An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design

A Dissertation
Presented to
the Academic Faculty

By

Yao Lin

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Mechanical Engineering

Georgia Institute of Technology
July 2004

An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design

Approved:

Farrokh Mistree, Chair
Professor, Mechanical Engineering

Janet K. Allen
Senior Research Scientist
Mechanical Engineering

Kwok-Leung Tsui
Professor
Industrial and Systems Engineering

Victoria Chen
Associate Professor
Industrial and Manufacturing Systems Engineering
The University of Texas at Arlington

David W. Rosen
Associate Professor
Mechanical Engineering

Wenjing Ye
Assistant Professor
Mechanical Engineering

Date Approved: July 29, 2004

ACKNOWLEDGMENTS

There are many I would like to acknowledge for contributing, in one form or another, to the research in this dissertation. First and foremost, I would like to thank my committee members for their comments and suggestions. Specially, I sincerely thank my advisors Dr. Farrokh Mistree and Dr. Janet K. Allen for their extraordinary guidance, inspiration, and support, which helped me grow both intellectually and personally. I also appreciate the valuable expertise, insight, support, and recommendations that Dr. Kwok-Leung Tsui offered for my dissertation. I would like to express my special thanks to Dr. Victoria Chen, who has given me a lot of insightful thoughts and valuable instructions on almost all aspects of the studies in this dissertation. I am also very grateful to Dr. David Rosen and Dr. Wenjing Ye for serving as my dissertation reading committee members; their insights and advice from viewpoints of Mechanical Engineering are very useful for my research in this dissertation. I would like to thank Dr. Tim Simpson, whose kriging computer codes are used to develop kriging metamodels in this dissertation.

I would like to express my gratitude to my colleagues in the Schools of Mechanical Engineering and Industrial and Systems Engineering at Georgia Institute of Technology. First I would like to thank my co-researcher Terrence Murphy in School of Industrial and Systems Engineering, who listened to me, understood my ideas, and provided insightful feedbacks. Thanks also go to Carolyn Conner Seepersad, Haejin Choi and Marco Fernandez, who helped me a lot with the LCA design example in

Chapter 7. I am grateful to Carolyn Conner Seepersad and Jitesh Panchal for their services on iSIGHT, ModelCenter, and other computer software used in this dissertation. I would like to thank Dr. Angran Xiao and Hongqing Wang for their help in trouble-shooting of technical problems with CAD/CAE software. Also, special thanks go to Chris Williams, Matt Chamberlain, Scott Duncan, Andrew Schnell, Benay Sager, John Reap, and all other students who have been or are currently in the Systems Realization Laboratory, for helping build a pleasant and inspiring environment in which my research is carried.

I owe great thanks to my family for their continual support and encouragement. Thanks to my parents for not only giving me the chance to come to the world, but also teaching me how to face and enjoy things in the world. I am very grateful to my Godmother. She is my instructor in the life and will always be the person I respect most. Thanks to my little sister and brother for so many happy memories. Special thanks go to my girlfriend, Min Jin, who is caring and supportive; her love has always been an encouragement to me. Also I would like to thank all my friends that helped me and spent good times with me in the past years.

Finally, I would like to offer deep thanks to the National Science Foundation for financial support under grants DMI-96-12365 and DMI-01-0100123 during my Master's and Ph.D. studies. I also appreciate the support from ONR Contract N00014-01-1-0267. The Systems Realization Laboratory has underwritten the cost of computer time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SUMMARY

Experimentation and approximation are essential for efficiency and effectiveness in concurrent engineering analyses of large-scale complex systems. The approximation-based design strategy is not fully utilized in industrial applications in which designers have to deal with multi-disciplinary, multi-variable, multi-response, and multi-objective analysis using very complicated and expensive-to-run computer analysis codes or physical experiments. With current experimental design and metamodeling techniques, it is difficult for engineers to develop acceptable metamodels for irregular responses and achieve good design solutions in large design spaces at low prices. To circumvent this problem, engineers tend to either adopt low-fidelity simulations or models with which important response properties may be lost, or restrict the study to very small design spaces. Information from expensive physical or computer experiments is often used as a validation in late design stages instead of analysis tools that are used in early-stage design. This increases the possibility of expensive re-design processes and the time-to-market.

In this dissertation, two methods, the Sequential Exploratory Experimental Design (SEED) and the Efficient Robust Concept Exploration Method (E-RCEM) are developed to address these problems. The SEED and E-RCEM methods help develop acceptable metamodels for irregular responses with expensive experiments and achieve satisficing design solutions in large design spaces with limited computational or

monetary resources. It is verified that more accurate metamodels are developed and better design solutions are achieved with SEED and E-RCEM than with traditional approximation-based design methods. SEED and E-RCEM facilitate the full utility of the simulation-and-approximation-based design strategy in engineering and scientific applications.

Several preliminary approaches for metamodel validation with additional validation points are proposed in this dissertation, after verifying that the most-widely-used method of leave-one-out cross-validation is theoretically inappropriate in testing the accuracy of metamodels. A comparison of the performance of kriging and MARS metamodels is done in this dissertation. Then a sequential metamodeling approach is proposed to utilize different types of metamodels along the design timeline.

Several single-variable or two-variable examples and two engineering example, the design of pressure vessels and the design of unit cells for linear cellular alloys, are used in this dissertation to facilitate our studies.

CHAPTER 1

FOUNDATIONS FOR SEQUENTIAL METAMODELING AND SEQUENTIAL DESIGN SPACE EXPLORATION

Experimentation and approximation are essential for efficiency and effectiveness in concurrent engineering analyses of large-scale complex systems in which designers have to deal with multi-disciplinary and multi-objective analysis using very complicated and expensive-to-run computer analysis codes. This process of experimentation and approximation is called metamodeling in which we need: (a) choosing an experimental design for generating data, (b) choosing a model to represent the data, and (c) fitting the model to the data. Sequential metamodeling and analyses are the development of series of metamodels with different sets of data by realizing these steps sequentially and repeatedly along the design timeline. It helps designers explore the design space to find satisfacing solutions in early design stages.

The heart of the chapter lies in Section 1.3 wherein the research objectives, hypotheses, and contributions for the work are described. Sections 1.1 and 1.2 contain the motivation, foundation, and references for investigating the proposed research and serve to establish context for the reader. The validation and verification strategy for this dissertation is presented in Section 1.4. Finally, Section 1.5 contains an overview of the dissertation.

1.1 MOTIVATION AND BACKGROUND

In the design of large-scale engineering systems, it is initially desirable to explore a large design space. Much of today's engineering analysis work consists of running complex computer programs – supplying a vector of design variables (inputs) $\boldsymbol{x}$ and receiving a vector of response (outputs) $\boldsymbol{y}$. However, a complete examination of a large design space can generate a substantial computational load. Many detailed analysis programs are available in the later stages of design, but they are often too expensive to use in exploring large design spaces. Furthermore, this mode of query-and-response often leads to a trial and error approach to design, an iterative spiral compounded by the requirements flowdown and feedback necessary in large-scale complex systems design. Thus a designer may never uncover the functional relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$ and never identify the best settings for the input values. To solve this problem, metamodels, i.e., "model of the model" (Kleijnen, 1987), are created to predict system performance at various sample points and then to develop a relationship between predicted performance and the variables under study. This process of experimentation and approximation consists: (a) choosing an experimental design for generating data, (b) choosing a model to represent the data, and (c) fitting the model to the data. Notice that the predicted performance is determined by the input variables and hence is deterministic and not based on random variation. Design decisions are made based on the results of the analysis of the metamodels. By using metamodels in design we sacrifice some

acceptable degrees of design "effectiveness" (accuracy) to gain more design "efficiency" (speed), which is reasonable and highly recommended in the early design stages.

Metamodeling is very useful in robust design of open engineering systems. Robust design is based on experimentation and development & analysis of metamodels. Taguchi proposes exploring the design space at selected points determined by one type of Design of Experiments (DOE), namely, orthogonal arrays (see, e.g., Taguchi, 1987); actual design performance is analyzed at selected points and response surfaces are generated to predict performance over the entire design space. In the literature, both the method of sampling the design space with orthogonal arrays and the generation of response surfaces to smooth the data have been questioned. Many variations of robust design have been proposed with advanced metamodeling techniques. Development in metamodeling and robust design techniques helps the design of open engineering systems, e.g., product families, in early design stages (Simpson, et al., 1997a).

During the design processes, the design information increases exponentially along the design timeline. At different points along the design timeline the design requirements are different; designers' knowledge also increases a lot from the beginning to the end of conceptual design. At the beginning period the design efficiency is much emphasized while as design goes on more and more focus is put on the design effectiveness. A designer in the early stages of conceptual design knows little about the problem or the design space and does not necessarily know which type of DOE or which metamodeling methods will be most effective and efficient for that particular problem. As the design evolves and more information becomes available, it may be possible to determine which

methods are appropriate for that particular problem. From the viewpoint of metamodeling, this shift of design requirements corresponds to the development of more accurate metamodels with sequential experiments. The Response Surface Methodology (RSM) is such a method in which sequential experimental designs and sequential metamodels are utilized to reflect the different information and requirements along the design timeline. Although RSM has been widely applied and proved to be useful, it has many weak points as well as strong points. It is confined to classical experimental designs and regression polynomial models (which is referred as RS models in this dissertation). This limits its usage in deterministic applications (for details, see, Welch, et al., 1990; Simpson, et al., 1997b). How to design sequential computer experiments and develop series of appropriate metamodels along the design timeline in accordance with the changing design information is still an open problem.

The primary objective of the research in this dissertation is to develop a systematic yet flexible method in which various metamodeling techniques are utilized in building series of appropriate metamodels for robust design space exploration in accordance with the change in information quality along the design timeline at the early stages of design. Development of the method will be accomplished by (1) studying measures for metamodel validation with deterministic computer experiments, (2) developing methods for sequential experimental design in fixed design spaces, (3) studying the integration of metamodeling and design space exploration within a fixed design space, and (4) developing methods for model selection along the design timeline.

In this section, the motivation and background for the research are introduced. An overview of the engineering design processes and the design of large-scale engineering systems is first given in Section 1.1.1. Open engineering systems design is then described in Section 1.1.2. Metamodeling techniques and its applications in robust design are introduced in Section 1.1.3.

1.1.1 Engineering Design Processes and Design of Large Scale Engineering Systems in Early Stages

To describe and improve engineering design processes, various theories and methodologies have been developed in previous research. Finger and Dixon (Finger and Dixon, 1989a; Finger and Dixon, 1989b) provide taxonomy distinctions among design methods based on observing how designers go about their work, namely, descriptive models of design processes (Ericsson and Simmon, 1980; Ericsson and Simon, 1984; Laird, et al., 1987), methods based on formal grammars and axioms, namely, prescriptive models of design processes (Hubka, 1982; Pahl and Beitz, 1986; Roth, 1982), and the design of computer-based models (Nevill, 1989) which help a designer in whatever methodology is used. Mistree and co-authors provide a comprehensive review of the aforementioned works and other developments in the field of design theories and methodologies (Mistree, et al., 1990a). Although models of design processes vary significantly under these different streams of research, there are some models which are widely acceptable and make intuitive sense to many designers. An example is the four major design phases identified by Pahl and Beitz (Pahl and Beitz, 1984):

- Clarification of the task – collection of information about the requirements to be embodied in the solution and also about the constraints.
- Conceptual design – establishment of function structures, the search for suitable solution principles and their combination into concept variants.
- Embodiment design – starting from the concept, a designer determines the layout and forms, and develops a technical product or system in accordance with technical and economic considerations. Embodiment design is sometimes called preliminary design.
- Detail design – all the details of the final design are specified and manufacturing drawings and documentation are produced.

By "early design stages" we mean activities that happen in the first two phases: clarification of the task and conceptual design. In the early design stages, design concepts are synthesized at the system level based on mission requirements or market opportunities. As a result, the *conceptual baseline* is developed and represented by a set of *top-level specifications*. The conceptual baseline then becomes the configuration input for preliminary design, where the system is decomposed for more sophisticated analysis by discipline, subsystem, or component (Chen, 1995). Top-level design specifications are the descriptions of system/subsystem concepts or the definitions of the complex system at the system/subsystem level. They are used as the starting point for the preliminary design at the subsystem level, and form the basis for the specifications (functional properties) that are developed during the preliminary design phase. The top-

level design specifications can be continuous, which means any value within a specified range can be used; they can also be discrete variables or different design concepts.

It has been shown that a significant portion of the total life cycle cost of systems is determined during the early design stages where the top-level specifications are generated. The quality engineering tools, e.g., the 7 management tools (Brassard and Ritter, 1994) and Quality Function Deployment (QFD) (Clausing, 1994) have become popular in the early stages of product design, while they are most frequently used in the engineering management level. The Robust Concept Exploration Method (RCEM), which was developed by Wei Chen in the Systems Realization Laboratory in 1995, offers a systematic method for integrating and transforming the overall design requirements into top-level design specifications in the early design stages. The RCEM has been successfully used in designing solar power irrigation systems, engines, aircraft, etc.

Another important issue to be addressed about design in the early stages is how engineers do tradeoffs between design effectiveness (accuracy) and design efficiency (speed). Along a design timeline, design information increases and design knowledge for design changes. In the early design stages where accurate analysis is unavailable or not needed while design time is limited, more emphasis is put on design efficiency than on design effectiveness. In the later design stages where more design knowledge is available and accurate analysis is needed to insure the product performance, design effectiveness is given much higher priority than design efficiency. In designing large-scale engineering systems, this issue has been more apparent.

Most large-scale engineering systems are *complex systems*. A complex system is a system composed of a number of subsystems where each subsystem is embodied by a particular set of components, or sub-subsystems. Each component has its own working principle. In addition, the system, subsystems, and components involve the interactions of multiple disciplines. Decomposition or partitioning of complex systems has long been viewed as beneficial to the efficient solution of a system. The decomposition schemes historically have been hierarchical in nature (Renaud, 1992; Koch, 1998), while many systems lend themselves to non-hierarchic decomposition schemes instead of hierarchical ones (Renaud, 1992). The decomposition and synthesis of large-scale engineering systems (especially complex systems) is not our research focus in this dissertation. We are interested in sequential metamodeling and design space exploration of robust solutions in designing such large-scale systems that have multi-disciplinary design variables, constraints, and goals.

In designing large-scale engineering systems, we usually have to do tradeoffs among various design goals and satisfy various design constraints in different disciplines through leveraging lots of design variables. The analyses of system performance are usually very complicated, which makes it necessary to introduce rigorous analysis tools in design in early stages. Rigorous analysis tools are sophisticated computer analyses or simulation programs to predict the behavior of product or process. Examples are finite element programs for stress analysis, engine cycle analysis programs for thermodynamic analysis, etc.

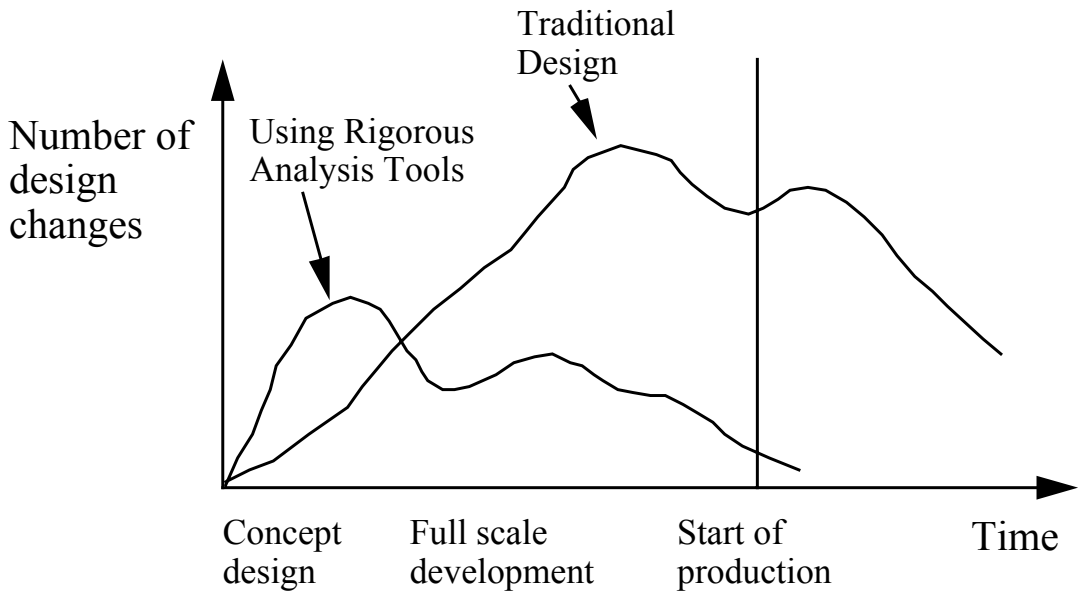


Figure 1.1 An Assumption – Using Rigorous Analysis Tools in Concept Design Will Reduce the Number of Design Changes (Chen, 1995)

The introduction of rigorous analysis tools in early design stages helps improve the comprehensiveness and fidelity of design analyses and reduce the number of design changes in later stages of design. The benefits of introducing rigorous analysis tools at the early stages of design are schematically demonstrated. In traditional design most of the design changes happen in late design stages (when and after the design is fully developed), which results in a high design cost. By using the rigorous analysis tools in the early design stages, product performance can be observed and evaluated so that great changes in later design stages are avoided; total design changes are decreased and design cost is reduced.

A significant property of most rigorous analysis tools is that they are deterministic computer analysis codes, which means that with the same input we will hopefully always

get the same output. This property is important in building metamodels (Section 1.1.3) for concept exploration.

1.1.2 Information Handling in Design of Open Engineering Systems

To develop complex engineering systems, such as engines, vehicles, etc., in the Industrial Era, manufacturers used "dedicated" engineering systems to mass-produce their products. While in today's increasingly competitive markets, the trend is toward mass customization, something that becomes increasingly feasible when modern information technologies are used to create open engineering systems (Simpson, et al., 1997a). The techniques studied in this dissertation aid designers to enhance product flexibility and variety (if not fully customized products) through the development of open engineering systems.

Open engineering systems are those of industrial products, services, and/or processes that are readily adaptable to changes in their environment which enable producers to remain competitive in a global marketplace through continuous improvement and indefinite growth of an existing technological base (Simpson, et al., 1997a). In essence, an open engineering system resembles a readily adapting system whose benefits include *increased quality, decreased time-to-market, improved customization*, and *increased return on investment* which are enhanced through the system's capability of being adapted to change. A system that cannot be adapted to a changing marketplace becomes extinct. It would be a waste of time and effort if we design new artifacts from scratch, but with open engineering systems we can easily adapt our design to the new requirements and get quality products.

There are many examples of open engineering systems, e.g., the IBM PC and the Boeing 747 series. Generations of IBM PCs have been developed (built around the Intel 80286, 80386, 80486, and Pentium chips, etc.), and the modularity of the components allows many variations to occur within each generation. Similarly, the Boeing 747-200, 747-300, 74-400, and 747-SP share a strong technological family resemblance; few would argue with Boeing's view either of the family or the models within the family (Simpson, et al., 1997a). Another example is from Zeneca. Zeneca Pharmaceutical Research is preparing now for what is to come. They are adapting their structure and management today to develop new drugs more than a decade away. They realize that the healthcare industry is changing and these "agents of change" are willing to be flexible to change with it to offer the best to patients and clinicians. They are accepting aspects and tools from the open systems paradigm we explained before.

The basic premise in designing an open engineering system is to quickly get a quality product to market and then remain competitive in the marketplace through continuous development of the product line. It relies heavily on three important requirements (as shown in Figure 1.2):

- Increasing design knowledge during early design phases,
- Maintaining design freedom during early design phases, and
- Increasing efficiency of the design process.

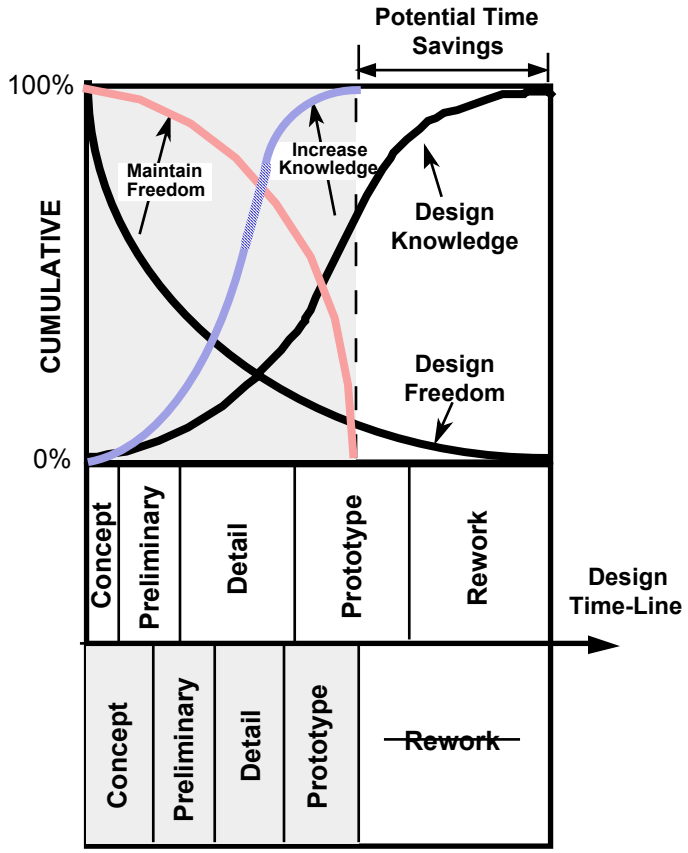


Figure 1.2 Reducing Time-To-Market by Increasing Design Knowledge and Maintaining Design Freedom (Simpson, 1995)

By increasing design knowledge during early design phases we are able to develop a better understanding of the system and get a feel for the system sensitivity. This enables us to answer questions about reliability and manufacturability that are usually posed in the later stages of product development and avoid rework. To maintain design freedom means that we should not restrict the choices that are available quite early in the design process. It is desirable to keep design freedom in designing complex systems so that changes are implemented more easily. Increasing efficiency implies

making the process quicker in terms of the computations involved and making wise approximations in order to increase the computational efficiency of the process. Wherever possible the process should be automated.

These notions are graphically represented in Figure 1.2, where the changes of the design knowledge and design freedom are shown as curves along the design timeline. The curves in solid represent changes in the design of "closed" systems and the curves in dash represent those changes in the design of open engineering systems. In switching from closed systems to open ones we reduce time-to-market and gain profits by compressing the design knowledge curve. We also want to maintain design freedom longer, which results in a different curve with only a gradual decrease at the beginning.

By spending a larger amount of time in conceptual design as shown in Figure 1.2 and by maintaining design freedom and increasing design knowledge, design changes (especially those which occur during later design stages) can be avoided, and a potential time savings and greater return on investment can be achieved. A lot of flexibility is provided for the later design stages and rework can be eliminated from the design process because maintaining design freedom and increasing design knowledge helps prepare for unforeseen changes in the later stages of design and facilitates adaptation to these changes (Simpson, 1995).

We could achieve the design of open engineering systems through the following three characteristics (Simpson, et al., 1997a): *Robustness*, *Modularity*, and *Mutability*. Given these three characteristics, in this dissertation we focus on the *robustness* of the complex systems. We propose to do sequential metamodeling along the design timeline

in approximation-based robust design to help increase design knowledge, increase design freedom, and increase efficiency in the early design stages:

- *Increase Design Knowledge in Early Design Stages*. We achieve this by: 1). Using rigorous analysis tools that abstract issues from later design stages to early design stages; 2). Grasping maximum design information with least effort through sequential experimentation and metamodeling; 3). Gaining insight into the relationships among the design factors and system performance; 4). Exploring the design space to study the system performance and robustness; 5). Studying changes in the design factors due to different scenarios or tradeoff studies; and 6). Answering "what-if" questions during the design process.
- *Increase Design Freedom*. We propose to achieve this by: 1). Searching for *satisficing ranged sets of solutions* rather than optimal or point solutions; 2). Incorporating robustness into the design by making the design insensitive to changes in the later design stages; 3). Enhancing concept exploration by not restricting the number of parameters considered or limiting their ranges; 4). Mathematically modeling the quality of information and not restricting the feasible design space based on uncertain information; and 5). Developing sequential metamodels with consideration of both metamodel uncertainty and the achievement of design goals.
- *Increase Efficiency*. We achieve this by: 1). Using sequential experimental design and sequential metamodeling to obtain design information quickly; 2).

Selecting and building appropriate metamodels for system performance to improve computational efficiency; and 3). Utilizing distributed design techniques to do computer experiments automatically.

As mentioned before, these proposed topics and methods are implemented in approximation-based robust design, Section 1.1.3.

1.1.3 Approximation-Based Robust Design and the Needs for Sequential Metamodeling and Sequential Design Space Exploration

As mentioned in Section 1.1.2, approximation-based robust design is preferred in designing complex engineering systems to help make the system open. The fundamental idea underlying robust design, originally proposed by Taguchi (1987), is to improve the quality of a product or process by minimizing the effects of variation without eliminating the causes of that variation while simultaneously striving to achieve performance targets. It is commonly accepted that the principles associated with Taguchi's approach are both useful and very appropriate for industrial product design (see, e.g., Byrne and Taguchi, 1987; Phadke, 1989; Ross, 1988) though certain limitations associated with Taguchi's method have been identified (see, Nair, 1992; Tsui, 1992; Box, 1988). The difference between "optimization" and "robust design" is shown in Figure 1.3 and Figure 1.4. In this dissertation, Figure 1.3 and Figure 1.4 are attributed to David Craig in the class of ME8104: Design Open Engineering System, Spring 1995. It is believed that the difference between these two approaches stems from what are considered to be good for the design in the context of the entire design process.

The possible result of design with "optimization" is illustrated in Figure 1.3. In optimization, there is a best solution for design in each stage of the process and a rigid optimal solution is prescribed. An unavoidable change made later in the design process will shift the design away from the optimum point without a clear idea of what will happen to the design as a whole. While in Figure 1.4, where robust design is illustrated, each step is left somewhat open to ensure that the design is still good even after new concerns for the design arise later.

To design open engineering systems we apply Taguchi's robust design techniques in the early stages of design, which helps us gain robustness in decision making and reduce the number of design changes and iterations. In robust design parameters and responses are identified to figure out the sources of variability. The focus in robust design is to reduce the variation of system performance caused by uncertain design parameters, or to reduce system sensitivity; solutions are sought to minimize response variation in addition to achieving performance targets. By taking this approach, robust solutions obtained for complex systems involving significant uncertainty are usually not *optimal* in the traditional sense, but *satisficing*. Here Taguchi's robust design principles are consistent with the notion of *satisficing* which was coined by Simon (Simon, 1982) to describe a particular form of less-than optimal solutions. Satisficing solutions are solutions that are good enough to be acceptable but are neither exact nor optimal.

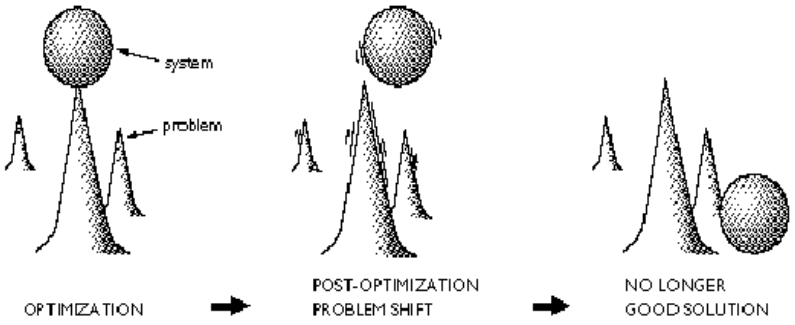


Figure 1.3 What Might Happen After a Rigid Optimal Solution is Prescribed (Chen, 1995)

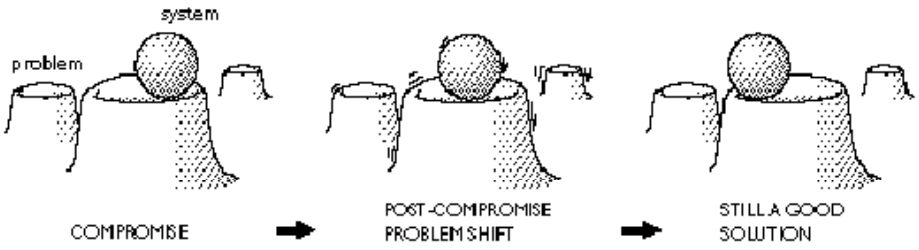


Figure 1.4 Robust with Respect to the Evolution of the Problem (Chen, 1995)

Another issue that is consistent with *satisficing* is the usage of approximation models for the rigorous analysis tools in early stages of design. As stated in Section 1.1.1, rigorous analysis tools are used in designing complex engineering systems to help increase both the design fidelity and design efficiency. The usage of simulations in the initial stage of concept exploration of the design space helps us obtain an overview of the system performance and get information about the feasible and satisficing regions of the design space. Although rigorous analysis tools are needed to achieve a high level of fidelity for concurrent system analysis, there are difficulties that must be overcome

before these tools can be used efficiently in the early stages of designing complex systems. When faced with a complex real-world problem in design, we do not recommend using the exact analysis codes to predict and plot the system behavior because it will still be time-consuming and inconvenient to run the computer simulation programs. Instead, we propose to generate and plot an approximation of the system behavior using some kind of heuristics. This approximation, or model-of-the-model, is called a metamodel (Kleijnen, 1987). A detailed description of metamodeling methods can be found in (Simpson, et al., 1997b). The metamodeling techniques will be further discussed in Chapter 2. When using metamodels in design the solutions obtained by the approximate algorithm or heuristics are *satisficing*. These solutions may be less than optimal, but they still meet the most important goals and constraints and at the same time, they provide enough flexibility to make the system open to the uncertain changes in later design without undue penalties in function, cost, time and other considerations.

In approximation based robust design, metamodeling techniques and robust design principles are combined and applied to help designers make complex engineering systems open in the early design stages by exploring for *satisficing* top-level specifications. The Robust Concept Exploration Method (RCEM), which is developed in the SRL in 1995, is a systematic approach to realize the approximation based robust design of complex engineering systems. Study in this dissertation will be conducted in the context of RCEM which is briefly introduced in Section 1.2.2.

In approximation-based robust design, we aim to achieve robust solutions *efficiently* and *effectively*. This is very closely related to the maintenance of design

freedom and increase of design knowledge as described in Section 1.1.2. By using experimentations and approximations, we are able to increase design knowledge quickly and maintain maximum possible design knowledge in early design stages. However, with different metamodeling techniques, the effects of approximation may be very different. As mentioned before, design requirements and information may change dramatically along the design timeline. Currently there are no systematic yet flexible methods in which various metamodeling techniques are utilized in building series of various types of appropriate metamodels for robust design space exploration in accordance with the change in information quality along the design timeline at early stages of design:

- The Response Surface Methodology (RSM) is a method in which sequential experimental designs and sequential metamodels are utilized to reflect the different information and requirements along the design timeline, while it is confined to classical experimental designs and regression polynomial models (referred to as response surface or RS models).
- Various types of design of experiments (DOE) have been proposed and studied, however, these DOE's are seldom used sequentially in metamodeling. For example, orthogonal arrays, latin hypocubes, etc., are usually used in single-stage experimental designs; it is difficult to add in new data points with these DOE techniques. Traditional experiments used in RSM could be designed sequentially, while their applications are limited as will be described in Chapter 2. A one-stage experimental design does not help maintain

maximum design freedom; the amount of design information gained may not be enough; and it is not efficient when additional data points are needed to develop more accurate metamodels. A more detailed discussion on DOE techniques will be presented in Chapter 2.

- Various types of metamodels have been used in engineering design, while they are seldom used together (sequentially or simultaneously) in the design process. Designers tend to stick to only one of these types of metamodels in design, which may result in less-flexible strategies.
- The integration of metamodeling and design space exploration is still limited. As will be discussed in Section 1.2.2 (RCEM) and Chapter 2, design space exploration of robust solutions is usually conducted after the development of metamodels. A more flexible strategy is needed to integrate the processes of metamodeling and design space exploration – robust solutions are achieved in the process of experimentation and metamodeling. This helps maintain maximum possible design freedom, increase design knowledge quickly, and save time and efforts in design.

To address the above problems, it is necessary to develop strategies for sequential metamodeling that incorporate techniques of sequential experimental design, sequential metamodel selection and development, and design space exploration. The goal is to grasp maximum design information with least time and effort in early design stages; design freedom could be maintained through management of the design information properly. Before detailed descriptions of these ideas in Section 1.3, Research Foci in

This Dissertation, in the next section, the frame of reference for research in this dissertation is presented and discussed.

1.2 FRAME OF REFERENCE

The technology base for the dissertation is described in this section. An overview of Decision-Based Design and the compromise Decision Support Problem is given in Section 1.2.1, followed by an overview of the Robust Concept Exploration Method (which is the context for our research in this thesis) in Section 1.2.2.

1.2.1 Decision-Based Design, the Decision Support Problem Technique, and the Compromise Decision Support Problem

Decision-Based Design (DBD) (Mistree, et al., 1990a; Shupe, et al., 1988), a phrase coined to emphasize a different perspective from which to develop methods for design, is used as the design paradigm for RCEM. This paradigm, which encompasses systems thinking and embodies the ideas of concurrent engineering design for the life cycle, is rooted in the notion that "the principal role of a designer, in the design of an artifact, is to make decisions" (see, e.g., Muster and Mistree, 1988; Mistree, et al., 1990b). This role is useful in providing a starting point to develop design methods based on paradigms that spring from the perspective of decisions made by designers (who may use computers) as opposed to design that is predicated on the use of computers, optimization methods (computer-aided design optimization), or methods that evolve from specific analysis tools such as finite element analysis.

The implementation of Decision-Based Design that is the Decision Support Problem (DSP) Technique (Muster and Mistree, 1988; Bras and Mistree, 1991; Mistree,

et al., 1993a), a technique that supports human judgment in designing systems which can be manufactured and maintained. As a foundation of the DSP Technique, designing is defined as *the process of converting information that characterizes the needs and requirements for a product into knowledge about a product* (Mistree, et al., 1990a). This definition is consistent with the design transformation shown in Figure 1.5 if *information that characterizes the needs and requirements for a product* is characterized simply as *requirements* and *knowledge about a product as specifications*. The necessary transformation of requirements for a design into design specifications within the DSP Technique then becomes a series of decisions. For a better description of the DSP Technique see (Mistree, et al., 1990a).

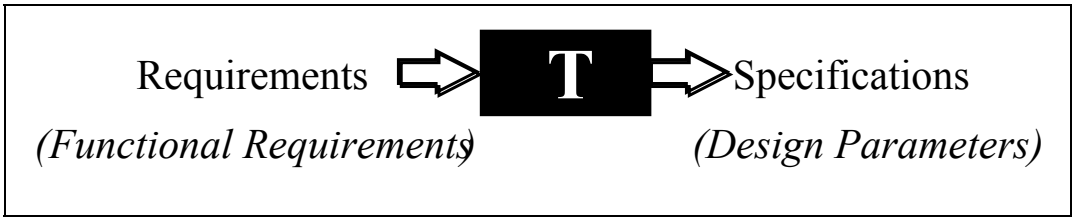


Figure 1.5 Design as a Transformation Between Requirements and Specifications (Koch, 1998)

Among the tools available within the DSP Technique, the compromise DSP (Mistree, et al., 1993b) is a general framework for solving multi-objective, non-linear optimization problems. In this dissertation, the compromise DSP is central to modeling multiple design objectives and assessing the tradeoffs pertinent to robust design of complex systems. Mathematically, the compromise DSP is a multi-objective decision model which is a hybrid formulation based on Mathematical Programming and Goal Programming (Mistree, et al., 1993b). The compromise DSP is used to determine the

values of the design variables which satisfy a set of constraints and bounds and achieve as closely as possible a set of conflicting goals. The compromise DSP in this dissertation is solved using the Adaptive Linear Programming (ALP) algorithm which is based on sequential linear programming and is part of the DSIDES (Decision Support in Designing Engineering Systems) software (Mistree, et al., 1993a).

Formulation of a compromise DSP begins with a word formulation and proceeds to a mathematical formulation. The word formulation consists of the keywords *given*, *find*, *satisfy*, *minimize* and their associated descriptors, as shown in Figure 1.6. *Given* an alternative and domain information for a problem at hand, the objective in the compromise DSP is to *find* the values of system design variables which *satisfy* a set of constraints and bounds and achieve as closely as possible a set of conflicting goals while *minimizing* a deviation function.

COMPROMISE DSP

Keywords	**Descriptors**
Given	An alternative to be improved through modification; assumptions, system parameters, constraints, bounds, goals, and the deviation function.
Find	Values of system variables and deviation variables.
Satisfy	System constraints and bounds (feasibility), and goals (desired target values or objectives).
Minimize	A deviation function.

Figure 1.6 Compromise DSP Word Formulation

Given

An alternative to be improved. Assumptions used to model the domain of interest.

The system parameters:

n number of system variables, q inequality constraints

p + q number of system constraints,

m number of system goals

$g_i(\mathbf{x})$ system constraint function

$f_k(d_i)$ function of deviation variables to be minimized at priority level k^{th} for the preemptive case.

Find

The values of the independent system variables:

$x_i \quad i = 1, \ldots, n;$

The values of the deviation variables:

$d_i^-, d_i^+ \quad i = 1, \ldots, m$

Satisfy

System constraints (linear, nonlinear)

$g_i(\mathbf{x}) = 0$ for $i = 1, .., p; \quad g_i(\mathbf{x}) \geq 0$ for $i = p+1, .., p+q$

System goals (linear, nonlinear)

$A_i(\mathbf{x}) + d_i^- + d_i^+ = G_i \quad i = 1, \ldots, m$

Bounds

$x_i^{min} \leq x_i \leq x_i^{max} \quad i = 1, \ldots, n$

$d_i^-, d_i^+ \geq 0 ; i = 1, \ldots, m; \quad d_i^- \cdot d_i^+ = 0 ; i = 1, \ldots, m$

Minimize

Preemptive deviation function (lexicographic minimum):

$\mathbf{Z} = [f_1(d_i^-, d_i^+), ..., f_k(d_k^-, d_k^+)]$

Figure 1.7 Mathematical Form of a Compromise DSP (Mistree, et al., 1993b)

The generic mathematical formulation of the compromise DSP is presented in Figure 1.7. The compromise DSPs are written in terms of *n system variables*, a vector *X*, defining the physical attributes of an artifact that can be altered. A set of p+q *system constraints* is used to model the limits placed on a system design, and must be satisfied for feasibility. Mathematically, system constraints are functions of system variables only, and may be a mix of linear and nonlinear functions. *Bounds* are specific limits placed on the magnitude of each of the system variables. A set of *m system goals* is used to model

the aspirations for the design. It relates the goal target, G_i, to the actual performance, $A_i(X)$, of the system with respect to the goal. The *deviation variables*, d_i^- and d_i^+, are introduced as a measure of achievement, the difference between $A_i(X)$ and G_i. In the compromise DSP the objective is to minimize a *deviation function*, $Z(\mathbf{d}^-, \mathbf{d}^+)$, a function of the deviation variables. The form of these formulations is given in Figure 1.7.

In the compromise DSP, goals may either be weighted in an Archimedean solution scheme or rank-ordered into priority levels using a preemptive approach to affect a solution on the basis of preference. For the preemptive approach, the lexicographic minimum concept (Ignizio, 1985) is used to quickly evaluate different design scenarios by changing the priority levels of the goals to be achieved. Differences between the Archimedean and preemptive deviation functions and a description of the ALP algorithm, design and deviation variables, system constraints, goals, and bounds are discussed in (Mistree, et al., 1993b).

A solution to the compromise DSP is a satisficing solution since it is a feasible point that achieves the system goals to the "best" extent that is possible. The efficacy of the compromise DSP in creating ranged sets of top-level design specifications has been demonstrated in both aircraft design (Lewis, et al., 1994; Simpson, et al., 1996) and ship design (Smith and Mistree, 1994). By finding a ranged set of solutions rather than a single point solution, greater design flexibility can be maintained during the design process. Finally, the compromise DSP also provides the cornerstone of the Robust Concept Exploration Method which is overviewed in the next section.

1.2.2 The Robust Concept Exploration Method

The Robust Concept Exploration Method (RCEM) has been developed to facilitate the quick evaluation of different design alternatives and generation of top-level design specifications with quality considerations in the early stages of design (see, e.g., Chen, et al., 1996a). It is primarily useful for designing complex systems which usually utilize computationally expensive analyses. The RCEM is created by integrating several methods and tools – robust design methods (see, e.g., Phadke, 1989), the Response Surface Methodology (see, e.g., Myers and Montgomery, 1995), and Suh's Design Axioms (Suh, 1990) — within the compromise DSP (Mistree, et al., 1993b). In applying robust concept exploration, robust design specifications are identified for the design of complex systems. In this context, robustness of specifications is measured in terms of sensitivity to changes in requirements – thus the focus is on minimizing the effects on the design of uncontrollable noise and/or downstream design changes. The computer infrastructure for implementing RCEM is shown in Figure 1.8.

There are five generic processors (A, B, D, E, and F) surrounding a central "slot" for inserting existing, domain-dependent analysis tools as simulation programs (C). The simulation programs are used to evaluate the performance of a number of design configurations. The RCEM processors increase computational efficiency and facilitate the generation of robust design specifications. The point generator (processor B) is used to design the necessary screening experiments. The experiments analyzer (processor D) is used to evaluate the results of the screening and to plan additional experiments. The response surface model processor (E) is used to create response surface models, and the

compromise DSP processor (F) is used to explore a design space and identify robust design specifications.

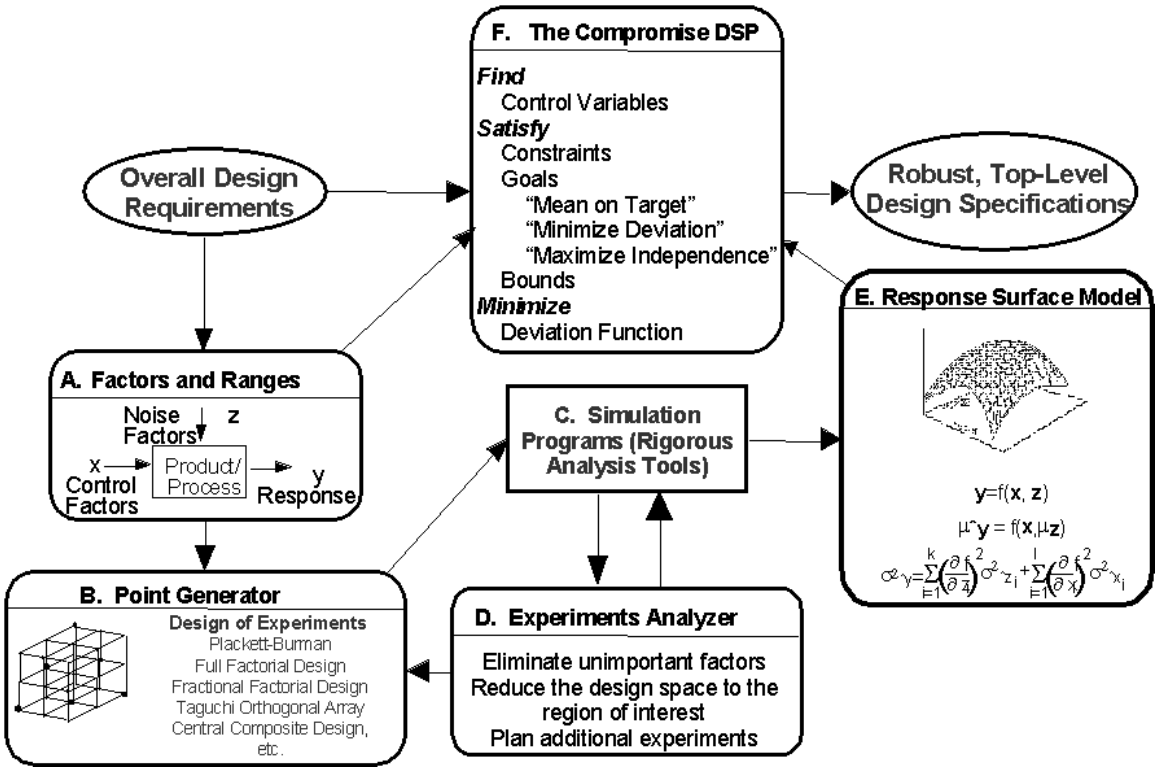


Figure 1.8 RCEM Computer Infrastructure (adapted from Chen, et al., 1996a)

The RCEM is a four-step process as shown in Figure 1.9. The steps are described as below:

Step 1 - Classify Design Parameters: Given the overall design requirements, this step involves the use of Processor A, see Figure 1.8, to (a) classify different design parameters as either control factors, noise factors, or responses following

the terminology used in robust design and (b) define the concept exploration space.

Step 2 - Screening Experiments: This step requires the use of the point generator (Processor B), simulation programs (Processor C), and an experiment analyzer (Processor D) shown in Figure 1.8 to set up and perform *initial screening experiments* and analyze the results. The results of the screening experiments are used to (a) fit low-order response surface models, (b) identify significant main effects, and (c) reduce the design region.

Step 3 - Elaborate the Response Surface Model: This step also requires the use of the point generator (Processor B), simulation programs (Processor C), and experiment analyzer (Processor D) to set up and perform *secondary experiments* and analyze the results. The results from the secondary experiments are used to (a) fit second-order response surface models (using Processor E) which replace the original computer analyses, (b) identify key design drivers and the significance of different design factors and their interactions, and (c) quickly evaluate different design alternatives and answer "what-if" questions in Step 4.

Step 4 - Generate Top-Level Design Specifications with Quality Considerations: Once accurate response surface models have been created, Step 4 involves the use of the compromise DSP (Processor F in Figure 1.8) to determine top-level design specifications with quality considerations. The original analysis or simulation program(s) is replaced by response surfaces which are functions of both control and noise factors. Different quality considerations and multiple objectives are

incorporated in the compromise DSP which is then solved to determine robust, top-level design specifications.

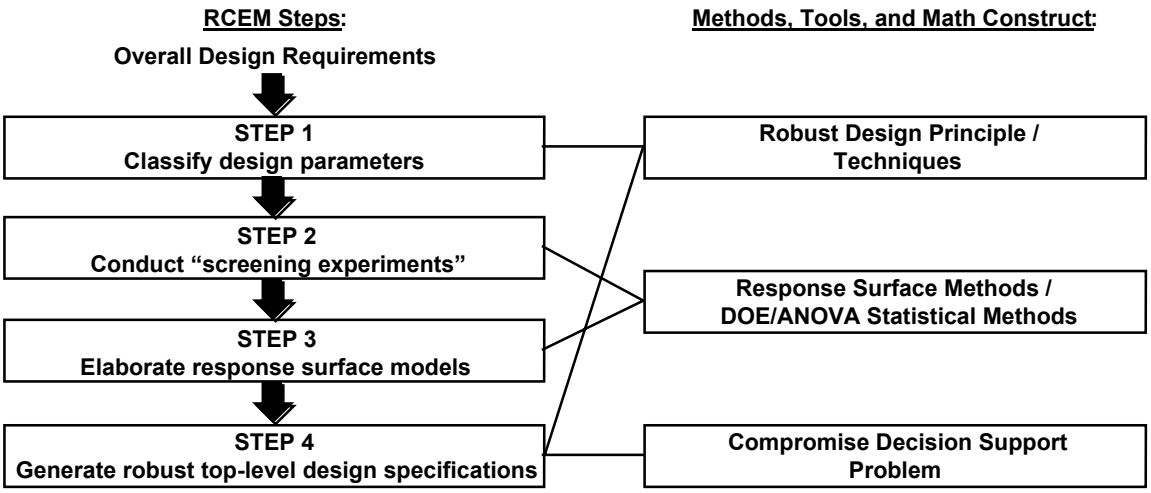


Figure 1.9 Steps and Tools of the RCEM (adapted from Chen, et al., 1996a)

In Figure 1.9 three categories of techniques or mathematical constructs are utilized in implementing RCEM. The robust design principles/techniques are taken into RCEM in Step 1 and 4 by classifying design parameters and formulating robust design goals in the compromise DSP. Metamodeling techniques (the metamodeling techniques used in RCEM are the Response Surface Methodology, which will be described in detail in Chapter 2) are used in Step 2 and 3. Then in Step 4 robust top-level design specifications are generated through solving compromise DSPs. Each step of the RCEM corresponds not only with the implementing techniques but also with the processors shown in the RCEM infrastructure (Figure 1.8).

A review of the wide variety of applications that have successfully employed the RCEM is given in (Simpson, et al., 1997b). In (Chen, et al., 1996a) it is shown that RCEM is used to explore airframe configurations and propulsion system designs and determine robust top-level design specifications for the HSCT (High Speed Civil Transport) system. In (Simpson, et al., 1996) the use of the RCEM in the conceptual design of a family of products is illustrated with the specific example of a general aviation aircraft. In (Rangarajan, 1998) the RCEM is used in designing automobile engines with lubrication considerations.

1.3 RESEARCH FOCUS IN THE DISSERTATION

The research focus in this thesis is embodied in the following:

- a set of research questions that capture motivation and specific issues to be addressed;
- a set of corresponding research hypotheses that offer context by which the research proceeds, defining the structure of the verification studies performed in this work; and
- a set of resulting research contributions that embody the deliverables from the research in terms of intellectual value, a repeatable method of solution, limitations, and avenues of further investigation.

In Section 1.3.1, a discussion on deficiencies of current metamodeling and design space exploration techniques in RCEM is presented. This leads to the proposed research as described in Section 1.3.2. A set of research questions and corresponding research hypotheses are listed and discussed. The verification and

validation strategy is presented in Section 1.3.3. Contributions of the proposed research summarized in Section 1.3.4; they are revisited in Chapter 8.

1.3.1 Metamodeling and Design Space Exploration – Problems to be Addressed

As illustrated in Figure 1.8 and Figure 1.9, metamodeling plays a significant role in RCEM. Robust top-level specifications are achieved through the development and elaboration of metamodels (specifically, RS models). However, the evolvement of design and manipulation of design information through metamodeling are limited in RCEM, as being explained in the following paragraphs.

Techniques used in metamodeling are confined to those from the Response Surface Methodology (RSM). Only traditional experimental designs (i.e., factorial design, central composite design, etc.) are used to develop a single type of metamodels, the RS models (regression polynomials). This constrains the amount of design information obtained in metamodeling; as design evolves, the experimental designs and corresponding metamodels may fail to provide sufficient design information. Space-filling experiments and kriging metamodels are used in recent applications of RCEM, but as will be discussed in Chapter 2, the experiments are still conducted at a single stage and there are still other types of metamodels that may be used to provide designers more flexibility. A method needs to be developed to (1) integrate the usage of different types of metamodels in accordance with different design requirements along the design timeline, and (2) facilitate sequential experimental designs.

Metamodel validation methods used in RCEM are not suitable for deterministic computer experiments. To validate the accuracy of a metamodel, in RCEM we use

various statistics such as F-statistics, etc. However, as is explained in Chapter 2, these statistics are theoretically unsuitable for computer experiments. Thus new methods to validate a metamodel are needed since our simulations are usually computer codes that yield deterministic results.

