(j,:);

    % We look for the best point to substitute the current one with
    bestsub=1;
    bestsubcrit=Inf;

    for i=1:n-ns
      % We replace the current, jth point with each of the
      % remaining points, one by one
      Xs(j,:)=Xr(i,:);
      crit=mmphi(Xs,q,p);

      if crit< bestsubcrit
          bestsubcrit=crit;
          bestsub=i;
      end
end

    if bestsubcrit<orig_crit
        Xs(j,:)=Xr(bestsub,:);
    else
      Xs(j,:)=orig_point;
    end
end
```

## 1.5 A Note on Harmonic Responses

We conclude our discussion of sampling plans with uniform projections with some 'small print' regarding cases when exact uniformity is actually undesirable.

If the function being sampled is expected to have a harmonic component with a wavelength comparable to an integer multiple of $1/n$ (the width of each bin in a Latin hypercube), the completely uniform projection properties of a Latin hypercube (or full factorial design) might lead to misleading samples (they always sample the signal in the same phase, therefore it will seem like the points could have come from a constant function). This potential (though somewhat unlikely) pitfall can be avoided by adding a small random perturbation to each point:

```
% Add a random perturbation
% if harmonic component suspected
if perturb ==
    X=X+(rand(n,k)-0.5)*(1/n)*0.25;
end
```

## 1.6 Some Pointers for Further Reading

The importance of sampling plan design in a wide range of disciplines is reflected in the richness of the relevant literature (though not all of this body of work uses the same terminology as the present text). Our practical introduction to the subject is limited to a small subset of the plethora of techniques that have emerged and it is inevitably subject, to some extent, to the authors' personal bias. More significantly, though, we have highlighted the approaches that, in our view, make the weakest assumptions regarding the type and the size of the problem.

We have also limited the descriptions of the theoretical backgrounds of the techniques covered to what we deemed to be necessary for their correct use. Here are a few additional resources the reader may wish to consult.

The history of Latin hypercubes began in 1979 with a *Technometrics* paper by McKay *et al*. (1979). A series of refinements have followed, including the application of intersite distance criteria to Latin hypercube plans. Of these we have focused here on the maximin criterion – the text by Santner *et al*. (2003) is a good source of information on others.

Another class of sampling plans that have generated some interest in recent decades are *orthogonal arrays*. An $n \times k$ matrix $\mathbf{X}$ with entries from a set of $s \geq 2$ symbols is called an orthogonal array of strength $r$, size $n$ with $k$ constraints and $s$ levels if each $n \times r$ submatrix of $\mathbf{X}$ contains all possible $1 \times r$ row vectors with the same frequency $\lambda$ (note the rather awkward restriction that the number of entries must be $n = \lambda s^r$). In a 1993 paper, Tang (1993) introduces orthogonal array based Latin hypercubes. These essentially extend the univariate stratification properties of Latin hypercubes to $r$-variate margins, again for a limited range of plan sizes.

The sampling plans discussed in this chapter are generally based on the assumption that the size $n$ of the plan is predetermined. This is not always the case, as, although we may know what the total computing time is we can budget for, it is not always obvious how many candidate designs we will be able to evaluate in that time (some designs may be harder to analyse than others).

Should we run out of time after evaluating, say, 80% of the points of our carefully constructed Morris–Mitchell optimal Latin hypercube, there will be no guarantees regarding the space-filling properties of that $0.8n$-strong subset.[6] In such cases it often makes sense to start with a plan small enough to fit safely into the budget and select subsequent points one by one (until the time runs out) based on models fitted to the points we have thus far, that is deciding where we are sampling next based on which areas appear promising. We shall discuss such procedures in detail later on; however, if they are infeasible for some reason (for example the budget is very large and thus the repeated model fitting process may be impractical), there is an alternative: Sobol sequences (Sobol, 1994). These are sampling plans with reasonably good space-filling properties (at least for large $n$) and have the property that, for any $n$ and $k > 1$, the sequence for $n-1$ and $k$ is a subset of the sequence for $n$ and $k$. Thus, from a 'space-fillingness' point of view, it does not matter when the time runs out.

[6] A question arising from here is, what is the ordering of a sampling plan $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ that, given the 'space-fillingness' metric $\Phi$, optimizes $\Phi(\{\mathbf{x}_1, \mathbf{x}_2\})$, $\Phi(\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}), \ldots$ and $\Phi(\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n\})$ simultaneously?

## References

Box, G. E. P. (1957) Evolutionary operation: a method for increasing industrial productivity. *Applied Statistics*, **6**(2), 81–101, June.

Johnson, M. E., Moore, L. M. and Ylvisaker, D. (1990) Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, **26**, 131–148.

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

McKay, M. D., Beckman, R. J. and Conover, W. J. (1979) A comparison of three methods for selecting values of input variables in the anslysis of output from a computer code. *Technometrics*, **21**(2), 239–245, May.

Morris, M. D. (1991) Factorial sampling plans for preliminary computational experiments. *Technometrics*, **33**(2), 161–174.

Morris, M. D. and Mitchell, T. J. (1995) Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, **43**, 381–402.

Raymer, D. P. (2006) *Aircraft Design: A Conceptual Approach*, Education Series, 4th edition, American Institute of Aeronautics and Astronautics, Washington, DC.

Santner, T. J., Williams, B. and Notz, W. (2003) *The Design and Analysis of Computer Experiments*, Springer-Verlag, Berlin.

Sobol, I. M. (1994) *A Primer for the Monte Carlo Method*, CRC Press, Boca Raton, Florida.

Tang, B. (1993) Orthogonal array-based latin hypercubes. *Journal of the American Statistical Association*, **88**(424), 1392–1397, December.

# 2

# Constructing a Surrogate

This text is written around the core problem of attempting to learn a mapping $y = f(\mathbf{x})$ that lives in a black box, which obscures the physics that converts the vector $\mathbf{x}$ into a scalar output $y$. This black box could take the form of either a physical or computer experiment, for example a finite element code, which calculates the maximum stress ($f$) for given product dimensions ($\mathbf{x}$). The generic solution method is to collect the output values $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$ that result from a set of inputs $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}$ and find a best guess $\widehat{f}(\mathbf{x})$ for the black box mapping $f$, based on these known observations.

In this chapter we discuss the fundamentals and some of the technical minutiae of a number of specific surrogate model types capable of accomplishing this learning process. We begin, however, with a generic discussion of the key stages of the surrogate model building process.

## 2.1 The Modelling Process

### *2.1.1 Stage One: Preparing the Data and Choosing a Modelling Approach*

Chapter 1 dealt with two of the preliminary steps of this stage. The first was the identification, through a small number of observations, of the inputs that have a significant impact on $f$; that is the determination of the shortest design variable vector $\mathbf{x} = \{x_1, x_2, \ldots, x_k\}^{\mathrm{T}}$ that, by sweeping the ranges of all of its variables, can still elicit most of the behaviour the black box is capable of. If the black box was an electronic device fitted with a large array of controls, this screening step would amount to identifying the $k$ controls that, when tweaked, influence its behaviour – an operation made difficult by the possible interactions between the controls. The ranges of the various design variables also have to be established at this stage.

The second step was to recruit $n$ of these $k$-vectors into a list $\mathbf{X} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(n)}\right\}^{\mathrm{T}}$, such that this represents the design space as thoroughly as possible, the challenge being

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

that $n$ is often small, as it is constrained by the cost (usually computational) associated with obtaining each observation.

It is perhaps worth reiterating here that it is a good idea to scale $\mathbf{x}$ at this stage into the unit cube $[0, 1]^k$, a step that can make some of the subsequent mathematics easier and can save us from a plethora of multidimensional scaling issues.

With Chapter 1 having put a suitable panoply of techniques in place for accomplishing the above, here we consider the next phase of the process, the actual attempt at learning $f$ through the data pairs $\{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$. This so-called *supervised* or *instance based learning* process is, essentially, a search across the space of conceivable functions $\widehat{f}$ that would replicate observations of $f$.

This space is infinite. After all, any number of (hyper)surfaces could be drawn to go through or pass within a certain range (accounting for experimental error) of the known observations. However, the overwhelming majority of these would generalize very poorly; that is they would be practically useless at predicting responses at new sites, which is what the purpose of the exercise is. Consider the somewhat extreme example of the 'needle(s) in the haystack' function:

$$\widehat{f}(\mathbf{x}) = \begin{cases} y^{(1)} & \text{if } \mathbf{x} = \mathbf{x}^{(1)} \\ y^{(2)} & \text{if } \mathbf{x} = \mathbf{x}^{(2)} \\ \ldots & \ldots\ldots \\ y^{(n)} & \text{if } \mathbf{x} = \mathbf{x}^{(n)} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Clearly, while all of the training data can be reproduced by this predictor, there is, at least as far as most engineering functions are concerned, something severely counter-intuitive and unsettling about it predicting 0 everywhere else. Of course, there is a minute chance that the function *does* look like (2.1) and by some extraordinary chance we happened to sample it exactly where the needles are, but this is quite unlikely.

One could invent many other, perhaps less contrived, examples that also feel somehow unnatural and, more to the point, generalize badly – ultimately all this suggests that we need some systematic means of filtering out such nonsensical predictors. Some scholars take a Bayesian stance here, for example the philosophy advocated by Rasmussen and Williams (2006) is '(speaking rather loosely) to give a prior probability to every possible function, where higher probabilities are given to functions that we consider to be more likely, for example because they are smoother than other functions'.

The approach we shall take in what follows is to hard-wire the *structure* of $\widehat{f}$ into the model selection algorithm and search over the space of its parameters to tune the approximation to the observations. For example, consider one of the simplest possible models: $\widehat{f}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + v$. Learning $f$ with this model implies that we have decided on its structure – it will be a hyperplane – and the model fitting process consists of finding the $k + 1$ parameters (the slope vector $\mathbf{w}$ and the intercept $v$) for which $\mathbf{w}^{\mathrm{T}}\mathbf{x} + v$ best fits the data (this will be accomplished by Stage Two).

All of the above is often further complicated by the presence of noise in the observed responses (we shall assume that the design vectors $\mathbf{x}$ are not corrupted in any way). We have discussed the *nature* of this noise early in Chapter 1. Here we concentrate on *learning from* such data, a process sometimes fraught with the danger of *overfitting*.

Overfitting occurs when the model is, in some sense, too flexible and it fits the data at too fine a scale, that is it fits the noise, as well as the actual underlying behaviour we are seeking to model. The second stage of the surrogate construction process deals with this *complexity control* problem through the process of estimating the parameters of the fixed structure model, but some foresight in this direction is also required here, at the model type selection stage.

This usually involves physics based considerations; that is the choice of modelling technique depends on our expectations of what the underlying response might look like. For example, if we have some observations of stress in an elastically deformed solid in response to small strains, it makes sense to model the stress with a simple linear approximation.

If such insights into the physics are not available and we fail to account for, say, the linearity of the data at this stage, we will end up adopting a complex, excessively flexible model. This will not be the end of the world (after all, the parameter estimation stage will hopefully 'linearize' the approximation through appropriate selection of the parameters that control its shape), but we will have missed an opportunity to obtain an algebraically simple, robust model. While they lack flexibility, simple linear (or polynomial) models have a lot going for them: for example, they can be used in subsequent symbolic calculations.

Conversely, if we assume erroneously that the data comes from, say, an underlying quadratic process and in reality the true function $f$ features multiple peaks and troughs, at the parameter estimation stage we will not be able to make up for the badly chosen model. A quadratic will simply be too stiff to fit a multimodal function, whatever its parameters.

### 2.1.2 Stage Two: Parameter Estimation and Training

Let us assume that in Stage One we have indentified the $k$ critical design variables, we have acquired the learning data set and we have selected a generic model structure $\widehat{f}(\mathbf{x}, \mathbf{w})$, the exact shape of the model being determined by the set of parameters $\mathbf{w}$. We are now faced with a *parameter estimation* problem: how do we select these $\mathbf{w}$'s such that the model best fits the data? Of the several known estimation criteria here we choose to discuss two.

#### Maximum Likelihood Estimation

Given a set of parameters $\mathbf{w}$ and thus the model $\widehat{f}(\mathbf{x}, \mathbf{w})$, we can compute the probability of the data set $\{(\mathbf{x}^{(1)}, y^{(1)} \pm \epsilon), (\mathbf{x}^{(2)}, y^{(2)} \pm \epsilon), \ldots, (\mathbf{x}^{(n)}, y^{(n)} \pm \epsilon)\}$ having resulted from $\widehat{f}$ (where $\epsilon$ is some small, constant margin around each point). For example, if we assume that the errors $\epsilon$ are independently randomly distributed according to a normal distribution with the standard deviation $\sigma$, the probability of the data set is

$$P = \frac{1}{(2\pi\sigma^2)^{\mathrm{n}/2}} \prod_{i=1}^{n} \left\{ \exp\left[ -\frac{1}{2} \left( \frac{y^{(i)} - \widehat{f}(\mathbf{x}, \mathbf{w})}{\sigma} \right)^2 \right] \epsilon \right\}. \tag{2.2}$$

Intuitively, this should be the same as its reverse, that is the likelihood of *the parameters given the data*. The evergreen scientific computing text of Press *et al*. (1986) explains this artifice, remarking that 'statistics is not a branch of mathematics'. Nonetheless, accepting

this intuitive relationship as a mathematical one facilitates model parameter estimation. Here is how. We simply maximize it, or, rather, to simplify calculations, minimize the negative of its natural logarithm:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \frac{\left[y^{(i)} - \widehat{f}(\mathbf{x}, \mathbf{w})\right]^2}{2\sigma^2} - n \ln \epsilon. \tag{2.3}$$

Note that, if we assume a constant $\sigma$ and a constant $\epsilon$, Equation (2.3) simplifies to the well-known least squares criterion:

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \left[y^{(i)} - \widehat{f}(\mathbf{x}, \mathbf{w})\right]^2. \tag{2.4}$$

**Cross-Validation**

Cross-validation involves splitting the data (randomly) into $q$ roughly equal subsets, then removing each of these subsets in turn and fitting the model to the remaining, aggregated, $q-1$ subsets. A loss function $L$ can then be computed, which measures the error between the predictor and the points in the subset we set aside at each iteration; the contributions to $L$ are then summed over the $q$ iterations.

More formally, if a mapping $\zeta : \{1, \ldots, n\} \to \{1, \ldots, q\}$ describes the allocation of the $n$ training points to one of the $q$ subsets and $\hat{f}^{-\zeta(i)}(\mathbf{x})$ is the value (at $\mathbf{x}$) of the predictor obtained by removing the subset $\zeta(i)$ (i.e. the subset to which observation $i$ belongs), the cross-validation measure, which we employ here as an estimate of the prediction error, is

$$\varepsilon_{\text{cv}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} L\left[y^{(i)}, \hat{f}^{-\zeta(i)}(\mathbf{x}^{(i)}, \mathbf{w})\right]. \tag{2.5}$$

Introducing the squared error in the role of the loss function and recalling from the previous section that our model $\widehat{f}$ is still a generic one, depending on the two parameters left undetermined there, we can rewrite Equation (2.5) as

$$\varepsilon_{\text{cv}}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} [y^{(i)} - \hat{f}^{-\zeta(i)}(\mathbf{x}^{(i)}, \mathbf{w})]^2. \tag{2.6}$$

To what extent Equation (2.6) is an unbiased estimator of true risk depends on the choice of $q$. It can be shown that if $q = n$, $\varepsilon_{\text{cv}}$ is an almost unbiased estimator of true risk. However, the variance of this *leave-one-out* measure can be very high, due to the $n$ subsets being very similar to each other. Hastie *et al*. (2001) recommend compromise values of $q = 5$ or $q = 10$. In practical terms, using fewer subsets has the added bonus of reducing the computational cost of the cross-validation process by reducing the number of models that have to be fitted.

### *2.1.3 Stage Three: Model Testing*

If the observational data are plentiful, a randomly selected subset (Hastie *et al*., (2001) recommend around $0.25n$ $\mathbf{x} \to y$ pairs) should be set aside at the outset for model testing

purposes. These observations must not be touched during Stages One and Two as their sole purpose is to allow us to evaluate the testing error (based on the difference between the true and approximated function values at the test sites) once the model has been built. We may, however, bizarre as it may sound, not be overly concerned with the global accuracy of our model if we are building an initial surrogate to seed a global infill criterion based strategy (see Section 3.2). In such cases the model testing phase may be skipped.

We note here that, ideally, the parameter estimation (Stage Two) should also be based on a separate subset, but observational data is seldom so abundant as to allow this luxury (if the function is very cheap to evaluate and we can choose our evaluation sites, we may not be in need of a surrogate model in the first place) if data are available for model testing and our main objective is a globally accurate model, we advocate using either a root mean squared error (RMSE) metric or the correlation coefficient ($r^2$). To test the model, we simply take a set of test data of size $n_t$ and a set of predictions at the locations of the test data and calculate either

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{n_t} \left(y^{(i)} - \widehat{y}^{(i)}\right)^2}{n_t}} \tag{2.7}$$

and/or

$$r^2 = \left( \frac{\text{cov}(\mathbf{y}, \widehat{\mathbf{y}})}{\sqrt{\text{var}(\mathbf{y})\text{var}(\widehat{\mathbf{y}})}} \right)^2 \tag{2.8}$$

$$= \left( \frac{n_t \sum_{i=0}^{n_t} y^{(i)} \widehat{y}^{(i)} - \sum_{i=0}^{n_t} y^{(i)} \sum_{i=0}^{n_t} \widehat{y}^{(i)}}{\sqrt{\left[n_t \sum_{i=0}^{n_t} y^{(i)2} - (\sum_{i=0}^{n_t} y^{(i)})^2\right]\left[n_t \sum_{i=0}^{n_t} \widehat{y}^{(i)2} - (\sum_{i=0}^{n_t} \widehat{y}^{(i)})^2\right]}} \right)^2, \tag{2.9}$$

Naturally we want the RMSE metric to be as small as possible, though it will, of course, be limited by any errors in the objective function ($f$) calculation. If, for example, the level of experimental or discretization error in an objective function calculation is known (i.e. the standard deviation), we might aim to fit a model with an RMSE within, say, one standard deviation. More likely one would aim for an RMSE within a certain percentage of the range of objective values in the observed data. As an example, Figure 2.1 shows the RMSE for a Kriging prediction as the number of sampling plan points is increased (Kriging will be covered in Section 2.4). The RMSE has been divided by the range of the test data. With random elements in both the sampling plan generation and the Kriging tuning process, there is a good deal of scatter in trend of lower RMSE for increased $n$. However, it is clear that more than 10 points is required for a reasonable global model, with RMSE $< 10\%$, and approximately 20 points will yield a very good model, with RMSE $< 2\%$.

The correlation coefficient can be used without worrying about the scaling of the two sets of data: it, in effect, only compares the shape of the landscapes, not the values. This is an advantage when comparing models of varying fidelity (we build multi-fidelity models in Chapter 8), where we may want to identify a low fidelity model that can predict the location of optima but not necessarily their values. In our experience, an $r^2 > 0.8$ indicates a surrogate with good predictive capabilities.

Figure 2.2 shows how $r^2$ behaves for the same Kriging models used to produce Figure 2.1, where $r^2 = 0.8$ roughly corresponds to a normalized RMSE of 0.1.

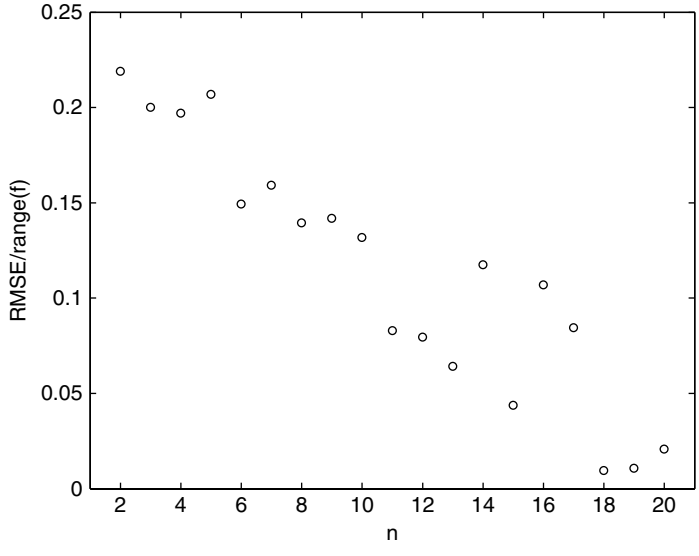



**Figure 2.1.** Normalized RMSE for a Kriging prediction of the Branin function as the number of sampling plan points is increased.

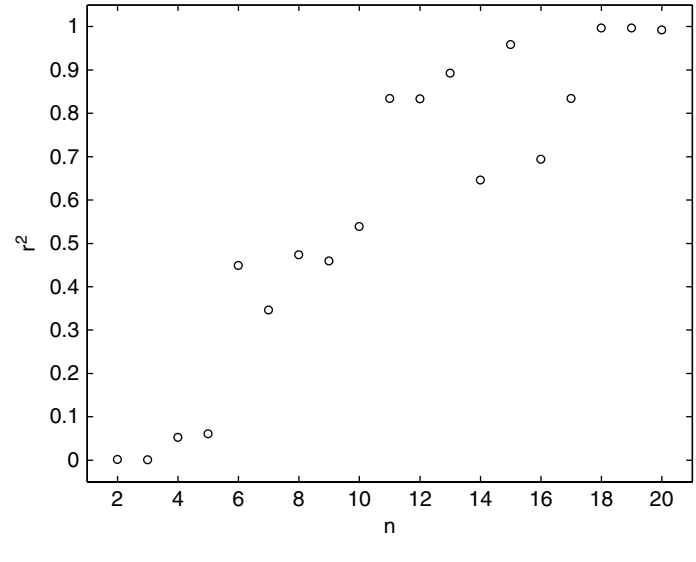



**Figure 2.2.** The correlation coefficient $r^2$ for a Kriging prediction of the Branin function as the number of sampling plan points is increased.

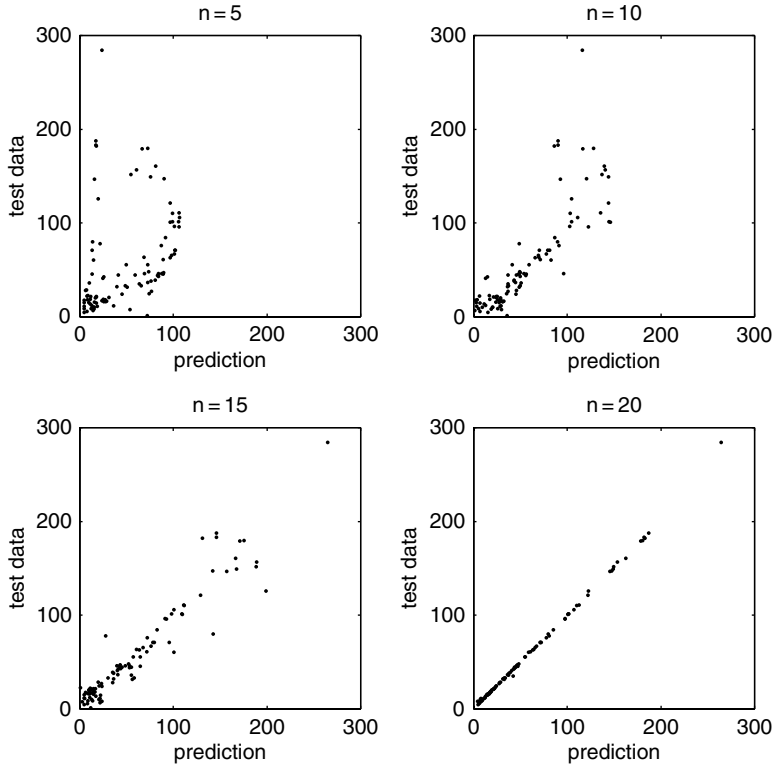



**Figure 2.3.** Predictions versus test data for varying $n$.

The above methods provide a quantitative measure of model accuracy, but it is also useful to have a visual understanding of the quality of the surrogate. In Figure 2.3 we have plotted the test data against the corresponding surrogate model predictions for four different sample plan sizes. From Figures 2.1 and 2.2 we know how the quality of the model improves as $n$ increases, and from Figure 2.3 we can see how the predictions gradually converge towards the values of the test data.

It is clear from the above figures that there is little point increasing $n$ beyond 20. The RMSE will never be precisely zero, but will fluctuate around the low value attained at $n = 18$, 19 and 20. At this stage the surrogate model is *saturated* with data, and any further additions will not improve the model *globally* (there will, of course, be local improvements at the newly added points if an interpolating model is used, as we are doing here). One could consider this as being rather like adding sugar to a cup of tea (the tea is the surrogate and the sugar is the data). A point is reached where no more sugar can dissolve and the tea cannot get any sweeter. A more flexible model (e.g. one with more parameters, or moving to interpolation rather than regression) will increase the saturation point – like making a hotter cup of tea!

## 2.2 Polynomial Models

Let us consider the scalar valued function $f$ observed according to the sampling plan $\mathbf{X} = \{x^{(1)}, x^{(2)}, \ldots x^{(n)}\}^{\mathrm{T}}$, yielding the responses $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots y^{(n)}\}^{\mathrm{T}}$. A polynomial approximation of $f$ of order $m$ can be written as

$$\widehat{f}(x, m, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i \tag{2.10}$$

In the spirit of the earlier discussion of maximum likelihood parameter estimation and equation (2.4), we shall seek to estimate $\mathbf{w} = \{w_0, w_1, \ldots, w_m\}^{\mathrm{T}}$ through a least squares solution of $\mathbf{\Phi w} = \mathbf{y}$, where $\mathbf{\Phi}$ is the Vandermonde matrix:

$$\mathbf{\Phi} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & \cdots & x_2^m \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & x_n & x_n^2 & \cdots & x_n^m \end{bmatrix} \tag{2.11}$$

The maximum likelihood estimate of $\mathbf{w}$ is thus

$$\mathbf{w} = \mathbf{\Phi}^{+}\mathbf{y} \tag{2.12}$$

where $\mathbf{\Phi}^{+} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}$ is the Moore–Penrose pseudo-inverse of $\mathbf{\Phi}$ (`pinv(Phi)` in *MATLAB* speak). This gives us a handy way of estimating $\mathbf{w}$, but it is not very useful in terms of estimating $m$.

The polynomial approximation (2.10) of order $m$ of an underlying function $f$ is, essentially, a Taylor series expansion of $f$ truncated after $m+1$ terms (Box and Draper, 1987). This suggests that greater values of $m$ (i.e. more Taylor expansion terms) will usually yield a more accurate approximation. However, the greater the number of terms, the more flexible the model becomes and there is a danger of overfitting the noise that may be corrupting the underlying response. Also, we run the risk of building an excessively 'snaking' polynomial with poor generalization.

We can prevent this by estimating the order $m$ through a number of different criteria (Cherkassky and Mulier, 1998, Ralston, 1965) – here we shall consider cross-validation. This means minimizing Equation (2.6) and, since this has to be done over the rather limited, discrete space of $m$'s ($m \in \mathbb{N}^*$ and usually $m \leq 15$), a direct search makes the most sense. We therefore compute the cross-validation measure for each $m = 1, 2, \ldots, 12$ by summing up the losses (squared errors) resulting when re-predicting the subsets of observations left out in, say, $q = 5$ rounds (as described in the second part of Section 2.1.2). At the end of the process the $m$ that yielded the smallest cross-validation metric is chosen.

Of course, the coefficient vectors $\mathbf{w}$ will be different for each $m$ considered (even in terms of their length) and for each cross-validation subset, so its least squares determination via Equation (2.12) has to be integrated as a lower level 'repair' step into the evaluation of each $m$.

Here is the *MATLAB* implementation of this recipe, followed by two examples of practical applications:

```
function [BestOrder, Coeff, MU]=polynomial(X,Y)
% Fits a one-variable polynomial to one-dimensional data
%
% Inputs:
%       X,Y - training data vectors
%
% Outputs:
%         BestOrder - the order of the polynomial, estimated using
%                     cross-validation
%         Coeff - the coefficients of the polynomial
%         MU - normalization vector

% The cross-validation will split the data into this many subsets
% (this may be changed if required)
q=5;

% This is the highest order polynomial that will be considered
MaxOrder=15;

n=length(X);

% X split into q randomly selected subsets
XS=randperm(n);

FullXS=XS;
% The beginnings of the subsets...
From=(1:round(n/q):n-1);
To=zeros(size(From));

% ...and their ends
for i=1:q-1
    To(i)=From(i+1)-1;
end

To(q)=n;

CrossVal=zeros(1,MaxOrder);

% Cycling through the possible orders
for Order=1:MaxOrder

    CrossVal(Order)=0;

    % Model fitting to subsets of the data
    for j=1:q
        Removed=XS(From(j):To(j));
        XS(From(j):To(j))=[];

        [P,S,MU]=polyfit (X(XS),Y(XS),Order);
```

(*continued*)

```
        CrossVal(Order)=CrossVal(Order)+...
           sum((Y(Removed)-polyval(P,X(Removed),S,MU)).^2)...
           /length(Removed);
        XS=FullXS;
    end
end

[MinCV, BestOrder]=min(CrossVal);

[Coeff,S,MU]=polyfit(X,Y,BestOrder);
```

### 2.2.1 Example One: Aerofoil Drag

The circles in Figure 2.4 represent 101 drag coefficient values obtained through a numerical simulation by iterating each member of a family of aerofoils towards a target lift value (see the Appendix, Section A.3). The members of the family have different shapes, as determined by the sampling plan $\mathbf{X} = \{x_1, x_2, \ldots, x_{101}\}$. Clearly, the responses $\mathbf{C_D} = \left\{C_{\mathrm{D}}^{(1)}, C_{\mathrm{D}}^{(2)}, \ldots, C_{\mathrm{D}}^{(101)}\right\}$ are corrupted by 'noise', these are deviations of the systematic variety (recall the Chapter 1 discussion of types of deviations), caused by small changes in the computational mesh from one design to the next.

To obtain the best polynomial through this data we simply compute:

```
>>[m,wrev,mnstd] = polynomial(X,CD)
```

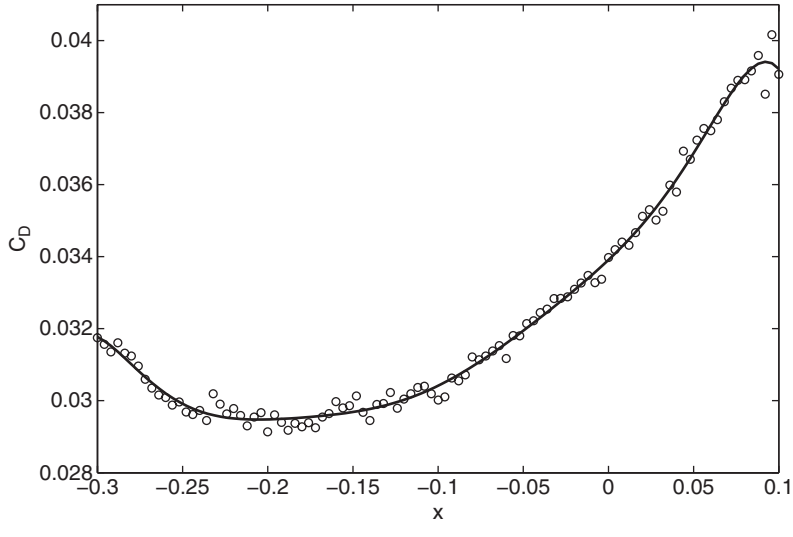



**Figure 2.4.** Eighth-order polynomial fitted through the aerofoil drag data – the order selected via cross-validation, the coefficients through likelihood maximization.

This yields `m=8`, that is, we now know that an eighth-order polynomial minimizes the cross-validation metric,[1] as well as the vector of coefficients:

```
>> wrev

wrev =

  Columns 1 through 4

  -0.00023896141908 -0.00015154784822 0.00120279110271
  0.00054322515686

  Columns 5 through 8

  -0.00172519329792 -0.00048110429064  0.00246816618471
  0.00271690687886

  Column 9

  0.03041507825750
```

and the vector `mnstd`, containing the mean $\mu$ and the standard deviation $\sigma$ of $\mathbf{X}$:

```
>> mnstd

mnstd =

  -0.10000000000000
  0.11720068259187
```

Making a prediction now using these values is very straightforward. There are two fundamental ways of doing this. Firstly, *MATLAB*'s `polyval` can be used – here is an example. Let us assume we wish to sample the predictor CD across the whole range, in steps of 0.005. We compute this by entering

```
>> CDhat=polyval(wrev,(-0.3:0.005:0.1),[],mnstd);
```

For a safer fitting process the data is normalized around its mean in `polynomial.m`,[2] hence the need to compute the vector `mnstd` and to feed it into `polyval` as well. The coefficient vector `wrev` is also computed based on the normalized data and this must be taken into account if a second evaluation method is selected: the explicit evaluation of the polynomial. This might be desired, for example, if further analytical calculations are to be performed on the fitted model.

[1] To be precise, this yields $m = 8$ *most of the time*. As the calculation of the cross-validation metric involves the random selection of subsets, the result may differ slightly (usually by one order either way) from one run to the next. This variability is particularly noticeable when there are relatively few training points. We therefore suggest running the function a few times and, if different values of `m` result, using the most frequently obtained value.

[2] On this occasion we did not scale the data into the [0, 1] interval; there is no need to do two scaling operations.

As `wrev` is actually **w** in reverse order, the polynomial approximation of $C_D$ will, in this case, be

$$\widehat{C_D}(x) = 10^{-3}(30.4151 + 2.7169\bar{x} + 2.4682\bar{x}^2 - 0.4811\bar{x}^3 - 1.7252\bar{x}^4 + 0.5432\bar{x}^5 + 1.2028\bar{x}^6 - 0.1515\bar{x}^7 - 0.2390\bar{x}^8) \quad (2.13)$$

where $\bar{x} = (x - \mu(\mathbf{X}))/\sigma(\mathbf{X}) = (x + 0.1)/0.1172$.

### 2.2.2 Example Two: a Multimodal Testcase

Let us consider the one-variable test function $f(x) = (6x-2)^2 \sin(12x-4)$ (see the Appendix, Section A.1), depicted by the dotted line in Figure 2.5. Its local minima of different depths can prove deceptive to some surrogate-based optimization procedures, so we shall revisit it in subsequent sections. Here we merely use it as another example, this time a multimodal (featuring multiple optima) one, of the two-level polynomial fitting procedure.

We have generated the training data (depicted by circles in Figure 2.5) by adding some normally distributed 'noise' to the function, as follows:

```
>>X=(0:0.02:1)';

>>n=length(X);

>>y=(6.*X-2).^2.*sin(12.*X-4)+randn(n,1)*1.1;
```

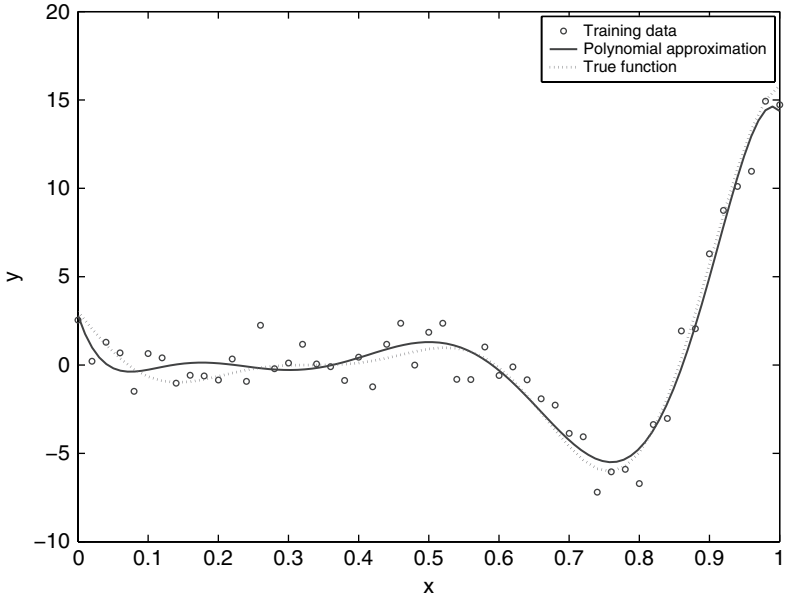



**Figure 2.5.** Seventh-order polynomial fitted through the data resulting from adding random noise to the test function depicted by the dotted line.

It turns out that a seventh-order polynomial fits this data best. This is shown by the continuous line in Figure 2.5.

### 2.2.3 What About the k-variable Case?

With the two illustrative examples presented above we conclude our brief foray into polynomial approximation, limiting ourselves here to the one-variable case. Nonetheless, it is, of course, possible to extend the formulation of Equation (2.10) to several variables. Perhaps the simplest way of viewing such approximations is in the form of a linear combination of *basis functions*

$$\widehat{f}(\mathbf{x}) = \sum_{i=1}^{n_b} w_i \psi^{(i)} \tag{2.14}$$

where the $\psi$'s are picked from a catalogue of all possible terms of order not greater than $m$. For example, for $m = 3$, $\psi^{(i)} \in \left\{1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2, x_1^2x_2, \ldots, x_3^3\right\}$. This now becomes a very complex parameter estimation problem, where not only the order $m$ and the coefficients $\mathbf{w}$ have to be determined but we also need to determine which entries we should pick from the catalogue. A great deal of work has been devoted to solving this problem and we shall conclude this chapter with some pointers (in Section 2.6) for the reader interested in a more detailed treatment. For now, we turn our attention to the most versatile of all the modelling approaches discussed in this book.

## 2.3 Radial Basis Function Models

Many branches of science often deal with complicated functions by expressing them in terms of a 'vocabulary' of basic functions, which have well-known properties and are more amenable to analysis. In fact, we have just touched upon an example, that of multivariable polynomials.

Perhaps the best known of such techniques are those that apply this logic to periodic functions (Fourier analysis), but here we are interested in the more general case of approximating any smooth, continuous function as a combination of simple basis functions. More specifically, we shall consider the case of symmetrical bases centred around a set of points (basis function centres) scattered around the design space. We begin with the case of *interpolating* radial basis function models, that is approximations built on the assumption that the data are not corrupted by noise.

### 2.3.1 Fitting Noise-Free Data

Let us consider the scalar valued function $f$ observed without error, according to the sampling plan $\mathbf{X} = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}^{\mathrm{T}}$, yielding the responses $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots y^{(n)}\}^{\mathrm{T}}$. We seek a radial basis function approximation to $\widehat{f}$ of the fixed form

$$\widehat{f}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\boldsymbol{\psi} = \sum_{i=1}^{n_c} w_i \psi(\|\mathbf{x} - \mathbf{c}^{(i)}\|) \tag{2.15}$$

where $\mathbf{c}^{(i)}$ denotes the $i$th of the $n_c$ basis function centres and $\boldsymbol{\psi}$ is the $n_c$-vector containing the values of the basis functions $\psi$ themselves, evaluated at the Euclidean distances between

the prediction site $\mathbf{x}$ and the centres $\mathbf{c}^{(i)}$ of the basis functions. Readers familiar with the technology of artificial neural networks will recognize this formulation as being identical to that of a single-layer neural network with radial coordinate neurons, featuring an input $\mathbf{x}$, hidden units $\psi$, weights $\mathbf{w}$, linear output transfer functions and output $\widehat{f}(\mathbf{x})$.

Thus far, the number of undetermined parameters stands at one per basis function and this will remain the situation if we choose one of a number of fixed bases. Examples include (with the relevant basis identification code in brackets, as defined for the purposes of our *MATLAB* implementation, to be discussed shortly)

- linear $\psi(r) = r$ (`ModelInfo.Code=1`),
- cubic $\psi(r) = r^3$ (`ModelInfo.Code=2`) and
- thin plate spline $\psi(r) = r^2 \ln r$ (`ModelInfo.Code=3`)

basis functions. More freedom to improve the generalization properties of Equation (2.15), at the expense of a more complex parameter estimation process, can be gained by using parametric basis functions, such as the

- Gaussian $\psi(r) = \mathrm{e}^{-r^2/(2\sigma^2)}$ (`ModelInfo.Code=4`),
- multiquadric $\psi(r) = (r^2 + \sigma^2)^{1/2}$ (`ModelInfo.Code=5`) or
- inverse multiquadric $\psi(r) = (r^2 + \sigma^2)^{-1/2}$ (`ModelInfo.Code=6`).

Whether we choose a set of parametric basis functions or fixed ones, the good news is that $\mathbf{w}$ is easy to estimate. This can be done via the interpolation condition

$$\widehat{f}\left(\mathbf{x}^{(j)}\right) = \sum_{i=1}^{n_c} w_i \psi(\|\mathbf{x}^{(j)} - \mathbf{c}^{(i)}\|) = y^{(j)}, \quad j = 1, \ldots, n \tag{2.16}$$

Herein lies the beauty of radial basis function approximations. Equation (2.16) is linear in terms of the basis function weights $\mathbf{w}$, yet the predictor $\widehat{f}$ can express highly nonlinear responses! It is easy to see that one of the conditions of obtaining a unique solution is that the system (2.16) must be 'square', that is $n_c = n$. It simplifies things if the bases actually coincide with the data points, that is $\mathbf{c}^{(i)} = \mathbf{x}^{(i)}, \ \forall i = 1, \ldots, n$, which leads to the matrix equation

$$\boldsymbol{\Psi}\mathbf{w} = \mathbf{y} \tag{2.17}$$

where $\boldsymbol{\Psi}$ denotes the so-called *Gram matrix* and it is defined as $\boldsymbol{\Psi}_{i,j} = \psi(\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|), \ i, j = 1, \ldots, n$. The fundamental step of the parameter estimation process is therefore the computation of $\mathbf{w} = \boldsymbol{\Psi}^{-1}\mathbf{y}$, and this is where the choice of basis function can have an important effect. For example, it can be shown that, under certain assumptions, Gaussian and inverse multiquadric basis functions always lead to a symmetric positive definite Gram matrix (Vapnik, 1998), ensuring safe computation of $\mathbf{w}$ via Cholesky factorization[3] – one

[3] A symmetric and positive definite $\boldsymbol{\Psi}$ can be decomposed into an upper triangular matrix and its transpose, such that Equation (2.17) becomes $\mathbf{U}^{\mathrm{T}}\mathbf{U}\mathbf{w} = \mathbf{y}$. This is then solved in two steps, first for $\mathbf{U}\mathbf{w}$ and then for $\mathbf{w}$. In *MATLAB*, this translates into performing the decomposition with `[U,p]=chol(Psi)`, followed by the computation of `w=U\(U'\y)`, where `p` is a positive integer if `Psi` is *not* positive definite and zero otherwise. If $\boldsymbol{\Psi}$ cannot be guaranteed to be positive definite, LU decomposition is used instead.

reason for the popularity of these basis functions.[4] It is also worth mentioning here that very close proximity of any two points in $\mathbf{X}$ can cause ill-conditioning (Michelli, 1986), with subsequent failure of the Cholesky factorization. This is a rather unlikely event if $\mathbf{X}$ is a space-filling sampling plan, but can become a nuisance if clusters of *infill* points (see Section 3.2) are added subsequently in specific areas of interest within the design domain.

Beyond determining $\mathbf{w}$, there is, of course, the additional task of estimating any other parameters introduced via the basis functions. A typical example is the $\sigma$ of the Gaussian basis function, usually taken to be the same for all basis functions, though a different one *can* be selected for each centre, as is customary in the case of the Kriging basis function, to be discussed shortly (once again, we trade additional parameter estimation complexity versus increased flexibility and, hopefully, better generalization).

While the correct choice of $\mathbf{w}$ will make sure that the approximation can reproduce the training data, the correct estimation of these additional parameters will enable us to minimize the (estimated) generalization error of the model. As discussed previously, this optimization step, say, the minimization of the cross-validation error estimate, can be performed at the top level, while the determination of $\mathbf{w}$ can be integrated into the process at the lower level, once for each candidate value of the parameter(s).

The function `rbf.m` is our *MATLAB* implementation of this parameter estimation process (based on a cross-validation routine), while `predrbf.m` will represent the surrogate, once its parameters have been estimated. The model building process is very simple. The bookkeeping device is the already alluded to structure `ModelInfo`, with `ModelInfo.X` containing the sampling plan $\mathbf{X}$ and `ModelInfo.y` the corresponding $n$-vector of responses $\mathbf{y}$. `ModelInfo.Code` specifies the type of basis function to be used. After running `rbf.m` the structure will also contain the estimated parameter values $\mathbf{w}$ and, if a parametric basis function is used, $\sigma$. These are stored in `ModelInfo.Weights` and `ModelInfo.Sigma` respectively. This is all the information `predrbf.m` needs to make a prediction, so its only input (other than `ModelInfo`, which must be made visible to it via `global`) is the $k$-vector representing the point where we wish to make the prediction.

The following *MATLAB* script illustrates the process through a simple example, where we attempt to learn the underlying response $f(\mathbf{x}) = 1/k \sum_{i=1}^{k} 1 - (2x_i - 1)^2, \ \mathbf{x} \in [0, 1]^k$ (`dome.m`), in two dimensions ($k = 2$) from a 10-point sample, using a thin plate spline radial basis function. The results are shown in Figure 2.6.

```
% Make ModelInfo visible to all functions

global ModelInfo

% Sampling plan
ModelInfo.X=bestlh(10,2,50,25);

% Compute objective function values – in this case using
% the dome.m test function
```

(*continued*)

[4] Theoretically, other bases can also be modified to have this property through the addition of a polynomial term; see, for example, Keane and Nair (2005).

```
for i=1:size(ModelInfo.X,1)
    ModelInfo.y(i)=dome(ModelInfo.X(i,:));
end

ModelInfo.y=ModelInfo.y';

% Basis function type:
ModelInfo.Code=3;

% Estimate model parameters
rbf

% Plot the predictor
x=(0:0.025:1);

for i=1:length(x)
    for j=1:length(x)
        M(j,i)=predrbf([x(i) x(j)]);
    end
end