No sequential metamodeling is involved in RCEM. Although experiments are conducted in a sequential manner (from factorial design to central composite design), the metamodel is developed only once after finishing all experiments. This is illustrated in Figure 1.8 in which there is only one loop among Processors B, C, and D, and there is no feedback from Processor E (development of metamodels) to the DOE loop B-C-D. It is a single-stage metamodeling in which information gained in the development of metamodels is not used in design of experiments and collection of future design information. Further more, if no screening happens in the DOE loop B-C-D, the experimental design could also be viewed as single-level experimental designs though it is conducted sequentially, because information collected in previous experiments has no influence on the conduction of future experiments. As a summary, the single-stage metamodeling in RCEM is not suitable for large-scale engineering systems design in which usually a series of metamodels are needed to meet the design requirements as the design evolves along the design timeline.

The integration of metamodeling and design space exploration is not completed in RCEM. As illustrated in Figure 1.8, there is no feedback from Processor F to the metamodeling process B-C-D-E. Information flows one way from metamodeling to design space exploration. This results in a single-stage conceptual design. However, in

the conceptual design of large-scale engineering systems, as design requirements and information change along the design timeline, a series of metamodeling and design space exploration processes are needed to help achieve robust top-level specifications. The information from previous metamodeling and design space exploration should be used as guidance in conducting future metamodeling and exploration of robust solutions. The two processes of metamodeling and design space exploration should not be separated and used sequentially, but rather integrated into one process. Metamodeling should be done in the process of design space exploration; and on the other hand, we could also say that design space exploration should be done in the process of metamodeling. The two processes of metamodeling and design space exploration are done simultaneously and interactively. Through this integration we are able to manipulate the design information (in a seamless process) that was previously (in RCEM) managed in two separated, sequential processes of metamodeling and design space exploration. This helps maintain the design freedom (more flexible, more options keep open before the end of metamodeling and design space exploration), increase design information (more information is obtained in an "active" mode of information collection than that in a "passive" mode), and improve the design efficiency (more information is gained with less time and effort).

As discussed above, to maintain design freedom and increase design information with less time and efforts, we must study and develop methods for metamodeling and design space exploration based on current research achievements. These includes:

- Metamodel validation techniques for deterministic computer experiments;

- Comparison of different types of metamodels and usage of different metamodels along the design timeline;
- Design of sequential computer experiments to achieve accurate metamodels; and
- Integration of metamodeling and design space exploration processes.

The design process at conceptual design stage (including the metamodeling process, design space exploration process, and their interactions) needs to be revised: information flow needs to be redirected and techniques used in the process need to be studied. Research questions of this dissertation are proposed in the next section based on our discussions above.

1.3.2 Research Questions and Hypotheses in this Dissertation

The principal goal in this dissertation is to examine and develop techniques in metamodeling and design space exploration. As discussed in previous sections, the design process at conceptual design stage (metamodeling process + design space exploration process) needs to be revised: information flow needs to be redirected; feedbacks need to be added; and techniques used in the process need to be studied. According to this, the primary objective of the proposed research is to develop a systematic yet flexible method in which various metamodeling techniques are utilized in building series of appropriate metamodels for robust design space exploration in accordance with the change in information quality along the design timeline in the early stages of design. Development of the method will be accomplished by (1) studying

measures for metamodel validation with deterministic computer experiments, (2) developing methods for designing sequential computer experiments, (3) developing methods to integrate the processes of metamodeling and design space exploration, and (4) comparing and using different types of metamodels according to the changing design requirements along the design timeline. Given these goals, the ***key question*** to be addressed in this dissertation is presented as:

KEY QUESTION:

> *How to explore the design space efficiently and effectively for satisficing solutions by employing sequential metamodeling and design space exploration techniques in accordance with the changing design information along the design timeline in early design stages?*

This key question defines the scope and goals of the research documented in this dissertation. Several research objectives are reflected in this key question. By using the phase of "sequential metamodeling and analysis" three research objectives are defined: (1) sequential experimental design – the core step in sequential metamodeling; (2) sequential metamodels – comparison of different types of metamodels, and selection and usage of different metamodels sequentially; and (3) metamodel validation – analysis of the accuracy of metamodels. With the phase of "explore the design space … for robust solutions … in early design stages", the context of research in this dissertation is fixed, i.e., the Robust Concept Exploration Method for developing top-level specifications.

Examples and case studies in this dissertation are designs at conceptual design stages; techniques and methods are developed and applied in robust design though it is not necessarily confined to this field. We aim at developing a systematic method similar to but more general than the Response Surface Methodology; RCEM will be a platform that we worked on with the proposed method. The usage of "explore the design space … by employing sequential metamodeling" identifies the research objective of developing a method to integrate the processes of metamodeling and design space exploration – to do metamodeling and design space exploration simultaneously and interactively. By using "in accordance with the changing design information" three issues are reflected: (1) the uncertainty associated with design requirements should be considered in metamodeling and design space exploration; (2) the uncertainty associated with metamodel accuracy should be considered in metamodeling and design space exploration; and (3) a measure of information uncertainty need to be developed. The words "efficiently" and "effectively" indicates my focus in the proposed research: achievement of the design goals and satisfy design requirements with least time and effort, by grasping and utilizing maximum design information in sequential metamodeling and design space exploration. The key question is expressed as four major research questions as listed below.

RESEARCH QUESTIONS:

> ***R.Q.1:*** *How to validate a metamodel with deterministic computer experiments?*

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

R.Q.4: *How to utilize different types of metamodels along the design timeline in accordance with the changing design information? (How to do sequential metamodeling to achieve robust design solutions?)*

To answer the first research question it is necessary to study the widely used technique, cross-validation, in deterministic applications; new approaches are developed to help validate metamodels with additional validation points in the design space. To answer the second research question, it is needed to study how to measure information and the worth of a point in the design space, and then apply this in the identification of new data points in sequential experimental design. A new method, the Sequential Exploratory Experimental Design (SEED), is documented based on studies under this research question. As for research under the third research question, design goals and requirements are considered in designing experiments and developing metamodels. The processes of metamodeling and robust design space exploration are integrated into one process; new data points are those which yields great information worth and/or response values close to target values. This results in a new robust design process in early design stages, of which SEED is the basis. To answer the forth research question, first I need to compare the performance of different types of metamodels in metamodeling, then

develop an approach to utilize these metamodels sequentially along the design timeline. In the following sections the supporting research questions and hypotheses of the four research questions are presented, respectively.

1.3.2.1 Research on Metamodel Validation

The research question to be explored in this section is the first research question: *How to validate a metamodel with deterministic computer experiments?* As presented below, this research question is studied and answered with work in two directions: one is to prove the inappropriateness of the currently widely used method, leave-one-out cross-validation, in deterministic applications, and the other is to develop new approaches of metamodel validation. The first research question can be expressed as:

R.Q.1: *How to validate a metamodel with deterministic computer experiments?*

R.Q.1.1: *Is leave-one-out cross-validation a suitable method of metamodel validation with computer experiments?*

R.Q.1.2: *How to test the accuracy a metamodel in deterministic applications?*

Hypothesis 1: Information from either previous additional validation points is needed in testing the accuracy of a metamodel with deterministic computer experiments.

Sub-Hypothesis 1.1: Leave-one-out cross-validation is not an appropriate method of metamodel validation with deterministic computer experiments.

Sub-Hypothesis 1.2: The accuracy of a metamodel could be validated through examining prediction errors at additional validation points.

As described above, Research Question 1 is separated into two supporting research questions. To answer Research Question 1.1, Sub-Hypothesis 1.1 is tested and verified. To answer Research Question 1.2, Sub-Hypotheses 1.2.1 and 1.2.2 are tested and verified. In Chapter 2, some background knowledge related to deterministic computer experiments and leave-one-out cross-validation is presented. Sub-Hypothesis 1.1 is primarily discussed and tested in Chapter 3. Sub-Hypothesis 1.2.1 is also primarily tested in Chapter 3, in which an approach to validate metamodels with additional validation points is developed for engineers. The "worth of possible new data points" is closely related to the measurement of information, which is discussed in Research Question 2 and will be studied in Chapter 4. Sub-Hypothesis 1.2.2 is then tested and verified in Chapter 5. Sub-Hypotheses 1.2.1 and 1.2.2 are occasionally revisited in Chapters 7 and 8.

1.3.2.2 Research on Sequential Exploratory Experimental Design

The research question to be addressed in this section is: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?* To answer this research question we propose to develop a method named Sequential Exploratory Experimental Design (SEED). In this method, data points

and validation points are added and metamodels are developed sequentially; information from previous points and metamodels is used to help identify new data and validation points. This research question could be expressed as:

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

R.Q.2.1: *How to measure the information worth of a point?*

R.Q.2.2: *How to select validation points to achieve a sequential design of computer experiments?*

R.Q.2.3: *How to utilize information from previous points and metamodels in identifying new data points?*

Hypothesis 2: Sequential experiments could be designed through analysis of information from data/validation points and metamodels.

Sub-Hypothesis 2.1: The information worth of a point could be measured with entropy.

Sub-Hypothesis 2.2: Selection of validation points should follow similar rules for selection of data points; information from validation points could be used as guidance in identifying new data points.

Sub-Hypothesis 2.3: Through maximizing entropy (as formulated based on Sub-Hypotheses 1.1 and 1.2) we are able to allocate new data points in the design space that yield maximum potential information.

There is a one-to-one correspondence between each research question and hypothesis. References for the proposed research include various types of Design of Experiments (DOE), D-optimality in DOE, entropy optimization from Information Theory, and maximum entropy sampling, which will be introduced in Chapter 2. The hypotheses are tested and verified in Chapter 4, in which the Sequential Exploratory Experimental Design method is developed. Information uncertainty of an experimental design could be measured with entropy; the utilization of information from previous data/validation points and metamodels could be used to adjust the formulation of entropy; then sequential experiments are achieved by maximizing entropy. The SEED method is further developed and tested in Chapter 5 and applied in Chapter 7 and 8. SEED is the basis of the integration of processes of metamodeling and robust design space exploration, for which a method is developed in Chapter 6.

1.3.2.3 Research on the Integration of Design Processes of Metamodeling and Robust Design Space Exploration

The research question to be addressed in this section is: *How to integrate the processes of metamodeling and robust design space exploration?* To put consideration of design goals and requirements (constraints) in sequential metamodeling results in a new design process in which metamodeling and design space exploration of robust solutions are integrated and done simultaneously. One way to achieve this is to reduce the design space in metamodeling based on previous information; this should be done after carefully examining the whole design space, which may be very difficult when there are a lot of design variables and goals and uncertainty on these goals. The other way is to keep the

design space unchanged but gradually add in points where design goals are met and requirements are satisfied, and/or places with great prediction errors; this could be done on the basis of SEED, with some adjustment. Research Question 3 could be expressed as:

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

R.Q.3.1: *How to design sequential experiments with consideration of design constraints?*

R.Q.3.2: *How to reduce the design space with information from previous metamodeling and design space exploration?*

R.Q.3.3: *How to do sequential metamodeling with consideration of design goals?*

Hypothesis 3: The processes of metamodeling and robust design space exploration could be integrated through building the information flow from C-DSP to the metamodeling cycle in the Robust Concept Exploration Method.

Sub-Hypothesis 3.1: Consideration of design constraints could be incorporated in the metamodeling process through construction irregular design spaces.

Sub-Hypothesis 3.2: Design space could be reduced through analysis of the information from previous metamodels.

Sub-Hypothesis 3.3: Design goals can be taken into consideration in metamodeling by formulating influential factors with the compromise DSP and using them in maximum entropy sampling.

The basis of research in this category is the method of SEED that will be developed in Chapter 4 and further verified in Chapter 5. Sub-Hypothesis 3.2 is studied in Chapter 5, and Sub-Hypotheses 3.1 and 3.3 are to be studied and tested in Chapter 6. A new process of robust design space exploration is developed based on the study under Sub-Hypothesis 3.3. This new method is validated in Chapter 6, and then applied and verified in Chapter 7 and 8 with more complicated engineering case studies.

1.3.2.4 Research on the Selection and Utilization of Metamodels along the Design Timeline

The research question to be addressed in this section is: *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?* In this dissertation, we will only focus on three types of metamodels, the RS model (regression polynomials), kriging model, and Multivariate Adaptive Regression Splines (MARS). To answer Research Question 4, first we need to answer the supporting research questions 4.1 and 4.2, as presented below:

R.Q.4: *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?*

R.Q.4.1: *How do different types of metamodels perform in engineering design?*

R.Q.4.2: *How to select different types of metamodels at different design stages?*

Hypothesis 4: Different types of metamodels should be used at different design stages in accordance with different requirements of design.

Sub-Hypothesis 4.1: Different types of metamodels have their strong and weak points.

Sub-Hypothesis 4.2: As design evolves, more complicated types of metamodels should be used to help yield good approximations with more computation time and efforts.

The introduction of different types of metamodels is presented in Chapter 2. To test Sub-Hypotheses 4.1 and 4.2, comparison of these metamodels is done in Chapter 5. An approach to utilize these metamodels sequentially along the design timeline is also proposed and tested in Chapter 5. This approach of selection and switch of types of metamodels, together with the SEED method developed in Chapter 4, and the new robust design space exploration process developed in Chapter 6, are applied in Chapter 7.

The relationship between hypotheses and chapters is shown in Table 1.1. Hypothesis 1 is mainly discussed and tested in Chapter 3. Hypothesis 2 is mainly discussed and tested in Chapter 4. Hypothesis 3 is mainly discussed and tested in Chapter 6. Hypothesis is mainly discussed and tested in Chapter 5.

There are several examples and case studies in this dissertation. Very simple one-variable or two-variable examples are used in Chapter 3, 4, 5, and 6 to help illustrate and validate our ideas. In Chapter 5, several engineering case studies are used to help

compare the performance of different types of metamodels, and validate the SEED method developed in Chapter 4. One of these case studies, the design of pressure vessels, is used in Chapter 6 to help develop the new process of robust design space exploration. The engineering example used in Chapter 7 is the design of cellular materials; our emphasis in this chapter is the validation of integration of processes of metamodeling and robust design space exploration.

Table 1.1 Relationship Between Hypotheses and Dissertation Chapters

	Hypothesis	Chapters Discussed	Chapters Tested
H1	Metamodel Validation		
SH1.1	Leave-one-out cross-validation	Chp 2, 3	Chp 3
SH1.2.1	Validation with additional validation points	Chp 3, 4, 5	Chp 5, 7
SH1.2.2	Validation with possible new data points	Chp 2, 4	Chp 4, 5, 7
H2	Sequential Exploratory Experimental Design		
SH2.1	Information worth of a point	Chp 2, 4, 6	Chp 4, 5, 7
SH2.2	Selection and usage of validation points	Chp 2, 3, 4, 6	Chp 4, 5, 7
SH2.3	Sequential experimental design	Chp 2, 4	Chp 4, 5, 7
H3	Integration of Processes of Metamodeling and Design Space Exploration		
SH3.1	Experiments in irregular design spaces	Chp 2, 5, 6	Chp 6, 7
SH3.2	Design space reduction	Chp 5, 6	Chp 5
SH3.3	Metamodeling with consideration of design goals and requirements	Chp 2, 6	Chp 6, 7
H4	Selection and Utilization of Metamodels		
SH4.1	Comparison of types of metamodels	Chp 2, 5	Chp 5
SH4.2	Sequential utilization of metamodels	Chp 2, 5	Chp 5, 7

1.3.3 Contributions from the Research

The hypotheses and sub-hypotheses, taken together, define the research presented in this dissertation and hence the contributions from the research. The expected contributions from the thesis are the following:

Expected Contributions related to Hypothesis 1 and Sub-Hypotheses 1.1-1.2:

- Verification of the inappropriateness of leave-one-out cross-validation in testing the accuracy of metamodels with deterministic computer experiments;
- An approach to validate the accuracy of metamodels with information from additional validation points.
- An approach to validate the accuracy of metamodels based on information worth of possible new points.

Expected Contributions related to Hypothesis 2 and Sub-Hypotheses 2.1-2.3:

- The method of Sequential Exploratory Experimental Design (SEED);
- Formulation of information uncertainty of metamodels with consideration of prediction errors.

Expected Contributions related to Hypothesis 3 and Sub-Hypotheses 3.1-3.3:

- The integration of processes of metamodeling and robust design space exploration;
- Design space reduction and sequential experimental design in irregular design spaces.

Expected Contributions related to Hypothesis 4 and Sub-Hypotheses 4.1-4.2:

- The comparison of different types of metamodels (RS, kriging and MARS);
- An approach to utilize different types of metamodels sequentially along the design timeline.

This being the first chapter of the dissertation, these contributions cannot be substantiated; therefore, they are revisited in Chapter 8 after all of the research findings have been documented and discussed. A validation and verification strategy for this dissertation is presented next.

1.4 A VALIDATION AND VERIFICATION STRATEGY FOR THIS DISSERTATION

The validation and verification strategy for this dissertation is based on the validation square by Pedersen and coauthors (Pedersen, et al., 2000). As noted by Pedersen and coauthors, validation (justification of knowledge claims, in a modeling context) of engineering research has typically been anchored in formal, rigorous, quantitative validation based on logical induction and/or deduction. As long as engineering design is based primarily on mathematical modeling, this approach works well. Engineering design methods, however, rely on subjective statements as well as mathematical modeling; thus, validation solely by means of logical induction or deduction is problematic. Pedersen and coauthors propose an alternative approach to the validation of engineering design methods based on a relativistic notion of epistemology in which "knowledge validation becomes a process of building confidence in its usefulness with respect to a purpose."

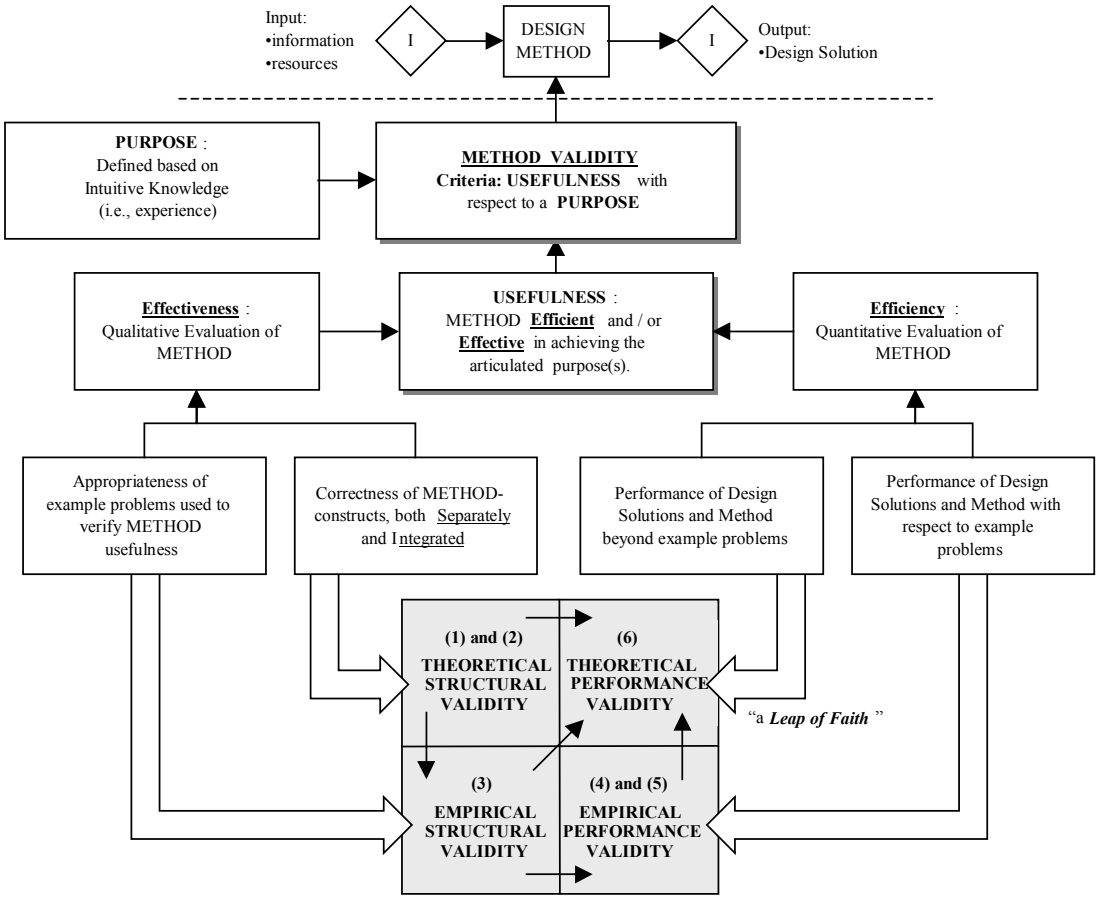


Figure 1.10 The Validation Square: Validating Design Theories or Methods (Pedersen, et al., 2000)

Pedersen and coauthors propose a framework for validating design methods in which the "usefulness" of a design method is associated with whether the method provides design solutions correctly (structural validity) and whether it provides correct design solutions (performance validity). This process of validation is represented in the Validation Square in Figure 1.10. With respect to the square, theoretical structural validity involves accepting the individual constructs constituting a method as well as the internal consistency of the assembly of constructs to form an overall method. Empirical

structural validity includes building confidence in the appropriateness of the example problems chosen for illustrating and verifying the performance of the design method. Theoretical performance validity involves building confidence in the generality of the method and accepting that the method is useful beyond the example problems. Empirical performance validity includes building confidence in the usefulness of a method using example problems and case studies.

How can this validation framework be implemented in a dissertation? Establishing theoretical structural validity involes searching and referencing the literature related to each of the constructs employed in the design method. In addition, flow charts are often useful for checking the internal consistency of the design method by verifying that there is adequate input for each step and that adequate output is provided for the next step. Establishing empirical structural validity consists of documenting that the example problems are similar to the problems for which the methods/constructs are generally accepted, that the example problems represent actual problems for which the method is intended, and that the data associated with the example problems can be used to support a conclusion. Empirical performance validity is established by using representative example problems to evaluate the outcome of the design method in terms of its usefulness. Metrics for usefulness should be related to the degree to which the method's purpose has been achieved. It is also important to establish that the resulting usefulness is, in fact, a result of applying the method. For example, solutions obtained with and without the construct/method can be compared and/or the contribution of each element of

the method can be evaluated in turn. An important part of empirical performance validity is empirical verification of data used to support empirical performance validation.

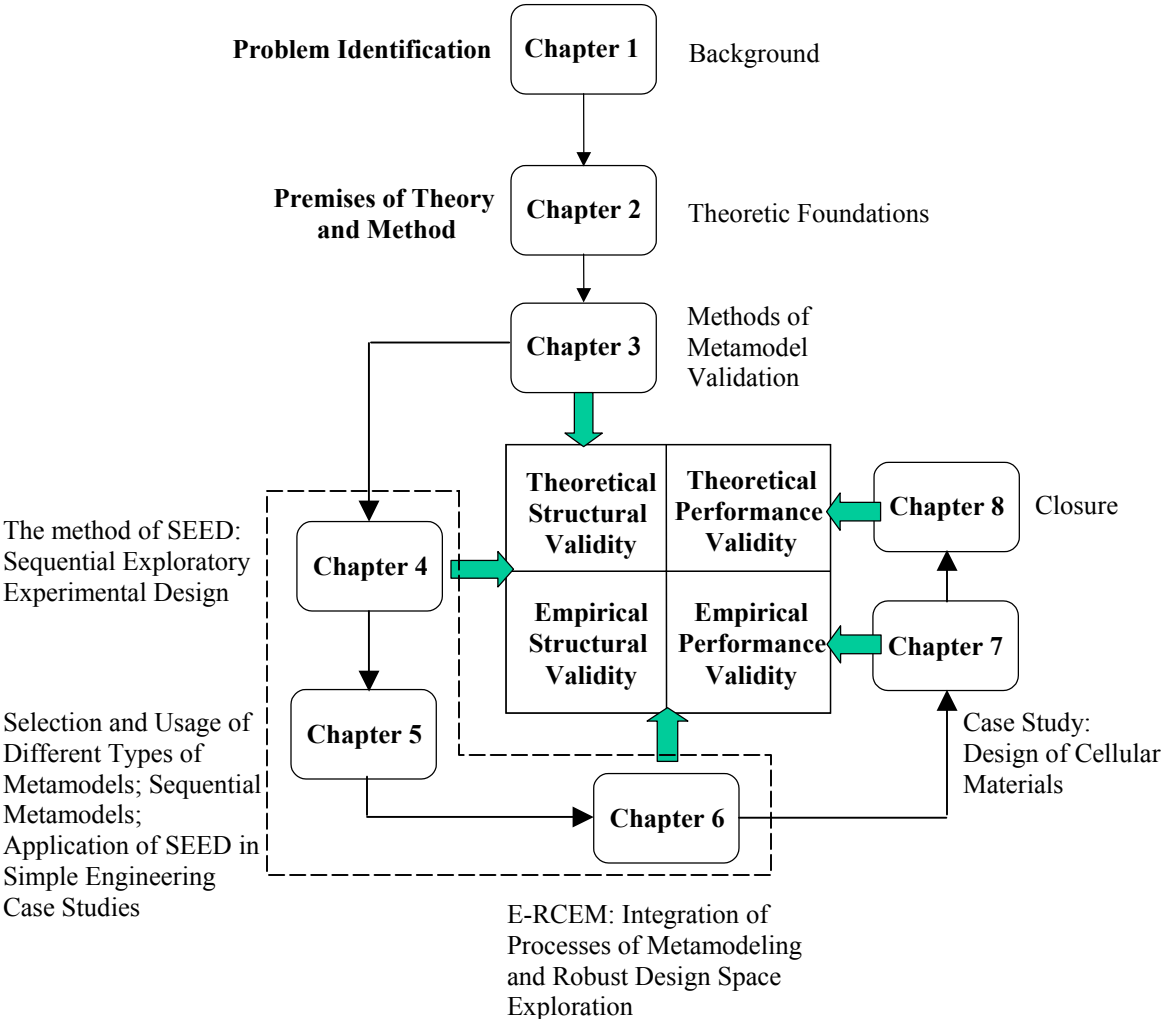


Figure 1.11 Organization of the Dissertation Based on The Validation Square

Empirical verification is established by demonstrating the accuracy and internal consistency of the data. For example, in optimization exercises, multiple starting points, active constraints and goals, and convergence can be documented to verify that the solution is stationary and robust. For any engineering model, it is important to verify that

data obtained from the model represent aspects of the real world that are relevant to the hypotheses in question. The model should react to inputs in an expected manner or in the same way that an actual system would react. Theoretical performance validity can be established by showing that the method/construct is useful beyond the example problems. This may involve showing that the problems are representative of a general class of problems and that the method is useful for these problems; from this, the general usefulness of the method can be inferred.

In Figure 1.11, an outline of the validation strategy for this dissertation is provided. It is arranged according to the validation square as described above and illustrated in Figure 1.10. An overview of this dissertation is presented in the next section.

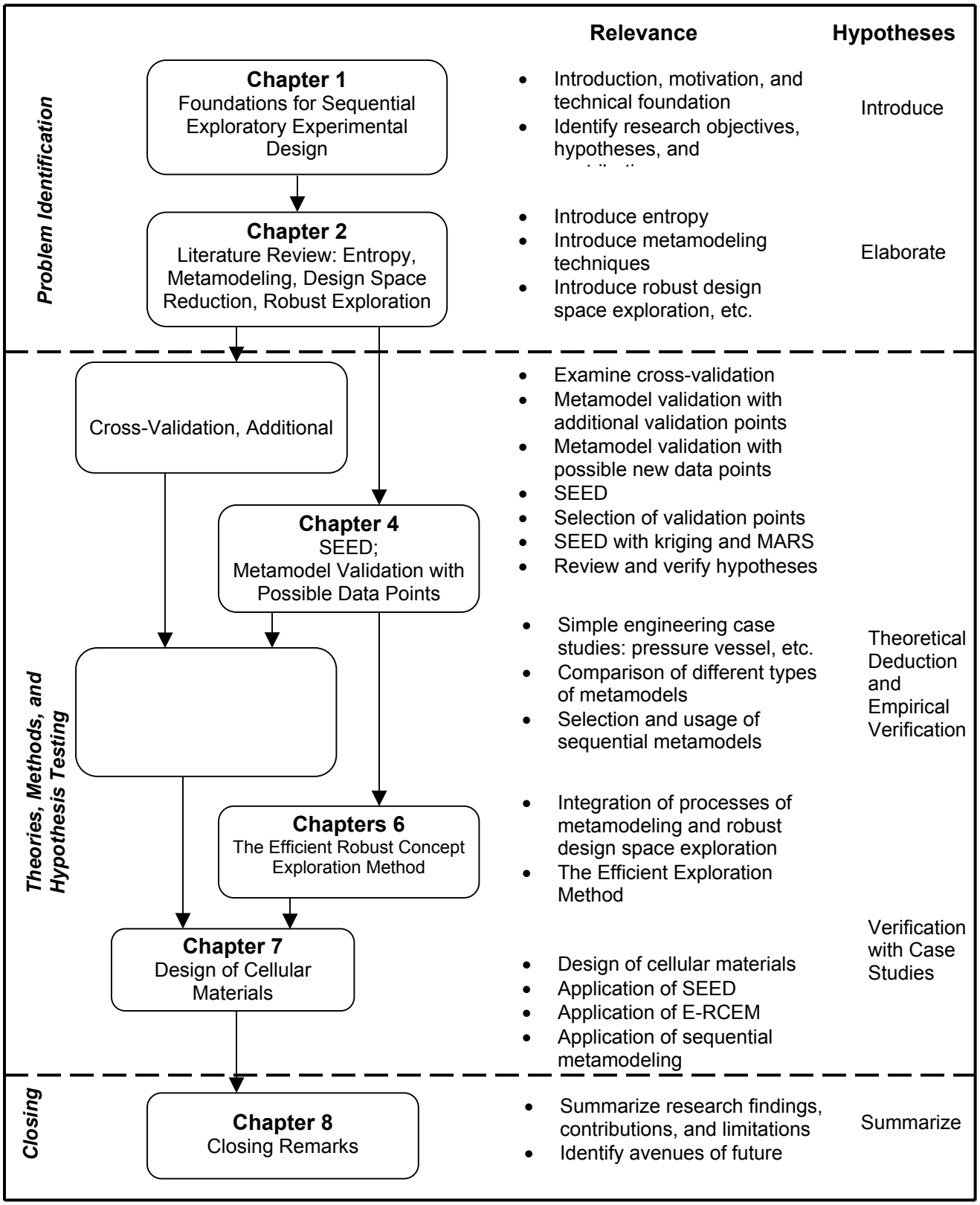


Figure 1.12 Overview of Thesis Chapters

1.5 ORGANIZATION OF THE DISSERTATION

To facilitate this discussion, an overview of the chapters in the dissertation is offered in Figure 1.12. Chapter 1 and 2 act as the first phase of the dissertation in which the background and motivation are given, and research scope is defined. Research in Chapter 3 is the foundation of research in Chapters 4, 5, and 6; thus we could view Chapter 3 as a "warm-up" chapter that provides tools and ideas for work in later chapters. Chapters 4, 5, and 6 are the heart of this dissertation in which several methods and approaches are developed for metamodeling and design space exploration. The method of Sequential Exploratory Experimental Design (SEED) is presented and verified with simple examples in Chapter 4. The approach to utilize different types of metamodels sequentially along the design timeline is proposed in Chapter 5. The new integrated process of metamodeling and robust design space exploration is described in Chapter 6. Several simple engineering case studies, e.g., the design of pressure vessels, are used in Chapter 5 and 6 to help illustrate and verify the proposed methods. Chapters 7 and 8 are validation of the proposed methods with more complicated engineering case studies, the design of cellular materials and electrical vehicle body structures, respectively. A summary of work in this dissertation is presented in Chapter 8.

Through this chapter the foundation for sequential metamodeling and robust design space exploration is introduced and the scope of our research in this dissertaion is defined. In the next chapter, our references are described in detail and the research in this dissertation is further elaborated.

CHAPTER 2

A LITERATURE REVIEW: DESIGN OF EXPERIMENTS, METAMODELING, INFORMATION THEORY, AND ROBUST DESIGN SPACE EXPLORATION

Given the research focus identified in Section 1.3, a survey of relevant work in design of experiments, metamodeling, information theory, and robust design exploration is presented in this chapter. A summary of problems and challenges in approximation-based design, an introduction of our research objectives, a description of available resources and a discussion on how the resources can be used to realize our research objectives are presented in Section 2.1. A close look at robust design space exploration is presented in Section 2.2. Taguchi's robust design method, robust design at early design stages, and the need of metamodeling are discussed. In Section 2.3 an overview of metamodeling techniques in deterministic computer experiments is presented. Then in Section 2.4 different types of metamodels are introduced and our focus is put on the regression polynomials (RS models), kriging models, and the multivariate adaptive regression splines (MARS). Various experimental designs are presented in Section 2.5; optimal experiments are introduced. Then the information theory and maximum entropy sampling are discussed in Section 2.6. Section 2.7 concludes the chapter with a summary of what has been presented and a preview of what is next.

2.1 OUR RESEARCH OBJECTIVES AND ORGANIZATION OF REFERENCES

This section is written to be a bridge between the discussion of research motivations and objectives in Chapter 1 and the introduction of references in Chapter 2. An overview of the researches in this dissertation and how these researches are done based on well-organized literature reviews is given in this section. In Section 2.1.1, the research motivations are re-emphasized with examples of problems in current research and industry. Possible ways to solve these problems are discussed in Section 2.1.2, which lead to the research questions and help define the research objectives for this dissertation. A description of references is presented in Section 2.1.3. The gap between the available knowledge and the desired metamodeling and design space exploration methods is shown in Section 2.1.4, which leads to the research questions and corresponding tasks.

2.1.1 Research Motivations: Problems and Challenges in Approximation-Based Robust Design

As described in Section 1.1, approximation-based design are introduced because the metamodels help designers 1) gain insight into the relationship between design variables and responses, 2) integrate discipline-dependent analysis codes, and 3) avoid usage of expensive analysis models. Metamodels have been widely used in early-stage design and analysis. In approximation-based design, the design effectiveness is sacrificed to gain efficiency; in other words, designers pursue the efficient exploration for

a satisficing solution instead of the expensive optimization for an optimal solution. This is the basis of the RCEM method, which is introduced in Section 1.2.2.

Although the approximation-based design strategy facilitates efficient exploration for satisficing design solutions, there are some problems that cannot be solved with current metamodeling and design space exploration techniques. As will be explained in detail, designers are strictly confined and thus real-world, industrial design applications are limited because of these unsolved problems.

Metamodels are introduced to replace expensive computer simulations or physical experiments. In the design space exploration process, designers may need to call the analysis codes many times to find the solution with an optimization algorithm. The usage of cheap-to-run metamodels facilitates the efficient examinations of response values, thus design space exploration is not an expensive process in approximation-based design. In approximation-based design, more time or money is spent on computer or physical experiments in the metamodeling process, which is the process before design space exploration in traditional design methods.

When the original computer simulation codes (or physical analysis experiments) are very computationally (or monetarily) expensive, designers cannot efficiently grasp the responses or achieve satisficing design solutions even with the assistance of metamodels. For instance, one crash simulation of a full passenger car takes 36 to 120 hours to compute, according to engineers at Ford Motor Company (Gu, 2001). In case of physical experiments, the experimental resources may be limited, e.g., engineers' access to some particular manufacturing facilities may be restricted, the experimental time may

be very long (this situation is similar to that with expensive computer simulations), materials used in the experiments may be very expensive, etc. In such cases with expensive experiments or simulations, the usage of metamodels helps greatly reduce the possible high expense in a trial-and-error approach; however, the metamodeling cost become so high that designers may not be able to develop acceptable metamodels at acceptable prices with current experimental design and metamodeling techniques. Thus, engineers may not be able to take enough information from the experiments or simulations for design. This leads to a difficult question: *how can engineers develop acceptable metamodels and achieve good design solutions at low cost when the experiments or simulations are very expensive?*

Another problem is that in multi-disciplinary, multi-variable, and multi-objective design cases, the actual responses are usually very nonlinear or irregular (which means the response is nonlinear in some regions but flat in others), and thus it may be difficult to develop acceptable metamodels with current single-stage experimental design and metamodeling techniques. For example, in Figure 2.1 we present a single-variable example in the development of a roller-warning device for road vehicles (Goldman, 2001). This single-variable simulation yields an irregular response of Load Transfer Ratio (LTR) that is represented by solid line, and the artificial neural networks metamodel is represented by the dotted line. Another single-variable example is illustrated in Figure 2.2, the suspension responses versus frequency in road profiling (Sayers and Karamihas, 1998). A two-variable example is illustrated in Figure 2.3, which is taken from the study on high-performance impact absorbing materials in

(Holnicki-Szulc, et al., 2003). In Figure 2.3, the response is the plastic-like energy dissipation, and two control variables are σ_1 and σ_2 describing the yield stresses on different elements in a structure example used in (Holnicki-Szulc, et al., 2003).

We give examples with only one or two variables and only one response in Figure 2.1, Figure 2.2, and Figure 2.3 because it is easy to illustrate the non-linearity of the response. In real-world industrial applications, which are characterized as multi-variable, multi-response, and multi-objective, it is expected that more nonlinear or irregular responses be involved in the studies and analyses. By using metamodels, engineers agree not to focus on the details of fluctuations of the responses, but try to grasp an approximated response-changing tendency with efficient and effective abstractions. However, when the responses are highly nonlinear or irregular, it is dangerous to use metamodels with low fidelity in early-design stages because designers may be led to a totally wrong direction. Even a very small error in early design stages may evolve to a huge mistake and result in expensive re-design processes. Thus in design, we suggest development of "acceptable" metamodels, which means that the metamodels are accurate enough to reflect major changes of the responses in the design space, and on the other hand, smooth (or "abstract") enough to ensure low cost in the metamodeling and design space exploration processes. Thus another difficult question is posed: *how can engineers develop acceptable metamodels and achieve good design solutions with low cost when the actual responses are highly nonlinear or irregular?*

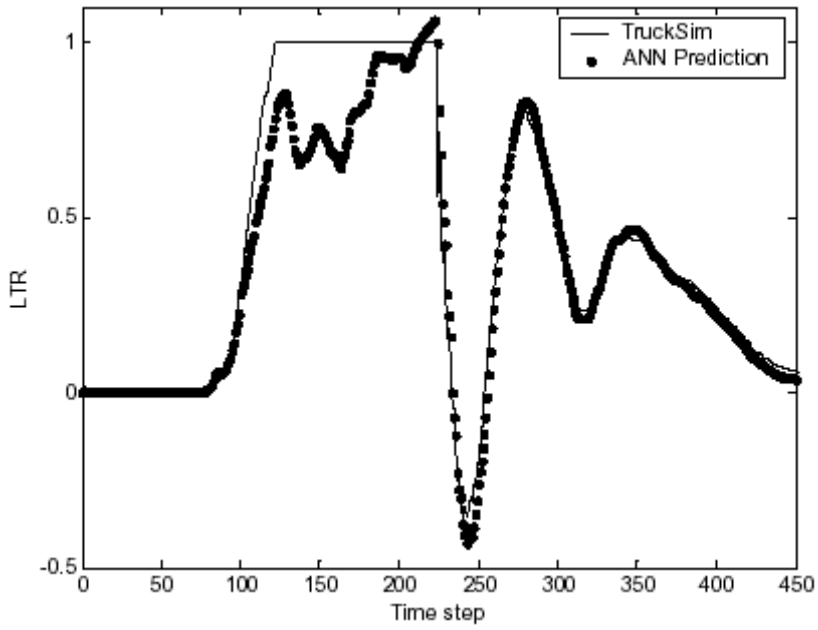


Figure 2.1 LTR Prediction for Maneuver ST Performed at 60 km/hr Using a 2-2 ANN (adapted from Goldman, 2001)

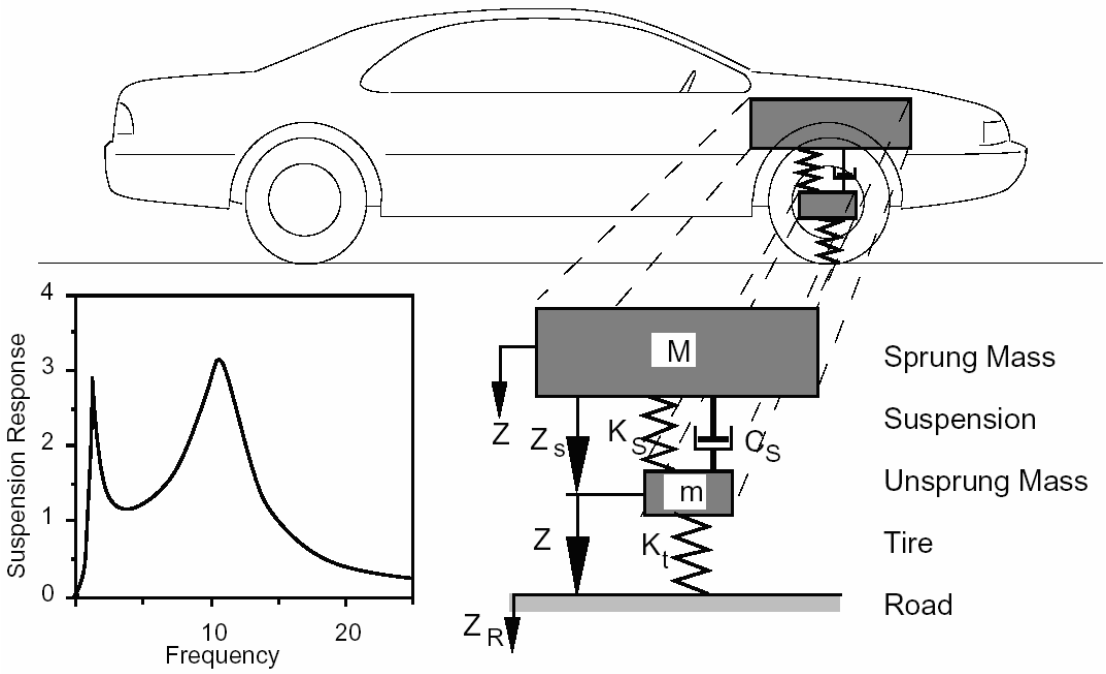


Figure 2.2 Suspension Response versus Frequency in Road Profiling (adapted from Sayers and Karamihas, 1998)

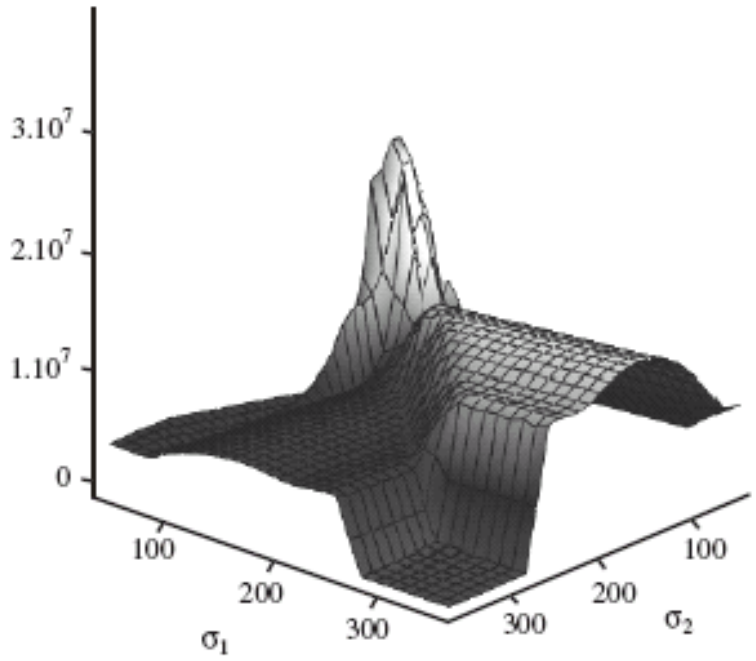


Figure 2.3 Energy Dissipation for Various Yield Stress Values (adapted from Holnicki-Szulc, et al., 2003)

In summary, it is difficult to utilize the approximation-based design strategy in industrial applications because:

- In industrial applications, the experiments or simulations can be very expensive. The metamodeling and design space exploration expense will be too high to afford with current single-stage experimental design techniques.
- In industrial applications, the responses are usually nonlinear or irregular. To develop acceptable metamodels and achieve good design solutions, current metamodeling and design space exploration techniques require observations at a lot of data points, and thus result in a high expense that may be unacceptable.

Without satisfactory answers to the two questions posed above, engineers cannot fully utilize an approximation-based design strategy in real-world industrial applications. Designers' freedom is strictly restricted, as listed below:

- To avoid expensive physical experiments, engineers have to develop computer simulations to do analyses. This should be encouraged because it represents the trend in the computerized world. However, not all physical experiments can be replaced by computer simulations; the computer analysis model may be inaccurate or even totally wrong when we do not fully understand the system that we are modeling (usually it is because that the system is too complicated, or the theory to describe and explain the system is incorrect). Since engineers cannot afford the physical experiments with current metamodeling and design space exploration techniques, they have to work with the inaccurate (or even incorrect) computer analysis models because they cannot do the expensive physical experiments at low expense.
- Since multiple runs of the expensive experiments or simulations cannot be afforded, engineers may tend not to use these analyses in early-stage design. Then the design becomes experience-based because the achievement of good design solutions is mainly dependent on the designers' knowledge. The expensive experiments or simulations are only used as validation tools in very late design stages. With such a design strategy, expensive re-design is very likely to take place, and the time-to-market is greatly increased.

- Engineers are advised not to develop expensive computer simulations in order to avoid intensive computation loads. Usually more abstractions and assumptions are made in the development of inexpensive simulations than that of expensive ones, and as a result, the inexpensive simulations can be very inaccurate. To use such analysis codes designers may not be able to capture important response properties in the design space, and thus are led to wrong directions; this will result in re-design and increased time-to-market.
- In order to reduce the metamodeling and design space exploration expense, designers usually choose a very small design space to ensure that the responses in this design space are not highly nonlinear or irregular. The selection of this design space is mainly based on designers' experience. This strategy confines designers' freedom to explore a large design space, and cannot ensure a good design solution.

2.1.2 Research Objectives

As discussed in Section 2.1.1, engineers cannot fully utilize an approximation-based design strategy (experiments or simulations, metamodels, and exploration of the design space) in early design stages of real-world industrial applications because of the high experimental expense and irregular (or highly nonlinear) responses. Without available methods and tools to address these concerns, engineers have to circumvent these problems in design. This confines designers' freedom; it is very likely that expensive re-design process will take place and the time to market will be increased.

To save time and money spent on expensive physical experiments or computer simulations, researchers proposed a lot of methods or heuristics to reduce the design space or facilitate sequential metamodeling processes in a fixed design space. The aim of these methods is to reduce the number of total observations or locate data points in "meaningful" regions through intermediate analyses of the response surfaces in the design space.

To reduce the design space, designers can either screen out unimportant design variables or reduce the ranges of design variables. The identification and elimination of unimportant design variables is an important step in the traditional Response Surface Methodology (RSM) (Myers and Montgomery, 1995). Other methods are also developed to reduce the dimensionality (e.g., see Box and Draper, 1969; Balabanov, et al., 1999; Giunta, et al., 1996; Welch, et al., 1992). More research is done to reduce the ranges of design variables. Chen and her co-authors developed heuristics to lead the surface refinement to a smaller design space (Chen, et al., 1997). The adaptive RSM (ARSM) method is developed to systematically reduces the size of the design space by discarding portions of it that correspond to objective function values larger than a given threshold value at each modeling-optimization iteration (Wang, 2001; Wang, 2003). Move limit strategies or trust regions are often used to identify "meaningful" design spaces (Wujek and Renaud, 1998a; Wujeck and Renuad, 1998b; Alexandrov, et al., 1998; Rodriguez, et al., 1997). Wang and Simpson propose an intuitive methodology to systematically reduce the design space to a relatively small region by incorporating the fuzzy c-Means clustering technique in the metamodeling process (Wang and Simpson, 2004).

In this dissertation, we develop methods that do not adopt the design-space-reduction strategy. Instead, we focus on problems in which the design space is fixed and unchanged during the design process. In such cases, strategies are expected to locate data points sequentially in the design space. With the sequential experimental design strategy, information from previous points and metamodels are utilized in identifying new points; new points are located at "critical" places. Such a strategy helps obtain maximum possible information with limited resources, and thus achieve good design solutions with acceptable computational or monetary expense. This helps answer the two questions posed in Section 2.1.1, *how can engineers develop acceptable metamodels and achieve good design solutions at low cost when the experiments or simulations are very expensive?* And, *how can engineers develop acceptable metamodels and achieve good design solutions with low cost when the actual responses are highly nonlinear or irregular?*

To address the problems as discussed in Section 2.1.1, in this dissertation we propose a systematic yet flexible method in which various metamodeling techniques are utilized in building series of appropriate metamodels for robust design space exploration in accordance with the change in information along the design timeline at the early stages of design. With this method designers are able to design sequential experiments and thus develop more accurate metamodels and achieve better design solutions with limited resources. This will give designers the freedom of utilizing expensive experiments or simulations in studies with a large design space in early design stages.

To develop the proposed method, we need to accomplish four tasks:

- Study measures for metamodel validation. This corresponds to Research Question 1 of this dissertation. After accomplishing this task we are able to judge whether a metamodel is **acceptable** or not. To accomplish this task we need to study existing metamodel validation approaches, especially in cases with deterministic computer experiments.
- Develop methods for sequential experimental design in fixed design spaces. This corresponds to Research Question 2. Criteria and tools are needed to define the "**potential information**" and identify "**critical**" regions. Then an algorithm needs to be developed to locate new points in "critical" regions that bring maximum "potential information".
- Study the integration of metamodeling and design space exploration within a fixed design space. This corresponds to Research Question 3. Through the integration of the metamodeling and design space exploration processes, we consider another criterion for "**critical**" regions – the achievement of design goals. The algorithms developed in answering Research Question 2 are further improved in this research.
- Develop methods for model selection along the design timeline. This corresponds to Research Question 4. This supports the methods developed for Research Questions 2 and 3. Metamodel **comparison** and **selection** are needed to ensure that acceptable metamodels can be developed with sequential experimental design methods.

Research objectives in this dissertation are to develop methods with which we are able to answer the 4 research questions presented in Chapter 1. The proposed methods should provide engineers with maximum freedom in the design process, help obtain maximum design information and knowledge with limited resources, facilitate efficient and effective metamodeling and design space exploration processes, reduce the possibility of re-design, and thus decrease the time-to-market of new products.

2.1.3 Organization of References

In Section 2.1.2 we described our research objectives, which is to develop systematic yet flexible methods to facilitate sequential experimental designs, with which engineers are able to develop acceptable metamodels for irregular responses and achieve satisficing design solutions with limited resources in early design stages. Such methods are developed based on previous research, incorporating ideas and tools from various fields, such as design of experiments, statistical modeling, information theory, and decision support problems.

The proposed methods in this dissertation are developed based on the Robust Concept Exploration Method (RCEM), which is introduced in Section 1.2.2. Robust design space exploration in RCEM, which is realized by incorporating Taguchi's robust design, statistical metamodeling, and the compromise DSP (Section 1.2.1), provides the framework for the proposed methods in this dissertation. The robust design space exploration is introduced in Section 2.2.

It has been shown in previous studies that some types of metamodels are theoretically appropriate for deterministic computer experiments, while others are not.

Some metamodel validation criteria used in physical experiments do not have statistical meanings with computer experiments. To compare and validate the accuracy of metamodels in such cases, leave-one-out cross-validation errors are widely used; however, previous research suggests that alternate criteria be used because it is empirically proved that leave-one-out cross-validation errors do not correlate with the true prediction error. This leads to the studies for Research Question 1, metamodel evaluation. The deterministic computer experiments and metamodel validation techniques are introduced in Section 2.3.

The development of various types of metamodels is important in the approximation-based design strategy. In this dissertation, we will study and use three types of metamodels, the response surface metamodel, kriging, and the multivariate adaptive regression splines. This study helps answer Research Question 4, metamodel comparison and selection. Different types of metamodels are introduced in Section 2.4.

The sequential experimental design method is developed based on maximum entropy sampling (which is an application of the information theory), which is actually a D-optimal design. Designs of experiments are introduced in Section 2.5, with emphasis on D-optimal experiments. The information theory and the maximum entropy sampling method are introduced in Section 2.6. This, together with the compromise DSP, provides the necessary basis for answers to Research Questions 2 and 3.

2.1.4 Organization of Research Questions: Removing Gaps Between Available Resources and Proposed Design Space Exploration Methods

After introducing the motivations in Section 2.1.1, research objectives in Section 2.1.2, and existing technical resources in Section 2.1.3, the gaps between existing technical resources and the research objectives are discussed in this section, which lead to the research questions of this dissertation.

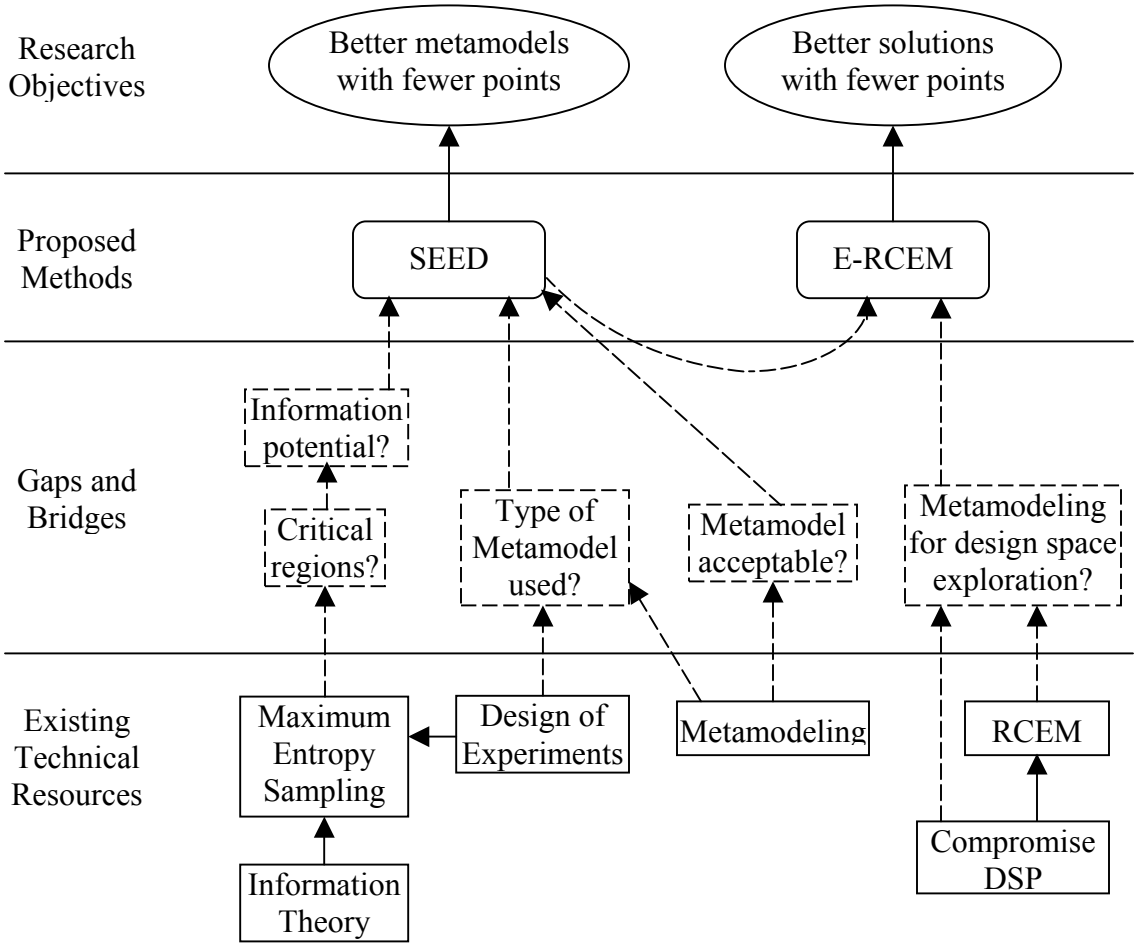


Figure 2.4 Gaps and Bridges between Research Objectives and Existing Technical Resources

The gaps between existing technical resources and research objectives are illustrated in Figure 2.4. The research objectives of this dissertation are to develop methods that facilitate efficient and effective approximation-based robust design, which are represented by ovals at the top of Figure 2.4. The research objectives are achieved by the development of two methods, SEED and E-RCEM, in this dissertation. As introduced in Section 2.1.3, the available technical resources are RCEM, C-DSP, DOE (Design of Experiments) techniques, Metamodeling Techniques, the information theory, and maximum entropy sampling, which are presented in boxes at the bottom of Figure 2.4. The gaps between existing techniques and the research objectives are illustrated with dashed boxes and arrows in Figure 2.4, which can also be viewed as bridges connecting the "known" and "unknown". The gaps (or bridges) are:

- Approaches to validate metamodel accuracy. Leave-one-out cross-validation is widely used to test the accuracy of metamodels; however, previous empirical studies shown that it may not be appropriate with deterministic computer experiments. A theoretical study of leave-one-out cross-validation in metamodel validation, and the development of appropriate metamodel validation approaches is necessary for the development of sequential metamodeling and design space exploration methods. This leads to Research Question 1.
- A method to reflect and utilize information during the metamodeling process. In order to save time and money spent on expensive experiments, a sequential experimental design strategy is necessary in which information from previous

observations can be used as guidance in selecting new data points. To develop such a method, we need to identify "critical regions" and evaluate the "information potential" of points. This can be achieved through the utilization of the information theory and maximum entropy sampling techniques. This corresponds to Research Question 2.

- The consideration of design goals in the metamodeling process. The integrated design process of metamodeling and design space exploration helps achieve better design solutions faster; to realize this an algorithm is needed to incorporate design goals in metamodeling. This algorithm can be developed based on the compromise DSP and the SEED method. This leads to Research Question 3.
- Utilization of appropriate types of metamodels. To identify and use the appropriate type of metamodels is important in the application of SEED and E-RCEM; thus a study is needed on the comparison and selection of different types of metamodels in sequential metamodeling and design space exploration. This leads to Research Question 4.

The organization of references, research questions (the gaps), and proposed studies and methods is illustrated in Figure 2.5. The proposed methods and studies, which are what we want to achieve in this dissertation, is illustrated at the top of Figure 2.5 (above two dashed lines). The existing technical resources are presented at the bottom of Figure 2.5 (below two dashed lines). The gap between the available resources

and the desired achievements is reflected by the research questions, which are listed between two dashed lines in Figure 2.5.

In Figure 2.5 the proposed studies and methods in this dissertation are illustrated as four ovals on the top, which stand for metamodel evaluation, metamodel comparison, sequential experimental design, and integration of design processes, respectively. The references, which are resources we have with existing techniques, are shown in rectangles at the bottom. Research Questions 1, 2, 3, and 4 provide the link between the existing techniques and proposed methods and studies. To answer Research Question 1, computer experiments and leave-one-out cross-validation are studied. To develop a sequential experimental design method (the SEED method), we need to answer Research Question 2; R.Q. 2 is answered based on studies of Design of Experiments, D-Optimal Design, Information Theory and Entropy, and Maximum Entropy Sampling. The comparison and selection of metamodels in design are based on the knowledge of various types of metamodels. The integration of design processes is realized by answering Research Question 4; the compromise DSP plays an important role in the incorporation of design goals and constraints in the metamodeling process.

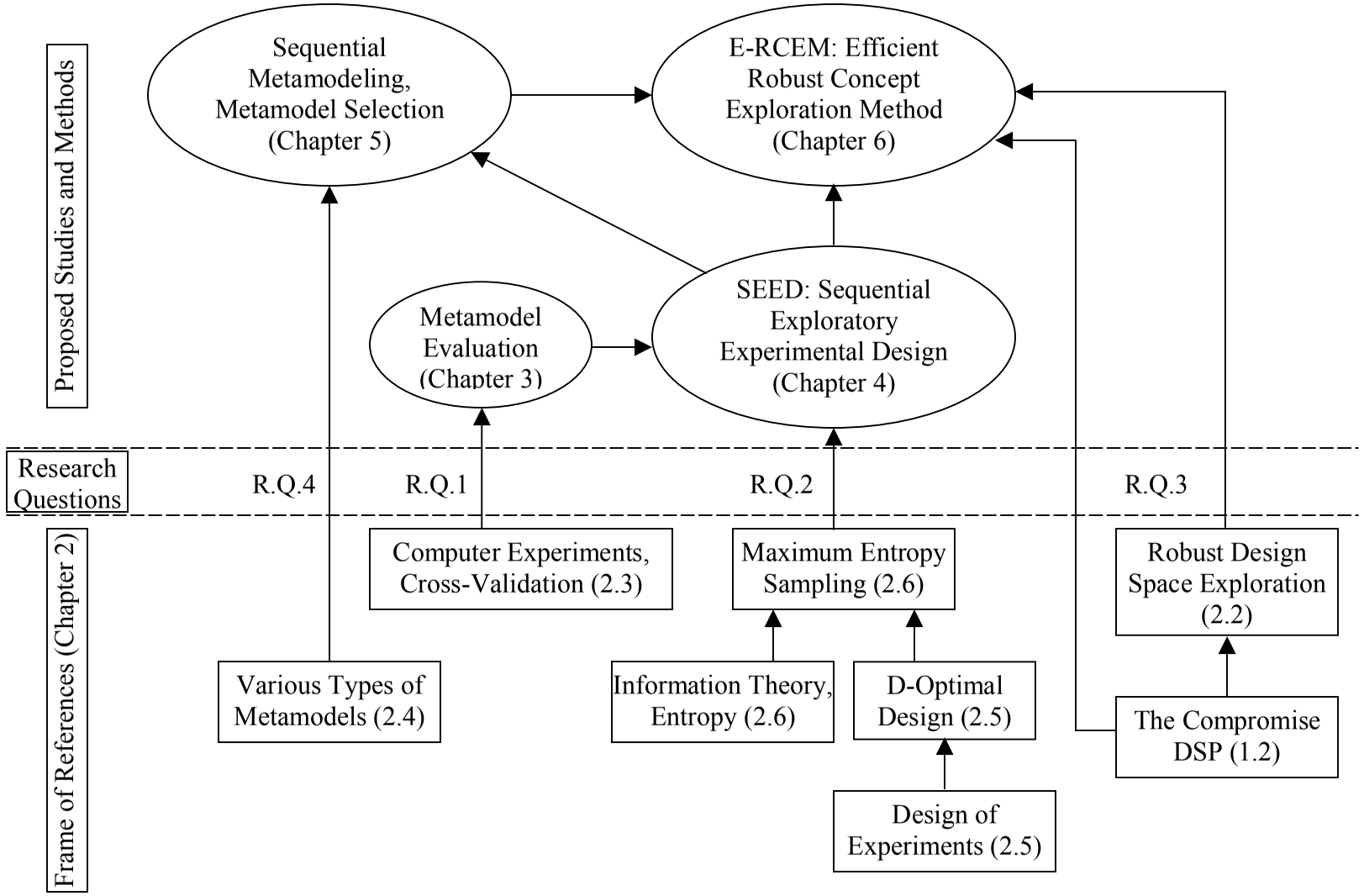


Figure 2.5 Organization of References

2.2 ROBUST DESIGN SPACE EXPLORATION

The fundamental motivation underlying robust design, as originally proposed by Taguchi, is to improve the quality of a product or process not only by striving to achieve performance targets but also by minimizing performance variation. Taguchi's methods have been widely used in industry, generally applied in the later stages of design to implement parameter design and tolerance design (see, e.g., Byrne and Taguchi, 1987; Phadke, 1989; Ross, 1988). Reviews of such applications are found in (e.g., Nair, 1992).

In robust design, the relationship between different types of design parameters or factors are represented with a P-diagram as shown in Figure 2.6 where P represents either product or process (Phadke, 1989). The three types of factors which serve as inputs to the P-diagram and that influence the (output) response $\boldsymbol{y}$ are:

- *Control Factors* ($\boldsymbol{x}$) – parameters which can be specified freely by a designer; the settings for the control factors are selected to minimize the effects of noise factors on the response y. It is a designer's responsibility to determine the best values for these parameters.

- *Noise Factors* ($\boldsymbol{z}$) – parameters not under a designer's control or whose settings are difficult or expensive to control. Noise factors cause the response, $\boldsymbol{y}$, to deviate from their target and lead to quality loss through performance variation. Noise factors may include system wear, variations in the operating environment, uncertain design parameters, and economic uncertainties.

- *Signal factors* (***M***) – parameters set by the designer to express the intended value for the response of the product; signal factors are those factors used to adjust the mean of the response but which no effect on the variation of the response.

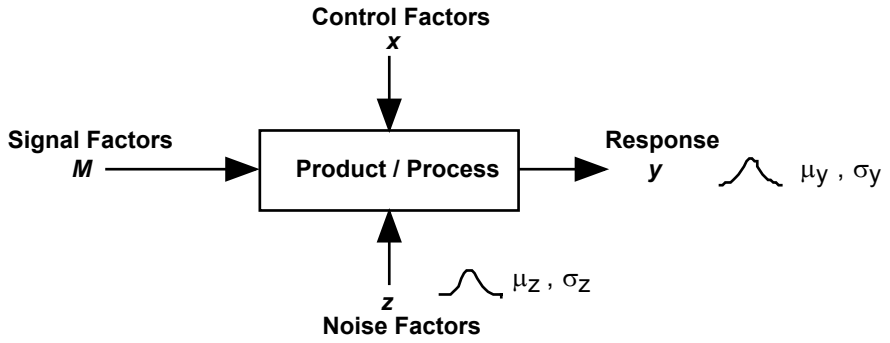


Figure 2.6 P-Diagram of a Product/Process in Robust Design (adapted from Phadke, 1989)

The terminology of robust design is used to classify design parameters and responses and to identify sources of variability. The objective in robust design is to reduce the variation of system performance caused by uncertain design parameters, thereby reducing system sensitivity. Variations in noise factors, shown in Figure 2.6 as normally distributed with mean μ_z and standard deviation σ_z, lead to variation in performance responses, also represented in Figure 2.6 as normally distributed with mean μ_y and standard deviation σ_y. In robust design, *solutions* (represented through settings of the control factors) are usually sought that minimize response variation in addition to achieving performance targets (mean, μ_y, on target, ***M***). In taking this approach, robust solutions obtained for complex systems involving significant uncertainty (or noise) are

usually not "optimal" in the traditional sense, but satisficing (Simon, 1982). When building approximate system models based on data obtained from statistical experimentation, models are required for the mean, μ_y, and standard deviation, σ_y, of each response.

In robust design space exploration, we aim at identifying robust design solutions at early design stages through the development of metamodels and trade-off among design goals. We could achieve this with the robust concept exploration method (RCEM), which is the hybrid of several methods and tools – robust design methods, the response surface methodology (metamodeling techniques), Suh's design axioms, and the compromise DSP. In Section 2.1.1, Taguchi's robust design is introduced and its limitation pointed out. Implementations of robust design at early design stages for large-scale engineering systems are presented in Section 2.1.2. Robust design space exploration is the context for research in this dissertation.

2.2.1 Taguchi's Method

What is of interest is Taguchi's definition of the "goodness" of a design. Whereas various other approaches assume that a good design meets a set of well-defined functional, technical performance, and cost goals, Taguchi states that a good design minimizes the quality loss over the life of a design. In Taguchi's method the quality loss is defined as the deviation from desired performance (Phadke, 1989; Ross, 1988; Taguchi, 1978; Taguchi, 1987; Taguchi, et al., 1989).

Based on the concept that loss is incurred when a product's functional quality characteristic deviates from its target value regardless of the amount of deviations, the

quality loss is measured based using the quadratic loss function as shown in Figure 2.7. As stated in (see, e.g., Phadke, 1989; Ross, 1988), the quality loss for being off-target by means of a quadratic quality loss function can be represented as:

$$L(y) = k\,(y - T)^2\,, \tag{2.1}$$

where

- y is the quality characteristic of a product/process,
- T is the target value for y, and
- k is a constant, the quality loss coefficient.

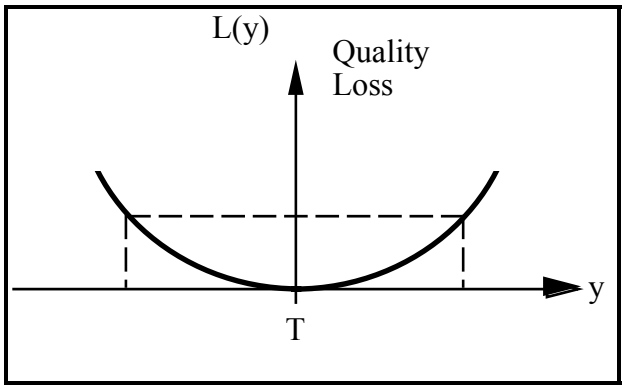


Figure 2.7 Quadratic Loss Function

Under this description, to maximize the quality the loss must be zero. The greater the loss, the lower quality. The quality loss is zero at y = T in Figure 2.7 and increases slowly near T but more rapidly farther from T. Equation (2.1) is the simplest mathematical equation exhibiting this behavior and the constant k in it must be determined to make the equation best approximates the actual loss in the region of interest.

Using Taguchi's robust method, a designer is concerned with the sensitivity of a design to uncontrollable factors that may be encountered in both manufacturing and use.

Under robust design considerations, noise factors cause the response, ***y***, to deviate from a target specified by a signal factor, ***M***, and therefore lead to quality loss. The objective of robust design is to choose the levels of control factors to dampen the variation of responses according to the criterion for robust ness criteria, e.g., "the target is best", "the larger the better", and "the smaller the better". Taguchi states the *parameter design concept* as that the fundamental principle of robust design is to improve the quality of a product by minimizing the effect of the causes of variation without eliminating the causes.

Design of experiments, specifically orthogonal arrays (OA), are typically employed in Taguchi's robust design method to systematically vary and test the different levels of each of the control factors. Taguchi advocates the use of an *inner-array* and *outer-array* approach to implement robust design (e.g., Byrne and Taguchi, 1987). The inner-array consists of an OA which contains the control factor settings; the outer-array consists of the OA which contains the noise factors and their settings which are under investigation. The combination of the inner-array and outer-array constitutes what is called the *product array*. The product array is used to systematically test various combinations of the control factor settings over all combinations of noise factors after which the mean response and standard deviation may be approximated *for each run* using the equations:

- Response mean: $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$

- Standard deviation: $S = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \bar{y})^2}{n-1}}$

Preferred parameter values can then be determined through analysis of the signal-to-noise (SN) ratio; factor levels that maximize the appropriate SN ratio are optimal. As stated in previous paragraphs, there are three "standard" types of SN ratios (see, e.g., Phadke, 1989):

- Nominal the best (for reducing variability around a target):

$$SN_T = 10 \log\left(\frac{\bar{y}^2}{S^2}\right) \tag{2.2}$$

- Smaller the better (for making the system response as small as possible):

$$SN_L = -10 \log\left(\frac{1}{n}\sum_{i=1}^{n}\frac{1}{y_i^2}\right) \tag{2.3}$$

- Larger the better (for making the system response as large as possible):

$$SN_S = -10 \log\left(\frac{1}{n}\sum_{i=1}^{n} y_i^2\right) \tag{2.4}$$

Once all of the SN ratios have been computed for each run of an experiment, there are two common options for analysis: Analysis of Variance (ANOVA) and a graphical approach. ANOVA can be used to determine which factors are statistically significant and the appropriate setting for each. The graphical approach is an alternative approach in which the SN ratios and average responses are plotted for each factor against its levels. The usual approach, then, is to examine the graphs and "pick the winner," i.e., pick the

factor levels which (1) best maximize SN and (2) bring the mean on target (or maximize or minimize the mean, as the case may be).

After the foundation of robust design by Taguchi, robustness has been taken as a design criterion to improve the qualities of both product and design process. Pignatiello provides a comprehensive review of the Taguchi Method and summarizes ten triumphs and tragedies (Pignatiello and Ramberg, 1991) and those relevant to engineering design practices are listed here:

- Taguchi helps industries to reduce the cost and improve a product's quality using the robust design concept.
- Taguchi brings the consideration of sensitivity analysis into the stage when an optimization problem is formulated.
- DOE techniques and many other statistical methods have become more and more widely used in the engineering design field with the promotion of Taguchi.

There are many criticisms of Taguchi's implementation of robust design through the inner and outer array approach (Montgomery, 1991; Nair, 1992; Otto and Antonsson, 1993; Shoemaker, et al., 1991; Tribus and Szonyi, 1989; Tsui, 1992). Consequently many variations of the Taguchi method have been proposed and developed; many researchers advocate modifications within the framework defined by Taguchi. Ramakrishnan and Rao (1991) formulate robust design as a nonlinear optimization problem using Taguchi's loss function as the objective. Sundaresan and co-authors (1993) incorporate a Sensitivity Index (SI) in the optimization procedure to determine a robust optimum. Otto and Antonsson (1993) argue the necessity of incorporating

constraints in robust design. Parkinson and co-authors (1993) propose including feasibility robustness as an important robust design category. Su and Renaud (1996) provide an in-depth review of several different robust optimization techniques based on the Taguchi method and investigate the computational costs associated with implementing them. Simpson and co-authors (1997c) give an extensive review of robust design formulations and use design capability indices to satisfy a "ranged set of requirements". Review of numerous robust design optimization methods can also be found in (Simpson, et al., 1997b; Tsui, 1992; Yu and Ishii, 1998). In the next section, our approach of robust design at early design stages for large-scale systems is presented.

2.2.2 Robust Design in the Early Design Stages

If we can model a concept variant in the conceptual design phase of a product, then we can implement Taguchi's robust design methods in the early stages of design. To accomplish this though the model must represent a good approximation of the real life product because it is necessary to have clearly defined target values that must be met for the product to be robust. The objective of abstracting robust design to early design stages is accomplished by integrating Taguchi's principles with response surface methodology and the compromise DSP which is elaborated in Section 1.2.1. As introduced in Section 1.2.2, the RCEM is developed to facilitate robust design of large-scale complex engineering systems at early design stages. To facilitate the implementation of robust design within the RCEM, second-order response surface models are created and used to approximate the design space, replacing the computer analysis code or simulation routine used to model the system. The major elements of the response surface model approach

for robust design applications are (see, e.g., Myers and Montgomery, 1995; Shoemaker, et al., 1991):

- Combining control and noise factors in a single array instead of using Taguchi's inner- and outer-array approach,
- Modeling the response itself rather than expected loss, and
- Approximating a prediction model for loss based on the fitted-response model.

Instead of using Taguchi's orthogonal array as the combined array for experiments, central composite designs are employed in the RCEM to fit second-order response surface models for integration with Taguchi's robust design. From the response surface model, it is possible to estimate the mean and variance of the response. The central composite design and the response surface model will be presented in detail as useful metamodeling techniques in following sections of this chapter.

While Taguchi's method is generally applied in later stages of design, we propose to extend considerations of robustness to the early design stages to help both increase the products' quality and reduce time to market. With RCEM we are able to measure the capability of meeting the specified range of overall design requirement in the concept exploration process where there are varying design parameters. It has been suggested by a number of researchers that separate goals be modeled for the response mean and variance in a robust design formulation (e.g., Chen, 1995; Chen, et al., 1996a). Their robust design methods can be represented as achieving the following goals simultaneously at early design stages:

(i) *The goal for the response mean:* Optimize (minimize or maximize) Mean, *or* Bring Mean on Target and

(ii) *The goal for response variance:* Minimize Variance (at the point under study).

In brief these goals can be stated as "bringing the mean on target" and "minimizing the deviation". To achieve these goals simultaneously a trade-off is necessary. This is accomplished with the compromise Decision Support Problem (C-DSP) (Mistree, et al., 1993b).

In an effort to generalize robust design for product design, two broad categories, or types, of robust design based on the source of variation are identified:

- Type I Robust Design: *minimizing variations in performance caused by variations in noise factors (uncontrollable parameters).*
- Type II Robust Design: *minimizing variations in performance caused by variations in control factors (design variables).*

Although the concepts behind the two major types of robust design are quite different, robust design is always concerned with aligning the peak of the bell shaped response distribution with the targeted quality (bringing the mean to the target), and making the bell shaped curve thinner (reduce the deviation). The two types of robust design are similar in that they both explore for a *flat* (or nearly *flat*) region (Chen, et al., 1996b).

The logic behind the two major types of robust design applications is illustrated in Figure 2.8 (Chen, et al., 1995). On the left-hand side of Figure 2.8, a P-diagram (Phadke, 1989) is used to represent different types of parameters in robust design, their

relationships with the whole system, and thus the differences in source of variation in response for Type I and Type II applications. As stated before, *Control factors (x)* are parameters which can be specified freely by a designer; *noise factors (z)* are parameters that are not under the control of a designer; and the *signal factor (M)* is the intended value for the *response (y)* of a product/process. In Type I applications, the deviation of the response is caused by variations in the noise factor, z, the uncontrollable parameter. Type II is different from Type I in that its input does not include a noise factor. The variation in performance is caused solely by variations in control factors or design variables in the region ($\pm\Delta x$).

As described in (Chen, 1995), on the right hand side of the figure is a schematic of the different concepts behind the two types of robust design. Taguchi's robust design method deals with only the Type I robust design. Type I robust design is highlighted in the upper right block of Figure 2.8. Basically, in the Taguchi method, a designer adjusts control factors, *x,* to dampen the variations caused by the noise factor, *z*. The two curves represent the performance variation as a function of noise factor when *x* is at two different levels, x = a and x = b. If the design objective is to achieve a performance as closely as possible to the target, *M*, the designs at both levels are acceptable because their means are the target *M*. However, introducing *robustness*, when x = a, the performance varies significantly with the deviation of noise factor, *z*; however, when x = b, the performance deviates much less. Therefore, x = b is more robust than x = a as a design solution because x = b dampens the effect of the noise factors more than x = a.

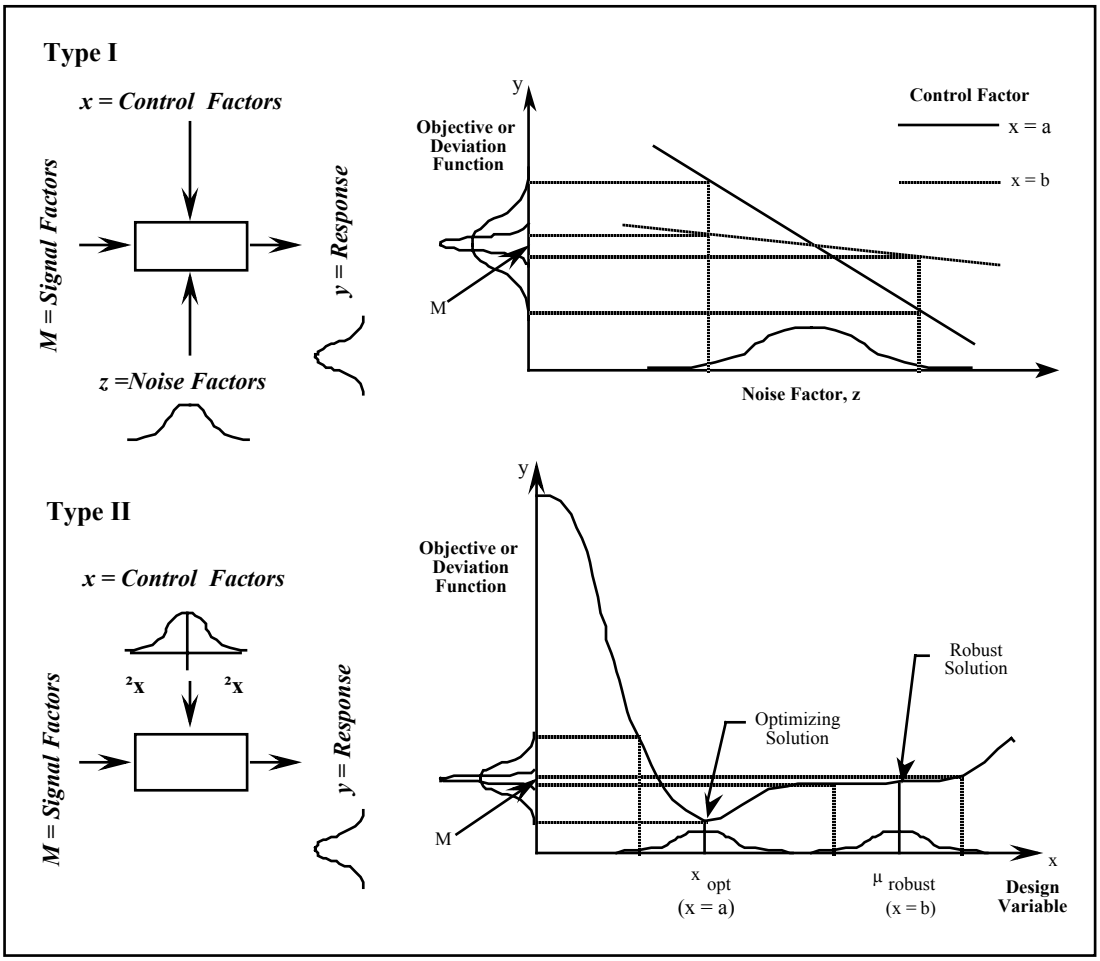


Figure 2.8 A Comparison of Two Types of Robust Design (Chen, et al., 1995)

The logic behind Type II robust design is represented in the lower right block of Figure 2.8. For purposes of illustration, assume that performance is a function of only one variable, *x*. In general, for this type of robust design, to reduce the variation of the response caused by the deviations of design variables, instead of seeking the peak or optimum value, a designer is interested in the flat part of a curve near the performance target. It is in this manner that *robustness can affect the compromise DSP*, as stated before. If the objective is to move the performance function towards target M and if a

robust design is not sought then obviously the point x = a is chosen. However, for a robust design, x = b is a better choice. This is because if design variables vary within the region $\pm\Delta x$ of their means, the resulting variation of response of the design at x = b is much smaller than that at x = a, while the means of the response at two designs are close. *The robust solution, x = b, is more desirable since it helps bring the mean responses of the system into the target values and minimizes deviation, which is a very important factor when solving the compromise DSP for multiple responses.*

In the next section, an overview of metamodeling techniques in deterministic computer experiments is presented. Then different metamodels, design of experiments, and their application in engineering design cases are discussed in following sections.

2.3 METAMODELING TECHNIQUES AND DETERMINISTIC COMPUTER EXPERIMENTS

As stated in Section 1.1.1, much of today's engineering analysis work consists of running complex computer codes – supplying a vector of design variables (inputs) $\boldsymbol{x}$ and receiving a vector of responses (outputs) $\boldsymbol{y}$. The expense of running many of these codes remains non-trivial despite continual advances in computing power and speed. Single evaluations of aerodynamic or finite-element codes can take from minutes to hours, if not longer. Furthermore, this mode of query-and-response often leads to a trial and error approach to design, an iterative spiral compounded by the requirements flowdown and feedback necessary in large-scale complex systems design. Thus a designer may never

uncover the functional relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$ and therefore may never identify the best settings for the input values.

Statistical techniques are widely used in engineering design to address these concerns. The basic approach is to construct *approximations* of the analysis codes that are much more efficient to run and that yield insight into the functional relationship between $\boldsymbol{x}$ and $\boldsymbol{y}$. This is where the approximation-based robust design comes from. To facilitate the implementation of robust design, metamodeling techniques are often employed to create approximations of the mean and variation of a response in the presence of noise. A metamodel is a "model of a model" (Kleijnen, 1987) which is used as a surrogate approximation for the actual analysis (i.e., computer code) during the design process. The general approach to response surface modeling is shown in Figure 2.9. In statistical terms, design variables are factors, and design objectives are responses; the factors and responses to be investigated for a particular design problem provide the input for the approach of Figure 2.9, and the solutions (improved or robust) are the output. To identify these solutions, this approach includes three sequential stages: screening, modeling building, and model exercising.

The first step (screening) is employed only if the problem includes a large number of factors (usually greater than 10); screening experiments are used to reduce the set of factors to those that are most important to the response(s) being investigated. Statistical experimentation is used to define the appropriate design analyses which must be run to evaluate the desired effects of the factors. Often two level fractional factorial designs or

Plackett-Burman designs are used for screening (Myers and Montgomery, 1995), and only main (linear) effects of each factor are investigated.

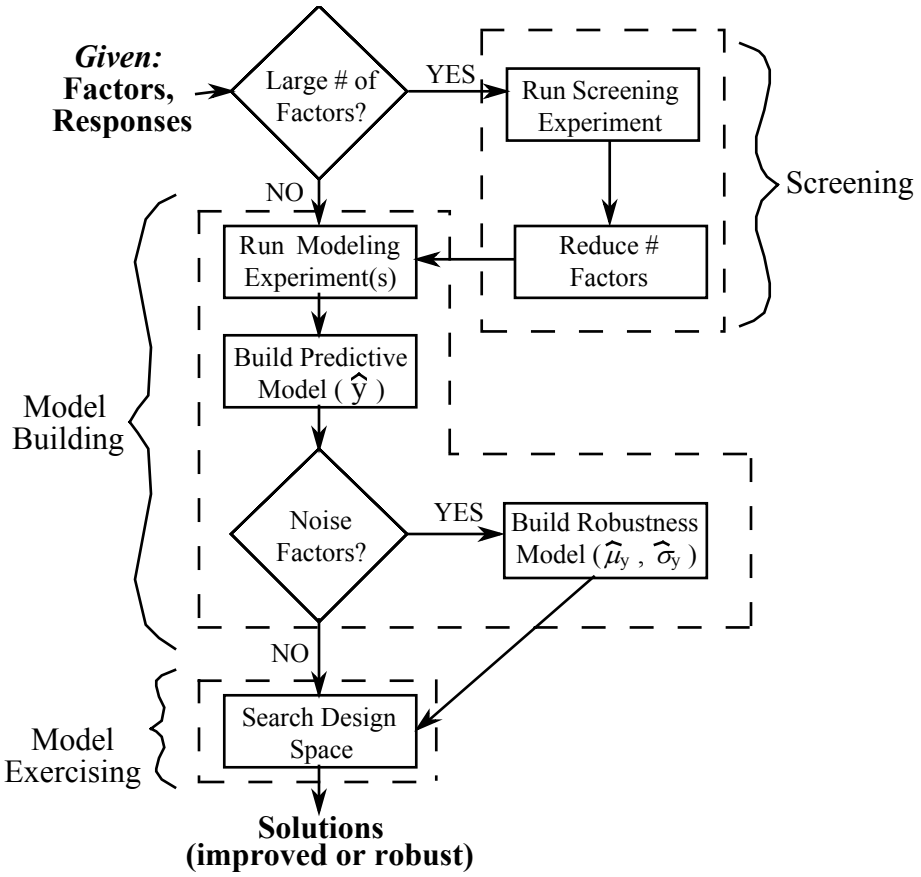


Figure 2.9 General Approach to Response Surface Metamodeling (Koch, et al., 1997)

In the second stage (model building) of the approach in Figure 2.9, response surface models are created to replace computationally expensive analyses and facilitate fast analysis and exploration of the design space. If little curvature appears to exist, a two level fractional factorial experiment is designed, and the first-order polynomial is used to approximate the response(s). If significant curvature exists, then a second-order

polynomial is commonly used. Among the various types of experimental design for fitting a second-order response surface model, the central composite design (CCD) is probably the most widely used experimental design for regularly shaped (spherical or cuboidal) design spaces (Myers and Montgomery, 1995). In the case of irregularly shaped design spaces, D-optimal designs have been successfully employed to build second order response surface models (see, e.g., Giunta, et al., 1994).

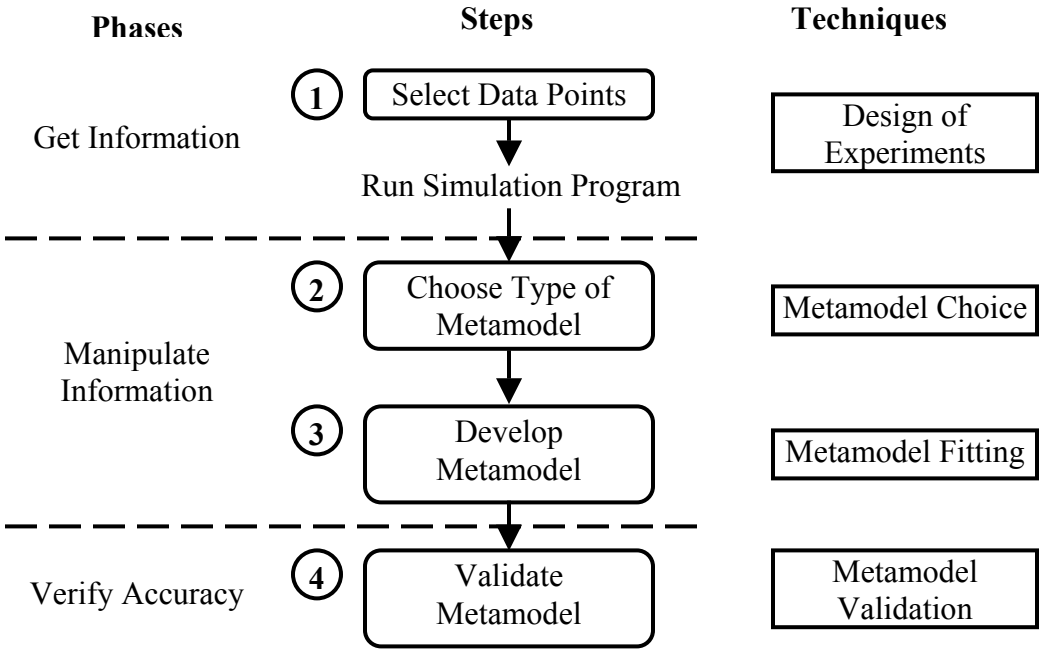


Figure 2.10 Phases, Steps, and Corresponding Techniques in the Metamodeling Process

As seen in Figure 2.10 and as evidenced by the preceding discussion, building approximations of computer analysis and simulation codes involves: (a) choosing an experimental design to sample the computer code, (b) choosing a model to represent the data, and (c) fitting the model to the observed data. Usually a fourth step is needed to

validate the accuracy of metamodels, as illustrated in Figure 2.10. There are a variety of options for each of these steps as shown in Figure 2.11, and some of the more prevalent approximation techniques have been highlighted. For example, response surface methodology usually employs central composite designs, second-order polynomials, and least squares regression analysis. The reader is referred to (Simpson, et al., 1997b) for a review of numerous mechanical and aerospace engineering applications of many of the metamodeling techniques shown in Figure 2.11 with particular emphasis on response surface methodology, neural networks, inductive learning, and kriging. An introduction of various kinds of metamodels is presented in Section 2.3.

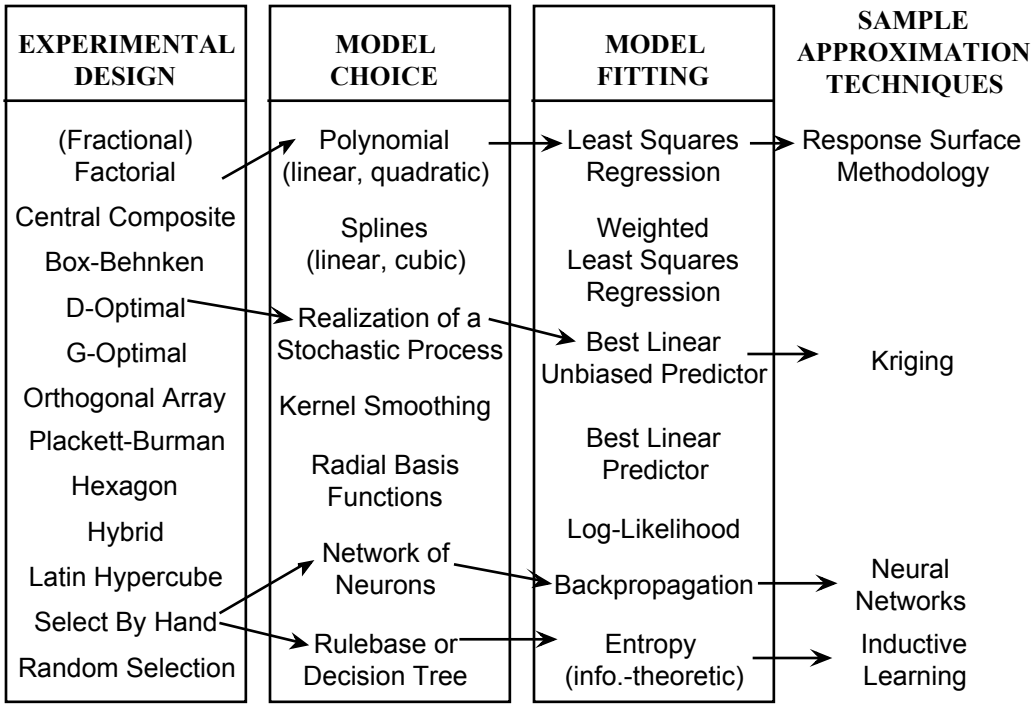


Figure 2.11 Techniques for Metamodeling (Simpson, et al., 1997b)

Metamodels for the actual analysis in complex systems are essential for efficiency and effectiveness in the early design stages in that:

- They yield insight into the relationship between responses, $\boldsymbol{y}$, and design variables, $\boldsymbol{x}$.
- They provide fast analysis tools for design space exploration since cheap-to-run approximations are used instead of the more expensive complete computer analyses.
- They facilitate the integration of discipline dependent analysis codes into the overall design strategy.

An additional advantage of typical metamodels is that they can smooth the data in the case of numerical noise which may hinder the performance of some gradient-based optimizers (see, e.g., Giunta, et al., 1994). This "smoothing" effect for different types of metamodels is both good and bad, depending on the problem and the degrees of "smoothness". Su and Renaud (1996) present an example where a second-order response surface smoothes out the variability in a response so that the robust solution is lost in the approximating function; a "flat region" does not exist in a second-order response surface, only an inflection point. Su and Renaud's example is taken as an example for this dissertation in Chapters 3 and 4.

In Section 2.2.1, the Response Surface Methodology is introduced as an application of metamodeling techniques in engineering fields. The deterministic computer experiment and its impact on metamodeling are discussed in Section 2.2.2. Metamodel validation with computer experiments is discussed in Section 2.2.3.

2.3.1 Response Surface Methodology

In designing large scale engineering systems, the design information increases dramatically along the design timeline. As stated in Section 1.1.1, at different stages of design the design emphasis is different. At the beginning period the design efficiency is much emphasized while as design goes on more and more focus is put on the design effectiveness. From the viewpoint of metamodeling, this shift of design requirements corresponds to the development of more and more accurate metamodels with sequential experiments. The Response Surface Methodology (RSM) is such a method in which sequential experimental designs and sequential metamodels are utilized to reflect the different information and requirements along the design timeline.

Different authors describe Response Surface Methodology differently. Myers and co-authors (1989) define RSM as "a collection of tools in design or data analysis that enhance the exploration of a region of design variables in one or more responses." Box and Draper (1987) state that, "Response surface methodology comprises a group of statistical techniques for empirical model building and model exploitation. By careful design and analysis of experiments, it seeks to relate a *response,* or *output* variable to the levels of a number of *predictors,* or *input* variables, that affect it." Finally, Biles (1984) defines RSM as the, "body of techniques by which one experimentally seeks an optimum set of system conditions".

RSM then encompasses and incorporates the design of experiments (particularly, classical experimental designs, Section 2.4), response surface model building (Section

2.3), and "model exploitation" for exploring a factor space and seeking optimum factor settings. The general RSM approach includes all or a subset of the following steps:

i) *screening:* when the number of factors is too large for a comprehensive exploration and/or when experimentation is expensive, screening experiments are used to reduce the set of factors to those that are most important to the response(s) being investigated;

ii) *first-order experimentation:* when the starting point is far from an optimum (or in general when knowledge about the space being explored is sought), first order-models and an approach such as steepest-ascent are employed to "rapidly and economically move to the vicinity of the optimum" (Montgomery and Evans, 1975);

iii) *second-order experimentation:* after the best solution using first-order methods is obtained, a second-order model is fit in the region of the first-order solution to evaluate curvature effects and attempt to improve the solution.

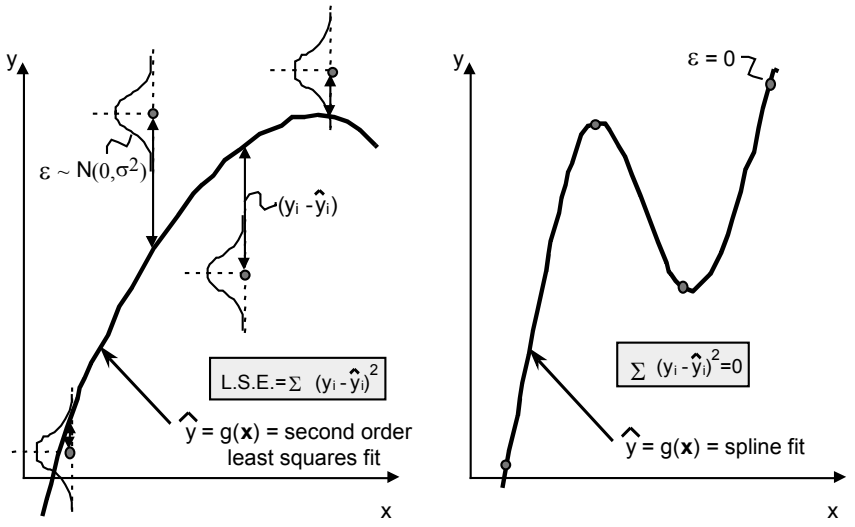
A more detailed description of RSM techniques and tools can be found in (Box and Draper, 1987) and (Myers and Montgomery, 1995); a comprehensive review of RSM developments and applications from 1966-1988 is given in (Myers, et al., 1989). These sequential experiments in RSM are utilized in RCEM to facilitate building sequential metamodels.

Although RSM has been widely applied and proved to be useful, it has many weak points as well as strong points. It is confined to classical experimental designs and regression polynomial models (which is referred as RS models). This limits its usage in deterministic applications as will be discussed in the next section, and in engineering

design, particularly, the robust design, which is pointed out in this thesis and will be studied more in future research.

2.3.2 Deterministic Computer Experiments

Previous research in SRL and other research groups points out that the deterministic property of computer experiments has a great influence in building metamodels for engineering design (see, e.g., Simpson, et al., 1997b; Koch, 1997; Simpson, 1998).



(a) Non-Deterministic Case **(b) Deterministic Case**

Figure 2.12 Deterministic and Non-Deterministic Curve Fitting (Simpson, et al., 1997)

Given a response of interest, y, and a vector of independent factors **x** thought to influence y, the relationship between y and **x** includes the random error term ε. To apply least squares regression, the error values for each data point are assumed to have identical

and independent normal distributions with means of zero and standard deviations of σ, or ε_i i.i.d. $N(0,\sigma^2)$. This scenario is shown in Figure 2.12(a). The least squares estimator then minimizes the sum of the squared differences between the actual data points and the values predicted by the model. It is acceptable if no data point actually lies on the predicted model, because it is assumed that the model "smoothes out" the random error. Of course, it is likely that the regression model itself is only an approximation of the true behavior between x and y, so that the final relationship is

$$y = g(x) + \varepsilon_{bias} + \varepsilon_{random} \tag{2.5}$$

where ε_{bias} represents the error of approximation. However, for deterministic computer experiments as illustrated in Figure 2.12(b), ε_{random} has mean zero *and* variance zero, so after model fitting we have the relationship

$$y = g(x) + \varepsilon_{bias} \tag{2.6}$$

The deterministic case of Equation (2.6) conflicts sharply with the methods of least squares regression. First, unless ε_{bias} is i.i.d. $N(0,\sigma^2)$ the assumptions for statistical inference from least squares regression are violated. Even further, because there is no random error there is little justification for smoothing across data points; instead the model should hit each point exactly and interpolate between them as shown in Figure 2.12(b). Finally, most standard tests for model and parameter significance are based on computations using ε_{random} (the mean squared error) and are therefore impossible to compute. These observations are supported by literature in the statistics community; as

Sacks, et al. (1989a) carefully point out, because deterministic computer experiments lack random error:

- Response surface model adequacy is determined solely by systematic bias,
- The usual measures of uncertainty derived from least-squares residuals have no obvious statistical meaning (deterministic measures of uncertainty exist, e.g., max $|\hat{y}(x) - y(x)|$ over x and a class of y's, but they may be very difficult to compute), and
- The classical notions of experimental blocking, replication and randomization are irrelevant.

Similarly, according to Welch and his co-authors (1990), current methods for the design and analysis of physical experiments (for example, (Box and Draper, 1987; Box, et al., 1978)) are not ideal for complex, deterministic computer models. "In the presence of systematic error rather than random error, statistical testing is inappropriate" (Welch, et al., 1990). Finally, a discussion of how the model *should* interpolate the observations can be found in (Sacks, et al., 1989b).

So where can these methods go wrong? Unfortunately it is very easy to unthinkingly classify the ε_{bias} term from a deterministic model fit as ε_{random} and then proceed with standard statistical testing. Several authors have reported statistical measures such as the F-statistics and root MSE for verification of model adequacy, e.g., (Healy, et al., 1975; Koch, et al., 1996; Simpson, et al., 1997b; Unal, et al., 1994; Venter, et al., 1996; Welch, et al., 1990). These measures have no statistical meaning since they assume the observations include an error term which has mean of zero and a non-zero standard deviation. Consequently, the use of stepwise regression for polynomial model

fitting is not appropriate since it utilizes F-statistic values when adding/removing model parameters.

R-Squared (when defined as the model sum of squares divided by the total sum of squares and thus varying from 0 to 1) is really the only measure for verifying model adequacy for deterministic computer experiments, and often this measure not sufficient (a high R-Squared value can be deceiving). Consequently, confirmation testing of model validity through use of additional (different) data points becomes essential. Residual plots may also be extremely helpful when verifying model adequacy for identifying trends in data, examining outliers, etc.

Some researchers (e.g., (Giunta, et al., 1996; Giunta, et al., 1994; Venter, et al., 1996)) have also employed metamodeling techniques such as RSM for modeling deterministic computer experiments which contain numerical noise. This numerical noise is used as a surrogate for random error, thus allowing the standard least-squares approach to be applied. However, the assumption of equating numerical noise to random error is questionable, and the appropriateness of their approach warrants further investigation.