contour(x,x,M)
```

Finally, a note on prediction error estimation. We have already indicated that the guarantee of a positive definite $\mathbf{\Phi}$ is one of the advantages of Gaussian radial basis functions. They also possess another desirable feature: it is relatively easy to estimate their prediction error at any $\mathbf{x}$ in the design space. Additionally, the expectation function of the

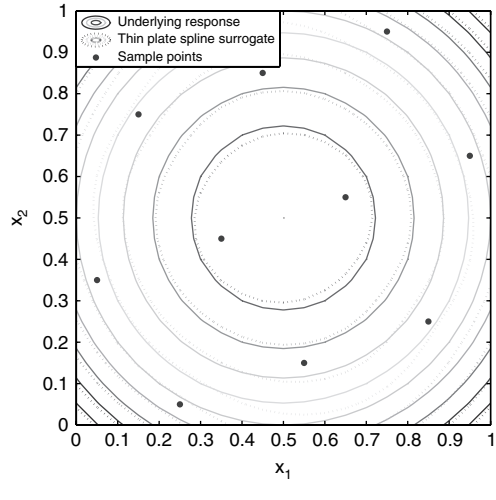



**Figure 2.6.** Contour plots of the underlying function $f(x_1, x_2) = 0.5[2 - (2x_1 - 1)^2 - (2x_2 - 1)^2]$ (`dome.m`) and its thin plate spline radial basis function approximation, along with the 10 points of a Morris–Mitchell optimal Latin hypercube sampling plan (obtained via `bestlh.m`).

improvement in minimum (or maximum) function value with respect to the minimum (or maximum) known so far can also be calculated quite easily, both of these features being very useful when the optimization of $f$ is the goal of the surrogate modelling process. We do not delve into these aspects here as in Section 3.2 we shall discuss them in great detail in the context of Kriging, a special case of which are Gaussian radial basis functions.

### 2.3.2 Radial Basis Function Models of Noisy Data

If the responses $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}^{\mathrm{T}}$ are corrupted by noise, following the recipe above may yield a model that overfits the data; that is it does not discriminate between the underlying response and the noise. Perhaps the easiest way around this is the introduction of added model flexibility in the form of the *regularization parameter* $\lambda$ (Poggio and Girosi, 1990). This is added to the main diagonal of the Gram matrix. As a result, the approximation will no longer pass through the training points and $\mathbf{w}$ will be the least squares solution of

$$\mathbf{w} = (\Phi + \lambda \mathbf{I})^{-1}\mathbf{y} \tag{2.18}$$

where $\mathbf{I}$ is an $n \times n$ identity matrix. Ideally, $\lambda$ should be set to the variance of the noise in the response data $\mathbf{y}$ (Keane and Nair, 2005), but since we usually do not know that, the remaining option is simply to add it to the list of parameters that need to be estimated.

Another means of constructing a regression model through noisy data using radial basis functions is to reduce the number of bases. Fewer than $n$ bases will, of course, yield a nonsquare $\mathbf{\Phi}$ and Equation (2.16) can then be used to obtain a least squares estimate of $\mathbf{w}$. Any additional parameters (e.g. the $\sigma$'s of Gaussian bases) can then be obtained once again through a higher level search, whereby we are seeking the optimum of some generalization error estimate (such as the cross-validation measure (2.6)). It is also possible to combine this with the regularization approach, adding further flexibility (but also making the parameter estimation process more complex).

The reduced number basis function method raises the nontrivial question of how to select the data points that will have bases attached to them. There are a number of ways in which this can be done. We shall discuss one such technique, the support vector regression method, later in this chapter; for the reader interested in others, pointers are included in Section 2.6.

The more general issue of fitting noisy data will be discussed in much more detail in the context of Kriging in Chapter 6.

## 2.4 Kriging

Of particular significance in surrogate based optimization, and as such given a section of its own, is the basis function of the form

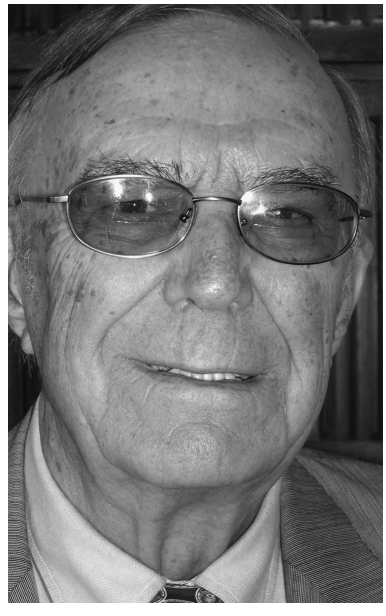
$$\psi^{(i)} = \exp\left(-\sum_{j=1}^{k} \theta_j \mid x_j^{(i)} - x_j \mid^{p_j}\right). \tag{2.19}$$

It is this basis function that is used in the method known as *Kriging* (see the historical note). Looking at Equation (2.19), we can see similarities with the Gaussian basis

function introduced in the previous section. Where a Gaussian radial basis function has $1/\sigma^2$, the Kriging basis has a vector $\boldsymbol{\theta} = \{\theta_1, \theta_2, \ldots, \theta_k\}^{\mathrm{T}}$, allowing the width of the basis function to vary from variable to variable. Also, where in the Gaussian basis the exponent is fixed at 2, giving a smooth function through the point $\mathbf{x}^{(i)}$, Kriging allows this exponent ($\mathbf{p}_j = \{p_1, p_2, \ldots, p_k\}^{\mathrm{T}}$) to vary (typically $p_j \in [1, 2]$) for each dimension in $\mathbf{x}$. With $\mathbf{p}$ fixed at $p_{(1,2,\ldots,k)} = 2$ and with a constant $\theta_j$ for all dimensions, the Kriging basis function is in fact the same as the Gaussian. We will first look at how to build a Kriging model and examine the benefits of using the vectors $\boldsymbol{\theta}$ and $\mathbf{p}$. For now, we will only consider Kriging interpolation. We will cover Kriging regression in Chapter 6.

**Historical note: Kriging and Danie G. Krige**

Matheron (1963) coined the term *Krigeage*, in honour of the South African mining engineer Danie Krige, who first developed the method now called *Kriging* (Krige, 1951). Kriging made its way into engineering design following the work of Sacks *et al*. (1989), who applied the method to the approximation of computer experiments.



Danie Krige (kindly provided by Prof. D. G. Krige)

Prof. Danie Krige's origins in South Africa stretch back for 350 years to the 17th century when the Cape was settled by the Dutch. In that period, the first South African Krige, of Dutch/German origin, married a French Huguenot girl and so initiated the South African Krige family tree. Danie was born in the Orange Free State in 1919, grew up on the Witwatersrand and graduated as a Mining Engineer at the University of the Witwatersrand at the end of 1938. His early career was spent in the gold mines, followed from 1945 by some 5 years in the Government Mining

(*continued*)

Engineer's Department in Johannesburg and by the major part of his career in the Head Office of the Anglovaal Mining Group from 1950. After retirement in 1980 he occupied the Chair of Mineral Economics at the University of the Witwatersrand for ten years. He has also been extensively involved in private consulting work for various mining concerns.

His research into the application of mathematical statistics in ore valuation started during his time in the Government Mining Engineer's office and was based on very practical analyses of the frequency distributions of gold sampling values and of the correlation patterns between the individual sample values, and also between the grade estimates of ore blocks based on limited sampling of the block perimeters and the subsequent sampling results from inside the ore blocks as they were being mined out. These distribution and correlation models led directly to the development of useful spatial patterns for the data and the implementation of the geostatistical Kriging and simulation techniques now in extensive use, particularly in mining circles worldwide.

During his career Danie received the MSc(Eng) and DSc(Eng) degrees from the University of the Witwatersrand as well as many honours, including three honorary degrees, including one from the Moscow State Mining University.

### *2.4.1 Building the Kriging Model*

We are going to start with a set of sample data, $\mathbf{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n)}\}^T$, with observed responses, $\mathbf{y} = \{y^{(1)}, y^{(2)}, \ldots, y^{(n)}\}^{\mathrm{T}}$, and we want to find an expression for a predicted value at a new point $\mathbf{x}$. There is a certain amount of matrix algebra in the derivation of the Kriging prediction, but the reader is encouraged to pay more attention to the concepts rather than the mathematical detail, which is explained in separate notes where necessary. We must begin with a slightly abstract concept, which is that we are going to view our observed responses as if they are from a stochastic process (even though they may in fact be from a deterministic computer code). We denote this using the set of random vectors

$$\mathbf{Y} = \begin{pmatrix} Y(\mathbf{x}^{(1)}) \\ \vdots \\ Y(\mathbf{x}^{(n)}) \end{pmatrix}.$$

This random field has a mean of $\mathbf{1}\mu$ ($\mathbf{1}$ is an $n \times 1$ column vector of ones). The random variables are correlated with each other using the basis function expression

$$\mathrm{cor}[Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(l)})] = \exp\left(-\sum_{j=1}^{k} \theta_j \mid x_j^{(i)} - x_j^{(l)} \mid^{p_j}\right). \tag{2.20}$$

From this we can construct an $n \times n$ correlation matrix of all the observed data:

$$\boldsymbol{\Psi} = \begin{pmatrix} \mathrm{cor}[Y(\mathbf{x}^{(1)}), Y(\mathbf{x}^{(1)})] & \cdots & \mathrm{cor}[Y(\mathbf{x}^{(1)}), Y(\mathbf{x}^{(n)})] \\ \vdots & \ddots & \vdots \\ \mathrm{cor}[Y(\mathbf{x}^{(n)}), Y(\mathbf{x}^{(1)})] & \cdots & \mathrm{cor}[Y(\mathbf{x}^{(n)}), Y(\mathbf{x}^{(n)})] \end{pmatrix}, \tag{2.21}$$

and a covariance matrix (see the following mathematical note)

$$\mathrm{Cov}(\mathbf{Y}, \mathbf{Y}) = \sigma^2 \boldsymbol{\Psi}. \tag{2.22}$$

This assumed correlation between the sample data reflects our expectation that an engineering function will behave in a certain way – most importantly that it will be smooth and continuous. This assumption has been a continuous thread throughout this chapter and Kriging is the most unassuming method yet, due to the greater number of model parameters.

**Mathematical Note: Covariance**

Covariance is a measure of the correlation between two or more sets of random variables

$$\mathrm{cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] \tag{2.23}$$

$$= E[XY] - \mu_X \mu_Y \tag{2.24}$$

where $\mu_X$ and $\mu_Y$ are the means of $X$ and $Y$ and $E$ is the expectation. From the covariance we can derive the correlation

$$\mathrm{cor}(X, Y) = \frac{\mathrm{cov}(X, Y)}{\sigma_X \sigma_Y} \tag{2.25}$$

where $\sigma_X$ and $\sigma_Y$ are the standard deviations of $X$ and $Y$.

For a vector of random variables,

$$\mathbf{Y} = \begin{pmatrix} Y^{(1)} \\ Y^{(2)} \\ \vdots \\ Y^{(n)} \end{pmatrix}$$

the covariance matrix is a matrix of covariances between elements of a vector:

$$\mathrm{cov}(\mathbf{Y}, \mathbf{Y}) = \begin{pmatrix} \mathrm{cov}(Y^{(1)}, Y^{(1)}) & \dots & \mathrm{cov}(Y^{(1)}, Y^{(n)}) \\ \vdots & \ddots & \vdots \\ \mathrm{cov}(Y^{(n)}, Y^{(1)}) & \dots & \mathrm{cov}(Y^{(n)}, Y^{(n)}) \end{pmatrix} \tag{2.26}$$

and, from Equation (2.25),

$$\mathrm{cov}(\mathbf{Y}, \mathbf{Y}) = \sigma_Y^2 \mathrm{cor}(\mathbf{Y}) \tag{2.27}$$

We now have a set of random variables ($\mathbf{Y}$) which are correlated in some way and this is described in our matrix $\boldsymbol{\Psi}$. These correlations depend on the absolute distance between the sample points $| x_j^{(i)} - x_j^{(l)} |$ and the parameters $p_j$ and $\theta_j$.

Figure 2.7 shows how $\exp(- | x_j^{(i)} - x_j |^{p_j})$ varies with the separation between the points. The correlation is intuitive insofar as when the two points move close together, $x_j^{(i)} - x_j \to 0$,

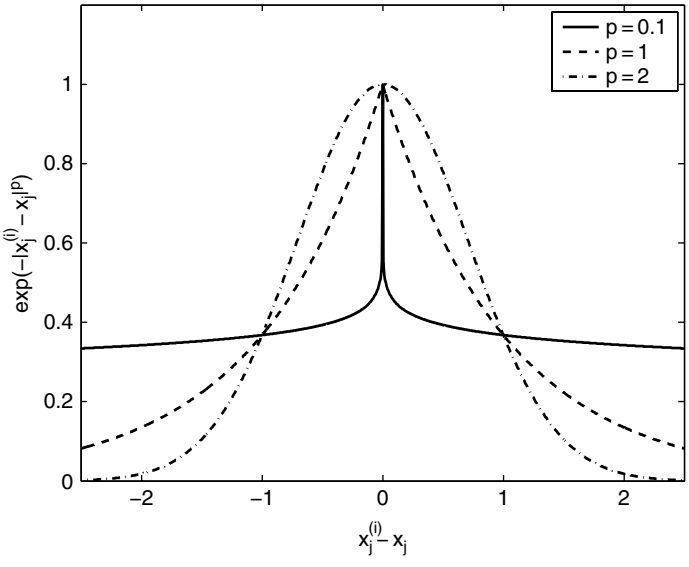



**Figure 2.7.** Correlations with varying **p**.

$\exp(-\mid x_j - x_j^{(i)} \mid^{p_j}) \to 1$ (the points show very close correlation and $Y(\mathbf{x}_j^{(i)}) = Y(\mathbf{x}_j)$), and when the points move apart, $x_j^{(i)} - x_j \to \infty$, $\exp(-\mid (x_j^{(i)} - x_j) \mid^{p_j}) \to 0$ (the points have no correlation). Three different correlations are shown in Figure 2.7: $p_j = 0.1$, 1 and 2. It is clear how this 'smoothness' parameter affects the correlation, with $p_j = 2$ we have a smooth correlation with a continuous gradient through $x_j^{(i)} - x_j = 0$. Reducing $p_j$ increases the rate at which the correlation initially drops as $\mid x_j^{(i)} - x_j \mid$ increases. With a very low value of $p_j = 0.1$, we are essentially saying that there is no immediate correlation between the two points and there is a near discontinuity between $Y(\mathbf{x}_j^{(i)})$ and $Y(\mathbf{x}_j)$.

Figure 2.8 shows how the choice of $\theta_j$ affects the correlation. It is essentially a width parameter that affects how far a sample point's influence extends. A low $\theta_j$ means that all points will have a high correlation, with $Y(x_j)$ being similar across our sample, while a high $\theta_j$ means that there is a significant difference between the $Y(x_j)$'s $\theta_j$. We can therefore consider $\theta_j$ as a measure of how 'active' the function we are approximating is. For example, if we performed a set of experiments where we measured the acceleration of a car for varying colour ($x_1$), fore and aft engine location ($x_2$) and engine size ($x_3$), we would expect to see $\theta_1 = 0$, since colour should have no effect on speed, $\theta_2$ would be slightly higher, since engine location would affect the traction and, because engine size would be the most dominant variable, $\theta_3$ would be the highest. Considering the 'activity' parameter $\theta_j$ in this way is helpful 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 are the most important variables and perhaps eliminate unimportant variables from future searches.

We mentioned the possibility of using Kriging to establish the order of importance of variables when considering the light aircraft wing weight function in Section 1.3.1.

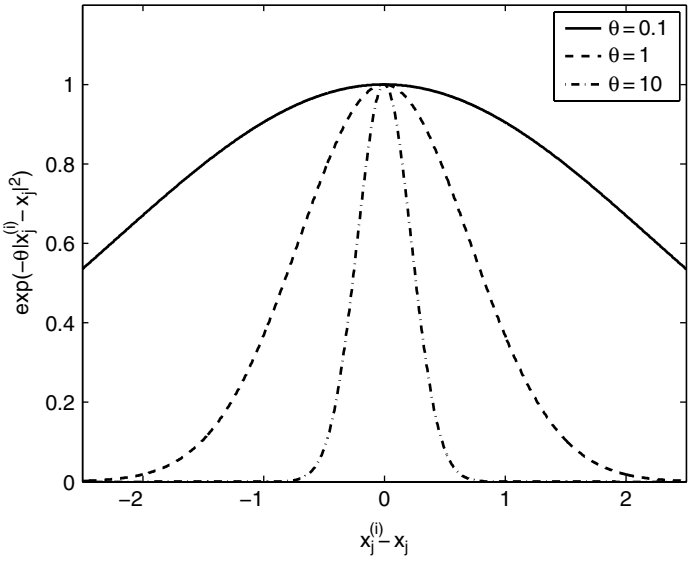



**Figure 2.8.** Correlations with varying $\boldsymbol{\theta}$.

**Table 2.1.** Ranking of the wing weight variables based on the Kriging $\boldsymbol{\theta}$ parameter

| Symbol | Parameter | $\theta$ |
|---|---|---|
| $N_z$ | Ultimate load factor | 0.1606 |
| $W_{dg}$ | Flight design gross weight (lb) | 0.1501 |
| $S_w$ | Wing area ($ft^2$) | 0.1194 |
| $tc$ | Aerofoil thickness to chord ratio | 0.1187 |
| $A$ | Aspect ratio | 0.1082 |
| $q$ | Dynamic pressure at cruise ($lb/ft^2$) | 0.0157 |
| $W_p$ | Paint weight ($lb/ft^2$) | 0.0086 |
| $\lambda$ | Taper ratio | 0.0018 |
| $W_{fw}$ | Weight of fuel in the wing (lb) | 0.0014 |
| $\Lambda$ | Quarter-chord sweep (deg) | 0.0012 |

Following the procedure we will outline in the remainder of this section, $\theta_j$'s have been estimated based on the same number of objective function values used in the screening study of Section 1.3.1. These estimates are used to rank the variables in Table 2.1. The $\boldsymbol{\theta}$ vector cannot tell us about interactions between the variables, but from Table 2.1 it is seen that the order of importance is roughly the same as that shown in Figure 1.2 and, more importantly, the same group of five variables with very little activity has been found.

We now know what our correlations mean, but how do we estimate the values of $\boldsymbol{\theta}$ and $\mathbf{p}$ and where does our observed data $\mathbf{y}$ come in? One answer is to choose $\boldsymbol{\theta}$ and $\mathbf{p}$ to maximize the likelihood of $\mathbf{y}$. Ultimately we hope that this will minimize the generalization error of the model.

In Equation (2.2) it was assumed that the errors $\epsilon$ are independently randomly distributed. In doing so we are effectively saying that our surrogate can exactly replicate the function and $\epsilon$ is purely due to errors in evaluating $\mathbf{y}$. In fact, these errors are likely to be due to error in the surrogate model and be largely a function of $\mathbf{x}$. For deterministic computer experiments without noise $\epsilon$ will be entirely due to surrogate model error and so the premise that $\epsilon$ is independently randomly distributed is completely false, albeit an often useful fiction. Here we are building a model which interpolates the data and we can eliminate $\epsilon$ because we assume that there is no error in $\mathbf{y}$ and we do not want to allow for modelling error. Our likelihood is therefore

$$L(\mathbf{Y}^{(1)}, \ldots, \mathbf{Y}^{(n)} \mid \mu, \sigma) = \frac{1}{(2\pi\sigma^2)^{n/2}} \exp\left[-\frac{\sum(\mathbf{Y}^{(i)} - \mu)^2}{2\sigma^2}\right],$$

which can be expressed in terms of the sample data as

$$L = \frac{1}{(2\pi\sigma^2)^{n/2}|\boldsymbol{\Psi}|^{1/2}} \exp\left[-\frac{(\mathbf{y} - \mathbf{1}\mu)^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\mu)}{2\sigma^2}\right]. \tag{2.28}$$

To simplify the likelihood maximization we take the natural logarithm to give

$$\ln(L) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\sigma^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}| - \frac{(\mathbf{y} - \mathbf{1}\mu)^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\mu)}{2\sigma^2}. \tag{2.29}$$

By taking derivatives of Equation (2.29) and setting to zero, we obtain maximum likelihood estimates (MLEs) for $\mu$ and $\sigma^2$:

$$\widehat{\mu} = \frac{\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\mathbf{y}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\mathbf{1}}, \tag{2.30}$$

$$\widehat{\sigma}^2 = \frac{(\mathbf{y} - \mathbf{1}\mu)^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\mu)}{n}. \tag{2.31}$$

These MLEs can now be substituted back into Equation (2.29) and constant terms removed to give what is known as the *concentrated ln-likelihood function*:

$$\ln(L) \approx -\frac{n}{2}\ln(\widehat{\sigma}^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}|. \tag{2.32}$$

The value of this function depends on our unknown parameters $\boldsymbol{\theta}$ and $\mathbf{p}$. We want to find values for these parameters which maximize Equation (2.32). This is rather difficult to achieve because we cannot differentiate the function like we did to find expressions for $\widehat{\mu}$ and $\widehat{\sigma}^2$. Instead, we use a numerical optimization technique. The concentrated ln-likelihood function is very quick to compute (if $n$ and $k$ are not too large) so we do not need to use a statistical model – we just search the function directly. A global search method such as a genetic algorithm or simulated annealing (see Section 3.1) usually produces the best results. Looking back at Figure 2.8, we see that there is as much change between $\theta = 0.1$ and $\theta = 1$ as between $\theta = 1$ and $\theta = 10$. It makes sense, therefore, to search for $\widehat{\boldsymbol{\theta}}$ on a logarithmic scale. We find that suitably wide search bounds are from $10^{-3}$ to $10^2$ (though this is by no means a hard and fast rule). The scaling of the observed data does not affect the

values of $\widehat{\boldsymbol{\theta}}$, but the scaling of the design space does. It is therefore, once again, advisable to always scale variable ranges to between zero and one in order that the values $\widehat{\theta}_j$ takes indicate the same degree of activity from problem to problem.

While tuning $\widehat{\mathbf{p}}$ is advantageous in producing accurate predictions in many problems, in order to reduce the complexity of our code, in our examples we have fixed this model parameter at $\widehat{\mathbf{p}} = \mathbf{2}$ and will only tune $\widehat{\boldsymbol{\theta}}$.

### Implementing an MLE of the Model Parameters

The matrix algebra involved in calculating the likelihood is the most computationally intensive part of the Kriging process and so care must be taken that the computer code is as efficient as possible.

Since $\boldsymbol{\Psi}$ is symmetric, only half of the matrix needs to be computed before adding to its tranpose. To calculate the ln-likelihood, a number of matrix inversions are required. It is quickest to perform one Cholesky factorization (`chol.m`) and then use backward and forward substitution for each inverse.

The Cholesky factorization will only work for a positive-definite matrix, which $\boldsymbol{\Psi}$ is. However, if $\boldsymbol{\Psi}$ becomes close to singular, which happens when $\mathbf{x}^{(i)}$'s are densely packed, the Cholesky factorization may fail. We could use an LU-decomposition in such cases, which would yield an answer, but it would be an unreliable one. In situations where $\boldsymbol{\Psi}$ is close to singular, the best options are to either use regression (see Chapter 6) or, as we do here, attach a poor likelihood value to the parameters which produced the near singular matrix and so divert the MLE search of the parameters to better conditioned $\boldsymbol{\Psi}$'s. The *MATLAB* function `chol.m` provides a convenient way of implementing this strategy. Using `[U,p]=chol(Psi)`, if `p` $\neq 0$, `Psi` is non-positive-definite (or in our case is close to singular, so that for all intents and purposes it is non-positive-definite), and we simply give the negative of the concentrated ln-likelihood a very high value in such cases.

Another subtlety of calculating the concentrated ln-likelihood is that `det(Psi)` $\rightarrow 0$ for poorly conditioned matrices, and it is therefore advisable to use twice the sum of the natural logarithms of the diagonal of the Cholesky factorization when calculating $\ln(|\boldsymbol{\Psi}|)$ in Equation (2.32).

The following *MATLAB* code constructs the correlation matrix $\boldsymbol{\Psi}$ and returns the negative of the concentrated ln-likelihood for given values of $\boldsymbol{\theta}$.

```
function [NegLnLike,Psi,U]=likelihood(x)
% Calculates the negative of the concentrated ln-likelihood
%
% Inputs:
%       x - vetor of log(theta) parameters
%
% Global variables used:
%       ModelInfo.X - n x k matrix of sample locations
%       ModelInfo.y - n x 1 vector of observed data
```

(*continued*)

```
%
% Outputs:
%       NegLnLike – concentrated ln – likelihood * – 1 for minimizing
%       Psi – correlation matrix
%       U – Cholesky factorization of correlation matrix

global ModelInfo
X=ModelInfo.X;
y=ModelInfo.y;
theta=10.^x;
n=size(X,1);
one=ones(n,1);

% Pre–allocate memory
Psi=zeros(n,n);
% Build upper half of correlation matrix
for i=1:n
    for j=i+1:n
            Psi(i,j)=exp(–sum(theta.*(X(i,:)–X(j,:)).^2));
    end
end

% Add upper and lower halves and diagonal of ones plus
% small number to reduce ill conditioning
Psi=Psi+Psi'+eye(n)+eye(n).*eps;

% Cholesky factorization
[U,p]=chol(Psi);

% Use penalty if ill–conditioned
if p > 0
    NegLnLike=1e4;
else

    % Sum lns of diagonal to find ln(det(Psi))
    LnDetPsi=2*sum(log(abs(diag(U))));

    % Use back–substitution of Cholesky instead of inverse
    mu=(one'*(U\(U'\y)))/(one'*(U\(U'\one)));
    SigmaSqr=((y–one*mu)'*(U\(U'\ (y–one*mu))))/n;
    NegLnLike=–1*(–(n/2)*log(SigmaSqr) – 0.5*LnDetPsi);
end
```

### Example: Training the Kriging Model to Fit the Branin Function

This example details the training of a Kriging model using the functions provided in on the book website. To train the Kriging model we first need our sample data. For the purposes of this tutorial we will use values from a simple test function known as the Branin function (see the appendix, Section A.2). The function has two variables ($k = 2$) and we will use

a sampling plan of 20 points ($n = 20$). The reader may wish to experiment with different values of $n$ and see how the quality of the model is affected (as we have done in Section 2.1.3).

Working through the *MATLAB* script below, as explained in Section 2.3, we first need to define `ModelInfo` as a global variable in order to make it available to all functions. We then define the number of variables and number of sample points. This information is then fed into the `bestlh.m` function to create the sampling plan. This plan is stored in the variable `X` which lives inside the structure `ModelInfo`. Each row of the sampling plan is fed into `branin.m` to find the observed data, which is stored in `ModelInfo.y`.

Now that we have the observed data we can set up the tuning of the Kriging model. We first set upper and lower bounds for the search of `Theta`. We are searching for the log of $\boldsymbol{\theta}$ so 2 and $-3$ refer to the upper and lower limits of $10^2$ and $10^{-3}$. The likelihood can now be searched using a genetic algorithm. After searching the likelihood, we then run `likelihood.m` again using the MLE values for $\boldsymbol{\theta}$ so that we can store the correlation matrix $\boldsymbol{\Psi}$ and its Cholesky factorization in `ModelInfo`, ready to be used when we make predictions using the Kriging model.

```
global ModelInfo
% Number of variables
k=2;
% Number of sample points
n=20;

% Create sampling plan
ModelInfo.X=bestlh(n,k,50,20);

% Calculate observed data
for i=1:n
  ModelInfo.y(i,1)=branin(ModelInfo.X(i,:));
end

% Set upper and lower bounds for search of log theta
UpperTheta=ones(1,k).*2;
LowerTheta=ones(1,k).*-3;

% Run GA search of likelihood
[ModelInfo.Theta,MinNegLnLikelihood]=...
ga(@likelihood,k,[],[],[],[],LowerTheta,UpperTheta);

% Put Cholesky factorization of Psi, into ModelInfo
[NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood(ModelInfo.Theta);
```

This search of the Kriging model parameters yields the values $\log_{10}\widehat{\theta}_1 = 0.7686$ and $\log_{10}\widehat{\theta}_2 = -0.6458$. We can immediately see, before making any predictions using the Kriging model, that, based on our 20 samples, the first variable is the more dominant, with $\widehat{\theta}_1$ much higher than $\widehat{\theta}_2$. With the model parameters found, we are now ready to make predictions at new points.

### 2.4.2 Kriging Prediction

In this section we use our Kriging correlation to predict new values based on the observed data. Our derivation of the Kriging predictor is taken from Jones, (2001) and is, we feel, the most straightforward and intuitive way of explaining the way predictions are made.

We have chosen correlation parameters which maximize the likelihood of the observed data, $\mathbf{y}$. A new prediction, $\widehat{y}$ at $\mathbf{x}$, should be consistent with the observed data and therefore with the correlation parameters we have found. Hence we choose a prediction which maximizes the likelihood of the sample data *and* the prediction, given our correlation parameters. To achieve this we first augment the observed data $\mathbf{y}$ with the new prediction $\widehat{y}$, the value of which is to be determined, to give the vector $\widetilde{\mathbf{y}} = \{\mathbf{y}^{\mathrm{T}},\ \widehat{y}\}^{\mathrm{T}}$. We also define a vector of correlations between the observed data and our new prediction:

$$\boldsymbol{\psi} = \begin{pmatrix} \mathrm{cor}[Y(\mathbf{x}^{(1)}),\, Y(\mathbf{x})] \\ \vdots \\ \mathrm{cor}[Y(\mathbf{x}^{(n)}),\, Y(\mathbf{x})] \end{pmatrix} = \begin{pmatrix} \psi^{(1)} \\ \vdots \\ \psi^{(n)} \end{pmatrix}. \tag{2.33}$$

Now we can construct an augmented correlation matrix:

$$\widetilde{\boldsymbol{\Psi}} = \begin{pmatrix} \boldsymbol{\Psi} & \boldsymbol{\psi} \\ \boldsymbol{\psi}^{\mathrm{T}} & 1 \end{pmatrix}. \tag{2.34}$$

Note that the last element of $\widetilde{\boldsymbol{\Psi}}$ is one. This is a continuation of the leading diagonal of ones in $\boldsymbol{\Psi}$, which represent the correlation of a point with itself where $|\mathbf{x}^{(i)} - \mathbf{x}^{(i)}| = 0$ and so $\mathrm{cor}[Y(\mathbf{x}^{(i)}), Y(\mathbf{x}^{(i)})] = 1$.

The ln-likelihood of the augmented data is

$$\ln(L) = -\frac{n}{2}\ln(2\pi) - \frac{n}{2}\ln(\widehat{\sigma}^2) - \frac{1}{2}\ln|\widetilde{\boldsymbol{\Psi}}| - \frac{(\widetilde{\mathbf{y}} - \mathbf{1}\widehat{\mu})^{\mathrm{T}}\widetilde{\boldsymbol{\Psi}}^{-1}(\widetilde{\mathbf{y}} - \mathbf{1}\widehat{\mu})}{2\widehat{\sigma}^2}, \tag{2.35}$$

and only the last term of this depends on $\widehat{y}$; so we need only consider this term in our maximization. Substituting in expressions for $\widetilde{\mathbf{y}}$ and $\widetilde{\boldsymbol{\Psi}}$ gives

$$\ln(L) \approx \frac{-\begin{pmatrix} \mathbf{y} - \mathbf{1}\widehat{\mu} \\ \widehat{y} - \widehat{\mu} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \boldsymbol{\Psi} & \boldsymbol{\psi} \\ \boldsymbol{\psi}^{\mathrm{T}} & 1 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} - \mathbf{1}\widehat{\mu} \\ \widehat{y} - \widehat{\mu} \end{pmatrix}}{2\widehat{\sigma}^2}. \tag{2.36}$$

To maximize this equation, we must first find the inverse of $\widetilde{\boldsymbol{\Psi}}$ using the partitioned inverse method of Theil (1971) (see the following mathematical note):

$$\widetilde{\boldsymbol{\Psi}}^{-1} = \begin{pmatrix} \boldsymbol{\Psi}^{-1} + \boldsymbol{\Psi}^{-1}\boldsymbol{\psi}(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})^{-1}\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1} & -\boldsymbol{\Psi}^{-1}\boldsymbol{\psi}(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})^{-1} \\ -(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})^{-1}\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1} & (1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})^{-1} \end{pmatrix}. \tag{2.37}$$

This can be substituted into Equation (2.36) and terms without $\widehat{y}$ removed to give

$$\ln(L) \approx \left( \frac{-1}{2\widehat{\sigma}^2(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})} \right) (\widehat{y} - \widehat{\mu})^2 + \left( \frac{\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu})}{\widehat{\sigma}^2(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})} \right) (\widehat{y} - \widehat{\mu}). \tag{2.38}$$

The maximum of this quadratic function of $\widehat{y}$ is then found by differentiating with respect to $\widehat{y}$ and setting to zero:

$$\left( \frac{-1}{\widehat{\sigma}^2(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})} \right) (\widehat{y} - \widehat{\mu}) + \left( \frac{\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu})}{\widehat{\sigma}^2(1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi})} \right) = 0. \tag{2.39}$$

Thus, our MLE for $\widehat{y}$ is

$$\widehat{y}(\mathbf{x}) = \widehat{\mu} + \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu}). \tag{2.40}$$

It is not immediately obvious how Equation (2.40) relates to the radial basis function Equation (2.15). In Equation (2.40) our basis functions are contained in the vector $\boldsymbol{\psi}$. These are centred around the $n$ sample points and are added to a mean base term $\mu$ with weightings $\mathbf{w} = \boldsymbol{\Psi}(\mathbf{y} - \mathbf{1}\mu)$. The model is constructed in such a way that the prediction goes through all the data points (it interpolates the data): if we make a prediction at $\mathbf{x}^{(i)}$, $\boldsymbol{\psi}$ is the $i$th column of $\boldsymbol{\Psi}$, and this means that $\boldsymbol{\psi}\boldsymbol{\Psi}^{-1}$ is the $i$th unit vector. Thus $\widehat{y}(\mathbf{x}) = \widehat{\mu} + y^{(i)} - \widehat{\mu} = y^{(i)}$.

**Mathematical Note: Partitioned Inverse**

For a nonsingular $n \times n$ matrix $\mathbf{A}$ there is a unique $n \times n$ inverse matrix $\mathbf{A}^{-1}$ which satisfies $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$ ($\mathbf{I}$ is a matrix of zeros with ones down the leading diagonal – the identity matrix). Therefore, given the nonsingular matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{P}_1 & \mathbf{R}_1 \\ \mathbf{R}_1^T & \mathbf{Q}_1 \end{pmatrix},$$

where $\mathbf{P}$ and $\mathbf{Q}$ are nonsingular submatrices, we wish to solve

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{I} = \begin{pmatrix} \mathbf{P}_2 & \mathbf{R}_2 \\ \mathbf{R}_2^{\mathrm{T}} & \mathbf{Q}_2 \end{pmatrix} \begin{pmatrix} \mathbf{P}_1 & \mathbf{R}_1 \\ \mathbf{R}_1^{\mathrm{T}} & \mathbf{Q}_1 \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}$$

($\mathbf{0}$ is a matrix of zeros). This splits into the four submatrix equations:

$$\mathbf{P}_2\mathbf{P}_1 + \mathbf{R}_2\mathbf{R}_1^{\mathrm{T}} = \mathbf{I}, \tag{2.41}$$

$$\mathbf{P}_2\mathbf{R}_1 + \mathbf{R}_2\mathbf{Q}_1 = \mathbf{0}, \tag{2.42}$$

$$\mathbf{R}_2^{\mathrm{T}}\mathbf{P}_1 + \mathbf{Q}_2\mathbf{R}_1^{\mathrm{T}} = \mathbf{0}, \tag{2.43}$$

$$\mathbf{R}_2^{\mathrm{T}}\mathbf{R}_1 + \mathbf{Q}_2\mathbf{Q}_1 = \mathbf{I}. \tag{2.44}$$

(*continued*)

From Equation (2.43) $\mathbf{R}_2^{\mathrm{T}} = -\mathbf{Q}_2\mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}$, which is substituted into Equation (2.44) to give $\mathbf{Q}_2(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1) = \mathbf{I}$. Thus

$$\mathbf{Q}_2 = (\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1} \tag{2.45}$$

and

$$\mathbf{R}_2^{\mathrm{T}} = -(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}\mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}, \tag{2.46}$$

$$\mathbf{R}_2 = -\mathbf{P}_1^{-1}\mathbf{R}_1(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}. \tag{2.47}$$

By substituting Equation (2.45) into Equation (2.41), $\mathbf{P}_2\mathbf{P}_1 - \mathbf{P}_1^{-1}\mathbf{R}_1(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}\mathbf{R}_1^{\mathrm{T}} = \mathbf{I}$ and so

$$\mathbf{P}_2 = \mathbf{P}_1^{-1} + \mathbf{P}_1^{-1}\mathbf{R}_1(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}\mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}. \tag{2.48}$$

Putting Equations (2.45), (2.46), (2.47) and (2.48) together, $\mathbf{A}^{-1}$ can now be expressed as

$$\begin{pmatrix} \mathbf{P}_1^{-1} + \mathbf{P}_1^{-1}\mathbf{R}_1(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}\mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1} & -\mathbf{P}_1^{-1}\mathbf{R}_1(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1} \\ -(\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1}\mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1} & (\mathbf{Q}_1 - \mathbf{R}_1^{\mathrm{T}}\mathbf{P}_1^{-1}\mathbf{R}_1)^{-1} \end{pmatrix}.$$

### Implementing a Kriging Prediction

With the model parameters estimated, the process of making a prediction at a new point is relatively straightforward.

The Kriging prediction is based on a large quantity of data and, rather than passing these explicitly to the prediction code or recalculating, it is easiest to bundle these up in a data structure and, as already mentioned, make this a global variable. This structure can now be accessed by the following function, which will return a prediction at a new point.

```
function f = pred(x)
% Calculates a Kriging prediction at x
%
% Inputs:
%       x – 1 x k vetor of design variables
%
% Global variables used:
%       ModelInfo.X – n x k matrix of sample locations
%       ModelInfo.y – n x 1 vector of observed data
%       ModelInfo.Theta – 1 x k vector of log(theta)
%       ModelInfo.U – n x n Cholesky factorization of Psi
%
% Outputs:
%       f – scalar Kriging prediction
```

(*continued*)

```
global ModelInfo
% Extract variables from data structure
% slower, but makes code easier to follow
X=ModelInfo.X;
y=ModelInfo.y;
theta=10.^ModelInfo.Theta;
U=ModelInfo.U;

% Calculate number of sample points
n=size(X,1);

% Vector of ones
one=ones(n,1);

% Calculate mu
mu=(one'*(U\(U'\y)))/(one'*(U\(U'\one)));

% Initialize psi to vector of ones
psi=ones(n,1);

% Fill psi vector
for i=1:n
  psi(i)=exp(-sum(theta.*abs(X(i,:)-x).^2));
end

% Calculate prediction
f=mu+psi'*(U\(U'\(y-one*mu)));
```

**Example: Making Predictions of the Branin Function with the Kriging Model**

We will now predict the shape of the Branin function using the observed data and parameters from the previous example. Following the *MATLAB* code below, the process is quite simple. We make predictions using the `pred.m` function. Here we are going to plot out our function over a $21 \times 21$ grid, so we start by making a vector of 21 points between 0 and 1. We then need two **`for`** loops, one nested inside the other to allow us to create a $21 \times 21$ matrix of predictions. Because the Branin function is very quick to evaluate, here we can also create a matrix of true function values to compare with our predictions.

```
Xplot=0:1/20:1;
for i=1:21
    for j=1:21
        BraninPred(j,i)=pred([Xplot(i) Xplot(j)]);
        BraninTrue(j,i)=branin([Xplot(i) Xplot(j)]);
    end
end
```

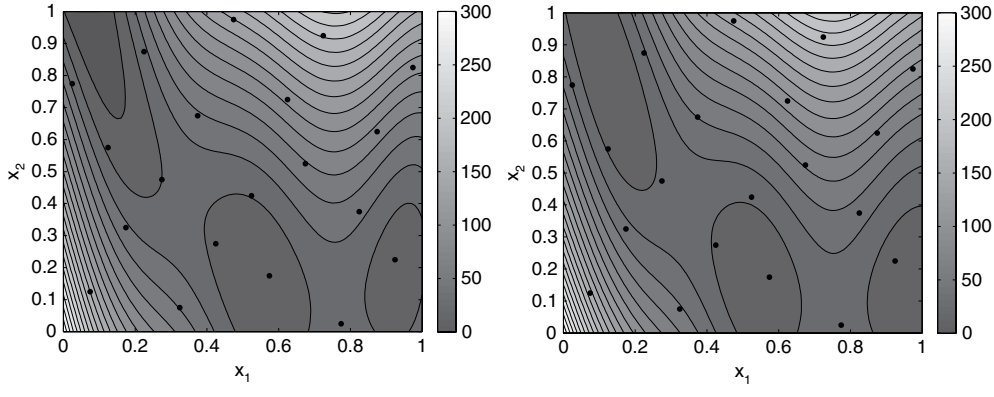



**Figure 2.9.** The Kriging prediction of the Branin function based on 20 sample points (left) compared with the true Branin function (right).

With the predictions and true values calculated, we can now plot the results to see the form of the function and how close the prediction is to the true Branin function. Figure 2.9 shows that the Kriging prediction is, in fact, a very close representation of the true Branin function.

Although Kriging is rather more complicated than other radial basis function methods, and so given its own section, it is, nevertheless, simply a sum of weighted basis functions. The 20 basis functions (with Gaussian form) used to construct the prediction of the Branin function are shown in Figure 2.10 and, after multiplication by their respective weightings, Figure 2.11. These weighted bases are summed and added to the MLE of the mean, $\mu$, to produce the prediction in Figure 2.9.

## 2.5 Support Vector Regression

Support vector regression (SVR) allows us to specify or calculate a margin $\varepsilon$ within which we are willing to accept errors in the sample data without them affecting the SVR prediction $(\widehat{f})$. This may be useful if our sample data has an element of random error due to, for example, finite mesh size, since through a mesh sensitivity study we could calculate a suitable value for $\varepsilon$. If the data is derived from a physical experiment, the accuracy of the measurements taken could be used to specify $\varepsilon$, e.g. if measurements are taken using a ruler $\varepsilon$ might be set at $\pm 0.5$ mm and the SVR would only consider deviations greater than this when fitting a prediction. To demonstrate this, we have sampled our one-dimensional test function (see the Appendix, Section A.1) at 21 evenly spaced points, but included a random error, with a mean of zero and a variance and standard deviation of one, to simulate physical or computer experiment error. With this known standard deviation, we have chosen $\varepsilon = 1$. The resulting SVR prediction $(\widehat{f})$ is shown in Figure 2.12. The sample points which lie within the $\pm\varepsilon$ band (known as the $\varepsilon$-tube) are ignored, with the predictor being defined entirely by those that lie on or outside this region: the *support vectors*.

SVR is usually considered as a special case of support vector machines (SVMs), with the majority of the literature concentrating on the use of SVMs for classification rather than SVR

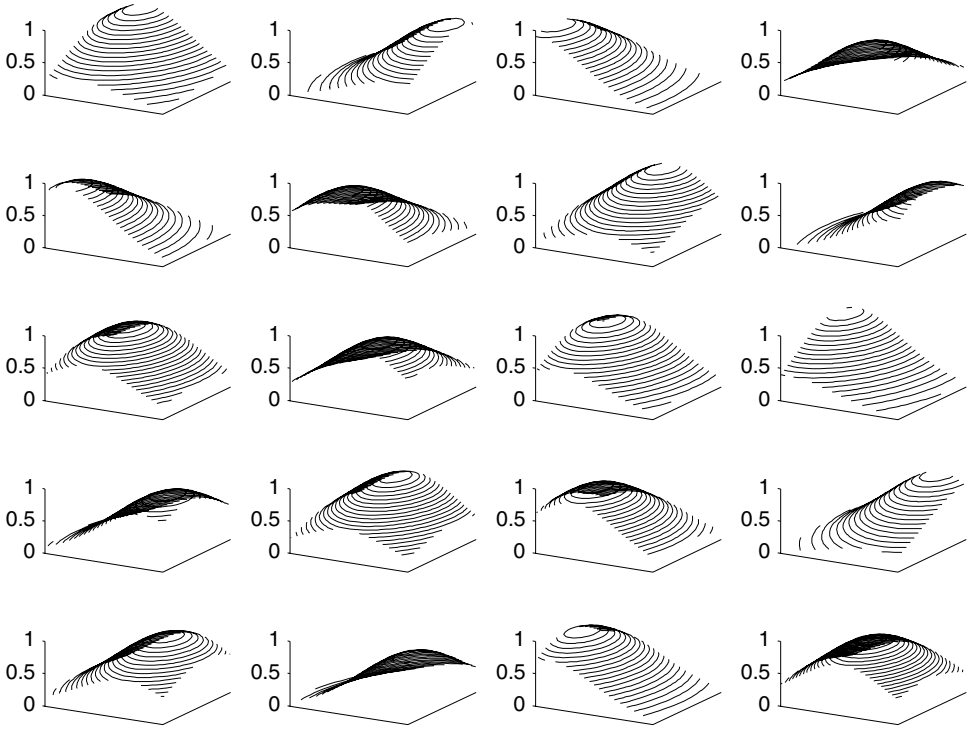


**Figure 2.10.** The 20 basis functions, with each plot showing the value of $\psi^{(i)}$ within the bounds of the design space $\mathbf{D}$. At the the centre of the basis function ($\mathbf{x}^{(i)}$) $\psi^{(i)} = 1$ and as $|\mathbf{x}^{(i)} - \mathbf{x}|$ gets larger the correlation reduces and $\psi^{(i)} \to 0$.

for function prediction. In engineering design there is usually more interest in predicting the actual value of an output rather than classifying sets of data. As such, we will consider SVR as an extension to radial basis function methods rather than SVMs. In fact, as will be seen later in this section, the interpolating radial basis function (RBF) models of Sections 2.3 and 2.4 occur as a special case of SVR.

The basic form of the SVR prediction is the familiar sum of basis functions $\psi^{(i)}$, with weightings $w^{(i)}$, added to a base term $\mu$. All are calculated in different ways to their counterparts in Sections 2.3 and 2.4, yet contribute to the prediction in the same way:

$$\widehat{f}(\mathbf{x}) = \mu + \sum_{i=1}^{n} w^{(i)} \psi(\mathbf{x}, \mathbf{x}^{(i)}). \tag{2.49}$$

### *2.5.1 The Support Vector Predictor*

To simplify matters, we will begin by considering linear regression, i.e. $\psi(.) = \mathbf{x}$:

$$\widehat{f}(\mathbf{x}) = \mu + \mathbf{w}^{\mathrm{T}}\mathbf{x}. \tag{2.50}$$

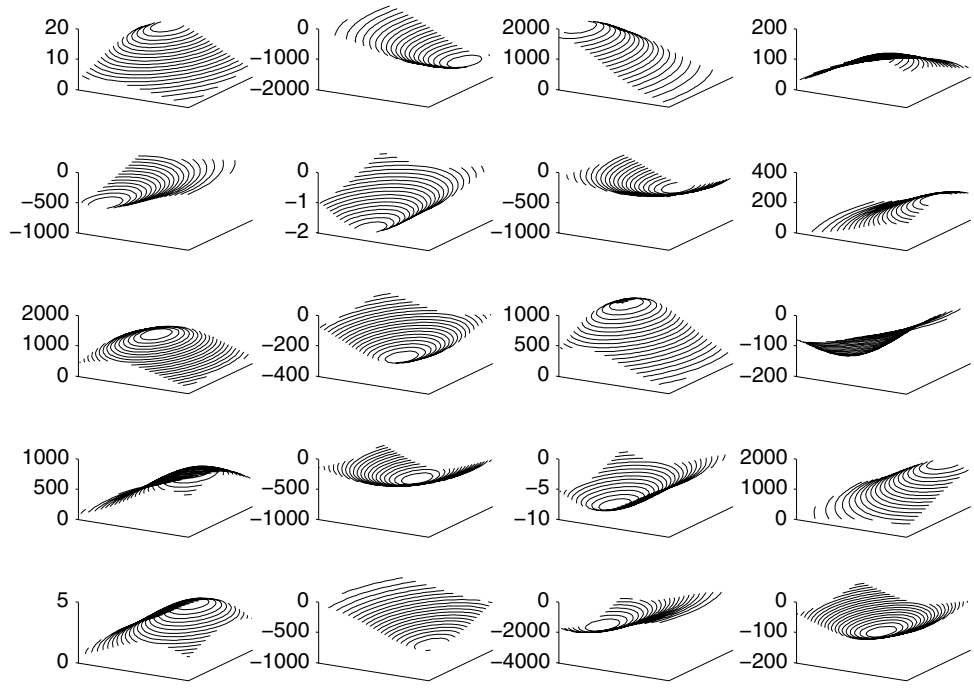



**Figure 2.11.** The 20 basis functions after multiplication by the weightings $\mathbf{w} = \boldsymbol{\Psi}(\mathbf{y} - \mathbf{1}\mu)$. These weighted functions are added to $\widehat{\mu}$ to give the prediction in Figure 2.9.

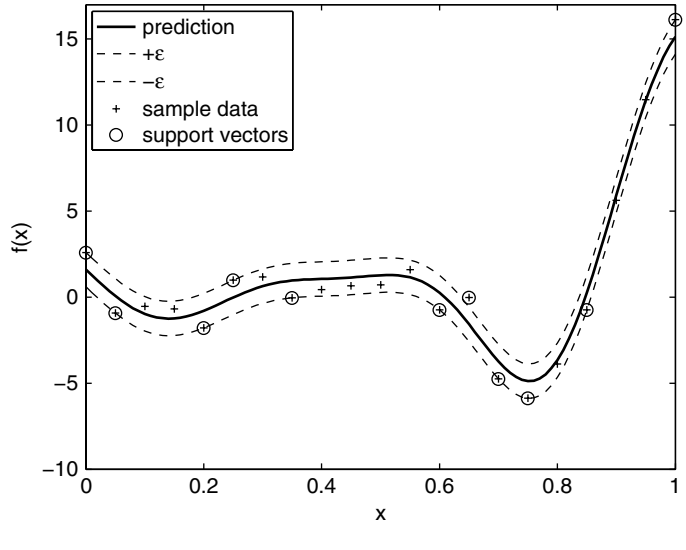



**Figure 2.12.** A SVR prediction using a Gaussian kernel through the one-dimensional test function with added noise. The area between the dashed lines is known as the $\varepsilon$-tube.

To produce a prediction which generalizes well, we wish to find the function with, at most, $\varepsilon$ deviations from $\mathbf{y}$ and, at the same time, minimum complexity.[5] We can minimize the model complexity by minimizing the vector norm $|\mathbf{w}|^2$, that is, the flatter the function the simpler it is and the more likely it is to generalize well. Cast as a constrained convex quadratic optimization problem, we wish to

$$\begin{aligned} &\text{minimize } \frac{1}{2}|\mathbf{w}|^2 \\ &\text{subject to } \begin{cases} y^{(i)} - \mathbf{w}\cdot\mathbf{x}^{(i)} - \mu \leq \varepsilon \\ \mathbf{w}\cdot\mathbf{x}^{(i)} + \mu - y^{(i)} \leq \varepsilon. \end{cases} \end{aligned} \tag{2.51}$$

Note that the constraints on this optimization problem assume that a function $\widehat{f}(\mathbf{x})$ exists that approximates all $y^{(i)}$ with precision $\varepsilon$. Such a solution may not actually exist and it is also likely that better predictions will be obtained if we allow for the possibility of outliers. This is achieved by introducing slack variables, $\xi^+$ for $\widehat{f}(\mathbf{x}^{(i)}) - y(\mathbf{x}^{(i)}) > \varepsilon$ and $\xi^-$ for $y(\mathbf{x}^{(i)}) - \widehat{f}(\mathbf{x}^{(i)}) > \varepsilon$. We now

$$\begin{aligned} &\text{minimize } \frac{1}{2}|\mathbf{w}|^2 + C\frac{1}{n}\sum_{i=1}^{n}(\xi^{+(i)} + \xi^{-(i)}) \\ &\text{subject to } \begin{cases} y^{(i)} - \mathbf{w}\cdot\mathbf{x}^{(i)} - \mu \leq \varepsilon + \xi^{+(i)} \\ \mathbf{w}\cdot\mathbf{x}^{(i)} + \mu - y^{(i)} \leq \varepsilon + \xi^{-(i)} \\ \xi^{+(i)}, \xi^{-(i)} \geq 0. \end{cases} \end{aligned} \tag{2.52}$$

From Equation (2.52) we see that the minimization is a trade-off between model complexity and the degree to which errors larger than $\varepsilon$ are tolerated. This trade-off is governed by the user defined constant $C \geq 0$ ($C = 0$ would correspond to a flat function through $\mu$). This method of tolerating errors is known as the $\varepsilon$-insensitive loss function and is shown in Figure 2.13. Points that lie inside the $\varepsilon$-tube (the $\varepsilon$-tube is shown in Figure 2.12) will have no loss associated with them, while points outside have a loss which increases linearly away from the prediction with the rate determined by $C$.

The constrained optimization problem of Equation (2.52) is solved by introducing Lagrange multipliers, $\eta^{+(i)}$, $\eta^{-(i)}$, $\alpha^{+(i)}$ and $\alpha^{-(i)}$, to give the Lagrangian

$$\begin{aligned} L = &\frac{1}{2}|\mathbf{w}|^2 + C\frac{1}{n}\sum_{i=1}^{n}(\xi^{+(i)} + \xi^{-(i)}) - \sum_{i=1}^{n}(\eta^{+(i)}\xi^{+(i)} + \eta^{-(i)}\xi^{-(i)}) \\ &- \sum_{i=1}^{n}\alpha^{+(i)}\left(\varepsilon + \xi^{+(i)} - y^{(i)} + \mathbf{w}\cdot\mathbf{x}^{(i)} + \mu\right) \\ &- \sum_{i=1}^{n}\alpha^{-(i)}\left(\varepsilon + \xi^{-(i)} + y^{(i)} - \mathbf{w}\cdot\mathbf{x}^{(i)} - \mu\right). \end{aligned} \tag{2.53}$$

[5] The requirement of minimizing model complexity to improve generalization derives from Occam's Razor (also Occam's Razor): *non sunt entia multiplicanda praeter necessitatem*, which translates to 'entities should not be multiplied beyond necessity' or, in lay terms, 'all things being equal, the simplest solution tends to be the best one'. This principle is attributed to William of Ockham, a 14th century English Franciscan Philosopher.

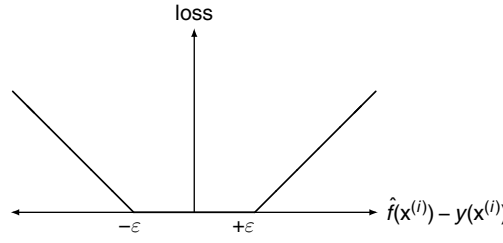



**Figure 2.13.** The $\varepsilon$-insensitive loss function.

which must be minimized with respect to $\mathbf{w}$, $\mu$ and $\xi^{\pm}$ (the *primal variables*) and maximized with respect to $\eta^{\pm(i)}$ and $\alpha^{\pm(i)}$ (the *dual variables*), where $\eta^{\pm(i)}$, $\alpha^{\pm(i)} \geq 0$ ($\pm$ refers to both $+$ and $-$ variables). For active constraints, the corresponding $(\alpha^{-(i)} + \alpha^{+(i)})$ will become the support vectors (the circled points in Figure 2.12), whereas for inactive constraints $(\alpha^{-(i)} + \alpha^{+(i)}) = 0$ and the corresponding $y^{(i)}$ will be excluded from the prediction.

The minimization of $L$ with respect to the primal variables and maximization with respect to the dual variables means we are looking for a saddle point, at which the derivatives with respect to the primal variables must vanish:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)}) \mathbf{x}^{(i)} = 0, \tag{2.54}$$

$$\frac{\partial L}{\partial \mu} = \sum_{i=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)}) = 0, \tag{2.55}$$

$$\frac{\partial L}{\partial \xi^{+}} = \frac{C}{n} - \alpha^{+(i)} - \eta^{-(i)} = 0, \tag{2.56}$$

$$\frac{\partial L}{\partial \xi^{-}} = \frac{C}{n} - \alpha^{-(i)} - \eta^{-(i)} = 0. \tag{2.57}$$

From Equation (2.54) we obtain

$$\mathbf{w} = \sum_{i=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)}) \mathbf{x}^{(i)} \tag{2.58}$$

and, by substituting into Equation (2.50), the SVR prediction is found to be

$$\widehat{f}(\mathbf{x}) = \mu + \sum_{i=1}^{n} \left(\alpha^{+(i)} - \alpha^{-(i)}\right) \left(\mathbf{x}^{(i)} \cdot \mathbf{x}\right). \tag{2.59}$$

### *2.5.2 The Kernel Trick*

Until now we have always considered our data $\mathbf{X}$ to exist in real coordinate space, which we will denote as $\chi \in \mathbb{R}^k$. We wish to extend equation (2.59) beyond linear regression to basis functions (known in the SV literature as *kernels*), which can capture more complicated

landscapes. To do this we say that $\mathbf{x}$ in Equation (2.59) is in *feature space*, denoted as $\mathcal{H}$, which may not coincide with $\mathbb{R}^k$. We can define a mapping between these two spaces, $\boldsymbol{\phi} : \chi \mapsto \mathcal{H}$. We are only dealing with the *inner product* (also known as the *dot product* or *scalar product*) $\mathbf{x} \cdot \mathbf{x}$ and, conveniently, $\mathbf{x} \cdot \mathbf{x} = \boldsymbol{\phi} \cdot \boldsymbol{\phi}$. We can actually choose the mapping $\boldsymbol{\phi}$ and can use our basis functions from Sections 2.3 and 2.4 by using $\boldsymbol{\psi} = \boldsymbol{\phi} \cdot \boldsymbol{\phi}$:

$$\widehat{f}(\mathbf{x}) = \mu + \sum_{i=1}^{n} \left(\alpha^{+(i)} - \alpha^{-(i)}\right) \psi^{(i)}. \tag{2.60}$$

The maths so far will still hold true, so long as certain conditions are satisfied:

1. $\psi$ is continuous,
2. $\psi$ is symmetric, i.e. $\psi(\mathbf{x}, \mathbf{x}^{(i)}) = \psi(\mathbf{x}^{(i)}, \mathbf{x})$, and
3. $\psi$ is positive definite, which means the correlation matrix $\boldsymbol{\Psi} = \boldsymbol{\Psi}^{\mathrm{T}}$ and has nonnegative eigenvalues.

Basis functions satisfying these conditions are known as *Mercer kernels*. Popular choices for $\psi$ are:

$$\begin{aligned}
&\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}), && \text{(linear)}\\
&\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^d, && (d \text{ degree homogeneous polynomial})\\
&\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \left(\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} + c\right)^d, && (d \text{ degree inhomogeneous polynomial})\\
&\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(\frac{-|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}|^2}{\sigma^2}\right), \text{and} && \text{(Gaussian)}\\
&\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\sum_{l=1}^{k} \theta_l |\mathrm{x}_l^{(i)} - \mathrm{x}_l^{(j)}|^{p_l}\right). && \text{(Kriging)}
\end{aligned} \tag{2.61}$$

Whichever form of $\psi$ is chosen, the method of finding the support vectors remains unchanged and, after this brief aside, we now attend to that task.

### 2.5.3 Finding the Support Vectors

With the kernel substitution made, the support vectors are found by substituting Equations (2.54), (2.55), (2.56) and (2.57) into Equation (2.53) to eliminate $\eta^{-(i)}$ and $\eta^{+(i)}$, and finally to obtain the dual variable optimization problem:

$$\begin{aligned}
&\text{maximize} \begin{cases} \frac{1}{2} \sum_{i,j=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)})(\alpha^{+(j)} - \alpha^{-(j)}) \boldsymbol{\Psi}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ -\varepsilon \frac{1}{2} \sum_{i=1}^{n} (\alpha^{+(i)} + \alpha^{-(i)}) + \sum_{i=1}^{n} y^{(i)} (\alpha^{+(i)} - \alpha^{-(i)}) \end{cases} \\
&\text{subject to} \begin{cases} \sum_{i=1}^{n} (\alpha^{+(i)} - \alpha^{-(i)}) = 0 \\ \alpha^{\pm(i)} \in [0, C/n]. \end{cases}
\end{aligned} \tag{2.62}$$

The formulation of quadratic programming algorithms used to solve problems such as (2.62) is outside of the scope of this book. However, the following code shows how to find support vectors by solving (2.62) using *MATLAB*'s `quadprog`. We will use the SVR prediction of our one-variable test function in Figure 2.12 as an example. The data was found using the following code:

```
% Calculate vector of test data from 1D test
% function with normally distributed errors
X=(0:0.05:1)';
n=length(X);
y=(6.*X-2).^2.*sin(12.*X-4)+randn(n,1);
```

A Gaussian basis function is used and we first build a matrix of correlations between the 21 sample points (see Section 2.4 for more details on correlation matrices):

```
% Build correlation matrix (arbitrarily
% set sigma=0.2 without tuning)
sigma=0.2;
Psi=zeros(n,n);
for i=1:n
    for j=1:n
        Psi(i,j)=exp(-(1/sigma^2)*(X(i)-X(j))^2);
    end
end
```

In order to find $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$, rather than a combined $(\boldsymbol{\alpha}^+ - \boldsymbol{\alpha}^-)$, we must rewrite problem (2.62) as

$$\begin{cases} \text{minimize} \begin{cases} \frac{1}{2}\begin{pmatrix} \boldsymbol{\alpha}^+ \\ -\boldsymbol{\alpha}^- \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \boldsymbol{\Psi} & -\boldsymbol{\Psi} \\ -\boldsymbol{\Psi} & \boldsymbol{\Psi} \end{pmatrix} \begin{pmatrix} \boldsymbol{\alpha}^+ \\ -\boldsymbol{\alpha}^- \end{pmatrix} \\ + \begin{pmatrix} \mathbf{1}^{\mathrm{T}}\varepsilon - \mathbf{y} \\ \mathbf{1}^{\mathrm{T}}\varepsilon + \mathbf{y} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \boldsymbol{\alpha}^+ \\ -\boldsymbol{\alpha}^- \end{pmatrix} \end{cases} \\ \text{subject to} \begin{cases} \mathbf{1}^{\mathrm{T}} \begin{pmatrix} \boldsymbol{\alpha}^+ \\ -\boldsymbol{\alpha}^- \end{pmatrix} = 0 \\ \boldsymbol{\alpha}^+, \boldsymbol{\alpha}^- \in [\mathbf{0}, \mathbf{C}/n]. \end{cases} \end{cases} \tag{2.63}$$

Note that we have also transformed the maximization problem into a minimization, which is solved in *MATLAB* as follows:

```
% User defined constants
e=1; C=1e3; xi=1 e-6;

% Matrix of correlations
Psi=[Psi -Psi;-Psi Psi];

% Constraint terms
c=[(e*ones(n,1)-y); (e*ones(n,1)+y)];

% Lower bound |alpha|>=0
lb=zeros(2*n,1);
```

(*continued*)

```
% Upper bound |alpha| <= 0
ub=C/n*ones(2*n,1);

% Start at alpha=[0;0;...;0]
x0=zeros(2*n,1);

% Set sum(alpha^+ - alpha^-)=0
Aeq=[-ones(1,n) ones(1,n)]; beq=0;

% Run quadprog
alpha=quadprog(Psi,c,[],[],Aeq,beq,lb,ub,x0);

% Combine alphas into nx1 vector of SVs
alpha_pm=alpha(1:n)-alpha(n+1:2*n);
```

### 2.5.4 Finding $\mu$

In order to find the constant term $\mu$, known as the bias, we exploit the fact that at the point of the solution of the optimization problem (2.62) the product between the dual variables and the constraints vanishes[6] and see that

$$\alpha^{+(i)}\left(\varepsilon+\xi^{+(i)}-y^{(i)}+\mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)})+\mu\right)=0, \tag{2.64}$$

$$\alpha^{-(i)}\left(\varepsilon+\xi^{-(i)}+y^{(i)}-\mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)})-\mu\right)=0 \tag{2.65}$$

and

$$\xi^{+(i)}\left(\frac{C}{n}-\alpha^{+(i)}\right)=0, \tag{2.66}$$

$$\xi^{-(i)}\left(\frac{C}{n}-\alpha^{-(i)}\right)=0. \tag{2.67}$$

From Equations (2.66) and (2.67) we see that either $(C/n-\alpha^{\pm(i)})=0$ or $\xi^{\pm(i)}=0$ and so all points outside the $\varepsilon$-tube (where the slack variable $\xi^{\pm(i)}>0$) must have a corresponding $\alpha^{\pm(i)}=C/n$. Along with Equations (2.64) and (2.65), and noting that $\mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)})=\sum_{j=1}^{n}(\alpha^{+(j)}-\alpha^{-(j)})\boldsymbol{\psi}(x^{(i)}),(x^{(j)})$, this tells us that either

$$\alpha^{+(i)}=0$$

and

$$\mu=y^{(i)}-\sum_{j=1}^{n}(\alpha^{+(j)}-\alpha^{-(j)})\boldsymbol{\psi}(\mathbf{x}^{(i)},\mathbf{x}^{(j)})+\varepsilon \quad \text{if } 0<\alpha^{-(i)}<\frac{C}{n}, \tag{2.68}$$

[6] This is one of the Karush–Kuhn–Tucker conditions (see, for example, Schölkopf and Smola, (2002)).

or

$$\alpha^{-(i)} = 0$$

and

$$\mu = y^{(i)} - \sum_{j=1}^{n} (\alpha^{+(j)} - \alpha^{-(j)}) \boldsymbol{\psi}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \varepsilon \quad \text{if } 0 < \alpha^{+(i)} < \frac{C}{n}. \tag{2.69}$$

Using Equations (2.68) and (2.69), we can compute $\mu$ from one or more $\alpha^{\pm(i)}$'s which are greater than zero and less than $C/n$. More accurate results will be obtained if an $\alpha^{\pm(i)}$ not too close to these bounds is used. The following code calculates $\mu$ from the support vector closest to $C/2n$.

```
% Find indices of SVs
sv_i=find(abs(alpha_pm)>xi);

% Find SV mid way between 0 and C for mu calculation
[sv_mid,sv_mid_i]=min(abs(abs(alpha_pm)-(C/(2*n))))

% Calculate mu
mu=y(sv_mid_i)-e*sign(alpha_pm(sv_mid_i))...
-alpha_pm(sv_i)'*Psi(sv_i,sv_mid_i)
```

With the support vectors and $\mu$ found, we can now use the *MATLAB* code below to calculate the SVR prediction at 101 points to produce the plot in Figure 2.12.