The initial motivation for introducing space filling experimental designs and different types of metamodels (e.g., kriging, ANN, etc.) into engineering design has been presented in this section. Though techniques used in RSM, such as RS models, validation statistics, etc., receive theoretical criticize in deterministic cases, there are few studies and applications in which they perform apparently weak. One exception are the experimental designs, for which an intensive study is performed in (Simpson, 1998) and

the conclusion is that space filling experimental designs act better than classical ones in deterministic applications.

2.3.3 Validation of Metamodels

As pointed out in Section 2.2.1, previously widely used statistics in RSM (e.g., F-statistics, etc.) may be meaningless or inappropriate in deterministic computer applications; other methods are needed to validate the metamodels. Mitchell and Morris, (1992a) propose the leave-one-out cross validation approach. In this approach, each sample point used to fit the model is removed one at a time, the model is rebuilt without that sample point, and the difference between the model without the sample point and actual value at the sample point is computed for all of the sample points. While study in (Simpson, 1998) shows that this method does not provide a good assessment of model accuracy, thus, additional validation points must be taken. A more detailed study on leave-one-out cross-validation is included in Chapter 3 of this dissertation.

If additional validation points can be afforded, then the maximum absolute error (MAX), average absolute error, and root mean square error (RMSE) for the additional validation points can be calculated to assess model accuracy. Usually NRMSE and NMAX are used; they refer to the values of RMSE and MAX when normalized against the sample range. These measures are summarized in Table 2.1. In the table, n_{error} is the number of random test points used, y_i is the actual value from the computer code/simulation, and $\hat{y}_i$ is the predicted value from the approximation model.

Table 2.1 Error Measures for Kriging Metamodels (Simpson, 1998)

Name	Error Measure	Eqn. #
Max. abs. Error	max. $\| y_i - \hat{y}_i \|$ i = 1, ..., n_{error}	(2.7)
Avg. abs. Error	$\frac{1}{n_{error}} \sum_{i=1}^{n_{error}} \left| y_i - \hat{y}_i \right|$	(2.8)
RMSE	$\sqrt{\frac{\sum_{i=1}^{n_{error}} (y_i - \hat{y}_i)^2}{n_{error}}}$	(2.9)

To select the validation points is another problem of experimental designs and this is where sequential experimental designs could take advantage. In previous research (see, e.g., Simpson, 1998), the validation points are selected spreading across the design space because 1). The problem is simple and it is possible to afford a great number of sample points and validation points, and 2). In previous research the focus is to study the properties of metamodeling techniques, but not the sequential development of metamodels along the design timeline. While in engineering design of large-scale complex systems, metamodeling must be considered to be a sequential process to fit the product realization procedure. Sequential experimental designs are needed to help develop sequential metamodels, and these sequential experimental designs must take the selection of validation points into account.

As stated early in this section, classical experiments in RSM are designed in a sequential manner to help gain efficiency, while few efforts are put on the sequential usage of space filling experiments. Although a single space filling experiment is proved to be more efficient than a single classical experiment (Simpson, 1998), it is possible that

classical experiments perform better than space filling experiments in a sequential case. One of my aims in this dissertation is to develop a method for designing sequential computer experiments in which information from previous data points and metamodels could be used as a guide in identifying new data points.

Different types of metamodels are introduced in the next section. Our focus is on the regression polynomials, kriging models, and multivariate adaptive regression splines.

2.4 DIFFERENT TYPES OF METAMODELS

In statistical modeling, the objective is to estimate the relationship between a response variable, typically univariate, and several predictor variables. The response surface represents the true mean response. In the case of metamodeling, it is assumed that there is no error variability in the observed response values; thus, the "mean" response coincides with the actual responses. There are several statistical methods available for estimating the response surface. In this section we present six of them: response surface (RS) models, kriging models, multivariate adaptive regression splines (MARS), regression trees, artificial neural networks (ANN), and wavelets in Section 2.3.1 – 2.3.4, respectively. However, only the RS model, kriging model, and MARS are used and studied in this dissertation.

2.4.1 Response Surface Models

RSM was first developed through the collaboration of a statistician and a chemist (Box and Wilson, 1951). Many authors have compared Taguchi techniques with traditional Response Surface Methodology (RSM) for different problems and advocate a

combined Taguchi-RSM approach (e.g., Lucas, 1994; Ramberg, et al., 1991; Unal, et al., 1994; Mavris, et al., 1999). RSM incorporates the design of experiments, response surface model building, and model exploitation to explore a factor space and seek optimal factor settings. The general form of response surface (RS) models (see Box and Draper, 1987) is a polynomial function. Since this is a linear model (in parameters), the usual linear model tools may be applied. Thus, RS models are very easy to use. The drawback is that the rigid structure of a pre-selected polynomial model may not be flexible enough to represent the true response surface.

The RS models studied in this thesis are second-order polynomials and expressed as the following:

$$\hat{y} = b_0 + \sum_{i=1}^{n} b_i x_i + \sum_{i=1}^{n} b_{ii} x_i^2 + \sum_{\substack{i \\ i<j}} \sum_{j} b_{ij} x_i x_j \qquad (2.10)$$

where b's are coefficients. For details see (Myers and Montgomery, 1995).

Second-order RS models are easy to use and implement; however, they have limited capability to model accurately non-linear functions of arbitrary shape. Some two variable examples of the types of surfaces that a second-order response surface can model are illustrated in Figure 2.13.

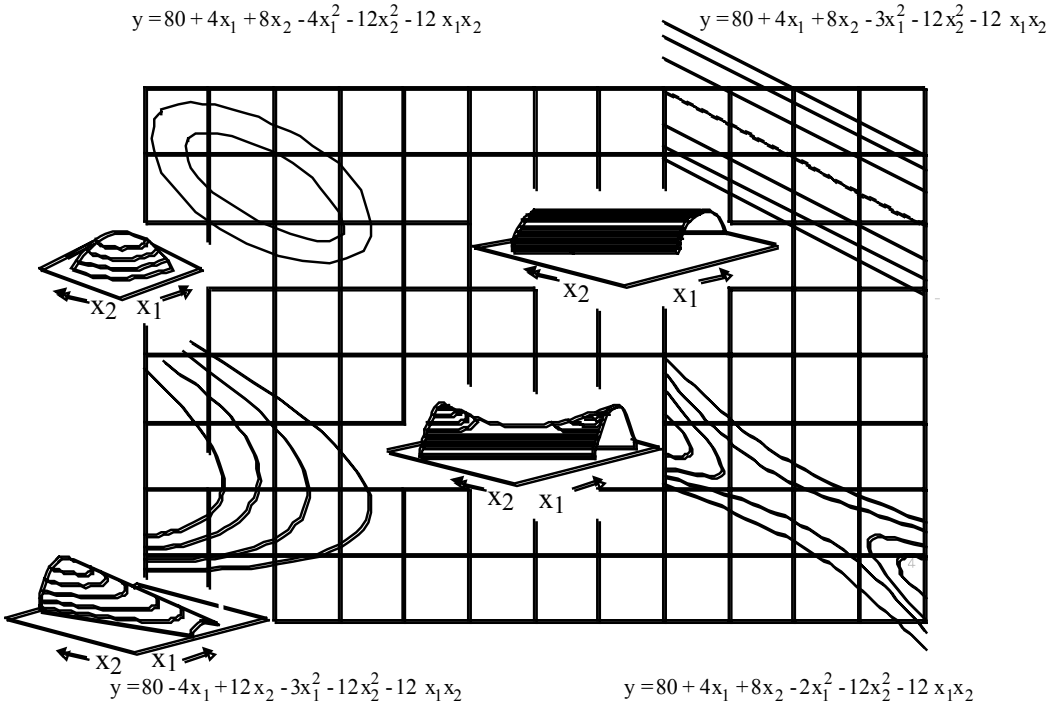


Figure 2.13 Sample Two Variables Second-Order Response Surfaces (adapted from Box and Draper, 1987)

Higher-order response surfaces can be used to model a non-linear design space; however, instabilities may arise (see, e.g., Barton, 1992), or it may be too difficult to take a sufficient number of sample points in order to estimate all of the coefficients in the polynomial equation, particularly in high dimensions. Hence, many researchers advocate the use of a *sequential* response surface modeling approach using move limits (see, e.g., Toropov, et al., 1996) or a trust region approach (see, e.g., Rodriguez, et al., 1997). More generally, the Concurrent SubSpace Optimization procedure uses data generated during concurrent subspace optimization to develop response surface approximations of the design space which form the basis of the subspace coordination procedure (Renaud and Gabriele, 1994; Renaud and Gabrielle, 1991; Wujek, et al., 1995). The Hierarchical and

Interactive Decision Refinement methodology uses statistical regression and other metamodeling techniques to recursively decompose the design space into subregions and fit each region with a separate model during design space refinement (Reddy, 1996). Finally, the Model Management Framework (Booker, et al., 1995; Dennis and Torczon, 1995) is being developed collaboratively by researchers at Boeing, IBM, and Rice to implement mathematically rigorous techniques to manage the use of approximation models in optimization.

Many of the previously mentioned sequential approaches are being developed for *single objective* optimization applications. Since much of engineering design is *multiobjective* in nature, it is often difficult to isolate a small region of good design which can be accurately represented by a low-order polynomial response surface model. Koch, et al. (1997) discuss the difficulties encountered when screening large variable problems with multiple objectives as part of the response surface approach. Barton (1992) states that the response region of interest will never be reduced to a "small neighborhood" which is good for all objectives during multiobjective optimization. Hence, there is a need to investigate alternative metamodeling techniques which have sufficient flexibility to build accurate global approximations of the design space and which are suitable for modeling computer experiments which are typically deterministic, i.e., contain no random error or variability, as discussed in Section 2.2. Alternative metamodels are introduced in following sections.

2.4.2 Kriging

Kriging evolved in the field of geostatistics (Matheron, 1963) and has recently become popular in the area of spatial statistics (Cressie, 1993). From a spatial perspective, the values of the predictor variables are points in the multi-dimensional predictor space. In kriging some form of spatial correlation between points in the predictor space is assumed, and this correlation is used to predict response values between observed points. The resulting estimated surface interpolates the observed responses (though it is possible to induce smoothed kriging models which do not interpolate).

Kriging is named after D. G. Krige, a South African mining engineer who, in the 1950's, developed empirical methods for determining true ore grade distributions from distributions based on sampled ore grades (Matheron, 1963). Several texts which describe kriging and its usefulness for predicting spatially correlated data (see, e.g., Cressie, 1993) and mining (see, e.g., Journel and Huijbregts, 1978) exist. These metamodels are extremely flexible due to the wide range of correlation functions which can be chosen for building the metamodel. Furthermore, depending on the choice of the correlation function, the metamodel can either "honor the data," providing an exact interpolation of the data, or "smooth the data," providing an inexact interpolation (Cressie, 1993). In this dissertation, as in most applications of kriging, the concern is solely on spatial prediction; it is assumed that the data are not temporally correlated.

These days, kriging goes by a variety of names including DACE (Design and Analysis of Computer Experiments) modeling—the title of the inaugural paper by Sacks,

et al. (1989a) — and spatial correlation metamodeling (see, e.g., Barton, 1994). There are also several types of kriging (cf., Cressie, 1993): ordinary kriging, universal kriging, lognormal kriging, and trans-Gaussian kriging. In this dissertation, ordinary kriging is employed, following the work in (e.g., Booker, et al., 1995; Koehler and Owen, 1996; Simpson, 1998), and only the term kriging is used.

Although there are more and more researches on kriging metamodels in engineering design, the usage of kriging metamodels in real-world engineering design is still limited after its introduction into the literature by Sacks, et al. (1989a). One reason may be that the estimated parameters of a kriging model are computationally intensive to obtain, and the assumptions related to the correlation function are difficult to verify. Initial applications of kriging in engineering design include:

- Giunta (1997) and Giunta, et al. (1998) perform a preliminary investigation into the use of kriging for the multidisciplinary design optimization of a High Speed Civil Transport aircraft.
- Sasena (1998) compares and contrasts kriging and smoothing splines for approximating noisy data.
- Schonlau, et al. (1997) use a global/local search algorithm based on kriging for shape optimization of an automobile piston engine.
- Osio and Amon (1996) develop a multistage numerical optimization strategy based on kriging which they demonstrate on the thermal design of embedded electronic package which has 5 design variables.
- Booker (1996) and Booker, et al. (1996) using a kriging approach to study the aeroelastic and dynamic response of a helicopter rotor during structural design.

- Simpson (1998) compares second-order RS models and kriging models with different correlation functions and applied kriging models in product family design of nozzles, electric motors, and aircraft.
- Lin (2000) studied the performance of kriging models in robust design; applications include design of electrical vehicle body structures and gear trains, etc.

Some researchers have also employed kriging-based strategies for numerical optimization (see, e.g., Cox and John, 1995; Trosset and Torczon, 1997). A look at the mathematics of kriging is offered next.

Mathematics of Kriging

Kriging postulates a combination of a polynomial model and departures of the form:

$$y(\boldsymbol{x}) = f(\boldsymbol{x}) + Z(\boldsymbol{x}) \tag{2.11}$$

where $y(\boldsymbol{x})$ is the unknown function of interest, $f(\boldsymbol{x})$ is a known polynomial function of $\boldsymbol{x}$, and $Z(\boldsymbol{x})$ is the realization of a stochastic process with mean zero, variance σ^2, and non-zero covariance. The $f(\boldsymbol{x})$ term in Equation 2.11 is similar to the polynomial model in a response surface, providing a "global" model of the design space. In many cases $f(\boldsymbol{x})$ is simply taken to be a constant term β (cf., Koehler and Owen, 1996; Sacks, et al., 1989a; Welch, et al., 1990). Only kriging models with constant underlying global models are investigated in this work as well.

While $f(\boldsymbol{x})$ "globally" approximates the design space, $Z(\boldsymbol{x})$ creates "localized" deviations so that the kriging model interpolates the n_s sampled data points. The covariance matrix of $Z(\boldsymbol{x})$ which dictates the local deviations is:

$$\text{Cov}[Z(\boldsymbol{x}^i),Z(\boldsymbol{x}^j)] = \sigma^2\, \boldsymbol{R}([R(\boldsymbol{x}^i,\boldsymbol{x}^j)] \tag{2.12}$$

where $\boldsymbol{R}$ is the correlation matrix, and $R(\boldsymbol{x}^i,\boldsymbol{x}^j)$ is the correlation function between any two of the n_s sampled data points $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$. $\boldsymbol{R}$ is an $n_s \times n_s$ symmetric, positive definite matrix with ones along the diagonal. The correlation function $R(\boldsymbol{x}^i,\boldsymbol{x}^j)$ is specified by the user.

Table 2.2 Summary of Correlation Functions

Name	Spatial Correlation Function	# Deriv.	Eqn. #
Exponential	$\prod_{k=1}^{n_{dv}} \exp(-\theta_k \lvert d_k \rvert)$	1	(2.13)
Gaussian	$\prod_{k=1}^{n_{dv}} \exp(-\theta_k \lvert d_k \rvert^2)$	∞	(2.14)
Cubic spline	$\prod_{k=1}^{n_{dv}} \begin{cases} 1-6(\theta_k \lvert d_k \rvert)^2 + 6(\theta_k \lvert d_k \rvert)^3 & \theta_k \lvert d_k \rvert < \frac{1}{2} \\ 2(1-\theta_k \lvert d_k \rvert)^3 & \frac{1}{2} \le \theta_k \lvert d_k \rvert < 1 \\ 0 & \theta_k \lvert d_k \rvert \ge 1 \end{cases}$	1	(2.15)
Matérn linear function	$\prod_{k=1}^{n_{dv}} \left[(1+\theta_k \lvert d_k \rvert)\exp(-\theta_k \lvert d_k \rvert)\right]$	1	(2.16)
Matérn cubic function	$\prod_{k=1}^{n_{dv}} \left[(1+\theta_k \lvert d_k \rvert + \frac{\theta_k^2 \lvert d_k \rvert^2}{3})\exp(-\theta_k \lvert d_k \rvert)\right]$	2	(2.17)

Five different correlation functions have been studied in previous work by the author of this dissertation, see Table 2.2. In all the correlation functions listed in the table, n_{dv} is the number of design variables, θ_k are the unknown correlation parameters used to fit the model, and $d_k = x_k^i - x_k^j$ which is the distance between the k^{th} components of sample points $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$. The correlation functions of Equations 2.13 and 2.14 are from (Sacks, et al., 1989a); the correlation functions of Equations 2.15 – 2.17 are from

(Mitchell and Morris, 1992b). In this dissertation, only the Gaussian correlation function (Equation 2.14) is used in developing kriging models because in the literature the Gaussian correlation is by far the most popular one in use. Correlation functions with multiple parameters per dimension exist; however, correlation functions with only one parameter per dimension are considered in this dissertation to facilitate finding the maximum likelihood estimates (MLEs) or "best guess" of the θ_k used to fit the model.

Once a correlation function has been selected, predicted estimates, $\hat{y}(\boldsymbol{x})$, of the response, $y(\boldsymbol{x})$, at untried values of $\boldsymbol{x}$ are given by:

$$\hat{y} = \hat{\beta} + \mathrm{r}^T(\mathrm{x})\mathrm{R}^{-1}(\mathrm{y} - \mathrm{f}\hat{\beta}) \tag{2.18}$$

where $\boldsymbol{y}$ is the column vector of length n_s (number of sample points) which contains the values of the response at each sample point, and $\mathbf{f}$ is a column vector of length n_s which is filled with ones when $f(\boldsymbol{x})$ in Equation 2.11 is taken as a constant. In Equation 2.18, $\boldsymbol{r}^{\mathrm{T}}(\boldsymbol{x})$ is the correlation vector of length n_s between an untried $\boldsymbol{x}$ and the sampled data points $\{\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^{n_s}\}$ and is given by:

$$\mathbf{r}^{\mathrm{T}}(\mathbf{x}) = [\mathrm{R}(\mathbf{x},\mathbf{x}^1), \mathrm{R}(\mathbf{x},\mathbf{x}^2), ..., \mathrm{R}(\mathbf{x},\mathbf{x}^{n_s})]^{\mathrm{T}} \tag{2.19}$$

Finally, the $\hat{\beta}$ in Equation 2.18 is estimated using the following expression.

$$\hat{\beta} = (\mathrm{f}^{\mathrm{T}}\mathrm{R}^{-1}\mathrm{f})^{-1}\mathrm{f}^{\mathrm{T}}\mathrm{R}^{-1}\mathrm{y} \tag{2.20}$$

When $f(\boldsymbol{x})$ is assumed to be a constant, then $\hat{\beta}$ is a scalar which simplifies the calculation of Equation 2.20 and all others involving $\hat{\beta}$.

The estimate of the variance, $\hat{\sigma}^2$, from the underlying global model (not the variance of the randomness in the observed data itself) is:

$$\hat{\sigma}^2 = \frac{(\mathrm{y} - \mathrm{f}\hat{\beta})^T \mathrm{R}^{-1} (\mathrm{y} - \mathrm{f}\hat{\beta})}{n_s} \tag{2.21}$$

where **f** is again a column vector of ones because $f(\boldsymbol{x})$ is assumed to be a constant. The maximum likelihood estimates (i.e., "best guesses") for the θ_k used to fit the model are found by maximizing Equation 2.22 over $\theta_k > 0$ (Booker, et al., 1995):

$$-\frac{[n_s \ln(\hat{\sigma}^2) + \ln|\mathrm{R}|]}{2} \tag{2.22}$$

Both $\hat{\sigma}^2$ and $|\mathbf{R}|$ are functions of θ_k. While *any* values for the θ_k create an interpolative approximation model, the "best" kriging model is found by solving the k-dimensional unconstrained nonlinear optimization problem given by Equation 2.22. It is worth noting that in some cases using a single correlation parameter gives sufficiently good results (Booker, et al., 1995; Osio and Amon, 1996; Sacks, et al., 1989a). In this dissertation, however, a unique θ value for each dimension is always considered based on past difficulties with scaling the design space to $[0,1]^k$ during the model fitting process.

2.4.3 Multivariate Adaptive Regression Splines

Multivariate Adaptive Regression Splines (MARS) were introduced by Friedman (Friedman, 1991). It is known that pre-specified parametric models are limited in flexibility and accuracy since accurate estimates are usually only possible when the true function is close to the pre-specified parametric one. Thus, when the form of the underlying true function is unknown, statisticians prefer methods like MARS that can adaptively create a statistical model.

MARS is essentially a linear model with a forward and backward stepwise algorithm to select the terms to include in the model. The piecewise-linear MARS approximation is a linear combination of linear basis functions that are truncated at knots. The knots determine where the approximation bends to model curvature, and one of the objectives of the forward stepwise algorithm is to select appropriate knots. After a reasonable piecewise-linear MARS approximation has been constructed, there is an option to smooth the approximation to achieve first derivative (or higher) continuity. MARS is both flexible and straightforward to implement with the computational effort primarily dependent on the number of basis functions added to the model. This approach has been successfully used in modeling the objective function in large-scale dynamic programming problems (Chen, 1999; Chen, et al., 1999).

The MARS model is built by taking the form of an expansion in product spline basis functions, where the basis functions are selected by the data. MARS uses the multiple regression model:

$$y_i = g(x_{j1}, x_{j2}, ..., x_{jn_v}) + \varepsilon_j, \qquad j = 1, ..., n_s \tag{2.23}$$

where n_v is the number of covariates $\boldsymbol{x} = (x_1, \ldots, x_{\mathrm{nv}})^{\mathrm{T}}$, n_s is the number of data points, the error ε_j is a random variable with mean equal to zero, and the "regression function" g is smooth but otherwise arbitrary.

The MARS procedure for estimating g consists of three parts:

1. A forward stepwise algorithm to select basis functions,
2. A backward stepwise algorithm to delete basis functions until the "best" set is found, and
3. A smoothing method which gives the final MARS approximation a certain degree of continuity.

This is an adaptive procedure because the selection of basis functions is data-based and specific to the problem in hand. The adaptive strategy has the ability to reduce the dimensionality of high dimensional problems.

The forward and backward stepwise procedures described in Friedman's paper are restated in the following sections. The forward stepwise algorithm takes most of the computational effort in MARS. One major focus on this research is to improve computational performance. To demonstrate the potential of improvement, the MARS forward stepwise algorithm will be explained step by step in Section 2.3.3.1. The backward stepwise procedure prunes the MARS approximation attained from the forward stepwise algorithm, by removing unnecessary basis functions one at a time. Robustness may be improved by pruning, which was discussed in (Tsai, 2002). A brief introduction

on the backward stepwise procedure will be given in Section 2.3.3.2. At last, to give MARS continuity and a continuous first and second derivative at the side knots, a MARS approximation with quintic basis functions derived in (Chen, et al., 1999) is presented in Section 2.3.3.3.

2.4.3.1 MARS Forward Stepwise Algorithm

The forward stepwise algorithm is the most computationally expensive component of MARS. The algorithm is described below, and the notation is introduced as follows. For more details, see (Tsai, 2002). $M_{\max}$ is the maximum number of basis functions, which is used to determine when to terminate MARS approximation. B_m is the m-th basis function. The quantity L_m is the number of splits that gave rise to B_m, $v(l,m)$ label the predictor variables that are in the l-th split of the m-th basis function and k represents values on the corresponding variables.

The forward stepwise algorithm starts with the constant basis function $B_1(\boldsymbol{x}) = 1$, and initializes the counter variable M. Within the M-loop beginning on the second step, basis functions M and $M + 1$ are added. The m-loop searches through the $M - 1$ basis functions that have already been added for the best one to "split". Univariate basis functions "split" the constant basis function at a knot k for covariate x_v in the form of truncated linear functions,

$$b^+(x_v - k) = [+(x_v - k)]_+, \qquad b^-(x_v - k) = [-(x_v - k)]_+, \tag{2.24}$$

where $[q]_+ = \max\{0,q\}$. Interaction basis functions are created by "splitting" (multiplying) an existing basis function $B_m(\boldsymbol{x})$ with a truncated linear function involving a

new covariate. Both the existing basis function and the newly created interaction basis function are used in the MARS approximation. Then the designers select the next two basis functions (M and $M + 1$) to add by loop through the possible choices for basis function (m), covariate (v), and knot (k).

Possible basis functions are compared with the lack-of-fit (lof). There are various options for lof, and the least-squared criterion is used in this dissertation and defined as:

$$LOF(\hat{g}_M) = \sum_{i=1}^{N} [y_i - \hat{g}_M(x_i)]^2 \tag{2.25}$$

The indices m, v, and k are stored for the "split" that currently yields the smallest lof. The algorithm stops when a certain number of basis functions constrained by $M_{\max}$ has been accumulated, where $M_{\max}$ is a user-specified constant. The MARS approximation approaches interpolation as the number of basis functions increases, but there is a trade-off between $M_{\max}$ and computational time. To save MARS computational effort during the forward stepwise search, instead of computing the least-squares lack-of-fit defined in Equation (2.25), $I(k)$ is used as the criterion to decide which knot would be added to the new basis function. To be specific, let $\hat{g}_M(x_i)$ be the i-th fitted value using the current set of orthonormal basis functions and $\hat{g}_{M+1}(x_i)$ be the i-th fitted value including basis function $M + 1$. The decrease in the lack-of-fit is proportional to

$$I(k) = \sum_{i=1}^{N} (y_i - \hat{g}_M(x_i))^2 - \sum_{i=1}^{N} (y_i - \hat{g}_{M+1}(x_i))^2 \tag{2.26}$$

The MARS algorithm actually adds two basis functions at a time and the corresponding $I(k)$ is of the following form:

$$I(k) = \sum_{i=1}^{N} (y_i - \hat{g}_{M-1}(x_i))^2 - \sum_{i=1}^{N} (y_i - \hat{g}_{M+1}(x_i))^2 \qquad (2.27)$$

Friedman pointed out that "$I(k)$ is the improvement in the residual sum of squares resulting from adding the corresponding basis function with knot location k," and "The decrease in the (least-squares) lack-of-fit to be evaluated in the innermost loop of the forward stepwise algorithm at each potential knot location k is proportional to $-I(k)$."

2.4.3.2 MARS Backward Stepwise Algorithm

The backward stepwise starts with all $M_{\max}$ basis functions derived from the forward stepwise algorithm. It omits one basis function at a time and finds the best set of basis functions for the MARS approximation.

At the beginning of the algorithm, J^* is used to represent the entire basis function set derived from the forward stepwise algorithm, and the lack-of-fit of this set is saved. The best set of $M_{\max} - 1$ basis functions is found by deleting one basis function at a time. It is the one whose removal either improves the fit the most or degrades it the least. Then it loops again starting with that best set to find the best set of $M_{\max} - 2$ basis functions. Throughout the algorithm, it keeps track of the overall best. After completion of the backward stepwise algorithm, J^* holds the best set of basis functions. The backward stepwise algorithm can be used to ensure a best model as well as to make the MARS approximation more robust.

2.4.3.3 Degree of Continuity

Since the resulting MARS estimate is nonlinear, in general the dynamic program requires a nonlinear minimization method that uses first and second derivatives to find the minimum. Friedman's MARS replaces the truncated linear basis functions $[\pm(x-k)]_+$ in the forward and backward stepwise algorithms with cubic functions, which provides a continuous first derivative and a continuous second derivative everywhere except at the side knots. To give MARS continuity and a continuous first and second derivative at the side knots, quintic functions derived in (Chen, et al., 1994) in place of Friedman's cubic functions are shown below. For sing s and knots k_-, k, and k_+, quintic functions defined as:

$$Q(x \mid s=1, k_-, k, k_+) = \begin{cases} 0, & x \le k_- \\ \alpha_+(x-k_-)^3 + \beta_+(x-k_-)^4 + \gamma_+(x-k_-)^5, & k_- < x < k_+ \\ x-k, & x \ge k_+ \end{cases} \tag{2.28}$$

$$Q(x \mid s=-1, k_-, k, k_+) = \begin{cases} k-x, & x \le k_- \\ \alpha_-(x-k_+)^3 + \beta_-(x-k_+)^4 + \gamma_-(x-k_+)^5, & k_- < x < k_+ \\ 0, & x \ge k_+ \end{cases} \tag{2.29}$$

satisfy the constraints as requiring

$$\alpha_+ = \frac{6k_+ - 10k + 4k_-}{(k_+ - k_-)^3},$$
$$\beta_+ = \frac{-8k_+ + 15k - 7k_-}{(k_+ - k_-)^4}, \tag{2.30}$$
$$\gamma_+ = \frac{3k_+ - 6k + 3k_-}{(k_+ - k_-)^5},$$

$$\alpha_- = \frac{(-1)(6k_+ - 10k + 4k_-)}{(k_- - k_+)^3},$$
$$\beta_- = \frac{(-1)(-8k_+ + 15k - 7k_-)}{(k_- - k_+)^4}, \tag{2.31}$$
$$\gamma_- = \frac{(-1)(3k_+ - 6k + 3k_-)}{(k_- - k_+)^5},$$

Nonconvexities are produced in the cubic and quintic basis functions when the center knot k is not close enough to the midpoint between k_- and k_+. When having

$$\frac{k_+ - k}{k_+ - k_-} < \frac{2}{5} \quad for \quad s = 1, \quad or$$
$$\frac{k - k_-}{k_+ - k_-} < \frac{2}{5} \quad for \quad s = -1. \tag{2.32}$$

Chen proves Equation (2.32) by considering four cases (Chen, 1993). For accurate minimization, it is desirable for the objective function to be convex. To avoid this potential cause for nonconvexity, the inequality shown in Equation (2.32) are to be checked when k_- and k_+ are chosen. If the ratio does not meet the constraints, the appropriate side knots need to be adjusted.

2.4.4 Other Types of Metamodels

In this section we briefly review other types of metamodels, say, Artificial Neural Networks (ANN), Regression trees, and wavelets.

ANN models have been very popular for modeling a variety of physical relationships. The original motivation for ANN comes from how "learning" strengthens connections along neurons in the brain. Commonly, an ANN model is represented by a diagram of nodes in various layers with weighted connections between nodes in different layers. At the input layer, the nodes are the predictor variables and at the output layer, the nodes are the response variable(s). In between, there is usually at least one "hidden" layer which induces flexibility into the modeling. Mathematically, an ANN model is a nonlinear statistical model, and a nonlinear method is used to estimate the parameters (weights) of the model. There are two main issues in building a network: 1). Specifying the architecture for the network, and 2). Training the network to perform well with reference to a training set. To a statistician, this is equivalent to (i). Specifying a regression model, and (ii). Estimating the parameters of the model given a set of data (Cheng and Titterington, 1994). If the architecture is made large enough, a neural network can be a nearly universal approximator (Rumelhart, et al., 1994).

Neural networks are best suited to approximate deterministic functions in regression-type applications. Cheng and Titterington (1994) note that “In most applications of neural networks that generate regression-like output, there is no explicit mention of randomness. Instead, the aim is function approximation.” Typical applications are speech recognition and handwritten character recognition. Although

ANN models are generally flexible enough to model any relationship, they are computationally intensive, and a significant quantity of representative data is required to both fit and validate the model. Very often the data is complex and of high dimensionality. Networks with tens of thousands of parameters are not unheard of, and the amount of training data is similar. Gathering the training data and determining the model parameters is a process that can be very computationally expensive. Consequently, neural networks are better suited for applications in which the models can be used repeatedly; for a single design application, the cost of building the model may outweigh the associated gain in exercising the model.

Regression trees (see Breiman, et al., 1984) are closely related to MARS. Instead of a piecewise-linear approximation, regression trees form a piecewise-constant approximation. Wavelet modeling is a relatively new technique that has found great success in image and signal processing (Mallet, 1998). A wavelet is a special form of basis function that is particularly effective in modeling sharp jumps in the response surface. The continuous wavelet transform maybe used to identify the locations of these jumps. Similar to ANN, wavelets are best used when a large quantity of data is available.

Various metamodels are introduced in this section and we emphasized on the RS, kriging, and MARS models in this dissertation. To compare the performance of RS, kriging, and MARS models in design, which we propose to do in this thesis, will help designers develop appropriate metamodels in their design activities. Note that in design (particularly for early stages of design), typically it is expensive to obtain lots of data for building metamodels. Thus the design of experiments for data points is very important.

In the next section, several experimental designs are presented in two categories: classical DOE and space filling DOE.

2.5 DESIGN OF EXPERIMENTS

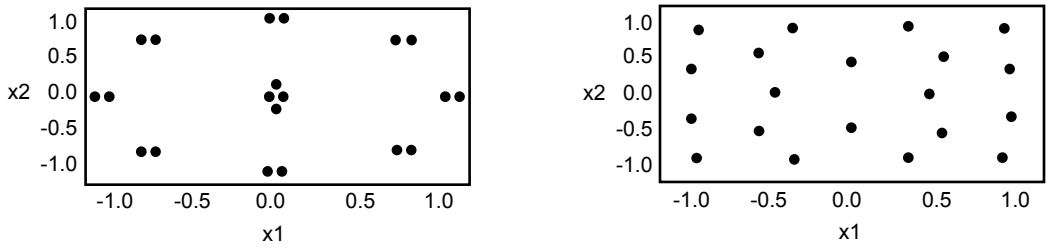
Properly designed experiments are essential for effective computer utilization. The traditional approach in engineering is to vary one parameter at a time within a computer analysis code and observe the effects or to randomly assign different combinations of factor settings to be used as alternative *parametric* analyses for comparisons. Design of Experiments (DOE) represents techniques with which we are able to reasonably select data point in the design space for fitting a model.

An experimental design formally represents a sequence of experiments to be performed, expressed in terms of *factors* (design variables) set at specified *levels*, or predefined values. An experimental design is represented mathematically by a matrix $\boldsymbol{X}$ where the rows denote experimental runs and the columns denote the particular factor setting for each run.

There are essentially two categories of experimental designs, say, the classical DOE and space filling DOE. Booker (1996) summarizes the difference between classical experimental designs and new space filling designs well. In the classical design and analysis of physical experiments, random variation is accounted for by spreading the sample points out in the design space and by taking multiple data points (replicates), see Figure 2.14a. In deterministic computer experiments, replication at a sample point is meaningless; therefore, the points should be chosen to fill the design space. One

approach is to minimize the integrated mean square error over the design region (cf., Sacks, et al., 1989b); the space filling design illustrated in Figure 2.14b is an example of such a design.

After generally talking about the *D*-optimal designs in Section 2.4.1, several kinds of classical DOE and space filling DOE are briefly introduced in Section 2.4.2.



(a) Classical design w/replicates (b) Space filling design w/o replicates

Figure 2.14 Example Classical and Space Filling Experimental Designs

2.5.1 D-Optimal Experiments

Selecting the appropriate design is essential for effective experimentation. Experimenters must balance the desire to gain as much information as possible about the response-factor relationships with the cost of experimentation and need for efficiency (measured in numbers of runs). There are several available measures of merit, useful for evaluating and comparing experimental designs to ensure the appropriate experiment is designed, while in this section, we will focus on the *D*-optimal experiments.

Much of the development of computer-generated designs is an outgrowth of work by Kiefer (1959, 1961) and Kiefer and Wolfowitz (1959) in the theory of optimal designs. An optimal design is a design that is "best" with respect to some criterion. The

usual approach is to specify a model, determine the region of interest, select the number of runs to make, specify the optimality criterion, and then choose the design points from a set of candidate points that the experimenter would consider using. Typically, the candidate points are a grid of points spaced over the feasible design region.

There are several popular design optimality criteria, and *D*-optimality criterion is perhaps the most widely used one. Unlike standard classical designs such as factorials and fractional factorials, *D*-optimal design matrices are usually not orthogonal and effect estimates are correlated. These types of designs are always an option regardless of the type of model the experimenter wishes to fit or the objective specified for the experiment (for example, screening, response surface, etc.). *D*-optimal designs are straight optimizations based on a chosen optimality criterion and the model that will be fit. The optimality criterion used in generating *D*-optimal designs is one of maximizing $|\boldsymbol{X}'\boldsymbol{X}|$ (or $\det(\boldsymbol{X}'\boldsymbol{X})$), the determinant of the information matrix $\boldsymbol{X}'\boldsymbol{X}$. In the case of *D*-optimality for regression designs, $\boldsymbol{X}$ is the expanded design matrix that has n rows (one for each design setting) and p columns (one column for each coefficient to be estimated plus one column for the overall mean). It was proved that a *D*-optimal design is also minimax, and on ther other hand, a minimax design is *D*-optimal (Kiefer and Wolfowitz, 1960).

This optimality criterion results in minimizing the generalized variance of the parameter estimates for a pre-specified model. As a result, the "optimality" of a given *D*-optimal design is model dependent. That is, the experimenter must specify a model for the design before a computer can generate the specific treatment combinations. Given the total number of treatment runs for an experiment and a specified model, the computer

algorithm chooses the optimal set of design runs from a *candidate set* of possible design treatment runs. This candidate set of treatment runs usually consists of all possible combinations of various factor levels that one wishes to use in the experiment.

Design of *D*-optimal experiments will be discussed in detail in Chapter 4. *D*-optimal experiments and maximum entropy sampling are basis of the method of Sequential Exploratory Experimental Design (SEED) developed in this dissertation.

2.5.2 Classical and Space-Filling Experimental Designs

Classical experimental designs are so named because they have been developed for what are considered to be the more "classical" applications of response surface metamodeling: physical experiments which are plagued by variability and random error (see, e.g., Box and Draper, 1987; Myers, et al., 1989; Myers and Montgomery, 1995). Among these designs, the factorial design, the central composite design (CCD), and face-centered central composite design (CCF) which is a special type of CCD, are well known and easily generated; thus they are utilized in designing experiments for the case studies in this thesis. A brief description of these three classical experimental designs could be found in (Lin, 2000).

As stated in Section 2.2, fractional factorial and central composite designs are integrated in RSM to help explore the design space and build RS models efficiently because of their sequential inherit property mentioned in the previous paragraph. This has been a plus of the RSM and also the classical experimental designs since space filling experiments are seldom designed for sequential usage.

Many researchers (see, e.g., Currin, et al., 1991; Sacks and Schiller, 1988) argue that classical experimental designs, such as the central composite designs and Box-Behnken designs, are not well-suited for sampling deterministic computer experiments. Sacks, et al. (1989) state that the "classical notions of experimental blocking, replication and randomization are irrelevant" when it comes to deterministic computer experiments which have no random error; hence, designs for deterministic computer experiments should "fill the space" as opposed to possess properties for estimating the variability in the data, as discussed in Section 2.2.

Numerous space filling experimental designs have been developed in an effort to provide more efficient and effective means for sampling deterministic computer experiments. For instance, Koehler and Owen (1996) describe several Bayesian and Frequentist types of space filling experimental designs, including maximin and minimax designs, maximum entropy designs, integrated mean squared error (IMSE) designs, orthogonal arrays, Latin hypercubes, scrambled nets and randomized grids. Latin hypercube designs were introduced in (McKay, et al., 1979) for use with computer codes and compared to random sampling and stratified sampling. Minimax and maximin designs were developed by Johnson, et al. (1990) specifically for use with computer experiments. Sherwy and Wynn (1987; 1988) and Currin, et al. (1991) use the maximum entropy principle to develop designs for computer experiments. Similarly, Sacks et al. (1989a) discuss entropy designs in addition to IMSE designs and maximum mean squared error designs for use with deterministic computer experiments. Finally, a review

of several Bayesian experimental designs for linear and nonlinear regression models is given in (Chaloner and Verdinelli, 1995).

Exploration of methods for sequential experimental design, together with the consideration of validation point selection (Section 2.2), is an important issue in metamodeling. In this dissertation, a method of Sequential Exploratory Experimental Design is proposed based on work in *D*-optimal design as discussed in this section and the maximum entropy sampling as will be introduced in the next section.

2.6 INFORMATION THEORY AND ENTROPY OPTIMIZATION PRINCIPLES

Information theory and entropy optimization are introduced in this section. The word entropy originated in the literature on thermodynamics around 1865 A.D. in Germany and was coined by Rudolf Clausius (Clausius, 1865) to represent a measure of the amount of energy in a thermodynamic system as a function of the temperature of the system and the heat that enters the system. The word entropy had belonged to the domain of physics until 1948 when Claude Shannon, while developing his theory of communication, used the term to represent a measure of information (Shannon, 1948). Since then, the concept of Shannon's entropy has penetrated a wide range of disciplines, including statistical mechanics (Jaynes, 1957), statistical inference (Tribus, 1969), business and finance (Cozzolino and Zahner, 1973; Yamada and Rajasekera, 1993), nonlinear spectral analysis (Shore, 1981), pattern recognition (Wang and Lu, 1992), transportation (Fang and Tsao, 1995), urban and regional planning (Kumar, et al., 1989; Scott and Jefferson, 1977), queueing theory (Guiasu, 1986), information theory (Shannon

and Weaver, 1962; Guiasu, 1977), parameter estimation, and linear and nonlinear programming (Fang and Tsao, 1993; Rajasekera and Fang, 1992). It is worth noting that, at the time when Shannon introduced his concept of entropy, no relationship, except for the similar mathematical expressions, was known to exist between Shannon's entropy and thermodynamics entropy. The relationship was only established later (Kapur and Kesavan, 1992).

The concept of entropy is closely tied to the concept of uncertainty embedded in a probability distribution. In fact, entropy can be defined as a measure of probabilistic uncertainty (the uncertainty associated with the probability of outcomes). Let $\mathbf{p} \equiv (p_1, p_2, \ldots, p_n)^T$ be a probability distribution associated with *n* possible outcomes, Shannon's entropy is defined as (Shannon and Weaver, 1962):

$$S_n(p) = -\sum_{j=1}^{n} p_j \ln p_j \tag{2.33}$$

where $\sum_{j=1}^{n} p_j = 1$, $0ln0 = 0$, $p_j \geq 0$ for $j = 1,\ldots,n$. Another formulation of Shannon's entropy, used as a measure of the uncertainty of the transmission of information, is:

$$H = -\int_{\Omega} p(s) \ln p(s) ds\,, \tag{2.34}$$

where *p(s)* is a Gaussian density function over the space Ω of the information signals transmitted. Such formulations of entropy can not only be used to measure "uncertainty" but can also be used to measure other concepts such as equality, disorder, diversity, lack of concentration, similarity, objectivity, unbiasedness, randomness, etc., and many other characteristics that do not even require probabilistic concepts for their description and

that have no relationship with uncertainty (Kapur and Kesavan, 1992). Thus, the word "entropy" has different meanings in different contexts, depending on how we define the p_i or $p(s)$ in its formulation.

Given the formulation of entropy, we can mathematically describe uncertainty in terms of entropy. We can choose the distribution that maximizes uncertainty subject to the given moment constraints. In this way, we make full use of all the information given to us but avoid making any assumption about any information that is not available. Such reasoning leads to the Maximum Entropy Principle: *Out of all possible distributions that are consistent with the moment constraints, choose the one that has the maximum entropy*.

Suppose now that, in addition to the constraints used in formulating Maximum Entropy Principle, we have an *a priori* probability distribution $\boldsymbol{p}^0$ that we think our probability distribution $\boldsymbol{p}$ should be close to. In fact, in the absence of the moment constraints, we might choose $\boldsymbol{p}^0$ for $\boldsymbol{p}$. However, with the presence of the moment constraints, we would choose the probability distribution that is the "closest" to the *a priori* distribution among those that satisfy the moment constraints. To be able to do so, we need a precise definition of "closeness" or "deviation". A simple measure for this "deviation" is the cross-entropy, also known as the Kullback-Liebler measure, which is defined as:

$$D(p, p^0) = \sum_{j=1}^{n} p_j \ln \frac{p_j}{p_j^0} \tag{2.35}$$

Note that whenever p_j^0 is 0, p_j is set to 0 and $0 \ln \frac{0}{0} = 0$. With cross-entropy interpreted as a measure of "deviation", we state the Minimum Cross Entropy Principle as: *Out of all possible distributions that are consistent with the moment constraints, choose the one that minimizes the cross-entropy with respect to the given a priori distribution*. Mathematical formulations of the entropy optimization principles can be found in (Fang, et al., 1997).

There is a diversity of entropy optimization principles besides the two mentioned above. To apply entropy and entropy optimization principles help solve many problems in various fields. In the field of design of experiments, entropy is usually used as a criterion (same role as IMSE, MMSE, minimax and maximin distance, discrepancy, etc.) to select an optimal design from a group of experimental designs (Ye, 1997), or choose a most informative subset of *s* random variables a set of *n* random variables (Lee, 2001). For details about maximum entropy designs, see (Lindley, 1956; Koehler and Owen, 1996; Sherwy and Wynn, 1987; Sherwy and Wynn, 1988; Currin, et al., 1991).

In this dissertation, entropy is used to help measure the information uncertainty associated with metamodels' prediction errors and achievement of design goals in engineering design. This leads to a sequential experimental design method with mathematical formulations similar to those from *D*-optimal designs. More details of entropy, and the application of entropy optimization in experimental design will be discussed in Chapter 4 in which the method of Sequential Exploratory Experimental Design (SEED) is developed based on research in *D*-optimal experiments and the maximum entropy sampling.

2.7 A LOOK BACK AND A LOOK AHEAD

Through the review of the literature which is presented in this chapter, the necessary knowledge for understanding and performing the proposed research in this dissertation is provided. The relationship between the research questions (and hypotheses) introduced in Section 1.3.2 and the techniques introduced in this chapter will be presented in this section, and the basis for studies in following chapters is laid.

As mentioned in Chapter 1, our research in this dissertation focuses on the development of sequential metamodeling and sequential design space exploration techniques. We propose to study the metamodeling techniques in the context of engineering design. The robust design space exploration as introduced in Section 2.1 provides the necessary engineering context of our proposed research.

The first step in our research is to examine the current metamodel validation techniques and develop new approaches to test the accuracy of metamodels. As illustrated in Section 2.2, metamodeling is necessary in early stages of design when there are expensive simulation programs. To validate the accuracy of a metamodel is needed to assure the achievement of right solutions. With deterministic computer experiments, statistics based on random errors, such as F-statistics, etc., are inappropriate. Our preliminary study also shows that the widely used method, leave-one-out cross-validation, may be incapable of testing the accuracy of metamodels. Thus, a close examination of leave-one-out cross-validation and development of new approaches to validate metamodels are necessary. This is mainly done in Chapter 3; some research in this direction is put in Chapter 4 and 5.

Various types of metamodels and their mathematics are presented in Section 2.3. This builds the foundation of our research on selection and usage of sequential metamodels along the design timeline, which will be discussed in Chapter 5.

D-optimal experiments and entropy optimization are briefly introduced in Sections 2.4 and 2.5, respectively. Design of *D*-optimal experiments and maximum entropy sampling will be discussed in detail in Chapter 4, as the basis for the proposed method of Sequential Exploratory Experimental Design. They are also the foundation of sequential design space exploration which will be studied in Chapter 6.

CHAPTER 3

METAMODEL VALIDATION WITH DETERMINISTIC COMPUTER EXPERIMENTS

In this chapter our focus is on the study of metamodel validation techniques in deterministic computer applications. Hypothesis 1 and its sub-hypotheses, SH1.1 and SH1.2.1, are tested in this chapter. A brief review of metamodel validation is put in Section 3.1. Sub-Hypothesis 1.1 is tested in Sections 3.2 and 3.3, in which we examine the performance of leave-one-out cross-validation from different viewpoints: our study in Section 3.2 is more theoretical and that in Section 3.3 more empirical. After proving that leave-one-out cross-validation is inappropriate for deterministic experiments in Sections 3.2 and 3.3, an approach to validation metamodels with additional validation points is proposed and tested in Section 3.4, where Sub-Hypothesis 1.2.1 is tested. A summary of research on metamodel validation is presented in Section 3.5.

The type of metamodel used in study in this chapter is the kriging model. However, we expect that our studies on cross-validate in this chapter are valid with other types of metamodels.

3.1 METAMODEL VALIDATION: CROSS-VALIDATION AND ADDITIONAL VALIDATION POINTS

As discussed in Chapter 2, for computer experiments the predicted performance is determined by the input variables and hence is deterministic and not based on random variation. This has a great influence in building metamodels for engineering design because, "In the presence of systematic error rather than random error, statistical testing is inappropriate" (Welch, et al., 1990). Several authors have reported statistical measures, such as the F-statistics and root MSE for verification of model adequacy, have no statistical meaning since they assume the observations include an error term which has mean of zero and a non-zero standard deviation.

When additional validation points can be afforded, the most important measures of model accuracy will be the root mean square error (RMSE) and the maximum absolute error (MAX) for the additional validation points. Formulations of RMSE and MAX are presented in the following equations:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n_{error}} (y_i - \hat{y}_i)^2}{n_{error}}} \tag{3.1}$$

$$MAX = \max \left| y_i - \hat{y}_i \right|, \ i=1,\ldots,n_{error} \tag{3.2}$$

where n_{error} is the number of random test points used, y_i is the actual value from the computer simulation, and $\hat{y}_i$ is the predicted value from the approximation model at validation points. The lower the value of RMSE and/or MAX, the more accurate the metamodel. RMSE is used to gauge the overall accuracy of the model; high values of

RMSE can lead a design space exploration into a region of bad design. MAX is used to gauge the local accuracy of the model; high values of MAX will cause local model inaccuracy (Lin, et al., 1999) and prevent the optimization algorithm from finding true solutions. Though previous experience recommends that a metamodel with normalized RMSE (RMSE divided by the sample range of responses) less than 5% and normalized MAX (MAX divided by the samples range of responses) less than 10% is acceptable for design space exploration at early design stages, there is no rigorously-defined guidance on model selection. Currently, with RMSE and MAX we cannot tell "how accurate" one metamodel is, and whether it meets the requirement of designers; what we can do is only to compare the accuracy of different models.

Leave-one-out cross-validation is probably the simplest and most widely used method for metamodels verification when additional validation points cannot be afforded. Leave-one-out cross-validation is a special case of cross-validation (Hastie, et al., 2001). In this approach, each sample point used to fit the model is removed one at a time, the model is rebuilt without that sample point, and the difference between the model without the sample point and actual value at the sample point is computed for all of the sample points. The cross-validation root mean square error (CVRMSE) is computed as below:

$$CVRMSE = \sqrt{\frac{\sum_{i=1}^{n_s} (y_i - \hat{y}_i)^2}{n_s}} \tag{3.3}$$

Note that only information from the n_s data points is needed in calculating CVRMSE; there is no need to collect information from additional validation points as we

do in Equations (3.1) and (3.2). The metamodel used in this chapter is the kriging model. In developing kriging models, unless there are very few data points or major outliers, usually we do not rebuild the kriging model since dropping a single observation usually has a negligible effect on the maximum likelihood estimates. The parameters estimated using all data points are used, together with the correlation matrix **R** and vectors **r** and ***y*** from the remaining (n_s – 1) points, to calculate the cross-validation root mean square error (Jones, et al., 1998). Similar to RMSE and MAX, it is believed that a smaller CVRMSE values indicates a more accurate metamodel.

3.2 THEORETICAL STUDY OF LEAVE-ONE-OUT CROSS-VALIDATION

The research question to be answered in this section is R.Q.1.1: *Is leave-one-out cross-validation a suitable method of metamodel validation with computer experiments?* The corresponding hypothesis is Sub-Hypothesis 1.1: Leave-one-out cross-validation is not an appropriate method of metamodel validation with deterministic computer experiments.

As stated in Section 3.1, the root mean square error (RMSE) in Equation (3.1) is the most reliable measurement for model accuracy when we have sufficient additional validation points. Leave-one-out cross-validation is used with the purpose of saving computation expense since it deals with only information from sample data points. In this section, we will study the performance of leave-one-out cross-validation in measuring accuracy of metamodels and illustrate its weakness with two single-variable

functions. These functions are treated as computer simulations; information at sample data points is collected, then kriging models are developed and validated.

The first single-variable function used in our study is originally taken from (Su and Renaud, 1996). The function is:

$$f(x) = \sum_{i=1}^{9} a_i (x - 900)^{(i-1)} \tag{3.4}$$

where:

$a_1 = -659.23$
$a_2 = 190.22$
$a_3 = -17.802$
$a_4 = 0.82691$
$a_5 = -0.021885$
$a_6 = 0.0003463$
$a_7 = -3.2446 \times 10^{-6}$
$a_8 = 1.6606 \times 10^{-8}$
$a_9 = -3.5757 \times 10^{-11}$

In this study we select the design space from $x = 912$ to $x = 1000$. In this design space, the maximum response value is 182.77 at $x = 1000$, and the minimum response value is around 13.96 at around $x = 932$; the response range is 168.81. A graph of this function is shown in Figure 3.1.

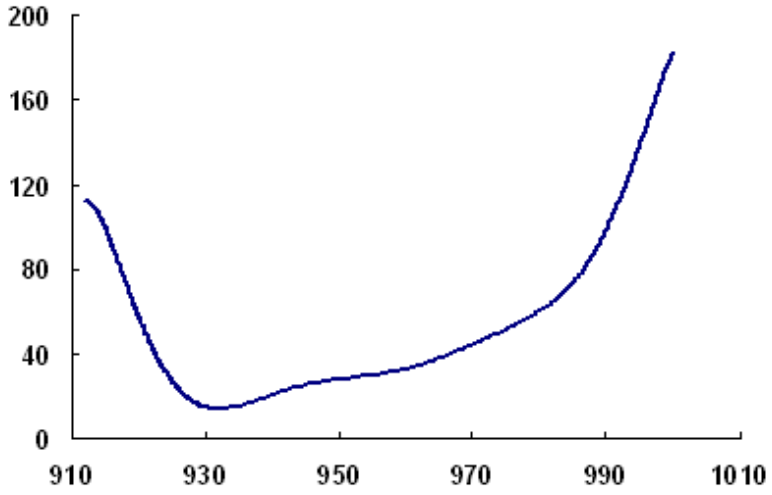


Figure 3.1 A Single-Variable Function (Su and Renaud, 1996)

To facilitate our study two kriging models are developed based on information from two different sets of data points, as shown in Table 3.1. For Data Set I, sample data points are "clustered" in the intervals of x = [912, 922] and [990, 1000], while data points in Data Set II are more evenly spreading over the whole design space. It is expected that Data Set II conveys more information and will afford more accurate metamodels.

Table 3.1 Response Values at Sample Data Points of the Single-Variable Function

Data Set I						
x	912	917	922	990	995	1000
y	112.08	84.43	43.98	97.98	137.56	182.77
Data Set II						
x	912	932	945	960	986	1000
y	112.08	13.96	25.20	32.92	77.31	182.77

Values of θ for kriging models are obtained by maximizing Equation (2.22) subject to $\theta>0$. In this case, we get $\theta= 28.4626$ for Data Set I and $\theta= 14.49733$ for Data Set II. The kriging models contain matrix expressions and are complicated; thus they are not listed here. However, the graphs of the two metamodels, which could help us get an idea on models' accuracy, are shown in Figure 3.2.

It is clearly seen from Figure 3.2 that the kriging model with Data Set II approximates the actual function better than the one with Data Set I. Comparison of RMSE and MAX for both models gives more concrete judgments. In order to calculate RMSE and MAX to validate the metamodels, for each kriging model we select 875 validation points evenly spreading from x = 912 to x = 1000 (not including the sample data points). RMSE and MAX values are listed in Table 3.2.

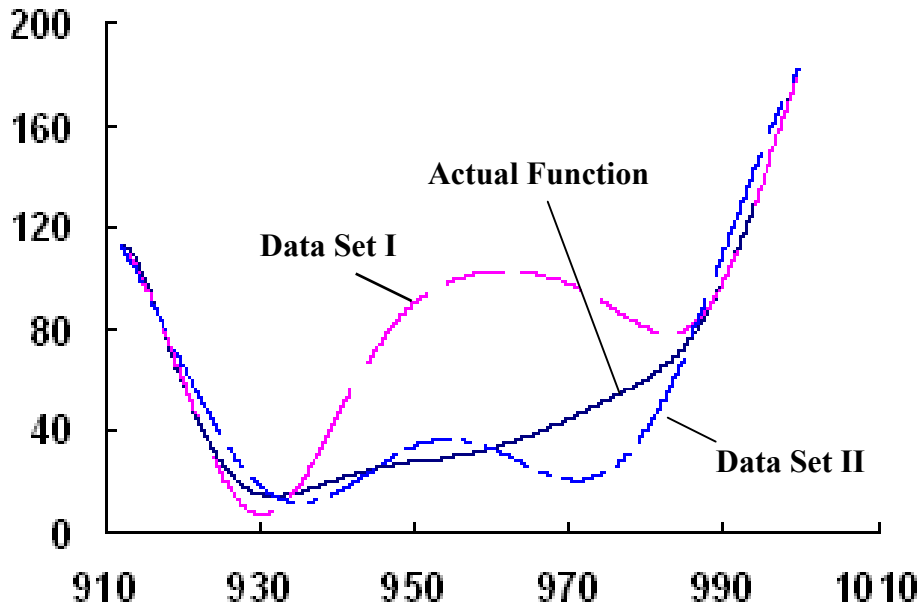


Figure 3.2 Kriging Models for the Single-Variable Function

Table 3.2 RMSE and MAX for Kriging Models

	RMSE	MAX
Data Set I	37.78	69.79
Data Set II	11.93	27.84

RMSE and MAX values listed in Table 3.2 support our claims that the kriging model with Data Set II is more accurate than the one with Data Set I since it has significantly smaller RMSE and MAX values. It is in accordance with our expectations too. The poor experimental design for Data Set I fails to reflect information in the middle of the design space.

Now let us compare the accuracy of these two kriging models with leave-one-out cross-validation, in which only information at sample data points are used with Equation (3.3). Information at 875 additional validation points is not used for cross-validation. In the calculation of CVRMSE for each kriging model in cross-validation, since there are only six sample data points as listed in Table 3.1, we decide to rebuild kriging models to predict responses at each data point using the other five data points. Graphs of these kriging models (one original kriging model plus six secondary kriging models for each data set) are shown in Figure 3.3 and Figure 3.4. Values for CVRMSE are listed in Table 3.3.

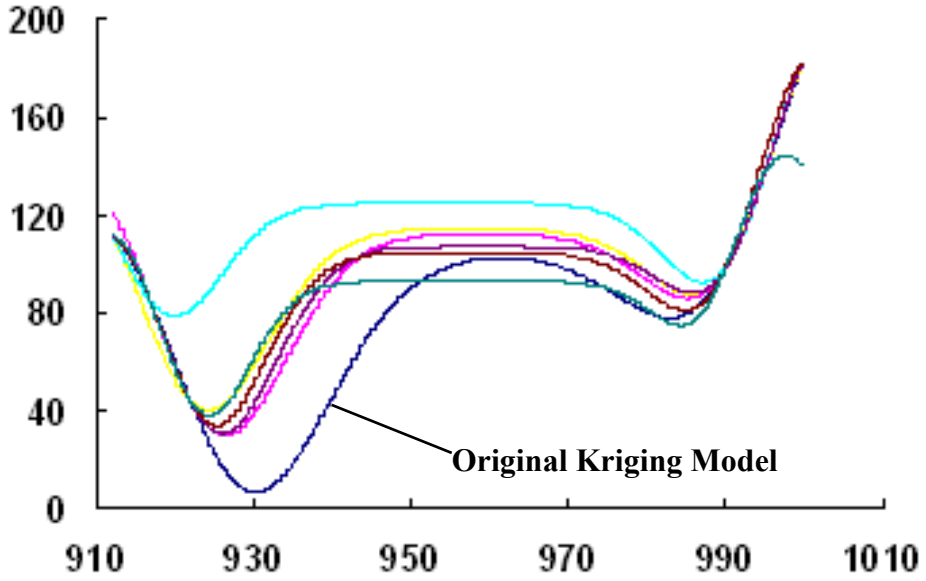


Figure 3.3 Kriging Models for Calculating CVRMSE with Data Set I

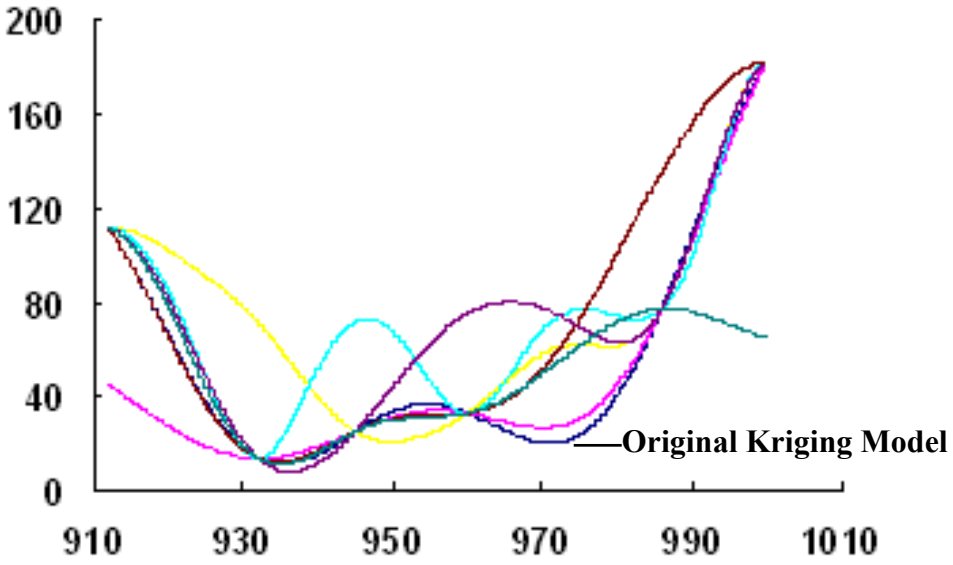


Figure 3.4 Kriging Models for Calculating CVRMSE with Data Set II

Table 3.3. CVRMSE Values for Kriging Models

	Data Set I	Data Set II
CVRMSE	24.21	69.60

From Table 3.3, we see that CVRMSE for Data Set I, which is 24.21, is much smaller than that for Data Set II, 69.60. This suggests that the kriging model with Data Set I is more accurate than the one with Data Set II. This is contrary to our conclusions with RMSE and MAX. Given that RMSE and MAX are the most reliable measurements, this observation shows that leave-one-out cross-validation may be insufficient for model validation.

Examination of kriging model plots in Figure 3.3 and Figure 3.4 helps us see the weakness in leave-one-out cross-validation. For Data Set I, since the data points are more clustered (only in the intervals [912, 922] and [990, 1000]), there is more "overlap" in the information they convey. Thus in leave-one-out cross-validation, to remove any one point may not significantly reduce the total amount of information conveyed and will not change the metamodel greatly. This "clustering" or "information overlap" of the data points results in a metamodel that is insensitive to removal of data points – which means lost information at any data point could be retrieved with only the model and the other data points. This idea is illustrated in Figure 3.3, in which we see that all models with five data points share curves similar to that of the model with six data points; consequently, we get a small CVRMSE for the original kriging model with Data Set I.

For Data Set II, data points spread all over the design space and there is little "information-overlap" among them. In this case the corresponding metamodel is more

affected by the removal of some data points – it is unlikely to retrieve the lost information with the metamodel. As shown in Figure 3.4, kriging models with five data points are very different from the original kriging model; consequently, we get a large CVRMSE for the original kriging model with Data Set II.

Observations above suggest that leave-one-out cross-validation is an insufficient measurement for metamodel accuracy. On the other hand, leave-one-out cross-validation is a good method for measuring the sensitivity of a metamodel to lost information due to the removal of some of its data points. A small value of CVRMSE indicates a metamodel that is more insensitive to lost information; a large value of CVRMSE indicates a metamodel that is sensitive to removal of data points. A discussion is conducted later in this paper on the sensitivity of metamodels to lost information at data points.

In the case above, **clustering data points** (information overlap) is the cause of an inaccurate metamodel that is also insensitive to lost information at data points. The insensitivity here may mislead designers since small CVRMSE values may be obtained for inaccurate metamodels in leave-one-out cross-validation. A space-filling experimental design for allocating data points may help avoid this situation because data points in a space-filling design tend to spread over the whole design space and this minimizes the information overlap. However, clustering data points (information overlap) is not the only cause for inaccurate metamodels which are also insensitive to lost information at data points, as shown in the following paragraphs with another single-variable function:

$$y = 2\sin\left(\pi\left(x - 0.5\right)\right) + 5 \tag{3.5}$$

In our study we set the continuous variable x = [0, 10]. If five data points are to be selected, following the "space-filling" rule, we may select x = 1, 3, 5, 7, 9, as presented by solid stars in Figure 3.5, which have the same response value, y = 7. It is apparent that the corresponding kriging model is a constant y = 7, shown as a horizontal line in Figure 3.5.

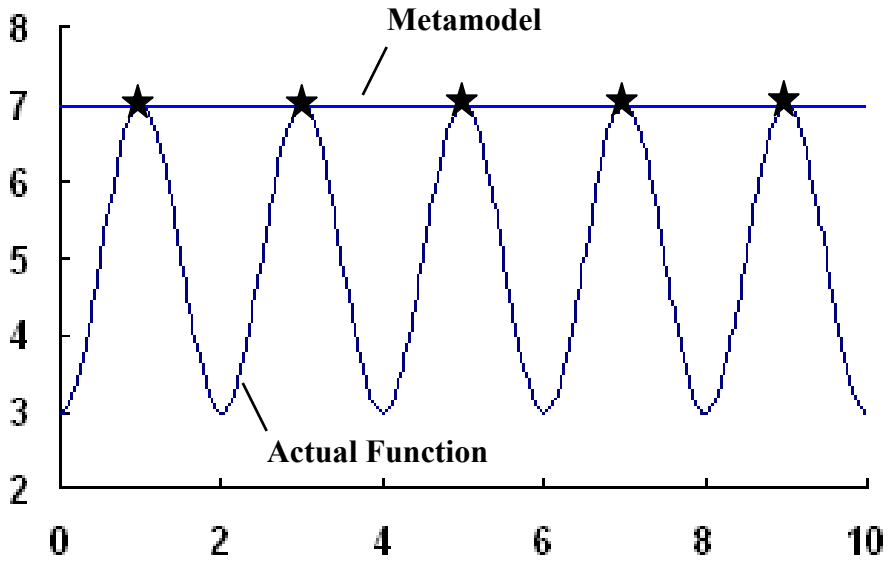


Figure 3.5 Inaccurate Metamodel Due to the Correlation Among Data Points

The kriging metamodel shown in Figure 3.5 is by no means acceptable. However, leave-one-out cross-validation (actually, not only leave-one-out cross-validation, but also *kth*-folder cross-validation with k less than 4 in this case) shows that this metamodel is perfectly accurate because the value of CVRMSE is zero. The kriging model is totally insensitive to lost information at data points, i.e., information at any

missing data point can be 100% retrieved with the metamodel itself. Though in the real world it is very rare to meet such situations as in Figure 3.5, this extreme example helps illustrate how great a mistake that leave-one-out cross-validation is possible to make in assessing metamodels.

There is no clustering of data points (little information overlap) in this example. The metamodel's insensitivity to lost information at data points is the result of another cause, which could be called "inappropriately correlated data points", representing a set of points whose ***x***'s and ***y***'s share a similar pattern and this pattern is very different from the actual function for which we develop metamodels. There may be various types of "inappropriate correlations" between data points, e.g., ***x***'s and ***y***'s of a set of data points may follow a quadratic or an exponential function, while the actual function may be much more complicated. In the case of Figure 3.5, the **"inappropriate correlation" among data points** is that they share the same response value – here the pattern is a constant-response function which is much different from the actual *sin* function.

Not all correlations among data points are bad. Actually, an accurate metamodel can only be developed with "appropriately correlated" data points whose ***x***'s and ***y***'s follow a pattern similar to (or ideally, the same as) the original actual function. The only difference, between "appropriately correlated" and "inappropriately correlated" sets of data points, is whether they follow a pattern that gives a similar response surface to the actual function or not. Unfortunately, with information only from some data points it is very difficult to tell whether a data set is appropriately correlated or not; additional validation points are necessary. This also shows that leave-one-out cross-validation is an

insufficient method for metamodel assessment; RMSE and MAX are more appropriate since they employ information at not only data points but also validation points.

To avoid employing inappropriately correlated sets of data points, it is very helpful to increase the total number of data points and design space-filling experiments. In this way we expect to have data points provide as much information as possible for regions as large as possible – though we still cannot assure the data points are appropriately correlated.

Leave-one-out cross-validation, insensitivity of metamodels to lost information at data points, and clustering and inappropriately correlated data points will be further discussed in the next section with a two-variable function.

3.3 IMPIRICAL STUDY OF LEAVE-ONE-OUT CROSS-VALIDATION

In this section, discussions on leave-one-out cross-validation in model assessment are further conducted with the case of a two-variable function – the Branin function (Dixon and Szego, 1978):

$$f = \left(x_2 - \frac{5.1}{4\pi^2} x_1^{\,2} + \frac{5}{\pi} x_1 - 6 \right)^2 + 10\left(1 - \frac{1}{8\pi} \right) \cos\left(x_1 \right) + 10 \tag{3.6}$$

where $x_1 = [-5, 10]$ and $x_2 = [0, 15]$. The 3-D wire-frame plot for Equation (3.6) is shown in Figure 3.6. In the design space the Branin function has three local minima at $\boldsymbol{x}$ = {3.1416, 2.2750}, {9.4248, 2.4750}, and {–3.1416, 12.2750} with identical function values of 0.3979. The maximum response is y = 308.1291 at $\boldsymbol{x}$ = {–5, 10}. The response range is 307.7312.

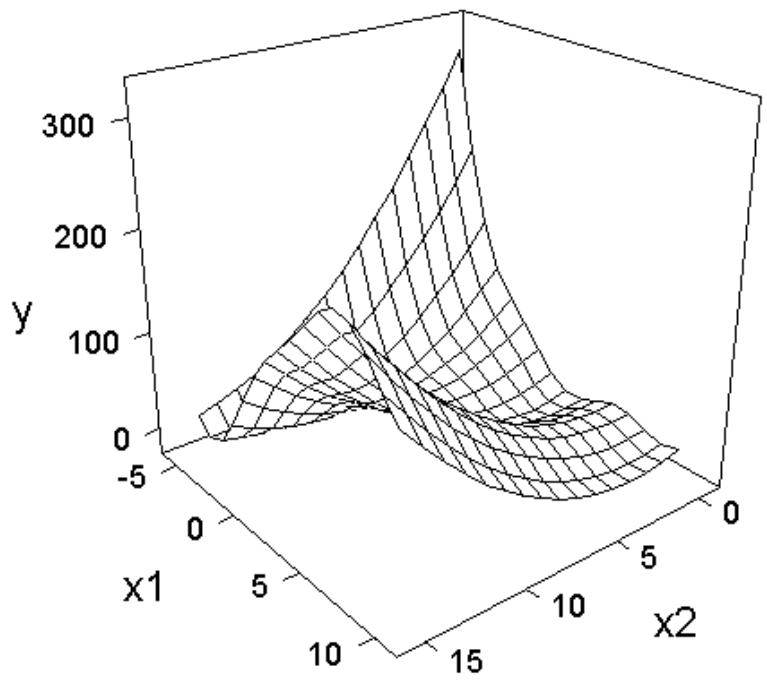


Figure 3.6 Wire-Frame Plot of the Branin Function

To facilitate our study with the Branin function, 18 sets of data points are selected and kriging models are developed for each data set. The numbers of data points in the data sets range from 9 to 22, as shown in Table 3.4. Thirteen of these 18 experimental designs, Data Sets 1, 3, 4, 6 – 11, 13, 14, 16, and 17, are Latin Hypercube (LH), one Orthogonal Array (OA) – Data Set 18, and four randomly selected points (S) – Data Sets 2, 5, 12, and 15. The LH and OA experiments are designed with iSIGHT®. Data Sets 5 and 12 are experimental designs with clustered or inappropriately-correlated data points, as will be shown later. Data points in Data Sets 2 and 15 are identified to convey critical information about the actual response surface given that we know the actual function.

Detailed information about the experiments and corresponding kriging models is not presented here due to space limitation.

Table 3.4 Values of RMSE, MAX, and CVRMSE of Kriging Models for the Branin Function

Data Set	DOE	# of Data Points	CVRMSE	RMSE	MAX
1	LH	10	58.23	20.56	65.26
2	S	18	27.71	7.09	19.39
3	LH	12	62.09	20.31	72.37
4	LH	19	5.62	4.79	24.66
5	S	13	8.31	54.49	206.47
6	LH	15	22.54	46.62	263.49
7	LH	16	10.03	27.83	183.47
8	LH	17	39.78	28.23	101.22
9	LH	18	9.18	6.89	34.25
10	LH	13	26.39	18.75	76.56
11	LH	14	28.85	27.07	100.27
12	S	18	2.08	76.67	298.59
13	LH	22	3.85	9.65	85.42
14	LH	11	37.77	31.78	165.64
15	S	15	64.27	9.06	25.07
16	LH	20	2.46	8.46	54.35
17	LH	21	8.41	4.00	25.55
18	OA	9	74.37	41.65	118.81

The accuracy of kriging models is examined with information from 255 validation points that are spread all over the design space; however, for some data sets the number of validation points may be less because some points are already listed in those data sets and cannot be used to validate the corresponding metamodels. We assume that: 1) in this case 255 validation points are enough for model validation, and then 2) the RMSE and MAX calculated with these points are regarded as unbiased measurement of model accuracy. CVRMSE for each kriging model are also calculated. Values of RMSE,

MAX, and CVRMSE for 18 kriging models are listed in Table 3.4. To fit in the table, values in Table 3.4 are rounded to two decimals. Note that with RMSE, MAX, or CVRMSE, we cannot decide whether a kriging model is acceptable or not; we can only compare two kriging models – the ones with smaller RMSE, MAX, or CVRMSE are considered to be more accurate as stated in our frame of reference.

In Table 3.4 we see that there is no critical relationships between CVRMSE and RMSE (or MAX). Kriging models with small values of CVRMSE, e.g., Data Set 5 and 12, may have very large values for RMSE and MAX, while those with large values of CVRMSE, e.g., Data Set 15, may have very small values for RMSE and MAX. The correlation coefficients between CVRMSE and RMSE/MAX are nearly zero, which shows that they have no significant linear correlations. To use CVRMSE to compare the accuracy of two metamodels may lead us to a wrong answer since RMSE and MAX are believed to be most reliable measurements.

Plots of RMSE and MAX versus CVRMSE are given in Figure 3.7 and Figure 3.8. If CVRMSE is linearly correlated with either RMSE or MAX, points in these figures should lie on a straight line. Since the data is widely scattered, CVRMSE is not correlated with either RMSE or MAX.

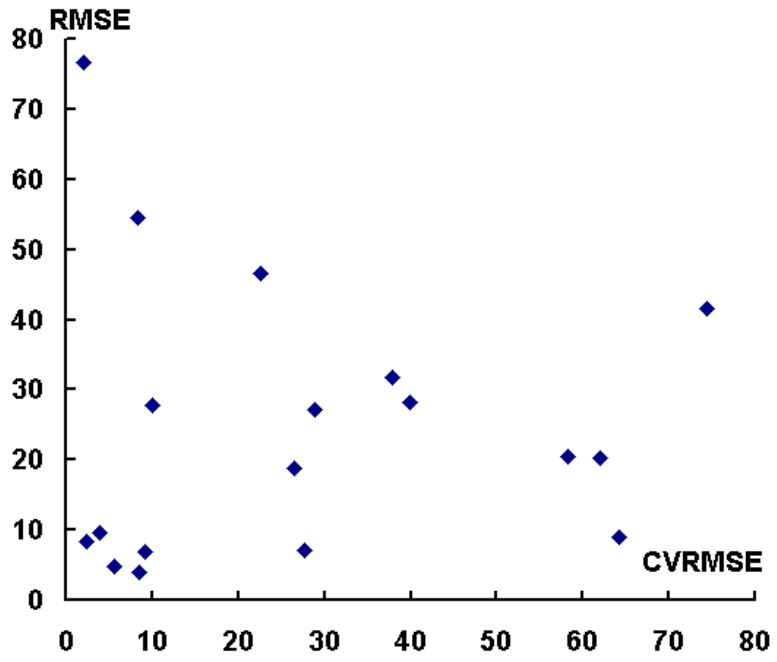


Figure 3.7 Scatter Plot of RMSE and CVRMSE for Kriging Models for the Branin Function

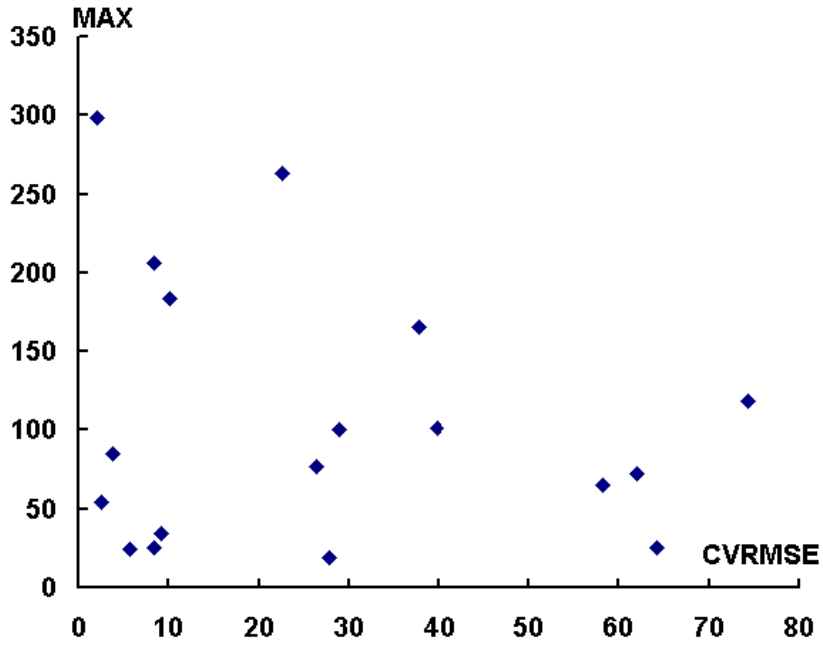


Figure 3.8 Scatter Plot of MAX and CVRMSE for Kriging Models for the Branin Function

Now let us look at Data Set 5 and 12, whose kriging models have small values of CVRMSE and large values of RMSE and MAX. Information about the data points for Data Set 5, 12, and 15 is listed in Table 3.5 (only one decimal is shown in the table due to space limitation). These two metamodels for Data Sets 5 and 12 are inaccurate and also insensitive to lost information at data points. Data Set 5 is a clustered experimental design in which data points are clustered in two regions, around points [0, 5] and [7, 11]. Data Set 12 is not only clustered, but also inappropriately correlated; the response range of its data points is less than 15 (note that the range of actual function values in the design space is about 308). This observation indicates: 1) with leave-one-out cross-validation designers are in danger of accepting an inaccurate metamodel that is insensitive to lost information at data points, and 2) inaccurate and insensitive metamodels are the results of poor experimental designs (clustering points or correlated data points) – space-filling experimental designs are recommended.

The case of Data Set 15 is different from those of Data Sets 5 and 12. Data points in this set are so well distributed that most waves in the response surface are captured with very few points. Each data point is set at a very critical position on the actual response surface where "waves" take place. Information overlap between data points is very little, thus each of them conveys a great deal of information and to lose any of them will substantially affect the metamodel. Without information from one data point the predicted response surface will be very different from the actual one. This is why the kriging model for Data Set 15 has a very large value for CVRMSE and small values for RMSE and MAX. Actually it is an accurate model that is also sensitive to lost

information at data points. The observation here indicates: 1) with leave-one-out cross-validation we are in danger of rejecting an accurate metamodel that is also sensitive to lost information at data points, and 2) a good metamodeling process (both efficient and effective, or get most accurate metamodel with least effort) may have large CVRMSE values and small RMSE and MAX values.

Table 3.5 Data Points for Data Set 5, 12, and 15

Data Set 5			Data Set 12			Data Set 15		
x_1	x_2	y	x_1	x_2	y	x_1	x_2	y
0	5	20.6	-2.5	12.5	5.2	-3.5	15	4.4
1	7	21.3	-3	13	1.6	-2	14	24.5
2	6	13.1	-4	14	4.0	6.5	14.5	198.6
3	8	32.0	-2	12	11.3	-3	12	0.5
-1	2	47.9	-2.5	13.5	9.7	-2.5	6	25.2
-2	3	50.9	-3.5	12.3	1.7	-5	0	308.1
-3	4	63.5	3.5	2.5	1.2	3	2.5	0.5
8	10	80.3	3	2	0.6	0	5.5	19.9
7	11	113.5	4	3	5.4	9.5	3	0.6
6	8	66.8	3.5	1.5	1.3	6	4	27.6
5	14	174.7	2.5	4	3.7	10	0	11.0
9	15	166.6	4.5	2.2	8.5	4.5	6	28.6
10	13	101.9	9.5	3	0.6	10	7.5	22.2
			9	4	4.7	0	0	55.6
			8	3	10.7	2	0	17.1
			8.5	3.5	7.1			
			9.5	4	2.6			
			8.2	3.6	10.6			

Now let us have a look at the performance of space-filling experiments. By removing Data Sets 2, 5, 12, 15, and 18, we have only Latin Hypercube experiments (a type of space filling experiments) left. In this case, Figure 3.7 and Figure 3.8 will not

change a lot; we still cannot see strong correlation between CVRMSE and RMSE (or MAX). We also observe that as the number of data points increases the corresponding metamodel becomes more and more accurate. When we have more than 20 data points in this case, both CVRMSE and RMSE (or MAX) become small; these metamodels are accurate, and also insensitive to lost information at data points. The metamodels' insensitivity to lost information indicates that we may have used redundant data points – it is possible to develop metamodels at the same level of accuracy with fewer data points, as we do with Data Sets 2 and 15. However, in this case, with sufficient data points in a space filling experimental design, we avoid either clustered or inappropriately correlated data points and are able to develop accurate metamodels.

In (Simpson, 1998) the author performs an empirical study on the relationship between CVRMSE and RMSE (or MAX). For six simple engineering problems he applied fifteen different types of experimental designs (for each type of experimental design there are various options on how many data points to be allocated) and developed corresponding kriging models with five different types of correlation functions. Overall, 11535 kriging models are constructed and values of CVRMSE, RMSE, and MAX are calculated. To eliminate effects of different units from different responses, the normalized CVRMSE, RMSE, and MAX are calculated by dividing the original values with the sample range of each data set. Plots of normalized RMSE (MAX) versus normalized CVRMSE are shown in Figure 3.9 and Figure 3.10. These plots indicate that there are no correlations between CVRMSE and RMSE (MAX); this supports our claims

that leave-one-out cross-validation is insufficient for model assessment and information from additional points is essential in validating metamodels.

In this section, our studies and observations show that leave-one-out cross-validation is insufficient for metamodel assessment. The reason is that leave-one-out cross-validation is actually a measurement for degrees of insensitivity of a metamodel to lost information at data points, while a metamodel which is insensitive to lost information at its data points is not necessarily an accurate metamodel. There are two causes for this insensitivity: clustering or inappropriately correlated data points. Designing space-filling experiments with sufficient number of data points is one way to prevent an inaccurate metamodel that is insensitive to lost information. With the case of the Branin function, we observe that with space filling experiments we may get accurate metamodels which are insensitive to lost information at its data points.

The conclusion here does not mean that previous applications with leave-one-out cross-validation are necessarily wrong. When the original actual function is not highly nonlinear (or the design space is not very large) and there are enough data points spread all over the design space, the danger of having clustering or inappropriately correlated data sets is small. However, the success of leave-one-out cross-validation in those examples is dependent on particular cases; real-world applications are usually more complicated and cannot meet the requirements mentioned above. Given that cross-validation is insufficient for assessing models, employing additional points is essential in metamodel validation.

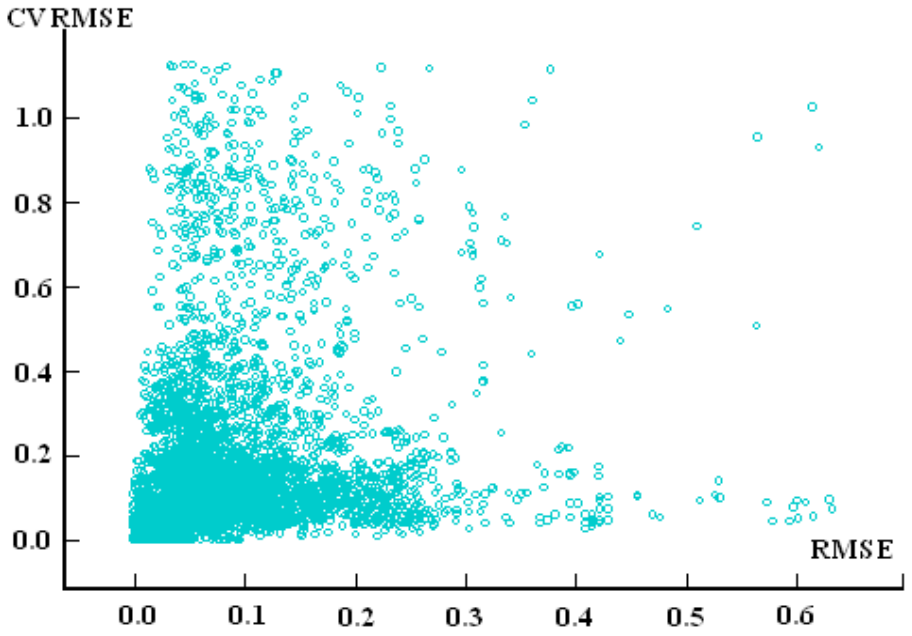


Figure 3.9 Correlation of Normalized CVRMSE and RMSE (Simpson, 1998)

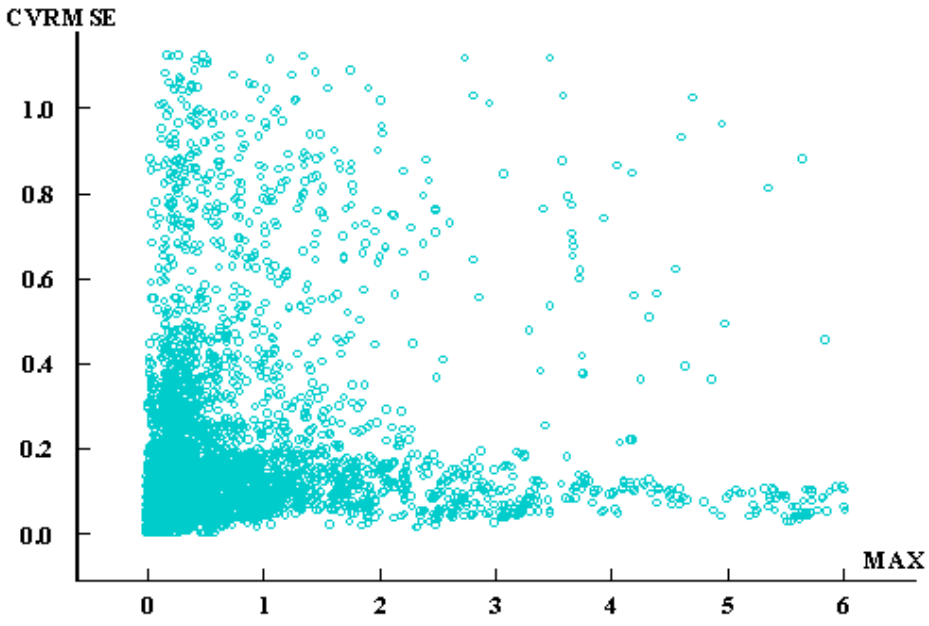


Figure 3.10 Correlation of Normalized CVRMSE and MAX (Simpson, 1998)

3.4 METAMODEL VALIDATION WITH INFORMATION FROM ADDITIONAL VALIDATION POINTS

As discussed in the previous section, it is necessary to use additional validation points for metamodel validation with computer experiments since leave-one-out cross-validation is insufficient. In this section, we will explore how to use this additional information to validate metamodels. Research question to be answered in this section is R.Q.1.2: *How to test the accuracy a metamodel in deterministic applications?* To answer this research question Sub-Hypothesis 1.2.1 needs to be tested: the accuracy of a metamodel could be validated through examining prediction errors at additional validation points.

Without information from additional points, the general formula for the prediction mean squared error at any new point for a kriging model is (Sacks, et al., 1989a):

$$\boldsymbol{s}^2(\boldsymbol{x}^*) = \sigma^2\left[1 - \boldsymbol{r}^T\boldsymbol{R}^{-1}\boldsymbol{r} + \frac{\left(1 - \boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{r}\right)^2}{\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{f}}\right] \tag{3.7}$$

The term $-\boldsymbol{r}^T\boldsymbol{R}^{-1}\boldsymbol{r}$ represents the reduction in prediction error due to the fact that $\boldsymbol{x}^*$ is correlated with the sampled points. The σ^2 here is the same as in Equation (2.21). The term $(1-\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{r})^2/\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{f}$ reflects the uncertainty that stems from our not knowing μ exactly, but rather having to estimate it from the data. The prediction error in Equation (3.7) is σ reduced by an amount that depends on how correlated the new point is to the sampled points. However, the prediction error in Equation (3.7) is based on the kriging model with no-random error only; it is more reliable to take information from validation points into account when assessing the accuracy of a kriging model.

Though one important benefit of using metamodels is to save expense in experiments, the addition of validation points, which eventually increases time and effort on computer simulations, does not hurt the importance of metamodeling very much. First, in computer experiments, moderate increases of computational expenses are usually affordable with fast computers in a distributed design environment. The situation here is different from that of physical experiments, e.g., collision analysis for vehicles, etc., in which the total number of experiments may be strictly restrained due to limits on material expenses. Second, as stated in our frame of reference, the use of metamodels not only helps us save experimental expenses but also integrates simulation codes from different disciplines to give insight into the relationships between input variables and output responses.

Two problems in metamodel validation with additional points are: 1) how many additional validation points should be used, and 2) how to allocate these validation points. The number of validation points should not be large in order to save computation time and effort, and it should not be too small to assure an "accurate" assessment. Information from the current kriging model may provide useful guidance: for a design space with highly nonlinear response surfaces where a single point provides little information, we should use a large number of validation points to gain enough information, while for a design space with smoother actual response surfaces we may use less. To decide where to put the additional validation points is a difficult problem under study by many researchers; tools from statistics, information theory, etc., may help develop methods for point allocation. To identify a reasonable number of validation

points is beyond our discussion here; in this chapter we will use large number of random points to validate the metamodels. The selection of validation points is studied in Chapter 4 in which a method for sequential experimental design is developed by utilizing validation points.

3.4.1 Preliminary Methods of Metamodel Validation for Engineers

In this section we present some ideas for engineers to gain knowledge in assessing the metamodels' accuracy with additional validation points. We assume that the number of validation points is large and the residuals at validation points follow normal distributions (though in some cases this needs to be verified). With kriging models we also assume that the parameters in kriging equations are known.

Assuming the degrees of freedom to be n_{error} (note that the additional validation points are not used in model fitting), the RMSE value from Equation (3.1) can be taken as the standard deviation s with $E(s^2) = \sigma_v^2$. Note that σ_v here represents the population standard deviation in model validation. We use s_{pred} to represent the prediction standard deviation at a new point; since we assume the distribution to be normal, the value $\frac{\hat{y} - y}{s_{pred}}$ is distributed as $t(n_{error})$. Then the $(1 - \alpha)$ prediction limits are:

$$\hat{y} \pm t(1 - \alpha / 2; n_{error}) s_{pred} \tag{3.8}$$

If we have some preset bounds, $\pm\Delta$, for the prediction error, the average confidence level that our predictions will fall in this range can be calculated using:

$$t(1-\alpha/2; n_{error}) \le \frac{\Delta}{s_{pred}} \tag{3.9}$$

In practice, we could use RMSE calculated from Equation (3.1) to replace S_{pred} in Equation (3.8). In making this replacement, we are actually seeking an approximate reference on how the kriging model performs on average in prediction at testing points; we also assume S_{pred} to be the same for new points, and thus Equation (3.8) gives the same size of prediction intervals for any new point. This is not a perfect method, but it is very simple and provides preliminary inspection of the kriging model we are studying.

A more accurate method is to study the validity of Equation (3.7) in calculating prediction errors by studying several plots. It is claimed that with a kriging model, we are approximately 99.7% confident (calculated based on normal distribution) that the predicted value at a new point lies in the interval of $\hat{y} \pm 3s(x^*)$, in which $s(\boldsymbol{x}^*)$ is calculated from Equation (3.7). Thus it is important to see whether the observed $\hat{y}$ at validation points lie in this interval or not. Instead of drawing confidence intervals, we can compute the number of standard errors that the actual value is above or below the predicted value, which we call the standardized residual:

$$\frac{y(x^*) - \hat{y}(x^*)}{s(x^*)} \tag{3.10}$$

Since we have n_{error} prediction points, there will be n_{error} standardized residuals from Equation (3.10). If these values are roughly in the interval [–3, +3], we say the kriging model correctly anticipates the magnitude of the prediction errors – thus we can use Equation (3.7) to calculate prediction errors. As shown later, a plot of standardized

residuals versus predicted values is very helpful in this study. To assure normal distribution, a normal probability plot of the standardized residuals versus the values that would be expected from a random sample of n_{error} independent standard normal variables needs to be drawn. The correlation coefficients of standardized residuals and normal distributed samples may also be calculated. Another useful plot is the plot of actual function values versus predicted values. Plots of predicted standard errors calculated from Equation (3.7), or actual residuals $(y - \hat{y})$, versus actual function values may help in study the performance of kriging models in prediction over the whole design space. These methods are illustrated in the next subsection with the Branin function.

3.4.2 Metamodel Validation with the Branin Function

In this study we use Data Set 15 as presented in Table 3.5. The RMSE value (s_{pred}) for the kriging model is 9.057 based on n_{error} = 248 validation points. At early design stages we need a metamodel with prediction errors in the range of ±15 on average; thus the value of Δ is about 5% of the sample range 307.63. The value of $\frac{\Delta}{s_{pred}}$ is about 1.67. With the information above, our calculation shows that we have about 90% confidence that on average the prediction value falls in the required limits. If we want to obtain the same level of confidence for the kriging model with Data Set 5 whose RMSE value is 54.49, the acceptable bound is about ±90.

To facilitate further study, mean squared errors for prediction at 248 validation points are calculated with Equation (3.7) using only information from the kriging model and the data points. The standardized residuals are then calculated for all validation

points following Equation (3.10). This information is then used in our study to help test whether the kriging model is good in prediction of both response values and variances.

Three plots are drawn, as shown in Figure 3.11, Figure 3.12, and Figure 3.13. In Figure 3.14, we see that the points follow the 45^{o} line; the Pearson product moment correlation coefficient between these two sets of values is 0.993. These roughly show that the metamodel is not bad in prediction of response values. In Figure 3.12 we see that the standardized residuals act roughly like normal deviates; the Pearson product moment correlation coefficient of standardized residuals with normally distributed samples is 0.99. Figure 3.13 provides two important pieces of information. First, all points fall within the three-standard-error limit – actually most of the standardized residuals at validation points are less than two standard errors (note that the standard errors are calculated with Equation 15, and are different at different validation points), which shows that to predict response errors with Equation 15 is acceptable. Second, points in Figure 3.13 follow a quadratic curve trend; ranges of the standardized residuals tend to increase when the predicted function values increase. This suggests a systematic bias in prediction with the kriging model, thus the plot of actual residuals $(y - \hat{y})$ versus actual function values, as shown in Figure 3.14, is very important for further analysis.

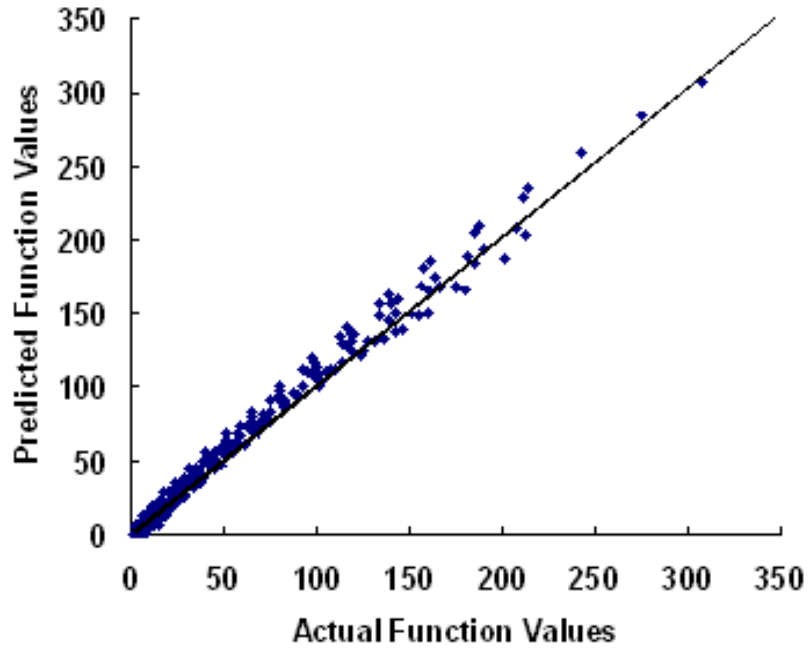


Figure 3.11 Plot of Predicted Values Versus Actual Function Values for the Branin Function with Data Set 15

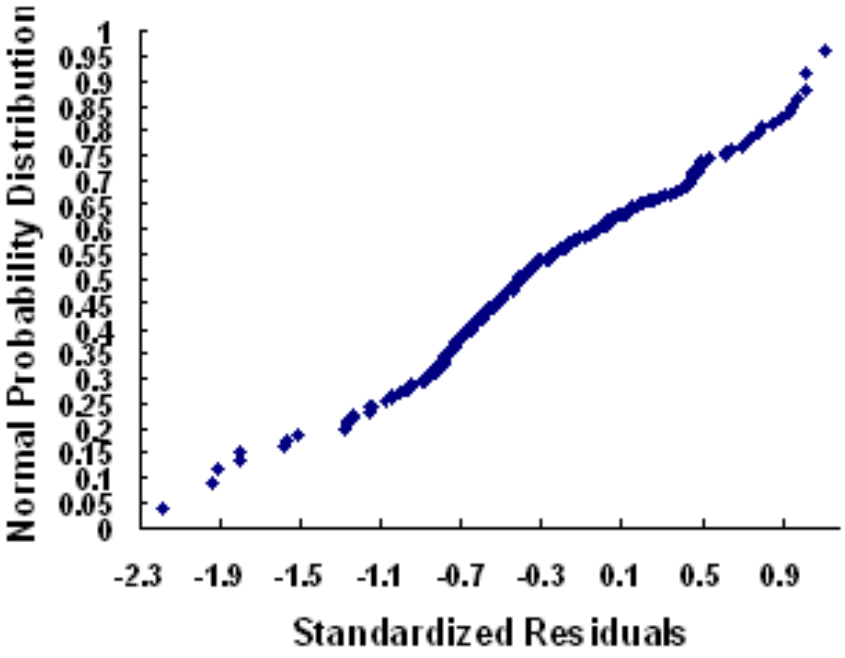


Figure 3.12 Normal Probability Plot for Standardized Residuals

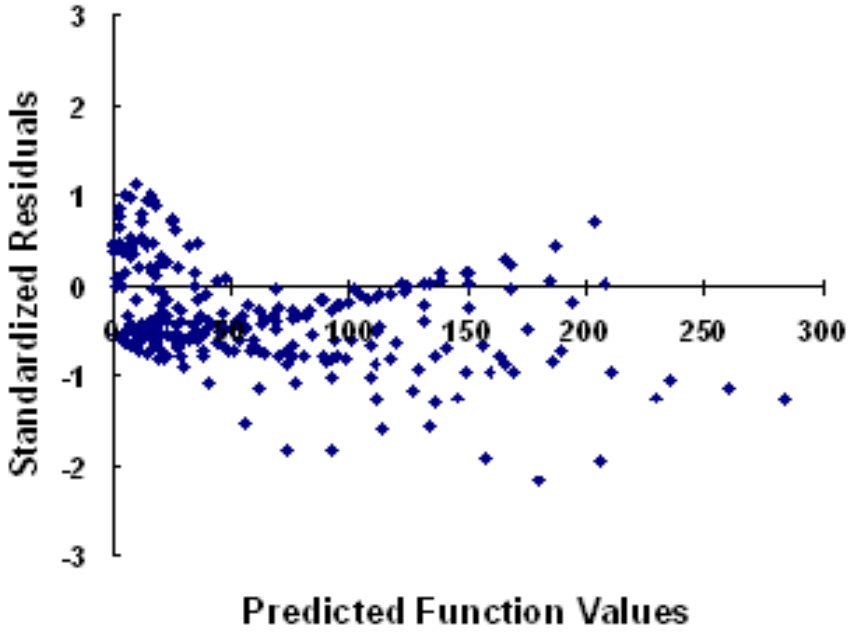


Figure 3.13 Standard Residual Plot for the Branin Function with Data Set 15

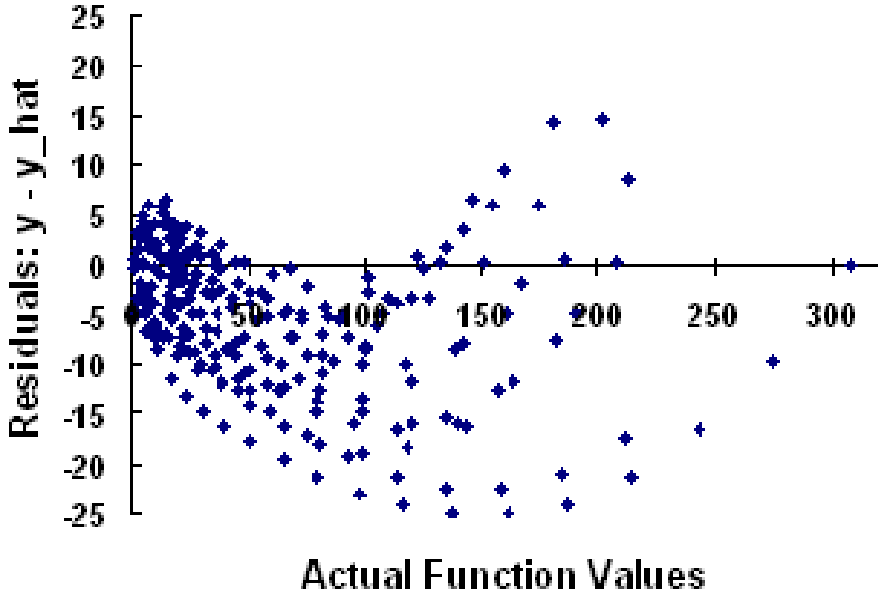


Figure 3.14 Plot of Residuals versus Actual Function Values

In Figure 3.14 we see that apparently the points follow a trend of a quadratic function. The ranges of residuals increase when actual function values increase from 0 to about 200, and decrease when actual function values increase from 200 to 300. In addition, for points with function values around 100, the actual function values are smaller than predicted values by 0 to 20. The reason is that there are almost no data points with mid-ranged function values in Data Set 5 for the kriging model (see Table 3.5). However, most of the actual residuals are within the acceptable limit of ±15 that we set at the beginning of this subsection.