```
% Points at which to plot prediction
x=[0:0.01:1];

for i=1:101
    % Basis function values at point to be predicted
    for j=1:n
        psi(j,1)=exp(-(1/sigma^2)*(x(i)-X(j))^2);
    end

    % SVR prediction
    pred(i)=mu+alpha_pm'*psi;
end
```

### 2.5.5 Choosing $C$ and $\varepsilon$

Our initial slack variable formulation (2.52) was a trade-off between model complexity and the degree to which errors larger than $\varepsilon$ are tolerated and is governed by the constant $C$.

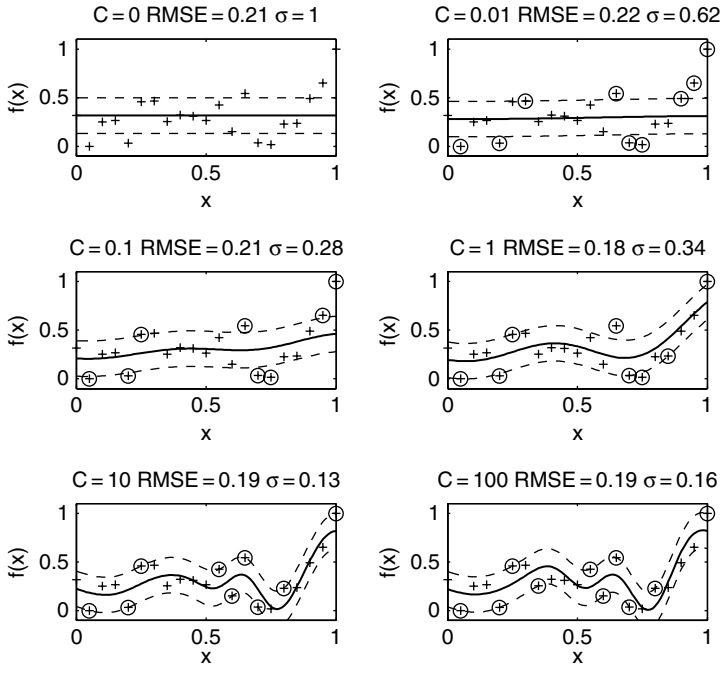



**Figure 2.14.** SVR predictions and corresponding RMSEs for varying $C$ ($\varepsilon = 4/\text{range} = 0.18(\mathbf{y})$).

A small constant will lead to a flatter prediction (more emphasis on minimizing $\frac{1}{2}|\mathbf{w}|^2$), usually with fewer SVs, while a larger constant will lead to a closer fitting of the data (more emphasis on minimizing $\sum_{i=1}^{n}(\xi^{+(i)} + \xi^{-(i)})$), usually with a greater number of SVs. We wish to choose $C$ which produces the model with the best generalization. The scaling of $\mathbf{y}$ will have an effect on the optimal value of $C$, so it is good practice to start by normalizing $y$ to have elements between zero and one. Figure 2.14 shows SVRs of the noisy one-dimensional function (this time with noise of standard deviation four), normalized between zero and one, for varying $C$. The Gaussian kernel variance, $\sigma^2$, has been tuned to minimize the RMSE of each prediction using 101 test points. This RMSE is displayed above each plot. It is clear that, although there is an optimum choice for $C$, this *exact* choice is not overly critical. It is sufficient to try a few $C$'s of varying orders of magnitude and choose that which gives the lowest RMSE for a test data set. For small problems it is possible to obtain a more accurate $C$ by using a simple bounded search such as *MATLAB's* `fminbnd`.

Here we have prior knowledge of the amount of 'noise' in the data and so have been able to choose $\varepsilon$ as the standard deviation of this 'noise'. There are many situations where we may be able to estimate the degree of 'noise', e.g. from a mesh dependency and solution convergence study. Situations, however, arise where the noise is an unknown quantity, e.g. a large amount of experimental data with measurements obtained by different researchers. In these situations we can calculate a value of $\varepsilon$ which will give the most accurate prediction by using $\nu$-SVR.

### 2.5.6 Computing $\varepsilon$: $\nu$-SVR

In $\nu$-SVR the parameter $\varepsilon$ is traded off against the model complexity and slack variables using the constant $\nu \in [0, 1]$. The corresponding constrained convex quadratic optimization problem is to

$$\begin{aligned} &\text{minimize } \frac{1}{2}|\mathbf{w}|^2 + C\left(\nu\varepsilon + \frac{1}{n}\sum_{i=1}^{n}(\xi^{+(i)} + \xi^{-(i)})\right) \\ &\text{subject to } \begin{cases} y^{(i)} - \mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)}) - \mu \leq \varepsilon + \xi^{+(i)} \\ \mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)}) + \mu - y^{(i)} \leq \varepsilon + \xi^{-(i)} \\ \xi^{\pm(i)}, \varepsilon \geq 0. \end{cases} \end{aligned} \tag{2.70}$$

In the same way as standard SVR, we introduce Lagrange multipliers to obtain the Lagrangian:

$$\begin{aligned} L = &\frac{1}{2}|\mathbf{w}|^2 + C\nu\varepsilon + C\frac{1}{n}\sum_{i=1}^{n}(\xi^{+(i)} + \xi^{-(i)}) - \beta\varepsilon - \sum_{i=1}^{n}(\eta^{+(i)}\xi^{+(i)} + \eta^{-(i)}\xi^{-(i)}) \\ &- \sum_{i=1}^{n}\alpha^{+(i)}\left(\varepsilon + \xi^{+(i)} - y^{(i)} + \mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)}) + \mu\right) \\ &- \sum_{i=1}^{n}\alpha^{-(i)}\left(\varepsilon + \xi^{-(i)} + y^{(i)} - \mathbf{w}\boldsymbol{\psi}(\mathbf{x}^{(i)}) - \mu\right). \end{aligned} \tag{2.71}$$

Taking derivatives with respect to $\mathbf{w}$ and substituting into Equation (2.50) yields the $\nu$-SVR prediction, which is the same as Equation (2.60). With the additional primal variable, $\varepsilon$, there is an additional derivative that vanishes at the saddle point:

$$\frac{\partial L}{\partial \varepsilon} = C\nu - \sum_{i=1}^{n}(\alpha^{+(i)} + \alpha^{-(i)}) - \beta = 0. \tag{2.72}$$

When substituted, along with Equations (2.54), (2.55), (2.56) and (2.57), into Equation (2.71) this allows us to eliminate $\varepsilon$ from the dual optimization problem, which is now to

$$\begin{aligned} &\text{maximize } \begin{cases} -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha^{+(i)} - \alpha^{-(i)})(\alpha^{+(i)} - \alpha^{-(i)})\boldsymbol{\Psi}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \\ +\sum_{i=1}^{n} y^{(i)}(\alpha^{+(i)} - \alpha^{-(i)}) \end{cases} \\ &\text{subject to } \begin{cases} \sum_{i=1}^{n}(\alpha^{+(i)} - \alpha^{-(i)}) = 0 \\ \alpha^{\pm(i)} \in [0, C/n] \\ \sum_{i=1}^{n}(\alpha^{+(i)} + \alpha^{-(i)}) \leq C\nu. \end{cases} \end{aligned} \tag{2.73}$$

It is possible to extract the value of $\varepsilon$ by equating Equations (2.68) and (2.69) to give

$$\begin{aligned} \varepsilon = \frac{1}{2}\left[y^{(j)} - y^{(l)} - \sum_{i=1}^{n}(\alpha^{+(i)} - \alpha^{-(i)})\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \sum_{i=1}^{n}(\alpha^{+(i)} - \alpha^{-(i)})\psi(\mathbf{x}^{(i)}, \mathbf{x}^{(l)})\right] \\ \text{if } 0 < \alpha^{+(j)} < C/n,\ 0 < \alpha^{-(l)} < C/n \end{aligned} \tag{2.74}$$

Thus to find $\varepsilon$ we need to find two support vectors with $\alpha^+ \approx C/2n$ and $\alpha^- \approx C/2n$. This is achieved via the following *MATLAB* code:

```
% Find SVs mid way between 0 and C/n for e and mu calculation
[sv_mid_p,sv_mid_p_i]=min(abs(abs(alpha(1:n))-(C/(2*n))));
[sv_mid_m,sv_mid_m_i]=min(abs(abs(alpha(n+1:2*n))-(C/(2*n))));

% Calculate e
e=0.5*(y(sv_mid_p_i)-y(sv_mid_m_i)-alpha_pm(sv_i)'*...
Psi(sv_i,sv_mid_p_i)+alpha_pm(sv_i)'* Psi(sv_i,sv_mid_m_i))

% Calculate mu
mu=y(sv_mid_p_i)-e*sign(alpha_pm(sv_mid_p_i))...
-alpha_pm(sv_i)'* Psi(sv_i,sv_mid_p_i)
```

Using this method the value of $\varepsilon$ is determined by the parameter $\nu$. It can be shown that $\nu \in [0, 1]$ is an upper bound on the fraction of training points which lie outside of the $\varepsilon$-tube and a lower bound on the fraction of training points which are support vectors. Figure 2.15 shows the effect of varying $\nu$ throughout its range for a fixed $C$ and with $\sigma$ tuned for minimum RMSE. As $\nu \to 0$, the prediction becomes a flat line with a very wide $\varepsilon$-tube and few SVs, while for $\nu = 1$, $\varepsilon = 0$ and all points become SVs. The optimum choice for $\nu$ puts a lower bound of 20 % on the number of SVs and yields $\varepsilon \approx \text{std}(\mathbf{y})$ (remember that the data

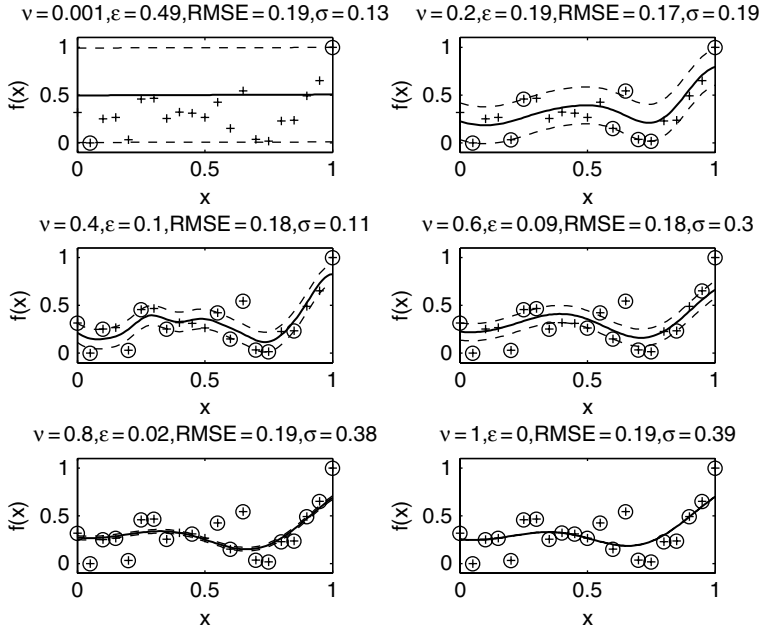



**Figure 2.15.** SVRs and RMSE for varying $\nu$ ($C = 10$).

has been normalized and $\text{std}(\mathbf{y}) = 0.18$). As with the choice of $C$, one can try a few $\nu$'s and pick the best, or use a formal search routine.

## 2.6 The Big(ger) Picture

The purpose of this chapter was to take the reader on a tour of what these authors view as the most important surrogate modelling approaches in current use today. The itinerary can be argued to have been somewhat subjective; the same is true of the time we have spent on each formulation. Should the reader wish to re-visit some of the 'attractions' for more in-depth study, here are a few pointers.

Firstly, we took a general look at surrogate model building and the basics of parameter estimation. We limited this discussion to what is necessary for an informed application of the model building methods discussed in this book. However, these are topics that, given sufficient time to delve into them, deserve entire books by themselves. Indeed, such books *have* been written, for example that by MacKay (2003).

Polynomials were our next stop. We only touched on them relatively briefly and limited the discussion to the one-variable case. The first reason is that polynomial models (or *response surface models*, as, for historical reasons, much of the literature refers to them) are unsuitable for the highly nonlinear, multimodal, multidimensional landscapes that the majority of engineering design problems may throw at the engineer. Secondly, once constructed, they offer relatively little indication as to where we should continue sampling the design space, should we wish to optimize $f$. By comparison, Kriging models, for instance, come equipped with a range of optimization handholds, as we shall see in the next chapter.

Nonetheless, polynomial models do have their uses and their advantages; for example they can be ideal for uncertainty analysis, where the analytical modelling of the propagation of probability distributions is often intractable when a complex model is fitted to the data. They have many other uses too and, consequently, a whole branch of statistical science behind them. The relevant literature deals with specific sampling planning and variable screening techniques, the challenge of elucidating landscape ridge systems, etc. There are a number of fine texts out there – the thorough treatise by Box and Draper (1987) is one of the most popular and deservedly so.

We then introduced radial basis functions. Under certain assumptions they can be shown to be universal approximators, their flexibility is easy to control (that is, we can decide on how many parameters the model will have), they lie at the foundations of other methods (e.g. Kriging) and they are easy to implement – just some of the reasons why they deserve a place in any discussion of surrogate modelling. The reader seeking more detail than can be found here may wish to consult some more general books, such as the statistical learning text of Hastie *et al.* (2001). On a more specific note, more details on radial basis function centre recruitment (noisy responses) can be found, for example, in Orr (1995).

We continued our coverage of radial basis functions with the particular case of Kriging, which we studied in some depth. Kriging has its origins in geostatistics and, although rather abstracted from our engineering concerns, Cressie (1993) contains a huge amount of material for the interested reader wishing to know more. Not as practically oriented as our discussion and directed more at statisticians, Santner *et al.* (2003) contains some more advanced concepts. A modified form of Kriging, which the reader may be interested in, is

one where the constant mean term $\mu$ is replaced with an unknown mean that best suits the data. This method is being developed by Joseph *et al*. (2008).

The final surrogate modelling approach was that of support vector regression. While our treatise on this method may seem quite in depth, it merely scratches the surface of the field of support vectors. The reader wishing to delve further into this area is unlikely to find a better initial text than that of Schölkopf and Smola (2002), which appears, watered down, in the paper by Smola and Schölkopf (2004). Indeed, our section on SVR is inspired by this paper.

## References

Box, E. P. and Draper, N. R. (1987) *Empirical Model Building and Response Surfaces*, John Wiley & Sons, Ltd, Chichester.

Cherkassky, V. and Mulier, F. (1998) *Learning from Data – Concepts, Theory, and Methods*, John Wiley & Sons, Ltd, Chichester.

Cressie, N. A. C. (1993) *Statistics for Spatial Data. Probability and Mathematical Statistics*, revised edition, John Wiley & Sons, Ltd, Chichester.

Hastie, T., Tibshirani, R. and Friedman, J. (2001) *The Elements of Statistical Learning*, Springer-Verlag, New York.

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

Joseph, V. R., Hung, Y. and Sudjianto, A. (2008) Blind Kriging: a new method for developing metamodels. *Trans. ASME, Journal of Mechanical Design* **130**(3), 31–102.

Keane, A. J. and Nair, P. B. (2005) *Computational Approaches to Aerospace Design: the Pursuit of Excellence*, John Wiley & Sons, Chichester.

Krige, D. G. (1951) A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Engineering Society of South Africa*, **52**(6), 119–139, December.

MacKay, D. J. C. (2003) *Information Theory, Inference and Learning Algorithms*, Cambridge University Press.

Matheron, G. (1963) Principles of geostatistics. *Economic Geology*, **58**, 1246–1266.

Michelli, A. (1986) Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, **2**, 11–22.

Orr, M. (1995) Regularisation in the selection of RBF centres. *Neural Computation*, **7**(3), 606–623.

Poggio, T. and Girosi, F. (1990) Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, **247**, 978–982.

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1986) *Numerical Recipes – The Art of Scientific Computing*, Cambridge University Press.

Ralston Anthony(1965) *A first course in numerical analysis*, McGraw-Hill, New York, 578.

Rasmussen, C. E. and Williams, C. K. I. (2006) *Gaussian Processes for Machine Learning*, The MIT Press, Cambridge, Massachusetts.

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

Santner, T. J., Williams, B. J. and Notz, W. I. (2003) *Design and Analysis of Computer Experiments*, Springer Series in Statistics. Springer, Berlin.

Schölkopf, B. and Smola, A. J. (2002) *Learning with Kernals*. MIT Press, Cambridge, Massachusetts.

Smola, A. J. and Schölkopf, B. (2004) A tutorial on support vector regression. *Statistics and Computing*, **14**, 199–222.

Theil, H. (1971) *Principles of Econometrics*, John Wiley & Sons, Inc., New York.

Vapnik, V. (1998) *Statistical Learning Theory*, John Wiley & Sons, Inc., New York.

# 3

# Exploring and Exploiting a Surrogate

Most of this chapter is about optimizing an expensive function $f$; that is, considering a continuous, smooth and well-posed mapping $f : \mathbf{x} \in D^k \rightarrow y \in \mathbb{R}$, and, without loss of generality, considering the case of minimization, we seek the value of $\mathbf{x}^\dagger \in D^k$ for which $f(\mathbf{x}^\dagger) < f(\mathbf{x})$ for any $\mathbf{x} \in D^k \neq \mathbf{x}^\dagger$.

Of course, more realistically, what we can usually do in practice is to find $\mathbf{x}^{\dagger\dagger}$ such that $|f(\mathbf{x}^{\dagger\dagger}) - f(\mathbf{x}^\dagger)|$ is as small as possible, within a budget of a certain number of evaluations of $f$, but, for simplicity, we shall refer to our best guess at the optimum simply as $\mathbf{x}^\dagger$. As this is a book about surrogate modelling, we shall discuss doing this with the aid of a cheap surrogate of $f$, denoted $\widehat{f}$.

The simplest surrogate based optimization recipe goes like this. If we can afford $n$ evaluations of $f$, we build a sampling plan $\mathbf{X} = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \mathbf{x}^{(n-1)}\right\}^{\mathrm{T}}$. We then calculate the responses $\mathbf{y} = \left\{y^{(1)}, y^{(2)}, \ldots, y^{(n-1)}\right\}^{\mathrm{T}}$ at these points and fit a surrogate model $\widehat{f}$ to this data. If we now assume that, for all intents and purposes, $\widehat{f}$ stands in for $f$, we can locate the $\mathbf{x}^\dagger$ that is as close to the true minimum of the function as we can make it. This should be a cheap operation and we can do it fairly thoroughly, because $\widehat{f}$ is cheap to evaluate – that is why we have built it in the first place. We now know the $\mathbf{x}^\dagger$ that minimizes $\widehat{f}$ and, of course, we know the cheaply computable estimated function value there as well: $\widehat{f}(\mathbf{x}^\dagger)$. However, the optimization task is not complete until we validate this function value against the true, expensive function $f$. This is what we have saved the $n$th evaluation of our budget for: the computation of $f(\mathbf{x}^\dagger)$.

Now, for this technique to work well, $\widehat{f}$ has to emulate $f$ fairly precisely, at least in terms of the locations of its optima. We shall discuss a number of alternative methods applicable when we have less faith in $\widehat{f}$, but before that we take a little detour for discussing the process of finding the global optimum $\mathbf{x}^\dagger$ of the cheap surrogate $\widehat{f}$.

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

## 3.1 Searching the Surrogate

Although the computational cost of evaluating $\widehat{f}(\mathbf{x})$ for a given $\mathbf{x}$ is usually minimal, the problem of finding the global optimum of $\widehat{f}$ is not always trivial. The chief ogre is multimodality, especially when the multiple local optima (troughs or valleys in the landscape) are of similar depth and can therefore be quite deceptive from an optimization perspective.

While a precise taxonomy of optimization methods is outside the scope of this book, it is worth noting that they broadly fall into two categories: *local* optimizers (also known as *hill-climbers*) and *global* searches.

Local optimizers (*exploiters*), while very efficient on many smooth, unimodal objective function landscapes, often provide less than satisfactory results when $\widehat{f}$ exhibits long valleys and/or multiple local optima. Once trapped in a valley or at a local optimum the search needs to be re-launched from a new (commonly random) starting point. This operation usually involves wasteful, lengthy exploration of unpromising regions of the search space, such as those with very poor objective values or virtually flat regions (visited before the neighbourhood of a local optimum is reached), and one can only hope that the new starting point is in the basin of attraction of a thus far unexploited local (or perhaps the global) optimum.

There is another two-way split within this class of search algorithms. Firstly, there are *gradient based optimizers*, which use landscape slope information explicitly in order to compute the best path towards the (local) optimum. These include the *Newton method* (with line-search or trust-region-type implementations), *quasi-Newton methods* (BFGS, DFP) and *conjugate gradient optimizers* (Fletcher–Reeves, Polak–Ribière).[1] Morè and Wright (1993) offer a good survey of these algorithms and their implementations and Keane and Nair (2005) also cover them in some detail.

Local optimizers that do not make explicit use of slope information, sometimes known as *direct search* methods, include the Simplex method (Nelder and Mead, 1965) (implemented in *MATLAB*'s `fminsearch.m`), the complex method of Box (1965) and the pattern search of Hooke and Jeeves (1961). In some sense Box's evolutionary operation method (Box, 1957), discussed in Chapter 1, also belongs here, though with increased step sizes this can be transformed into a global *explorer*.

Returning to the top level of our mini-taxonomy, the second major group of search methods, global explorers, such as *genetic algorithms* (see Goldberg, 1989, the classic introductory text) or *simulated annealing* (Kirkpatrick *et al.*, 1983), are good at leaving poor objective value regions behind quickly, while simultaneously exploring several basins of attraction. The exploration ability of population based global searches can be enhanced by using a space-filling sampling plan, such as those described in Chapter 1, as the initial population. *MATLAB* provides a genetic algorithm toolbox which we use for the examples presented in this book. We have included our own `ga.m` on the book website.

In comparison with local search engines, what these explorers sometimes lack is a high convergence speed (though this is less of a problem in terms of searching a surrogate, as we can usually afford a large number of evaluations) and precision in the exploitation of individual local optima.

---

[1] Some of these methods can estimate the landscape gradients themselves, say, by finite differencing, but they are, of course, more efficient if $\partial\widehat{f}/\partial x_1$, $\partial\widehat{f}/\partial x_2$, etc., are available analytically.

As the reader can see from this very cursory discussion, the science of optimization does not provide 'silver bullet' solutions – there are advantages and disadvantages to every method. In the present text, wherever an optimization process is needed we do not advocate any method over another – we merely use one that, in our experience, appears to work effectively.

## 3.2 Infill Criteria

Because our surrogate model, $\widehat{f}$, is only an approximation of the true function $f$ we wish to optimize, it is prudent to enhance the accuracy of the model using further function calls (*infill* or *update points*), in addition to the initial sampling plan. We may wish to improve the accuracy solely in the region of the optimum predicted by the surrogate to obtain an accurate optimal value quickly: local exploitation. We may, however, be unsure of the global accuracy of the surrogate and employ an infill strategy which enhances the general accuracy of the model: global exploration. We will consider each of these avenues in turn, before looking at methods which combine both schools of thought. The following sections are inspired by the excellent *Taxonomy of global optimization methods based on response surfaces* by Jones (2001).

### *3.2.1 Prediction Based Exploitation*

Applying infill points at the optimum predicted by the surrogate allows us to quickly converge upon an optimum value. However, this may not be the global optimum. Imagine we are searching a function with the form $f(x) = (6x-2)^2 \sin(12x-4)$ in the range $x \in [0, 0.5]$ using an interpolating model. The 'true' function, $f$, and a Gaussian RBF approximation, $\widehat{f}$, through three sample points are shown in Figure 3.1. Based on such a small sample the surrogate is not particulary accurate.

If we minimize the RBF model and add a function evaluation at this location, upon refitting the RBF, the approximation in Figure 3.2 is obtained. The model is now more accurate in the region of the previous optimum. Repeating the process of adding infill points at the minimum of the approximation results in the optimization converging on the optimum, as shown in Figure 3.3.

We now optimize the same function again, but this time using a second-order polynomial regression. The initial approximation is shown in Figure 3.4 and the situation after the addition of two infill points is shown in Figure 3.5. Note how the addition of new data does not necessarily improve the predictive capability of the surrogate and we may in fact never find the minimum of the function. Adding further updates to the two shown in Figure 3.5 will not change the polynomial sufficiently to divert the search away from the flat spot on the function – the optimization has stalled. This example highlights a key benefit of interpolating models: they continually improve with the addition of infill points.[2] The situation could be improved by using a higher order polynomial. A third-order polynomial has been used in Figure 3.6, and the search is, in fact, slowly converging towards the optimum.

[2] This trait can, in some cases where the function is extremely multimodal, be a disadvantage, and regression may be required. We will look at such a scenario later in Chapter 6.

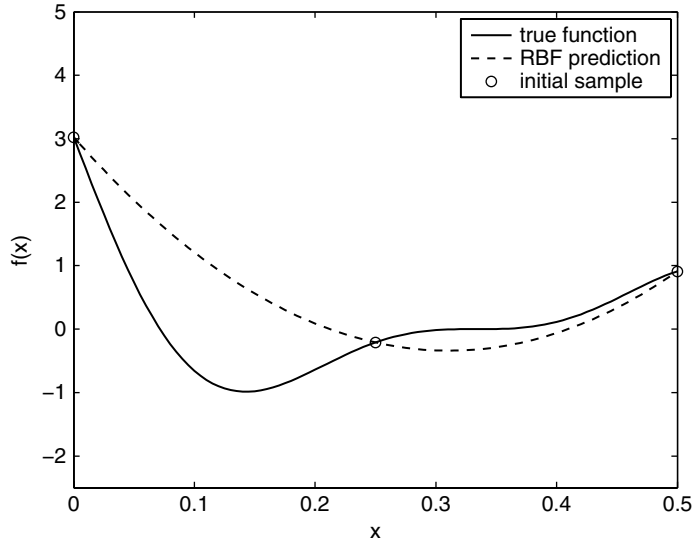



**Figure 3.1.** The function $f(x) = (6x-2)^2 \sin(12x-4)$ with a Gaussian RBF through three sample points.

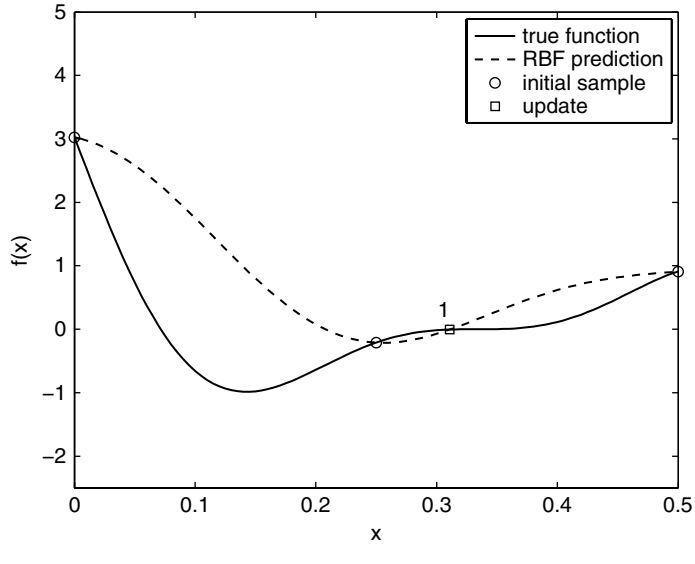



**Figure 3.2.** The situation after the RBF prediction in Figure 3.1 is enhanced with one update at the minimum of the prediction.

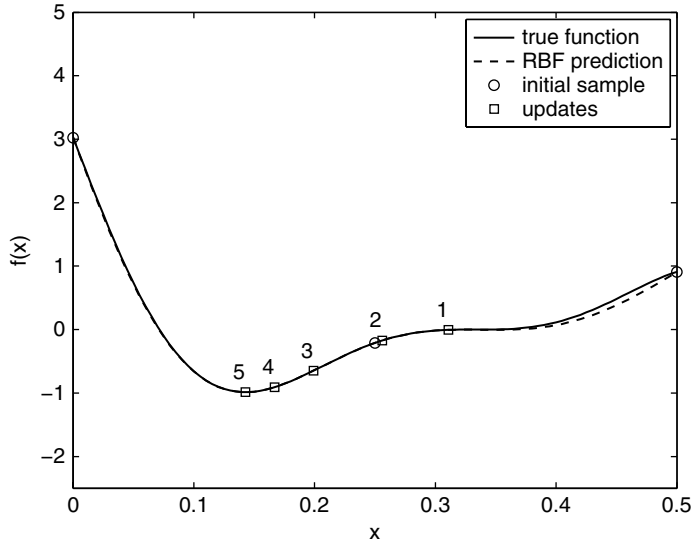



**Figure 3.3.** The infill strategy converges on the optimum after five updates at the minimum of the prediction.

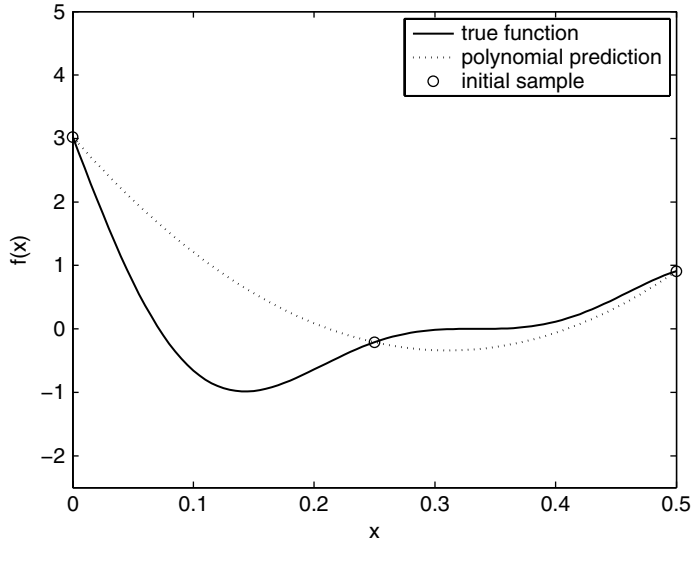



**Figure 3.4.** The function $f(x) = (6x-2)^2 \sin(12x-4)$ with a second-order polynomial through three sample points.

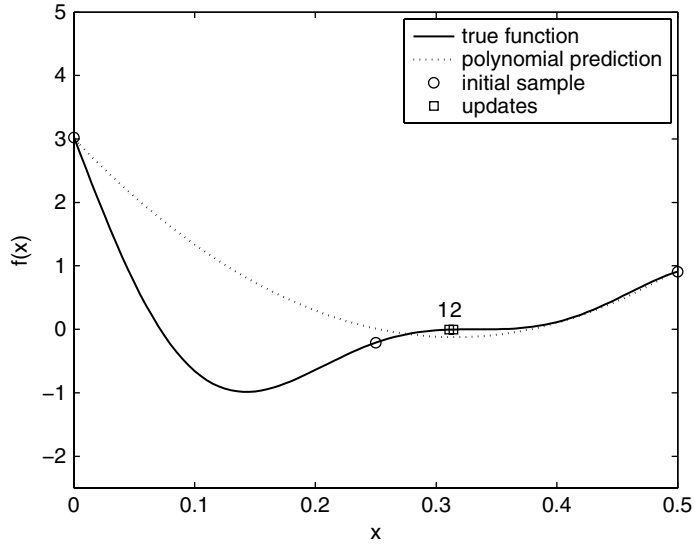



**Figure 3.5.** The situation after the polynomial prediction in Figure 3.1 is 'enhanced' with two updates at the minimum of the prediction. The infill strategy cannot escape the flat spot of the function.

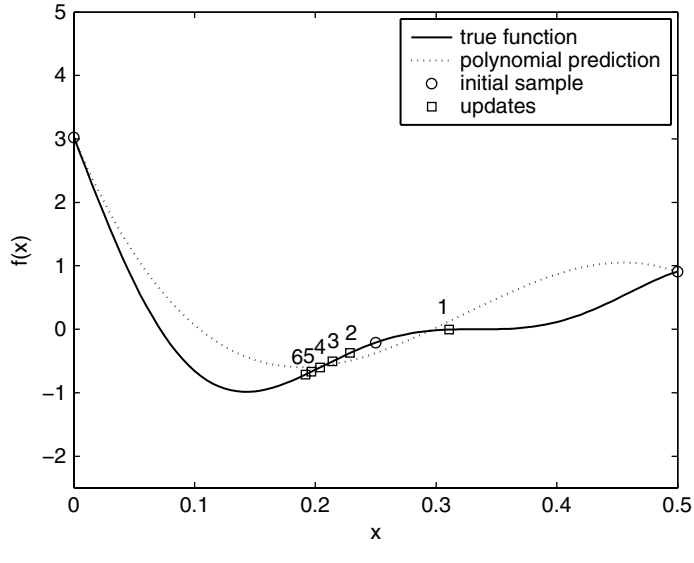



**Figure 3.6.** A minimum prediction based infill strategy using a third degree polynomial improves upon the second-degree polynomial in Figure 3.5, though the convergence towards the optimum is slow.

Now consider the search of the same function over the range [0,1], which is shown in Figure 3.7 along with a Gaussian RBF and a polynomial based on three sample points. Updating the RBF model at its minimum does not lead to the discovery of the optimum of this deceptive function: the search gets stuck at a local minimum, as shown in Figure 3.8.

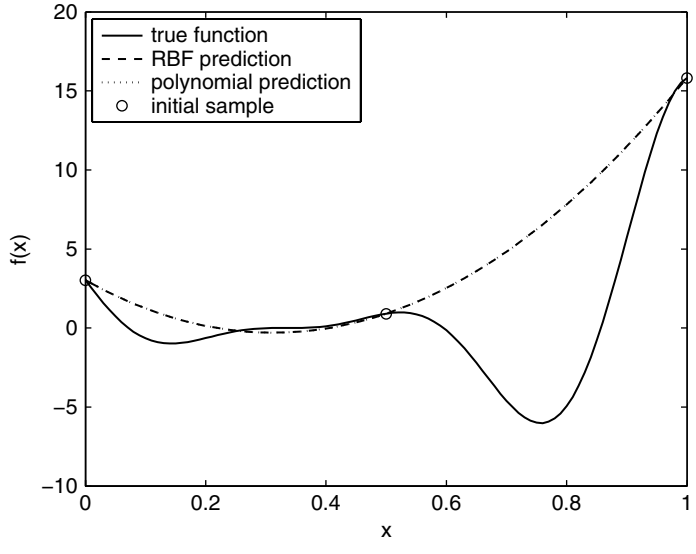



**Figure 3.7.** The function $f(x) = (6x-2)^2 \sin(12x-4), \quad x \in [0,1]$, with a Gaussian RBF and a second-order polynomial through three sample points.

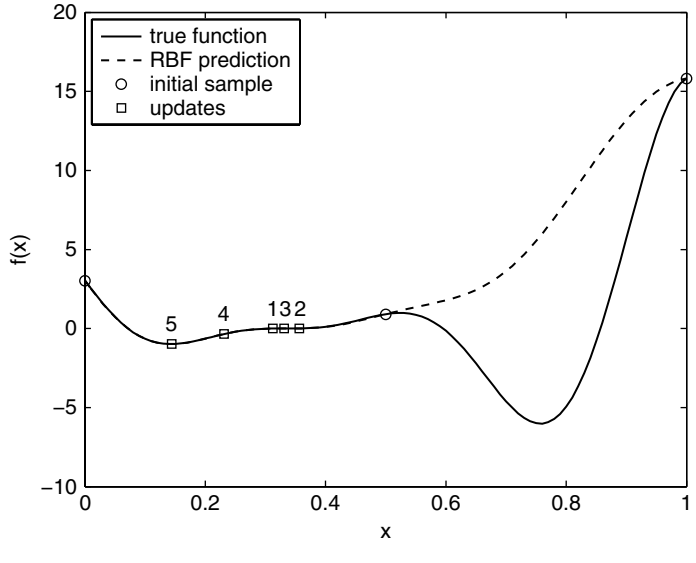



**Figure 3.8.** A minimum prediction based infill strategy starting from the Gaussian RBF prediction in Figure 3.7 fails to find the global optimum of the function.

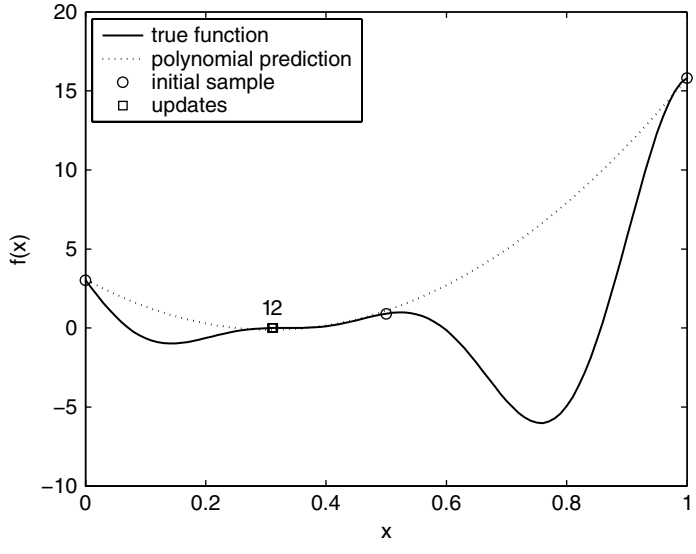



**Figure 3.9.** A minimum prediction based infill strategy starting from the second-order polynomial prediction in Figure 3.7 fails to find even a local optimum of the function.

Using the polynomial model is even worse. The search does not even find a local minimum, as shown in Figure 3.9. Clearly, for multimodal functions where the initial model does not approximate the whole function well, an infill strategy that can search away from the current minimum and explore other regions is required.

### *3.2.2 Error Based Exploration*

The Gaussian process based models discussed in Chapter 2 permit the calculation of an estimated error in the model and so it is possible to use this to position infill points where our uncertainty in the predictions of the model is highest. This represents a key advantage of Gaussian process based models.

The mean squared error (MSE) in a Gaussian process based prediction is

$$\widehat{s}^2(\mathbf{x}) = \sigma^2 \left[ 1 - \boldsymbol{\psi}^{\mathrm{T}} \boldsymbol{\Psi}^{-1} \boldsymbol{\psi} + \frac{1 - \mathbf{1}^{\mathrm{T}} \boldsymbol{\Psi}^{-1} \boldsymbol{\psi}}{\mathbf{1}^{\mathrm{T}} \boldsymbol{\Psi}^{-1} \mathbf{1}} \right] \tag{3.1}$$

The derivation of this equation can be found in Sacks *et al.* (1989). The third term inside the square parentheses, which is due to uncertainty in the estimate of $\mu$, is very small and is often omitted (it does not appear if the derivation is considered from a Bayesian stance).

Figure 3.10 shows the estimated root mean squared error in the prediction in Figure 3.8, found using Equation (3.1). Note how $\widehat{s}^2$ reduces to zero at the sample points. This is evident

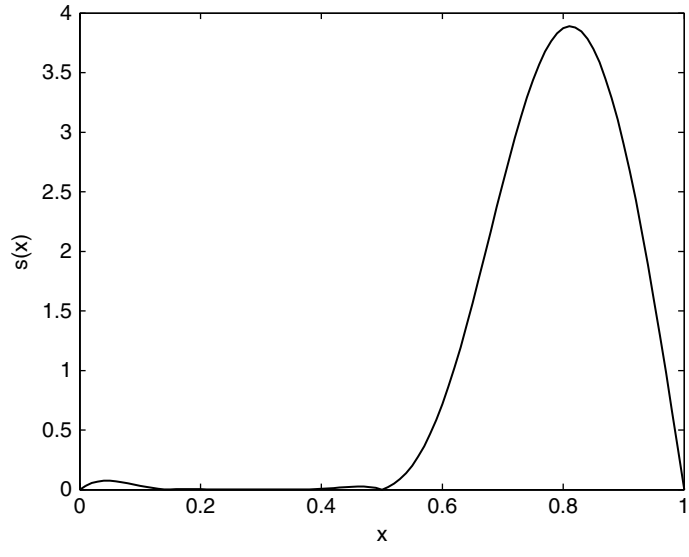



**Figure 3.10.** The value of $s(\mathbf{x})$, found from Equation (3.1), for the prediction in Figure 3.8.

by examining Equation (3.1). If we are calculating $\widehat{s}^2(\mathbf{x})$ at a sample point $\mathbf{x}^{(i)}$, $\mathbf{x}$ is an element of $\mathbf{X}$ and so $\boldsymbol{\psi}$ is a column of $\boldsymbol{\Psi}$. Thus $\boldsymbol{\Psi}^{-1}\boldsymbol{\psi}$ is the $i$th unit vector so

$$\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi} = \psi^{(i)} = 1 \tag{3.2}$$

and

$$\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi} = 1. \tag{3.3}$$

Substituting Equations (3.2) and (3.3) into Equation (3.1) yields $\widehat{s}^2(\mathbf{x}^{(i)}) = 0$. This follows our intuition that if we are interpolating a point at which we know the answer, the error in the prediction must be zero.[3]

We could choose to use maximizing the predicted error as an infill criterion. It is clear from the values of $\widehat{s}^2$ in Figure 3.10 that we would escape the local minimum, however, such an infill strategy is tantamount to just filling in the gaps and could be achieved by simply using a larger sampling plan. We would also be faced with the question of when should we stop adding points at maximum error and start exploiting the model? Instead of either exploiting or exploring the model we can use infill criteria which balance these options.

### *3.2.3 Balanced Exploitation and Exploration*

Using the estimated error of a Gaussian process based prediction found by Equation (3.1) we can model our uncertainty in the prediction by considering it as the realization of a normally distributed random variable $Y(\mathbf{x})$ with mean $\widehat{y}(\mathbf{x})$ (the most 'likely' prediction

[3] This logic does not apply if we are not sure about the observed value $y^{(i)}$ at $\mathbf{x}^{(i)}$ and Equation (3.1) must be modified. We will deal with this scenario in Chapter 6.

found as an MLE) and variance $\widehat{s}^2(\mathbf{x})$. By considering the possibility that $Y(\mathbf{x})$ could take different values, due to the size of $\widehat{s}^2(\mathbf{x})$, we can construct infill criteria which balance the values of $\widehat{y}(\mathbf{x})$ and $\widehat{s}^2(\mathbf{x})$.

### Statistical Lower Bound

The simplest way of balancing exploitation of the prediction $\widehat{y}(\mathbf{x})$ and exploration using $\widehat{s}^2(\mathbf{x})$ is to minimize a *statistical lower bound*:

$$\mathrm{LB}(\mathbf{x}) = \widehat{y}(\mathbf{x}) - A\widehat{s}(\mathbf{x}), \tag{3.4}$$

where $A$ is a constant that controls the exploitation/exploration balance. As $A \to 0$, $\mathrm{LB}(\mathbf{x}) \to \widehat{y}(\mathbf{x})$ (pure exploitation) and as $A \to \infty$, the effect of $\widehat{y}(\mathbf{x})$ becomes negligible and minimizing $\mathrm{LB}(\mathbf{x})$ is equivalent to maximizing $\widehat{s}(\mathbf{x})$ (pure exploration).

With the observed data, model parameters, correlation matrix, its Cholesky factorization and the constant $A$ stored in a global structure `ModelInfo`, the following *MATLAB* function will calculate the statistical lower bound. Storing the surrogate model information in this way significantly reduces the computational expense of the large number of calls required to search `lb.m`. We will use this method of storing and passing surrogate model information to infill functions throughout this section:

```
function LowerBound=lb(x)
% Calculates a Kriging prediction at x minus A
% estimated standard deviations
%
% Inputs:
%       x-1 x k vector of design variables
%
% Global variables used:
%       ModelInfo.X – n x k matrix of sample locations
%       ModelInfo.y – n x 1 vector of observed data
%       ModelInfo.Theta – 1 x k vector of log(theta)
%       ModelInfo.U – n x n Cholesky factorization of Psi
%       ModelInfo.A – scalar weighting parameter
% Outputs:
%       LowerBound – scalar lower bound

global ModelInfo
% extract variables from data structure
% slower, but (makes code easier to follow)
X=ModelInfo.X;
y=ModelInfo.y;
theta=10.^ ModelInfo.Theta;
U=ModelInfo.U;
A=ModelInfo.A;

% Calculate number of sample points
n=size(X,1);

% Vector of ones
one=ones(n,1);
```

(*continued*)

```
% Calculate mu
mu=(one'*(U\(U'\y)))/(one'*(U\(U'\one)));

% Calculate sigma^2
SigmaSqr=((y-one*mu)'*(U\(U'\(y-one*mu))))/n;

% Initialize psi to vector of ones
psi=ones(n,1);

% Fill psi vector
for i=1:n
      psi(i)=exp(-sum(theta.*abs(X(i,:)-x).^2));
end

% Calculate prediction
f=mu+psi'*(U\(U'\(y-one*mu)));

% Error
SSqr=SigmaSqr*(1-psi'*(U\(U'\psi)));

% Lower bound
LowerBound=f-A*(sqrt(SSqr));
```

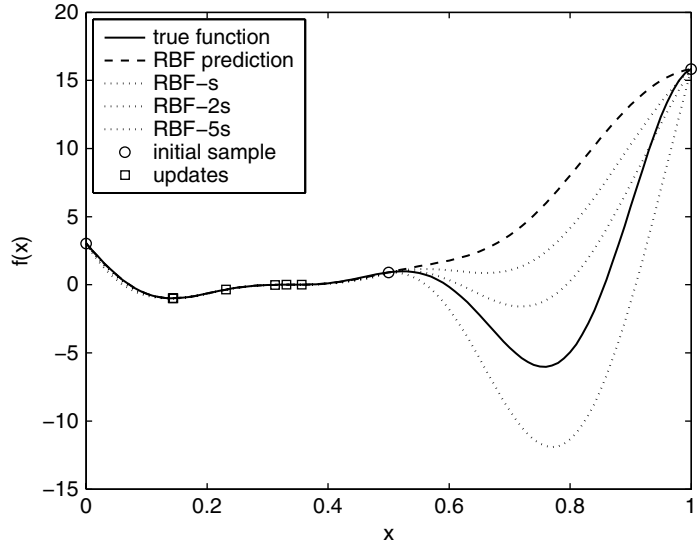



**Figure 3.11.** The statistical lower bound for the prediction in Figure 3.8 with varying $A$.

Figure 3.11 shows the lower bound for the prediction in Figure 3.8 for varying $A$. It is clear that optimization using this infill criterion *could* find the global minimum, but it is unclear how one should choose the user defined parameter $A$ to obtain a good balance between exploitation and exploration. We now move on to more elegant methods which do not rely on a user defined parameter.

### Probability of Improvement

When performing a search and infill strategy, we wish to position the next infill point at the value of $\mathbf{x}$ that will lead to an improvement on the best observed value so far, $y_{\min}$. By considering $\widehat{y}(\mathbf{x})$ as the realization of a random variable we can calculate the probability of an improvement $I = y_{\min} - Y(\mathbf{x})$ upon $y_{\min}$, the *probability of improvement*:[4]

$$P[I(\mathbf{x})] = \frac{1}{\widehat{s}\sqrt{2\pi}} \int_{-\infty}^{0} \mathrm{e}^{-[I-\widehat{y}(\mathbf{x})]^2/(2s^2)} \mathrm{d}I \tag{3.5}$$

This is calculated using the error function as

$$P[I(\mathbf{x})] = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{y_{\min} - \widehat{y}(\mathbf{x})}{\widehat{s}\sqrt{2}}\right)\right] \tag{3.6}$$

The *MATLAB* code used to calculate $P[I(\mathbf{x})]$ only differs slightly from `lb.m` (above). We no longer require $A$, and the last line of code is replaced with:

```
ProbImp=(0.5+0.5*erf((min(y)-y_hat)/sqrt(SSqr*2)));
```

Equation (3.6) is interpreted graphically in Figure 3.12. The figure shows the prediction in Figure 3.8 along with a vertical Gaussian distribution with variance $s^2(\mathbf{x})$ centred around

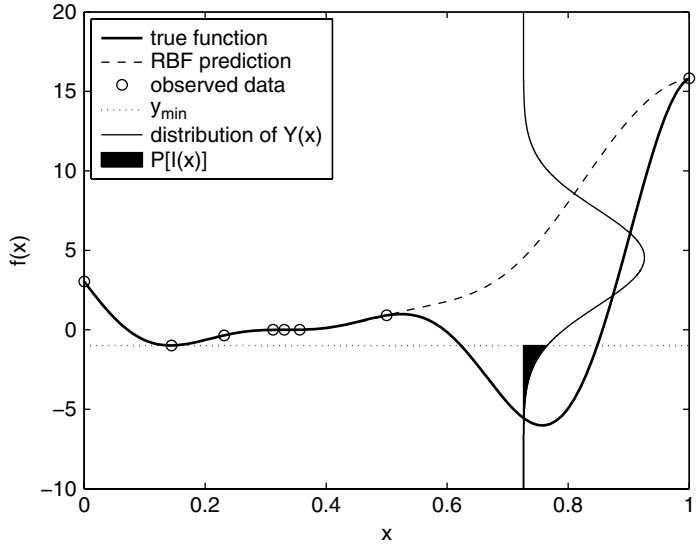



**Figure 3.12.** A graphical interpretation of the probability of improvement.

[4] $y_{\min} - Y(\mathbf{x})$ should be replaced by $Y(\mathbf{x}) - y_{\max}$, for a maximization problem, but in practice it is easier to take the negative of the data so that all problems can be treated as minimization problems.

$\widehat{y}(\mathbf{x})$. This Gaussian distribution represents the uncertainty in the prediction $\widehat{y}(\mathbf{x})$ and the part of the distribution below the dotted line indicates the possibility of improving on the best observed value (the quantity we are integrating in Equation (3.5)). The probability of improvement is the area enclosed by the Gaussian distribution below the best observed value so far (the value of the integral in Equation (3.5)).

Plotting the probability of improvement for the prediction in Figure 3.8 for all values of $\mathbf{x}$ yields the plot in Figure 3.13. The highest probability of improvement is in the local minimum but there is a probability of improvement in the region of the global optimum and further infill points would eventually find this region. We know that the global optimum will eventually be found because $P[I(\mathbf{x})] = 0$ when $\widehat{s} = 0$, so there is no probability of improvement at a point that has already been sampled and therefore no possibility of re-sampling. This guarantees global convergence to a global optimum (given certain assumptions which we will consider later) because the sampling will eventually become dense.

Note that although Figure 3.13 indicates where an improvement might be found it does not show us how big that improvement could be. We will now turn to an infill criterion which does just that.

### Expected Improvement

Instead of simply finding the probability that there will be some improvement, we can calculate the amount of improvement we expect, given the mean $\widehat{y}(\mathbf{x})$ and variance $\widehat{s}^2(\mathbf{x})$. This *expected improvement* is given by

$$E[I(\mathbf{x})] = \begin{cases} (y_{\min} - \widehat{y}(\mathbf{x}))\Phi\left(\dfrac{y_{\min} - \widehat{y}(\mathbf{x})}{\widehat{s}(\mathbf{x})}\right) + s\phi\left(\dfrac{y_{\min} - \widehat{y}(\mathbf{x})}{\widehat{s}(\mathbf{x})}\right) & \text{if } s > 0 \\ 0 & \text{if } s = 0 \end{cases} \tag{3.7}$$

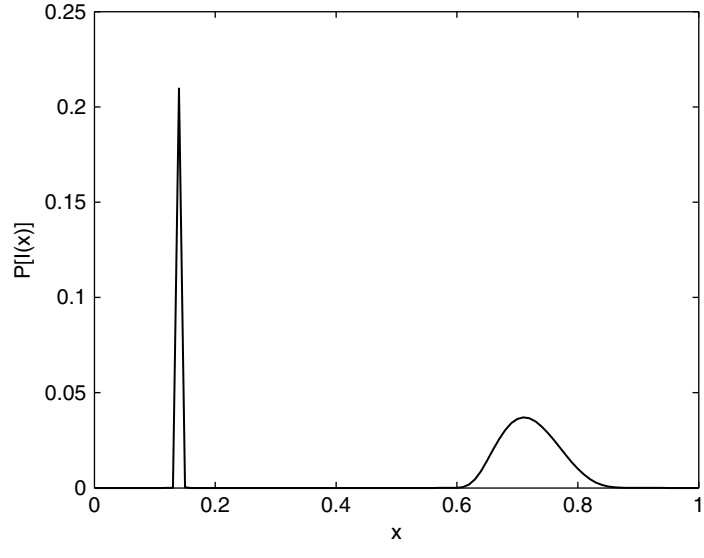



**Figure 3.13.** The probability of improvement in the prediction shown in Figure 3.8.

where $\Phi(.)$ and $\phi(.)$ are the cumulative distribution function and probability density function respectively. This equation can be interpreted graphically from Figure 3.12 as the first moment of area enclosed by the Gaussian distribution below the best observed value so far. Note that because, like $P[I(\mathbf{x})]$, $E[I(\mathbf{x})] = 0$ when $\widehat{s} = 0$, a maximum expected improvement infill procedure will also eventually find the global optimum.

Equation (3.7) is evaluated using the error function as

$$E[I(\mathbf{x})] = (y_{\min} - \widehat{y}(\mathbf{x}))\left[\frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{y_{\min} - \widehat{y}(\mathbf{x})}{\widehat{s}\sqrt{2}}\right)\right] + \widehat{s}\frac{1}{\sqrt{2\pi}}\exp\left[\frac{-(y_{\min} - \widehat{y}(\mathbf{x}))^2}{2\widehat{s}^2}\right] \tag{3.8}$$

(see also Mathematical Note on page 148) or in *MATLAB* as:

```
function ExpImp=ei(x)

...
% Same as lb.m
...