Based on the previous analysis, designers have several options for the next step:

- Accept this model. This is primarily because over the whole design space the prediction errors are generally smaller than the acceptable limits (±15); the systematic bias should not be fatal in this early design stages. Also note that the model performs well when the actual function value is either very small or large, thus this kriging model will be very suitable when we want to minimize or maximize the Branin function. The prediction error at new points can be estimated with Equation 15. Designers need to notice that for function values around 100 the predicted values are usually larger than the actual values by 0 to 20, and some adjustments may be helpful when using the metamodel in design – this is very important.
- Transform the function. We can also improve the fit of the kriging model by transforming the function, e.g., using the log transformation or the inverse transformation, etc. Sometimes this works well.

- Design sequential experiments to improve the metamodel. If the designers decide not to accept the kriging model because of the systematic bias illustrated in Figure 3.13 and Figure 3.14, the information obtained in the model validation will be very helpful in identifying future data points. In this case, since the kriging model has large prediction errors at points with mid-ranged function values, we may try to add in data points which satisfy: 1) not clustered with previous data points, and 2) having mid-ranged predicted (using the previous kriging model) function values.

In this section we described our preliminary methods for engineers to validate metamodels with additional validation points. The Branin function is used to help illustrate our ideas. Applications with real-world problems will be presented later in this dissertation.

3.5 SUMMARY OF RESEARCH ON METAMODEL VALIDATION

In this chapter first we studied the performance of leave-one-out cross-validation method in validating metamodels with deterministic computer experiments. With several simple functions we illustrated that cross-validation is an insufficient method, thus to use additional validation points becomes essential in metamodel validation. Then we describe some preliminary methods on how to utilize the information from additional validation points. Our ideas are illustrated with the Branin function. Kriging metamodels are used in this paper to facilitate our study.

The reason why leave-one-out cross-validation is insufficient in metamodel validation is that it is actually a measurement for degrees of insensitivity of a metamodel to lost information at its data points, while an insensitive metamodel is not necessarily accurate. There are two causes for this insensitivity: clustering or inappropriately correlated data points. To design space-filling experiments with a sufficient number of data points is one way to prevent an inaccurate and insensitive model, while this cannot assure the validity of the leave-one-out cross-validation method. We recommend starting with space filling experimental designs in the development of metamodels in engineering applications.

The conclusion here does not mean that previous applications with leave-one-out cross-validation are necessarily wrong. When the original actual function is not highly nonlinear (or the design space is not very large) and there are enough data points spreading all over the design space, the danger of having clustering or inappropriately correlated data sets is small. However, the success of leave-one-out cross-validation in those examples is dependent on particular cases; real-world applications are usually more complicated and cannot meet the requirements mentioned above. Thus to use additional validation points are necessary in metamodel validation.

Though one important benefit of using metamodels is to save expenses on experiments, to add in additional validation points, which eventually increases time and effort on computer simulations, does not hurt the importance of metamodeling very much. First, in computer experiments, moderate increases of computational expenses are usually affordable with fast computers in a distributed design environment. Second, to

use metamodels not only helps us save experimental expenses but also integrates simulation codes from different disciplines to give insight into the relationships between input variables and output responses.

Several methods are described to help engineers gain insight into the performance of metamodels over the whole design space. Equation (3.9) provides engineers a very rough estimate of confidence levels on how well the metamodels performs on average in response prediction.

An alternative method is to check whether we are able to use Equation 15, which does not utilize information from additional validation points, to predict prediction errors at new points. If a kriging model performs well in predicting its own prediction errors, and its errors are acceptable, we may accept this metamodel. To test this various plots may be drawn to help our analyses, as illustrated with the Branin function. These analyses with plots provide much useful information for design along a timeline: 1) if we decide to accept the kriging models, knowledge we get in these analyses tell us when and where and how to make amends to our results in later design stages, and 2) if we decide to develop a more accurate model, knowledge we obtain in these analyses provides guidance on how to allocate future data points – this leads to possible methods for sequential experimental designs.

One unsolved problem in research on model validation in this chapter is how to select validation points, e.g., how many validation points should be used, and how to allocate these points. This is closely related to sequential experimental design methods, and is studied in Chapter 4 and 6. Also, a method for making decisions about metamodel

validation and selection in multi-disciplinary, multi-response applications is another possible avenue, for which some preliminary work is conducted in Chapter 5. Another approach to validate metamodels with information from possible new data points will be proposed in Chapter 4 and 5 during the process of testing Sub-Hypothesis 1.2.2.

3.6 A LOOK BACK AND A LOOK AHEAD

Studies in this chapter are the basis of ideas and methods developed in later chapters. In this chapter we visited Research Question 1, its sub-questions, and the corresponding hypotheses (except Sub-Hypothesis 1.2.2), as shown below:

R.Q.1: *How to validate a metamodel with deterministic computer experiments?*

Hypothesis 1: Information from either previous additional validation points is needed in testing the accuracy of a metamodel with deterministic computer experiments.

R.Q.1.1: *Is leave-one-out cross-validation a suitable method of metamodel validation with computer experiments?*

Sub-Hypothesis 1.1: Leave-one-out cross-validation is not an appropriate method of metamodel validation with deterministic computer experiments.

R.Q.1.2: *How to test the accuracy a metamodel in deterministic applications?*

Sub-Hypothesis 1.2.1: The accuracy of a metamodel could be validated through examining prediction errors at additional validation points

Research Question 1.1 and Sub-Hypothesis 1.1 are visited in Sections 3.2 and 3.3. In Section 3.2 with two single-variable examples we observe that leave-one-out cross-

validation is insufficient in metamodel validation because it is actually a measurement for degrees of insensitivity of a metamodel to lost information at its data points, while an insensitive metamodel is not necessarily accurate. After careful examination, we point out that there are two causes for this insensitivity: clustering or inappropriately correlated data points. To design space-filling experiments with a sufficient number of data points is one way to prevent an inaccurate and insensitive model, while this cannot assure the validity of the leave-one-out cross-validation method. Our conclusion is verified through empirical study in Section 3.3. Research Question 1.1 is answered and Sub-Hypothesis 1.1 is tested.

Research Question 1.2 and Sub-Hypothesis 1.2.1 are visited in Section 3.4, in which approaches are proposed for engineers to test the accuracy of metamodels. Several methods are described to help engineers gain insight into the performance of metamodels over the whole design space. Information from additional validation points is utilized in these approaches. Examination of prediction errors in the design space leads to ideas on sequential metamodeling (DOE) and design space exploration which will be studied in Chapter 6.

In the next chapter, the usage of additional validation points leads to a sequential experimental design method. Unsolved problems on validation points in this chapter, e.g., the problem of selection of validation points as stated in Section 3.4 and 3.5, are to be studied in Chapter 4 and 5. The focus of Chapter 4 is Research Question 2, its sub-questions, and corresponding hypotheses; however, Research Question 1.2 and Sub-Hypothesis 1.2.2 will also be visited as a side-product in Chapter 4.

CHAPTER 4

SEQUENTIAL EXPLORATORY EXPERIMENTAL DESIGN

In this chapter, the method of Sequential Exploratory Experimental Design (SEED) is developed based on D-optimal design and maximum entropy sampling. Several simple examples are used to help illustrate the SEED method. The research questions Q.2.1, 2.2, and 2.3 are answered and Sub-Hypotheses 2.1, 2.2, and 2.3 are tested. A brief overview of the organization of this chapter is presented in Section 4.1. The problem of sequential experimental design is defined in Section 4.2. In Sections 4.3 and 4.4, previous work on D-optimal design and maximum entropy sampling is introduced. The method of Sequential Exploratory Experimental Design is then developed in Section 4.5. The SEED method is then tested with a single-variable example in Section 4.6. A look back and look forward is enclosed in Section 4.7.

4.1 WHAT IS PRESENTED IN THIS CHAPTER

In Chapter 3 we studied techniques used in metamodel validation with deterministic computer experiments. One important conclusion in Chapter 3 is that it is necessary to use additional validation points in verifying the accuracy of metamodels. Thus, in designing computer experiments, the designers need to consider the identification of not only data points but also validation points.

To save time and effort in metamodeling, it is desirable to add in data points sequentially; information from previous data points could be used as a guide for selecting future data points. Given that it is necessary to have validation points in consideration, our goal in this chapter is to develop a method with which sequential experiments (of data points and validation points) could be designed. The research question to be addressed in this chapter is Research Question 2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?* To answer this research question, we plan to test the hypothesis that sequential experiments could be designed through analysis of information from previous data/validation points and metamodels. This consists research in three steps: measurement of information, utilization of information from validation points, and identification of new data points.

After introducing the definitions and nomenclatures of the problem of sequential experimental design in Section 4.2, foundations of research in this chapter, D-optimal design and maximum entropy sampling, are discussed in detail in Section 4.3 and 4.4. This literature review helps answer the first sub-research question – Research Question

2.1: *How to measure the information worth of a point?* The method of Sequential Exploratory Experimental Design (SEED) is developed in Section 4.5, which help answer Research Question 2.2: *How to select validation points to achieve a sequential design of computer experiments?* and Research Question 2.3: *How to utilize information from previous points and metamodels in identifying new data points?* Information from validation points is utilized in evaluating the information worth of a point and thus new data points with maximum potential information are selected to form sequential experiments. A framework of the SEED method is presented in Section 4.5.

The example of a single-variable function is studied in Section 4.6 to help verify and illustrate the SEED method. In Section 4.7 we revisit the research questions and hypotheses discussed in this chapter.

4.2 DESIGN OF SEQUENTIAL EXPERIMENTS: PROBLEM OVERVIEW

In this section, the problems of experimental design, metamodeling, and sequential experiments are defined. The notions and nomenclatures stated in this section will be used later in Chapter 4 and across the whole dissertation (if not otherwise defined).

Let $\boldsymbol{x} \in \Re^p$ denote the vector of input values chosen for the computer program. In this chapter, we will write $\boldsymbol{x}$ as the row vector $(x_1 \ldots, x_p)$ using subscripts to denote components of $\boldsymbol{x}$. Here p is the number of design variables. We assume that each component x_j ($j = 1,\ldots,p$) is continuously adjustable between a lower and an upper limit, which after a linear transformation can be taken to be 0 and 1 respectively. The computer

program is denoted by f and it computes q output quantities. In the studies here, we take $q = 1$, i.e., only one output is taken into consideration. Thus the deterministic computer simulation at each data point is illustrated as following:

$$y = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in [0,1]^p \tag{4.1}$$

where $\boldsymbol{x}$ denotes a data point, defined as (based on previous discussions):

$$\boldsymbol{x} = [x_1, x_2, ..., x_p] \tag{4.2}$$

where, again, p is the number of design variables. The design space as presented in Equation (4.1) is $[0,1]^p$, which means all the design variables have already been scaled to [0,1] before the experimental design.

There are many different but related goals that arise in computer experiments, including (Koehler and Owen, 1996): finding a good value for $\boldsymbol{x}$ according to some criteria on $\boldsymbol{y}$, finding a simple approximation $\hat{f}$ that is accurate enough over a region R of $\boldsymbol{x}$ values, estimating the size of the error $\hat{f}(\boldsymbol{x}_0) - f(\boldsymbol{x}_0)$ for some $\boldsymbol{x}_0 \in R$, estimating $\int_A f d\boldsymbol{x}$, sensitivity analysis of $\boldsymbol{y}$ with respect to changes in $\boldsymbol{x}$, finding which x_j are most important for each response y_k, finding which competing goals for $\boldsymbol{y}$ conflict the most, visualizing the function f and uncovering bugs in the implementation of f. In this chapter, I will focus on the problem of how to find good values of $\boldsymbol{x}$'s so that as much information as possible could be reflected and thus more accurate metamodels could be developed.

Based on previous discussions in Chapter 2 (Figure 2.5), metamodeling is actually the process of designing experiments, collecting information, finding and fitting the

appropriate approximation function $\hat{f}$ for f, and finally, validating the accuracy of the approximation. Among these steps in metamodeling, design of experiments (DOE) and metamodel building are two most important steps. In this dissertation, the word "metamodeling" is used at two levels: at higher level "metamodeling" represents the whole process in Figure 2.5, which consists four steps as mentioned above; at lower level "metamodeling" represents the steps of finding and fitting the appropriate approximation functions, which corresponds Step 2 and 3 in Figure 2.5. For the exact meaning of "metamodeling" in different cases, please pay attention to the context that "metamodeling" is used.

In DOE, the points where information is collected for developing metamodels are called data points, which is an aggregation of $\boldsymbol{x}$ as defined in Equation (4.2). Validation points are also defined with the same equation, while information collected from these points is not used for developing metamodels but for testing the accuracy of the metamodels. A possible data or validation point in the design space is called a candidate point.

The aggregation of all points (data points, validation points, and candidate points) is denoted by *U*, and in this dissertation, *U* is defined as $[0,1]^p$ as illustrated in Equation (4.1). In most common cases where the design variables are continuous, *U* contains infinite number of points; while in traditional design methods, *U* is restricted to have finite number of points that are pre-selected in the design space, which helps save time and effort in finding out the "best" set of data points. In our studies in Chapters 4, 5, and

6, the U with infinite number of points is used to ensure the appropriateness of the proposed research; while in real-world case studies where information at large number of points is to be examined, we should use U with finite number of points.

The aggregation of data points, denoted by D, is a subset of U. An experimental design with n data points is specifically defined as an n-design, denoted by D_n. D_n is a design of size n with the set of responses represented by:

$$\boldsymbol{y}_D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_i, \boldsymbol{y}_i), \ldots, (\boldsymbol{x}_n, \boldsymbol{y}_n)\} \tag{4.3}$$

where $\boldsymbol{x}$ again is a data points as defined in Equation (4.2), and $\boldsymbol{y}$ is a vector that represents q responses. In our study in this chapter, we have $q = 1$, i.e., only one response is considered.

In this dissertation, the aggregation of validation points, denoted by A, is also a subset of U. The number of validation points is denoted by n_{error}. The complement of D in U, denoted by $\overline{D}$, is the aggregation of candidate points and validation points. The relationship between data points, validation points, and candidate points is illustrated in the following figure.

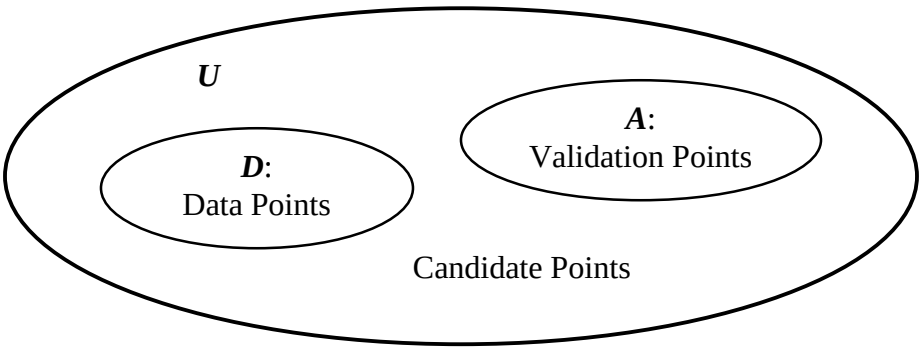


Figure 4.1 Data Points, Validation Points, and Candidate Points

The problem of Design of Experiments (DOE) is on how to identify the aggregation of *D* in *U* to best reflect response information in the whole design space. In designing deterministic computer experiments, based on research in Chapter 3, it is also necessary to identify the aggregation of *A* in *U* to test the accuracy of metamodels developed with *D*. In sequential DOE, data points are selected in iterations; information from previous data points and metamodels could be used as guidance in identifying future data points. Thus in sequential experiments, the aggregation of *D* grows gradually along the design timeline.

Various techniques are proposed to help identify the aggregation of *D* in designing computer experiments. As discussed in (Koehler and Owen, 1996), there are two main statistical approaches to computer experiments, one based on Bayesian statistics and a frequentist one based on sampling techniques. The frequentist approach to prediction and inference in computer experiments is based on numerical integration. Without anything known about the distribution of the output data in the region of interest, intuitively there is no general guideline for selecting good sample points. However, statistically there are some sampling techniques that are in general better than the others. Such sampling techniques include *Grids*, *Good Lattice Points*, *Latin Hypercube Sampling*, *Randomized Orthogonal Arrays*, and *Scrambled Nets*, etc.

In the Bayesian framework, one sets a prior distribution on the space of all functions from inputs (design variables) to outputs (responses). Given the values of inputs and outputs, a posterior distribution is generated. The prior distribution is usually taken to be Gaussian so that any finite list of function values has a multivariate normal

distribution (Koehler and Owen, 1996). Given observed function values, the posterior distribution is also multivariate normal. The posterior mean interpolates the observed values and the posterior variance may be used to give 95% posterior probability intervals (Koehler and Owen, 1996). The method extends naturally to incorporate measurement and prediction of derivatives, partial derivatives and definite integrals of the function. The Bayesian framework is well developed but as is common with Bayesian methods there may be difficulty in finding an appropriate prior distribution. Such Bayesian experimental design techniques include *Maximum Entropy Sampling*, *Mean Squared-Error Designs*, *Maximin*, and *Minimax Designs*, etc. Mathematics behind such Bayesian techniques as *Maximum Entropy Sampling*, etc. is the same as that behind the kriging metamodel which was introduced in Chapter 2.

There are also other ways to classify experimental design methods. As discussed in Chapter 2, classical DOE's, e.g., factorial designs, CCD, etc., are widely used in designing physical experiments. Classical experiments as used in the Response Surface Methodology (RSM) are conducted in a sequential manner; fractional factorial experiments are used first to help screen out unimportant design variables, then central composite designs are constructed for developing quadratic regression metamodels. However, in a fixed design space, i.e., when no screening is allowed, DOE in RSM becomes a single-stage experimental design. Space-filling experiments are proved to be suitable for designing deterministic computer experiments, while these designs are usually in a single-stage manner in that information from previous data points and metamodels have no influence on the selection of future data points. The selection and

usage of validation points are not discussed yet in designing sequential computer experiments. In this chapter, a method of Sequential Exploratory Experimental Design is developed in which information from previous data/validation points and metamodels are used as guidance in identifying new data/validation points. The aggregations of *D* and *A* grow gradually along the design timeline; they are selected to ensure that information at $\overline{D}$ could be predicted with maximum confidence based on information from *D* and thus accurate metamodels could be developed with the data points. Research in this chapter is based on two DOE techniques, *D*-optimal designs and Bayesian Entropy Sampling, as will be discussed in Sections 4.3 and 4.4.

4.3 CONSTRUCTION OF D-OPTIMAL DESIGNS

In DOE, an optimal design is one that has some optimum properties. A systematic study of the specification of optimum experimental designs was undertaken in (Kiefer 1958; Kiefer, 1959), where he introduced various optimality criteria (A, D, E, L, M), and discussed interrelations amongst these and established the optimality property of some well-known designs in some particular problems.

D-optimal designs are straight optimizations based on a chosen optimality criterion and the model that will be fit. The optimality criterion used in generating *D*-optimal designs is one of maximizing $|\boldsymbol{X}'\boldsymbol{X}|$ (or $\det(\boldsymbol{X}'\boldsymbol{X})$), the determinant of the information matrix $\boldsymbol{X}'\boldsymbol{X}$. This criterion of maximizing the determinant of $\boldsymbol{X}'\boldsymbol{X}$ is proposed as a means of maximizing the local power of the F-ratio for testing a linear hypothesis on the parameters of certain fixed-effects analysis of variance models (Wald,

1943). The matrix $\boldsymbol{X}'\boldsymbol{X}$ is called the information matrix. It is proportional to the inverse of the covariance matrix of the parameters. So maximizing *det*($\boldsymbol{X}'\boldsymbol{X}$) is equivalent to minimizing the determinant of the covariance of the parameters. In the case of *D*-optimality for regression designs, $\boldsymbol{X}$ is the expanded design matrix that has *n* rows (one for each design setting) and *p* columns (one column for each coefficient to be estimated plus one column for the overall mean). It was proved that a *D*-optimal design is also minimax, and on ther other hand, a minimax design is *D*-optimal (Kiefer and Wolfowitz, 1960).

To construct the information matrix is an important step in building *D*-optimal designs. As discussed in (Zacks, 1996), let $A.F. = \{F(\cdot \mid \boldsymbol{x}, \boldsymbol{\theta}); \boldsymbol{\theta} \in \Theta, \boldsymbol{x} \in \chi\}$ be a regular family of distribution functions of random variables $\boldsymbol{y}_x$, where $\boldsymbol{x}$ are design variables in the design space χ, χ is a subset in $\Re^p$, $\boldsymbol{\theta}$ are unknown parameters of the distribution in the parameter space Θ, and Θ is an open set in $\Re^k$. The regularity of *A.F.* means that all its elements satisfy the well known Cramer-Rao regularity conditions (Wijsman, 1973). Let $f(y; x, \theta)$ be a p.d.f. of $F(\cdot \mid \boldsymbol{x}, \boldsymbol{\theta})$ with respect to some σ-finite measure μ. The information matrix is formulated as:

$$I(\boldsymbol{\theta}; \boldsymbol{x}) = Var_{\theta, x}\left\{\frac{\partial}{\partial \boldsymbol{\theta}} \log f(\boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta})\right\} \tag{4.4}$$

where, in the *k*-parameter case, $I(\boldsymbol{\theta}; \boldsymbol{x})$ denotes a $k \times k$ covariance matrix, and $\frac{\partial}{\partial \boldsymbol{\theta}} \log f(\boldsymbol{y}; \boldsymbol{x}, \boldsymbol{\theta})$ is a gradient vector (score vector). A design is called optimal with respect to the information, if it maximizes some functional of the information matrix in

Equation (4.4). In the case of *D*-optimal design, the determinant of the information matrix $I(\boldsymbol{\theta}; \boldsymbol{x})$ is maximized.

Since this *D*-optimality criterion results in minimizing the generalized variance of the parameter estimates for a pre-specified model, as a result, the "optimality" of a given *D*-optimal design is model dependent. That is, the experimenter must specify a model for the design before a computer can generate the specific treatment combinations.

Generally speaking, for linear models the optimum designs (including *D*-optimal designs, of course) do not depend on the values of the parameters of the metamodel. However, for nonlinear metamodels, the optimum experimental designs depend heavily on the values of the unknown parameters. One way to accommodate the dependence of optimum design on the chosen parameter values is to introduce a prior distribution on the parameters and to incorporate this distribution into appropriate design criteria. Bayes formula is a useful equation from probability theory that expresses the conditional probability of an event A occurring, given that the event *B* has occurred (written P(A|B)), in terms of unconditional probabilities and the probability the event *B* has occurred, given that *A* has occurred. In other words, Bayes formula inverts which of the events is the conditioning event. For more details, see (Bernardo and Smith, 1994; Congdon, 2001, etc.)

Bayesian *D*-optimality design is thus developed in this regard (see, Chaloner and Verdinelli, 1995; Bernardo and Smith 1994; Pilz, 1991; Pukelsheim, 1993). In Bayesian *D*-optimal design, the expectation of the logarithm of the determinant of the information matrix as represented below is maximized (Atkinson and Haines, 1996):

$$\mathrm{E}_{\theta} \log\det I(\theta,\xi) = \int \log\det I(\theta,\xi)\, p(\theta)\, d\theta \qquad (4.5)$$

where $p(\theta)$ is the prior distribution on θ, ξ is a probability measure for an n-trial design over the design space χ as introduced in the approximate or continuous design theory in (Kiefer, 1985). In Bayesian optimal design, to maximize Equation (4.5) is also cited as maximizing the expected utility for the particular experiment. A formal justification for the criterion as stated in Equation (4.5) within the Bayesian paradigm is provided in (Chaloner and Verdinelli, 1995).

Based on the well-done previous work on optimal design (especially *D*-optimal design), our aim in this chapter is to develop some method to design sequential experiments. In designing optimal experiments, given the total number of treatment runs for an experiment and a specified model, the computer algorithm chooses the optimal set of design runs from a *candidate set* of possible design treatment runs. This candidate set of treatment runs usually consists of all possible combinations of various factor levels that one wishes to use in the experiment. A good review on how to develop *D*-optimal designs for regression metamodels is done in (John and Draper, 1975). Many algorithms and systems are developed for designing *D*-optimal experiments (e.g., see Clyde, 1994; Dumouchel and Jones, 1994; Steinberg, 1985, etc.). Sequential designs of *D*-optimal experiments are also studied. For example, in (Berry and Fristedt, 1985) the authors studied sequential experiments with bandit problems; Freeman (1970) solved the Bayesian sequential design problem exactly for a very small and simple binary regression experiment.

In this chapter, we will develop the method of Sequential Exploratory Experimental Design (SEED) through utilizing information at previous validation points and metamodels, which is not seen in literature. To reflect this information, the key issue is how to formulate the information matrix as appeared in Equations (4.4) and (4.5). An intuitive method is proposed in this chapter based on maximum entropy sampling that will be introduced in the next section. As stated in many literatures (e.g., see Sebastiani and Wynn, 2000; Chaloner and Verdinelli, 1995; Sebastiani and Wynn, 2001; Sebastiani and Wynn, 1997; Pukelsheim, 1993; Bernardo, 1979, etc.) and will be described in the next section, Shannon information has been widely used in the statistical literature of Bayesian design as formulation for the utility function in Equation (4.5).

4.4 BAYESIAN ENTROPY DESIGN

As introduced in Chapter 2, the word entropy first originated in the literature on thermodynamics to represent a measure of the amount of energy in a thermodynamic system as a function of the temperature of the system and heat that enters the system. It was first used as a measure of information in 1948 when Claude Shannon developed his theory of communication (Shannon, 1948). The relationship between Shannon's entropy and thermodynamic entropy was established in (Kapur and Kesavan, 1992). The concept of entropy is closely tied to the concept of uncertainty embedded in a probability distribution. In fact, entropy can be defined as a measure of probabilistic uncertainty (the uncertainty associated with the probability of outcomes). Let $\mathbf{p} \equiv (p_1, p_2, \ldots, p_n)^T$ be a

probability distribution associated with *n* possible outcomes, Shannon's entropy is defined as (Shannon and Weaver, 1962):

$$S_n(p) = -\sum_{j=1}^{n} p_j \ln p_j \tag{4.6}$$

where $\sum_{j=1}^{n} p_j = 1$, $0ln0 = 0$, $p_j \geq 0$ for $j = 1,\ldots,n$. Another formulation of Shannon's entropy, used as a measure of the uncertainty of the transmission of information, is:

$$H = -\int_{\Omega} p(s) \ln p(s) ds\,, \tag{4.7}$$

where *p(s)* is a Gaussian density function over the space Ω of the information signals transmitted. The word "entropy" has different meanings in different contexts, depending on how we define the p_i or *p(s)* in its formulation.

To use information theoretic ideas in experimental design has a considerable history, with definite papers by Lindley (1956), Stone (1959), and Renyi (1961). An elegant summary appears in (Renyi, 1970). Many of the ideas have been absorbed into the flourishing area of Bayesian optimal design as talked about in Section 4.3. The Bayesian information theoretic approach, which states that the optimal design maximizes the expected information worth of the experiments, has been well studied in literature in the past 20 years (see, e.g., Chaloner and Verdinelli, 1995; Pilz, 1991; Bernardo and Smith, 1994, etc.). As introduced in the optimal designs, Bayesian design requires a specification of a utility function, and Shannon's information theoretic formulation of entropy has been widely used in literature. It is stated that the Bayesian entropy design,

which maximizes the entropy of the prior design, is able to simplify the formulation of the design criterion while keeping the computational complexity manageable compared to most other techniques.

Shannon's entropy was first introduced in the field of design of experiments in (Lindley, 1956), in which the author interpreted entropy as the amount of information gained by a data point. With the aim of maximizing the gain in information for prediction at new data points, maximum entropy sampling (MES) was used as a criterion for the choice of experiments in (Shewry and Wynn, 1987). This criterion was then adopted as one of the main methods for computer experiments in (Sacks, et al., 1989a). Based on these studies, a maximum entropy design strategy is proposed in (Currin, et al., 1991) in which new data points are added sequentially in the design space such that maximum expected information is gained from the set of experiments. A good paper on computer experiments and maximum entropy sampling is (Koehler and Own, 1996). These papers as mentioned above represent previous work in Bayesian entropy design, which is the basis of our SEED method of designing sequential experiments with consideration of prediction errors in previous metamodeling processes. In the following paragraphs, I will describe how Bayesian entropy design works; for more details, please refer to the papers mentioned above.

4.4.1 Prior and Posterior Distributions

In Bayesian methods, we need to specify a prior knowledge about a function. In our research, prior uncertainty about the function y is expressed by means of a random

function Y, which is taken to be a Gaussian stochastic process. The mean of the posterior process is used as the prediction function $\hat{y}(x)$, and the variance can be used as a measure of uncertainty. This kind of approach is strongly related to the kriging methods as introduced in Chapter 2. Thus in this chapter, we will use kriging metamodels in developing the Sequential Exploratory Experimental Design (SEED) method.

As mentioned above, the prior knowledge about the unknown function $y(\boldsymbol{x})$ is set to be Gaussian process Y. Given an n-design D_n (as defined in Section 4.2), the prior distribution of the design is multivariate normal with mean vector and the positive definite covariance matrix as:

$$E[Y_D] = \boldsymbol{\mu}_D = [\mu_i] \tag{4.8}$$

$$\text{cov}(Y_D, Y_D) = \boldsymbol{\sigma}_{DD} = \left[\sigma_{ij}\right]_{n\times n} \tag{4.9}$$

where i and j corresponds two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j \in D$. Elements in the vector of $[\mu_i]$ is defined as the expected mean of the normal distribution Y_i at a point $\boldsymbol{x}_i \in D$:

$$E(Y_i) = \mu_i \tag{4.10}$$

And the entries of the covariance matrix is defined as:

$$\sigma_{ii} = \text{cov}(Y_i, Y_i) = Var(Y_i) \qquad \sigma_{ij} = \text{cov}(Y_i, Y_j) \tag{4.11}$$

In Equation (4.11) we see that two prior distributions at $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are not statistically independent; we will talk about this correlation later.

As defined in Section 4.2, D is a subset of all the possible points in the design space, denoted as $D \subset U$. Based on the discussion above, the posterior process, given the

vector of observed response y_D on D_n, is well known and is also Gaussian. Suppose we want to examine responses at a finite number of new points $S \subset U$, the mean and variance at S is given by:

$$\boldsymbol{\mu}_{S|D} = E\left[\boldsymbol{Y}_S \mid \boldsymbol{y}_D\right] = \boldsymbol{\mu}_S + \boldsymbol{\sigma}_{SD}\boldsymbol{\sigma}_{DD}^{-1}(\boldsymbol{y}_D - \boldsymbol{\mu}_D) \tag{4.12}$$

$$\boldsymbol{\sigma}_{SS|D} = \operatorname{cov}\left[\boldsymbol{Y}_S, \boldsymbol{Y}_S \mid \boldsymbol{y}_D\right] = \boldsymbol{\sigma}_{SS} - \boldsymbol{\sigma}_{SD}\boldsymbol{\sigma}_{DD}^{-1}\boldsymbol{\sigma}_{DS} = \left[\sigma_{ij|D}\right] \tag{4.13}$$

where $\boldsymbol{\sigma}_{SD} = \boldsymbol{\sigma}'_{DS} = \operatorname{cov}(\boldsymbol{Y}_S, \boldsymbol{Y}_D)$. In Equation (4.13), i and j corresponds two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j \in \bar{D}$; please note the difference between this definition and that for Equation (4.9). With Equation (4.13) we are able to estimate the posterior covariance matrix based on prior distributions. From a Bayesian viewpoint, the posterior process is very important; a Bayesian estimate for y at new observation sites (new points) is the mean of the posterior distribution:

$$\hat{y}(\boldsymbol{x}) = \mu_{x|D} = \mu_{\boldsymbol{x}} + \sigma_{xD}\sigma_{DD}^{-1}(y_D - \mu_D) \tag{4.14}$$

where $\boldsymbol{x}$ is a new point to observe in the design space U. Given an experimental design D, the response value at a new point $\boldsymbol{x}$ could be estimated with Equation (4.14). This is actually the essence of Gaussian interpolation and kriging metamodels; see Equation (2.18) in Chapter 2 for analogy. This analogy is explained below with the stationary assumption.

4.4.2 The Stationary Assumption

As explained in (Currin, et al., 1991), the stationary assumption is introduced to help develop a general method without eliciting and implementing problem-specific prior

information. With this goal some conditions of stationarity are needed to produce prior processes that are non-informative, or at least impartial in some respects. In particular, the prior mean and variance is required to be constant for all $\boldsymbol{x}$ in U: $\mu_{\boldsymbol{x}} = \mu, \sigma_{\boldsymbol{xx}} = \sigma^2$, and at any two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in U, the prior correlation ρ_{ij} between Y_i and Y_j depends only on their Euclidian distance $d = \left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|$ through a suitable correlation function R. The correlation function R must satisfy that $\rho_{ij} = R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) = R(d)$, and $R(0) = 1$; for any finite set of points S, the correlation matrix ρ_{SS} generated by R must be positive definite.

With the stationary assumption mentioned above, the covariance matrix σ_{DD} is invariant to any isometric transformation in the points in U. The prior distribution for Y_s at points S does not change if S is shifted – the correlation is only related to the relative distance between points but not the absolute location of the points. Thus Equations (4.12) and (4.13) become:

$$\boldsymbol{\mu}_{S|D} = \boldsymbol{\mu}\mathbf{f} + \boldsymbol{\rho}_{SD}\boldsymbol{\rho}_{DD}^{-1}(\boldsymbol{y}_D - \boldsymbol{\mu}\mathbf{f}) \tag{4.15}$$

$$\boldsymbol{\sigma}_{SS|D} = \sigma^2\left[\boldsymbol{\rho}_{SS} - \boldsymbol{\rho}_{SD}\boldsymbol{\rho}_{DD}^{-1}\boldsymbol{\rho}_{DS}\right] \tag{4.16}$$

where $\mathbf{f}$ is a vector of 1's. For prediction at a single site $\boldsymbol{x}$, we have

$$\hat{y}(\boldsymbol{x}) = \mu_{\boldsymbol{x}|D} = \mu + \boldsymbol{\rho}_{\boldsymbol{x}D}\boldsymbol{\rho}_{DD}^{-1}(\boldsymbol{y}_D - \mu\mathbf{f}) \tag{4.17}$$

$$\boldsymbol{\sigma}_{\boldsymbol{xx}|D} = \sigma^2\left[1 - \boldsymbol{\rho}_{\boldsymbol{x}D}\boldsymbol{\rho}_{DD}^{-1}\boldsymbol{\rho}_{D\boldsymbol{x}}\right] \tag{4.18}$$

Equation (4.17) is the same as Equation (2.18), calculation of predicted estimates in kriging metamodeling. The covariance of prior distributions at two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is:

$$\operatorname{cov}\left(Y_i, Y_j\right) = \sigma_{ij} = \sigma^2 R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \qquad \boldsymbol{x}_i, \boldsymbol{x}_j \in D \qquad i \neq j \tag{4.19}$$

$$\operatorname{var}\left(Y_i\right) = \sigma_{ii} = \sigma^2 \tag{4.20}$$

Equations (4.19) and (4.20) are used to formulate entries of the covariance matrix as used in maximum entropy sampling (Section 4.4.3).

There are several choices for the correlation function R. In (Simpson, 1998; Lin, 2000), we have studied five types of correlation functions, namely, the exponential function, the Gaussian function, cubic spline, Matérn linear function, and Matérn cubic function. However, in this dissertation, we will use the Gaussian correlation function (Equation (2.14)), which is by far the most popular one in use.

4.4.3 The Entropy Criterion

Assuming that we have an n-design D_n in the design space U. Once we got information at the data points, knowledge of the function y at other points will be embodied in the multivariate normal distribution of $Y_{S|D}$ generated by the predictive process. The mean $\mu_{S|D}$ and the covariance matrix $\sigma_{SS|D}$ of this distribution could be calculated with Equations (4.15) and (4.16). The problem of experimental design here is actually to choose D to minimize the "amount of uncertainty" in $Y_{S|D}$. Shannon's entropy could be used to help achieve this goal. For a continuous multidimensional random variable $\boldsymbol{X}$, Shannon's entropy formulation is:

$$H(X) = E\left[-\ln p_x(X)\right] - \ln dx \tag{4.21}$$

where $p_x(x)$ is the density of $\boldsymbol{X}$ at x and dx is the volume element in an arbitrarily fine discretization of the design space. The formulation of entropy in Equation (4.21) is always nonnegative; the lower the entropy, the more precise is the knowledge represented by $\boldsymbol{X}$. In practice, we ignore the second term of ($-\ln dx$) since it does not depend on the distribution of $\boldsymbol{X}$.

In (Lindley, 1956), the author proposed using the expected reduction in entropy as a criterion for design. Experiments that minimize the entropy of the posterior distributions $H(Y_{S|D})$ should be chosen as the design. This idea is further developed in (Shewry and Wynn, 1987) in which the authors showed that the posterior entropy could be minimized by choosing D that maximizes the prior entropy, $H(Y_D)$. For Gaussian priors, the design dependent part of $H(Y_D)$ is $\frac{1}{2}\ln\det(\sigma_{DD})$. Thus, to maximize the Gaussian prior entropy is equivalent to maximize the determinant of the covariance matrix. Given the stationary assumption, this is the same as maximizing the determinant of the correlation matrix. Thus, here is the maximum entropy DOE strategy:

> *In order to achieve maximum entropy sampling, the designers should choose data points D_n with maximum determinant of the prior covariance matrix* $\det(\sigma_{DD})$.

This could be viewed as *D*-optimality because it minimizes the posterior generalized variance of the unknowns, as the usual *D*-optimality criterion in the linear model does. In (Johnson, et al., 1990), the authors show that when the prior correlation between points is extremely weak and is a decreasing function of an appropriately defined distance, the entropy criterion maximizes the minimum distance among design points. The tendency of *D*-optimality to maximize distances between points is evident in augmenting existing designs.

As defined in Section 4.2, the design space is denoted by U, which contains infinite number of possible data points in $[0,1]^p$. In literature, in order to save time and effort in building experiments, a grid is constructed in the design space, with each node represents a possible input vector. The grid should not be very large to ensure achievement of "best" experimental designs, and it should not be too small so that remarkable computation time and effort could be saved. In this way the design space is reduced from one with infinite number of points (continuous) to one with limited number of points (discrete). Then the problem of experimental design becomes that how to select a certain number of points in the pool of all possible points to convey maximum information of the response surface. Currently, this usage of grid to save computation time is used in nearly all Bayesian entropy designs in literature. In our research in this chapter, in order to prove the feasibility and effectiveness of our SEED method, we will use a continuous design space as defined in Section 4.2, instead of the discrete design space that is used in the Bayesian entropy sampling introduced in this section.

Currin et al. (1991) took advantage of the Shewry and Wynn's result (Shewry and Wynn, 1987) for a one-point augmentation to an existing *n*-design: Should one desire to augment one more experiment to an existing set of experiments, the new experiment must be conducted at a point $\boldsymbol{x}_i \in \overline{D}_n$, with the largest variance of the posterior distribution. In other words, the best $\boldsymbol{x}_i$ to conduct a new experiment is the one at which $\sigma_{ii|D_n}$ is maximum. In the algorithm suggested by (Currin, et al., 1991), experiments are augmented one-by-one to the current set. A multiple-search is conducted over *U* to identify $\boldsymbol{x}_i \in \overline{D}_n$ that maximizes $\sigma_{ii|D_n}$. A "hikers" method for optimization is proposed to help save computation time and effort while global optimum is not guaranteed. In our research in this chapter, we try several global optimization algorithms to find the set of points with maximum determinants of the covariance matrix; computation time for the optimization is not considered here since our focus is on verifying our ideas and methods for sequential experimental design.

4.5 THE SEQUENTIAL EXPLORATORY EXPERIMENTAL DESIGN METHOD

As introduced in Sections 4.3 and 4.4, in Bayesian entropy design, information uncertainty is reflected with the Bayesian method and the most informative experiments are designed with maximizing the entropy of prior distributions. This helps answer our Research Question 2.1: *How to measure the information worth of a point?* Based on the literature review, our answer is that: given the prior distributions of a current set of data points, the new point which helps maximizes the determinant of the covariance matrix for

prior distributions is most informative (or maximizes the prior entropy). Points that help achieve larger determinants are more informative than those with smaller determinants.

Assuming we already have an *n*-design D_n, and our aim is to find out and add in a new point that yields maximum information potential. Assuming Gaussian priors, suppose we have two candidate points, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, with $\sigma_{\boldsymbol{x}_i\boldsymbol{x}_k} \leq \sigma_{\boldsymbol{x}_j\boldsymbol{x}_k}$ for all $\boldsymbol{x}_k \in D$. The question is: which point, $\boldsymbol{x}_i$ or $\boldsymbol{x}_j$, is more informative? From Equations (4.13) and (4.18), we got that:

$$\sigma_{\boldsymbol{x}_i\boldsymbol{x}_i|D} = \sigma^2 - \sigma_{\boldsymbol{x}_iD}\sigma_{DD}^{-1}\sigma_{D\boldsymbol{x}_i} \tag{4.22}$$

$$\sigma_{\boldsymbol{x}_j\boldsymbol{x}_j|D} = \sigma^2 - \sigma_{\boldsymbol{x}_jD}\sigma_{DD}^{-1}\sigma_{D\boldsymbol{x}_j} \tag{4.23}$$

Since we have $0 < \sigma_{\boldsymbol{x}_i\boldsymbol{x}_k} \leq \sigma_{\boldsymbol{x}_j\boldsymbol{x}_k}$ for all $\boldsymbol{x}_k \in D$, from Equations (4.22) and (4.23) we could deduct that $\sigma_{\boldsymbol{x}_iD} > \sigma_{\boldsymbol{x}_jD}$. Since the point $\boldsymbol{x}_i$ helps achieve a larger prior variance (if $\boldsymbol{x}_i$ is added to *D*), our conclusion is that the point $\boldsymbol{x}_i$ is with more information potential. Thus, our answer here to Research Question 2.1 is that: Given two candidate points, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and a current set of data points, *D*, assuming Gaussian priors, the point $\boldsymbol{x}_i$ is more informative than $\boldsymbol{x}_j$, if $\sigma_{\boldsymbol{x}_i\boldsymbol{x}_k} \leq \sigma_{\boldsymbol{x}_j\boldsymbol{x}_k}$ for all $\boldsymbol{x}_k \in D$. This will be revisited and further explained after the development of the SEED method.

Given that we have answered Research Question 2.1 and propose to use Bayesian entropy method to measure the information potential of candidate points in the design

space, our next step is to develop a sequential experimental design method to answer Research Questions 2.2 and 2.3, as listed below:

> ***R.Q.2.2:*** *How to select validation points to achieve a sequential design of computer experiments?*
>
> ***R.Q.2.3:*** *How to utilize information from previous points and metamodels in identifying new data points?*

As mentioned in Section 4.4, Currin, et al., (1991) suggest an algorithm to successively augment new data points to an existing experimental design. In our viewpoint, Currin's method is actually not a "sequential" experimental design method since information from previous data points and metamodels is not used as a major guidance in identifying new data points. In maximum entropy sampling, the designers tend to add in new points that are as far away from current points as possible; information of the response values takes no place in the decision making process. In this sense, the method proposed by Currin, et al., (1991) is not flexible since it does not affiliate to specific simulations (or say, problems). We say that it is not a sequential method – a sequential experimental design method is capable of placing new data points at positions that are believed to yield maximum information potentials based on analysis of information from observations at previous data points and metamodels. For example, intuitively, given a simulation (or function), more data points should be located in regions that are highly nonlinear, and fewer data points should be located in flat regions. A single-stage maximum entropy sampling method, as Currin's, cannot achieve this goal

A new sequential approach for DOE, named Sequential Exploratory Experimental Design (SEED) is introduced in this section to address the above-mentioned shortcomings. Information at previous data/validation points and metamodels is updated sequentially during the process, and it is utilized in identifying new data points in the design space. This is the core of Chapter 4.

4.5.1 Overview of the Sequential Exploratory Experimental Design Method

At the beginning of metamodeling, the designers have no information about the response surface in the design space. The simulation code is totally a "black box" that designers have no idea what outputs it will generate with specific inputs. In this case, the maximum entropy sampling method introduced in (Currin, et al., 1991) could be used to design starting experiments. This is a non-informative method since the stationary assumption is used and no information of response values is involved in allocating the points. All candidate points in the design space U have the same distribution a priori.

The starting experiments could also be designed in other ways. Designers could start with other types of experiments. Frequentist experiments are usually preferred at this stage so that designers could avoid having to specify a distribution for f (Koehler and Owen, 1996). Sometimes designers may already have some knowledge of the response surface, i.e., observations of responses at some points in the design space have already been done; in this case, these points could be used as the starting experiments though they may not be most informative (from the entropy viewpoint) or space-filling (from the frequentist viewpoint).

After running the first-round experiments, the first-round metamodel could be developed. The next step is to identify validation points. This step is necessary because 1). We need to study the prediction accuracy of the metamodel to decide whether further experimentation and metamodeling is necessary, 2). Our study in Chapter 3 suggests that additional validation points are necessary in testing metamodels, and 3). in a sequential experimental design it is very possible that we convert these validation points to data points in the future. In selecting validation points, two issues are essential: the number of validation points and the location of validation points. In this study, we do not consider too much on the number of validation points (except for the "possible last" round) because in sequential experiments, we do not really "validate" metamodels in the mid-run – what we seek with the validation points is the information of prediction errors they provide. With an existing set of data points and a corresponding metamodel, there are several ways that help identify validation points, e.g., Maximum-Scaled-Distance-Approach, Cross-Validation-Approach (Jin, et al., 2002). In this chapter, as will be discussed later, we will select validation points that are "most informative", similar to the selection of new data points.

After observation at validation points, we get information of the prediction errors of the current metamodel at validation points. If the errors are relatively small and suggest ending of experimentation and metamodeling (refer to Lin, et al., 2002), we may need to collect information at more validation points (to have enough validation points is essential for statistical validation; also, refer to Lin, et al., 2002). If prediction errors are large, next round experiments are to be designed with the information at hand. In

iterations, new data points are to be identified, new metamodels built, and new validation points observed for new validations. Ideas on how to utilize the information in identifying new data points are discussed in Section 4.5.2, while the realization of these ideas with mathematical formulations are presented in Section 4.5.3.

4.5.2 Identification of New Data Points through Utilization of Information at Previous Data/Validation Points and Metamodels

There are various ideas on how to select future data points. The first one is to select future data points that "best spread over" the design space with current data points. In this method no information from current data points and metamodels is considered; the new experimental design is still a maximum entropy sampling, or a space-filling design that have all points spread over the whole design space as evenly as possible. Of course this idea is not suitable for sequential experimental design in which we seek to maximize information with limited resources. This is explained in the following paragraph.

In maximum entropy sampling (Currin, et al., 1991), the key issue is the correlation function used to calculate the correlation between points and build the covariance matrix. Through maximizing entropy, new points are added as far away as possible from current data points. This results from the properties of the correlation function under the stationary assumption described in Section 4.4.2. As introduced in Section 4.4, the correlation function must satisfy that $\rho_{ij} = R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) = R(d)$, and $R(0)$ = 1; for any finite set of points S, the correlation matrix ρ_{SS} generated by R must be positive definite. Under the stationary assumption as described in Section 4.4.2, the

correlation between points is dependent only on their distance (relative positions), but not on their absolute locations in the design space. It is defined to be a decrease function of the Euclidian distance between points: as the distance between two points increases, their correlation decreases. This correlation is used in kriging metamodels to predict response values at unobserved points. To understand this, we could assume that each point, $\boldsymbol{x}_i$, in the design space conveys information about the response values in its vicinity; for prediction of the response value at its very location this point reflects 100% information, while for points nearby ($\boldsymbol{x}_j$), it transmits only a certain amount of information. The farther $\boldsymbol{x}_j$ is from $\boldsymbol{x}_i$, the less information that $\boldsymbol{x}_i$ transmits at $\boldsymbol{x}_j$ because of the decreasing correlation function. Thus, in the one-stage maximum entropy sampling (Currin, et al., 1991), this correlation is dependent on the value of θ and the Euclidian distance only; the location of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is not considered. This maximum entropy sampling is actually a space-filling design in which points are selected to "spread over" the design space.

In sequential experiments, information from previous observations should be used as a guide in identifying new data points. Thus, the stationary assumption of equal variance in (Currin, et al., 1991) should be modified to reflect the property of different locations in the design space. After designing the starting experiments and developing the original metamodel, new data points should be added not to spread over the design space (as maximum entropy sampling does); instead, they should be located at "crucial" locations where more potential information about the response surface could be reflected. How to identify "crucial" locations is the problem to be studied in following paragraphs.

Given information from previous experiments and metamodels, there are generally two philosophies on how to use this information for a sequential experimental design, one is to identify and add more points in the most-likely-to-succeed regions, the other is to add more points to regions with large model uncertainty. At early design stages where uncertainties on design requirements may not be controlled, it is reasonable not to reduce the design space, and thus in this chapter, we seek methods based on the second philosophy, i.e., to add in new points to help reduce overall uncertainty of the metamodel. Studies on methods based on the first philosophy, or the combination of the first and second, is done in Chapter 6.

There are also two ideas on how to identify "crucial" locations in the design space where more potential information is to be reflected. One is that the "crucial" region should be one with irregular (or highly nonlinear) response surfaces. With this belief designers should locate more future data points in regions with great response changes (or large numbers of peaks/bottoms). This is intuitive; in the interpretation of Equation (4.11) we could imagine that points in "flat" regions should have more information influence on neighborhoods than ones in "steep" regions. This idea is illustrated in (Farhang-Mehr and Azarm, 2002), as described below.

As illustrated in Figure 4.2, there are two candidate points, *A* and *B*, in the design space [0,1]. In Bayesian entropy sampling, both of these two points have influence in their neighborhoods; this influence is reflected by the correlation between them and nearby points (*A'* and *B'*). Intuitively, we see that the influence of Point *A* on Point *A'* is weaker than that of Point *B* on Point *B'*, i.e., the correlation between *A* and *A'* should be

smaller than that between *B* and *B'*. This is because Point *A* is located in a highly nonlinear region while Point *B* is located in a flat region. It is intuitive and reasonable to locate more data points in the multi-modal region around Point *A* to enable a more accurate modeling of the response function. In contrast, not that many data points are needed in the less irregular region around Point *B*. In this sense, the region around Point *A* is a "crucial" region where more potential information could be reflected, while the region around Point *B* is not.