% Best point so far
y_min=min(y);
% Expected improvement
if SSqr==0
    ExpImp=0;
else
    ei_termone=(y_min-y_hat)*(0.5+0.5*erf((1/sqrt(2))*...
    ((y_min-y_hat)/sqrt(abs(SSqr)))));
    ei_termtwo=sqrt(abs(SSqr))*(1/sqrt(2*pi))*exp(-(1/2)*...
    ((y_min-y_hat)^2/SSqr));
    ExpImp=ei_termone+ei_termtwo;
end
```

Plotting the expected improvement for the prediction in Figure 3.8 for all values of $\mathbf{x}$ yields the plot in Figure 3.14. In contrast to the probability of improvement in Figure 3.13, the expected improvement is greatest in the unsampled area of the global optimum. This is because, although there is a high probability of some improvement at the point that maximizes $P[I(\mathbf{x})]$, the actual amount of improvement is likely to be greater at the point that maximizes $E[I(\mathbf{x})]$. The expected improvement ($\approx 0.043$) is in fact much smaller than the true improvement that would be obtained at that point ($\approx 9$) due to the deceptive nature of the function. We will consider this in the next section together with our assumptions about the convergence of $P[I(\mathbf{x})]$ and $E[I(\mathbf{x})]$ based infill criteria. First though, we will look at the performance of what seems to be our most promising criterion so far: maximizing $E[I(\mathbf{x})]$.

Figure 3.15 shows the progress of a search of the one-variable test function in the range [0,1] using a maximum $E[I(\mathbf{x})]$ infill strategy starting from an initial sample of three points. The left-hand column shows the progress of the Gaussian process based RBF prediction and the right-hand column shows the expected improvement at each stage. To begin with,

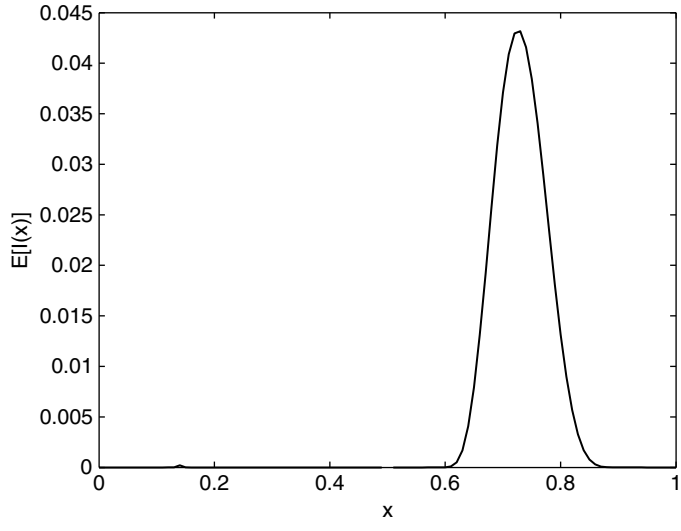



**Figure 3.14.** The expected improvement in the prediction shown in Figure 3.8.

the search follows a similar route to the RBF predictor based infill strategy (Figure 3.8). However, after isolating the local optimum to the left of the plot, there is still an expectation of improvement to the right, and so the global optimum is found. Note that the scale of the $E[I(\mathbf{x})]$ plots varies. To begin with there is high expectation. This diminishes through the first three pairs of plots as the prediction 'thinks' it has found a very smooth function. When the dip to the left is found the prediction becomes more multimodal and 'realizes' that, in fact, the errors in the prediction may be higher, giving the possibility of improvements: hence the higher $E[I(\mathbf{x})]$ appears in the fifth row. By the final pair of plots $E[I(\mathbf{x})]$ has diminished to the point where we can have more confidence that the global optimum has been found. However, were we predicting the output of a higher dimensional problem where we could not easily visualize the surrogate model, we could not be completely certain.

In many situations maximizing $E[I(\mathbf{x})]$ will prove to be the best route to finding the global optimum. Should the assumptions through which our confidence is based in this method prove to be false, maximizing $E[I(\mathbf{x})]$ (and all other estimated error based criteria we have considered) may converge very slowly or not at all. It is therefore worth considering a breed of infill criteria which can alleviate this pitfall.

### *3.2.4 Conditional Likelihood Approaches*

All of the methods we have covered so far could *possibly* be 'tricked' by a particularly poor or unlucky initial sample and a very deceptively positioned optimum. Consider the third pair of plots in Figure 3.15 where $E[I(\mathbf{x})]$ diminished to a very small value because the prediction 'thinks' that the function is very smooth. In such a situation the variance of the Gaussian correlation becomes very large (for Kriging the $\boldsymbol{\theta}$ parameter becomes very small), which results in a very small estimated error $\widehat{s}^2(\mathbf{x})$. This is because when the variance of the correlation increases all the elements of $\boldsymbol{\Psi}$ and $\boldsymbol{\psi}$ get closer to 1 and so $\boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi} \to 1$; thus

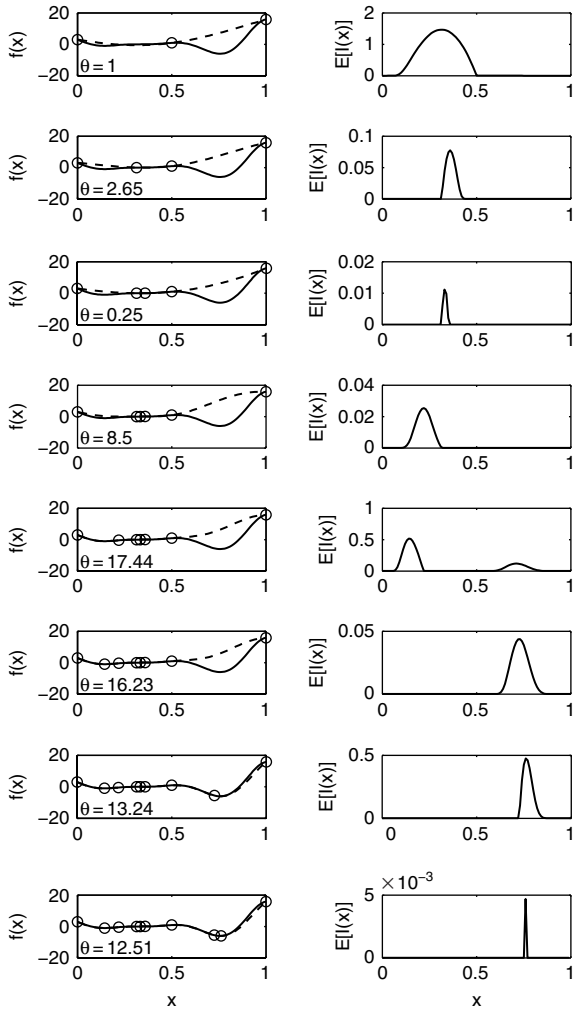



**Figure 3.15.** The progress of a search of the one-variable test function in the range [0, 1] using a maximum $E[I(\mathbf{x})]$ infill strategy.

Equation (3.1) tends to zero. The erroneously small error leads to an overemphasis on exploitation of the prediction and the search dwells for too long on local optima. The infill criteria has fallen into this trap because we have assumed that the unknown Gaussian process based model parameters have been estimated correctly. After the second update in Figure 3.15, $\theta$ is estimated at 0.25, whereas after six updates (when the prediction starts to match the true function) we find the true value for $\theta$ is approximately 13.

If we start to play Devil's advocate, we can concoct similar, but more severe, scenarios where an estimated error based infill criterion will make no progress at all. Consider the function shown in Figure 3.16. We have been unlucky enough to sample the function at three points with the same function value. Although in Chapter 1 we have considered how best to avoid this problem when sampling harmonic responses, it could still occur, particularly if

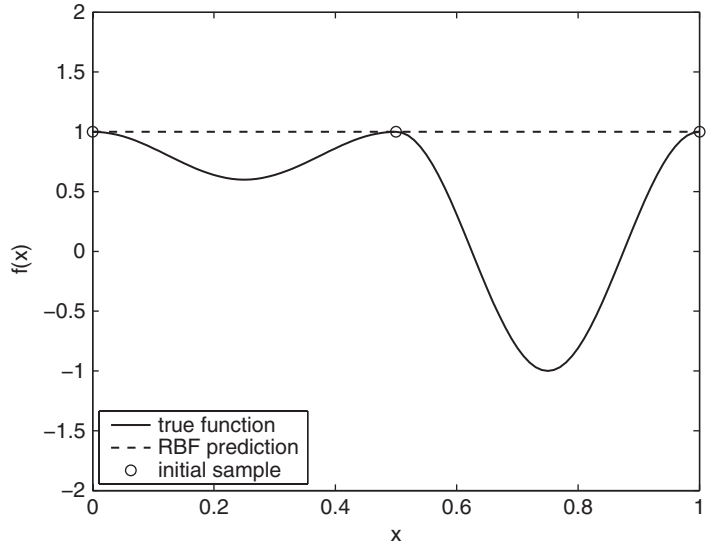



**Figure 3.16.** A deceptive function with a particularly unlucky sampling plan.

large portions of the function landscape are, in fact, flat. An error based infill criterion cannot cope with the prediction in Figure 3.16 because the estimated error is zero for all values of $x$.

In rare situations like that in Figure 3.16 we must, and in situations like that in the third pair of plots in Figure 3.15 it is advantageous to employ an infill criterion that takes into account the possibility that a deceptive sample may have resulted in significant error in our estimation of the model parameters. We will consider two appropriate infill criteria for use in such scenarios. These criteria do not use the surrogate to find the minimum, but rather use the minimum to find the surrogate. Or, in a sound bite (paraphrased from Jones and Welch, 1996), ask not what the surrogate implies about the minimum, ask what the minimum implies about the surrogate.

### Goal Seeking

In some cases we may be able to estimate the value of the objective function at the optimum, or it may be that we would like to search for a specific improvement, e.g. over a current product (even if it is not known if that improvement is possible), rather than an unknown optimum. In such situations we can use a method which does not search for expectations or probabilities of improvement, but assesses the likelihood that an objective function value *could* exist at a given point (Jones, 2001).

In the derivation of the Kriging predictor in Section 2.4 we found the predictor as the maximum of the likelihood of the sample data augmented with the point to be predicted. Now, instead of estimating the value $\widehat{y}(\mathbf{x})$ for a given $\mathbf{x}$, we assume the predictor passes through a goal $y^{\mathrm{g}}$ as well as the sample data and find the value of $\widehat{\mathbf{x}}^{\mathrm{g}}$ which best fits this assumption. To do this we maximize the *conditional ln-likelihood*

$$-\frac{n}{2}\ln(2\pi)-\frac{n}{2}\ln(\widehat{\sigma}^2)-\frac{1}{2}\ln|\mathbf{C}|-\frac{(\mathbf{y}-\mathbf{m})^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{y}-\mathbf{m})^{\mathrm{T}}}{2\widehat{\sigma}^2}, \tag{3.9}$$

where

$$\mathbf{m} = \mathbf{1}\mu + \boldsymbol{\psi}(\hat{y}^{\mathrm{g}} - \mu) \tag{3.10}$$

and

$$\mathbf{C} = \boldsymbol{\Psi} - \boldsymbol{\psi}\boldsymbol{\psi}^{\mathrm{T}}, \tag{3.11}$$

by varying $\hat{\mathbf{x}}^{\mathrm{g}}$ *and* the model parameters (at this stage we may wish to widen the upper and lower bounds on $\boldsymbol{\theta}$). The position of the goal, $\hat{\mathbf{x}}^{\mathrm{g}}$, appears in Equation (3.9) via its vector of correlations with the observed data, $\boldsymbol{\psi}$. When deriving the Kriging predictor we could differentiate the augmented ln-likelihood and set it to zero. Here we must maximize the conditional ln-likelihood numerically in the same way as for tuning the model parameters, e.g. using a genetic algorithm. We can first make the same substitution for the MLE $\hat{\sigma}^2$ as we made in Section 2.4 to give the concentrated conditional ln-likelihood:

$$-\frac{n}{2}\ln(\hat{\sigma}^2) - \frac{1}{2}\ln|\mathbf{C}|. \tag{3.12}$$

This can be calculated in *MATLAB* using the function below,

```
function NegCondLnLike=condlikelihood(x)
% Calculates the negative of the conditional
% ln – likelihood at x(k+1:2*K)
%
% Inputs:
%       x – 1 x 2k vector of log theta and hypothesized point
%
% Global variables used:
%       ModelInfo.X – n x k matrix of sample locations
%       ModelInfo.y – n x 1 vector of observed data
%       ModelInfo.Goal – scalar goal
%
% Outputs:
%       NegCondLnLike – scalar negative ln–likelihood

global ModelInfo
X=ModelInfo.X;
y=ModelInfo.y;
[n,k]=size(X);
theta=10.^x(1:k);

% Hypothesized point
% xHyp=x(k+1:2*k)

% Pre – allocate memory
Psi=zeros(n,n);
```

(*continued*)

```
% Build upper half of correlation matrix
for i=1:n
  for j=i+1:n
     Psi(i,j)=exp(-sum(theta.*(X(i,:)-X(j,:)).^2));
  end
end

% Add upper and lower halves and diagonal of ones plus
% small number to reduce ill-conditioning
Psi=Psi+Psi'+eye(n)+eye(n).*eps;

% Cholesky factorization
U=chol (Psi);

% Vector of ones
one=ones(n,1);

% Calculate mu
mu=(one'*(U\(U'\y)))/(one'*(U\(U'\one)));

% Initialize psi to vector of ones
psi=ones (n,1);

% Fill psi vector
for i=1:n
    psi(i)=exp(-sum(theta.*abs(X(i,:)-xHyp).^2));
end

% Build conditional covariance matrix
m=one*mu+psi*(ModelInfo.Goal-mu);
C=Psi-psi*psi';

% Cholesky factorization of C
U=chol(C);

% Sum lns of diagonal to find ln(abs(det(Psi)))
LnDetC=2*sum(log(abs(diag(U))));

% Use back-substitution of Cholesky instead of inverse
SigmaSqr=((y-m)'*(U\(U'\(y-m))))/n;
NegCondLnLike=-1*(-(n/2)*log(SigmaSqr)-0.5*LnDetC);
```

To see how effective this method can be we will consider the search of our one-dimensional test function. We begin with three sample points, and set an objective function goal of $-5$:

```
ModelInfo.X=[0 0.5 1]';
for i=1:3
   ModelInfo.y(i,1)=onevar(ModelInfo.X(i));
end
ModelInfo.Goal=-5;
```

After building the initial sample we tune a Kriging model (with such a sparsity of data, we have reduced the upper bound on $\theta$ to obtain a more reasonable prediction, as we have done to produce the figures for all the above infill criteria):

```
% Tune Kriging model of objective
[ModelInfo.Theta,MaxLikelihood]=fminbnd(@likelihood, −4,1);
% Put Cholesky factorization of Psi into ModelInfo
[NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood(ModelInfo.Theta);
```

before maximizing the concentrated conditional ln-likelihood:

```
[OptVar,NegCondLike]=ga(@condlikelihood, 2, [], [], [], [], ...
[-4 0], [1 1]);
```

We can now evaluate the function at `OptVar`, add this to the sampling plan and repeat the process until the goal is found. In fact, we reach the goal after just four updates, as shown in Figure 3.17.

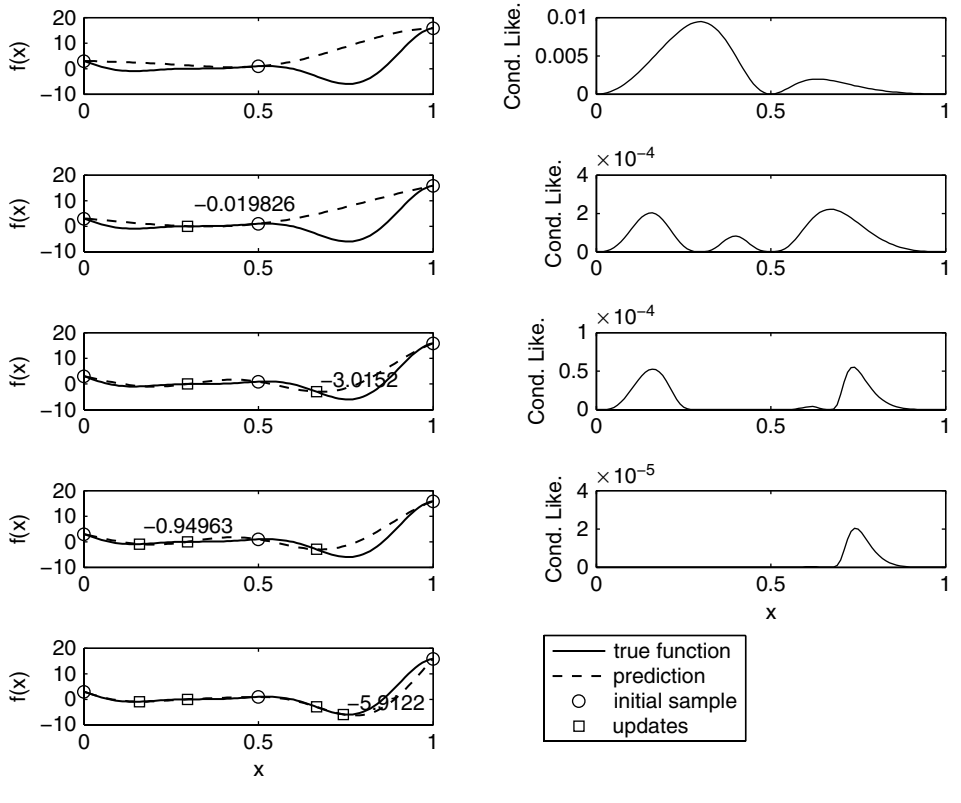



**Figure 3.17.** The progress of a search of the one-variable test function in the range [0, 1] using a goal seeking infill strategy.

### The Conditional Lower Bound

In many cases we will not be able to specify a goal for the optimization, but we can still use a conditional likelihood approach. Instead of finding the $\mathbf{x}$ that gives the highest likelihood conditional upon $\widehat{y}(\mathbf{x})$ passing through a goal, we find the $\mathbf{x}$ which minimizes $\widehat{y}(\mathbf{x})$ subject to the conditional likelihood not being too low. We will define what we mean by 'too low' in due course. This method was first proposed by Jones and Welch (1996).

Again, consider the prediction of the deceptive one-variable test function based on an initial sample of three points. This is shown in Figure 3.18, along with the statistical lower bound found by subtracting the estimated RMSE ($\widehat{s}(\mathbf{x})$). At $x = 0.7572$, which we know is the minimum of the function, a point with $y^h = \widehat{y}(\mathbf{x})$ has been imputed (i.e. we have hypothesized that this point is part of the sample data, even though it has not actually been observed). The likelihood conditional upon the prediction passing through this point is shown. Subsequently, we have imputed lower and lower values at $x = 0.7572$ and re-optimized $\theta$ to produce a prediction through these points. These values fall well below our statistical lower bound, but still have a conditional likelihood and so represent possible values at $x = 0.7572$. As the imputed value reduces, the conditional likelihood becomes extremely low and we clearly need a systematic method of dismissing imputations which are very unlikely. We achieve this using a *likelihood ratio test* (see the mathematical note at the end of this section).

By calculating the ratio of the conditional likelihood of the MLE prediction, $L_0$, to the conditional likelihood, $L_{\text{cond}}$, of the prediction passing through the imputed point and comparing it to the $\chi^2$ distribution, we can make a decision as to whether to accept the value of the imputed point. To be accepted

$$2 \ln \frac{L_0}{L_{\text{cond}}} < \chi^2_{\text{critical}}(\text{limit}, \text{DOF}) \tag{3.13}$$

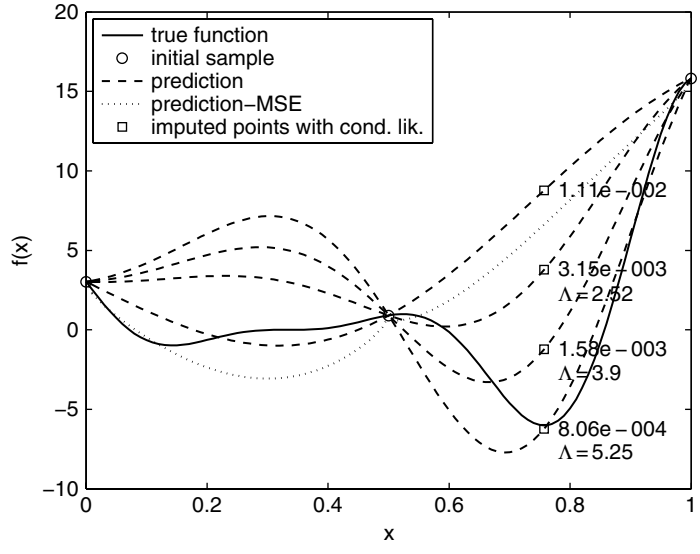



**Figure 3.18.** The conditional likelihood and likelihood ratio for hypothesized points with increasingly lower objective function values.

must be satisfied. The value of the critical $\chi^2$ value will depend upon the confidence limit that we wish to obtain and the number of degrees of freedom (DOF) (the number of model parameters). For the example in Figure 3.18, if we wish to obtain a confidence interval of 0.95, we use limit = 0.975 (we are only considering the lower bound) and DOF = 1 to obtain $\chi^2_{\text{critical}} = 5.0239$ (from tables or using `chi2inv(0.975,1)` in *MATLAB*). Figure 3.18 shows the likelihood ratio for each hypothesized point that has been imputed, calculated using the conditional likelihoods shown. The lowest value would be rejected based on $\chi^2_{\text{critical}}$.

Using this likelihood ratio test we can systematically compute an upper and lower confidence bound for the prediction. In practice it will be the lower bound that will be of use in formulating an infill criterion. To find the lower bound we minimize $y^h$ by varying $y^h$ and the model parameters, subject to the constraint defined by Equation (3.14). The minimum of this lower bound can then be used as an infill criterion. So to choose a new infill point we must minimize $y^h$ by varying $y^h$, $\mathbf{x}$ and the model parameters, subject to the constraint defined by Equation (3.14). In *MATLAB* this constraint may be defined as follows:

```
function [C,Ceq]=likelihoodratiotest(x)
% Performs a likelihood ratio test to evaluate whether a
% hypothesized point at x falls within a confidence limit
%
% Inputs:
%   x – 1 x 2k vector of y^h, theta parameter and hypothesized
%   point
%
% Global variables used:
%   ModelInfo.X – n x k matrix of sample locations
%   ModelInfo.y – n x 1 vector of observed data
%   ModelInfo.Theta – 1 x k vector of log(theta)
%   ModelInfo.U – n x n Cholesky factorization of Psi
%   ModelInfo.Confidence – scalar confidence limit
%
% Outputs:
%    C – negative if hypothesis within confidence limit
%    Ceq – empty equality constraint
%
% Calls:
%    predictor.m, condlikelihood.m

global ModelInfo
k=size(ModelInfo.X,2);

% Catch when location coincides with sample data
if ismember(x(end),ModelInfo.X)
    C=-chi2inv(ModelInfo.Confidence,k);
    Ceq=[];
else
```

(*continued*)

```
    % Prediction at hypothesized point
    ModelInfo.Option='Pred';
    ModelInfo.Goal=predictor(x(k+2:2*k+1));

    % L_O
    L0=condlikelihood([ModelInfo.Theta x(k+2:end)]);

    % L_cond
    ModelInfo.Goal=x(1);
    Lcond=condlikelihood(x(2:end));

    % Value of C must be less than zero
    C=2*(-L0+Lcond)-chi2inv(ModelInfo.Confidence,k);

    % Empty equality constraint
    Ceq=[];
end
```

Starting from $\widehat{y}(\mathbf{x})$, the lower bound at $\mathbf{x}$ can be found using a constrained local search (there are no local minima). Thus, instead of a global search using a genetic algorithm, it may be quicker to use multiple restarts of a hill climber. Figure 3.19 shows the progress of a

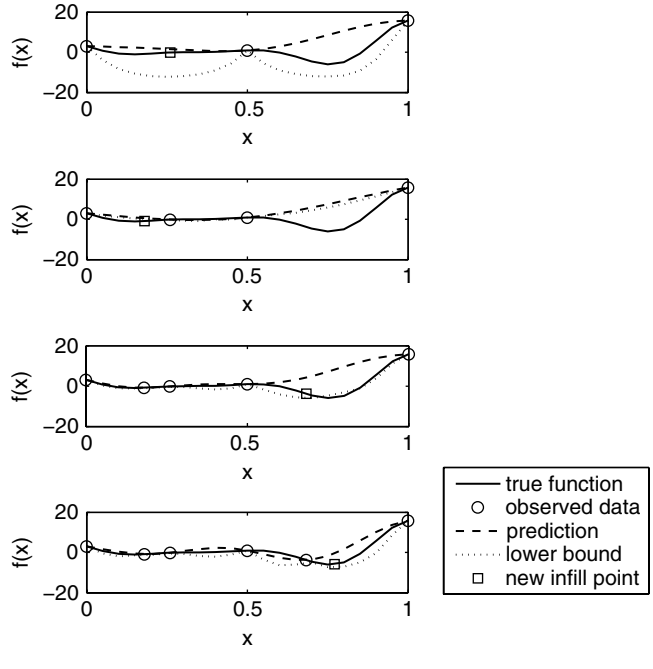



**Figure 3.19.** The progress of a search of the one-variable test function in the range [0, 1] using a conditional lower bound infill strategy.

search of the deceptive one-variable test problem using this infill criterion, starting from the same three-point initial sample. A 95 % confidence interval has been chosen, with *MATLAB*'s genetic algorithm used to find the minimum of the lower bound. This infill criterion can be applied using the following *MATLAB* code:

```
% confidence interval
ModelInfo.Confidence=0.975;
for I=1:4
    % Tune Kriging model of objective using upper and lower bounds
    % on theta of 10^-3 and 10^3
    [ModelInfo.Theta,MaxLikelihood]=fminbnd(@likelihood, -3,3);
    % Put Cholesky factorization of Psi into ModelInfo
    [NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood...
    (ModelInfo.Theta);

    % Search lower bound with wide upper and lower bounds on y
    OptVars=ga(@returnx,3,[],[],[],[],[-50 -3 0],[20 3 1],...
    @likelihoodratiotest);

    % add infill point and calculate objective
    ModelInfo.X(end+1)=OptVars(end);
    ModelInfo.y(end+1)=onevar(OptVars(end));
end
```

**Mathematical Note: The Likelihood Ratio Test**

Likelihood ratios are used to test *nested hypotheses* which are dependent upon different numbers of parameters (known as degrees of freedom). If we wish to test the performance of a hypothesis $H$ which has $n$ additional degrees of freedom to our original hypothesis $H_0$, we calculate the maximum likelihood of an outcome using each hypothesis and calculate the ratio:

$$\Lambda = \frac{\text{MLE}(H_0)}{\text{MLE}(H)} \tag{3.14}$$

The distribution of likelihood ratio, $\Lambda$, can be approximated to the $\chi^2$ cumulative distribution function by $\chi^2 = -2\ln(\Lambda)$. We can therefore compare the $-2\ln(\Lambda)$ to the critical value of the $\chi^2$ distribution with $n$ degrees of freedom and reject or accept hypothesis $H$ over $H_0$ accordingly. Figure 3.20 shows the $\chi^2$ cumulative distribution function for varying degrees of freedom. The dashed line shows the critical value for a confidence limit of 0.975 for two degrees of freedom and is approximately 7.4. Thus, if $-2\ln(\Lambda)$ is greater than this critical value, we should reject $H$ because it is significantly worse than $H_0$.

Note that, in accordance with how the likelihood ratio is used to compute infill criteria, we are testing to see how much worse $H$ is compared to $H_0$. In statistical theory it is more usual to test whether the hypothesis with more degrees of freedom is significantly better than $H_0$.

(*continued*)

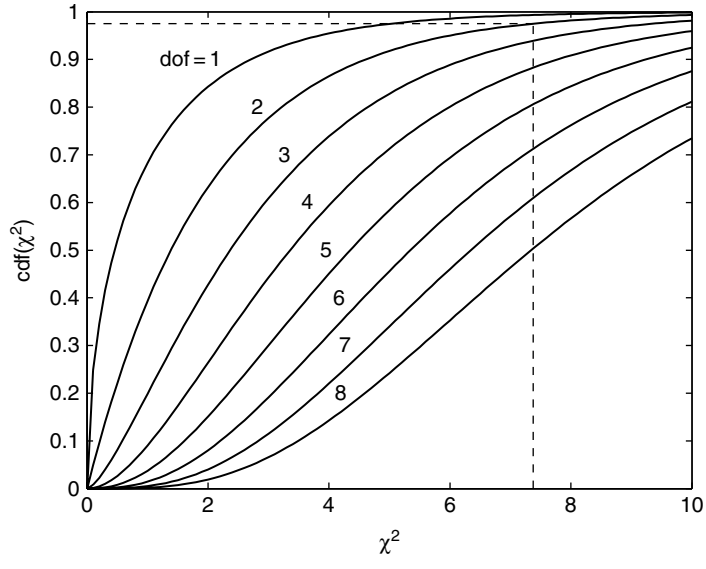



**Figure 3.20.** The $\chi^2$ cumulative distribution function for varying degrees of freedom.

### *3.2.5 Other Methods*

We have already taken up a lot of space with our discourse on infill criteria, and rightly so, since this is a key element of surrogate based design. We have covered what we believe represents the core building blocks of surrogate model infill criteria. However, this is an active area of research and there are many permutations of the criteria we have looked at. We will briefly consider two types of infill strategy, derived from those we have already considered.

#### Parallel Infill Points

It may be that it is possible to conduct objective function evaluations in a parallel computing environment. In such cases it would be helpful to identify a number of infill point locations and evaluate these in parallel.

We can identify multiple points in one of two ways. With no modification to the infill criteria presented above, we can identify multiple local optima of the infill criterion landscape in question (notice the multiple peaks in the conditional likelihood in Figure 3.17) using multiple starts of a local optimizer or clusters of points found by a global optimizer. The reader interested to know more should consult Sóbester *et al*. (2004) for more information about this technique. A problem with the above technique is that we never know how many points we are going to get – there could be any number of local optima – thus making it difficult to align this technique with parallel computing capabilities.

We can choose how many infill points are to be applied by modifying any of the error based infill criteria. We first choose an infill point based on the criterion. Then we impute a new sample point $y^{(n+1)} = \widehat{y}^{(n+1)}$ at this location. A new infill point is then chosen based on

the augmented set of sample data. The process is repeated for the number of infill points required. This method is described in the context of $E[I(\mathbf{x})]$ by Schönlau (1997). The main problem with the method is that there is no guarantee that there will be a significant distance between the infill points found.

**Hybrid Criteria**

We have looked at many methods which strive for a good balance between exploitation and exploration. There is a body of research material that attempts to tailor the balance to the problem at hand – something which, it could be argued, the methods already considered do. Take, for example, the $E[I(\mathbf{x})]$ criterion. The first term is based more on the level of improvement at $\mathbf{x}$ and the second on the amount of error. By changing the ratio of the terms, the balance between exploitation and exploration can be altered. Sóbester *et al*. (2005) presents an in-depth study of the use of this *weighted expected improvement* criterion on functions of varying complexity. The same vein of thought can be applied to the statistical lower bound in Section 3.2.3. A weighted sum of $\widehat{y}$ and $\widehat{s}$ can give a complete sliding scale from exploitation to exploration. A method of dynamically choosing the weighting through reinforcement learning is presented by Forrester (2004).

## 3.3 Managing a Surrogate Based Optimization Process

In the preceding chapters we first looked at how to sample the design space we then presented a variety of surrogate modelling techniques, followed by a look at various infill criteria in this chapter. We will conclude by considering how to manage these various aspects of surrogate based design optimization, followed by an example.

### *3.3.1 Which Surrogate for What Use?*

Table 3.1 contains the various surrogate methods we have covered in Chapter 2 along with what we can do with them, that is either create a global surrogate for visualizing and comprehending the design space or use various infill criteria. The suitability of each surrogate for each use is indicated. Also shown are the numbers of design variables and sample points that the methods can accommodate before becoming too computationally expensive. Naturally, these numbers are somewhat problem-dependent and only serve as a guide. For example, if the analysis code, which the surrogate is being used in lieu of, is extremely expensive, then a parametric surrogate model might be used with $k > 20$ and/or $n > 500$. Likewise, if the analysis is very cheap, it would not be worth training a parametric surrogate unless there were significantly fewer than 20 dimensions. The taxonomy in Table 3.1 should not be taken too literally; its purpose is to provide guidelines for the inexperienced user and different implementations of different methods will blur our clearly defined boundaries.

### *3.3.2 How Many Sample Plan and Infill Points?*

Table 3.1 indicates the number of sample points required for different uses of a surrogate model. If the surrogate model is to be used purely for design space visualization and

**Table 3.1.** A taxonomy of surrogate methods

| Sample plan to infill points ratio | | | | ∞ | | > 2 : 1 | ≈ 1 : 2 | | < 1 : 2 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | Comprehension | | Optimization | | | |
| | | | | Simple landscape | Complex landscape | Local search | $P[I(\mathbf{x})]$, $E[I(\mathbf{x})]$ | Goal seeking | Conditional lower bound |
| $k > 20$ | $n > 500$ | SVR | | ✓ | ✓ | ✓ | | | |
| | | Fixed bases, e.g. cubic, thin plate | | ✓ | ✓ | ✓ | | | |
| | | Polynomials | | ✓ | | ✓ | | | |
| $k < 20$ | $n < 500$ | Parametric bases | Gaussian bases, e.g. Kriging | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | | others | ✓ | ✓ | ✓ | | | |

comprehension, all of the sample points can be chosen by a sampling plan procedure. If we wish to perform local optimization as well, then a few points should be positioned according to an infill criterion. If finding the global optimum design is the overriding objective then most of the points should be positioned using an infill criterion rather than a sampling plan. Although Gaussian process based criteria such as $\max\{E[I(\mathbf{x})]\}$ could be initialized with just a two-point sampling plan, studies have shown that, for this infill criterion, approximately one-third of the total number of points should be in the sampling plan, and two-thirds infill points (Sóbester *et al.*, 2005). Here we are assuming that a maximum number of design evaluations has been specified. While it is most often the case that optimization studies must be conducted within a fixed time or computational resource budget, we should also consider an open-ended scenario when we simply wish to find the best design as quickly as possible. In such situations we can use a small initial sample, though we would recommend that this contain significantly more than two points and use either $\max\{P[I(\mathbf{x})]\}$, $\max\{E[I(\mathbf{x})]\}$ or, if a goal is available or $k/n$ is particularly large, a goal seeking or conditional lower bound search.

### *3.3.3 Convergence Criteria*

Choosing a suitable convergence criterion to determine when to stop the surrogate infill process is rather subjective. We can split the choice of criteria into three categories: exploitation, exploration and balanced exploitation/exploration convergence criteria.

#### Exploitation

When choosing infill points based on minimizing the prediction (exploitation), be it regression or interpolation, the convergence criterion is simple: we stop when the change in a

number of successive infill point objective values is small. The residual value will usually be specified as a percentage of the range of observed objective function values. A similar criterion is to stop when the Euclidean distance between a number of successive infill points becomes very small. As we have shown in Section 3.2.1, the minimum found may not actually be the minimum of the function, but these criteria, if sufficiently stringent, will yield the best value that could be found with pure exploitation of the surrogate prediction.

#### Exploration

Pure exploration is useful when improving the global quality of the surrogate is the object, so a criterion based on convergence to an optimum is not appropriate. Rather, we wish to know when the surrogate will not improve if further points are added, i.e. it has become *saturated*. Here we need to use validation techniques, such as the mean squared error and the correlation coefficient, to compare successive models (recall Section 2.1.3). If the MSE or $r^2$ between predictions from a number of successive surrogates (built from increasing quantities of observed data) plateaus, we can assume that adding more exploration based infill points will not *globally* improve the surrogate. The surrogate might now be used for design space visualization or the infill strategy could be switched to exploitation or balanced exploitation/exploration.

#### Balanced Exploitation/Exploration

When using the probability or expectation of improvement, we can simply stop when the probability is very low or the expectation is smaller than a percentage of the range of observed objective function values. Care should, however, be taken since the estimated MSE of Gaussian process based models is often an underestimate and the search may be stopped prematurely. It is wise to set an overly stringent threshold and wait for a consistently low $P[I(\mathbf{x})]$ or $E[I(\mathbf{x})]$.

Goal seeking is an obvious winner in terms of convergence criteria if the aim of the search is to find a specific objective value, perhaps to beat an existing design by a specified amount, and nothing need be added to the method itself.

When minimizing a lower bound – either $\widehat{y}(\mathbf{x}) - A\widehat{s}(\mathbf{x})$ or the conditional lower bound – there is no quantitative indicator of convergence and we are limited to the convergence criteria used for exploitation. Unfortunately, an infill strategy may dwell in the region of a local minimum before jumping to another, so we cannot guarantee that a series of similar objective values means that the global optimum has been found. Although the conditional lower bound might be a panacea for problems with poor parameter estimates when sampling is sparse or the function is deceptive, it may be beneficial to switch to a maximum $P[I(\mathbf{x})]$ or $E[I(\mathbf{x})]$ infill strategy to conclude the search, in order to facilitate the choice of a stopping criterion.

## 3.4 Search of the Vibration Isolator Geometry Feasibility Using Kriging Goal Seeking

The vibration isolator geometry feasibility problem described in the Appendix, Section A.6, is particularly suited to the goal seeking infill criterion described in Section 3.2.4: there is

a clear goal, i.e. no intersections in the geometry. Also, with 18 design variables, there is little possibility of sampling the design space densely enough to create a globally accurate surrogate. If the goal is not met, the structure cannot be built, so it is not sensible to prescribe a maximum number of design evaluations and choose the initial sample as a fraction of this. We therefore start with an arbitrarily small sampling plan of 20 points and apply infill points at the position of maximum conditional likelihood of the goal. The following *MATLAB* code performs this search and returns a feasible design on the eighth infill point.

```
global ModelInfo
% Create sampling plan
k=18;
n=20;
ModelInfo.X=bestlh(n,k,20,10)

% Calculate observed data
for i=1:n
    ModelInfo.y(i,:)=intersections(ModelInfo.X(i,:));
end

% Search goal
ModelInfo.Goal=0;

% Iterate until goal is attained
while min(ModelInfo.y)>ModelInfo.Goal

    % Tune Kriging model of objective
    options=gaoptimset('PopulationSize',50);
    [ModelInfo.Theta,MaxLikelihood]=...
    ga(@likelihood,k,[],[],[],[],ones(1,k).*-3,ones(1,k).*3,[],...
    options);

    % Put Cholesky factorization of Psi into ModelInfo
    [NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood(ModelInfo....
    Theta);

    % Find location which maximizes likelihood of goal
    options=gaoptimset ('PopulationSize',100);
    [OptVar,NegCondLike]=ga(@condlikelihood,2*k,[],[],[],[],...
    [ones(1,k).*3 zeros (1,k],[ones(1,k).*3 ones(1,k)],[],options);

    % Add infill point and calculate objective value
    ModelInfo.X(end+1,:)=OptVar(k+1:2*k);
    ModelInfo.y(end+1)=intersections(ModelInfo.X(end,:));
end
```

Starting from the same sampling plan, a $\max\{E[I(\mathbf{x})]\}$ based search finds a feasible design on the 72nd infill point. The range of objective function values visited by both infill strategies are shown in Figure 3.21. Although we cannot easily visualize such a high-dimensional design space, we can see from Figure 3.21 that the $\max\{E[I(\mathbf{x})]\}$ has visited

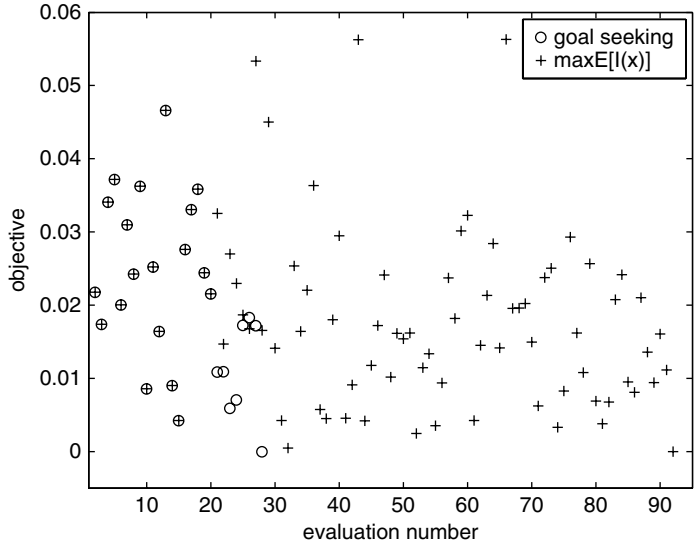



**Figure 3.21.** Objective function evaluation history for the goal seeking and $\max\{E[I(\mathbf{x})]\}$ search of the vibration isolator geometry feasibility. Note that the first 20 points form the sampling plan and so are common to both searches.

many local basins of attraction containing infeasible designs before finally finding a design with no intersections. This is due to the extremely small sampling plan, leading to poor model parameter estimates and many, apparently promising, sparsely populated areas of the design space.

## References

Box, G. E. P. (1957) Evolutionary operation: a method for increasing industrial productivity. *Applied Statistics*, **6**(2), 81–101, June.

Box, M. J. (1965) A new method of constrained optimization and comparison with other methods. *Computer Journal*, **8**(1), 42–52.

Forrester, A. I. J. (2004) *Efficient Global Optimisation Using Expensive CFD Simulations*. PhD thesis, University of Southampton, Southampton, November.

Goldberg, D. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

Hooke, R. and Jeeves, T. A. (1961) Direct search solution of numerical and statistical problems. *Journal of the Association of Computing Machinery*, **8**, 212–229.

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

Jones, D. R. and Welch, W. J. (1996) Global optimization using response surfaces, in Fifth SIAM Conference on *Optimization*, Victoria, Canada, 20–22 May.

Keane, A. J. and Nair, P. B. (2005) *Computational Approaches to Aerospace Design: the Pursuit of Excellence*, John Wiley & Sons, Ltd, Chichester.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983) Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

More, J. J. and Wright, S. J. (1993) Optimization software guide. *SIAM Frontiers in Applied Mathematics*, **14**.

Nelder, J. A. and Mead, R. (1965) A simplex method for function minimization. *Computer Journal*, **8**(1), 308–313.

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

Schönlau, M. (1997) *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada.

Sóbester, A., Leary, S. J. and Keane, A. J. (2004) A parallel updating scheme for approximating and optimizing high fidelity computer simulations, structural and multidisciplinary optimization, 27, 371–383.

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

# Part II
# Advanced Concepts

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

# 4

# Visualization

The advent of the computer has brought about dramatic improvements in our ability to model multidimensional data and multidimensional landscapes. Previously, fitting a model to data using pen and paper would invariably come up against the already mentioned curse of dimensionality, which would often render anything beyond two or three variables intractable.

What has not changed significantly, however, is our (in)ability to build a mental image of a multidimensional model, once such a model has been constructed. In fact, the popularity of the term 'landscape' to refer to such models of functions, regardless of whether they have a single variable or a hundred, indicates that the highest dimensionality we can safely grasp is that of the topography surrounding us. This is, of course, described by a function (height above, say, mean sea level) of precisely two variables (say, latitude and longitude), a rather unsatisfactory number in most engineering applications.

If we also wish to capture our understanding of a landscape in an easily retrievable form, we lose a further variable, as our books and computer screens are flat. We therefore find ourselves resorting to some simple 'cheats', such as surface plots that create a '3D' illusion or contour plots. The latter type either represents the values of a function via its level curves or by mapping its range of values against a range of colours.

A rather ubiquitous example combining the two varieties of contour plot is the synoptic chart used by meteorologists (see Figure 4.1), where the colours represent surface height above mean sea level and the pressure level curves (isobars) depict the surface atmospheric pressure situation at a given time. The pressure chart is, of course, a surrogate model of the real pressure distribution, based on measurements at a network of weather stations, fairly uniformly distributed across the globe (at least on dry land). This example also suggests that the colour/level curve based contour plot is a very efficient way of representing two functions at once. Clearly, in spite of the large amount of information being conveyed

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

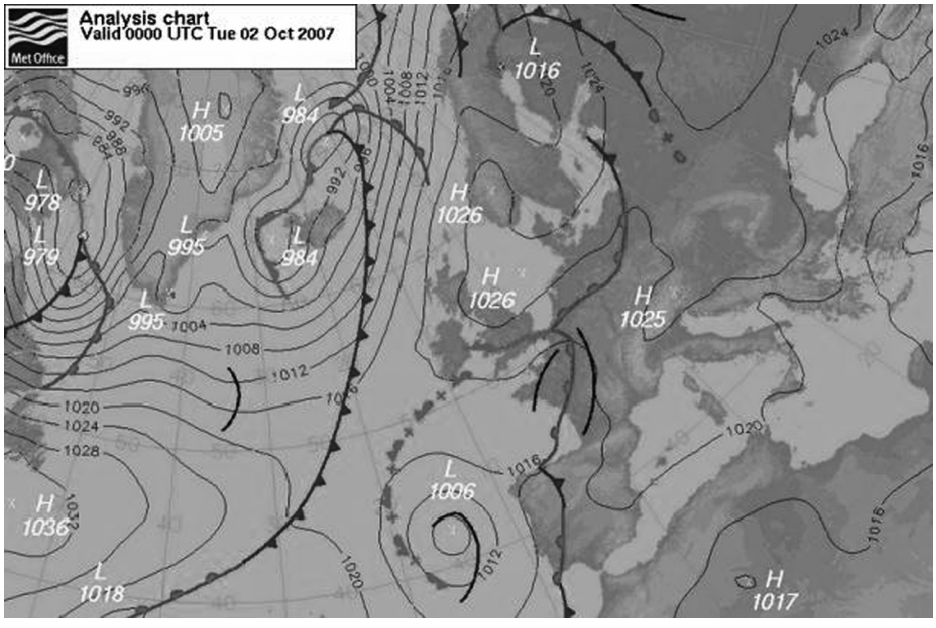



**Figure 4.1.** A flat depiction of two functions of the same two variables – a colour-coded topographic map (of height above mean sea level) and labelled isobars representing surface pressure (This is Crown copyright material which is reproduced with the permission of the Controller of HMSO and the Queen's Printer for Scotland) (See Plate III for colour version).

in a relatively compact manner, there is little chance of confusing a mountain range with an area of high pressure.

Such artifices, however, only buy us back the one dimension lost by transferring our mental image of the function to a flat surface and further tricks are necessary to extend our ability to plot surrogate models and the data they are built upon to higher dimensionalities. We discuss two such techniques next.

## 4.1 Matrices of Contour Plots

A fairly straightforward way of gaining an insight into the features of a multidimensional landscape is to extract strategically placed two-variable slices from it and arrange these as tiles in some systematic way, generally in a matrix. The fundamental question is, where should the projections of these two-dimensional slices be on the remaining dimensions?

In general, considering the problem of representing a $k$-dimensional space, we have little choice but to pick a baseline value in each dimension and hold these constant while sweeping the ranges of all possible two-variable pairs (we shall consider alternatives for specific values of $k$ shortly). We have seen an example of this earlier, in Plate I, a depiction of the

ten-variable wing weight function. This plot was generated using the function `tileplot.m`, the header of which is listed below:

```
function tileplot (Baseline, Range, Labels, Objhandle,...
                                        Mesh, Lower, Upper, Cont)
% Generates the (k-1)x(k-1) tile plot of a k-variable function
%
% Inputs:
%       Baseline - 1xk vector of baseline values assigned to each
%                  variable on a tile where they are not active
%       Range - 2x4 matrix of minimum and maximum values for each
%               variable
%       Labels - cell array containing the names of the variables
%       Objhandle - name of the objective function
%       Mesh - the objective function will be sampled on a mesh x mesh
%              full factorial plan on each tile
%       Lower/Upper - minimum/maximum value of the objective
%                     function - this is required to adjust the colour
%                     range of each tile with respect to the full
%                     colour range of the function (if not known, set
%                     to[] and the function will estimate it).
%       Cont - if Cont = 1 contour lines are plotted and the spaces
%              between them are filled with a colour
%              determined by the function value. Otherwise
%              a colour-shaded plot is generated.
```

As the header above indicates, implementation allows two types of tiles. Firstly, the same function can be represented as a set of level curves and as a top view of a surface whose colouring, varying in the same discrete steps as the spacing of the contours, depends on the function values (`Cont` set to `1`). This is similar, in principle, to the synoptic chart shown in Plate III, but in this case the same function is plotted using both methods to achieve the increased clarity demanded by the small space available for each tile (especially for large $k$). Plate I is an example of this approach. Alternatively, if `Cont` is set to any other value, plain, shaded contour plots result (this is an interpolated checkerboard, as per *MATLAB*'s `pcolor.m`, the resolution of which depends on the value chosen for the variable).

The script `wing.m` illustrates the use of `tileplot.m`. Essentially, the user's own objective function can be defined simply as `function` [f1, ...] = `name(Design)`, where the first output argument `f1` will be plotted by `tilelplot`. The input arguments should be specified in the vector `Design`. If the objective function is constrained, it should return `NaN` (Not-a-Number) for `Design` vectors that violate the constraints – the infeasible regions will then be plotted in white – or the maximum function value (`Upper`), in which case the infeasible area will be greyed out.

## 4.2 Nested Dimensions

It is not always obvious how to choose the baseline set of values where the inactive variables should be held on matrices of contour plots (recall that all variables are inactive, except the two being plotted against each other on a tile). For certain values of $k$ there are work-arounds (Mihalisin *et al.*, 1991). For example, for $k = 4$ we can simply select two variables to plot against each other on each tile and we can generate a matrix of such tiles, where the row and column of a tile determines the values of the remaining two variables. It is possible to imagine various ways in which such hierarchical, nested axes could be used to depict spaces of higher dimensionalities – here we limit ourselves to the four-variable case.

Our implementation of this method, `nested4.m`, features a header similar to that of `tileplot.m`, the main difference between the two being that we need to specify the order in which the variables are assigned to the nested axes:

```
function nested4(Varorder, Div, Range, Labels, Objhandle,...
                                        Mesh, Lower, Upper, cont)
% Generates a four variable hierarchical axis plot
%
% Inputs:
%       Varorder – four – element vector specifying the assignment of
%                  the variables to each of the four axes [main
%                  horizontal main vertical, tile horizontal, tile
%                  vertical]
%       Div – 1 x 2 vector of two variables specifying the number of
%             tiles along the main horizontal and main vertical axes
%             respectively
%       Range – 2 x 4 matrix of minimum and maximum values for each
%               variable
%       Labels – cell array containing the names of the variables
%       Objhandle – name of the objective function
%       Mesh – the objective function will be sampled on a mesh x mesh
%              full factorial plan on each tile
%       Lower/Upper – minimum/maximum value of the objective
%                     function – this is required to adjust the colour
%                     range of each tile with respect to the full
%                     colour range of the function (if not known set
%                     to[] and the function will. estimate it).
%       Cont – if Cont = 1 contour lines are plotted and the spaces
%              between them are filled with a colour determined by the
%              function value. Otherwise a colour-shaded plot is
%              generated.
```

Finally, the following script illustrates the use of this function to represent a four-dimensional radial basis function surrogate model, built upon data generated with a 40-point optimized Latin hypercube sampling plan and the `dome.m` function as the objective. The results are shown in Figure 4.2.

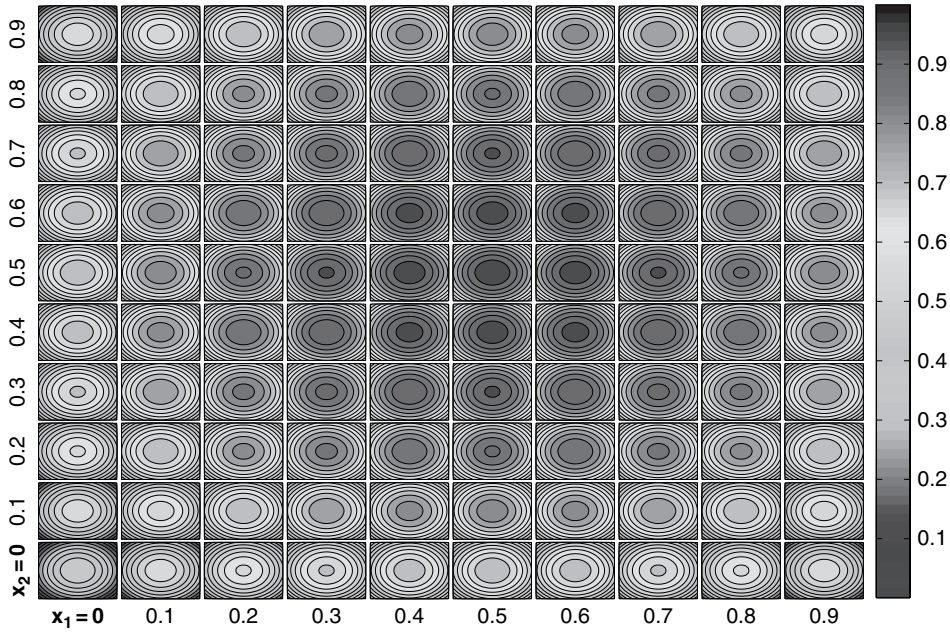



**Figure 4.2.** Four-variable nested plot of the surrogate of the function $f(\mathrm{x}) = 1/4 \sum_{i=1}^{4} 1 - (2x_i - 1)^2$, $\mathrm{x} \in [0, 1]^4$, generated using `nested4.m`. Here $x_3$ varies along the horizontal axis of each tile, $x_4$ along the vertical axes, while the values of $x_1$ and $x_2$ can be read off the bottom of each column of tiles and the beginning of each row respectively (See Plate II for colour version).