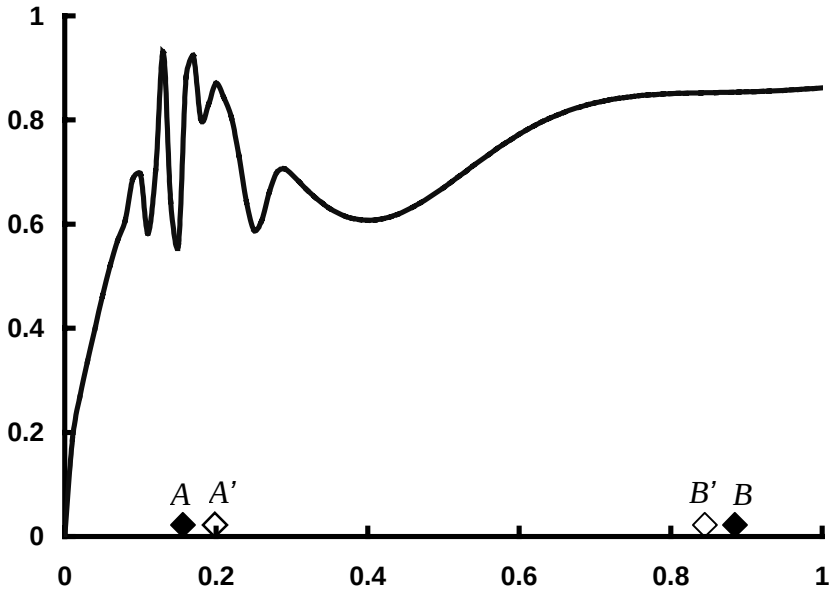


Figure 4.2 Metamodeling Uncertainty at Nonlinear and Flat Regions (Modified from Farhang-Mehr and Azarm, 2002)

The observation above is intuitive and sounds reasonable. However, we claim that a "crucial" region with great potential information is one with great prediction errors

(given information from current points and metamodel), not necessarily a highly nonlinear one with great response changes. The key issue in identifying new data points is the study and improvement of the prediction ability of current metamodels, which is not necessarily related to studies of the non-linearity of the response surface. This idea is incorporated in our method of Sequential Exploratory Experimental Design, in which designers are engaged in identifying regions with large prediction errors to find out points with great potential information. This idea is illustrated in Figure 4.3.

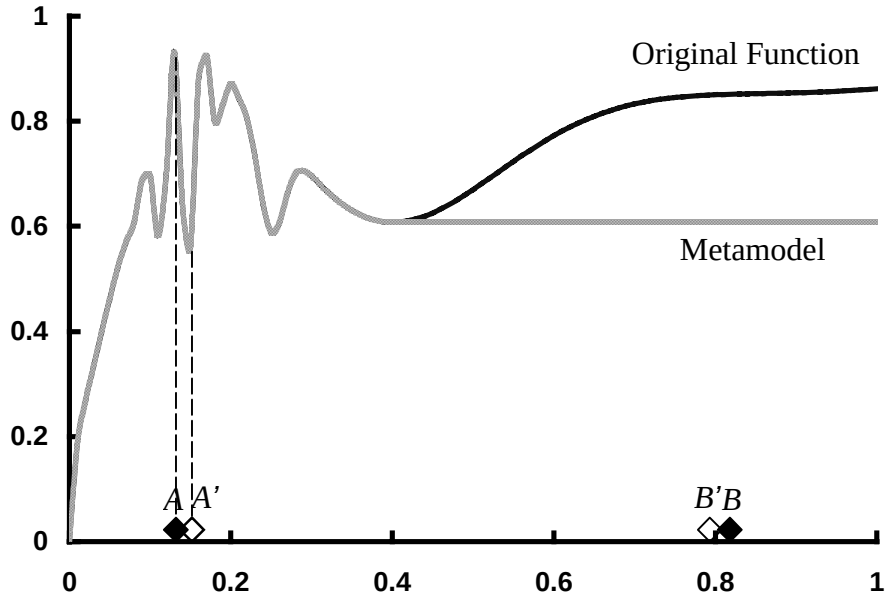


Figure 4.3 Metamodeling Uncertainty at Regions with Large and Small Prediction Errors

In Figure 4.3 the original function is the same as at shown in Figure 4.2. Suppose now we have developed a metamodel for this function; as illustrated in Figure 4.3, this

metamodel is very accurate in the region around Point *A* but has large prediction errors in the region around Point *B*. Such a metamodel could be developed through placing quite a few data points in the multimodal region around Point *A*. For example, selecting data points at each peak/bottom may yield a metamodel that is very accurate around Point *A* – this could be a result of the application of "locating more points in multimodal regions" as explained with Figure 4.2. Now the question is, given the metamodel in Figure 4.3, where should we locate new data points?

Following the idea of "locating data points in highly-nonlinear regions", it is apparent that we should add new data points in the multimodal region around Point *A* because the correlation between points in this region dampens very quickly as the distance between points increases; it is expected that points in this region have greater potential information. However, from Figure 4.3 we see that, given the metamodel in this case, points in the flat region (around Point *B*) is more informative than those in the multimodal region (around Point *A*). If the metamodel and some knowledge on its prediction errors in the design space were given, we would add in new points around Point *B*, in the region where large prediction errors take place instead of the region with high nonlinearity.

Where could be wrong with the idea of "locating data points in highly-nonlinear regions"? To answer this question, first we should look deeper into the motivation for sequential experiments. In designing sequential experiments, our aim is to add in new data points with greatest potential information about the response surface; in this way we could save time and effort on expensive simulations. New data points should be in the

regions where we have greatest uncertainty with the current experimental design and corresponding metamodel. With the natural and intuitive idea of "locating data points in highly-nonlinear regions" the assumption is that regions with high nonlinearity are those we could not predict very accurately with current metamodels. This assumption is not necessarily valid, as we see in Figure 4.3. Though in many cases, we do have greater uncertainty on response surfaces in highly nonlinear regions, the link between "high nonlinearity" and "high uncertainty with current metamodels" is not stable for all cases. Thus it is very dangerous to follow the idea of "locating data points in highly-nonlinear regions" blindly. On the other side, the idea of "locating data points in regions with large expected prediction errors" is more appropriate. The assumption behind this idea is that a region of great uncertainty is one with great prediction errors given the current metamodel. This assumption is always true since we could just express "great uncertainty" as "large prediction errors".

Another way to answer the question why "locating data points in regions with great prediction errors" is preferred to "locating data points in highly-nonlinear regions" is to study the correlation among points in these regions. The region around Point *A* in Figure 4.3 is highly nonlinear, so if we build a kriging metamodel for response surfaces in this region, the value of θ in the Gaussian correlation function (Equation (2.14)) should be very large; the region around Point *B* in Figure 4.3 is very flat, and the value of θ in the Gaussian correlation function for the kriging metamodel in this region is small. Based on this observation, the idea of "locating data points in highly-nonlinear regions" is proposed since the information that one point conveys dampens quickly in its neighborhood in a

nonlinear region, while it dampens slowly in a flat region. This is a good strategy based on very direct observations; however, a closer look would reveal its shortcomings. Suppose now we already have two data points, *A* and *A'*, in Figure 4.3. For either of these two points, we would say that it transmits only a little information in its neighborhood. While by locating them together, much more information is transmitted – they reflect the response surface between them very well. In this case, no more data points should be added between *A* and *A'*. This simple example clearly shows that once we have reflected information in a region very well (with small prediction errors), it is of little value to add in new data points though it may be highly nonlinear in this region. In sequential experiments, given current metamodels, whether the correlation among points is great or not should not be used as a guide for identifying new data points (as in "locating data points in highly-nonlinear regions"); instead, whether the correlation is explained well or not could be used in identifying new data points (as in "locating data points in regions with large prediction errors").

Generally speaking, more data points should be located in highly nonlinear regions. To build an accurate metamodel, in Figure 4.3, more data points should be located on the left half than on the right half of the design space. In this sense, the idea of "locating data points in highly nonlinear regions" sounds reasonable. This is because that the highly nonlinear regions are often with large prediction errors with current metamodels – but on the other hand, it is not always the case. Thus, in our method for designing sequential experiments, we advocate the idea of "locating data points in regions

with large prediction errors". The problem here is how to figure out the "expected prediction errors" of the current metamodel throughout the design space.

For kriging metamodels, without information from validation points, the general formula for prediction mean squared error at any new points is presented in Equation (3.7). This could be used in identifying points with great potential information of the response surface. However, in SEED we do not adopt this equation because:

1. It works only for kriging metamodels,
2. Our research in Chapter 3 suggests that only information from validation points could be used to verify the metamodel (i.e., calculate the prediction errors), and
3. Equation (3.7) is developed based on the stationary assumption (in each dimension). As we discussed before, with the stationary assumption in maximum entropy sampling, data points are selected so that minimum distances among them are maximized. Thus it is not surprising that with Equation (3.7), a candidate point far away from previous data points is usually with larger prediction mean squared errors. This is not a good estimate of real prediction errors in the design space.

In the SEED method, we propose to gather information from current data and validation points to estimate prediction errors at points throughout the design space. After developing the metamodel with current set of data points, validation points are identified to help validate the metamodel. If the metamodel is not accurate enough, prediction errors at the data points (with values of 0's if we use kriging metamodel) and

the validation points are used to build a metamodel for predicting prediction errors. Predicted prediction errors at points in the design space are then calculated to facilitate the identification of new data points. Like data points, validation points are also added sequentially to help yield more accurate estimates of prediction errors. In iteration, more and more metamodels for both response prediction and corresponding prediction errors are developed until finally we stop our process with an acceptable metamodel. Our approach is explained in detail with mathematical formulations in the next section.

4.5.3 Mathematical Formulations of Entries in the Adjusted Covariance Matrix in Sequential Exploratory Experimental Design

Mathematical realization of our ideas on sequential experimental design is described in this section. As introduced in Sections 4.3 and 4.4, the key issue in designing *D*-optimal experiments and Bayesian maximum entropy experiments is the formulation of the information matrix (the covariance matrix). Basically, our discussions in this section are built directly on maximum entropy sampling (and *D*-optimal experiments also, though indirectly, according to our discussions in previous sections).

In a design of sequential experiments, suppose currently we have n data points D_n, a corresponding metamodel for responses $\hat{f}$, and n_{error} validation points A_{ne}. The metamodel $\hat{f}$ is not accurate enough and a certain number (suppose to be m) of new data points are to be added to update the metamodel. The number of new data points is decided arbitrarily by designers after contemplating the simulation complexity, computational expense, and accuracy of the current metamodel. Our task here is then

how to identify these *m* new data points given current data/validation points and metamodels.

In the SEED approach, given current data points and the metamodel for responses, prediction errors at the validation points could be calculated and a metamodel could be developed for predicting prediction errors across the entire design space, denoted by $\hat{f}_e$. Now we got two metamodels, one for predicting response values, and the other for predicting prediction errors.

As discussed at the beginning of Section 4.5 with Equations (4.20) and (4.21), we have: given two candidate points, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, and a current set of data points, D, assuming Gaussian priors, the point $\boldsymbol{x}_i$ is more informative than $\boldsymbol{x}_j$, if $\sigma_{\boldsymbol{x}_i \boldsymbol{x}_k} \leq \sigma_{\boldsymbol{x}_j \boldsymbol{x}_k}$ for all $\boldsymbol{x}_k \in D$. According to this theorem, in maximum entropy sampling, given the stationary assumption as introduced in Section 4.4.2, new data points are allocated far from current data points (Euclidian distance) which means that the correlation between new data points and current data points is managed to be small. In (Currin, et al., 1991), the authors point out that this is equivalent to maximizing the determinant of the prior covariance matrix in Bayesian entropy design, as expressed in Section 4.4.3.

The discussion above enables us to develop a method in which prediction errors could be accounted in sequential experimental design. In maximum entropy sampling, given the Gaussian priors, with the stationary assumption, the correlation between points is merely based on their Euclidian distance. The farther the distance is, the smaller the correlations are. A point with weak correlations with other data points is one with large

potential information. In a single-stage Bayesian entropy design, by maximizing the determinant of the covariance matrix, the information content of a set of data points is maximized and as a result, data points are allocated to spread over the design space. In SEED, since we have information on both response prediction and error prediction, we could assign weaker correlations in the regions with large prediction errors, which increase the informational worth of an experiment conducted in those regions.

In the SEED method, the stationary assumption of the covariance matrix used in single-stage maximum entropy sampling, as introduced in Section 4.4.3, is no longer hold valid. Prior to the design of the first set of experiments, no information is available about the actual response function and as reasoned before, the stationary assumption is appropriate for the prior distribution. Thus a set of maximum entropy experiments, or as stated before, a space-filling experiment or a previous set of data points, could be used to develop the first-round metamodel. However, in selecting new data points, prediction errors are considered and the covariance will be modified to reflect this information; in this case, the stationary assumption no longer holds.

How to modify the formulation of entries of the covariance matrix to reflect the information on prediction errors in the design space? As introduced in Section 4.4.2, entries of the covariance matrix in maximum entropy sampling is formulated after Equations (4.19) and (4.20). In the SEED method, following the idea of "locating new data points in regions with large prediction errors", we decrease the correlations between points in regions with large prediction errors. This decreased correlation ensures that new data points will be "dragged" to the corresponding regions through entropy maximization.

There are two ways to modify the formula of entries in the covariance matrix (Equation (4.19)), one is to modify the formula of the correlation function *R*, the other is to introduce a correcting factor in Equation (4.19) without changing the correlation function. This is described in following paragraphs.

4.5.3.1 Formulation of Entries in the Covariance Matrix without Changing the Correlation Function

The information influence of a data points in its neighborhood is small when the predicted prediction error is large; thus we introduce some correcting factor for the covariance to incorporate prediction errors and update the covariance of two points (Equation (4.19)) as:

$$\sigma_{ij}^{adj} = \alpha_i \alpha_j \sigma_{ij} = \alpha_i \alpha_j \sigma^2 R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \tag{4.24}$$

In Equation (4.24), α_i is the coefficient to reflect the current metamodel's uncertainty (prediction errors) at point $\boldsymbol{x}_i$, and α_j is the coefficient to reflect the current metamodel's uncertainty at point $\boldsymbol{x}_j$. Theoretically, α_i and α_j should have values between (0, 1]. A value close to 1 means that the prediction error is small, and thus no much adjustment is needed on covariance between this point and others. A value close to 0 means that with the current metamodel we can hardly tell the actual response value at this point, and thus, correlations of this point with other points should be greatly decreased. To use Equation (4.24) in SEED is like "pulling" data points to regions with large metamodel uncertainty through assigning small correlations to points in those regions.

The formulation of coefficients (α_i and α_j) should satisfy the following criteria and ideas:

- $0 < \alpha_i, \alpha_j \leq 1$.
- α_i (and α_j) should be a decreasing function of predicted prediction errors, i.e., larger values should be assigned to α_i (and α_j) for points with smaller prediction errors.
- In the process of designing sequential experiments, since the information from current metamodels of response values and prediction errors is usually inaccurate, we should balance between "locating points in regions with large prediction errors" and "having points spread over the design space". New data points may not be those with largest predicted prediction errors with current metamodels; they should also have as long distance from current data points as possible. A trade-off is needed. This is like "twisting" the data points with two forces, one pulling points to regions with large predicted prediction errors, and the other to regions far from current data points. Based on the discussions above, it may be better not to define $\alpha_i \in (0,1]$ in practice. Points with very large predicted prediction error should not have values of α_i close to 0; otherwise the trade-off between "removing prediction error" and "spreading over the design space" will be damaged because new data points will tend to be located where the covariance (Equation (4.24)) is close to 0.

- As will be shown later, a factor λ is introduced in SEED to balance the weight of consideration of "prediction errors" and "space-filling" in the identification of new data points. In practice, we have $\alpha_i \in \left[1-\frac{1}{\lambda},1\right]$.

In this dissertation, we calculate α_i with the following equation:

$$\alpha_i = 1 - relative.uncert = 1 - \left|\frac{e_i}{\lambda e_{\max}}\right| \tag{4.25}$$

where *realive.uncert* is the measurement of relative uncertainty on prediction, which should range in [0,1), representing high uncertainty with values close to 1 and low uncertainty with values close to 0. e_i is the predicted prediction error at the current point, and $e_{\max}$ is the maximum predicted prediction error in the design space. In practice, it may be difficult to find the global maximum predicted prediction error with the metamodel, thus, we may just use $e_{\max}$ from a certain optimization; when e_i at some particular points exceeds $e_{\max}$, we may force the value of $\frac{e_i}{e_{\max}}$=1. In some cases we may use a value of e_{max} that is smaller than the known value to remove sharp peaks and increase the number of points with "maximum prediction errors". λ is the coefficient used to adjust the value of α_i. As discussed before, in our entropy optimization process, the allocation of new data points is affected by two factors: one is to make points "spread over" the design space as evenly as possible, and the other is to locate points in "regions of interest" (or "regions with large prediction errors"). As it is often the case, in the beginning of metamodeling (usually first iteration) we do not have much information, and

our estimation of prediction errors is also with great uncertainty; thus at this time we should not emphasize too much on "regions of interest" and try to have points spread over the design space. A large value of λ helps achieve this. When design evolves and we have more information and confidence on our prediction of prediction errors of the metamodel, we may emphasize more on "regions of interest"; a small value of λ helps achieve this goal. In this dissertation we use λ=2, which makes α_i ranging in [0.5,1]; in this way we do not "exaggerate too much" in adjusting the covariance. Equation (4.24) is extended as:

$$\sigma_{ij}^{adj} = \sigma_{ij}\alpha_i\alpha_j = \left[\sigma\left(1-\left|\frac{e_i}{\lambda e_{\max}}\right|\right)\right]\left[\sigma\left(1-\left|\frac{e_j}{\lambda e_{\max}}\right|\right)\right]R\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right) \tag{4.26}$$

Note that Equation (4.26) is only used for calculating the covariance between one candidate point and one current data point. Suppose we have a set of data points D_n and m candidate points C_m. The $(n+m)\times(n+m)$ covariance matrix is expressed as:

$$\text{Cov} = \sigma^2 \begin{pmatrix} \begin{bmatrix} & \\ \boldsymbol{n\times n} & \\ & \end{bmatrix} & \begin{bmatrix} \\ \boldsymbol{m\times n} \\ \end{bmatrix} \\ \begin{bmatrix} & \\ \boldsymbol{n\times m} & \\ & \end{bmatrix} & \begin{bmatrix} \\ \boldsymbol{m\times m} \\ \end{bmatrix} \end{pmatrix} \tag{4.27}$$

In the covariance matrix as presented in Equation (4.27), the $n \times n$ sub-matrix contains the covariance between current data points, and the $m \times m$ sub-matrix represents covariance between candidate points. Note that the diagonal entries of the covariance matrix are filled with 1's (see Equation (4.20)). Equation (4.26) is only used to calculate entries in the $n \times m$ and $m \times n$ sub-matrices in Equation (4.27). Thus, in SEED, entries of the covariance matrix is calculated as below:

$$\sigma_{ij} = \sigma^2 \cdot \begin{cases} \left(1 - \left|\frac{e_i}{\lambda e_{\max}}\right|\right)\left(1 - \left|\frac{e_j}{\lambda e_{\max}}\right|\right) R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) & when \begin{cases} i \le n, j > n \\ i > n, j \le n \end{cases} .and. R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \ne 1 \\ 1 & when \begin{cases} i \le n, j > n \\ i > n, j \le n \end{cases} .and. R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) = 1 \\ R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) & when \begin{cases} i \le n, j \le n \\ i > n, j > n \end{cases} \quad i \ne j \\ 1 & when \quad i = j \end{cases} \tag{4.28}$$

In Equations (4.27) and (4.28), we see that the covariance among current data points (the $n \times n$ sub-matrix) is not adjusted; it remains the same as in Equation (4.19). This is natural since there is no prediction error at data points (supposing we are using the kriging metamodels) and there will be no adjustment following Equation (4.26). We also see that the covariance among candidate points (the $m \times m$ sub-matrix) also remains unadjusted. This is because we do not want to have multiple new data points clustering in the region with large prediction errors – if the formulation of covariance among candidate points follows Equation (4.26), it is very likely that all new data points are identical (or very close to one another). To formulate the covariance among candidate points in the normal way (following Equation (4.19)) is actually to force new data points spread all over the design space. To use the covariance matrix in sequential experiments,

the focus is on the correlations between current data points and candidate points, i.e., the $n \times m$ and $m \times n$ sub-matrices in Equation (4.27). We only adjust entries in these two sub-matrices.

As for the correlation function $R(x_i, x_j)$ in Equation (4.28), we use the Gaussian function, as shown below:

$$R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) = \prod_{k=1}^{n_{dv}} \exp(-\theta_k \left|d_k\right|^2) \tag{4.29}$$

where n_{dv} is the number of design variables, θ_k are the unknown correlation parameters used to fit the model, and $d_k = x_k^i - x_k^j$ which is the distance between the k^{th} components of points $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$.

In single-stage maximum entropy sampling, the covariance matrix is built by using identical values for all θ_k's. This is reasonable because no information is available at the very beginning of design. While in sequential experimental design, when kriging metamodel is used, θ_k's from current kriging metamodel could be used in next round of experimental design, i.e., values of θ_k's in Equation (4.29) keep being updated in accordance with the kriging metamodels.

Equations (4.27) and (4.28) are used to formulate the covariance matrix in SEED (without changing the correlation function). In the next section, we will discuss formulations of the covariance matrix in SEED through changing the correlation function.

4.5.3.2 Formulation of Entries in the Covariance Matrix through Changing the Correlation Function

In Section 4.5.3.1, we discussed methods to formulate entries of the covariance matrix without changing the correlation function. The idea is expressed in Equation (4.24) and the mathematical formulation of the covariance matrix is presented in Equations (4.27) and (4.28). In this section, we explore the method of adjusting entries in the covariance matrix through modifying the correlation function between points.

In Section 4.5.3.1, the adjustment of entries in the covariance matrix is done by adding correcting coefficients, α_i and α_j, in Equation (4.24). In that method, the correlation function, $R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right)$, is not changed (except that the values of θ_k's are updated in accordance with the kriging metamodels); the adjustment happens outside of the correlation function. Now let us explore ways to adjust the entries in the covariance matrix through modification of the correlation function.

Since we use the Gaussian correlation function as shown in Equation (4.29), the covariance between two points (Equation (4.19)) could be expressed as:

$$\sigma_{ij} = \sigma^2 R\left(x_i, x_j\right) = \sigma^2 \prod_{k=1}^{n_{dv}} \exp(-\theta_k \left|d_k\right|^2) \qquad (4.30)$$

The values of θ_k's in Equation (4.30) are actually indicators of degrees of correlations between the points. The larger the value of θ_k is, the less covariance between two points in the direction of the *k*th component (as indicated in Equation (4.30)), and thus the less information that one point transmits in its neighborhood. As discussed in Section 4.5.3.1, in designing sequential experiments, we should adjust entries in the covariance matrix to

incorporate the information of prediction errors. Points in regions with large prediction errors should have smaller covariance (thus, correlations) than in regions with small prediction errors. This could be achieved through modifying the formulation of correlation functions (Equation (4.29)) to:

$$R^{adj}(x_i, x_j) = \prod_{k=1}^{n_{dv}} \exp(-\theta_k^{adj} |d_k|^2) = \prod_{k=1}^{n_{dv}} \exp(-\beta_i \beta_j \theta_k |d_k|^2) \qquad (4.31)$$

As we see in Equation (4.31), two correcting coefficients, β_i and β_j, are added in the correlation function to reflect information on prediction errors. Similar to α_i and α_j in Equation (4.24), β_i is the coefficient to reflect the current metamodel's uncertainty (prediction errors) at point $\boldsymbol{x}_i$, and β_j is the coefficient to reflect the current metamodel's uncertainty at point $\boldsymbol{x}_j$. The formulation of coefficients β_i and β_j should satisfy the following criteria and ideas:

- When the prediction error at a point $\boldsymbol{x}_i$ is 0 (e.g., current data points with kriging metamodels), the corresponding coefficient β_i should have the value of 1, which means no adjustment is needed at this point. This could be viewed as the lower bound of β_i.
- When the prediction error at a point $\boldsymbol{x}_i$ is large, correlation with this point should be adjusted to a smaller value, which means that the corresponding coefficient β_i should have a value larger than 1.
- The upper bound for β_i (and β_j) is decided arbitrarily by the designers. Based on previous experience, usually, the value of θ_k^{adj} should not be too

large or small to yield efficient and effective computation results in maximum entropy sampling (say, e.g., smaller than 100 and larger than 5). This puts constraints on the upper bound for β_i. In order not to get too large values of θ_k^{adj} (no larger than 100), designers may select smaller upper bounds for β_i (and β_j).

- In some cases, when the difference between upper and lower bounds of β_i are too small, designers may also want to lower the lower bound. This is like "shifting" the range of β_i from [1, *upper bound*] to [*lower bound*, *upper bound*], where *lower bound* is smaller than 1 and larger than 0. An extreme example is to use $\beta_i \in$ [*lower bound*, 1].

Based on discussions above, we could calculate β_i with the following equation:

$$\beta_i = 1 + \lambda \left| \frac{e_i}{e_{\max}} \right| \tag{4.32}$$

where again, e_i is the predicted prediction error at a candidate point, e_{max} is the maximum predicted prediction error with current metamodels. In practice, we get the value of e_{max} with optimization tools; global optimum is not guaranteed. When e_i is larger than the estimated e_{max} at some candidate points, we force the value of $\left| \frac{e_i}{e_{\max}} \right|$ to be 1. Similar to the method discussed in Section 4.5.3.1, λ is used to gauge how much the parameter θ_k^{adj} is adjusted or "twisted". Usually we set $\lambda = 1$. The range of β_i with Equation (4.32) is [1, 1+λ].

There are also many other possible formulations for β_i. For example, if in some case we want to have β_i ranged in [*lower bound*, 1], we could use the following equation:

$$\beta_i = \frac{1}{\lambda} + \left|\frac{e_i}{e_{\max}}\right| - \frac{1}{\lambda}\left|\frac{e_i}{e_{\max}}\right| \tag{4.33}$$

With Equation (4.33), the range of β_i is $\left[\frac{1}{\lambda},1\right]$. Equation (4.33) could be used in cases where all θ_k's are very large; thus instead of increasing values of θ_k's for points with large prediction errors, we decrease values of θ_k's for points with small prediction errors.

Given the equations of β_i, entries of the adjusted covariance matrix (Equation (4.27)) could be calculated as:

$$\begin{aligned}\sigma_{ij} &= \sigma^2 R^{adj}\left(x_i, x_j\right) \\ &= \sigma^2 \prod_{k=1}^{n_{dv}} \exp(-\theta_k^{adj}\left|d_k\right|^2) \\ &= \sigma^2 \prod_{k=1}^{n_{dv}} \exp(-\beta_i\beta_j\theta_k\left|d_k\right|^2)\end{aligned} \tag{4.34}$$

β_i and β_j are calculated with Equations (4.32) or (4.33). Equations (4.27) and (4.34) are the formulations of the adjusted covariance matrix with the modified correlation function that reflects information of prediction errors in the design space. Note that Equation (4.34) is appropriate for calculating all entries in the covariance matrix (Equation (4.27)). This is different from the method discussed in Section 4.5.3.1, in which we have different equations for entries in different sub-matrices in the covariance matrix. The reason is that: the correcting coefficients, β_i and β_j, are multiplied by the distance $\left|d_k\right|$ (and then

used in the exponential calculation) in the correlation function; maximum entropy sampling with Equation (4.34) will not result in a clustering of new data points. As for the method introduced in Section 4.5.3.1, the correcting coefficients, α_i and α_j, are put outside of the exponential calculation; clustering could happen if we use the adjusted equation (Equation (4.26)) to calculate the covariance among the candidate points.

In this section, we discussed two approaches to formulate entries of the adjusted covariance matrix so that information of prediction errors in the design space could be taken into consideration in identifying new data points. Correcting coefficients are used in the mathematical formulation to "drag" candidate points to regions with large prediction errors. After formulation of the adjusted covariance matrix, new data points could be identified through maximizing the determinant of the covariance matrix. The metamodel is then updated and new validation points are added to validate the metamodel. Selection of new validation points could follow similar strategy to that of new data points because we want to gain maximum possible information with every new point, no matter it is used as a data point or a validation point. It is very possible that some validation points change to data points in sequential experiments. This is incorporated in the Sequential Exploratory Experimental Design method. Steps and flowchart of the SEED are described in the next section with practical considerations and discussions.

4.5.4 Flowchart and Steps of the Sequential Exploratory Experimental Design Method

The method of Sequential Exploratory Experimental Design is developed to facilitate sequential design of computer experiments. In this dissertation, it is used in the frame of RCEM (and later, the Efficient Robust Concept Exploration Method as will be developed in Chapter 6) to help build appropriate metamodels in exploration of robust solutions in the design space. The flowchart of SEED is presented in Figure 4.4.

As illustrated in Figure 4.4, appropriate metamodels of responses are developed through designing sequential experiments in multiple iterations. Step 1 and Step 2 in SEED are the initialization of the whole metamodeling process. Each iteration in SEED consists six steps, from Step 3 to Step 8 as shown in Figure 4.4. Details of actions in each step are described below.

Step 1 – Initial Experimental Design. As described earlier, there may be three ways to design the initial experiments. If previous observations at some data points are available, these points may be used as the first set of experiments. Space-filling experiments or traditional experiments may also be used as the initial experiments. Or we could design experiments following the maximum entropy sampling method with stationary assumptions (no adjustment to the covariance matrix) – maximizes the determinant of the prior covariance matrix, thus maximizes the expected reduction of the entropy due to the experimentation, and maximum expected information is gained from the set of experiments. The covariance matrix could be constructed using Equations

(4.19) and (4.20). Assuming a rapidly decaying correlation, the value of θ could be set at a large value, e.g., 10.

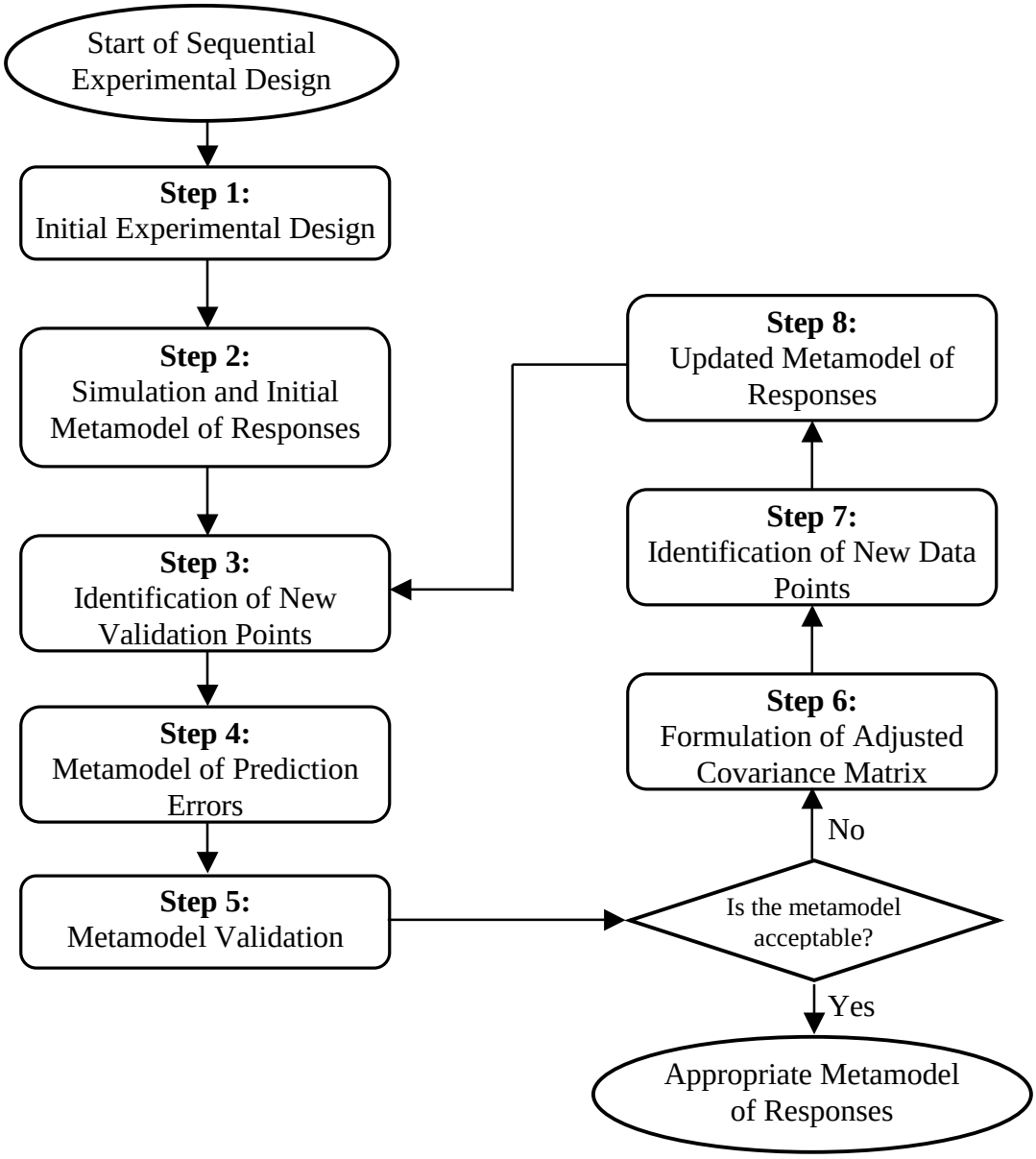


Figure 4.4 Flowchart of the Sequential Exploratory Experimental Design Method

Step 2: Simulation and Initial Metamodel of Responses. In this step, observations are made at all data points in the initial experimental design, and the corresponding initial metamodel of responses is developed based on information from these observations.

Step 3: Identification of New Validation Points. In this step validation points are identified and information at these points is collected. At the beginning of sequential experimental design, when we still have great uncertainty with current metamodels we had better select validation points to "spread over" the design space so that we could get information of the responses across the design space instead of being constrained to some narrow regions. Thus at the beginning of sequential metamodeling, we use the "single-stage" maximum entropy sampling method (as discussed in Section 4.4 and quoted as an optional approach in *Step 1*) to identify validation points. When there is sufficient information – again, this is a decision made by designers arbitrarily – from previous validation points, methods used to help identify new data points (as will be described in *Step 6, 7*) could be used to identify validation points; differences from that of identifying new data points lie in that:

1. When identifying new validation points we examine all possible points in the design space except current observed points (data and validation points), while when identifying new data points we examine all points that were not used as data points (which means, current validation points are considered); and
2. In the process of identifying new validation points, the roles of validation points and data points are temporarily switched. A metamodel of response is

developed with all validation points, then prediction errors at data points are calculated and a metamodel of prediction errors is built. In this way we expect to bring in most informative new validation points given current observed data and validation points.

We then try to put validation points in regions where we expect to have large uncertainty on the response values based on the previous metamodel. However, this approach should only to be used when we are quite certain about the metamodels for both responses and prediction errors, i.e., observations at quite a lot points have been made. The number of validation points is also a problem. In this chapter, in the intermediate iterations, we try to maintain the number of validation points, n_{error}, equal to the number of data points plus 1, i.e., $n_d + 1$.

Step 4: Metamodel of Prediction Errors. In this step, prediction errors at data and validation points are calculated, and a metamodel for predicting prediction errors is developed. The maximum absolute predicted prediction error across the design space is obtained. Information from the metamodel of prediction errors may help validating the metamodel of responses.

Step 5: Metamodel Validation. In this step we follow the method discussed in Chapter 3. If the result suggests that new data points be needed, we go to the next step, *Step 6*. Otherwise we could stop and use the current metamodel in later design stages; in this case we could also go back to *Step 3* and add in more validation points if we could afford more observations.

Different stopping criteria may be used in sequential experimental design. We could stop once an appropriate metamodel is built (based on some preset requirement of metamodel accuracy), as discussed above and in Chapter 3. Or, we could preset the total number of points (or data points) that we could afford; once we finish observations at enough points, the process of sequential experiments and metamodeling will stop. This stopping criterion is usually used when the simulation is very expensive.

After sequential experimental design finishes, the current metamodel of responses could be used in future design processes (e.g., exploration for robust solutions). We could also incorporate data and validation points and develop a new metamodel; it is expected that a more accurate metamodel could be achieved with more data points. However, this approach is inappropriate in some cases when enough data points have been observed. Response development and prediction with a metamodel (especially the kriging model) will be very time-consuming, which makes the multi-objective exploration of the multivariable design space very expensive. In such cases, we prefer using an appropriate metamodel with as few data points as possible.

Step 6: Formulation of the Adjusted Covariance Matrix. This step is the core of the SEED method. As described in Section 4.5.3, we have two approaches to adjust entries in the covariance matrix to reflect information from the metamodel of prediction errors. Either approach could be used to formulate the adjusted covariance matrix. Suppose the number of current data points is n_d, and we decide to add in n_{new} data points. The covariance matrix should be $(n_d + n_{new}) \times (n_d + n_{new})$. The first $n_d \times n_d$ rows and

columns of the matrix correspond to current data points. The entries are updated according to Equations (4.27), (4.28), and (4.34). For more details, see discussions in Section 4.5.3.

In the formulation of the adjusted covariance matrix, we should pay much attention to the selection of values of θ, λ, and e_{max}. For discussions and instructions, see Section 4.5.3.

Step 7: Identification of New Data Points. In this step, through maximizing the determinant of the adjusted $(n_d + n_{new}) \times (n_d + n_{new})$ covariance matrix developed in *Step 6*, we could identify a set of n_{new} new data points.

Step 8: Updated Metamodel of Responses. In this step we develop a new metamodel with information from the new set of data points. After development of the new metamodel, we go to Steps 3 and 4 in the next iteration to validate its accuracy. Metamodeling in the current iteration stops at this step.

The method of Sequential Exploratory Experimental Design is described in this section through the overview of sequential experiments (Section 4.5.1), discussion on selection of data and validation points (Section 4.5.2), mathematical formulations of the adjusted covariance matrix in maximum entropy sampling (Section 4.5.3), and presentation of the flowchart and steps of SEED (Section 4.5.4). In the next section, the SEED method is tested with a single-variable function.

4.6 APPLICATION OF THE SEED METHOD – A SINGLE-VARIABLE EXAMPLE

Mathematical formulations and designing steps of the SEED method are described in Section 4.5. In this section, the SEED method is tested with a single-variable function. The single-variable function is introduced in Section 4.6.1. In Section 4.6.1, we also presented two designs for comparison with the SEED method: in one of which we identify all data points in one step, and in the other one we identify the data point sequentially but without adjusting the covariance matrix (i.e., following the approach as described in (Currin, et al., 1991)). The SEED method with Formulation I (as discussed in Section 4.5.3.1) is applied with the single-variable function in Section 4.6.2. The SEED method with Formulation II (as discussed in Section 4.5.3.2) is applied with the single-variable function in Section 4.6.3. In this section, we use kriging metamodels.

4.6.1 Single-Stage Experimental Design with A Single-Variable Function

In this section, we use a single-variable function, presented in Equation (4.35), as the deterministic computer simulation for which we develop kriging metamodels. A graph of this function is shown in Figure 4.5. As we see from the equation and graph, the design space is $x = [0, 1]$. In this design space, the maximum response value is $y = 1.852$ at $x = 0.04$, and the minimum response value is around $y = -1.563$ at around $x = 0.14$; the response range is 3.415.

$$f(x)=\begin{cases}\dfrac{Sin\left(10\pi\left(x+0.01\right)\right)}{x+0.5} & 0\le x\le 0.19\\ 0 & 0.19<x\le 1\end{cases} \tag{4.35}$$

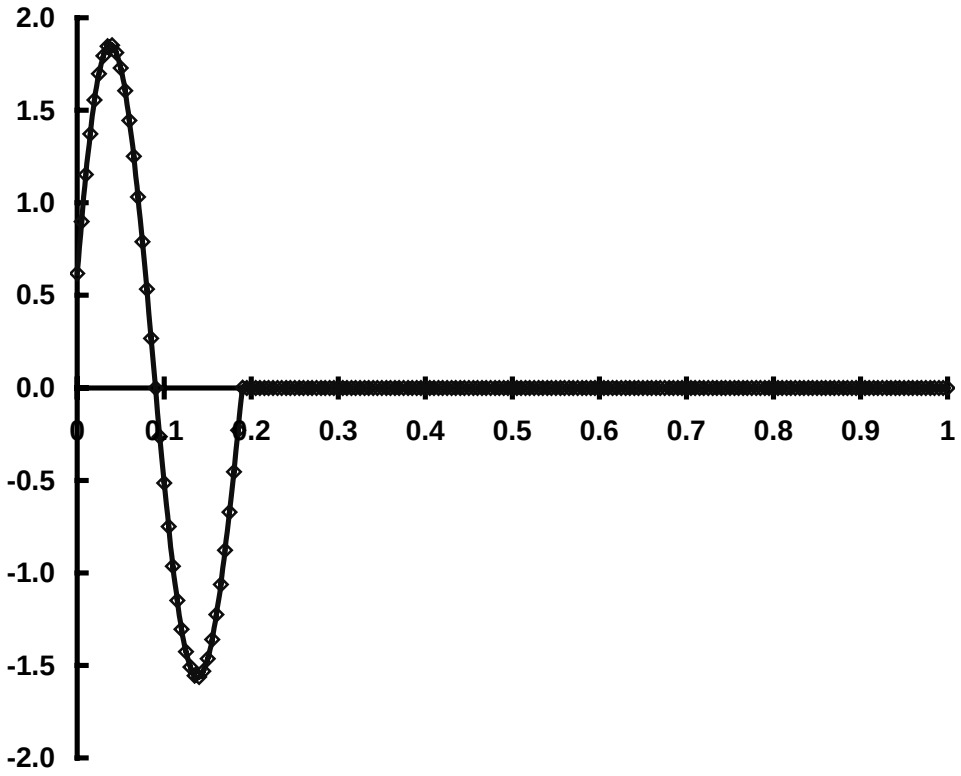


Figure 4.5 A Single-Variable Function

To develop a kriging metamodel for this single-variable function, suppose that we plan to use 11 observed points. In Sections 4.6.2 and 4.6.3, these 11 observed points are identified with the SEED method. As a comparison, in this section the data points are identified in a "single-stage" manner in which the covariance matrix is not adjusted with

information on prediction errors. With the word "single-stage" we mean two approaches as explained below:

- One is to identify 11 data points in one step; the easiest way is to have 11 points evenly spread over the design space, as listed in Table 4.1. We name this set of points as Data Set I. Plot of the corresponding kriging metamodel is illustrated in Figure 4.6. The value of θ for this kriging model is 99.99993.
- The other is to design "sequential" experiments following the approach in (Currin, et al., 1991). First a 3×3 covariance matrix is built to help identify the first 3 points. Then based on this information, a 7×7 covariance matrix is built to help find out 4 more points. After this, we add in one new data point and one new validation point in each iteration until finally we get 11 points. We still use Equations (4.19) and (4.20) in this approach. With this approach, we got two sets of points that are "equally" good – without information of responses at the observed points we cannot tell which data set is better. However, the first 11 data points identified in these two sets are the same (though the sequence of the points are different). These 11 data points are listed in Table 4.2. We name this set of points as Data Set II. Plot of the corresponding metamodel is illustrated in Figure 4.7. The value of θ for this kriging model is 99.9999.

As discussed at the end of Section 4.4, in this chapter, since we are dealing with very simple examples (thus computation time on entropy optimization is not a problem),

to ensure the correctness of our comparison and verification, we do not use the "hiker" method as described in (Currin, et al., 1991) to find out the optimal set of data points. Instead, various optimization algorithms, such as sequential quadratic programming, simulated annealing, etc., are used to ensure the achievement of global optimum in maximizing the determinant of the covariance matrix.

Table 4.1 Data Set I for the Single-Variable Function – 11 Data Points Evenly Spread Over the Design Space

x	0	0.1	0.2	0.3	0.4	0.5
y	0.618	-0.515	0.0	0.0	0.0	0.0
x	0.6	0.7	0.8	0.9	1.0	
y	0.0	0.0	0.0	0.0	0.0	

Table 4.2 Data Set II for the Single-Variable Function – 11 Data Points Identified in A Single-Stage 6-Step Manner

Step	**Point**	**Data Set II**	
		x	***y***
I	**1**	0	0.618
	2	0.5	0.0
	3	1	0.0
II	**4**	0.167	-0.991
	5	0.333	0.0
	6	0.667	0.0
	7	0.833	0.0
III	**8**	0.917	0.0
IV	**9**	0.417	0.0
V	**10**	0.083	0.374
VI	**11**	0.583	0.0

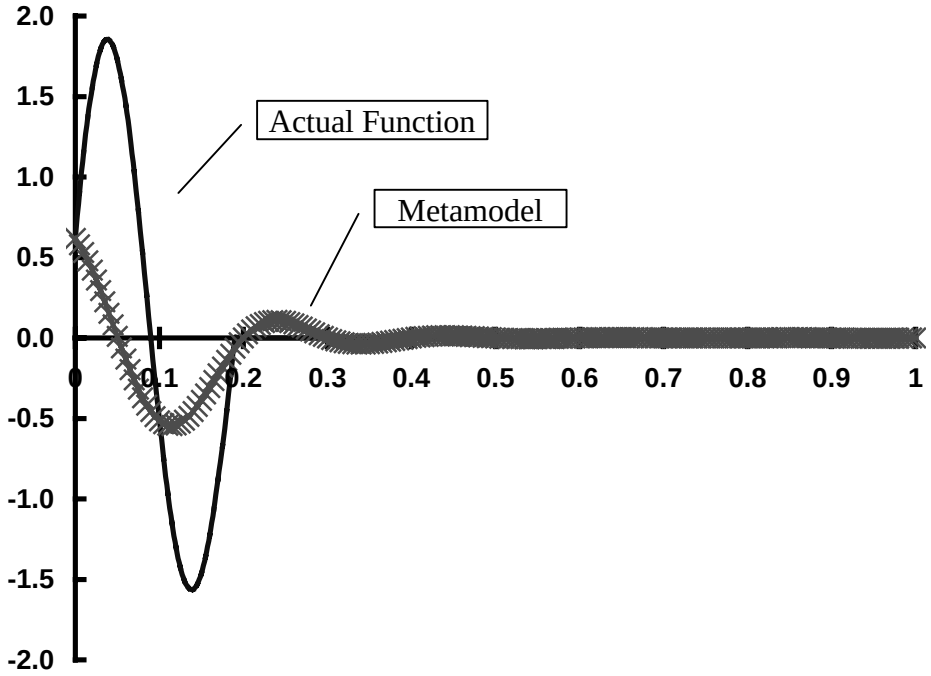


Figure 4.6 Metamodel (I) – For Data Set I

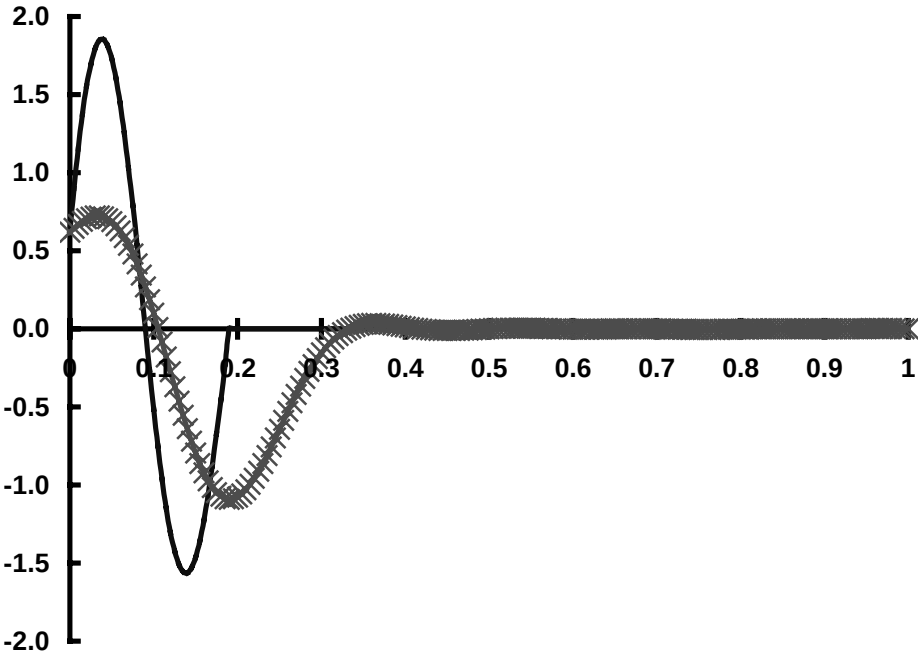


Figure 4.7 Metamodel (II) – For Data Set II

In the 3rd step of the "sequential" experiments as presented in Table 4.2, the new data point is identified as $x = 0.917$. It could be proved that there exists another solution, $x = 0.083$, which has the same (which is maximum) value of the determinant of the covariance matrix as the solution of $x = 0.917$ does. Thus, in experimental design, the designer may select another set of data points. However, after identifying 11 data points we found that the two different ways yield the same result; the only difference lies in the sequence that new points are added. It can be illustrated that starting from the 12th point, there will also be two different sets of data points that are equally good from the viewpoint of Currin's single-stage experimental design.

In Figure 4.6 and Figure 4.7 we see that the kriging metamodels (I and II) with "single-stage" experiments are accurate when values of the input variable x are not small. When x is smaller than 0.2, the kriging metamodels are inaccurate – the peak and the bottom of the response surface at low x values are not fully captured by the kriging metamodels.

The maximum absolute error (MAX) and root mean squared error (RMSE) of these three metamodels are calculated with Equations (2.7) and (2.9), and listed in Table 4.3. To calculate MAX and RMSE we use observations from 201 points that evenly spread over the design space of [0,1]. As discussed in Chapter 2, the smaller the values of MAX and RMSE, the more accurate the corresponding metamodel is. Values of MAX and RMSE from Table 4.3 are consistent with our observations with Figure 4.6 and Figure 4.7. These values will be further used in comparison with those of metamodels developed in the following sections with the SEED method.

Table 4.3 MAX and RMSE of Three Metamodels

	Metamodel (I)	Metamodel (II)
MAX	1.730	1.711
RMSE	0.452	0.472

4.6.2 Application of SEED in the Single-Variable Example – Formulation I

In this section, the SEED method with Formulation I (as described in Section 4.5.3.1, Equations (4.27) and (4.28)) is applied to facilitate the development of an acceptable kriging metamodel for the single-variable function as introduced in Section 4.6.1. In this design of sequential experiments, we plan to identify 3 data points and 4 validation points first; after this, we add in one new data point and one new validation point in each iteration until finally we get 11 points. Stopping criteria on metamodel accuracy will not be used in this example, thus no metamodel validation is done in Step 5 in SEED. In this example, at the end of the sequential experimental design and metamodeling, we will develop a "final" metamodel with all 11 points and compare the accuracy of the metamodel with Metamodels (I) and (II) that are developed in Section 4.6.1.

Iteration I – Step 1: Initial Experimental Design. In this step we design the initial experiments. The number of data points to be identified is $n_d = 3$. As discussed in Section 4.5.4, there are three ways to design the initial experiments. In this case, we decide to use the method of maximum entropy sampling as stated in (Currin, et al., 1991). The stationary assumption holds and no adjustment is done to the covariance matrix. Entries of the covariance matrix are calculated with Equations (4.19) and (4.20). Since

there is no previous information available, we set the value of θ as 10 in building the covariance matrix. The results (initial set of data points) are listed in Table 4.4.