```
clear global

global ModelInfo

% Sampling plan
ModelInfo.X = bestlh(50,4,50,75);

% Compute objective function values – in this case using the dome.m
test function – you would insert your own objective function here
for i = 1:size(ModelInfo.X,1)
    ModelInfo.y(i) = dome(ModelInfo.X(i,:));
end

% y must be a column vector
ModelInfo.y = ModelInfo.y';

% Select basis function type:
ModelInfo.Code = 4;

% Estimate model parameters
rbf
```

(*continued*)

```
% Plot the surrogate if the model was successfully fitted
if ModelInfo.Success == 1
   nested4([1 2 3 4], [10 10], [0 0 0 0; 1 1 1 1],...
        {'x_1','x_2','x_3','x_4'}, '@predrbf',...
        30, 0, 1, 1)
else
    display('Could not fit model. Try a different basis
    function.')
end
```

## Reference

Mihalisin, T. Timlin, J. and Schwegler, J. (1991) Visualization and analysis of multi-variate data: a technique for all fields, in *Proceedings of Visualization'91*, San Diego, California, 1991, pp. 171–178.

# 5

# Constraints

Thus far in this book, we have focused on surrogates of a single objective function. Engineering design is almost never so simple in practice – most problems have multiple, often conflicting, goals and invariably a host of ever more demanding constraints, coming from regulatory, safety and environmental concerns, as well as those that simply ensure that the product being designed performs as expected. We turn next to how surrogate based approaches to design improvement work alongside a consideration of constraints – we deal with multiple objectives in Chapter 9.

## 5.1 Satisfaction of Constraints by Construction

The first observation to make about constraints in the design process is that the designer should aim to remove as many as possible at the outset. This is not as pointless a statement as it at first seems – in many cases designers will have seemingly problematic constraints, but at the same time have considerable knowledge on what drives them. Given such knowledge it may be possible to reformulate a design problem, incorporating such knowledge and reducing the number of constraints that must be explicitly modelled.

First one should aim to eliminate any constraints that are unlikely to be active at the optimum being sought, since they will not influence the outcome (it is wise to check that this is so at the end of any search, of course). Then, any constraints that are almost bound to be active, and where a strong relationship with a single design variable exists, should be dealt with. For example, in stress analysis it is common for yield stress or some fraction of it to be used as a constraint on allowable stress. At the same time the designer may well know that certain thicknesses in the structural specification may be very directly related to the working stress levels. Therefore, rather than leaving wall thickness as a design variable and letting stress be constrained, it may be far better to relate stress to thickness directly

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

and simply set a particular thickness based on stress assessments, so that the desired stress levels are achieved, relying on the essentially inverse linear relationship between the two.

Similarly, a common constraint encountered during aerodynamic design of aircraft wing sections and wings is to maintain a fixed lift while varying the geometry to minimize drag. Usually the angle of attack can be freely varied and since lift is so directly controlled by the angle of attack it is natural to try and eliminate the lift constraint by setting the angle so as to satisfy the constraint.

## 5.2 Penalty Functions

In many cases of practical importance it is not possible to eliminate constraints by construction and these must be dealt with directly. Perhaps the most uniformly applicable approach that can be taken when dealing with constrained design is via the mechanism of penalty functions. The approach is essentially very simple – whenever a design is considered that violates (or perhaps nearly violates) one or more constraints, any objective functions being considered are penalized in some way to reflect the concerns involved. If the constraints being dealt with can be rapidly calculated then the resulting penalties can be simply added directly to any surrogate models in use with no concerns over the accuracy of the applied penalties (though choosing the penalty forms still remains an issue).

Often, however, the constraint calculations will be as expensive or perhaps more expensive than evaluating any objectives. For example, calculating a fatigue life using a finite element approach with contact mechanics will be many orders of magnitude more expensive than assessing the weight or even the cost of an engine component. In such circumstances it will be natural to construct a surrogate model for the constraint (or perhaps the penalty to be used which is based on the constraint). The use of surrogates in this way raises a number of further topics that the designer should be aware of.

Firstly, should a constraint surrogate be built from the same set of design points as is being used for the objective or other constraint surrogates (assuming that more than one surrogate is in use)? This consideration applies to any initial sample set of data and to any update sequences.

Secondly, what functional form should be used for the constraint surrogate? A stress constraint may be more or less complex than the other functions being dealt with and might best be modelled in a different way. This reflects directly back on the first concern of choice of data sets. For example, a complex constraint may well require more data during surrogate construction to achieve an acceptable level of accuracy than a relatively simple objective function.

Thirdly, the designer must be aware that constraints are always used in conjunction with some form of limit; that is we are generally interested in a level curve (or contour) of the constraint function. It is often the case that such curves are considerably more sensitive to modelling errors than the broad overall shape of the function being represented. A small change in surrogate modelling may well shift the position of the level curve considerably across the design space with a resulting significant change in constraint violation. This may lead the designer to control any penalty in a different way than would be natural given direct evaluations of the constraint. Typically it may be wise, especially at the outset of a design search, to use a less severe penalty function than later on when a more accurate surface has been constructed through the process of surrogate update. This way of working fits naturally with the penalty mechanisms originally designed for use with sequential unconstrained

minimization techniques (SUMT) (see, for example, Siddall, 1982). In such schemes a moderate initial penalty applied to both sides of the constraint boundary is typically used to begin with, which is then increased in severity as more data is accumulated, especially when applied to results close to the constraint limit itself.

The simplest pure penalty approach, the so called *one pass external function*, is to just add a very large constant to the objective function value wherever any constraint is violated (or if we are maximizing to just subtract this): i.e. $f_p(\mathbf{x}) = f(\mathbf{x}) + P$ if any constraint is violated, otherwise $f_p(\mathbf{x}) = f(\mathbf{x})$. Then the penalized function is searched instead of the original objective. Provided the penalty added ($P$) is very much larger than the function being dealt with, this will create a severe cliff in the objective function landscape that will tend to make search methods reject infeasible designs. This approach is *external* because it is only applied in the infeasible regions of the search space and *one pass* because it is immediately severe enough to ensure rejection of any infeasible designs. Although simple to apply, the approach suffers from a number of drawbacks:

1. the slope of the objective function surface will not, in general, point towards feasible space in the infeasible region so that if the search starts from, or falls into, the infeasible region it will be unlikely to recover from this;
2. there is a severe discontinuity in the shape of the penalized objective at the constraint boundary and so any final design that lies on the boundary is very hard to converge to with precision, especially using efficient gradient descent methods (and commonly many optima in design are defined by constraint boundaries);
3. it takes no account of the number of constraint violations at any infeasible point.

Because of these limitations a number of modifications have been proposed. First, a separate penalty may be applied for each violated constraint. Additionally the penalty may be multiplied by the degree of violation of the constraint and, thirdly, some modification of the penalty may be made in the feasible region of the search near the boundary (the so-called *interior* space). None of these changes is as simple as might at first be supposed.

Consider first adding a penalty for each violated constraint: $f_p(\mathbf{x}) = f(\mathbf{x}) + mP$ where $m$ is the number of violated constraints. This has the benefit of making the penalty more severe when multiple constraints are violated. Care must be taken, however, to ensure that the combined effect does not cause machine overflow. More importantly, it will be clear that this approach adds more cliffs to the landscape – now there are cliffs along each constraint boundary so that the infeasible region may be riddled with them. The more discontinuities present in the objective function space, the harder it is to search, particularly with gradient based methods.

Scaling the total penalty by multiplying by the degree of infeasibility can again lead to machine overflow. Now $f_p(\mathbf{x}) = f(\mathbf{x}) + \sum P\langle|g_i(\mathbf{x})|\rangle + \sum P|h_j(\mathbf{x})|$, where the inequality constraints $g_i(\mathbf{x})$ are taken to be greater than zero, the equality constraints $h_j(\mathbf{x})$ equal to zero and the angle brackets $\langle . \rangle$ are taken to be zero if the constraint is satisfied but return the argument value otherwise. In addition, if the desire is to cause the objective function surface to point back towards feasibility then knowledge is required of how rapidly the constraint function varies in the infeasible region as compared to the objective function. If multiple infeasible constraints are to be dealt with in this fashion, they will need normalizing together so that their relative scales are appropriate (consider dealing with a stress infeasibility in

Pa and a weight limit in metric tonnes – such elements will commonly be six orders of magnitude different before scaling). If individual constraint scaling is to be carried out we need a separate $P_i$ and $P_j$ for each inequality and equality constraint respectively: $f_p(\mathbf{x}) = f(\mathbf{x}) + \sum P_i \langle |g_i(\mathbf{x})| \rangle + \sum P_j |h_j(\mathbf{x})|$. Finding appropriate values for all these penalties requires knowledge of the problem being dealt with, which may not be immediately obvious – in such cases much trial and error may be needed before appropriate values are found.

Providing an interior component for a penalty function is even harder. The aim of such a function is, in some sense, to 'warn' the search engine of an approaching constraint boundary so that action can be taken before the search stumbles over the cliff. Typically this requires yet a further set of scaled penalties $S_i$, so that $f_p(\mathbf{x}) = f(\mathbf{x}) + \sum P_i \langle |g_i(\mathbf{x})| \rangle + \sum P_j |h_j(\mathbf{x})| + \sum S_i / g_i^s(\mathbf{x})$, where the superscript $s$ indicates a satisfied inequality constraint. Since the interior inequality constraint penalty goes to infinity at the boundary (where $g_i(\mathbf{x})$ goes to zero) and decreases as the constraint is increasingly satisfied (positive), this provides a shoulder on the feasible side of the function. However, in common with all interior penalties, this potentially changes the location of the true objective away from the boundary into the feasible region. Now this may be desirable in design contexts, where a design that is on the brink of violating a constraint is normally highly undesirable, but again it introduces another complexity.

All of these penalties have been defined in terms of the design responses in use. If the SUMT approach is used, each penalty varies depending on the stage of the search, so that they are weakened at the beginning of an optimization run when the search engine is exploring the landscape, with suitably severe penalties applied at the end before the final optimum is returned; i.e. the $P_i$, $P_j$ and $S_i$ all become functions of the search progress. Typically the $P$ values start small and increase while the $S$ values start large and tend to zero. To see this consider Figure 5.1 (after Siddall, 1982, Figure 6.27) where a single inequality constraint is shown for a two-dimensional problem and the final optimal design is defined by the constraint location. It is clear from the cross-section that the penalized objective evolves as the constraints change, so that initially a search approaching the boundary from either side sees only a gentle distortion to the search surface, but finally this ends up looking identical to the cliff of a simple one pass exterior penalty. A number of variants on these themes have

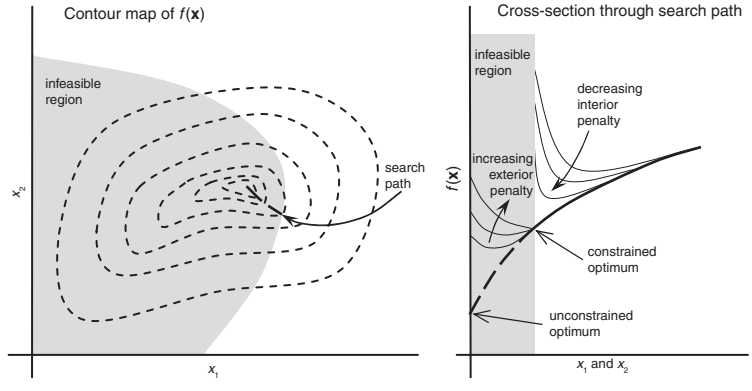



**Figure 5.1.** Evolution of a penalty function in a sequential unconstrained minimization technique (SUMT) (after Siddall, 1982, Figure 6.27).

been proposed and they are more fully explained by Siddall (1982), but it remains the case that the best choice will be problem specific and often therefore a matter of taste and experience. Here we simply adopt the one-pass external function and instead we focus on the role of surrogate models.

## 5.3 Example Constrained Problem

To illustrate these issues we begin by examining a simple test problem based on the Branin function (see the Appendix, Section A.2). In this case we add the simple constraint that the product of the two variables should be greater than 0.2; i.e. the feasible region is above the hyperbola in Figure 5.2. Here the global constrained optimum lies at (0.96773, 0.20667) where the optimum design is defined by the intersection of the objective function surface and the product constraint and the function value is 5.5757. Note that the product constraint is relatively simple but that the nature of the objective function leads to subtle variations of the objective along the boundary itself, with three areas where the global optimum might lie.

### *5.3.1 Using a Kriging Model of the Constraint Function*

We will first consider a search of this constrained Branin function where we assume that the Branin function itself is very quick to evaluate so we do not need a surrogate. However, the product constraint is expensive to evaluate and we need to employ a surrogate (here we will use Kriging) in order to evaluate the function many times during the search.

To start with, following the logic already applied for constructing surrogates of goal functions we sample the design space using an optimal Latin hypercube of six points (recall

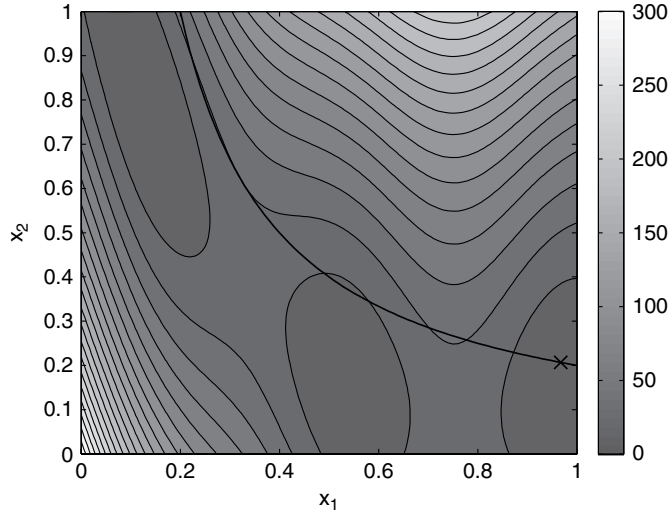



**Figure 5.2.** The constrained Branin test function. The cross indicates the location of the true optimum.

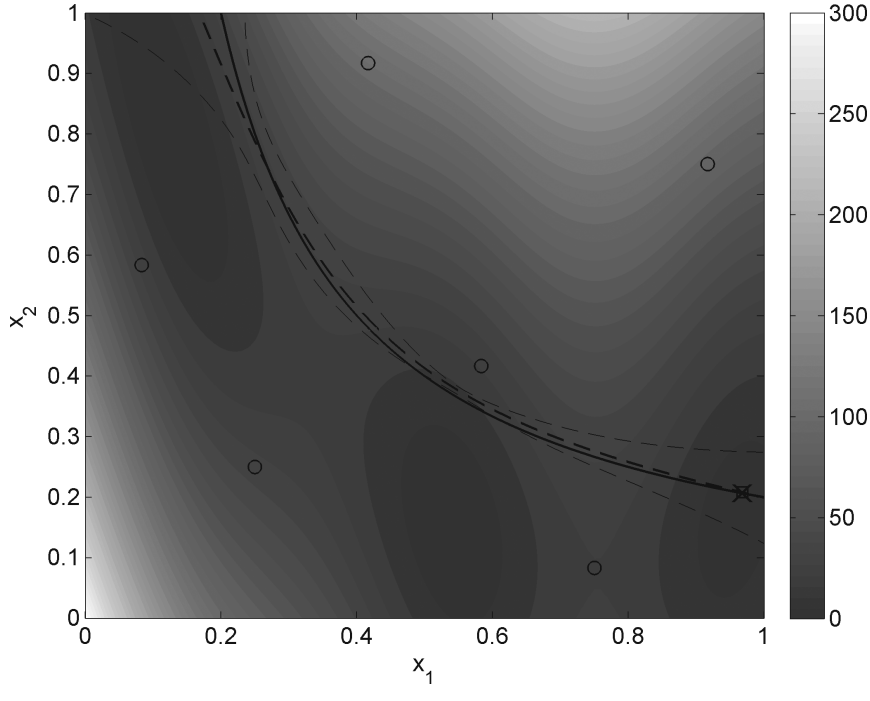



**Figure 5.3.** Contours of the Branin function, the product constraint level curve (black) and Kriging estimate (bold red), together with ± one standard error (fine red). The figure also shows the location of the data points used to sample the constraint (circles), the minimum of the function, subject to the Kriging model of the constraint (square) and the true optimum (cross) (See Plate IV for colour version).

Chapter 1). These are then used to build a suitably trained Kriging prediction (see Section 2.4) of the product constraint function. We then intersect this function with the plane at 0.2 to gain the level curve that defines the constraint boundary; this may be compared to the actual level curve of the constraint (see Figure 5.3), which also shows the locations of the sample points. The differences in these curves illustrate the actual errors in the surrogate.

Also shown on the plot are the level curves generated by using the error estimates coming from the Kriging process. We simply add or subtract one MSE from the mean predictions of the Kriging prediction of the constraint function and intersect the result at 0.2 to gain estimated error level curves on either side of the mean estimate. It is clear that the actual errors lie within these bounds.

Note that the error curves show significant curvature because of the placing of the data points used to construct the model. Even so, since constraints apply by way of level curves, having error bounds on them can be useful when carrying out a penalty function search. For example, it may be sensible to use the error bounds to define a region over which a penalty grows as the space is traversed from the feasible interior to the infeasible exterior.

One could, of course, use other forms of surrogate such as radial basis functions. Provided the form chosen is able to reflect the curvature of the functions being dealt with, broadly similar results are obtained.

It is clear from Plate IV that the use of a Kriging model of the constraint has resulted in an optimum (found using a genetic algorithm search) very close to the true answer, the distance between the two being just $2.1 \times 10^{-3}$. At this stage the designer may seek to improve/confirm the predicted optimum by adding an evaluation at the predicted optimal location (note that, while here we have been able to check that we are very close to the true optimum, this would not be possible in a real-time optimization process). This allows the Kriging model to be refined, so that it now contains seven data points with the added point anticipated to be near the region of most interest to the designer. Upon re-tuning the Kriging parameters we arrive at the situation shown in Figure 5.4. By comparing Plates IV and V we can see that the discrepancies in the prediction of the constraint level curve are reduced. The predicted optimum now has an error of just $2.5 \times 10^{-4}$. The infill process can be repeated until a suitable convergence criterion has been reached (see Section 3.3.3).

### *5.3.2 Using a Kriging Model of the Objective Function*

We next repeat this process but we now build a Kriging surrogate of the objective function surface rather than the constraint (we assume now that the Branin function is too expensive to

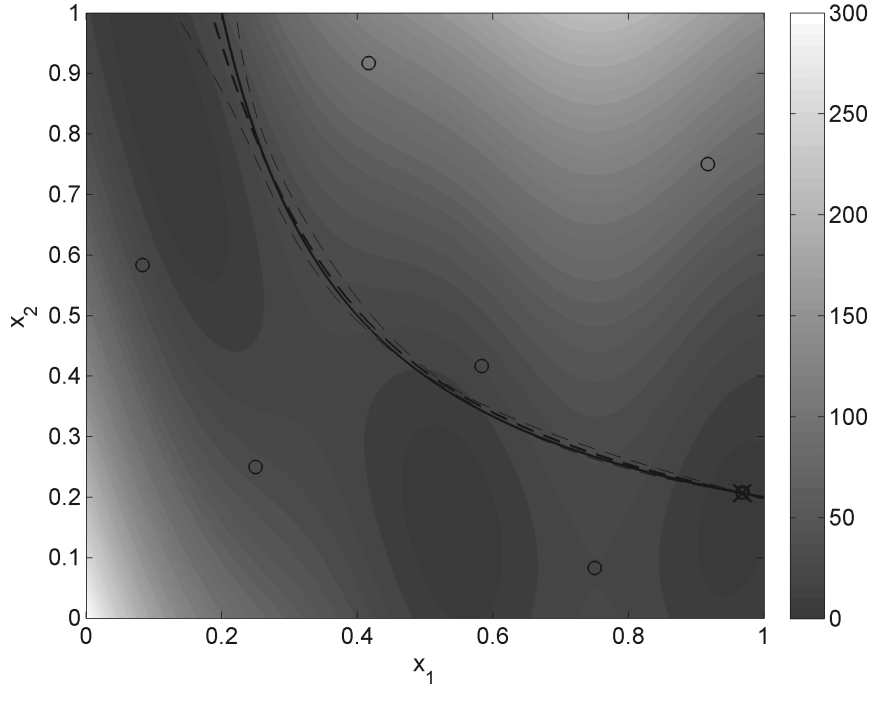



**Figure 5.4.** Contours of the Branin function and the product constraint level curve and Kriging estimate after one infill point (See Plate V for colour version).

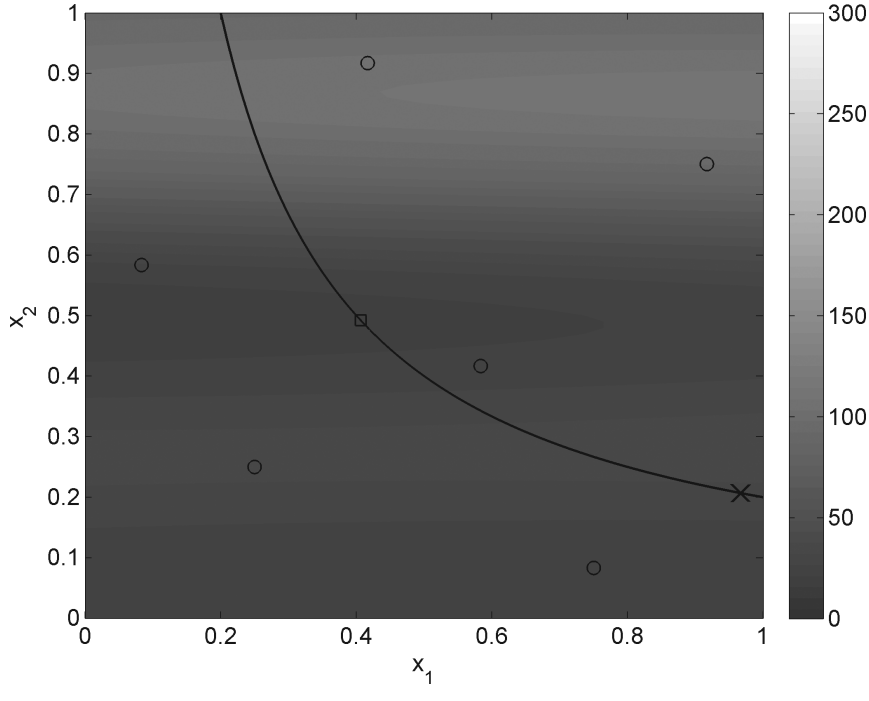



**Figure 5.5.** Contours of a Kriging prediction of the Branin function, the true product constraint level curve, sample points (circles), the optimum of the Kriging prediction subject to the constraint (square) and the true constrained optimum (cross) (See Plate VI for colour version).

search, but the constraint can be evaluated directly). We start by using the same six initial sample points used in the previous calculation, which yield the prediction in Figure 5.5. It is immediately clear that this surrogate is a rather limited model of the true objective and that this will have a significant impact on any optimization process (note that this surface is somewhat more complex than that of the constraint and one might have used more sample points to build the initial model). A search of the Kriging prediction leads to the point shown in the figure, and this may then be used to update the response surface in the normal way.

If five minimum prediction (subject to the constraint) based infill points are applied, the resulting situation is shown in Figure 5.6. The infill strategy has located the region of the global optimum, though not fully exploited it, and it will not explore other possible locations because the initial sampling did not indicate these to be regions of interest; that is the prediction based infill strategy suffers from the problems highlighted in Section 3.2.1. The update points all lie on the constraint boundary, of course, since this is calculated from the true constraint function during the searches.

As we have shown in Section 3.2, an infill strategy that includes an element of exploration is often required to search the design space. We now consider constrained infill strategies which balance exploitation and exploration.

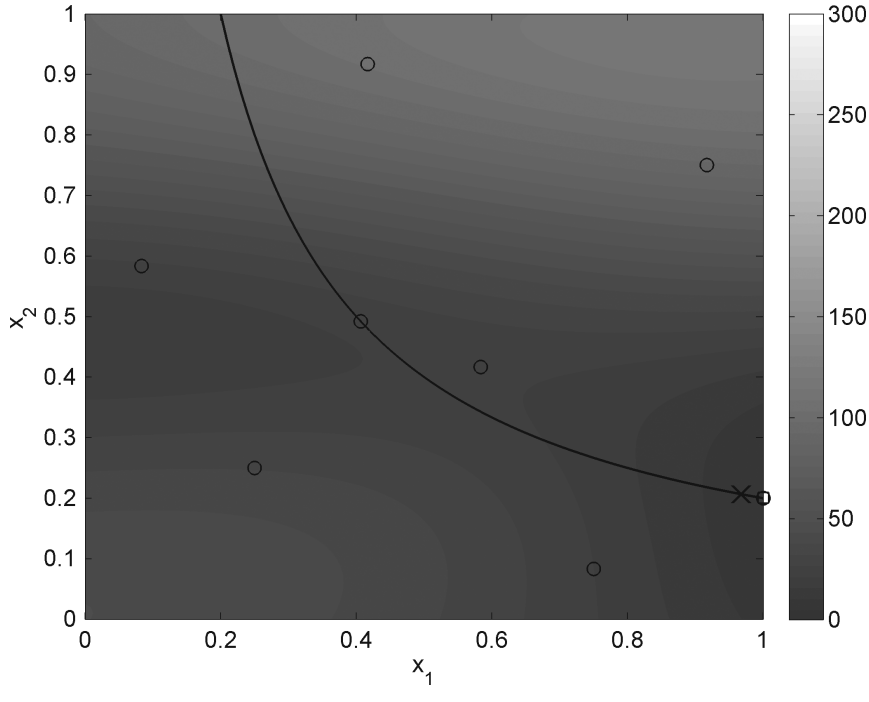



**Figure 5.6.** Contours of the Kriging prediction of the Branin function and the product constraint level curve after five infill points. (See Plate VII for colour version).

## 5.4 Expected Improvement Based Approaches

Having set out what may be achieved by greedily exploiting the surrogates in this constrained problem, we next turn to schemes that seek to balance the quality of the surrogates with the desire to rapidly close on the best solution. This naturally leads to the idea of constrained expected improvement, an approach that builds on the idea of simple expected improvement introduced in Section 3.2.3. There, only the objective was being modelled and the aim was to balance exploration and exploitation. The aim remains the same, but now also using constraint data and, where possible, including constraint uncertainty information in the process.

In unconstrained expected improvement we estimate the probability that a new design will be better than any produced so far, given the current surrogate, leading to new designs both in areas of promising performance and also in places where there is little data and thus significant uncertainty in the predictions. In using the expected improvement idea for constrained problems we must also allow for the feasibility of such new points – clearly if we are certain that a given new point is not feasible then it cannot improve on the design and so its expected improvement must be zero.

This is a simple change to make to our approach if we are working directly with the constraint calculations – We simply set the improvement to zero for all sets of design variables that the direct constraint calculations indicate as violating any constraint. Life

becomes more complex if we also have surrogates for the constraints. In such cases we only have estimates for the objective and the active constraints but, if we use suitable surrogates, we have error measures for these quantities as well. We may then allow for errors in the constraint surrogates as well as in those in the surrogate of the objective function if we wish.

### 5.4.1 Expected Improvement With Simple Penalty Function

We begin by taking our previous example and constructing an expected improvement model for the objective function following the approach outlined in Section 3.2.3, additionally using a Kriging model of the constraint function to remove predicted infeasible points from being considered as the current best in the formulation. Additionally, when searching for new points a simple one-pass penalty function is applied to the expectation operator, i.e. $E[I(\mathbf{x})]_P = E[I(\mathbf{x})] - P$, in the region predicted to violate the constraint (note that this prevents any further exploration of the regions predicted to be infeasible and means that it is not strictly necessary to remove infeasible points in the construction of the expectation; still, we prefer to exclude them in this way since this is in keeping with the logic of the probability of improvement – an issue we return to shortly).

Figure 5.7 shows the expected improvement contours of the objective function model laid over those of the true objective function, along with the constraint boundary and six initial sample points. We have only plotted the expected improvement in the region which satisfies the Kriging model of the constraint, since no infill points will be applied in the region where the constraint is not satisfied because of the penalty function. The expected improvement falls away at each sample location since there is no chance of improvement in these locations (clearly visible for the sample point at $x_1 \approx 0.6,\ x_2 \approx 0.4$). Figure 5.8 shows the effect of adding an infill point to the maximum shown in Plate VIII. The point which maximizes the expected improvement inside the feasible region is now close to the global optimum.

Further infill points begin to explore the surrogate: Figure 5.9 shows that, after nine updates, six points have been positioned in promising regions along the constraint boundary and a further three along the upper limit of $x_1$ where there is a trough in the function. The next infill point will be in the local optimum close to where the upper limit of $x_2$ intersects the constraint boundary. Clearly this method is more appropriate for deceptive functions, where a global search routine is required, than the previous methods covered in this chapter, although a deceptive constraint function would present problems: it would cause the penalty to be applied in the wrong areas.

### 5.4.2 Constrained Expected Improvement

We now take a fully probabilistic approach to the added constraint instead of the simpler one-pass penalty function. This requires a new formulation to extend that presented in Section 3.2, which we outline next. However, before delving into the mathematical presentation it is useful to set out what we might expect when using Gaussian process (e.g. Kriging) models for all the quantities of interest in this way. If, at a given point in the design space, the predicted errors in the constraint models are low and the surrogate shows a constraint violation then the expectation of improvement will also be low, but not zero, since there is a finite possibility that a full evaluation of the constraints may actually reveal a feasible design. Conversely, if the errors in the constraints are large then there will be a

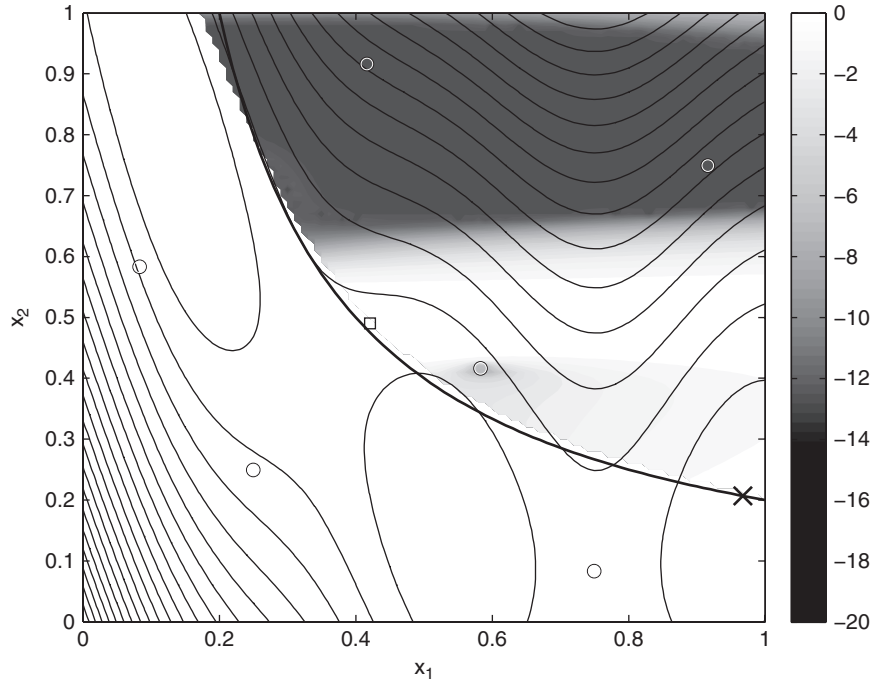



**Figure 5.7.** The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of predicted constraint satisfaction (filled contours) shown together with contours of the true Branin function and the sample points. The true constraint limit is also shown (bold contour), along with the locations of the maximum $E[I(\mathbf{x})]$ (with penalty applied, square) and the actual optimum (cross). The irregular contours at low $\log(E[I(\mathbf{x})])$ are where the $E[I(\mathbf{x})]$ calculation is encountering problems with floating point underflow. See the mathematical note in Section 6.2.1 for more information and a way to avoid this (See Plate VIII for colour version).

significant chance that the constraint predictions are wrong and that a new point will, in fact, be feasible. Thus the expectation of improvement will be greater. Clearly, for a fully probabilistic approach we must factor these ideas into the calculation of the expectation. It turns out that this is relatively simple to do, although it is rarely mentioned in the literature (a formulation can be found in the thesis of Schönlau (1997)). Provided that we assume that the constraints and objective are all uncorrelated, a closed-form solution can readily be derived. If not, and if the correlations can be defined, then numerical integration in probability space is required. Since such data is almost never available this idea is not pursued further here.

In Section 3.2.3 we discussed the probability of improvement infill criterion. Now consider a situation when we have a constraint function, also modelled by a Gaussian process, based on sample data in exactly the same way. Rather than calculating $P[I(\mathbf{x})]$, we could use this model to calculate the probability of the prediction being greater than the constraint limit, i.e. the probability that the constraint is met, $P[F(\mathbf{x})]$. The probability that a design is feasible will be calculated following the same logic as for an improvement, only now

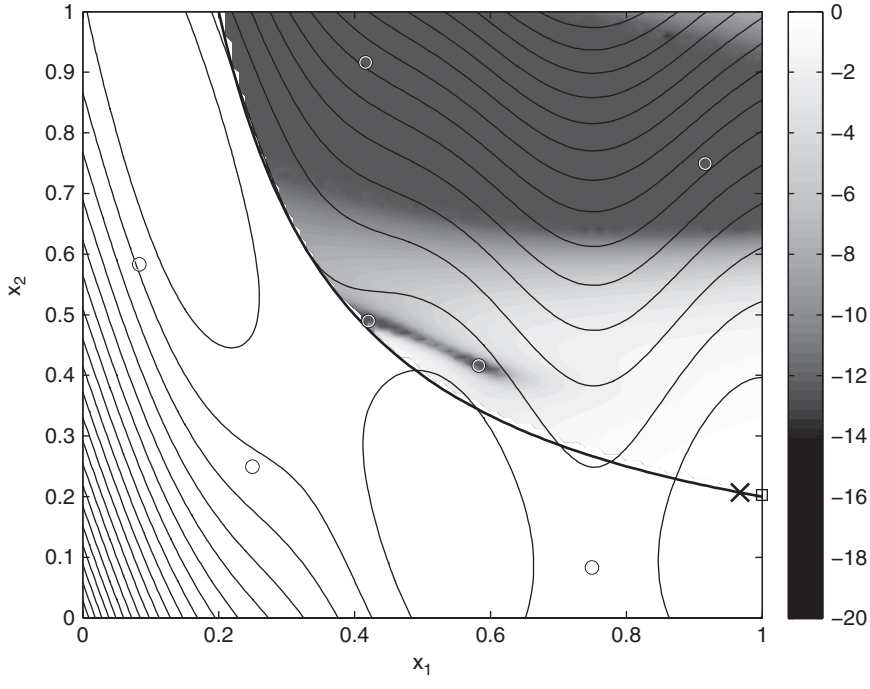



**Figure 5.8.** The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of predicted constraint satisfaction after one infill point (key as Plate VIII). The maximum $E[I(\mathbf{x})]$ (with penalty applied) has located the region of the global optimum (See Plate IX for colour version).

instead of using the current best design as the dividing point in probability space we use the constraint limit value, i.e.

$$P[F(\mathbf{x})] = \frac{1}{\widehat{s}\sqrt{2\pi}} \int_0^\infty \mathrm{e}^{-(F-\widehat{g}(\mathbf{x}))^2/(2\widehat{s}^2)} \mathrm{d}G, \tag{5.1}$$

where $g$ is the constraint function, $g_{\min}$ is the limit value, $F$ is the measure of feasibility $G(\mathbf{x}) - g_{\min}$, $G(\mathbf{x})$ is a random variable and $\widehat{s}$ is the variance of the Kriging model of the constraint. We can couple this result to the probability of improvement from a Kriging model of the objective, and the probability that a new infill point both improves on the current best point and is also feasible is then just

$$P[I(\mathbf{x}) \cap F(\mathbf{x})] = P[I(\mathbf{x})]P[F(\mathbf{x})], \tag{5.2}$$

since these are independent models.

We can also use the probability that a point will be feasible to formulate a constrained expected improvement. We simply multiply $E[I(\mathbf{x})]$ (Equation (3.7)) by $P[F(\mathbf{x}) > g_{\min}]$:

$$E[I(\mathbf{x}) \cap F(\mathbf{x})] = E[I(\mathbf{x})]P[F(\mathbf{x})]. \tag{5.3}$$

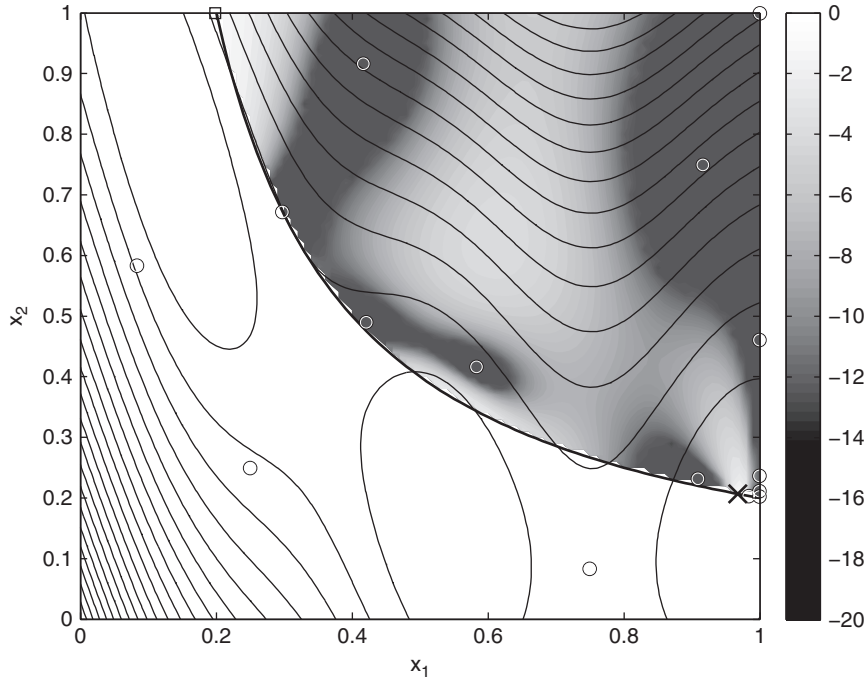



**Figure 5.9.** The $\log(E[I(\mathbf{x})])$ of the Kriging model in the region of constraint satisfaction after nine infill points (key as Plate VIII) (See Plate X for colour version).

In *MATLAB*, this constrained expected improvement can be calculated using the following function, which calls `predictor.m`.

```
function NegLogConExpImp=constrainedei(x)
% Calculates the negative of the log of the
% constrained expected improvement at x
%
% Inputs:
%    x — 1 x k vetor of of design variables
%
% Global variables used:
%    ObjectiveInfo — structure
%    ConstraintInfo — structured cell array
%
% Outputs:
%    NegLogConExpImp — scalar -log(E[I(x)] P[F(x)])
%
global ModelInfo
global ObjectiveInfo
global ConstraintInfo
% Calculate unconstrained E[I(x)]
```

(*continued*)

```
ModelInfo=ObjectiveInfo;
ModelInfo.Option='NegLogExpImp';
NegLogExpImp=predictor(x);
% Calculate P[F(x)] for each constraint
for i=1:size(ConstraintInfo,2)
    ModelInfo=ConstraintInfo{i};
    ModelInfo.Option='NegProbImp';
    NegProbImp(i)=predictor(x);
end

% Calculate E[I(x)]P[F(x)] (add 1e50 before taking logs)
NegLogConExpImp=-(-NegLogExpImp+sum(log10(-NegProbImp+1e-50)));
```

Note how it is necessary to store `ObjectiveInfo` and `ConstraintInfo` as global variables in order that they can be passed to `predictor.m` in turn to find $-\log_{10} E[I(\mathbf{x})]$ and $-P[F(\mathbf{x})]$ respectively, before calculating $-\log_{10} E[I(\mathbf{x}) \cap F(\mathbf{x})]$. `ConstraintInfo` is a cell array structure, in order that a number of constraints can be applied.

To make this last analysis more concrete we return to the test function used in the previous sections and combine the errors from the product constraint and the objective function models following the analysis just set out. We again start by constructing surrogates of the goal and constraint functions by sampling the design space using the six-point sampling plan. These are then used to build the two suitably trained Kriging models, one of the objective and the other of the product constraint function. Now instead of intersecting the constraint with the limit plane at 0.2 to gain the constraint level curve we instead calculate the constrained expected improvement, as per Equation (5.3). This function can be plotted out for variations of the two design variables (see Figure 5.10, we again plot the logarithm). It is important to note that in this formulation the search for $E[I(\mathbf{x}) \cap F(\mathbf{x})]$ is, itself, unconstrained, since the action of the constraint has been merged directly into the objective function – the principal benefit of the approach.

It is immediately apparent from Plate XI that the expected improvement falls away at the sample points as in simple expected improvement. Additionally, the expected improvement drops as the constraint boundary is approached but allows for infeasible points to be sampled to improve the overall model quality (of both objective and constraint). This function can be searched for the location that maximizes the expectation of improvement, where the true functions can be evaluated to update the models, in this case a region near the centre of Plate XI. Figure 5.11 shows the situation after an infill point is added at this location. A noticeable ridge has appeared in the constrained expected improvement, and this ridge is roughly aligned with the constraint limit, since $P[F(\mathbf{x})] \to 0$ in the region which violates the constraint and $P[F(\mathbf{x})] \to 1$ in the region which satisfies the constraint. At this stage it is clear that areas below and to the left of the constraint boundary are mostly returning poor expectations of improvement, except for the extreme lower left corner where the absence of sample data starts to lift the function because there is a small probability that the constraint could be met.

As more update points are added the search space is widely sampled and the ridge in the contours of the constrained expected improvement becomes more closely aligned with the actual constraint boundary (see Figure 5.12, which is plotted after nine updates). Most of

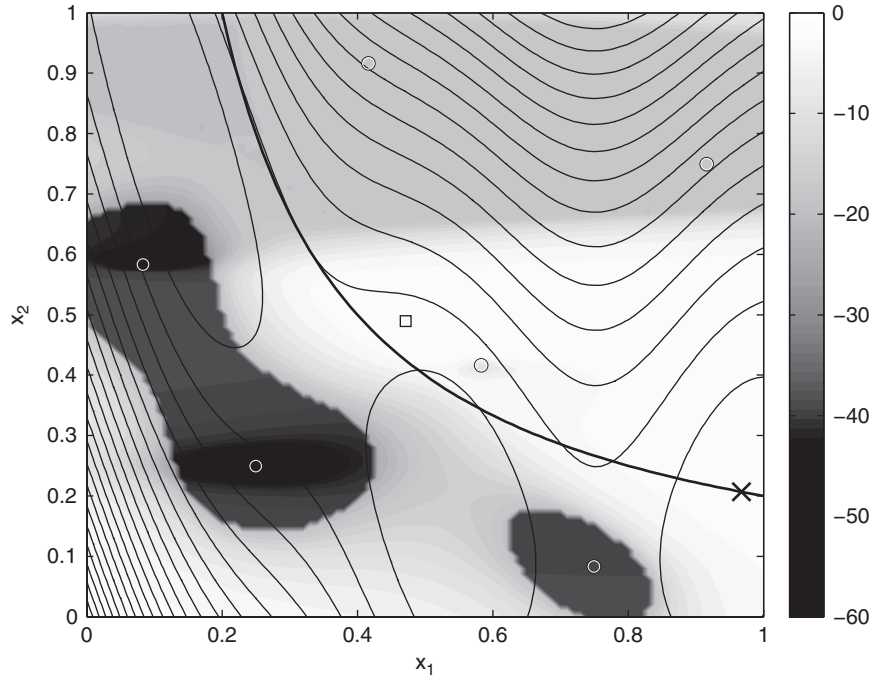



**Figure 5.10.** The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ based on the initial sample of six points (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross). (See Plate XI for colour version).

the nine infill points are located near to the constraint boundary, with a cluster around the global optimum, and the maximum constrained expected improvement is now very close to the global optimum (the error is now $4.1 \times 10^{-3}$).

Because our product constraint is rather simple to approximate, the constrained expected improvement does not perform significantly better than the standard expected improvement with a penalty function applied. However, the constrained expected improvement method is better able to handle more difficult constraint functions and balances the desires of exploration and exploitation more rationally, while avoiding the need for explicit penalty functions.

## 5.5 Missing Data

So far in this chapter we have considered the problem of constrained optimization when a constraint function can be calculated. We will now consider the situation where the constraint takes the form of an inability to calculate the objective function. This situation arises, for example, when a geometry model is used which cannot cope with all parameter combinations or a computational fluid dynamics simulation fails to converge.

In an ideal world, a seamless parameterization would ensure that the optimizer could visit wide ranging areas of the search space and move between them without interruption. In reality, however, the uniform, flawless coverage of the search space is fraught with difficulties. Of course, if areas of objective function failure are rectangular, they can simply

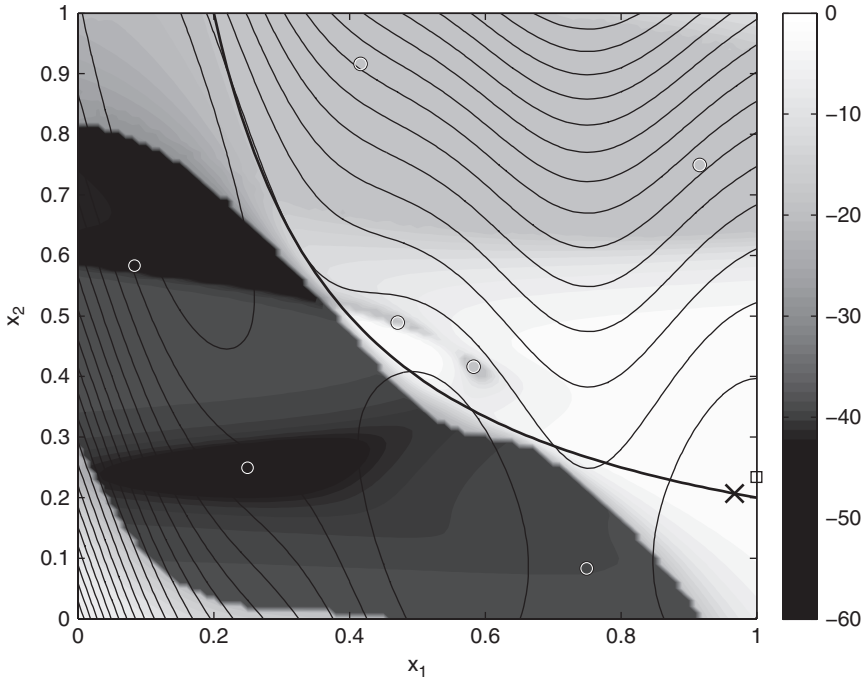



**Figure 5.11.** The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ after one infill point (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross) (See Plate XII for colour version).

be avoided by adjusting the bounds of the relevant variables, but this is rarely the case. Most of the time such regions will have complex, irregular and even disconnected shapes, and are much harder to identify and avoid. In the presence of such problems it is difficult to accomplish entirely automated optimization without narrowing the bounds of the problem to the extent that promising designs are likely to be excluded. This negates the possible benefits of global optimization routines, and the designer may well choose to revert to a more conservative local optimization around a known good design.

Parallels can be drawn between the situation of encountering infeasible designs in optimization and *missing data* as it appears in statistical literature (see, for example, Little and Rubin, 1987). When performing statistical analysis with missing data, it must be ascertained whether the data is Missing At Random (MAR), and therefore ignorable, or whether there is some relationship between the data and its 'missingness'. A surrogate model based on a sampling plan is, in essence, a missing data problem, where data in the gaps between the sample points is MAR, due to the space-filling nature of the sampling plan, and so is ignored when making predictions. When, however, some of the sampling plan or infill points fail, it is likely that this missing data is not MAR and the missingness is in fact a function of $\mathbf{x}$.

Surrogate model infill processes may be performed after ignoring missing data in the sampling plan, whether it is MAR or otherwise. However, when an infill design evaluation fails, the process will fail: if no new data is added to the model, the infill criterion, be it

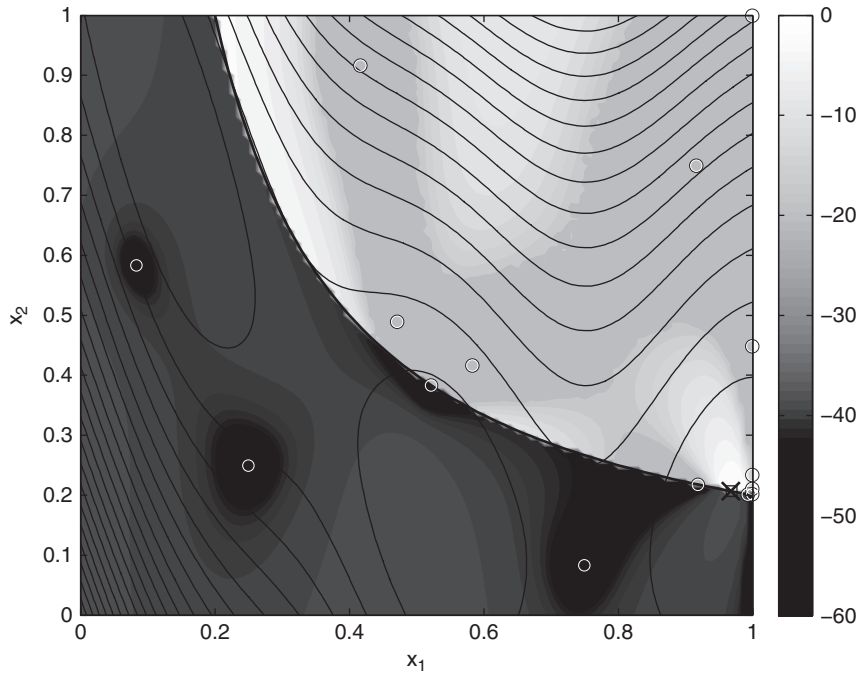



**Figure 5.12.** The $\log(E[I(\mathbf{x}) \cap F(\mathbf{x})])$ after nine infill points (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross) (See Plate XIII for colour version).

based on $\widehat{y}$, $\widehat{s}^2$, $E[I(x)]$ or some other surrogate based criterion, remains unchanged and the process will stall. The process may be jump-started by perturbing the prediction with the addition of a random infill point (a succession of random points may be required before a feasible design is found). However, ignoring this missing data may lead to distortions in the surrogate model, causing continued reselection of infeasible designs, and the majority of the sampling may end up being based on random points. As such, because failed design evaluations are not MAR we should *impute* values to the missing data before training the model (Forrester *et al*., 2006).

### *5.5.1 Imputing Data for Infeasible Designs*

While the statistical literature deals with feasible missing data (data missing due to stochastic sampling) and so the value of the imputation is important, here we are faced with infeasible missing data – there can be no successful outcome to the deterministic sampling.[1] Thus the imputed data should serve only to divert the optimization towards the feasible region. The

[1] There may, of course, be a physical value to our design criterion at the location of the missing data (if the design is physically realizable), which might be obtained through a different solution setup, but here we are interested in automated processes with a generic design evaluation suited to the majority of viable configurations.

process of imputation alone, regardless of the values, to some extent serves this purpose: the presence of an imputed sample point reduces the estimated error (Equation (3.1)), and hence $E[I(\mathbf{x})]$ at this point, to zero, thus diverting further updates from this location. A value better than the optimum may still, however, draw the optimization towards the region of infeasibility. We now go on to consider which is the most suitable model by which to select imputation values when using Kriging.

Moving away from the area of feasible designs, $\|\mathbf{x}^{(n+1)} - \mathbf{x}^{(i)}\| \to \infty$, $\psi^{(i)} \to 0$, and so $\widehat{y}(\mathbf{x}^{(n+1)}) \to \widehat{\mu}$ (from Equation (2.40)). Thus predictions in infeasible areas will tend to be higher than the optimum region found so far. However, the rate at which the prediction returns to $\widehat{\mu}$ is strongly linked to $\widehat{\boldsymbol{\theta}}$. Therefore, for functions of low modality, i.e low $\widehat{\boldsymbol{\theta}}$, where there is a trend towards better designs on the edge of the feasible region, imputations based on $\widehat{y}$ may draw the update process towards the surrounding region of infeasibility. It therefore seems logical to take into account the expected error in the predictor to penalize the imputed points by using a statistical upper bound, $\widehat{y} + \widehat{s}^2$. Now as $\|\mathbf{x}^{(n+1)} - \mathbf{x}^{(i)}\| \to \infty$, $\widehat{y}(\mathbf{x}^{(n+1)}) + \widehat{s}^2(\mathbf{x}^{(n+1)}) \to \widehat{\mu} + \widehat{\sigma^2}$ (from Equations (2.40) and (3.1)), while we still retain the necessary characteristic for guaranteed global convergence: as $\|\mathbf{x}^{(n+1)} - \mathbf{x}^{(i)}\| \to 0$, $\widehat{y}(\mathbf{x}^{(n+1)}) + \widehat{s}^2(\mathbf{x}^{(n+1)}) \to y(\mathbf{x}^{(i)})$, i.e. our imputation model interpolates the feasible data and so does not affect the asymptotic convergence of the maximum $E[I(\mathbf{x})]$ criterion in the feasible region. Although we advocate the use of $\widehat{s}^2$ to penalize imputations, one could use other error based metrics. The advantage of using $\widehat{s}^2$, rather than the more intuitive $s$, is that the surrogate will remain smooth when there are imputations very close to feasible points.

To demonstrate this method, instead of using the product constraint function to calculate the feasibility of a point, we will make the Branin function fail in the region of infeasibility: the function `braninfailures.m` returns `NaN` (Not-a-Number) if `prod(x)<0.2`. Since we know that a number of points in our sampling plan will fail, we will use a larger sample. The following *MATLAB* code creates a 12-point sampling plan, identifies the successful and failed points, and builds and trains a Kriging model using the successful points:

```
global ModelInfo
% Create sampling plan
k=2;
n=12;
TotalX=bestlh(n,k,20,10);
% Calculate observed data
for i=1:n
    Totaly(i,1)=braninfailures(TotalX(i,:));
end

% Find successful points
ySuc=Totaly(find(~isnan(Totaly)));
XSuc=TotalX(find(~isnan(Totaly)),:);
XImp=TotalX(find(isnan(Totaly)),:);

% Number of imputations
Imps=size(XImp,1);
```

(*continued*)

```
% Only use successful points in Kriging model
ModelInfo.X=XSuc;
ModelInfo.y=ySuc;

% Set upper and lower bounds for search of log theta
UpperTheta=ones(1,k).*2;
LowerTheta=ones(1,k).*-3;

% Run GA search of likelihood
[ModelInfo.Theta,MinNegLnLikelihood]=...
ga(@likelihood,k,[],[],[],[], LowerTheta,UpperTheta);
% Put Cholesky factorization of Psi into ModelInfo
[NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood(ModelInfo.Theta);
```

We then calculate the values to impute at the failed points, $\widehat{y}(\mathbf{x}) + \widehat{s}^2(\mathbf{x})$, using the Kriging model through the successful points, combine the imputations and successful points into one data set, and build and search a Kriging model though these points using $\max\{E[I(\mathbf{x})]\}$:

```
if Imps>0
   % Calculate imputations based on Kriging model through successful
   % points
   for i=1:Imps
         ModelInfo.Option='Pred';
         [PredImp(i,1)]=predictor(XImp(i,:));
         ModelInfo.Option='RMSE';
         [RMSEImp(i,1)]=predictor(XImp(i,:));
   end
   yImp=PredImp+RMSEImp.^2;

   % Concatenate successful and imputed points
   ModelInfo.y=[ySuc;yImp];
   ModelInfo.X=[XSuc;XImp];
   % Build Kriging model through successful and imputed points
   % Tune theta using the MATLAB GA
   [ModelInfo.Theta,MinNegLnLikelihood]=...
   ga(@likelihood,k,[],[], [],[], LowerTheta,UpperTheta);
   % Put Cholesky factorization of Psi into ModelInfo
   [NegLnLike,ModelInfo.Psi,ModelInfo.U]=likelihood(ModelInfo.
   Theta);
end
% Search model
ModelInfo.Option='NegLogExpImp';
options=gaoptimset('PopulationSize',50,'Generations',100);
[OptVar, OptEI]=ga(@predictor,k,[],[],[],[],zeros(k,1),...
ones(k,1), [],options)

% Add infill point
TotalX(end+1,:)=OptVar;
Totaly(end+1)=braninfailures(OptVar);
```

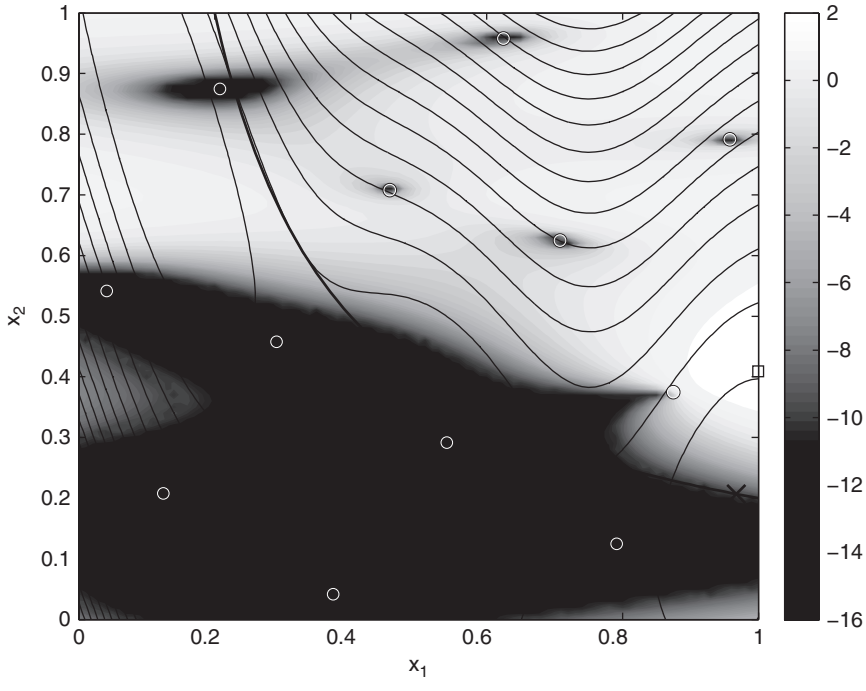



**Figure 5.13.** The $\log(E[I(\mathbf{x})])$ of a Kriging model through the combined successful and imputed data set (filled contours), along with the contours of the true Branin function, the true constraint limit (bold contour), the sample points (circles) and the true optimum (cross) (See Plate XIV for colour version).

Figure 5.13 shows the expected improvement of the Kriging model through the combined successful and imputed data set. $E[I(\mathbf{x})]$ is significantly reduced in areas of infeasibility, though there is no definite ridge along the constraint boundary because we cannot model the constraint directly. Nevertheless, the maximum of the expected improvement is in the region of the global optimum. The above *MATLAB* code can be put inside a loop to iterate the search, infill process. Without constraint function information the search cannot be expected to perform as efficiently as the previous methods in this chapter, however, it does allow a search to continue when sample or infill point evaluations fail.

## 5.6 Design of a Helical Compression Spring Using Constrained Expected Improvement

We will now apply the constrained expected improvement infill criterion to the optimization of a steel spring. The spring design problem described in the Appendix, Section A.5, has two objectives: spring mass and maximum number of cycles to fatigue failure. Here we will assume that mass is of no consequence and simply maximize the number of cycles (in fact, we will minimize its negative) subject to the constraints on shear safety factor and buckling stress. Assuming we have a budget sufficient for 30 spring evaluations, we will use our rule-of-thumb of a 1:2 sampling plan to infill points ratio. The following *MATLAB* code

creates a 10-point sample and augments this with 20 infill points based on maximizing the constrained expected improvement criterion.

Note that the fatigue cycles objective may not yield an answer when one of the constraints has been violated. In such cases, `fatigue cycles.m` returns `NaN` and we must filter out these values when building the Kriging surrogate of the objective.

```
global ModelInfo
global ObjectiveInfo
global ConstraintInfo
% Number of variables
k=3;
% Number of sample points
n=10;

% Create sampling plan
ObjectiveInfo.X=bestlh(n,k,20,10);
ConstraintInfo{1}.X=ObjectiveInfo.X;
ConstraintInfo{2}.X=ObjectiveInfo.X;

% Run 'experiments' to get observed data
for i=1:n
    ObjectiveInfo.y(i,1)=-1*springcycles...
    (ObjectiveInfo.X(i,:));
    ConstraintInfo{1}.y(i,1)=buckling...
    (ConstraintInfo{1}.X(i,:));
    ConstraintInfo{2}.y(i,1)=shearsafetyfact...
    (ConstraintInfo{2}.X(i,:));
end

% Constraint limits
ConstraintInfo{1}.ConstraintLimit=0;
ConstraintInfo{2}.ConstraintLimit=0;

% Start iterating infill points
for I=1:20
    % Use only successful sample points in objective function model
    ObjectiveInfo.X=...
    ObjectiveInfo.X(find(~isnan(ObjectiveInfo.y)),:);
    ObjectiveInfo.y=...
    ObjectiveInfo.y(find(~isnan(ObjectiveInfo.y)));

    % Set upper and lower bounds for search of log theta
    UpperTheta=ones(1,k).*2;
    LowerTheta=ones(1,k).*-3;

    % Tune Kriging model of objective
    ModelInfo=ObjectiveInfo;
    [ObjectiveInfo.Theta,MaxLikelihood]=ga(@likelihood,k,...
    [],[],[],[],LowerTheta,UpperTheta);
```

(*continued*)

```
    [NegLnLike,ObjectiveInfo.Psi,ObjectiveInfo.U]=...
    likelihood(ObjectiveInfo.Theta);

for i=1:size(ConstraintInfo,2)
    % Tune Kriging model of constraint
    ModelInfo=ConstraintInfo{i};
    [ConstraintInfo{i}.Theta,MaxLikelihood]=...
      ga(@likelihood,k,[],[],[],[],LowerTheta,UpperTheta);
    [NegLnLike,ConstraintInfo{i}.Psi,ConstraintInfo{i}.U]=...
      likelihood(ConstraintInfo{i}.Theta);
end
    % Search constrained expected improvement
    options=gaoptimset('PopulationSize',100);
    [OptVar,OptEI]=ga(@constrainedei,k,[],[],[],[],[0 0 0],...
    [1 1 1],[],options);

    % Add infill point
    ObjectiveInfo.X(end+1,:)=OptVar;
    ObjectiveInfo.y(end+1)=-1*springcycles(OptVar);

    ConstraintInfo{1}.X(end+1,:)=OptVar;
    ConstraintInfo{2}.X(end+1,:)=OptVar;
    ConstraintInfo{1}.y(end+1)=buckling(OptVar);
    ConstraintInfo{2}.y(end+1)=shearsafetyfact(OptVar);
end
```

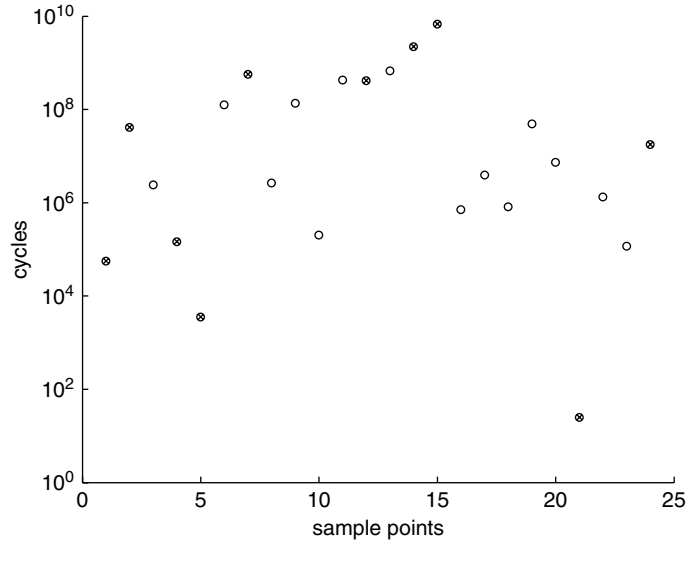



**Figure 5.14.** Number of fatigue cycles to failure at each sample point (circles), with constraint violating points (crosses). The objective function failed for four of the initial samples, with 18 of the infill point evaluations being successful.

Figure 5.14 shows the objective function values of all the successful spring evaluations. Designs which violated one or both constraints are indicated by a cross. The best design, with $6.8 \times 10^8$ cycles to failure, was found to be the seventh infill point. By comparing this to the fatigue cycles values on the Pareto front in Figure A.9 in the appendix, it can be seen that we have successfully identified one end of the optimal trade-off. After the next few infill points, with higher objective values, but which violated constraint(s), the search begins to explore other areas of the design space, but does not find a better objective value.

## 5.7 Summary

In this chapter attention has focused on the role of surrogates when dealing with constrained problems. It has been shown that a range of different approaches can be taken and a key decision required by the designer is whether or not to seek, greedily, the best results at the risk of missing possible global optima or, instead, to try and balance exploitation with exploration to improve surrogate quality while at the same time finding good designs. Various Kriging based alternatives have been set out and demonstrated on a simple two-dimensional example problem.

## References

Forrester, A. I. J., Sóbester, A. and Keane, A. J. (2006) Optimisation with missing data. *Proceedings of the Royal Society A*, **462**(2067), 935–945.

Little, R. J. A. and Rubin, D. B. (1987) *Statistical Analysis with Missing Data*, John Wiley & Sons, Inc, New York.

Schönlau, M. (1997) *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada.

Siddall, J. N. (1982) *Optimal Engineering Design: Principles and Applications*, Marcel Dekker, New York.

# 6

# Infill Criteria with Noisy Data

The infill criteria presented in Section 3.2 were formulated on the premise that the true engineering function to be approximated is smooth and continuous. Thus we can use a smooth, continuous interpolating surrogate in lieu of calculations of the true function. Most functions that are encountered do indeed have smooth and continuous *trends*, but function evaluations are often scattered about this trend due to errors in the physical experiment or computer simulation used to calculate the function. One of the key advantages of using a surrogate is that the scatter, or noise, can be filtered out, leaving a smooth trend to be searched by the optimization algorithm.

An example of 'noisy' data from a computer experiment is shown in Figure 6.1. The figure shows the drag coefficient ($C_D$) found from 101 Computational Fluid Dynamics (CFD) simulations of an aerofoil as a shape parameter changes (see the Appendix, Section A.3, for more information on the aerofoil data). There is an obvious trend in the data, but the small degree of scatter about this trend can make optimization rather difficult. Figure 6.2 shows a $\max\{E[I(\mathbf{x})]\}$ search of a Kriging approximation starting from a uniform sampling plan of four points. Even for this one-variable example the search is running into trouble after five updates. Note how the prediction is becoming erratic and there is high expected improvement in areas far from the optimum. It is easy to see that the search process is failing and where the true optimum is in this simple problem, but in a high-dimensional problem the update process might continue to search in areas of poor designs based on high, but misleading, expectations of improvement.

We have already considered the most simple form of noise filtering models in Section 2.2, that is polynomial regression. By using any surrogate to filter noise we are assuming that the function is smooth, but polynomial approximations go further to assume that the function takes on a specific form. This approach can naturally lead to under- or overfitting, with key trends in the data being filtered out along with the noise. Instead, an RBF or Kriging model (recall Section 2.4) can be modified to filter noise without having to guess the underlying

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

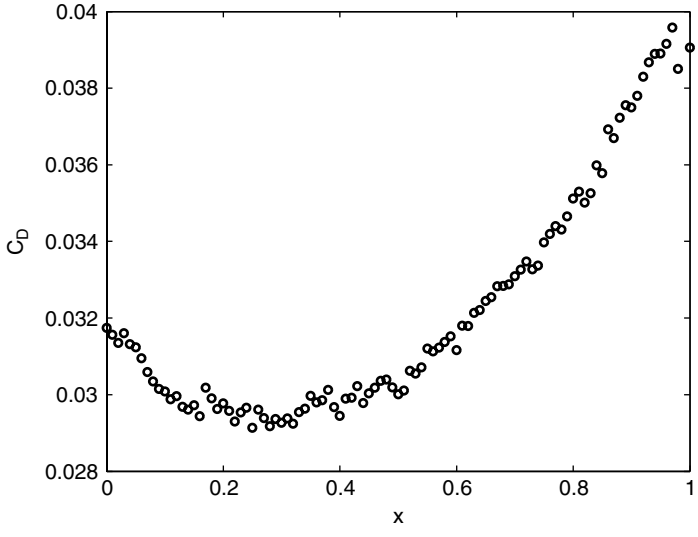



**Figure 6.1.** An example of data containing 'noise' due to dicretization error. See the Appendix, Section A.3, for details on this aerofoil design problem.

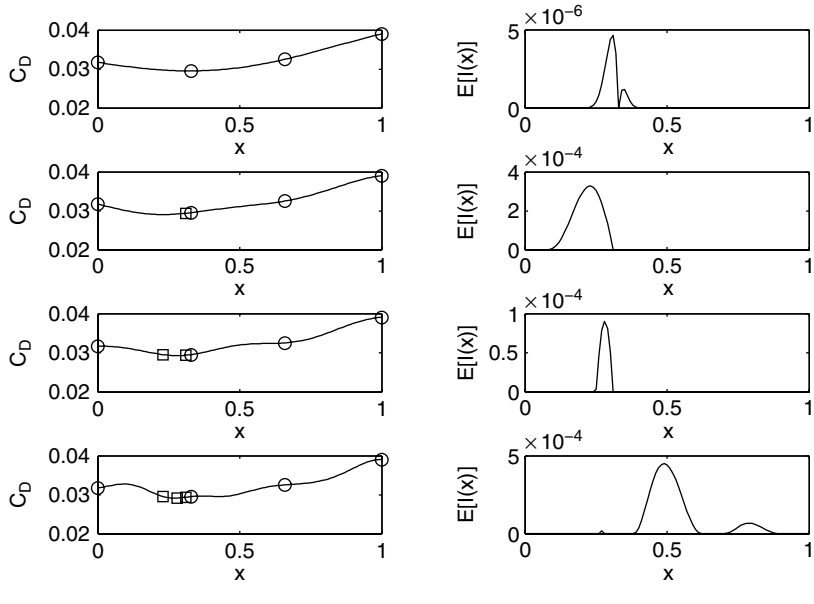



**Figure 6.2.** A series of $\max\{E[I(\mathbf{x})]\}$ updates of a Kriging interpolation. The process is beginning to run into trouble as the $\boldsymbol{\Psi}$ matrix becomes ill-conditioned after the fourth update.

structure of the function. This can also be achieved using support vector regression, but using an RBF or Kriging model, which can be thought of as a SVR with all points as support vectors, has the added benefit of more readily obtainable error estimates.

## 6.1 Regressing Kriging

In the derivation of the Kriging predictor in Section 2.4 we showed how the prediction interpolates the data because at a sample location the vector of correlations of the predicted point with the sample data $\boldsymbol{\psi}$ is a column of the correlation matrix $\boldsymbol{\Psi}$. To filter noise, a regression constant $\lambda$ can be added to the leading diagonal of $\boldsymbol{\Psi}$ (Hoerl and Kennard, 1970; Tikhonov and Arsenin, 1977), that is, we now have $\boldsymbol{\Psi} + \lambda \mathbf{I}$ ($\mathbf{I}$ is an $n \times n$ identity matrix), so that, as $|\mathbf{x}^{(i)} - \mathbf{x}| \to 0$, $\text{cor}(\mathbf{x}^{(i)}, \mathbf{x}) = 1 + \lambda$. Now $\boldsymbol{\psi}$ is never a column of $\boldsymbol{\Psi}$ and so the data is not interpolated. Using the same method of derivation as for interpolating Kriging, the regressing Kriging prediction is given by

$$\widehat{y}_r = \widehat{\mu}_r + \boldsymbol{\psi}^{\mathrm{T}}(\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu}_r), \tag{6.1}$$

where

$$\widehat{\mu}_r = \frac{\mathbf{1}^{\mathrm{T}}(\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1}\mathbf{y}}{\mathbf{1}^{\mathrm{T}}(\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1}\mathbf{1}}. \tag{6.2}$$

The last plot of Figure 6.2 is reproduced in Figure 6.3, but with a Kriging regression included, which provides a far more feasible prediction of the true function.

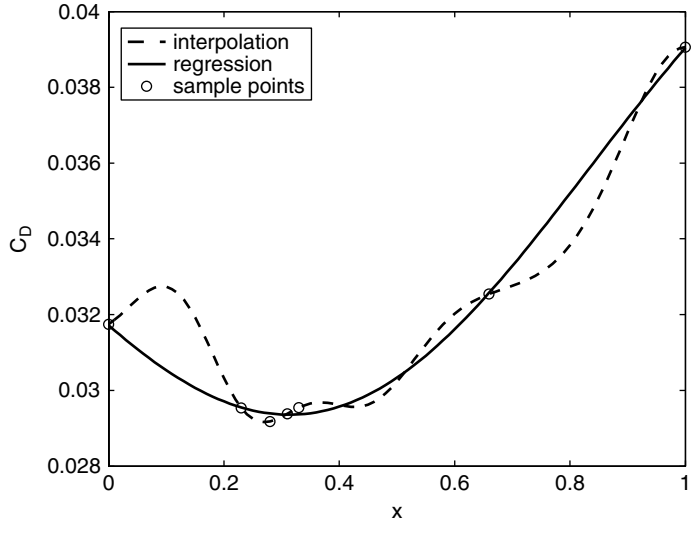



**Figure 6.3.** A regressing Kriging model significantly improves the interpolating Kriging prediction, which is reproduced from the final plot of Figure 6.2.

The regression constant $\lambda$ is determined using maximum likelihood estimation in a similar manner as the other model parameters (see Section 2.4.1). Indeed, the only deviation from the code in Section 2.4.1, which builds $\boldsymbol{\Psi}$ is the addition of $\lambda$ to the leading diagonal:[1]

```
Psi=Psi+Psi'+eye(n)+eye(n).* lambda;
```

Suitable upper and lower bounds for the search of an MLE for $\lambda$ are $10^{-6}$ and 1 respectively. The Kriging regression in Figure 6.3 was produced using `reglikelihood.m` and `regpredictor.m` in the same manner as described for Kriging interpolation in Section 2.4.

## 6.2 Searching the Regression Model

In order to search the regressing Kriging model using $E[I(\mathbf{x})]$ we require an estimate of the error in the model. By following the same derivation as for the interpolating Kriging model, but including the regression constant $\lambda$, we obtain the following expression:

$$\widehat{s}_r^{\,2}(\mathbf{x}) = \widehat{\sigma}_r^2 \left[ 1 + \lambda - \boldsymbol{\psi}^{\mathrm{T}} (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \boldsymbol{\psi} + \frac{1 - \mathbf{1}^{\mathrm{T}} (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \boldsymbol{\psi}}{\mathbf{1}^{\mathrm{T}} (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} \mathbf{1}} \right], \tag{6.3}$$

where

$$\widehat{\sigma}_r^2 = \frac{(\mathbf{y} - \mathbf{1}\widehat{\mu}_r)^{\mathrm{T}} (\boldsymbol{\Psi} + \lambda \mathbf{I})^{-1} (\mathbf{y} - \mathbf{1}\widehat{\mu}_r)}{n}. \tag{6.4}$$

(a full derivation can be found in Hoyle, 2006). Equation (6.3) *includes* the error associated with the noise in the data. This means that there is a predicted error at the sample points (note that Equation (6.3) does not reduce to zero when $\mathbf{x} \in \mathbf{X}$). Figure 6.4 shows the estimated error, found using Equation (6.3), in the regression prediction shown in Figure 6.3. With nonzero error in all areas there is also an expectation of improvement in all areas, and so there is the possibility of re-sampling during a $\max\{E[I(\mathbf{x})]\}$ search. In a situation where a repeated experiment may yield a new result, i.e. a stochastic physical experiment, this is a desirable characteristic of a search. The possibility of re-sampling means that the search cannot be guaranteed to find the global optimum, but this is expected when the data contain errors.

When repeated experiments yield identical results, i.e. they are deterministic, re-sampling is not only counter-intuitive but would lead to a stalled search process. Figure 6.5 shows what happens if we employ a $\max\{E[I(\mathbf{x})]\}$ infill strategy using regressing Kriging. In the first plot the initial regressing prediction passes close to, but not through, four sample points, thus leading to a predicted error at these points. A high expected improvement is seen at the best sample point due to the low function value and predicted error at this location. The maximum of the expected improvement is in fact adjacent to the sample

[1] In fact we were already employing a very small amount of regression (using `eye(n).*eps`) to help prevent ill-conditioning of $\boldsymbol{\Psi}$ during the likelihood maximization.

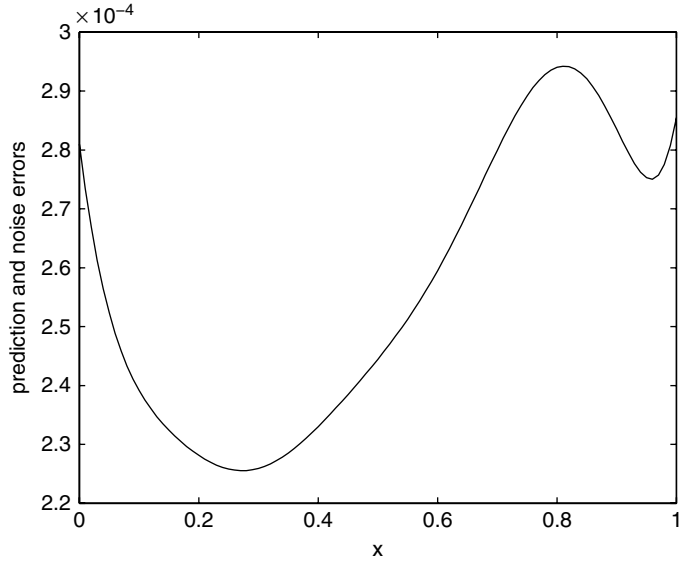



**Figure 6.4.** The estimated error in the regressing Kriging model *including* the errors due to the 'noise' in the data.

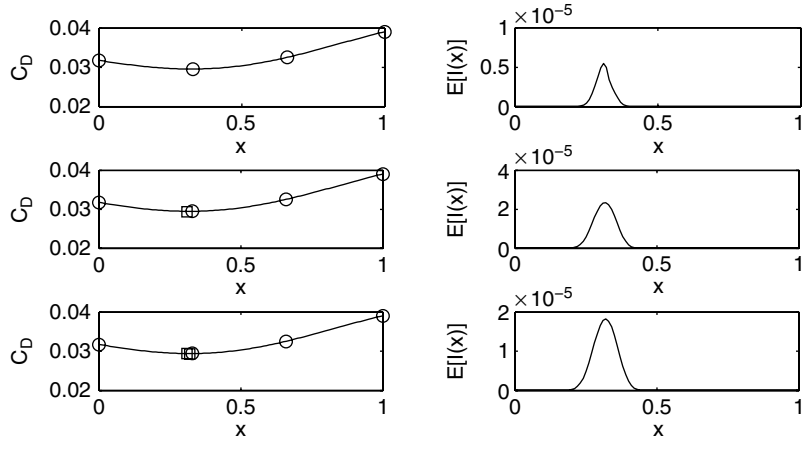



**Figure 6.5.** Updates using maximum $E[I(\mathbf{x})]$ of a Kriging regression.

point and an infill point is applied here, the model parameters are re-optimized with the new point included and a new prediction is made. The process is repeated for a second infill point, but after this stage the expected improvement is at a maximum at the location of the first update point. The optimization is now stalled and cannot progress towards the optimum. Even if the maximum of the expected improvement does not occur at a sample point, it is seen from the first two updates that, prior to stalling, the optimization progresses very slowly

when using this method of updating. The expected improvement is not diminishing since there is no change in the function, nor in the error, as all points are closely packed together. The incorrect approximation of the error also means that the sample points will not be dense and so global convergence cannot be guaranteed.

### 6.2.1 Re-Interpolation

To solve this problem we re-define error to mean uncertainty in the result and therefore eliminate the errors due to 'noise' from our model. This is achieved by basing the estimated error on an interpolation of points *predicted* by the regression model at the sample locations.[2] The error in the model *excluding* the error due to noise is found as follows. First we find values for the Kriging regression at the sample locations. The column vector of these values is given by

$$\widehat{\mathbf{y}}_r = \mathbf{1}\widehat{\mu} + \boldsymbol{\Psi}(\boldsymbol{\Psi} + \lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu}). \tag{6.5}$$

This can now be substituted into the expression for the interpolating Kriging predictor (Equation (2.40)), which in turn is substituted into the variance MLE (Equation (2.31)) to give

$$\widehat{\sigma}_{ri}^2 = \frac{(\mathbf{y} - \mathbf{1}\widehat{\mu})^{\mathrm{T}}(\boldsymbol{\Psi} + \lambda\mathbf{I})^{-1}\boldsymbol{\Psi}(\boldsymbol{\Psi} + \lambda\mathbf{I})^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu})}{n}. \tag{6.6}$$

This expression for the variance is then used in the interpolating Kriging error estimate to give a *re-interpolation* error estimate:

$$\widehat{s}_{ri}^{\,2}(\mathbf{x}) = \widehat{\sigma}_{ri}^2\left[1 - \boldsymbol{\psi}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi} + \frac{1 - \mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\boldsymbol{\psi}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}^{-1}\mathbf{1}}\right] \tag{6.7}$$

The model parameters remain unchanged from the original regression model, with no need to re-tune. The estimated error in the regression prediction shown in Figure 6.3 from Equation (6.7) is shown in Figure 6.6. Note how the error now diminishes to zero at the sample points, meaning that $\max\{P[I(\mathbf{x})]\}$ and $\max\{E[I(\mathbf{x})]\}$ infill criteria will regain their global convergence properties when using this method of error estimation. The function `reintpredictor.m` implements this error formulation and can be used in the same way as `regpredictor.m`.

A $\max\{E[I(\mathbf{x})]\}$ infill strategy based on this method of re-interpolation, and executed using the *MATLAB* code below, is shown in Figure 6.7. The initial prediction is identical to that in Figure 6.5, but the expected improvement is based on the same prediction interpolating the points shown as crosses. The expected improvement diminishes to zero at all sample points and the regression model produces a smooth prediction of the function, guiding the search towards the optimum, despite the 'noise' in the data. Note that, contrary to the interpolating update example in Figure 6.3, the expected improvement diminishes steadily throughout the

[2] There can be no certainty that the noise filtering offered by the addition of the $\lambda$ regression constant removes errors solely due to noise. There may be some smoothing of trends in the data, though this will be minimal as the sampling density increases during a search/update process.

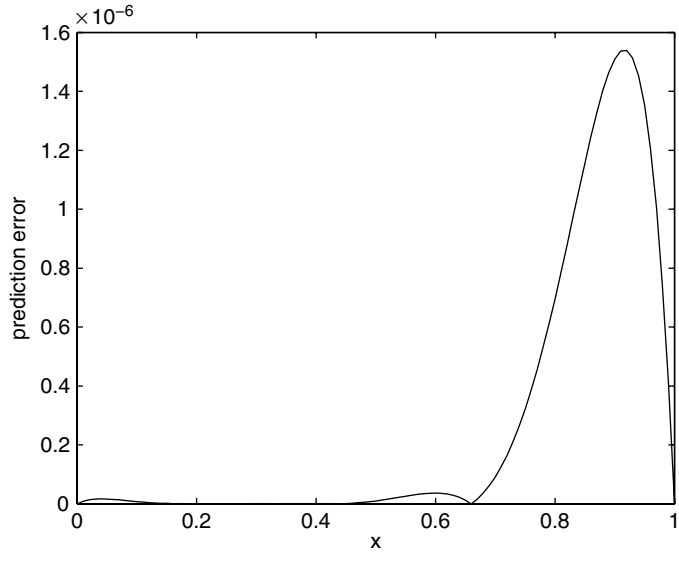



**Figure 6.6.** The estimated error in the regressing Kriging model *excluding* the errors due to the 'noise' in the data.

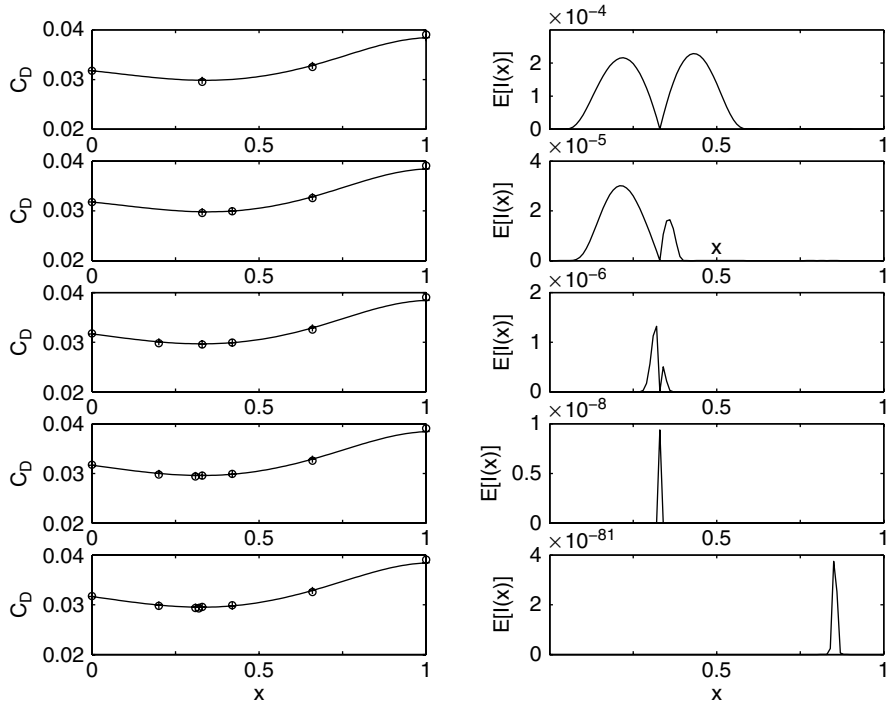



**Figure 6.7.** Updates using maximum $E[I(\mathbf{x})]$ of a Kriging re-interpolation.

optimization process (note that the scale of $E[I(\mathbf{x})]$ changes from plot to plot). Indeed, when using re-interpolation to remove the errors due to noise, the estimated error can diminish to a level where problems with floating point underflow occur when calculating $E[I(\mathbf{x})]$ (see the mathematical note at the end of this section).

```
% Create sampling plan
n=4;
k=1;
ModelInfo.X=[0 0.33 0.66 1]';

% Calculate observed data
for i=1:n
    ModelInfo.y(i,1)=aerofoilcd(ModelInfo.X(i));
end

% Start iterating infill points
for I=1:4
    % Tune regressing Kriging model
    [Params,MaxLikelihood]=...
    ga(@reglikelihood,2,[],[],[],[],[-3 -6],[2 0]);

    % Extract model parameters
    ModelInfo.Theta(1:k)=Params(1:k);
    ModelInfo.Lambda=Params(end);

    % Put Choleski factorization of Psi
    % into ModelInfo
    [NegLnLike,ModelInfo.Psi,ModelInfo.U]=...
    reglikelihood([ModelInfo.Theta ModelInfo.Lambda]);

    % Find location which maximizes EI of re-interpolation
    ModelInfo.Option='NegLogExpImp';
    [OptVar,OptEI]=ga(@reintpredictor,1,[],[],[],[],0,1);

    % Add infill point
    ModelInfo.X(end+1)=OptVar;
    ModelInfo.y(end+1)=aerofoilcd(ModelInfo.X(end));
end
```

**Mathematical Note: Avoiding Floating Point Underflow When Calculating $E[I(\mathbf{x})]$**

The expected improvement (Equation (3.7)) is typically calculated as

$$(\min\{\mathbf{y}\}-\widehat{y}(\mathbf{x}))\left[\frac{1}{2}+\frac{1}{2}\mathrm{erf}\left(\frac{\min\{\mathbf{y}\}-\widehat{y}(\mathbf{x})}{\sqrt{2}\widehat{s}(\mathbf{x})}\right)\right]+\frac{\widehat{s}(\mathbf{x})}{\sqrt{2\pi}}\exp\left[-\frac{(\min\{\mathbf{y}\}-\widehat{y}(\mathbf{x}))^2}{2\widehat{s}^2(\mathbf{x})}\right] \qquad (6.8)$$

(*continued*)

When using re-interpolation $\widehat{s}(\mathbf{x})$ is typically small and so $\text{erf}(.) \to -1$ and $\exp(.) \to 0$. This often leads to floating point underflow and $E[I(\mathbf{x})] = 0$. Using the substitution $a = (\min\{\mathbf{y}\} - \widehat{y}(\mathbf{x}))/\sqrt{2\widehat{s}}(\mathbf{x})$, when $a \ll -1$, $\text{erf}(a)$ can be expressed using a Maclaurin series expansion and the first term of Equation (6.8) becomes

$$(\min\{\mathbf{y}\} - \widehat{y}(\mathbf{x})) \left[ \frac{1}{2\sqrt{\pi}} \exp(-a^2) \sum_{n=0}^{\infty} \frac{(-1)^n (2n-1)!!}{2^n} a^{-(2n+1)} \right]$$

Note that $\exp(-a^2)$ appears in the second term in Equation (6.8) and so $E[I(\mathbf{x})]$ can now be expressed as

$$\left[ (\min\{\mathbf{y}\} - \widehat{y}(\mathbf{x})) \frac{1}{2\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n (2n-1)!!}{2^n} a^{-(2n+1)} + \frac{\widehat{s}(\mathbf{x})}{\sqrt{2\pi}} \right]$$
$$\times \exp\left[ -\frac{(\min\{\mathbf{y}\} - \widehat{y}(\mathbf{x}))^2}{2\widehat{s}^2(\mathbf{x})} \right] \tag{6.9}$$

We can now take natural logarithms and $\ln E[I(\mathbf{x})]$ can be searched by the optimizer without problems with floating point underflow. This note is drawn from Forrester *et al*. (2007).

### 6.2.2 Re-Interpolation With Conditional Likelihood Approaches

The inclusion of the regression parameter $\lambda$ is more problematic in conditional likelihood based infill criteria. Recall from Section 3.2 that in such approaches we force the prediction to pass through a hypothesized point and re-optimize the model parameters to see how likely that point is. Using the regressing Kriging formulation directly will result in the hypothesized point being filtered out as noise, by $\lambda$ increasing upon re-optimization of the model parameters. This makes no sense, whether the data is from a physical or a computer experiment. In the case of physical experiments the fix is simple: we can hold $\lambda$ constant at the MLE found for the observed data. This ensures that no extra regression will be used simply because the hypothesized point is unreasonable. This fix is not sufficient for computer experiments, because there is a possibility of maximizing the conditional likelihood at a previously sampled point. Thus the infill strategy may stall in similar scenarios to the $\max\{E[I(\mathbf{x})]\}$ criterion considered in the previous section.

Figure 6.8 shows the progress of a goal seeking infill strategy on the aerofoil design problem using this regressed conditional likelihood method. The *MATLAB* code used to perform the infill strategy is shown below. Only a slight change to the original goal seeking code in Section 3.2.4 is required: namely the addition of an extra model parameter for the search of `reglikelihood.m` and the use of `regcondlikelihood.m` instead of `condlikelihood.m`. The goal is $C_D = 0.0292$. Initially the MLE for $\lambda$ is very low and the conditional likelihood diminishes to almost zero at the sample points. As points begin to cluster the MLE for $\lambda$ increases and the conditional likelihood has a wide peak in the region of the optimum despite there being sample points here, leading to a stalled search after three updates. The code would actually keep running inside the **`while`** loop because the goal is never reached.

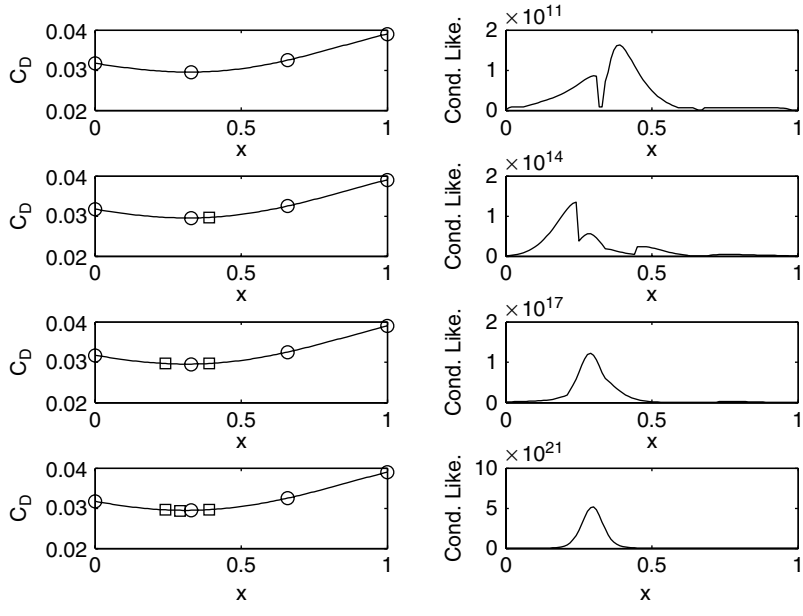



**Figure 6.8.** A goal seeking search of the aerofoil problem using regressing Kriging.

```
global ModelInfo
% Create sampling plan
n=4;
k=1;
ModelInfo.X=[0 0.33 0.66 1]';

% Calculate observed data
for i=1:n
    ModelInfo.y(i,1)=aerofoilcd(ModelInfo.X(i));
end

% Set goal
ModelInfo.Goal=0.0292;

% Iterate infill points until goal is met
while min(ModelInfo.y)>ModelInfo.Goal
    % Tune regressing Kriging model
    [Params,MaxLikelihood]=ga(@reglikelihood,2,[],[],[],[],...
    [-3 -6],[2 0]);

    % Extract model parameters
    ModelInfo.Theta(1:k)=Params(1:k);
    ModelInfo.Lambda=Params(end);

    % Put Choleski factorization of Psi
    into ModelInfo
```

(*continued*)

```
    [NegLnLike,ModelInfo.Psi,ModelInfo.U]=reglikelihood...
    ([ModelInfo.Theta ModelInfo.Lambda]);

    % Find location which maximizes likelihood of goal
    [OptVar,NegCondLike]=ga(@regcondlikelihood,2,[],[],[],[],...
    [-3 0],[3 1]);

    % Add infill point
    ModelInfo.X(end+1)=OptVar(k+1:end);
    ModelInfo.y(end+1)=aerofoilcd (ModelInfo.X(end));
end
```

To solve this problem of high conditional likelihoods at sample points we re-interpolate the observed data before finding the conditional likelihood. The function `reintcondlikelihood.m` achieves this through the addition of the following few lines of code to the original `condlikelihood.m` function:

```
...
% Replace observed data with regressed points
ModelInfo.Option='Pred';
for I=1:n
    y(i,1)=regpredictor(ModelInfo.X(i));
end
...
```

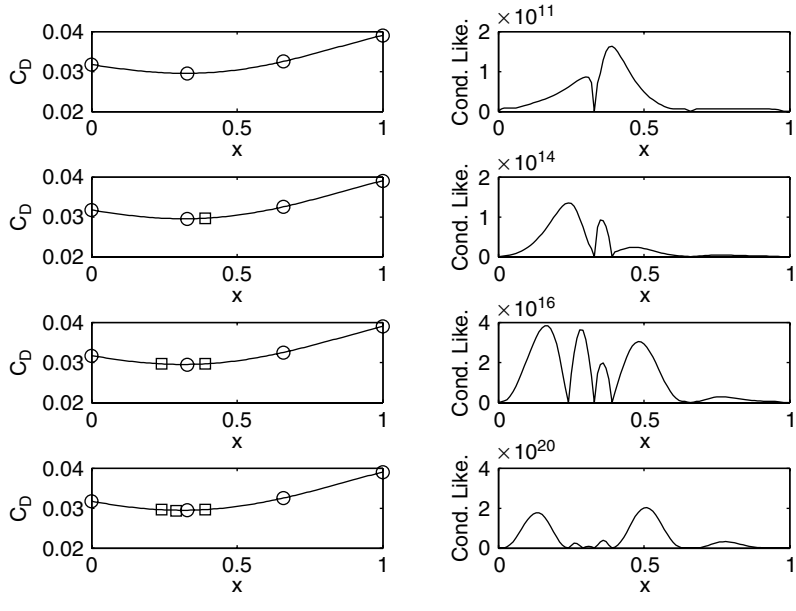



**Figure 6.9.** A goal seeking search of the aerofoil problem using re-interpolation.

The progress of an infill strategy based on this re-interpolated conditional likelihood is shown in Figure 6.9. As expected, the conditional likelihood drops to zero at all sample locations and the search is truly global.

## 6.3 A Note on Matrix Ill-Conditioning

The use of the regression constant $\lambda$ filters noise and reduces ill-conditioning of the correlation matrix. When re-interpolating the data, there are no longer problems with noise, but ill-conditioning can still be a problem, even though the data is now smooth. This is not normally a problem if data is sparse, which is usually the case when employing surrogates. As data become more dense, $\boldsymbol{\Psi}$ can become ill-conditioned and the Cholesky factorization may fail. LU decomposition will always provide an answer when used to invert $\boldsymbol{\Psi}$ but, with an ill-conditioned $\boldsymbol{\Psi}$, this answer will be unreliable and it is preferable to add a small value to the leading diagonal of $\boldsymbol{\Psi}$ (usually between `eps` and $10^{-6}$), i.e. some regression is still used. It is possible to direct the model parameter search towards parameters that produce well-conditioned matrices by applying a penalty when the matrix of the Cholesky factorization fails. For example, the following code is used in our *MATLAB* likelihood functions:

```
[U,p]=chol(Psi);
if p>0
    NegCondLnLike=1e4;
else
...
% Likelihood calculation
...
end
```

## 6.4 Summary

In this chapter we have concentrated on what we believe to be the most appropriate methods for dealing with noise in surrogate model based optimization. However, these are by no means the only solutions to the problem. A least-squares Kriging regression can be obtained directly by solving the prediction equation (2.40) using singular value decomposition (SVD) (Press *et al*., 1992), which is simple to implement in *MATLAB* by using `svd.m` in place of `chol.m` in a Kriging interpolation. However, the error estimates obtained may lead to stalled or lengthy optimization processes.

In Section 2.2 the use of polynomial regression has been discussed. Further research in the use of polynomials with 'noisy' data in an engineering design context can be found in Kim *et al*. (2001), Narducci *et al*. (1995) and Papila and Haftka (2000). Section 2.5 discussed the use of SVR. This method is still in its infancy in terms of engineering optimization and little research is available. A good starting point is Schölkopf and Smola (2002).

While we have spent some time discussing the regressing Kriging method, we have not entered into a full derivation of the regressing Kriging error estimate, since it is not required for practical implementation. The derivation and further discussion on noise in surrogate based optimization can be found in Forrester *et al*. (2006).

## References

Forrester, A. I. J., Keane, A. J. and Bressloff, N. W. (2006) Design and analysis of 'noisy' computer experiments. *American Institute of Aeronautics and Astronautics Journal*, **44**(10), 2331–2339.

Forrester, A. I. J., Sobester, A. and Keane, A. J. (2007) Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A,* **463**(2088), 3251–3269, (doi:10.1098/rspa.2007.1900).

Hoerl, A. E. and Kennard, R. W. (1970) Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, **12**(1), 55–67, February.

Hoyle, N. (2006) *Automated Multi-stage Geometry Parameterization of Internal Fluid Flow Applications*. PhD thesis, University of Southampton, Southampton.

Kim, H., Papila, M., Mason, W., Haftka, R. T., Watson, L. T. and Grossman, B. (2001) Detection and repair of poorly converged optimization runs. *American Institute of Aeronautics and Astronautics Journal*, **39**(12), 2242–2249, December.

Narducci, R., Grossman, B., Valorani, M., Dadone, A. and Haftka, R. T. (1995) Optimization methods for non-smooth or noisy objective functions in fluid design problems, in *12th AIAA Computational Fluid Dynamics Conference*, San Diego, California, AIAA-1995-1648, June.

Papila, M. and Haftka, R. T. (2000) Response surface approximations: noise, error repair, and modeling errors. *American Institute of Aeronautics and Astronautics Journal*, **33**(12), 2336–2343, December.

Press, W. H., Flannery, B. P., Teulolsky, S. A. and Vetterling, W. T. (1992) *Numerical Recipes in C*, 2nd edition, Cambridge University Press.

Schölkopf, B. and Smola, A. J. (2002) *Learning with Kernals*, MIT Press, Cambridge, Massachusetts.

Tikhonov, A. N. and Arsenin, V. Y. (1977) *Solutions of Ill-posed Problems*, Winston, Washington.

# 7

# Exploiting Gradient Information

A key benefit of a surrogate model based search is that the gradients of the objective function are not required. If gradient information *is* available, the designer may in fact choose to employ a localized gradient descent search of the objective function with no surrogate model. However, if a global optimum is sought, the gradient information can be used to enhance the accuracy of a surrogate model of the design landscape, which can then be searched using a global optimizer.

Although we are concerned mainly with the process of building and searching a surrogate model which utilizes gradients, we begin this chapter with a brief overview of how gradients of the objective function might be obtained. We then go on to show how this information (and higher derivatives) can be used to enhance the surrogate model.

## 7.1 Obtaining Gradients

### *7.1.1 Finite Differencing*

The simplest way to obtain gradient information is through finite differencing. Using a one-sided forward difference the gradient in the $l$th dimension is found by

$$\frac{\partial f(\mathbf{x})}{\partial x_l} = \frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x})}{h} + \mathcal{O}(h), \tag{7.1}$$

where $\mathbf{h}$ is a vector of zeros of length $k$ with the $l$th element set to $h$, the finite difference step. The truncation error is $\mathcal{O}(h)$. Using central differencing the $l$th gradient is found by

$$\frac{\partial f(\mathbf{x})}{\partial x_l} = \frac{f(\mathbf{x}+\mathbf{h}) - f(\mathbf{x}-\mathbf{h})}{2h} + \mathcal{O}(h^2). \tag{7.2}$$

An additional $k$ (forward differencing) or $2k$ (central differencing) calculations of the objective function are required to find all $k$ derivatives. The choice of the step size $h$ represents something of a dilemma – a small step reduces the truncation error but increases errors due to subtractive cancellation. This is the only method for calculating derivatives if we do not have access to the objective function source code.

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

### 7.1.2 Complex Step Approximation

With access to the source code, derivatives may be found more accurately using a complex step approximation (Squire and Trapp, 1998). By defining the vector $\mathbf{x}$ as a complex variable, $\mathbf{x} + i\mathbf{h}$,

$$\frac{\partial f(\mathbf{x})}{\partial x_l} = \frac{\Im[f(\mathbf{x} + i\mathbf{h})]}{h} + \mathcal{O}(h^2), \tag{7.3}$$

where $\Im[.]$ is the imaginary part of the complex objective function. Computing gradients this way is more expensive than finite differencing, since complex arithmetic is more time consuming. However, because there is no subtractive cancellation, $h$ can be very small and so the truncation error which is $\mathcal{O}(h^2)$ can be all but eliminated. Most programming languages can perform complex arithmetic and so implementation is quite straightforward. Some operators such as `abs()` (which is not analytic) must be overloaded to permit the use of complex numbers (see Martins *et al*., 2003, for more details on implementation).

The additional evaluations required to compute gradients using finite difference and complex step approximation methods would likely be better spent on producing a more space-filling sampling plan (see Chapter 1). Gradient information is only useful if it can be obtained more cheaply. The most promising ways of obtaining gradient information are through Algorithmic Differentiation (AD) (also known as Automatic Differentiation, though the term 'automatic' instills false hope, since manual intervention is required in most cases), which requires access to the objective function source code, and adjoint codes, which require the creation of a whole new source code.

### 7.1.3 Adjoint Methods and Algorithmic Differentiation

A detailed description of the adjoint method and AD is beyond the scope of this book, but we will highlight the strengths and weaknesses of each method. Further information on the adjoint approach can be found in Giles and Pierce (2000) and for more details on AD see Griewank (2000).

The adjoint approach comes in two flavours: the discrete approach, where governing equations are discretized before forming the adjoint, and the continuous approach, where the adjoint formulation deals directly with the governing equations and is then discretized. The benefits of the discrete approach are that the exact gradient of the discrete objective function is obtained and the creation of the adjoint problem is more straightforward. Using the continuous approach, the physical significance of the adjoint variables and the role of the adjoint boundary conditions is clearer and the adjoint program is simpler and requires less memory (Giles and Pierce, 2000).

AD can be performed in forward mode or reverse mode. In forward mode the computer program to be differentiated is decomposed into elementary operations whose derivatives are accumulated according to the chain rule. If all derivatives are required, this process leads to a $k$-fold increase in computational cost compared to evaluating $f(\mathbf{x})$ alone. Although equivalent in computational cost to one-sided finite differencing, forward mode AD is preferable, albeit more difficult to implement, since more accurate derivatives are obtained. Forward mode is equivalent to a complex step approximation, though it is more attractive in terms of computational cost.

A forward mode AD plug-in for *MATLAB* called *MAD* has been developed by Forth (2005). The authors have used this package successfully, finding its implementation particularly straightforward.

In reverse mode the function value is calculated first before a backward pass to compute the derivatives. The derivative of the final output is computed with respect to an intermediate quantity, which can be thought of as an adjoint quantity. The reverse mode never requires more than five times the cost of evaluating the underlying function (Griewank, 1989) and in practice much higher computational efficiency can be obtained. As such the reverse mode is preferred over the forward mode for all but very low dimensional problems (when the benefit of computing gradients is doubtful anyway). A drawback with this method is that memory requirements may be prohibitive for very high dimensional problems.

Other than at the beginning of the development of a new objective function code, the designer does not have a choice between using the adjoint method or AD. In general the adjoint approach is more efficient in terms of memory requirements, but requires more programming effort than applying AD to the objective function code. AD provides exact results for the derivatives and is arguably more suited to a continuously developing code. The code can be differentiated by an AD tool after each modification, whereas development of an adjoint code requires more laborious hand coding, which is more prone to errors.

Howsoever the derivatives of the objective function have been found, the methods by which we incorporate them into the surrogate model are the same. This is the subject of the remainder of this chapter.

## 7.2 Gradient-enhanced Modelling

The basis function methods in Sections 2.3 and 2.4 are built from the sum of a number of basis functions centred around the sample data. The height of these functions determines the value of the prediction at the sample points (usually such that the model interpolates the data) and the width determines the rate at which the function moves away from this value. If gradient information is available at the sample locations, we can incorporate this into the model, using a second set of basis functions, also centred around the sample data. The observed data is now a $(k+1)n$ column vector:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \\ \dfrac{\partial y^{(1)}}{\partial x_1^{(1)}} \\ \vdots \\ \dfrac{\partial y^{(n)}}{\partial x_1^{(n)}} \\ \vdots \\ \dfrac{\partial y^{(1)}}{\partial x_k^{(1)}} \\ \vdots \\ \dfrac{\partial y^{(n)}}{\partial x_k^{(n)}} \end{pmatrix}. \tag{7.4}$$

The additional basis functions determine the gradient of the prediction at the sample points and the rate at which the function moves away from this gradient. We will concentrate on the use of Gaussian basis functions, that is Kriging with the exponent parameter $p$ set at 2.

The form of the basis function used to incorporate the gradient information is simply the derivative of the first $n$ Gaussian basis functions with respect to the design variables:

$$\frac{\partial \psi^{(i)}}{\partial x_l^{(i)}} = \frac{\partial \exp\left(-\sum_{l=1}^{k} \theta_l (x_l^{(i)} - x_l)^2\right)}{\partial x_l^{(i)}} = -2\theta_l (x_l^{(i)} - x_l)\psi^{(i)}. \tag{7.5}$$

Note that it is necessary to use $(x_l^{(i)} - x_l)$ instead of $|x_l^{(i)} - x_l|$ and set $p = 2$ in order that the basis function can be differentiated through $x_l^{(i)} - x_l = 0$. Figure 7.1 shows how this function behaves as $x_l^{(i)} - x_l$ varies. Recall from Section 2.4 how the Kriging basis function had the intuitive property that as $x_l^{(i)} - x_l \to 0$, $\psi \to 1$, i.e. the value at $x_l$ approaches that at $x_l^{(i)}$. In Figure 7.1 it is seen that $\partial\psi/\partial x \to 0$ as $x_l^{(i)} - x_l \to 0$. Here we are looking at how the prediction will be distorted from the model produced by the first $n$ basis functions and so it is also intuitive that no distortion should be applied at a sampled point (we can learn no more about the value at this point than the sample data objective values). As we move away from the point, the function pulls the prediction up or down. As before, the $\theta_l$ parameter determines the activity of the function: a higher $\theta_l$ leads to a small region of distortion, with the value of $\partial\psi/\partial x_l$ quickly returning to zero, while a low $\partial\psi/\partial x_l$ means that a larger area is influenced by the value of the gradient in the $l$th direction at $x^{(i)}$.

The $-2\theta_l(x_l^{(i)} - x_l)$ multiplier means that the maximum value of the differentiated correlation is affected by the value of $\theta_l$. Referring to Figure 7.1, the higher value and steeper

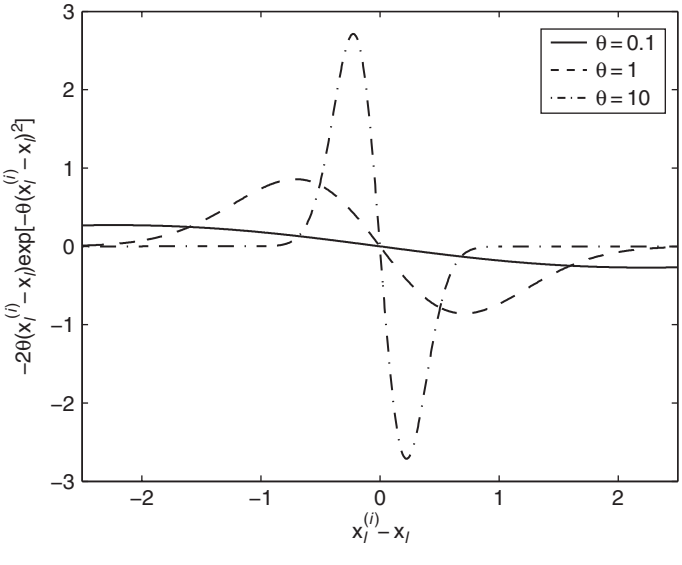



**Figure 7.1.** Differentiated correlations for varying $\boldsymbol{\theta}$.

slope of the $\theta_l = 10$ correlation does not mean that it will lead to a prediction with a steeper gradient through $x^{(i)}$ than lower $\theta_l$'s. The height of the basis functions is dictated by the constants $\boldsymbol{\Psi}(\mathbf{y} - \mathbf{1}\mu)$ in the Kriging predictor (Equation (2.40)). Here we are purely concerned with the effect of $\theta_l$ on the width of the correlations. The $-2\theta_l(x_l^{(i)} - x_l)$ multiplier also means that the differentiated correlation is not symmetric about $x^{(i)}$. This is, of course, necessary if a nonzero gradient is to be applied at $x^{(i)}$: the predictor is distorted up on one side and down on the other, with the direction and magnitude dictated by the sign and value of $\dot{\boldsymbol{\Psi}}(\mathbf{y} - \mathbf{1}\mu)$.

But how do we construct $\dot{\boldsymbol{\Psi}}$? In gradient-enhanced Kriging the correlation matrix $\boldsymbol{\Psi}$ must now include the correlation between the data and the gradients and the gradients and themselves as well as the correlations between the data, and will be denoted by the $(k+1)n \times (k+1)n$ matrix $\dot{\boldsymbol{\Psi}}$. The matrix, for a one dimensional problem $(k = 1)$ is constructed as follows:

$$\dot{\boldsymbol{\Psi}} = \begin{pmatrix} \boldsymbol{\Psi} & \dfrac{\partial \boldsymbol{\Psi}}{\partial x^{(i)}} \\ \dfrac{\partial \boldsymbol{\Psi}}{\partial x^{(j)}} & \dfrac{\partial^2 \boldsymbol{\Psi}}{\partial x^{(i)} \partial x^{(j)}} \end{pmatrix}. \tag{7.6}$$

The superscripts in Equation (7.6) refer to which way round the subtraction is being performed when calculating the distance in the correlation $\psi$. This is not important when we are squaring the result but, after differentiating, sign changes will appear depending upon whether we differentiate with respect to $x^{(i)}$ or $x^{(j)}$. Using the product and chain rule, the following derivatives are obtained:

$$\frac{\partial \boldsymbol{\Psi}^{(i,j)}}{\partial x^{(i)}} = -2\theta(x^{(i)} - x^{(j)})\boldsymbol{\Psi}^{(i,j)}, \tag{7.7}$$

$$\frac{\partial \boldsymbol{\Psi}^{(i,j)}}{\partial x^{(j)}} = 2\theta(x^{(i)} - x^{(j)})\boldsymbol{\Psi}^{(i,j)}, \tag{7.8}$$

$$\frac{\partial^2 \boldsymbol{\Psi}^{(i,j)}}{\partial x^{(i)} \partial x^{(j)}} = [2\theta - 4\theta^2(x^{(i)} - x^{(j)})^2]\boldsymbol{\Psi}^{(i,j)}. \tag{7.9}$$

In *MATLAB*, given that the $\boldsymbol{\Psi}$ matrix has already been constructed (see Section 2.4), the correlation matrix $\dot{\boldsymbol{\Psi}}$ is built as follows:

```
% Pre-allocate memory
dPsidX=zeros(n,n); d2PsidX2=zeros(n,n);

% Build half matrices
for i=1:n
    for j=i+1:n
        dPsidX(i,j)=2*theta*(X(i)-X(j))*Psi(i,j);

        d2PsidX2(i,j)=(2*theta...
        -4*theta^2*(X(i)-X(j))^2*Psi(i,j);
    end
end
```

(*continued*)

```
% Add upper and lower halves and diagonal
PsiDot=[Psi (dPsidX-dPsidX'); (dPsidX+dPsidX')
(d2PsidX2+d2PsidX2'+eye(n) .*(2*theta))];
```

For problems with more dimensions ($k > 1$), the correlation matrix becomes rather more unwieldy:

$$\dot{\boldsymbol{\Psi}} = \begin{pmatrix} \boldsymbol{\Psi} & \frac{\partial \boldsymbol{\Psi}}{\partial x_1^{(i)}} & \frac{\partial \boldsymbol{\Psi}}{\partial x_2^{(i)}} & \cdots & \frac{\partial \boldsymbol{\Psi}}{\partial x_k^{(i)}} \\ \frac{\partial \boldsymbol{\Psi}}{\partial x_1^{(j)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(i)} \partial x_1^{(j)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(i)} \partial x_2^{(j)}} & \cdots & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(i)} \partial x_k^{(j)}} \\ \frac{\partial \boldsymbol{\Psi}}{\partial x_2^{(j)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(j)} \partial x_2^{(i)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_2^{(i)} \partial x_2^{(j)}} & \cdots & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_2^{(i)} \partial x_k^{(j)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{\Psi}}{\partial x_k^{(j)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(j)} \partial x_k^{(i)}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_2^{(j)} \partial x_k^{(i)}} & \cdots & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_k^{(i)} \partial x_k^{(j)}} \end{pmatrix}. \tag{7.10}$$

The additional correlation between the derivatives of different dimensions of the problem is

$$\frac{\partial^2 \boldsymbol{\Psi}^{(i,j)}}{\partial x_l^{(i)} \partial x_m^{(i)}} = -4\theta_l \theta_m (x_l^{(i)} - x_l^{(j)})(x_m^{(i)} - x_m^{(j)}) \boldsymbol{\Psi}^{(i,j)}. \tag{7.11}$$

Noting that $\dot{\boldsymbol{\Psi}}$ is built from three forms of correlation, $\partial \boldsymbol{\Psi}/\partial x_l$, $\partial^2 \boldsymbol{\Psi}/\partial x_l \partial x_m$ and $\partial^2 \boldsymbol{\Psi}/\partial x_l^2$, an efficient method for constructing $\dot{\boldsymbol{\Psi}}$ in *MATLAB* is as follows:

```
% Pre-allocate memory
PsiDot=zeros ((k+1)*n, (k+1)*n)

% Build upper half of PsiDot
for l=1:k
  for m=l:k
    if l==1

    % Build upper half of dPsidX
      for i=1:n
        for j=i+1:n
          PsiDot (i,m* n+j)=2* theta(m)*(X(i,m)-X(j, m))* Psi(i,j);
        end
      end

      % Add upper and lower halves
       PsiDot (1:n,m * n+1:(m+1)* n)=PsiDot (1:n,m * n+1:(m+1)* n)...
                                    - PsiDot (1:n,m* n+1:(m+1)* n)';
    end
```

(*continued*)

```
    if m==1
    % Build upper half of d2PsidX^2
    for i=1:n
      for j=i+1:n
        PsiDot (l*n+i, m*n+j)=...
        (2*theta(l)-4*theta(l)^2*(X(i,l)-X(j,l))^2)*Psi(i,j);
      end
    end

    % Add half diagonal
     PsiDot (l*n+1:(l+1)*n,m*n+1:(m+1)*n)= ...
     PsiDot (l*n+1:(l+1)*n, m*n+1:(m+1)*n)+eye(n).*theta(l);
    else

    % Build upper half of d2PsidXdX
      for i=1:n
        for j=i+1:n
          PsiDot (l*n+i, m*n+j)=...
         -4*theta(l)*theta(m)*(X(i,l)-X(j,l))*(X(i,m)-X(j,m))...
         *Psi(i,j);
        end
      end

      % Add upper and lower halves
       PsiDot (l*n+1:(l+1)*n,m*n+1:(m+1)*n)=...
       PsiDot(l*n+1:(l+1)*n, m*n+1:(m+1)*n)+PsiDot(l*n+1:(l+1)...
       *n,m*n+1:(m+1)*n)';
    end
  end
end

% Add upper and lower halves to Psi
PsiDot=[Psi zeros(n,k*n); zeros(k*n,(k+1)*n)]+PsiDot+PsiDot';
```

The $\boldsymbol{\theta}$ parameter is found by maximizing the concentrated ln-likelihood in the same manner as for Kriging (see Section 2.4). Other than the above correlations, the only difference in the construction of the gradient-enhanced model is that $\mathbf{1}$ is now a $(k+1)n \times 1$ column vector of $n$ ones followed by $nk$ zeros. The gradient-enhanced predictor is

$$\widehat{y}(\mathbf{x}) = \widehat{\mu} + \dot{\boldsymbol{\psi}}^{\mathrm{T}} \dot{\boldsymbol{\Psi}}^{-1} (\mathbf{y} - \mathbf{1}\widehat{\mu}), \tag{7.12}$$

where

$$\dot{\boldsymbol{\psi}} = \left( \boldsymbol{\psi}, \frac{\partial \boldsymbol{\psi}}{\partial x_1}, \ldots, \frac{\partial \boldsymbol{\psi}}{\partial x_k} \right)^{\mathrm{T}}. \tag{7.13}$$

Figure 7.2 shows a contour plot of the Branin function along with a gradient-enhanced Kriging prediction based on nine sample points. True gradients and gradients calculated using a finite difference of the gradient-enhanced Kriging prediction are also shown. The agreement between the functions and gradients is remarkable for this function, but the method is unlikely to perform quite so well on true engineering functions.

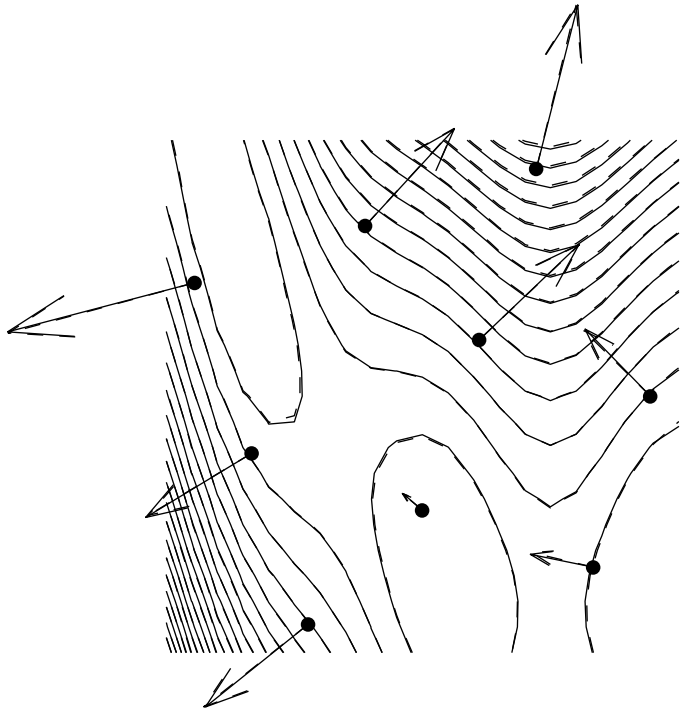


**Figure 7.2.** Contours of the Branin function (solid) and a gradient-enhanced prediction (dashed) based on nine points (dots). True gradients (solid arrows) and gradients calculated using a finite difference of the gradient-enhanced Kriging prediction (dashed arrows) are also shown. Note that the true function and the prediction are so close that the solid contours and arrows almost completely obscure their dashed counterparts.

## 7.3 Hessian-enhanced Modelling

We can take the use of gradients to the next step and include second derivatives in an Hessian-enhanced Kriging model. The basis function used to incorporate the second derivative information is the second derivative of the first $n$ Gaussian basis functions with respect to the design variables:

$$\frac{\partial^2 \psi^{(i)}}{\partial x_l^{(i)2}} = \frac{\partial^2 \exp\left[-\sum_{l=1}^{k} \theta_l (x_l^{(i)} - x_l)^2\right]}{\partial x_l^{(i)2}} = \left[-2\theta_l + 4\theta_l^2 (x_l^{(i)} - x_l)^2\right] \psi^{(i)}. \tag{7.14}$$

Figure 7.3 shows how the twice differentiated basis function behaves for varying $\theta$. Rather counter-intuitively, the function does not reduce to zero as $x^{(i)} - x \to 0$. At the sample points the weighting applied to $\psi$ and $\ddot{\psi}$ must be such that the model interpolates the data. More intuitively, as $\theta$ increases the basis function has high curvature in the immediate vicinity of $x^{(i)}$ and quickly returns to zero. As with $\dot{\psi}$, the maximum value of the basis function is affected by $\theta$ and we must bear in mind that the height of the function will be modified by the weights $\ddot{\boldsymbol{\Psi}}(\mathbf{y} - \mathbf{1}\mu)$.

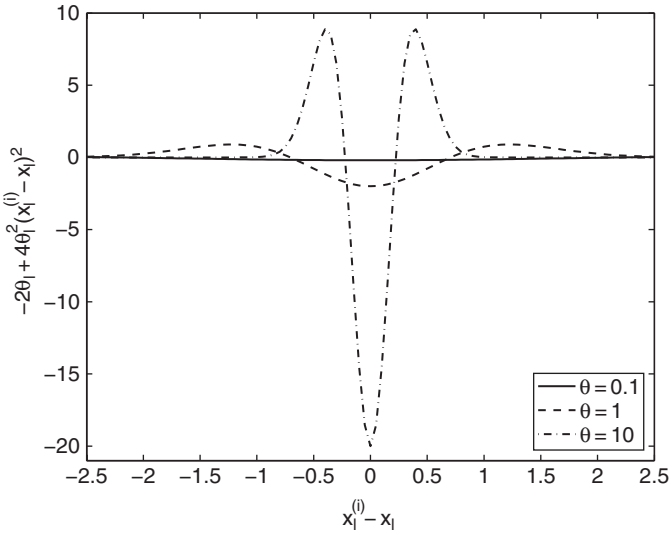



**Figure 7.3.** Twice differentiated correlations for varying $\boldsymbol{\theta}$.

The correlations between the data and its first and second derivatives are described by a $(2k+1)n \times (2k+1)n$ matrix:

$$\ddot{\boldsymbol{\Psi}} = \begin{pmatrix} \dot{\boldsymbol{\Psi}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(i)2}} & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_2^{(i)2}} & \cdots & \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_k^{(i)2}} \\ \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_1^{(j)2}} & \frac{\partial^4 \boldsymbol{\Psi}}{\partial x_1^{(i)2} \partial x_1^{(j)2}} & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_1^{(i)2} \partial x_2^{(j)}} & \cdots & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_1^{(i)2} \partial x_k^{(j)}} \\ \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_2^{(j)2}} & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_1^{(j)} \partial x_2^{(i)2}} & \frac{\partial^4 \boldsymbol{\Psi}}{\partial x_2^{(i)2} \partial x_2^{(j)2}} & \cdots & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_2^{(i)2} \partial x_k^{(j)}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \boldsymbol{\Psi}}{\partial x_k^{(j)2}} & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_1^{(j)} \partial x_k^{(i)2}} & \frac{\partial^3 \boldsymbol{\Psi}}{\partial x_2^{(j)} \partial x_k^{(i)2}} & \cdots & \frac{\partial^4 \boldsymbol{\Psi}}{\partial x_k^{(i)2} \partial x_k^{(j)2}} \end{pmatrix}. \tag{7.15}$$

The model parameters are tuned and predictions made in an equivalent manner to the gradient-enhanced modelling in Section 7.2.

Figure 7.4 shows three predictions of our one-variable test function: Kriging, gradient-enhanced Kriging and Hessian-enhanced Kriging. Given the small initial sample, the maximum likelihood search yields a poor choice of $\theta = 100$ (our upper bound) for the Kriging model, producing a mean fit to the data with deviations to interpolate the sample data. It is clear in this simple example that the Kriging model is poor and a lower $\theta$ would be more appropriate. However, in a high-dimensional problem the shortcomings of the Kriging model would not be so obvious. The gradient-enhanced model yields a more appropriate $\theta = 15.6$ and this prediction is closer to the true function. The inclusion of second derivatives results in an increased $\theta = 40.5$, due to the high second derivative at $x = 1$ indicating a more active

function. This prediction offers a significant improvement and has its minimum within the basin of the deceptive global optimum.

The nine weighted basis functions and mean used to build the prediction in Figure 7.4 are shown in Figure 7.5. It is clear from this figure how each type of basis function affects

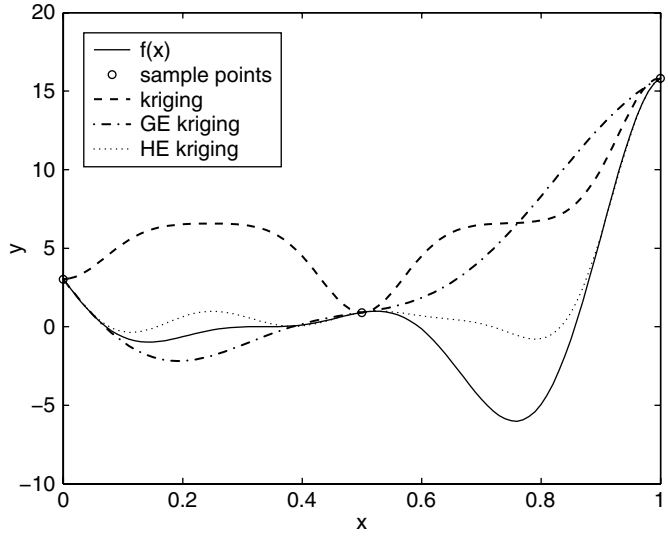



**Figure 7.4.** Kriging, gradient-enhanced Kriging, and Hessian-enhanced Kriging predictions of $f(x) = (6x-2)^2 \sin(12x-4)$ using three sample points.

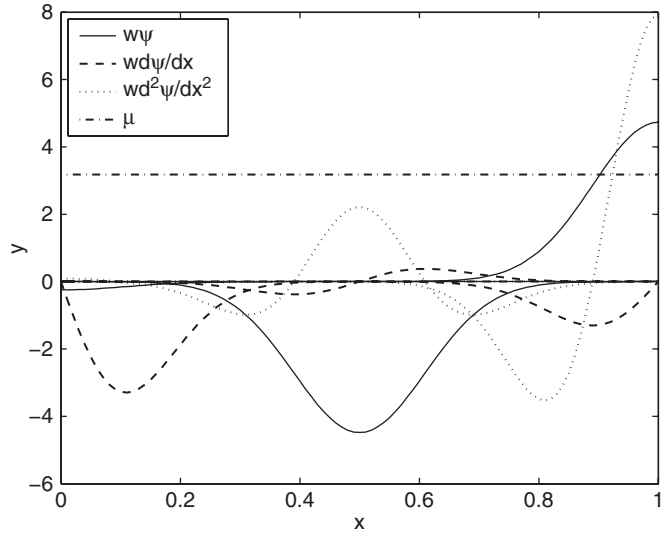



**Figure 7.5.** The nine basis functions used to construct the Hessian-enhanced Kriging prediction, multiplied by their weights, $\mathbf{w} = \ddot{\mathbf{\Psi}}^{-1}(\mathbf{y} - \mathbf{1}\mu)$. These are added to the constant $\mu$ to produce the prediction in Figure 7.4.

the prediction. The first three ($\psi$) are simple deviations from the mean and the second three ($\dot{\psi}$) clearly match the gradient at the sample points. Of the final three bases ($\ddot{\psi}$), the first has little effect (the gradient is near constant at this point), the second works against $\psi$ to flatten the function, while the third adds to the curvature, resulting in the steep curve into the global minimum.

## 7.4 Summary

The use of derivative information adds considerable complexity to the model and the increased size of the correlation matrix leads to lengthier parameter estimation, but there is clearly the possibility of building more accurate predictions. Schemes to reduce model parameter estimation times for large correlation matrices are always the target of research effort, though a panacea is yet to reveal itself!

Second derivatives are not often available to the designer but, with the increasing use of algorithmic differentiation tools, models which can take advantage of this information may soon provide significant speed-ups compared to using additional objective function values, particularly in very high dimensional problems.

## References

Forth, S. A. (2005) An efficient overloaded implementation of forward mode automatic differentiation in MATLAB. *ACM Transactions on Mathematical Software*, **3**(2), 195–222, June.

Giles, M. B. and Pierce, N. A. (2000) An introduction to the adjoint approach to design. *Flow, Turbulance and Combustion*, **65**, 393–2000.

Griewank, A. (1989) Chapter on Automatic differentiation, in *Mathematical Programming: Recent Developments and Applications*, pp. 83–108, Kluwer Academic Publishers, Dordrecht, The Netherlands.

Griewank, A. (2000) *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Frontiers in Applied Mathematics. SIAM, Philadelphia, Pennsylvania.

Martins, J. R. R. A., Sturdza, P. and Alonso, J. J. (2003) The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, **29**, 245–262.

Squire, W. and Trapp, G. (1998) Using complex variables to estimate derivatives of real functions. *SIAM Review*, **40**, 110–112.

# 8

# Multi-fidelity Analysis

Often situations arise when we have more information about our objective function than the simple vector of function values that we have considered elsewhere in this book. It may be, for example, that as well as using finite element analysis or computational fluid dynamics, a quick calculation can be made using empirical equations, more simple beam theory or panel methods. A greater quantity of this cheap data may be coupled with a small amount of expensive data to enhance the accuracy of a surrogate of the expensive function. To make use of the cheap data, we must formulate some form of correction process which models the differences between the cheap and expensive function(s).

Combining multiple sets of data naturally leads to a complex notation and we will try to simplify this by limiting ourselves to two data sets. Our most accurate expensive data has values $\mathbf{y}_e$ at points $\mathbf{X}_e$ and the less accurate cheap data has values $\mathbf{y}_c$ at points $\mathbf{X}_c$. The formulation of a correction process is simplified if the expensive function sample locations coincide with a subset of the cheap sample locations ($\mathbf{X}_e \subset \mathbf{X}_c$). The correction process will usually take the form:

$$\mathbf{y}_e = Z_\rho \mathbf{y}_c + Z_d. \tag{8.1}$$

With $Z_d = 0$, $Z_\rho$ can take the form of any approximation model fitted to $\mathbf{y}_e / \mathbf{y}_c(\mathbf{X}_e)$. Likewise, with $Z_\rho = 1$, $Z_d$ can take the form of an approximation fitted to $\mathbf{y}_e - \mathbf{y}_c(\mathbf{X}_e)$. These processes are then used to correct $\mathbf{y}_c$ when making predictions of the expensive function $f_e$. If the correction process is simpler than $f_e$, then we can expect predictions based on a large quantity of cheap data with a simple correction to be more accurate than predictions based on a small quantity of expensive data.

## 8.1 Co-Kriging

A more powerful method of calculating the correction processes is that of co-Kriging. The following presentation of the co-Kriging method is based on Forrester *et al*. (2007).

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

Co-Kriging is a form of Kriging that correlates multiple sets of data. Using our two sets of data, cheap and expensive, we begin the co-Kriging formulation by concatenating the sample locations to give the combined set of sample points

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_\mathrm{c} \\ \mathbf{X}_\mathrm{e} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_\mathrm{c}^{(1)} \\ \vdots \\ \mathbf{x}_\mathrm{c}^{(n_\mathrm{c})} \\ \mathbf{x}_\mathrm{e}^{(1)} \\ \vdots \\ \mathbf{x}_\mathrm{e}^{(n_\mathrm{e})} \end{pmatrix}.$$

As with Kriging, the value at a point in $\mathbf{X}$ is treated as if it were the realization of a stochastic process. For co-Kriging we therefore have the random field

$$\mathbf{Y} = \begin{pmatrix} \mathbf{Y}_\mathrm{c}(\mathbf{X}_\mathrm{c}) \\ \mathbf{Y}_\mathrm{e}(\mathbf{X}_\mathrm{e}) \end{pmatrix} = \begin{pmatrix} Y_\mathrm{c}(\mathbf{x}_\mathrm{c}^{(1)}) \\ \vdots \\ Y_\mathrm{c}(\mathbf{x}_\mathrm{c}^{(n_\mathrm{c})}) \\ Y_\mathrm{e}(\mathbf{x}_\mathrm{e}^{(1)}) \\ \vdots \\ Y_\mathrm{e}(\mathbf{x}_\mathrm{e}^{(n_\mathrm{e})}) \end{pmatrix}.$$

Here we use the *auto-regressive* model of Kennedy and O'Hagan (2000), which assumes that $\mathrm{cov}\left\{Y_e(\mathbf{x}^{(i)}), Y_c(\mathbf{x}) | Y_c(\mathbf{x}^{(i)})\right\} = 0, \ \forall \mathbf{x} \neq \mathbf{x}^{(i)}$. This means that no more can be learnt about $Y_\mathrm{e}(\mathbf{x}^{(i)})$ from the cheaper code if the value of the expensive function at $\mathbf{x}^{(i)}$ is known (this is known as a Markov property which, in essence, says we assume that the expensive simulation is correct and any inaccuracies lie wholly in the cheaper simulation).

Gaussian processes $Z_\mathrm{c}(.)$ and $Z_\mathrm{e}(.)$ represent the local features of the cheap and expensive codes. Using the auto-regressive model we are essentially approximating the expensive code as the cheap code multiplied by a constant scaling factor $\rho$ plus a Gaussian process $Z_\mathrm{d}(.)$, which represents the difference between $\rho Z_\mathrm{c}(.)$ and $Z_\mathrm{e}(.)$:

$$Z_\mathrm{e}(\mathbf{x}) = \rho Z_\mathrm{c}(\mathbf{x}) + Z_\mathrm{d}(\mathbf{x}). \tag{8.2}$$

Where in Kriging we have a covariance matrix $\mathrm{cov}\{\mathbf{Y}(\mathbf{X}), \ \mathbf{Y}(\mathbf{X})\} = \sigma^2 \boldsymbol{\Psi}(\mathbf{X}, \mathbf{X})$, we now have a covariance matrix constructed as follows:

$$\begin{aligned} \mathrm{cov}\{\mathbf{Y}_\mathrm{c}(\mathbf{X}_\mathrm{c}), \mathbf{Y}_\mathrm{c}(\mathbf{X}_\mathrm{c})\} &= \mathrm{cov}\{Z_\mathrm{c}(\mathbf{X}_\mathrm{c}), Z_\mathrm{c}(\mathbf{X}_\mathrm{c})\} \\ &= \sigma_\mathrm{c}^2 \boldsymbol{\Psi}_\mathrm{c}(\mathbf{X}_\mathrm{c}, \mathbf{X}_\mathrm{c}) \\ \mathrm{cov}\{\mathbf{Y}_\mathrm{e}(\mathbf{X}_\mathrm{e}), \mathbf{Y}_\mathrm{c}(\mathbf{X}_\mathrm{c})\} &= \mathrm{cov}\{\rho Z_\mathrm{c}(\mathbf{X}_\mathrm{c}) + Z_\mathrm{d}(\mathbf{X}_\mathrm{c}), Z_\mathrm{c}(\mathbf{X}_\mathrm{e})\} \\ &= \rho \sigma_c^2 \boldsymbol{\Psi}_\mathrm{c}(\mathbf{X}_\mathrm{c}, \mathbf{X}_\mathrm{e}) \end{aligned}$$

$$\begin{aligned}\operatorname{cov}\{\mathbf{Y}_e(\mathbf{X}_e), \mathbf{Y}_e(\mathbf{X}_e)\} &= \operatorname{cov}\{\rho Z_c(\mathbf{X}_e) + Z_d(\mathbf{X}_e), \rho Z_c(\mathbf{X}_e) + Z_d(\mathbf{X}_e)\} \\ &= \rho^2 \operatorname{cov}\{Z_c(\mathbf{X}_e), Z_c(\mathbf{X}_e)\} + \operatorname{cov}\{Z_d(\mathbf{X}_e), Z_d(\mathbf{X}_e)\} \\ &= \rho^2 \sigma_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_e) + \sigma_d^2 \boldsymbol{\Psi}_d(\mathbf{X}_e, \mathbf{X}_e).\end{aligned}$$

The notation $\boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_c)$, for example, denotes a matrix of correlations of the form $\psi_c$ between the data $\mathbf{X}_e$ and $\mathbf{X}_c$. Our complete covariance matrix is thus:

$$\mathbf{C} = \begin{pmatrix} \sigma_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c) & \rho \sigma_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_e) \\ \rho \sigma_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_c) & \rho^2 \sigma_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_e) + \sigma_d^2 \boldsymbol{\Psi}_d(\mathbf{X}_e, \mathbf{X}_e) \end{pmatrix}. \tag{8.3}$$

The correlations are of the same form as Equation (2.20), but there are two correlations, $\psi_c$ and $\psi_d$, and we therefore have more parameters to estimate: $\boldsymbol{\theta}_c$, $\boldsymbol{\theta}_d$, $\mathbf{p}_c$, $\mathbf{p}_d$ and the scaling parameter $\rho$. Our cheap data is considered to be independent of the expensive data and we can find MLEs for $\mu_c$, $\sigma_c^2$, $\boldsymbol{\theta}_c$ and $\mathbf{p}_c$ by maximizing the ln-likelihood (ignoring constant terms):

$$-\frac{n}{2}\ln(\sigma_c^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)| - \frac{(\mathbf{y}_c - \mathbf{1}\mu_c)^T \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)^{-1}(\mathbf{y}_c - \mathbf{1}\mu_c)}{2\sigma_c^2}. \tag{8.4}$$

By setting the derivatives of (8.4) w.r.t. $\mu_c$ and $\sigma_c^2$ to zero and solving, we find MLEs of

$$\widehat{\mu}_c = \frac{\mathbf{1}^T \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)^{-1} \mathbf{y}_c}{\mathbf{1}^T \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)^{-1} \mathbf{1}} \tag{8.5}$$

and

$$\widehat{\sigma}_c^2 = \frac{(\mathbf{y}_c - \mathbf{1}\widehat{\mu}_c)^T \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)^{-1}(\mathbf{y}_c - \mathbf{1}\widehat{\mu}_c)}{n_c}. \tag{8.6}$$

Substituting Equations (8.5) and (8.6) into (8.4) yields the concentrated ln-likelihood:

$$-\frac{n_c}{2}\ln(\widehat{\sigma}_c^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)| \tag{8.7}$$

and $\widehat{\boldsymbol{\theta}}_c$ and $\widehat{\mathbf{p}}_c$ (if not set at **2**) are found by maximizing this equation. This is performed in the same manner as for Kriging (see Section 2.4), using the function `likelihoodc.m` to calculate the concentrated ln-likelihood.

To estimate $\mu_d$, $\sigma_d^2$, $\boldsymbol{\theta}_d$, $\mathbf{p}_d$ and $\rho$, we first define

$$\mathbf{d} = \mathbf{y}_e - \rho \mathbf{y}_c(\mathbf{X}_e). \tag{8.8}$$

where $\mathbf{y}_c(\mathbf{X}_e)$ are the values of $\mathbf{y}_c$ at locations common to those of $\mathbf{X}_e$ (the Markov property implies that we only need to consider this data). If $\mathbf{y}_c$ is not available at $\mathbf{X}_e$, we may estimate $\rho$ at little additional cost by using Kriging estimates $\widehat{\mathbf{y}}_c(\mathbf{X}_e)$ found from Equation (2.40) using the already determined parameters $\widehat{\boldsymbol{\theta}}_c$ and $\widehat{\mathbf{p}}_c$. The ln-likelihood of the expensive data is now

$$-\frac{n}{2}\ln(\sigma_d^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c)| - \frac{(\mathbf{d} - \mathbf{1}\mu_d)^T \boldsymbol{\Psi}_d(\mathbf{X}_e, \mathbf{X}_e)^{-1}(\mathbf{d} - \mathbf{1}\mu_d)}{2\sigma_d^2}, \tag{8.9}$$

yielding MLEs of

$$\widehat{\mu}_{\mathrm{d}} = \frac{\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}_{\mathrm{d}}(\mathbf{X}_{\mathrm{e}}, \mathbf{X}_{\mathrm{e}})^{-1}\mathbf{d}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{\Psi}_{\mathrm{d}}(\mathbf{X}_{\mathrm{e}}, \mathbf{X}_{\mathrm{e}})^{-1}\mathbf{1}}$$

and

$$\widehat{\sigma}_{\mathrm{d}}^2 = \frac{(\mathbf{d} - \mathbf{1}\widehat{\mu}_{\mathrm{d}})^{\mathrm{T}}\boldsymbol{\Psi}_{\mathrm{d}}(\mathbf{X}_{\mathrm{e}}, \mathbf{X}_{\mathrm{e}})^{-1}(\mathbf{d} - \mathbf{1}\widehat{\mu}_{\mathrm{d}})}{n_{\mathrm{e}}}, \tag{8.10}$$

with $\widehat{\boldsymbol{\theta}}_{\mathrm{d}}$, $\widehat{\mathbf{p}}_{\mathrm{d}}$ (again, if not set at **2**) and $\widehat{\rho}$ found by maximizing

$$-\frac{n_{\mathrm{e}}}{2}\ln(\widehat{\sigma}_{\mathrm{d}}^2) - \frac{1}{2}\ln|\boldsymbol{\Psi}_{\mathrm{c}}(\mathbf{X}_{\mathrm{c}}, \mathbf{X}_{\mathrm{c}})|. \tag{8.11}$$

The *MATLAB* function `likelihoodd.m` below can be used for this calculation.

```
function NegLnLiked=likelihoodd (x)
    % Calculates the negative of the concentrated ln-likelihood
    %
    % Inputs:
    %     x - vetor of log(thetad) parameters
    %
    % Global variables used:
    %     ModelInfo.Xe - ne x k matrix of expensive sample locations
    %     ModelInfo.ye - ne x 1 vector of expensive observed data
    %     ModelInfo.yc - nc x 1 vector of cheap observed data
    %
    % Outputs:
    %     NegLnLiked-concentrated ln-likelihood * -1 for minimizing
    %
global ModelInfo
Xe=ModelInfo.Xe;
ye=ModelInfo.ye;
yc=ModelInfo.yc;
[ne,k]=size(Xe);
thetad=10.^x(1:k);
rho=x(k+1);
one=ones(ne,1);

% Pre - allocate memory
PsidXe=zeros(ne,ne);

% Build upper half of correlation matrix
for i=1:ne
    for j=i+1:ne
        PsidXe(i,j)=exp(-sum(thetad.*(Xe(i,:) ...
        -Xe(j,:)).^2));
    end
end

% Add upper and lower halves and diagonal of ones plus
```

(*continued*)

```
% small number to reduce ill-conditioning
PsidXe=PsidXe+PsidXe'+eye(ne)+eye(ne).*eps;

% Cholesky factorization
[U,p]=chol(PsidXe);

% Use penalty if ill-conditioned
if p > 0
    NegLnLiked=1e4;
else
    % Sum lns of diagonal to find ln(det(Psi))
    LnDetPsidXe=2*sum(log(abs(diag(U))));

    % Difference vector
    d=ye-rho.*yc(end-ne+1:end);

    % Use back-substitution of Cholesky instead of inverse
    mud=(one'*(U\(U'\d)))/(one'*(U\(U'\one)));
    SigmaSqrd=(d-one.*mud)'*(U\(U'\(d-one.*mud)))/ne;
    NegLnLiked=-1*(-(ne/2)*log(SigmaSqrd)-0.5*LnDetPsidXe);
end
```

As with Kriging (see Section 2.4), Equations (8.7) and (8.11) must be maximized numerically using a suitable global search routine such as a genetic algorithm. Depending upon the cost of evaluating the cheap and expensive functions $f_c$ and $f_e$, for very high dimensional problems the multiple matrix inversions involved in the likelihood maximization may render the use of the co-Kriging model impractical (the size of the matrices depends directly on the quantities of data available, and the number of search steps needed in the MLE process is linked to the number of parameters being tuned). Typically a statistical model used as a surrogate will be tuned many fewer times than the number of evaluations of $f_e$ required by a direct search. The cost of tuning the model can therefore be allowed to exceed the cost of computing $f_e$ and still provide significant speed-up. For large $k$ and $n$ the time required to find MLEs can be reduced by using a constant $\theta_{c,j}$ and $\theta_{d,j}$ for all elements of $\boldsymbol{\theta}_c$ and $\boldsymbol{\theta}_d$ to simplify the maximization, though this may affect the accuracy of the approximation.

To derive the co-Kriging predictor we follow a similar method to that of ordinary Kriging (recall Section 2.4). The basis of this method is that we wish our prediction of a new expensive point to be consistent with the observed data and the MLEs for the model parameters. We therefore augment the observed data with a predicted value and maximize the likelihood of this augmented data set by varying our prediction while keeping the model parameters fixed. This gives us a MLE $\widehat{y}_e(\mathbf{x})$.

The augmented data set is defined as $\widetilde{\mathbf{X}} = \{\mathbf{X}_c^T \ \mathbf{X}_e^T \mathbf{x}^T\}^T$ and $\widetilde{\mathbf{y}} = \{\mathbf{y}_c^T \mathbf{y}_e^T \ \widehat{y}_{e(\mathbf{x})}\}^T$, with the covariance matrix $\widetilde{\mathbf{C}}$ given by

$$\begin{pmatrix} \widehat{\sigma}_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_c) & \rho \widehat{\sigma}_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_c, \mathbf{X}_e) & \rho \widehat{\sigma}_c^2 \boldsymbol{\psi}_c(\mathbf{X}_c, \mathbf{x}) \\ \rho \widehat{\sigma}_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_c) & \rho_c^2 \widehat{\sigma}_c^2 \boldsymbol{\Psi}_c(\mathbf{X}_e, \mathbf{X}_e) + \widehat{\sigma}_d^2 \boldsymbol{\Psi}_d(\mathbf{X}_e, \mathbf{X}_e) & (\rho^2 \widehat{\sigma}_c^2 + \widehat{\sigma}_d^2) \boldsymbol{\psi}_d(\mathbf{X}_e, \mathbf{x}) \\ \rho \widehat{\sigma}_c^2 \boldsymbol{\psi}_c(\mathbf{X}_c, \mathbf{x})^T & (\rho^2 \widehat{\sigma}_c^2 + \widehat{\sigma}_d^2) \boldsymbol{\psi}_d(\mathbf{X}_e, \mathbf{x})^T & \rho^2 \widehat{\sigma}_c^2 + \widehat{\sigma}_d^2 \end{pmatrix},$$

which, defining $\mathbf{c}$ as a column vector of the covariance of $\mathbf{X}$ and $\mathbf{x}$, can be expressed as

$$\widetilde{\mathbf{C}} = \begin{pmatrix} \mathbf{C} & \mathbf{c} \\ \mathbf{c}^{\mathrm{T}} & \rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 \end{pmatrix}. \tag{8.12}$$

In Equations (8.4) and (8.9) it is seen that only the last term of the ln-likelihood contains the sample data and so to find a MLE $\widehat{y}_{\mathrm{e}}(\mathbf{x})$ we need to maximize:

$$-\frac{1}{2}(\widetilde{\mathbf{y}} - \mathbf{1}\mu)^{\mathrm{T}}\widetilde{\mathbf{C}}^{-1}(\widetilde{\mathbf{y}} - \mathbf{1}\mu),$$

which may be expressed as

$$-\frac{1}{2}\begin{pmatrix} \mathbf{y} - \mathbf{1}\widehat{\mu} \\ \widehat{y}_{\mathrm{e}}(\mathbf{x}) - \widehat{\mu} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \mathbf{C} & \mathbf{c} \\ \mathbf{c}^{\mathrm{T}} & \rho_{\mathrm{c}}^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{y} - \mathbf{1}\widehat{\mu} \\ \widehat{y}_{\mathrm{e}}(\mathbf{x}) - \widehat{\mu} \end{pmatrix}. \tag{8.13}$$

The inverse of the augmented covariance matrix $\widetilde{\mathbf{C}}^{-1}$ is found using the partitioned inverse formula (Theil, 1971) (recall the mathematical note in Section 2.4):

$$\begin{pmatrix} \mathbf{C}^{-1} + \mathbf{C}^{-1}\mathbf{c}(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})^{-1}\mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1} & -\mathbf{C}^{-1}\mathbf{c}(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})^{-1} \\ -(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})^{-1}\mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1} & (\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})^{-1} \end{pmatrix}. \tag{8.14}$$

Substituting (8.14) into (8.13) and ignoring terms without $\widehat{y}_{\mathrm{e}}(\mathbf{x})$ we obtain

$$\left(\frac{-1}{2(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})}\right)(\widehat{y}_{\mathrm{e}}(\mathbf{x}) - \widehat{\mu})^2 + \left(\frac{\mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu})}{(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})}\right)(\widehat{y}_{\mathrm{e}}(\mathbf{x}) - \widehat{\mu}).$$

This expression is maximized by taking the derivative with respect to $\widehat{y}_{\mathrm{e}}(\mathbf{x})$ and setting to zero:

$$\left(\frac{-1}{\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c}}\right)(\widehat{y}_{\mathrm{e}}(\mathbf{x}) - \widehat{\mu}) + \left(\frac{\mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu})}{(\rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c})}\right) = 0. \tag{8.15}$$

Solving for $\widehat{y}_{\mathrm{e}}(\mathbf{x})$ now gives

$$\widehat{y}_{\mathrm{e}}(\mathbf{x}) = \widehat{\mu} + \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}(\mathbf{y} - \mathbf{1}\widehat{\mu}). \tag{8.16}$$

If we make a prediction at one of the expensive points, $\mathbf{x}^{(n+1)} = \mathbf{x}_{\mathrm{e}}^{(i)}$ and $\mathbf{c}$ is the $n_{\mathrm{c}} + i$th column of $\mathbf{C}$, then $\mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}$ is the $n_{\mathrm{c}} + i$th unit vector and $\widehat{y}_{\mathrm{e}}(\mathbf{x}_{\mathrm{e}}^{(i)}) = \widehat{\mu} + \mathbf{y}^{(n_{\mathrm{c}}+i)} - \widehat{\mu} = y_{\mathrm{e}}^{(i)}$. We see, therefore, that Equation (8.16) is an interpolator of the expensive data (just like ordinary Kriging), but will in some sense regress the cheap data unless it coincides with $\mathbf{y}_{\mathrm{e}}$.

The estimated mean squared error in this prediction is similar to the Kriging error (Equation (3.1)), and is calculated as

$$\widehat{s}^2(\mathbf{x}) \approx \rho^2\widehat{\sigma}_{\mathrm{c}}^2 + \widehat{\sigma}_{\mathrm{d}}^2 - \mathbf{c}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c} + \frac{1 - \mathbf{1}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{c}}{\mathbf{1}^{\mathrm{T}}\mathbf{C}^{-1}\mathbf{1}}. \tag{8.17}$$

For $\mathbf{x} = \mathbf{x}_e^{(i)}$, $\mathbf{c}^T\mathbf{C}^{-1}$ is the $n_c + i$th unit vector, $\mathbf{c}^T\mathbf{C}^{-1}\mathbf{c} = \mathbf{c}^{(n_c+i)} = \rho_c^2\sigma_c^2 + \sigma_d^2$ and so $\widehat{s}^2(\mathbf{x})$ is zero (just like ordinary Kriging). For $\mathbf{X}_c \setminus \mathbf{X}_e$, $s^2(\mathbf{x}) \neq 0$ unless $\mathbf{y}_e = \mathbf{y}_c(\mathbf{X}_e)$. The error at these points is determined by the character of $Y_d$. If this difference between $\rho Y_c(\mathbf{X}_e)$ and $Y_e(\mathbf{X}_e)$ is simple (characterized by low $\theta_{d,j}$'s) the error will be low, whereas a more complex difference (high $\theta_{d,j}$'s) will lead to high error estimates.

By using Equation (8.17) we can formulate the various infill criteria discussed in Section 3.2. *MATLAB* code to calculate these infill criteria is available on the book website.

As shown for ordinary Kriging in Chapter 6, a regression parameter can be added to the leading diagonal of the correlation matrix when noise is present. In fact, two parameters may be used: one for the cheap data and one for the expensive data.

## 8.2 One-variable Demonstration

We will now look at how co-Kriging behaves using a simple one-variable function example. Imagine that our expensive to compute data is calculated by the function $f_e(x) = (6x - 2)^2 \sin(12x - 4)$, $x \in [0, 1]$, and a cheaper estimate of this data is given by $f_c(x) = Af_e + B(x - 0.5) - C$. We sample the design space extensively using the cheap function at $\mathbf{X}_c = \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, but only run the expensive function at four of these points, $\mathbf{X}_e = \{0, 0.4, 0.6, 1\}$.

Figure 8.1 shows the functions $f_e$ and $f_c$ with $A = 0.5$, $B = 10$ and $C = -5$. A Kriging prediction through $\mathbf{y}_e$ gives a poor approximation to the deliberately deceptive function, but the co-Kriging prediction lies very close to $f_e$, being better than both the standard Kriging

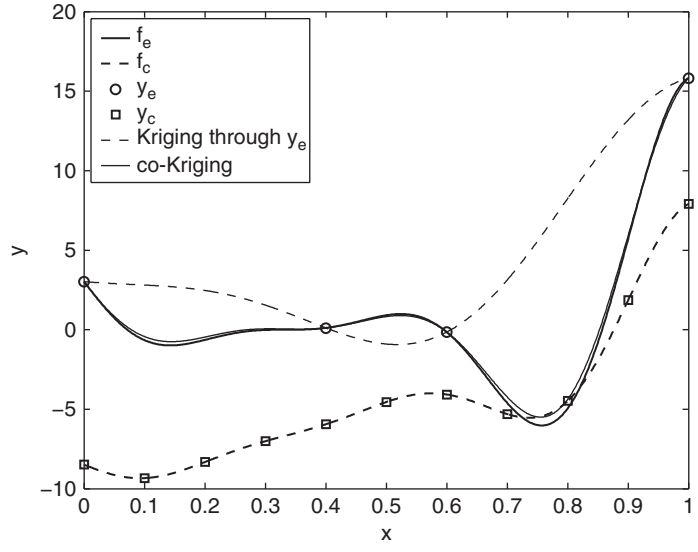



**Figure 8.1.** A one variable co-Kriging example. The Kriging approximation using four expensive data points ($\mathbf{y}_e$) has been significantly improved using extensive sampling from the cheap function ($\mathbf{y}_c$).

model and the cheap data. This co-Kriging prediction was produced using the *MATLAB* code below:

```
global ModelInfo
% Expensive points
ModelInfo.Xe=[0; 0.4; 0.6; 1];

% Cheap points
ModelInfo.Xc=[0.1;0.2;0.3;0.5;0.7;0.8;0.9;0;0.4;0.6;1];
k=1;

% Calculate expensive observations
for i=1:size(ModelInfo.Xe,1)
    ModelInfo.ye(i,1)=onevar(ModelInfo.Xe(i));
end

% Calculate cheap observations
for i=1:size(ModelInfo.Xc,1)
    ModelInfo.yc(i,1)=cheaponevar(ModelInfo.Xc(i));
end

% Estimate cheap model parameters
ModelInfo.Thetac=fminbnd(@likelihoodc, -3,3);

% Estimate difference model parameters
Params=ga(@likelihoodd, k+1,[],[],[],[],[-3 -5],[3 5]);
ModelInfo.Thetad=Params(1:k);
ModelInfo.rho=Params (k+1);

% Construct covariance matrix
buildcokriging

% Make predictions in range 0,1
Xplot=0:0.01:1;
ModelInfo.Option='Pred';
for i=1:101
    pred(i)=cokrigingpredictor(Xplot(i));
end
```

Note that, since there is no need to calculate $\mathbf{C}$ in either of the likelihood maximizations, an additional function `buildcokriging.m` is used for this purpose. Despite the considerable differences between $f_e$ and $f_c$, a simple relationship has been found between the expensive and cheap data and the estimated error reduces almost to zero at $\mathbf{X}_c$ (see Figure 8.2).

We have chosen the relationship between our low- and high-fidelity test function in order to show how the parameters of the co-Kriging model behave. The parameters pertaining to the cheap data, $\boldsymbol{\theta}_c$ and $\mathbf{p}_c$, are only affected by this data and behave as described in Section 2.4. Moving on to the scaling parameter $\rho$: if our cheap model parameter $A$ (the multiplying term linking the cheap and expensive functions) is varied such that $1/A \in [-10, 10]$, we obtain the values for $\widehat{\rho}$ shown in Figure 8.3 and see that $\widehat{\rho} = 1/A$. Similar trials show that the parameters $B$ and $C$ have no effect on $\rho$. Thus we

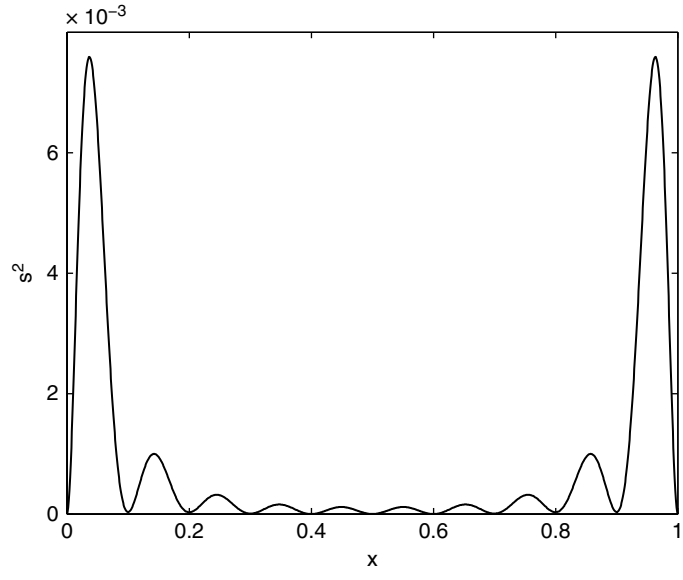



**Figure 8.2.** Estimated error in the co-Kriging prediction in Figure 8.1. The simple relationship between the data results in low error estimates at $\mathbf{X}_c$ as well as $\mathbf{X}_e$.

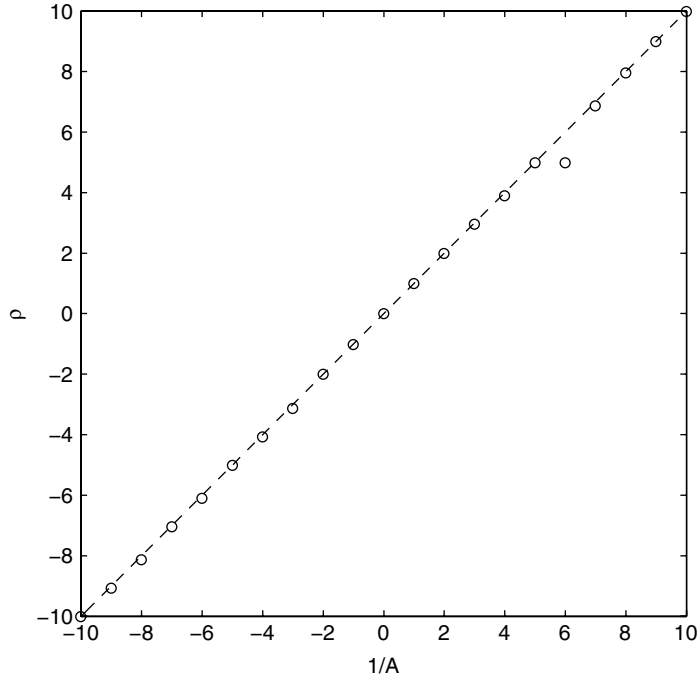



**Figure 8.3.** This plot of $\widehat{\rho}$ versus $1/A$ shows that the MLE for $\rho$ is a scaling factor between $Z_c(.)$ and $Z_e(.)$, following the formulation in Equation (8.2). There is a singularity at $1/A = 0$ so we have used $1/A = 0.01$ at this point.

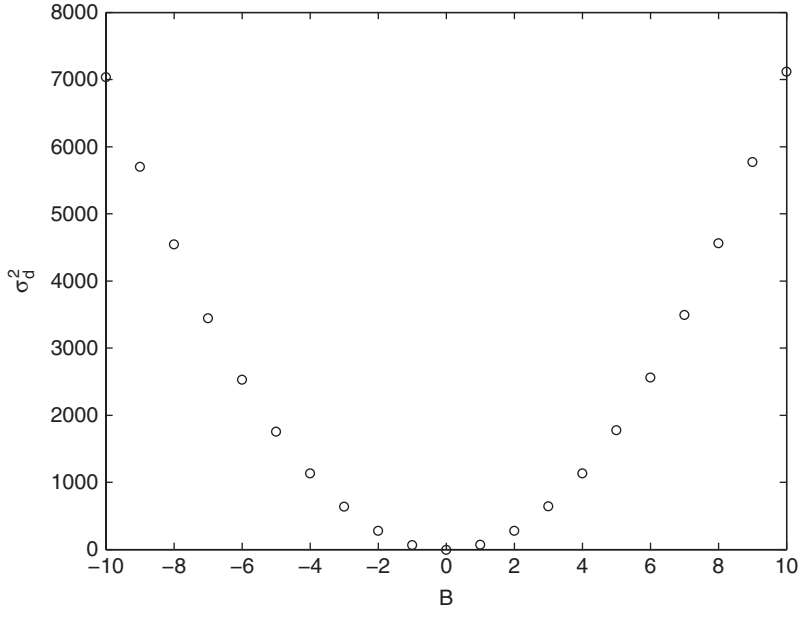



**Figure 8.4.** Variance $\sigma_d^2$ as the cheap function coefficient $B$ is altered. The variance reduces to zero as the difference between $f_e$ and $f_c$ can be modelled purely by the scaling parameter $\rho$.

see that $\rho$ is purely a scaling parameter. Note that $\widehat{\rho}$ is only an *indicator* of the scaling, since this value is estimated based on the data available. For the data in Figure 8.1, $\widehat{\rho} = 1.87$ (close to the true value of 2), but for small samples of $\mathbf{y}_e$ the MLE may be misleading (the slight deviations of the data in Figure 8.3 from $\widehat{\rho} = 1/A$ (shown as a dashed line) are where our GA search has not found the true MLE).

Recall that $\mathbf{d} = \mathbf{y}_e - \rho \mathbf{y}_c(\mathbf{X}_e)$ (Equation (8.8)) and so, with $\widehat{\rho} \approx \mathbf{y}_e/\mathbf{y}_c$, $\mathbf{d}$ represents the difference in trends between the cheap and expensive data. Thus for our one-variable example, when $B, C = 0$, $\widehat{\mu}_d, \widehat{\sigma}_d^2 \to 0$ for all values of $A$ if $\widehat{\rho}$ is estimated accurately. Figure 8.4 shows how $\widehat{\sigma}_d^2$ varies for $B \in [-10, 10]$ and we see that, as $B \to 0$ and therefore $\mathbf{d} \to \mathbf{0}$, $\widehat{\sigma}_d^2$ also approaches zero. Note that $\widehat{\boldsymbol{\theta}}_e$ and $\widehat{\mathbf{p}}_e$ will not be affected, since the correlation in Equation (2.20) is unaffected by the scaling of the objective data (it is, however, affected by the scaling of $\mathbf{X}$).

## 8.3 Choosing $\mathbf{X_c}$ and $\mathbf{X_e}$

Selecting $n_c$ and $n_e$ and the corresponding $\mathbf{X}_c$ and $\mathbf{X}_e$ is trivial in our one-variable example, but in higher dimensions the process is rather more challenging. Assuming $f_c$ is cheap, we can use an extravagantly large $n_c$ to ensure that $\widehat{f}_c$ is globally accurate. If, however, each $y_c^{(i)}$ evaluation incurs a significant cost, albeit lower than that for a $y_e^{(i)}$ evaluation, we may want to be rather more conservative in our choice of $n_c$. The resulting $\mathbf{X}_c$ (found using the techniques of Chapter 1) can then be augmented according to an exploration based infill strategy until an accurate model of $f_c$ is produced (see Section 3.3.3). With $\mathbf{X}_c$ in place, a

particularly conservative value of $n_e$ can be chosen and a space-filling $\mathbf{X}_e$ selected from $\mathbf{X}_c$ according to either the greedy search or exchange algorithm in Section 1.4.4. We can then proceed with an infill strategy from Section 3.2.

## 8.4 Summary

Our choice of a cheap function for the above example is somewhat contrived, but this has allowed us to show that the co-Kriging model and its parameters are behaving as we would expect. For our test function the correction process $Z_d(.)$ is linear. Co-Kriging will work effectively for more complex correction processes with the proviso that $Z_d(.)$ must be simpler to model than $Z_e(.)$. Although we have only considered combining two levels of analyses, the co-Kriging method can be extended to multiple levels by using additional $\rho$'s and $\mathbf{d}$'s (see Kennedy and O'Hagan, 2000, for more details).

As mentioned at the beginning of this chapter, multi-level modelling can be achieved simply by combining independent surrogates of the ratios or differences between data. However, the co-Kriging method is more powerful, both in terms of the complexity of relationships it can handle and its ability to provide error estimates which can be used to formulate the infill criteria from Section 3.2.

## References

Forrester, A. I. J., Sóbester, A. and Keane, A. J. (2007) Multi-fidelity optimization via surrogate modelling. *Proceedings of the Royal Society A*, **463**(2088), 3251–3269.

Kennedy, M. C. and O'Hagan, A. (2000) Predicting the output from complex computer code when fast approximations are available. *Biometrika*, **87**(1), 1–13.

Theil, H. (1971) *Principles of Econometrics*, John Wiley & Sons, Inc., New York.

# 9

# Multiple Design Objectives

As already noted at the beginning of Chapter 5, engineering design is almost always concerned with most problems that have multiple, often conflicting goals and a host of demanding constraints. We turn now to the use of surrogate based approaches to design improvement when dealing explicitly with multiple objectives. We note in passing that in some cases the designer may be able to reduce problems with multiple goals to single objective problems by some suitable weighting function that combines the goals of interest. In aerospace applications this can often be something like weight, payload capacity or cost, provided suitable conversions to a common form can be devised. When such an approach can be taken the problem reverts to a single objective search and the techniques already discussed can be applied. Sometimes, however, the correct weighting to apply between goals is not obvious or the designer does not wish to commit to a fixed weighting while carrying out design searches: this leads to the concept of Pareto optimality and sets of designs that must be found and considered simultaneously.

## 9.1 Pareto Optimization

In aerospace design, for example, it is common to be aiming for light weight, low cost, robust, high performance systems. These aspirations are clearly in tension with each other and so compromise solutions have to be sought. The final selection between such compromises inevitably involves deciding on some form of weighting between the goals. However, before this stage is reached it is possible to study design problems from the perspective of Pareto sets. A Pareto set of designs is one whose members are all optimal in some sense, but where the relative weighting between the competing goals is yet to be finally fixed (see, for example, Fonseca and Fleming, 1995). More formally, a Pareto set of designs contains systems that are sufficiently optimized that, to improve the performance of any set member in any one goal function, its performance in at least one of the other functions must be made worse. Moreover, the designs in the set are said to be *non-dominated* in that no other set member exceeds a given design's performance in all goals. It is customary to illustrate

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

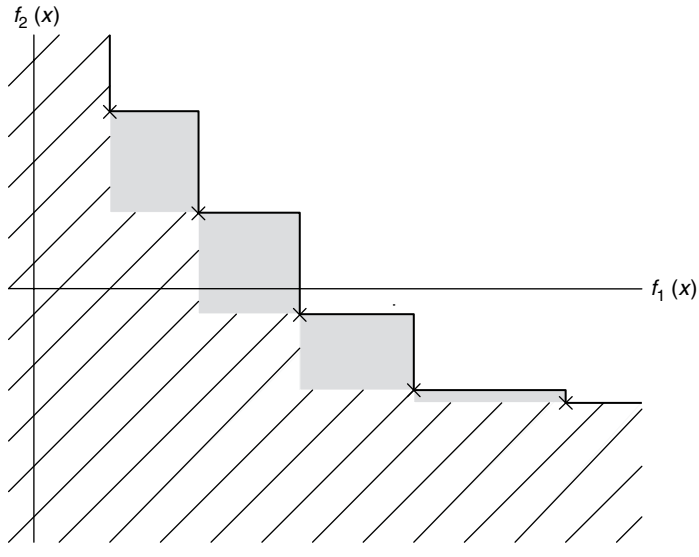



**Figure 9.1.** A Pareto set of five non-dominated points ($\times$) for a problem with two objectives. The solid line is the Pareto front. The shaded area shows where new points would augment the Pareto front, while the hatched area is where new points would dominate and replace the existing set of non-dominated points.

a Pareto set by plotting the performance of its members against each goal function (see Figure 9.1, where the two axes are for two competing goal functions that must both be minimized). The series of horizontal and vertical lines joining the set members is referred to as the *Pareto front* – any design lying above and to the right of this line is dominated by members of the set.

There are a number of technical difficulties associated with constructing Pareto sets. First, the set members need to be optimal in some sense. Since it is desirable to have a good range of designs in the set, this means that an order of magnitude of more optimization effort is usually required to produce such a set than to find a single design that is optimal against just one goal. Second, it is usually necessary to provide a wide and even coverage in the set in terms of the goal function space. Since the mapping between design parameters and goal functions is usually highly nonlinear, gaining such coverage is far from simple. Finally, and in common with a single objective design, many problems of practical interest involve the use of expensive computer simulations to evaluate the performance of each candidate, and this means that only a limited number of such simulations can usually be afforded.

Currently, there appear to be two popular ways of constructing Pareto sets. First, and most simply, one chooses a weighting function to combine all the goals in the problem of interest into a single quantity and carries out a single objective optimization. The weighting function is then changed and the process repeated. By slowly working through a range of weightings it is possible to build up a Pareto set of designs. This approach allows the full gamut of single objective search methods to be applied, including the use of the

sampling and surrogate modelling technologies in this book. It does, however, suffer from a major drawback: it is by no means clear what weighting function to use and how to alter it so as to be able to reach all parts of the potential design space (and thus to have a wide-ranging Pareto set). The nonlinear nature of most design problems will make it very difficult to ensure that the designs achieved are reasonably evenly spaced out through the design space.

In an attempt to address this limitation designers have turned to a second way of constructing Pareto sets via the use of population based search schemes. In such schemes a set of designs is worked on concurrently and evolved towards the final Pareto set in one process. In doing this, designs are compared to each other and progressed if they are of high quality and if they are widely spaced apart from other competing designs. Moreover, such schemes usually avoid the need for an explicit weighting function to combine the goals being studied. Perhaps the most well known of these schemes is the NSGA-II method introduced by Deb *et al*. (2002). In this approach a genetic algorithm is used to carry out the search but the goal function used to drive the genetic process is based on the relative ranking and spacing of the designs in the set rather than their combined weighted performance. More specifically, at each generation all the designs are compared and the non-dominated designs set to one side. These are assigned rank one. The remaining designs are compared and those that now dominate are assigned rank two and so on. Thus the whole population is sorted into rank order based on dominance. This sorting into rank order dominance can be carried out irrespective of the relative importance of the objectives being dealt with or the relative magnitudes and scalings of these quantities.

Having sorted the population of designs into ranks they are next rewarded or penalized depending on how close they are to each other in goal space (and sometimes also in design variable space). This provides pressure to cause the search to fan out and explore the whole design space, but does require that the competing objectives be suitably scaled – an issue that arises in many aspects of dealing with multi-objective approaches to design. When combined with the traditional genetic algorithm operators of selection, crossover and mutation, the NSGA-II scheme is remarkably successful in evolving high quality Pareto sets. As originally described, however, no means were provided for mitigating run time issues arising from using expensive computer simulations in assessing competing designs.

To overcome the problem of long run times a number of workers have advocated the use of surrogate modelling approaches within Pareto front frameworks (Wilson *et al*., 2001; Knowles and Hughes, 2005). It is also possible to combine tools such as NSGA-II with surrogates (Voutchkov *et al*., 2006). In such schemes an initial sampling plan is evaluated and surrogate models built as per the single objective case, but now there is one surrogate for each goal function. In the NSGA-II approach the search is simply applied to the resulting surrogates and used to produce a Pareto set of designs. These designs are then used to form an infill point set and, after running full computations, the surrogates are refined and the approach continued. Although sometimes quite successful, this approach suffers from an inability to explicitly balance exploration and exploitation in the surrogate model construction, in just the same way as when using a prediction based infill criterion in single objective search (recall Section 3.2.1), although the crowding or niching measures normally used help mitigate these problems to some extent. Here we will additionally consider statistically based operators for use in surrogate model based multi-objective search so as to explicitly tackle this problem.

## 9.2 Multi-objective Expected Improvement

To begin with, consider a problem where there is a need to minimize two objective functions $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$, which we can sample to find observed outputs $y_1$ and $y_2$. For simplicity, assume that $\mathbf{x}$ consists of just one design variable $x$ ($k = 1$). By evaluating a sampling plan, $X$, we can obtain observed responses $\mathbf{y}_1$ and $\mathbf{y}_2$. This will allow us to identify the initial Pareto set of $m$ designs that dominate all the others in the training set:

$$\mathbf{y}^*_{1,2} = \left\{ [y_1^{*(1)}(x^{*(1)}), y_2^{*(1)}(x^{*(1)})], [y_1^{*(2)}(x^{*(2)}), y_2^{*(2)}(x^{*(2)})], \ldots, [y_1^{*(m)}(x^{*(m)}), y_2^{*(m)}(x^{*(m)})] \right\}$$

In this set the superscript $^*$ indicates that the designs are non-dominated. We may plot these results on the Pareto front axes as per Figure 9.1 discussed in the previous section. In that figure the solid line is the Pareto front and the hatched and shaded areas represents locations where new designs would need to lie if they are to become members of the Pareto set. Note that if new designs lie in the shaded area they augment the set and that if they lie in the hatched area they will replace at least one member of the set (since they will then dominate some members of the old set). It is possible to set up our new metric such that an improvement is achieved if we can augment the set or, alternatively, only if we can dominate at least one set member – here we consider the latter metric only.

Given the training set it is also possible to build a pair of Gaussian process based models (e.g. Kriging models). As when dealing with constrained surrogates, it is assumed that these models are independent (though it is also possible to build correlated models by using co-Kriging, as per Chapter 8). The Gaussian processes have means $\widehat{y}_1(x)$ and $\widehat{y}_2(x)$ (the MLE predictions, from Equation (2.40)), and variances $\widehat{s}_1^2(x)$ and $\widehat{s}_2^2(x)$ (from Equation (3.1)). These values may then be used to construct a two-dimensional Gaussian probability density function for the predicted responses of the form

$$\phi(Y_1, Y_2) = \frac{1}{\widehat{s}_1(x)\sqrt{2\pi}} \exp\left[ -\frac{(Y_1(x) - \widehat{y}_1(x))^2}{2\widehat{s}_1^2(x)} \right] \frac{1}{\widehat{s}_2(x)\sqrt{2\pi}} \exp\left[ -\frac{(Y_2(x) - \widehat{y}_2(x))^2}{2\widehat{s}_2^2(x)} \right], \tag{9.1}$$

where it is made explicitly clear that $\widehat{y}_1(x)$, $\widehat{s}_1^2(x)$, $\widehat{y}_2(x)$ and $\widehat{s}_2^2(x)$ are all functions of the location at which an estimate is being sought. Clearly this joint probability density function accords with the predicted mean and errors coming from the two Kriging models at $x$. When seeking to add a new point to the training data we wish to know the likelihood that any newly calculated point will be good enough to become a member of the current Pareto set and, when comparing competing potential designs, which will improve the Pareto set most.

We first consider the probability that a new design at $x$ will dominate a single member of the existing Pareto set, say $[y_1^{*(1)}, y_2^{*(1)}]$. For a two-objective problem this may arise in one of three ways: either the new point improves over the existing set member in goal one, or in goal two, or in both (see Figure 9.2). The probability of the new design being an improvement is simply $P[Y_1(x) < y_1^{*(i)} \cap Y_2(x) < y_2^{*(i)}]$, which is given by integrating the volume under the joint probability density function, i.e. by integrating over the hatched area in Figure 9.2, to get

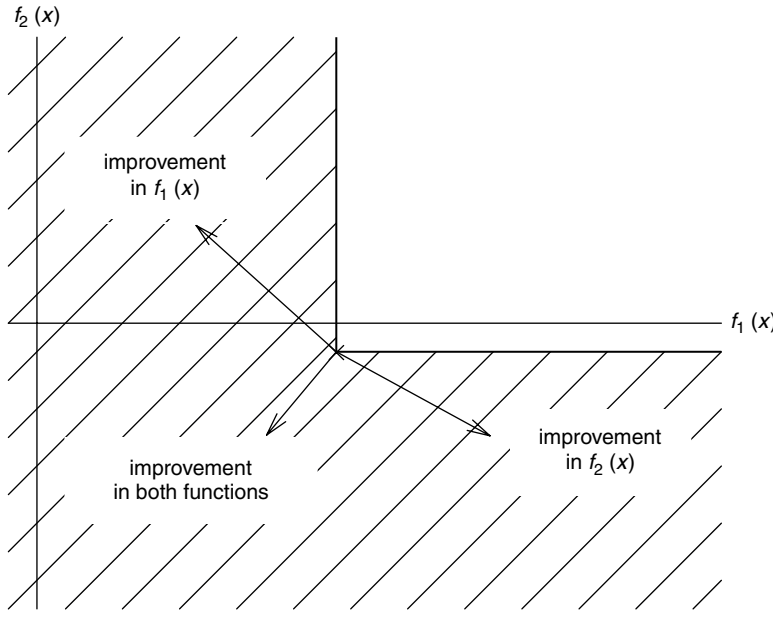



**Figure 9.2.** Improvements possible from a single point in the Pareto set.

$$P[Y_1(x) < y_1^{*(i)} \cap Y_2(x) < y_2^{*(i)}] = \Phi\left(\frac{y_1^{*(i)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right) + \Phi\left(\frac{y_2^{*(i)} - \widehat{y}_2(x)}{\widehat{s}_2(x)}\right) - \Phi\left(\frac{y_1^{*(i)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right)\Phi\left(\frac{y_2^{*(i)} - \widehat{y}_2(x)}{\widehat{s}_2(x)}\right), \tag{9.2}$$

where $\mathbf{\Phi}(.)$ is the Gaussian cumulative distribution function.

Next consider the probability that the new point is an improvement, given all the points in the Pareto set. Now we must integrate over the hatched area in Figure 9.1. We can distinguish whether we want the new point to augment the existing Pareto set or dominate at least one set member by changing the area over which the integration takes place. Here we will only consider points which dominate the Pareto set (for formulations which deal with Pareto set augmentation see Keane and Nair, 2005). Carrying out the desired integral is best done by considering the various rectangles that comprise the hatched area in Figure 9.1 and this gives

$$\begin{aligned} P[Y_1(x) < \mathbf{y}_1^* \cap Y_2(x) < \mathbf{y}_2^*] = & \int_{-\infty}^{y_1^{*(1)}} \int_{-\infty}^{\infty} Y_1 \phi(Y_1, Y_2)\, \mathrm{d}Y_2\, \mathrm{d}Y_1 \\ & + \sum_{i=1}^{m-1} \int_{y_1^{*(i)}}^{y_1^{*(i+1)}} \int_{-\infty}^{y_2^{*(i+1)}} Y_1 \phi(Y_1, Y_2)\, \mathrm{d}Y_2\, \mathrm{d}Y_1 \\ & + \int_{y_1^{*(m)}}^{\infty} \int_{-\infty}^{y_2^{*(m)}} Y_1 \phi(Y_1, Y_2)\, \mathrm{d}Y_2\, \mathrm{d}Y_1, \end{aligned} \tag{9.3}$$

or

$$P[Y_1(x) < \mathbf{y}_1^* \cap Y_2(x) < \mathbf{y}_2^*] = \Phi\left(\frac{y_1^{*(i)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right) + \sum_{i=1}^{m-1}\left\{\Phi\left(\frac{y_1^{*(i+1)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right) - \Phi\left(\frac{y_1^{*(i)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right)\right\} \times \Phi\left(\frac{y_2^{*(i+1)} - \widehat{y}_2(x)}{\widehat{s}_2(x)}\right) + \left\{1 - \Phi\left(\frac{y_1^{*(m)} - \widehat{y}_1(x)}{\widehat{s}_1(x)}\right)\right\}\Phi\left(\frac{y_2^{*(m)} - \widehat{y}_2(x)}{\widehat{s}_2(x)}\right). \quad (9.4)$$

This is the multi-objective equivalent of the $P[I(\mathbf{x})]$ formulation used in Section 3.2.3. It will work irrespective of the relative scaling of the objectives being dealt with. When used as an infill criterion it will not, however, necessarily encourage very wide ranging exploration since it is not biased by the degree of improvement being achieved. To do this we must consider the first moment of the integral, as before when dealing with single objective problems (recall Section 3.2.3).

The equivalent improvement metric we require for the two objective cases will be the first moment of the joint probability density function integral taken over the area where improvements occur, calculated about the current Pareto front. Now, while it is simple to understand the region over which the integral is to be taken (it is just the same as in Equation (9.3)) the moment arm about the current Pareto front is a less obvious concept. To understand what is involved, it is useful to return to the geometrical interpretation of $E[I(\mathbf{x})]$ (shown in Figure 3.12 for the single objective case). $P[I(\mathbf{x}^*)]$ represents integration over the probability density function in the area below and to the left of the Pareto front where improvements can occur. $E[I(\mathbf{x}^*)]$ (we will use the $^*$ superscript to denote the multi-objective formulation) is the first moment of the integral over this area about the Pareto front. Now the distance the centroid of the $E[I(\mathbf{x}^*)]$ integral lies from the front is simply $E[I(\mathbf{x}^*)]$ divided by $P[I(\mathbf{x}^*)]$ (see Figure 9.3). Given this position and $P[I(\mathbf{x}^*)]$ it is simple to calculate $E[I(\mathbf{x}^*)]$ based on any location along the front. Hence we first calculate $P[I(\mathbf{x}^*)]$ and the location of the centroid of its integral, $(\bar{Y}_1, \bar{Y}_2)$ (by integration with respect to the origin and division by $P[I(\mathbf{x}^*)]$). It is then possible to establish the Euclidean distance the centroid lies from each member of the Pareto set. The expected improvement criterion is subsequently calculated using the set member closest to the centroid, $(y_1^*(x^*), y_2^*(x^*))$, by taking the product of the volume under the probability density function with the Euclidean distance between this member and the centroid, shown by the arrow in Figure 9.3. This leads to the following definition of $E[I(\mathbf{x}^*)]$.

$$E[I(\mathbf{x}^*)] = P[I(\mathbf{x}^*)]\sqrt{\left(\bar{Y}_1(x) - y_1^*(x^*)\right)^2 + \left(\bar{Y}_2(x) - y_2^*(x^*)\right)^2}, \quad (9.5)$$

where

$$\bar{Y}_1(x) = \left\{\begin{array}{l} \int_{-\infty}^{y_1^{*(1)}}\int_{-\infty}^{\infty} Y_1\phi(Y_1, Y_2)\,\mathrm{d}Y_2\,\mathrm{d}Y_1 \\ + \sum_{i=1}^{m-1}\int_{y_1^{*(i)}}^{y_1^{*(i+1)}}\int_{-\infty}^{y_2^{*(i+1)}} Y_1\phi(Y_1, Y_2)\,\mathrm{d}Y_2\,\mathrm{d}Y_1 \\ + \int_{y_1^{*(m)}}^{\infty}\int_{-\infty}^{y_2^{*(m)}} Y_1\phi(Y_1, Y_2)\,\mathrm{d}Y_2\,\mathrm{d}Y_1 \end{array}\right\} \Big/ P[I(\mathbf{x}^*)] \quad (9.6)$$

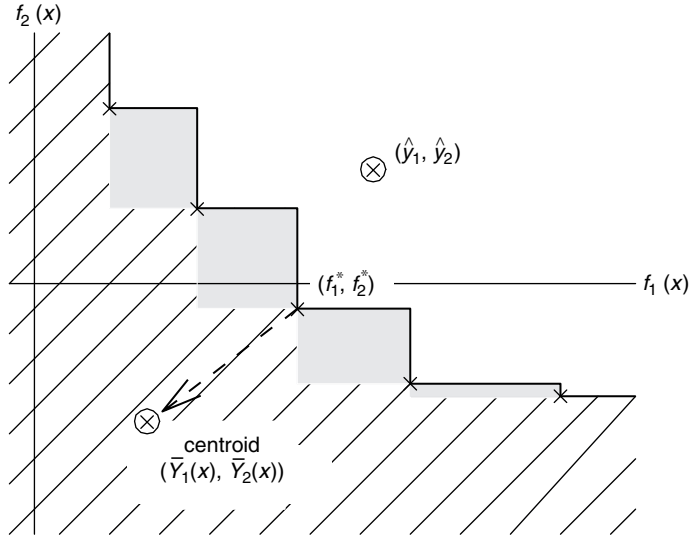



**Figure 9.3.** Centroid of the probability integral and moment arm used in calculating $E[I(\mathbf{x}^*)]$, also showing the predicted position of the currently postulated update.

and $\bar{Y}_2(x)$ is defined similarly. The integrals of Equation (9.6) are somewhat tedious, but may be carried out, by parts, to yield

$$\bar{Y}_1(x) = \left[ \begin{array}{l} \hat{y}_1(x)\Phi\left(\dfrac{y_1^{*(1)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right) - \hat{s}_1(x)\phi\left(\dfrac{y_1^{*(1)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right) \\ \\ + \sum_{i=1}^{m-1} \left\{ \begin{array}{l} \left[\hat{y}_1(x)\Phi\left(\dfrac{y_1^{*(i+1)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right) - \hat{s}_1(x)\phi\left(\dfrac{y_1^{*(i+1)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right)\right] \\ - \left[\hat{y}_1(x)\Phi\left(\dfrac{y_1^{*(i)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right) - \hat{s}_1(x)\phi\left(\dfrac{y_1^{*(i)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right)\right] \end{array} \right\} \\ \times \Phi\left(\dfrac{y_2^{*(i+1)} - \hat{y}_2(x)}{\hat{s}_2(x)}\right) \\ \\ + \left[\hat{y}_1(x)\Phi\left(\dfrac{y_1^{*(m)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right) - \hat{s}_1(x)\phi\left(\dfrac{y_1^{*(m)} - \hat{y}_1(x)}{\hat{s}_1(x)}\right)\right] \\ \\ \times \Phi\left(\dfrac{y_2^{*(m)} - \hat{y}_2(x)}{\hat{s}_2(x)}\right) \end{array} \right] \Big/ P[I(\mathbf{x}^*)]. \tag{9.7}$$

When defined in these ways $E[I(\mathbf{x}^*)]$ varies with the location of the predicted position of the currently postulated update $(\widehat{y}_1, \widehat{y}_2)$, also shown in Figure 9.3, and also with the estimated errors in this prediction, $\widehat{s}_1$ and $\widehat{s}_2$, since it is these quantities that define the probability density function being integrated.

The further the predicted update location lies below and to the left of the current Pareto front the further the centroid will lie from the front. Moreover, the further the prediction lies in this direction the closer the integral becomes to unity (since the greater the probability of the update offering an improvement). Both tendencies will drive updates to be improved with regard to the design objectives. Note that if there is a significant gap in the points forming the existing Pareto front, then centroidal positions lying in or near such a gap will score proportionately higher values of $E[I(\mathbf{x}^*)]$, since the Euclidean distances to the nearest point will then be greater. This pressure will tend to encourage an even spacing in the front as it is updated. Also, when the data points used to construct the Gaussian process model (i.e. all points available and not just those in the Pareto set) are widely spaced, the error terms will be larger and this tends to further increase exploration. Thus, this $E[I(\mathbf{x}^*)]$ definition balances exploration and exploitation in just the same way as its one-dimensional equivalent in Section 3.2.3.

When calculating the location of the centroid there is still no requirement to scale the objectives being studied but, when deciding which member of the current Pareto set lies closest to the centroid, relative scaling will be important (i.e. when calculating the Euclidean distance). This is an unavoidable and difficult issue that arises whenever explicitly attempting to space out points along the Pareto front, whatever method is used to do this.

Before moving on to study examples making use of these metrics, it is worth noting that there is no fundamental difficulty in extending this form of analysis to problems with more than two goal functions. This does, of course, increase the dimensionality of the Pareto surfaces being dealt with, and so inevitably complicates further the expressions needed to calculate the improvement metrics. Nonetheless, they always remain expressible in closed form, it always being possible to define the metrics in terms of summations over known integrable functions.

## 9.3 Design of the Nowacki Cantilever Beam Using Multi-objective, Constrained Expected Improvement

The first multi-objective problem considered here is a variant of the classic Nowacki beam problem (Nowacki, 1980). In this problem the aim is to design a tip loaded encastre cantilever beam for minimum cross-sectional area and lowest bending stress subject to a number of constraints (see the Appendix, Section A.4). To tackle this problem we will use the function `constrainedmultiei.m` (below), which calls `multiei.m` in the same manner as `constrainedei.m` calls `predictor.m` (see Chapter 5):

```
function NegLogConExpImp=constrainedmultiei(x)
% Calculates the negative of the log of the
% constrained multi-objective expected improvement at x
%
% Inputs:
%       x – 1 x k vetor of design variables
%
```

(*continued*)

```
% Global variables used:
%   ObjeciveInfo – structured cell array
%   ModelInfo – structure
%   ConstraintInfo – structured cell array
%
% Outputs:
%   NegLogConExpImp – scalar -log(E[I(x*)]P[F(x*)>gmin])
%
% Calls:
%   multiei.m, predictor.m

global ModelInfo
global ConstraintInfo

% Calculate unconstrained E[I(x*)]
ModelInfo.Option='NegLogExpImp';
NegLogExpImp=multiei(x);

% Calculate P[F(x*)] for each constraint
for i=1:size(ConstraintInfo,2)
    ModelInfo=ConstraintInfo{i};
    ModelInfo.Option='NegProbImp';
    NegProbImp(i)=predictor(x);
end

% Calculate E[I(x*)]P[F(x*)] (add 1e50 before taking logs)
NegLogConExpImp=-(-NegLogExpImp+sum(log10(-NegProbImp+1e-50)));
```

The following *MATLAB* code begins from a 10-point sample and adds 40 infill points based on the (constrained) $E[I(\mathbf{x}^*)]$ criterion, $E[I(\mathbf{x}^*) \cap F(\mathbf{x}^*)]$. Note that we use rather more infill points than in single objective optimization because, instead of looking for a single optimal design, we are searching for a host of optimal trade-offs. The functions which calculate the constraints are formulated such that a positive value is returned when the constraint is violated.

We begin by defining the beam properties, building a sampling plan and calculating the objective function and constraint functions at the sample points.

```
global ModelInfo
global ObjectiveInfo
global ConstraintInfo

% Set up beam properties
global BeamProperties
BeamProperties.F=5e3;
BeamProperties.L=1.5;
```

(*continued*)

```
BeamProperties.SigmaY=240e6;
BeamProperties.E=216.62e9;
BeamProperties.G=86.65e9;
BeamProperties.Nu=0.27;
BeamProperties.SF=2;

% Create sampling plan
n=10;
k=2;
% Put into ObjectiveInfo
ObjectiveInfo{1}.X=bestlh(n,k,20,10)
ObjectiveInfo{2}.X=ObjectiveInfo{1}.X;

% ... and ConstraintInfo
nConstraints=5;
for i=1:nConstraints
    ConstraintInfo{i}.ConstraintLimit=0;
    ConstraintInfo{i}.X=ObjectiveInfo{1}.X;
end

% Calculate observed data
for i=1:n
    ObjectiveInfo{1}.y(i,1)=area(ObjectiveInfo{1}.X(i,:));
    ObjectiveInfo{2}.y(i,1)=bending(ObjectiveInfo{2}.X(i,:));
    ConstraintInfo{1}.y(i,1)=arearatioconstraint...
                    (ConstraintInfo{1}.X(i,:));
    ConstraintInfo{2}.y(i,1)=bendingconstraint...
                    (ConstraintInfo{2}.X(i,:));
    ConstraintInfo{3}.y(i,1)=bucklingconstraint...
                    (ConstraintInfo{3}.X(i,:));
    ConstraintInfo{4}.y(i,1)=deflectionconstraint...
                    (ConstraintInfo{4}. X(i,:));
    ConstraintInfo{5}.y(i,1)=shearconstraint...
                    (ConstraintInfo{5}.X(i,:));
end
```

Based on this observed data, we perform an infill strategy. This involves optimizing the parameters of the Kriging models of the objective and constraint functions, followed by a genetic algorithm search of `constrainedmultiei.m`. The result of this search is then added to the observed data set, before repeating the process. At each stage we also plot the current set of non-dominated solutions.

```
figure
% Iterate over 40 infill points
for I=1:40
    % Tune Kriging models of objectives
    options=gaoptimset('PopulationSize',20,'Generations',10);
```

(*continued*)

```
for i=1:2
    ModelInfo=ObjectiveInfo{i};
    ObjectiveInfo{i}.Theta=ga(@likelihood,k,[],[],[],...
    [],ones(k,1).*-3,ones(k,1).*3,[],options);
    [NegLnLike,ObjectiveInfo{i}.Psi,ObjectiveInfo{i}.U]=...
    likelihood(ObjectiveInfo{i}.Theta);
end

% Tune Kriging models of constraints
for i=1:nConstraints
    ModelInfo=ConstraintInfo{i};
    ConstraintInfo{i}.Theta=ga(@likelihood,k,[],[],[],...
    [],ones(k,1).*-3,ones(k,1).*3,[],options);
    [NegLnLike,ConstraintInfo{i}.Psi,ConstraintInfo{i}.U]=...
    likelihood(ConstraintInfo{i}.Theta);
end

% Find points which satisfy constraints
y1temp=ObjectiveInfo{1}.y;
y2temp=ObjectiveInfo{2}.y;
Xtemp=ObjectiveInfo{2}.X;

for i=1:length(y1temp)
    for j=1:nConstraints
        if ConstraintInfo{j}.y(i)>ConstraintInfo{j}.
            ConstraintLimit
           y1temp(i)=nan;
           y2temp(i)=nan;
        end
    end
end

Xtemp=Xtemp (find (~ isnan(y2temp)),:);
y1temp=y1temp(find(~isnan(y1temp)));
y2temp=y2temp(find(~isnan(y2temp)));

% Find Pareto set
clear PX Py1 Py2
% ... first sort according to objective 1
[a,b]=sort(y1temp);

% ... yields first non-dominated point
PX(1,1:k)=Xtemp(b(1),1:k);
Py1(1)=y1temp(b(1));
Py2(1)=y2temp(b(1));

% ... then cycle through remaining sorted list
Pnum=1;
for i=2:length(y1temp)
```

(*continued*)

```
        % ... and look for better values of objective 2
        if y2temp(b(i))<=Py2(end)
            Pnum=Pnum+1;
            PX(Pnum,1:k)=Xtemp(b(i),1:k);
            Py1(Pnum)=y1temp(b(i));
            Py2(Pnum)=y2temp(b(i));
        end
    end

    % Plot Pareto front so far
    plot(Py1,Py2,'ko')
    title('Tradeoff')
    xlabel('A␣(m)')
    ylabel('\sigma_B␣(Pa)')
    axis square
    drawnow

    % Search constrained multi-objective E[I(x)]
    options=gaoptimset('PopulationSize',50,'Generations',20);
    [VarOpt,EIOpt]= ga(@constrainedmultiei,k,[],[],[],[],...
    zeros(k,1),ones(k,1),[],options)

    % Add infill point
    ObjectiveInfo{1}.X(end+1,:)=VarOpt;
    ObjectiveInfo{2}.X=ObjectiveInfo{1}.X;

    for i=1:nConstraints
      ConstraintInfo{i}.X=ObjectiveInfo{1}.X;
    end

    % Calculate observations at infill point
    ObjectiveInfo{1}.y(end+1)=area...
    (ObjectiveInfo{1}.X(end,:));
    ObjectiveInfo{2}.y(end+1,1)=bending...
    (ObjectiveInfo{2}.X(end,:));
    ConstraintInfo{1}.y(end+1,1)=arearatioconstraint...
    (ConstraintInfo{1}.X(end,:));
    ConstraintInfo{2}.y(end+1,1)=bendingconstraint...
    (ConstraintInfo{2}.X(end,:));
    ConstraintInfo{3}.y(end+1,1)=bucklingconstraint...
    (ConstraintInfo{3}.X(end,:));
    ConstraintInfo{4}.y(end+1,1)=deflectionconstraint...
    (ConstraintInfo{4}.X(end,:));
    ConstraintInfo{5}.y(end+1,1)=shearconstraint...
    (ConstraintInfo{5}.X(end,:));
end
```

Figure 9.4 shows the 9 non-dominated solutions found using this infill strategy. The true Pareto front, found from a $101 \times 101$ grid of points, is also shown. It is clear that a very

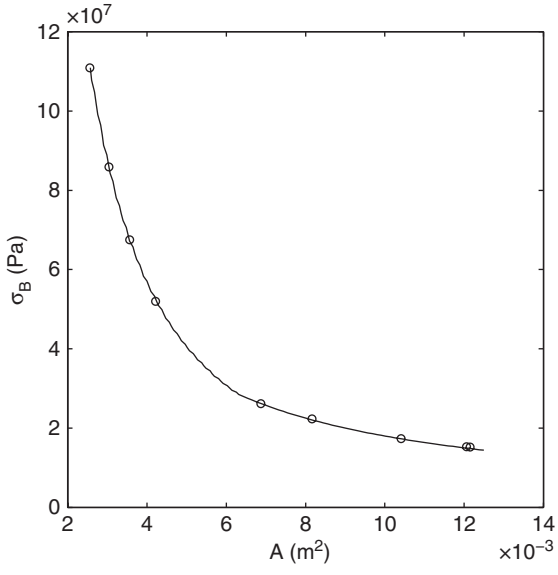



**Figure 9.4.** The 9 non-dominated solutions found using the $\max\{E[I(\mathbf{x}^*) \cap F(\mathbf{x}^*)]\}$ infill strategy (circles) and the true Pareto front (line).

good indication of the optimal trade-off has been found, with all the points lying on the Pareto front.

## 9.4 Design of a Helical Compression Spring Using Multi-objective, Constrained Expected Improvement

The second example is the design of a helical spring, the formulation of which is described in the Appendix, Section A.5. We have already considered maximizing the number of cycles to fatigue failure in Chapter 5. Here we add the second objective – that of minimizing the mass of the spring. The problem is solved in a similar way to the Nowacki beam problem and we leave it to the reader to modify the code used for that problem. There is a subtle difference in the code required to solve this problem, which is that we must handle the `NaN`s which are occasionally produced by the `springcycles.m` function. We simply delete these designs from both `ObjectiveInfo{i}` and `ConstraintInfo{i}`. To produce a set of non-dominated designs, the objective function values of which are shown in Figure 9.5, we have used a 15-point maximin sampling plan augmented by 60 infill points at the maximum $E[I(\mathbf{x}^*) \cap F(\mathbf{x}^*)]$. This is a more difficult problem than the Nowacki beam, with an extra design variable and a smaller area of feasible designs. Although the non-dominated points are not as close to the true Pareto front as in Figure 9.4, a good representation of the location of the optimal trade-off has been found.

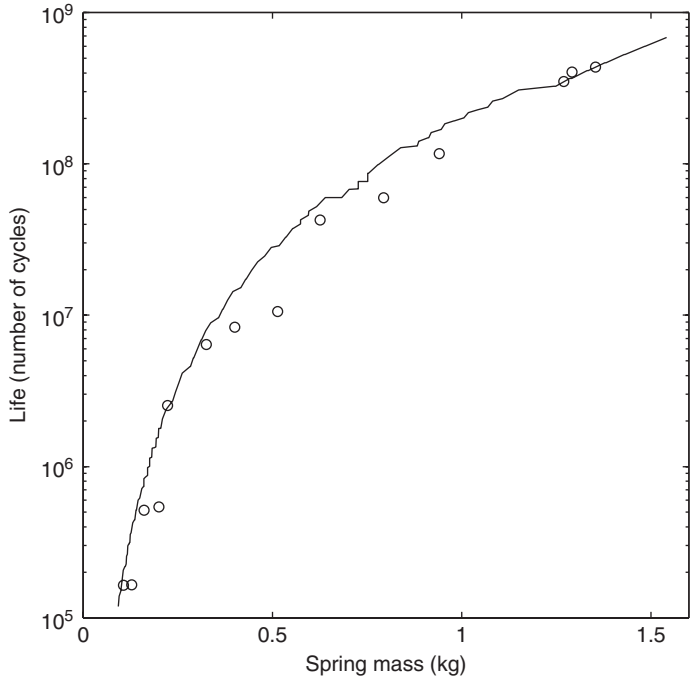



**Figure 9.5.** The 14 non-dominated designs found using a $\max\{E[I(\mathbf{x}^*) \cap F(\mathbf{x}^*)]\}$ search (circles), shown alongside the true Pareto front (derived from the data in Figure A.9).

## 9.5 Summary

The objective and constraint functions used in the above examples could be modelled effectively using more simple surrogate methods, exploited within a multi-objective search routine. However, for more complex objective and constraint functions, the $E[I(\mathbf{x}^*)]$ criterion can offer significant improvements over exploitation based infill criteria, similar to the advantages of the single objective $E[I(\mathbf{x})]$ over pure exploitation (recall Section 3.2). Compared with directly searching the objective and constraint functions using a multi-objective search algorithm, the possible benefits are huge, requiring tens, rather than thousands, of calls to the objective and constraint functions. While in the example presented in this chapter the cost of calls to the objective and constraint functions is negligible, if each experiment were a lengthy computer simulation, or even a destructive test, the potential time and cost savings are significant.

## References

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002) A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2), 182–197, April.

Fonseca, C. M. and Fleming, P. J. (1995) An overview of evolutionary algorithms in mutliobjective optimization. *IEEE Transactions on Evolutionary Computation*, **3**(1), 1–16.

Keane, A. J. and Nair, P. B. (2005) *Computational Approaches for Aerospace Design: The Pursuit of Excellence*, John Wiley & Sons, Inc., New York.

Knowles, J. and Hughes, E. J. (2005) Multi-objective optimization on a budget of 250 evaluations, in *Evolutionary Multi-criterion Optimization (EMO-2005)* (eds C. Coello *et al.*), Volume 3410 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin.

Nowacki, H. (1980) Modelling of design decisions for CAD, in *CAD Modelling, Systems Engineering, CAD-Systems*, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Voutchkov, I. I. and Keane, A. J. (2006). "Multi-objective optimization using surrogates", in Proc. 7th Int. Conf. Adaptive Computing in Design and Manufacture, (ACDM 2006, ISBN 0-9552885-0-9), Bristol, pp. 167–175.

Wilson, B., Cappelleri, D., Simpson, W. and Frecker, M. (2001) Efficient Pareto frontier exploration using surrogate approximations. *Optimization and Engineering*, **2**, 31–50.

# Appendix: Example Problems

## A.1 One-Variable Test Function

Our first example problem is the one-variable function

$$f(x) = (6x-2)^2 \sin(12x-4),\ x \in [0, 0.5] \text{ or } x \in [0, 1], \tag{A.1}$$

calculated by `onevar.m`. Using the term 'test function' we are referring to a contrived function, with no physical meaning, which is useful in demonstrating optimization methodologies. This function is used to represent a multimodal objective function landscape, i.e. one where a search routine could become trapped in a local minima. Figure A.1 shows the function across the full zero to one range, and it can be seen that there is one global minimum, one local minimum and a region containing a zero gradient point of inflexion. We make extensive use of this function in Section 3.2 to assess the ability of surrogate infill strategies to find the global optimum.

To demonstrate the noise filtering capabilities of surrogate models we also use this function with the addition of a normally distributed 'noise' component, using the *MATLAB* expression `randn(n,1)`. A true engineering function that exhibits 'noise' is described in Section A.3.

We have also constructed a multi-fidelity form of this function (`cheaponevar.m`):

$$f_c(x) = Af_e + B(x-0.5) - C, \tag{A.2}$$

where $f_e$ is calculated from Equation A.1, which is considered to be a less accurate, 'cheap' form of the more accurate, 'expensive' original function. By varying $A$, $B$ and $C$, we can make the cheap function a better or worse approximation of the expensive function. We use this formulation to demonstrate the multi-fidelity method of co-Kriging described in Chapter 8.

*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane
 ISBN: 978-0-470-06068-1

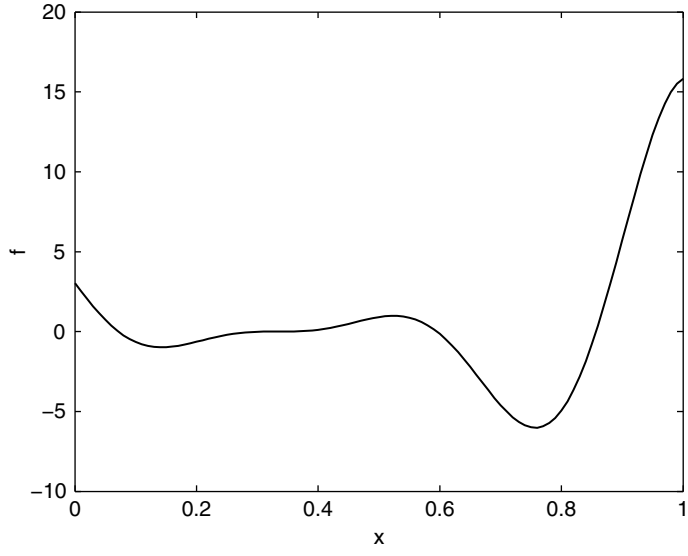



**Figure A.1.** The one-variable function: $f(x) = (6x-2)^2 \sin(12x-4)$.

## A.2 Branin Test Function

Our second function is a modified version of the two-variable Branin function:

$$f(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left[\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 1\right] + 5x_1, x_1 \in [-5, 10], x_2 \in [0, 15], \tag{A.3}$$

The final term is a modification to the traditional Branin function, and means that there are two local optima and one global optimum, rather than three global optima of equal value. We feel this modification makes the function more representative of engineering functions. The *MATLAB* function `branin.m` accepts inputs in the range $[0, 1]$ and scales these to the ranges required by Equation (A.3). Figure A.2 shows the contours of the Branin function. The global optimum is towards the upper left corner of the plot.

To demonstrate the constrained surrogate methods in Chapter 5, we apply the condition that

$$g(x) = x_1 x_2 > 0.2, x_1, x_2 \in [0, 1]. \tag{A.4}$$

This means that only the area of the Branin function shown in Figure A.3 is considered. The optimum is now towards the lower right corner.

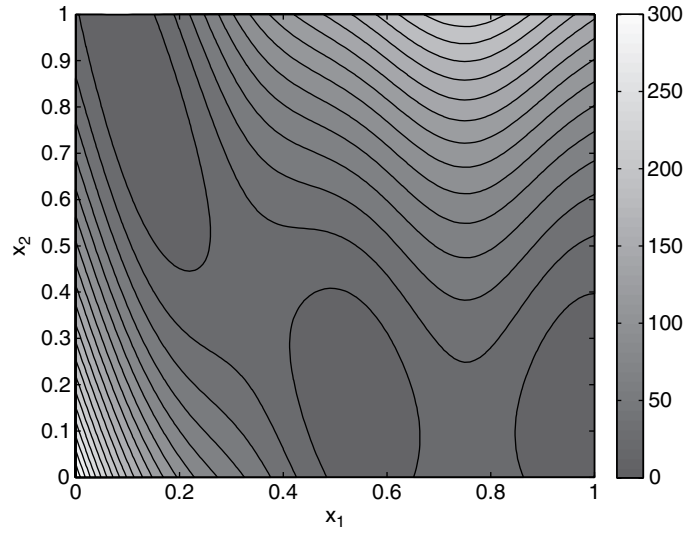



**Figure A.2.** The modified Branin function.

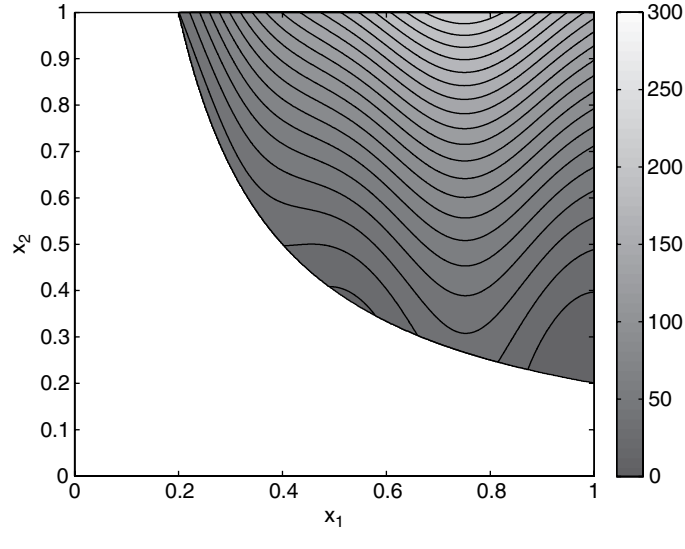



**Figure A.3.** The constrained modified Branin function.

## A.3 Aerofoil Design

Computational fluid dynamics simulations are notorious for producing 'noisy' data due to the discretization error arising from solving the governing equations on a finite mesh. The aerofoil drag coefficients returned by the function `aerofoilcd.m` exhibit just such noise.

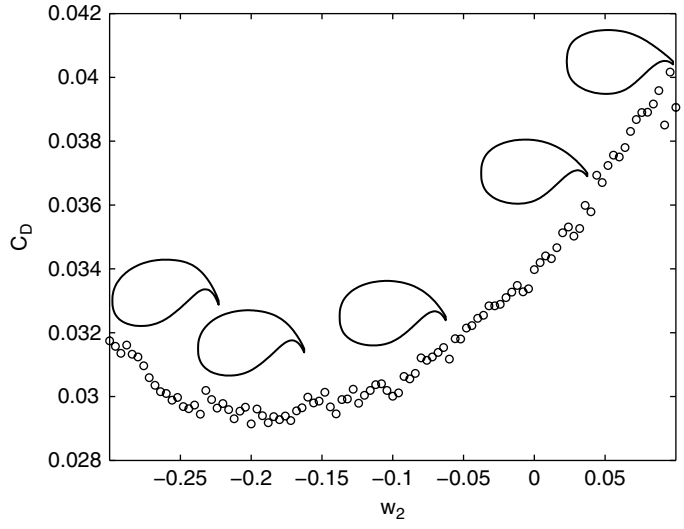



**Figure A.4.** The 101 $C_D$ values, for a fixed $C_L = 0.6$, as the aerofoil camber parameter changes. The thickness to chord ratio has been increased to help show the changing camber.

The drag coefficient, $C_D$, values correspond to an Euler simulation of a transonic aerofoil at a fixed lift coefficient, $C_L = 0.6$. A rather coarse mesh has been used to accentuate the discretization error. The function accepts a normalized input variable $x \in [0, 1]$ which determines the value of a shape parameter, $w_2 \in [-0.3, 0.1]$, which affects the camber towards the rear of the aerofoil (see the aerofoils in Figure A.4). For more information on the way the geometry is parameterized, see Robinson and Keane (2001).

The function does not actually run a CFD simulation, rather it returns one of 101 discrete $C_D$ values, with $x$ rounded to the nearest 0.01. Minimizing $C_D$ in this one-variable design space seems like a trivial task. However, it is surprisingly difficult to find the precise location of the optimum, as shown when we tackled this problem in Chapter 6.

## A.4 The Nowacki Beam

Based on the design problem described by Nowacki (1980), the aim is to design a tip-loaded encastre cantilever beam for minimum cross-sectional area and lowest bending stress subject to a number of constraints. The beam length $l = 1.5\,\text{m}$ and is subject to a tip load $F = 5\,\text{kN}$. The beam is taken to be rectangular in section, with breadth $b$ and height $h$ giving a cross-sectional area $A$ (see Figure A.5).

The design is subject to the following criteria:

1. a maximum tip deflection, $\delta = Fl^3/(3EI_Y)$, of 5 mm, where $I_Y = bh^3/12$;
2. a maximum allowable direct (bending) stress, $\sigma_B = 6Fl/(bh^2)$, equal to the yield stress of the material, $\sigma_Y$;
3. a maximum allowable shear stress, $\tau = 3F/(2bh)$, equal to one half the yield stress of the material;

4. a maximum height to breadth ratio, $h/b$, for the cross-section of 10;
5. the failure force for twist buckling, $F_{\text{CRIT}} = (4/l^2)\sqrt{GI_T EI_Z/(1-\nu^2)}$, to be greater than the tip force multiplied by a safety factor, $f$, of two, where $I_T = (b^3h + bh^3)/12$ and $I_Z = b^3h/12$.

The material used is mild steel with a yield stress of $\sigma_Y = 240\,\text{MPa}$, Young's modulus $E = 216.62\,\text{GPa}$, $\nu = 0.27$ and shear modulus calculated as $G = 86.65\,\text{GPa}$.

We wish to minimize the cross-sectional area (i.e. the cost) and bending stress by varying the height ($20\,\text{mm} > b > 250\,\text{mm}$) and breadth ($10\,\text{mm} > b > 50\,\text{mm}$). Notice that it is not clear from the above specification which of the design limits will control the design, although clearly at least one will, if the beam is to have a nonzero cross-sectional area.

The two objectives and five constraints are calculated by `area.m`, `bending.m`, `arearatioconstraint.m`, `bendingconstraint.m`, `bucklingconstraint.m`, `deflectionconstraint.m` and `shearconstraint.m`. The functions accept $h \in [0, 1]$ and $b \in \{0, 1\}$ [scaling to the above limits is carried out within the functions]. The constraint functions return a value greater than zero in the event that the constraint is violated.

Figures A.6 and A.7 show how the two objectives vary with the normalized $h$ and $b$. Clearly there is to be a trade-off between minimum bending stress and minimum cross-sectional area solutions. To solve this problem, we wish to identify the Pareto front of optimal trade-offs, which is shown in Figure A.8. Interestingly the constraint on the bending stress marks one end of the resulting Pareto front, while setting both variables to their maximum values defines the other end of the front. Locating the minimum stress solution requires an exploration of the constraint boundaries, a characteristic common in engineering problems, but one that some search methods find difficult to deal with.

We attempt to solve this problem using constrained multi-objective expected improvement in Section 9.3.

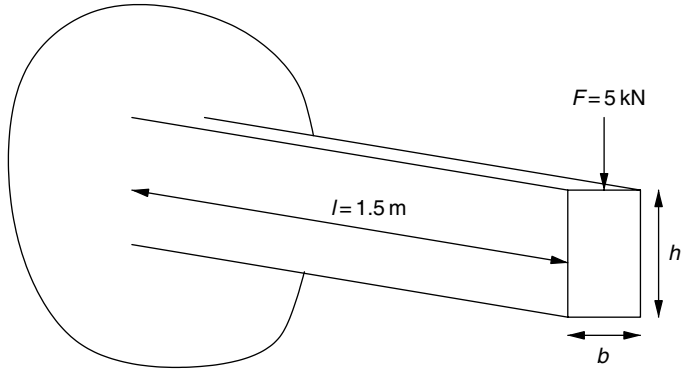



**Figure A.5.** Sketch of the Nowacki beam problem.

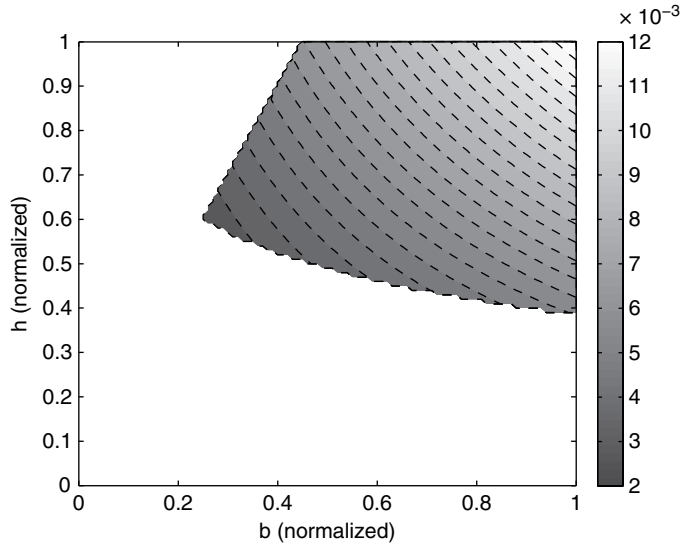



**Figure A.6.** Beam cross-sectional area, $A$, for varying $h$ and $b$.

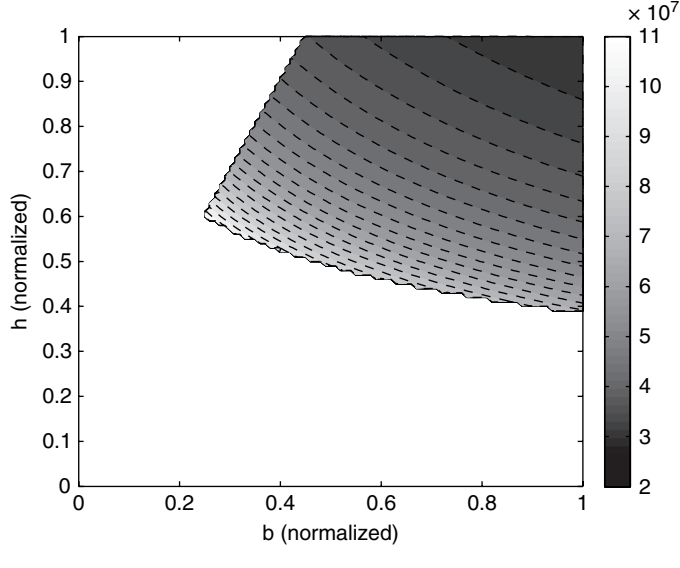



**Figure A.7.** Beam bending stress, $\sigma_B$, for varying $h$ and $b$.

## A.5 Multi-objective, Constrained Optimal Design of a Helical Compression Spring

Let us consider the following problem (Tudose and Jucan, 2007). A helical compression spring is to be designed to work over a stroke of $h = 50\,\text{mm}$ with a corresponding load

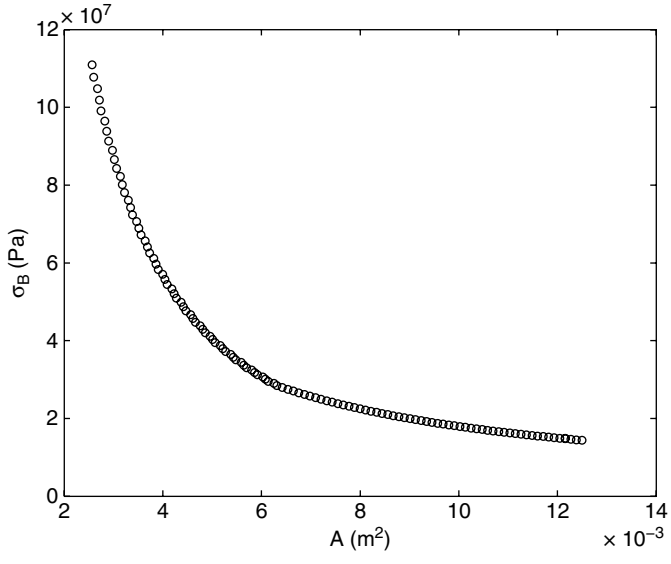



**Figure A.8.** The Pareto front for the Nowacki beam problem. This curve represents the optimal trade-off between $A$ and $\sigma_B$.

variation between $F_{min} = 40\,N$ and $F_{max} = 500\,N$. ASTM A229/SAE J315 oil tempered wire is to be used with a Young's modulus of $E = 2.06 \times 10^5$ MPa, a density of $\rho = 7.87 \times 10^{-6}$kg/mm$^3$ and a rigidity modulus of $G = 0.78 \times 10^5$ MPa.

There are two goals to be optimized here. Firstly, the mass of the spring is to be minimized and this rather simple first objective is computed using the function `springmass.m`. The second, competing, objective is that we wish to maximize the fatigue life of the spring. The number of cycles until fatigue failure are computed by the function `springcycles.m`.

There are also two constraints. We have to make sure that the wire does not fail in shear, with a factor of safety of 1.05 – the function `shearsafetyfact.m` computes this (a negative value returned by the function means that the spring satisfies the constraint). We also have to check that the spring does not buckle within its working range – a negative value returned by `buckling.m` for a given design means that it does not.

As with any other design optimization problem, the parameterization of the design is a crucial element of the process. Here we use three design variables. The first is the wire diameter $d$ taking values between 0.5 mm and 7 mm. The second is the index $i$ of the spring, defined as the ratio of the mean diameter of the spring (the diameter of the helix, measured in the centre of the wire) and $d$. We allow this to vary between 4 and 16. Finally, $k_\Delta$, the maximum load intercoil distance coefficient (the ratio of the distance between adjacent coils of the fully loaded spring and the wire diameter)-this can vary between 0.1 and 1.1.

All four *MATLAB* functions cited above therefore take a single vector `design` as the input, consisting of the normalized values of *d, i* and $k_\Delta$ (the actual values are calculated within each function, based on the ranges specified above).

While elsewhere in the book these functions (the two objectives and the two constraints) have been used as 'pretend-expensive' testcases, they are, in fact, very cheap to compute,

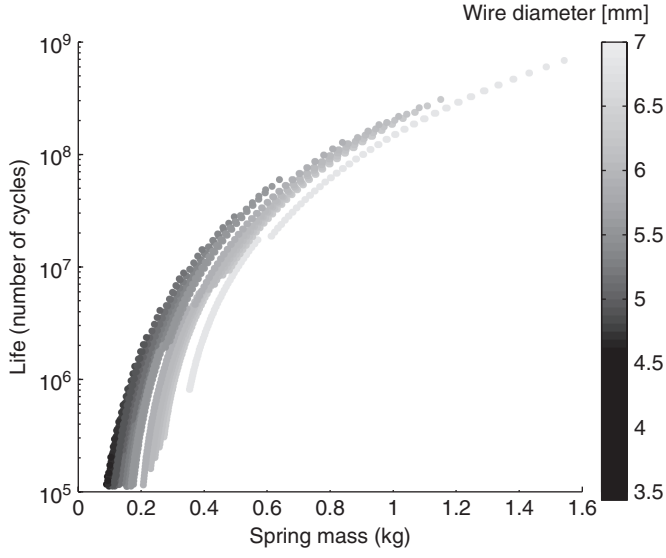



**Figure A.9.** The 97 099 feasible designs, representing around 15.2 % of a 640 000 point full factorial sampling plan, covering the three-dimensional design space. The requirement for light, long life springs means that the north-western boundary of this cloud of points represents the Pareto front.

as the reader can find out by running `springpareto.m`, a script that generates a $64 \times 100 \times 100$ point full factorial sampling plan and computes the objectives and constraints for each of the 640 000 designs. A scatter plot of the two objectives and the corresponding wire diameters for all of the feasible designs is shown in Figure A.9.

## A.6 Novel Passive Vibration Isolator Feasibility

The final engineering example is, as the title suggests, a little bit different. Figure A.10 shows an optimized prototype vibration isolator test structure. The top of the structure is passively isolated from vibrations at the base by exploiting reflections in vibration energy in the complex structure between the two. The structure takes the form of a triangular truss, which is 'folded' such that each section sits amongst the last. There are two optimization problems in designing the structure: maximizing the vibration isolation itself and searching for a structure that can actually be manufactured. We will consider the latter. The complex nature of the structure, with many interleaved aluminium rods, means that most designs contain intersections between rods. In fact, only 0.2 % of designs are feasible for the three-section structure in Figure A.10.

The variables that determine the design are the *x,y,z* position of all the intermediate joints. For the three-section structure in Figure A.10, $k = 18$. More complex structures, with hopefully better vibration isolation, can be built with $k = 36, 54, \ldots, 9(N_{\text{sections}} - 1)$, $N_{\text{sections}} = 2a + 1$, where $a \in \mathbb{N}$.

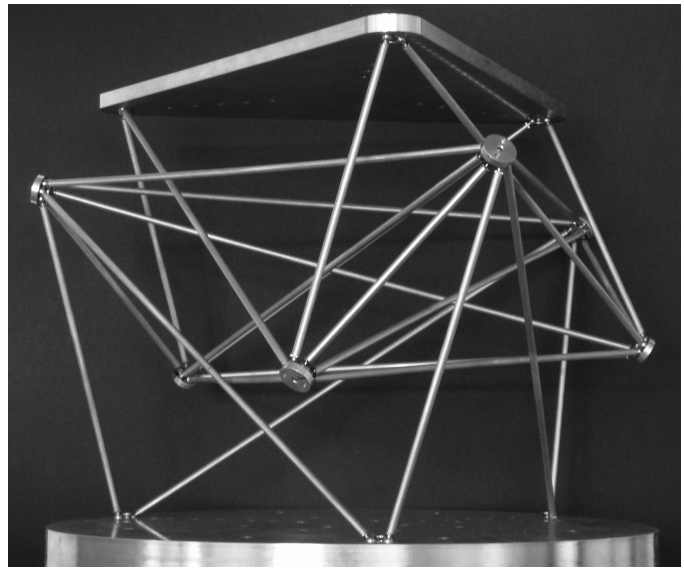


**Figure A.10.** An optimized passive vibration isolator test structure.

A feasible design is one with no intersections. The distance between each rod pair is calculated using the line-to-line distance formula

$$D = \frac{|\mathbf{c} \cdot (\mathbf{a} \times \mathbf{b})|}{|\mathbf{a} \times \mathbf{b}|}, \tag{A.5}$$

where $\mathbf{a} = \mathbf{x}_2 - \mathbf{x}_1$, $\mathbf{b} = \mathbf{x}_4 - \mathbf{x}_3$ and $\mathbf{c} = \mathbf{x}_3 - \mathbf{x}_1$, and $\mathbf{x}_1$, $\mathbf{x}_2$ are the Cartesian coordinates of the ends of the first beam and $\mathbf{x}_3$, $\mathbf{x}_4$ the ends of the other beam in the pair. The rods are 5 mm in diameter and we want a gap of at least 2.5 mm. To calculate the objective function, we therefore sum $D < 7.5$ mm over all beam pairs, and the structure is feasible if $D = 0$. The function `intersections.m` takes a $k$-vector of variables in the range [0,1] and outputs the feasibility objective. Setting all the variables at 0.5 describes a uniform structure 0.3 m high, with the intermediate joints sitting 0.01 m inside the top and base, and an equilateral plan-view with 0.3 m sides. The range [0,1] allows the intermediate points to vary inside 0.25 m × 0.25 m × 0.25 m cubes. The geometry concept is described in more detail by Forrester and Keane (2007).

We use this problem to demonstrate the *goal seeking* infill criterion in Section 3.3.4.

## References

Forrester, A. I. J. and Keane, A. J. (2007) Multi-variable geometry repair and optimization of passive vibration isolators, in 3rd AIAA Multidisciplinary Design Optimization Specialist Conference, Hawaii.

Nowacki, H. (1980) Modelling of design decisions for CAD, in *CAD Modelling, Systems Engineering, CAD-Systems*, Lecture Notes in Computer Science, Springer-Verlag, Berlin.

Robinson, G. M. and Keane, A. J. (2001) Concise orthogonal representation of supercritical aerofoils. *Journal of Aircraft*, **38**(3), 580–583.

Tudose, L. and Jucan, D. (2007) Pareto approach in multi-objective optimal design of helical compression springs. *Annals of the Oradea University, Fascicle of Management and Technological Engineering*, **6**(16).

# Index




*Engineering Design via Surrogate Modelling: A Practical Guide* A. I. J. Forrester, A. Sóbester and A. J. Keane



ISBN: 978-0-470-06068-1