We wrote a FORTRAN program to construct the covariance matrix given a number of candidate points. Then the determinant of this covariance matrix is calculated with another FORTRAN program. These FORTRAN programs are linked in iSIGHT, and optimization is done to find out the set of candidate points with the largest value of determinant of the covariance matrix. In our study, since the computation expense in entropy optimization is not high with the single-variable example, we do not use the "hiker" method as introduced in (Currin, et al., 1991). Instead, we use various optimization techniques as implemented in iSIGHT to ensure the achievement of global optimum. These optimization techniques include: Sequential Quadratic Programming (DONLP, NLPQL), Method of Feasible Directions (CONMIN), Modified Method of Feasible Directions, Mixed Integer Optimization (MOST), and Simulated Annealing (SA). In real-world applications, we could either use the "hiker" approach or any of the first 6 techniques listed above. When the computation expense is expected to be high, the SA technique may not be used since it requires a long time for convergence. The C programs, usage of iSIGHT, and introduction of these optimization techniques are described in Appendix A in detail.

Iteration I – Step 2: Simulation and Initial Metamodel of Responses. Response values are observed at 3 data points. Data points and the corresponding response values are listed in Table 4.4. A kriging metamodel is then developed based on

the information. The value of θ is 98.71232; the kriging metamodel is illustrated in Figure 4.8.

Table 4.4 Initial Experiments

x	0.0	0.5	1.0
y	0.618	0.0	0.0

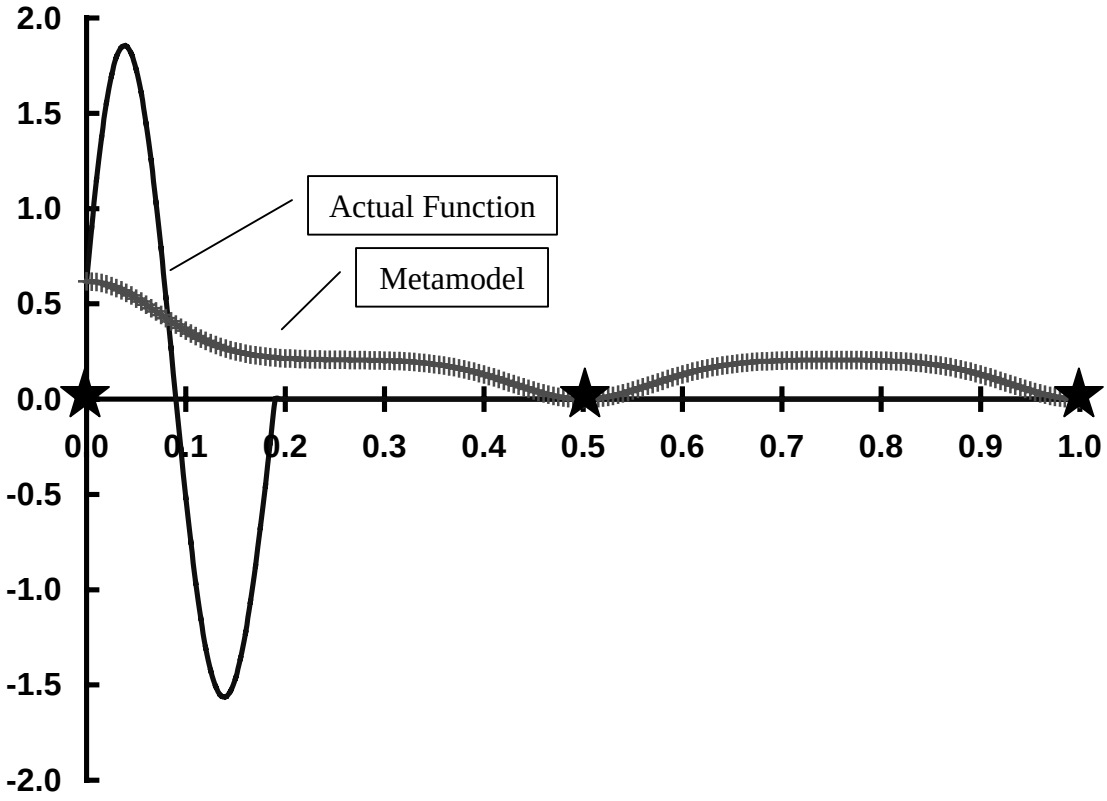


Figure 4.8 Initial Metamodel with 3 Data Points

Iteration I – Step 3: Identification of New Validation Points. In the first iteration, we only have information from data points and the initial metamodel. In this step we need to identify validation points for the first iteration. As described before, without enough information for metamodel validation, we will use the same method as that for data points (in *Step 1*) to select validation points. In this sense, the validation points could be viewed as "possible data points" if we had decided to select more data points in *Step 1*. Since we have n_d = 3 data points in the one-dimension problem, the number of validation points could be $n_{error} = n_d + 1 = 4$.

A covariance matrix is constructed with the first 3 rows and columns corresponding to the 3 data points that we already decided, and the last 4 rows and columns corresponding to 4 candidate points. Through maximization of the determinant of this 7×7 covariance matrix, we could identify 3 validation points for the first iteration as listed in Table 4.5.

Table 4.5 Validation Points in the 1st Iteration

x	0.167	0.333	0.667	0.833
y_{pred}	0.232	0.193	0.193	0.193
y_{actual}	-0.991	0.0	0.0	0.0
y_{error}	1.223	0.193	0.193	0.193

Iteration I – Step 4: Metamodel of Prediction Errors. In this step, prediction errors at both data and validation points are used to develop a metamodel to predict prediction errors across the design space. The prediction errors are calculated following

the equation of $y_{error} = y_{pred} - y_{actual}$, and listed in Table 4.5. Note that prediction errors at 3 data points are zero, which is not shown in Table 4.5.

A kriging metamodel for predicting prediction errors is developed, and the plot for predicted prediction errors $\hat{y}_{error}$ vs. x is drawn in Figure 4.9. The value of θ is 99.99880. The data points are represented by stars and validation points are presented by solid circles in Figure 4.9. In Figure 4.9 we see that the predicted prediction error is large when x values are small, and tend to be smaller when x values become larger. This is the same as we observed from Figure 4.8. The usage of validation points not only helps us know how accurate a metamodel is, but also provides us information on how the metamodel performs in the design space. However, the metamodel of prediction error is not very accurate because we have information at only 3 data points and 4 validation points. The maximum absolute predicted prediction error, $e_{max} \approx 1.3$, is found through optimization.

In Figure 4.9 we also see that local maximum predicted prediction errors tend to locate at validation points. This is partly because that we do not have sufficient validation points to provide more accurate information on prediction errors. Another reason may be that the example is a single-variable function; in multivariable examples validation points may not have (local) maximum predicted prediction errors because the surface of prediction errors may be "twisted" due to interactions among the design variables. In each step of the sequential experimental design, we should try to get as accurate information as possible; however, it is not necessary to get very accurate

metamodels (neither for responses nor for prediction errors) in early iterations. Usually, more accurate metamodels (for both responses and prediction errors) could be obtained through iterations, when we get more accurate information with more data and validation points.

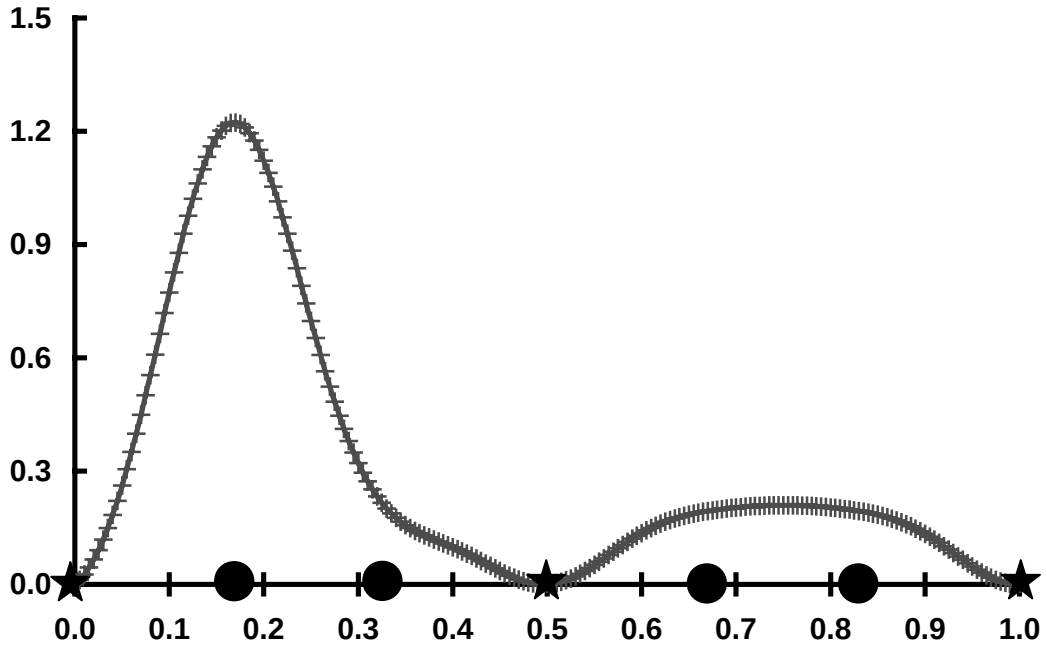


Figure 4.9 Metamodel of Prediction Errors in the 1st Iteration

Iteration I – Step 5: Metamodel Validation. Since in this study we do not use the accuracy of the metamodel as the stopping criterion, we will not check the accuracy of the metamodel before finishing designing sequential experiments. This step is then skipped here.

Iteration I – Step 6: Formulation of the Adjusted Covariance Matrix. To get more accurate metamodels, we decide to add in n_{new} = 1 data points. As mentioned in *Iteration I – Step 4*, the maximum absolute predicted prediction error of the metamodel developed in *Iteration I – Step 2* is about 29; this will be used to calculate entries of the adjusted covariance matrix.

In this step, a 4×4 covariance matrix is first built with Equations (4.19) and (4.20) – holding the stationary assumption; this is done with the same FORTRAN program as mentioned in *Iteration I – Step 1*. The first 3 rows and columns of the covariance matrix correspond to the 3 data points as identified in *Iteration I – Step 1*, and the last row and column correspond to the candidate point. Then another FORTRAN program is used to predict prediction errors, e_i, at the candidate point using the kriging metamodel developed in *Iteration I – Step 4*. These prediction errors are used to calculate correcting coefficients following Equation (4.25), and the correcting coefficients are used to adjust the covariance matrix following Equations (4.27) and (4.28). This is done in another FORTRAN program. These FORTRAN programs are linked in iSIGHT.

Iteration I – Step 7: Identification of New Data Points. In this step, by maximizing the determinant of the adjusted covariance matrix as developed in *Iteration I – Step 6*, the new data point is identified as x = 0.180. Since the new data point, x = 0.180, is very close to one of the validation points, x = 0.167, we decide to use x = 0.167 as the new data point; this avoids clustering of data/validation points and ensures great efficiency in the experimentation. To decide whether a candidate point is too close to an existing point, we need to compare their distance with that between evenly allocated

points. In this case, there are totally 8 points (4 data points plus 4 validation points) in a one-dimension design space, thus the average minimum distance between evenly allocated points should be around 0.143. We can use 10% of this distance as a standard value in judging whether two points are too close or not; in cases where high nonlinearity exists, this value may be smaller and in cases where the expected response surface is flat, this value should be larger. In this case, the smallest distance between the candidate point and existing points is 0.013, which is much smaller than 10% of 0.143, i.e., 0.0143. Future research may be needed in determining whether two points are too close or not.

Iteration I – Step 8: Updated Metamodel of Responses. Now we have 4 data points, as listed in Table 4.6. A new kriging metamodel is developed with information from these 4 data points. We got θ as 99.99233. The kriging model is as illustrated in Figure 4.10.

In Figure 4.10, we see that the new kriging metamodel is more accurate than the initial metamodel as illustrated in Figure 4.8. However, the new kriging model still does not catch the details of the actual responses at low x values (as illustrated in Figure 4.5). Following the flowchart in Figure 4.4, we go to Step 3 of the 2nd iteration.

Table 4.6 Four Data Points

x	0.0	0.167	0.5	1.0
y	0.618	-0.991	0.0	0.0

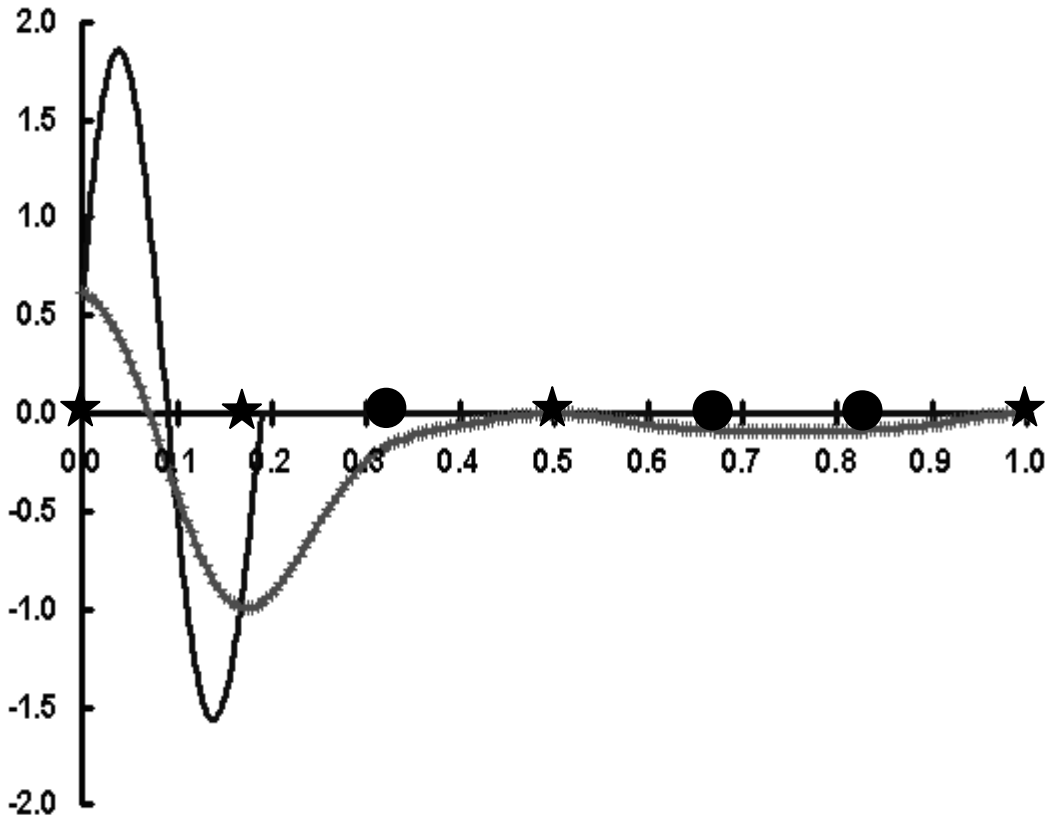


Figure 4.10 Kriging Metamodel with 4 Data Points

Iteration II – Step 3: Identification of New Validation Points. We need to select 2 more validation points in order to have 9 observed points after this step. Following the method described in Section IV, a metamodel of response is developed with 3 validation points (as illustrated in Figure 4.11), and prediction errors of this particular metamodel are observed at 4 data points. A metamodel of prediction errors is then developed and illustrated in Figure 4.12. Note that in Figure 4.11 and Figure 4.12 stars represent data points (in order to bring in most informative new validation points, the data points are used as validation points in *Iteration II – Step 3*), and solid dots

represent validation points (in this step, these validation points are used to develop a metamodel of response to help identify most informative new validation points).

A 9×9 covariance matrix is then formulated, with the first 3 rows and columns corresponding to the validation points, the 4th to 7th rows and columns corresponding to data points, and the last 2 rows and columns corresponding to new validation points. Following the same method as used in *Iteration I – Step 6* and *Step 7*, the covariance matrix is adjusted and new validation points are identified, at x = 0.122, and x = 0.235, as listed in Table 4.7.

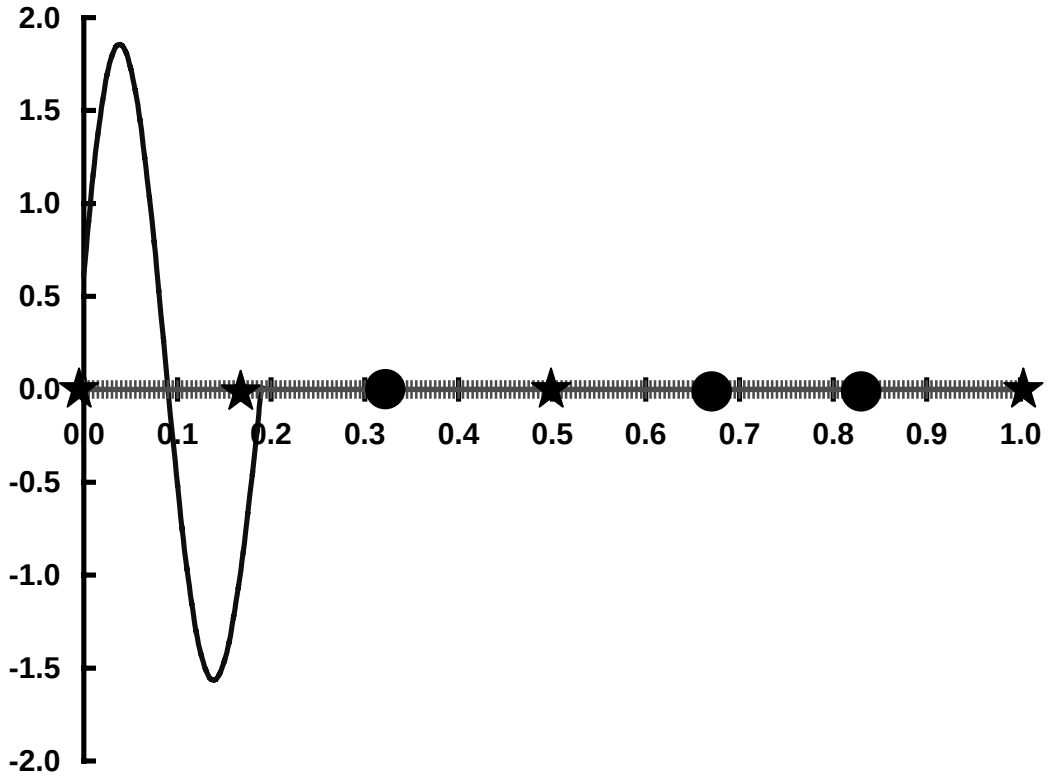


Figure 4.11 Metamodel Developed with 3 Validation Points in Iteration II – Step 3

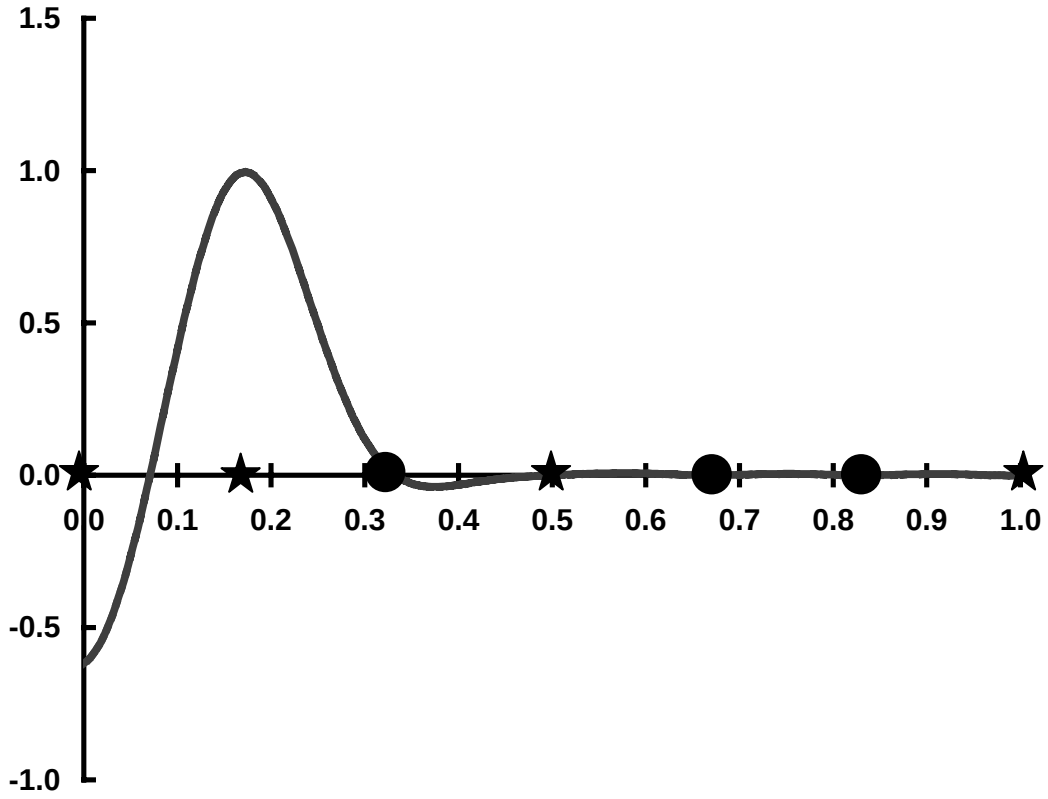


Figure 4.12 Metamodel of Prediction Errors Calculated in Iteration II – Step 3

Table 4.7 New Validation Point Added in the 2nd Iteration

x	0.122	0.235
y	-1.357	0.0

Iteration II – Step 4: Metamodel of Prediction Errors. Prediction errors at 5 validation points are listed in Table 4.8 and illustrated in Figure 4.13 by solid circles. As

shown before, prediction errors at 4 data points are all zero. A kriging metamodel of prediction errors is built with information from these 9 points, and illustrated in Figure 4.13. The maximum absolute predicted prediction error is $e_{max} \approx 0.8$.

Table 4.8 Prediction Errors at 5 Validation Points

$\boldsymbol{x}$	0.122	0.235	0.333	0.667	0.833
$\boldsymbol{y}_{pred}$	-0.691	-0.684	-0.145	-0.085	-0.085
$\boldsymbol{y}_{actual}$	-1.357	0.0	0.0	0.0	0.0
$\boldsymbol{y}_{error}$	0.666	-0.684	-0.145	-0.085	-0.085

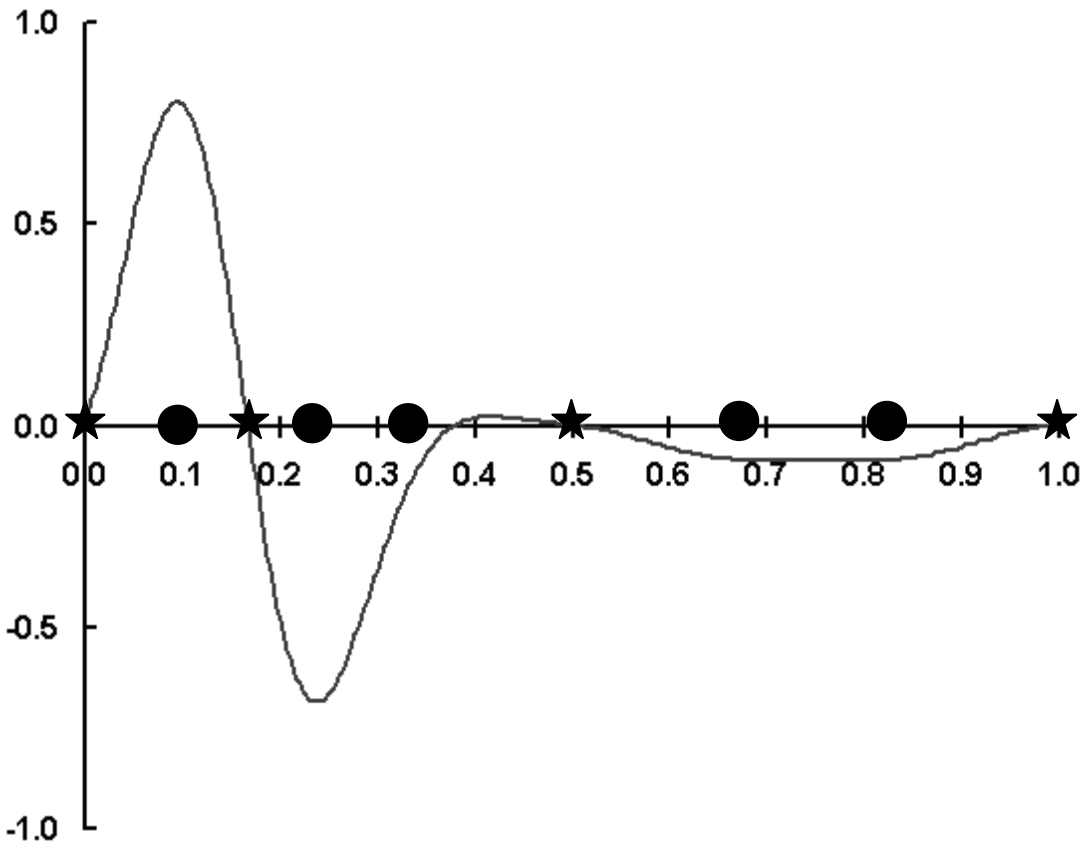


Figure 4.13 Metamodel of Prediction Errors with 5 Validation Points

Iteration II – Step 5: Metamodel Validation. As explained in *Iteration I – Step 5*, this step is skipped because the accuracy of metamodels is not used as the stopping criterion in SEED with the single-variable example.

Iteration II – Step 6: Formulation of the Adjusted Covariance Matrix. In this iteration we will add in n_{new} = 1 new data point. The maximum absolute predicted prediction error is about 0.8 with the current metamodel (by finding out the maximum absolute value of the metamodel developed in *Iteration II – Step 4*). To formulate the adjusted covariance matrix we follow similar method to that in *Iteration I – Step* 6. A 5×5 correlation matrix is developed, with the first 4 rows and columns corresponding to the 4 data points we already had, and the rest corresponding to the candidate point. The value of λ in Equation (4.28) is set to be 2.

Iteration II – Step 7: Identification of New Data Points. By maximizing the determinant of the adjusted correlation matrix as built in *Iteration II – Step 6*, we are able to identify the possible new data point as x = 0.75. Since the possible new data points, x = 0.75, is not very close to any of the validation points, we take it as the new data point. All 5 data points are listed in Table 4.9.

Table 4.9 Five Data Points

x	0.0	0.167	0.5	1.0	0.75
y	0.618	-0.991	0	0	0

Iteration II – Step 8: Updated Metamodel of Responses. A new kriging metamodel is developed with information from the 5 data points as listed in Table 4.9. We got θ as 99.99987. The kriging model is illustrated in Figure 4.14.

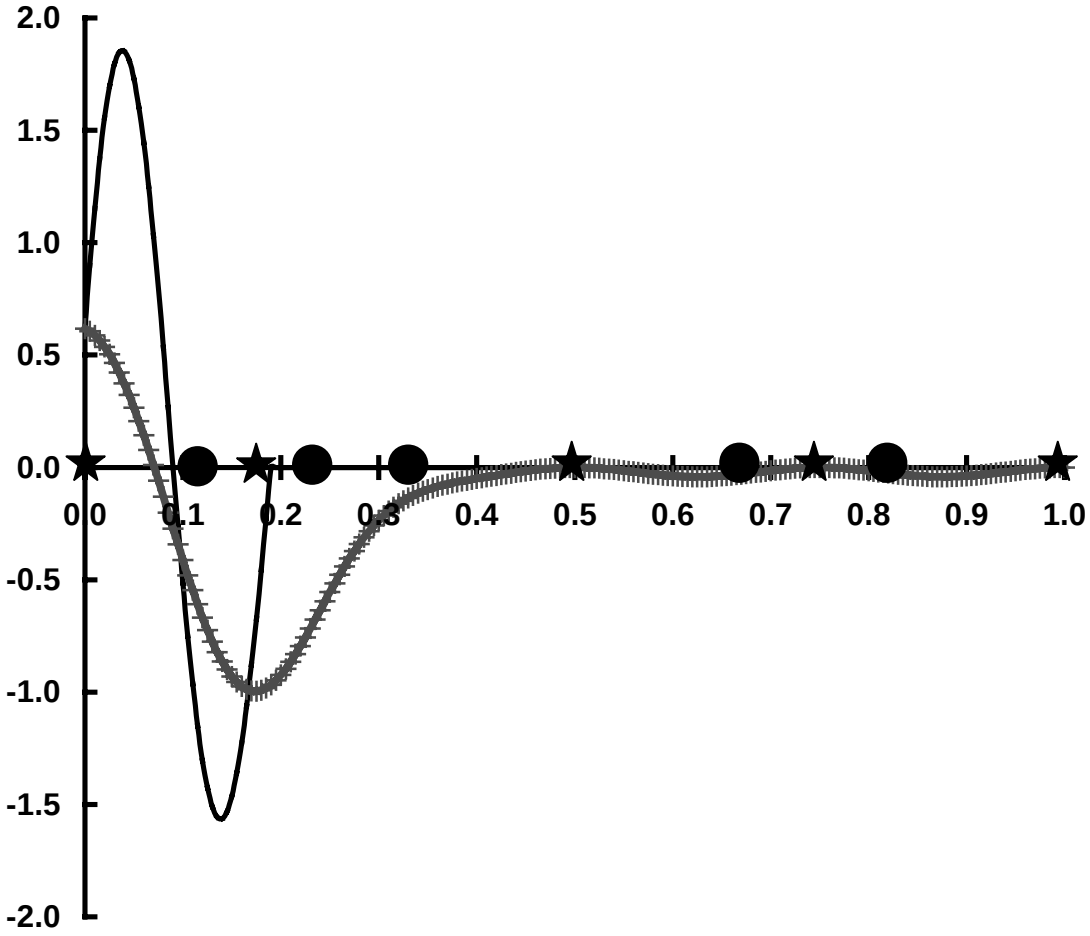


Figure 4.14 Metamodel of Responses with 5 Data Points

Iteration III – Step 3: Identification of New Validation Points. Now we have 5 data points and 5 validation points. Because we have a limit on the total number of points observed (11 points), we can only add in one more point in Iteration III. Following

similar approach as in *Iteration II – Step 3*, in this step we first develop metamodel of responses with 5 validation points (as illustrated in Figure 4.15), then prediction errors at 5 data points are observed and a metamodel of prediction errors is built (as illustrated in Figure 4.16).

An 11×11 covariance matrix is then formulated, with the first 5 rows and columns corresponding to the validation points, the 6^{th} to 10^{th} rows and columns corresponding to data points, and the last row and column corresponding to the new validation point. Following the same method as used in *Iteration I – Step 6* and *Step 7*, the covariance matrix is adjusted and new validation points are identified, at $x = 0.047$.

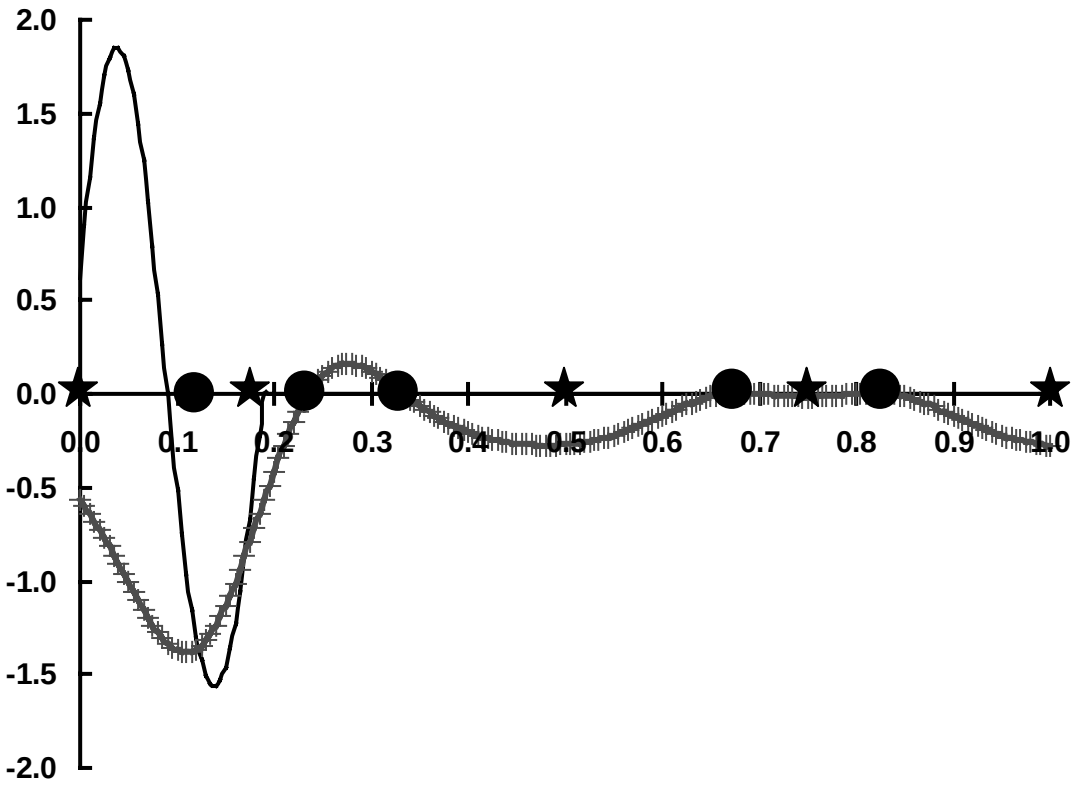


Figure 4.15 Metamodel Developed with 5 Validation Points in Iteration III – Step 3

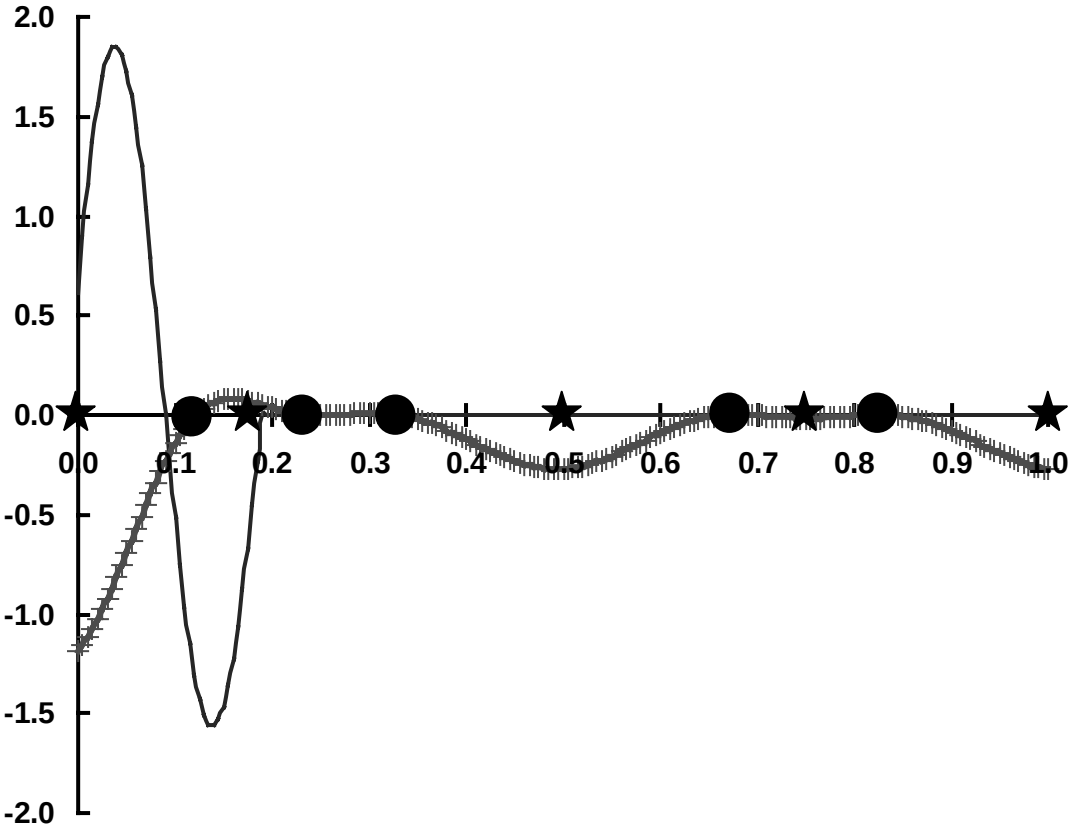


Figure 4.16 Metamodel of Prediction Errors Calculated in Iteration III – Step 3

Iteration III – Step 4: Metamodel of Prediction Errors. Prediction errors at 6 validation points are listed in Table 4.10 and illustrated in Figure 4.17 by solid circles. As shown before, prediction errors at 5 data points are all zero. A metamodel of prediction errors is built with information from these 11 points, and illustrated in Figure 4.17. The value of θ for this kriging metamodel is 99.99995. The maximum absolute predicted prediction error is $e_{max} \approx 1.5$.

Table 4.10 Prediction Errors at 6 Validation Points

x	0.122	0.235	0.333	0.667	0.833	0.047
y_{pred}	-0.691	-0.684	-0.145	-0.085	-0.085	0.3
y_{actual}	-1.357	0	0	0	0	1.784
y_{error}	0.666	-0.684	-0.145	-0.085	-0.085	-1.484

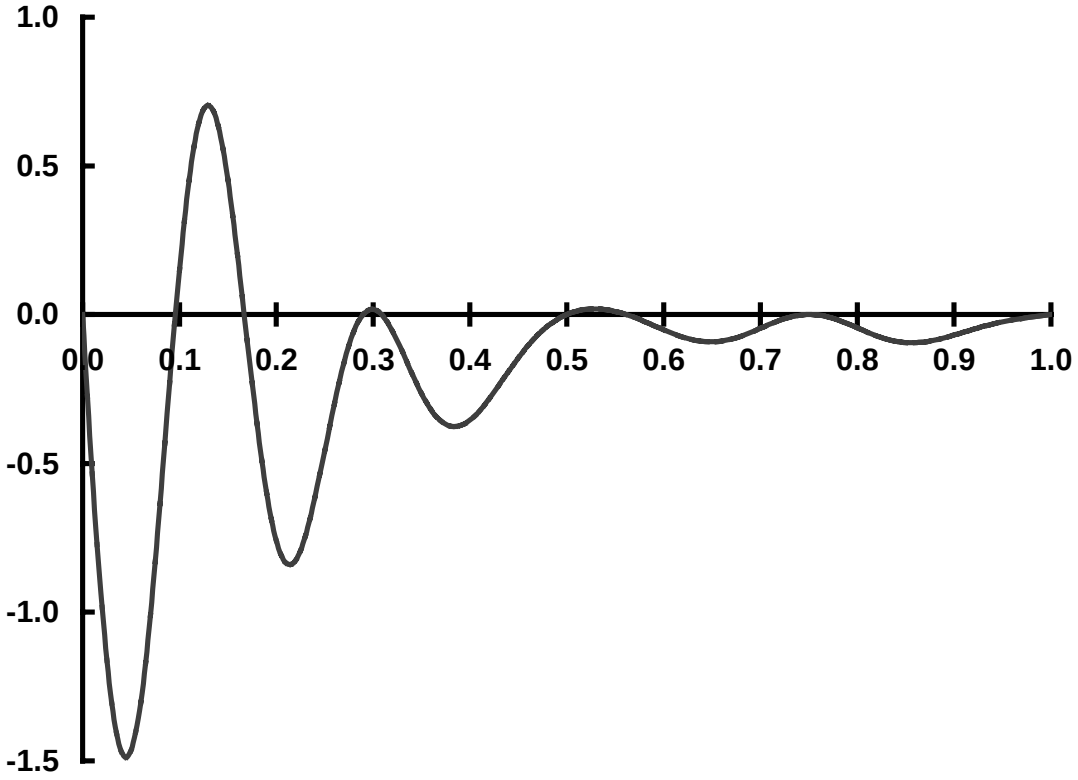


Figure 4.17 Metamodel of Prediction Errors with 6 Validation Points

Iteration III – Step 5: Metamodel Validation. In this example, the SEED method finally stopped in Iteration III when 11 points are observed. Data and validation points are listed in Table 4.11. As stated at the beginning of this section, the final metamodel is developed with all 11 observed points. We name this set of points as Data Set III. The corresponding metamodel is illustrated in Figure 4.18.

Table 4.11 Points Obtained with SEED (Formulation I) – Data Set III

Data Points	***x***	0.0	0.5	1.0	0.167	0.75	
	y	0.618	0.0	0.0	-0.991	0.0	
Validation Points	***x***	0.333	0.667	0.833	0.122	0.235	0.047
	y	0.0	0.0	0.0	-1.357	0.0	1.784

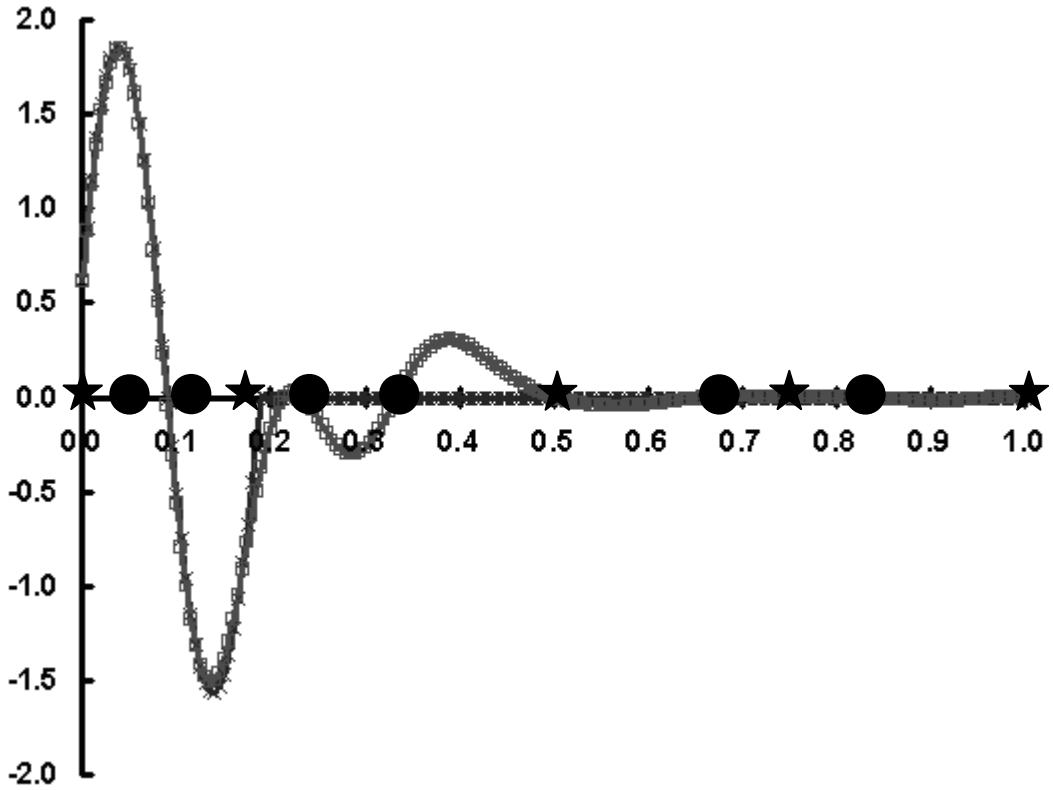


Figure 4.18 Metamodel of Responses with 11 Points (SEED Formulation I)

After validating this metamodel with 201 validation points that evenly spread over the design space, we got the maximum absolute prediction error of this metamodel is MAX = 0.371, and the root mean squared error is RMSE = 0.113. Comparison and discussion of the result in this section with that from single-stage experiments (Section

4.6.1) will be done after we demonstrate and study the application of Formulation II of the SEED method with the single-variable example in Section 4.6.3.

4.6.3 Application of SEED in the Single-Variable Example – Formulation II

In this section, the SEED method with Formulation I (as described in Section 4.5.3.2, Equations (4.27) and (4.34)) is applied to facilitate the development of an acceptable kriging metamodel for the single-variable function as introduced in Section 4.6.1. In this design of sequential experiments, similar to what have done in Section 4.6.2, we plan to identify 3 data points and 4 validation points first; 4 more points will be added in following iterations until eventually we get 11 observed points. Stopping criteria on metamodel accuracy will not be used in this example, thus no metamodel validation is done in Step 5 in SEED.

Iteration I – Step 1: Initial Experimental Design. In this step, we use the same approach as in Section 4.6.2 to design the initial experiments. Since all conditions are the same, we got the same set of data points in the initial experimental design as listed in Table 4.4.

Iteration I – Step 2: Simulation and Initial Metamodel of Responses. Since the initial experiments are the same as that in Section 4.6.2, the initial metamodel is also the same as that illustrated in Figure 4.8.

Iteration I – Step 3: Identification of New Validation Points. In this step, since the information from current data points and metamodel is the same as that in Section

4.6.2, new validation points should be the same as that in *Iteration I – Step 3* in Section 4.6.2. This set of validation points is listed in Table 4.5.

Iteration I – Step 4: Metamodel of Prediction Errors. In this step, a metamodel of prediction errors is developed with information from 3 data points and 4 validation points. The metamodel of prediction errors is the same as that developed in *Iteration I – Step 4* in Section 4.6.2.

Iteration I – Step 5: Metamodel Validation. Similar to our strategy in Section 4.6.2, the step of metamodel validation is skipped in the study of the single-variable example.

Iteration I – Step 6: Formulation of the Adjusted Covariance Matrix. In this iteration we decide to add in $n_{new} = 1$ new data point. In this step, entries of the adjusted covariance matrix are calculated following Formulation II of the SEED method as described in Section 4.5.3.2. The key equations here are Equations (4.27) and (4.34), which are different from those used in Section 4.6.2 (Equations (4.27) and (4.28)).

In this step, a 4×4 covariance matrix is first built with Equations (4.19) and (4.20) – holding the stationary assumption; this is done with the same FORTRAN program as used in *Iteration I – Step 6* in Section 4.6.2. The first 3 rows and columns of the covariance matrix correspond to the 3 data points as identified in *Iteration I – Step 1*, and the last row and column correspond to the candidate point. Then another FORTRAN program is used to predict prediction errors, e_i, at the two candidate points using the kriging metamodel developed in *Iteration I – Step 4*. These prediction errors are used to calculate correcting coefficients following Equation (4.32), and the correcting coefficients

are used to adjust the covariance matrix following Equations (4.27) and (4.34). This is done in another FORTRAN program. These FORTRAN programs are linked in iSIGHT. For details, see Appendix A.

Iteration I – Step 7: Identification of New Data Points. By maximizing the determinant of the covariance matrix developed in *Iteration I – Step 6*, we are able to identify two possible new data points as $x = 0.17$.

Note that the possible new data point, $x = 0.17$, is very close to one of the validation points, $x = 0.167$, thus we decide to use $x = 0.167$ as the new data point to avoid clustering. Four data points are listed in Table 4.12.

Table 4.12 Four Data Points Identified in the 1st Iteration

x	0.0	0.5	1.0	0.167
y	0.618	0.0	0.0	-0.991

Iteration I – Step 8: Updated Metamodel of Responses. Now we have 4 data points as listed in Table 4.12. In this step, a new metamodel of responses is developed based on the information from Table 4.12. The value of θ for this metamodel is 99.99964. The new kriging metamodel is illustrated in Figure 4.19. Since we have only 7 observed points in this iteration, we will go to the next iteration for more points.

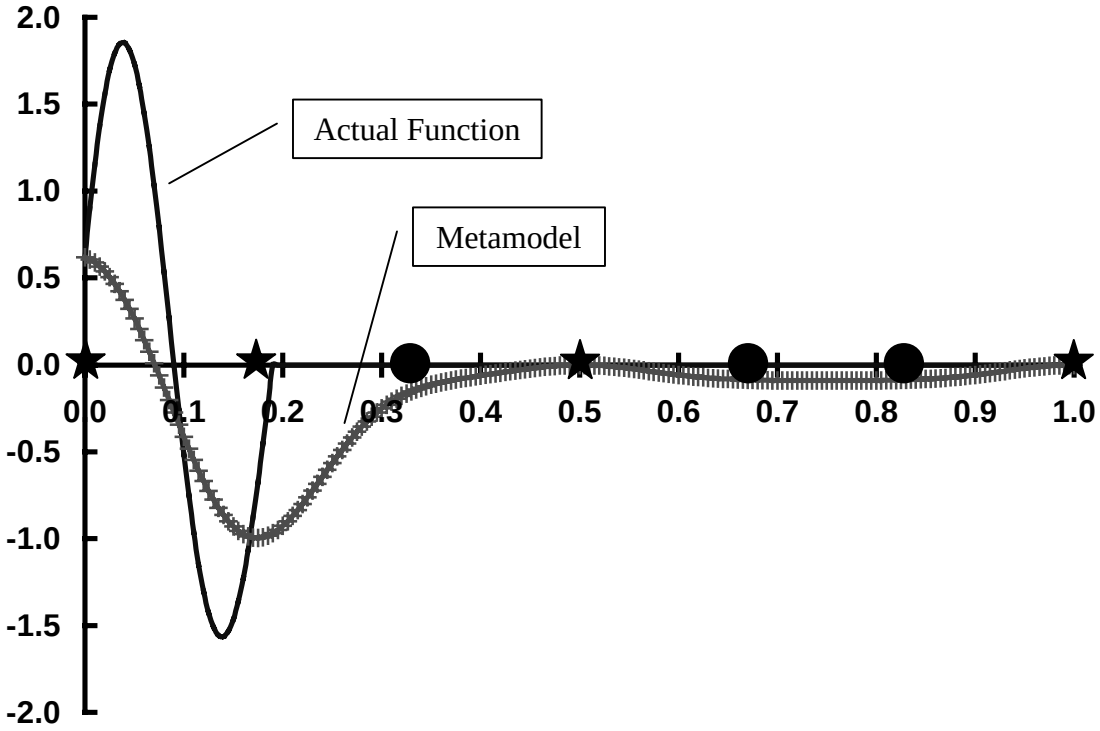


Figure 4.19 Metamodel of Responses with 4 Data Points in the 1st Iteration

Iteration II – Step 3: Identification of New Validation Points. Now we have 4 data points and 3 validation points. In this step, we need to identify 2 new validation points to ensure that we have one more validation points than data points. Following the method described in Section IV, a metamodel of response is developed with 4 validation points (as illustrated in Figure 4.20), and prediction errors of this particular metamodel are observed at 4 data points. A metamodel of prediction errors is then developed and illustrated in Figure 4.21. Note that in Figure 4.20 and Figure 4.21 stars represent data points (in order to bring in most informative new validation points, data points are used

as validation points in this step), and solid dots represent validation points (in this step, these validation points are used to develop a metamodel of response to help identify most informative new validation points).

A 9×9 covariance matrix is then formulated, with the first 4 rows and columns corresponding to the validation points, the 5th to 7th rows and columns corresponding to data points, and the last 2 rows and columns corresponding to the new validation points. Following the same method as used in *Iteration I – Step 6* and *Step 7*, the covariance matrix is adjusted and new validation points are identified as listed in Table 4.13.

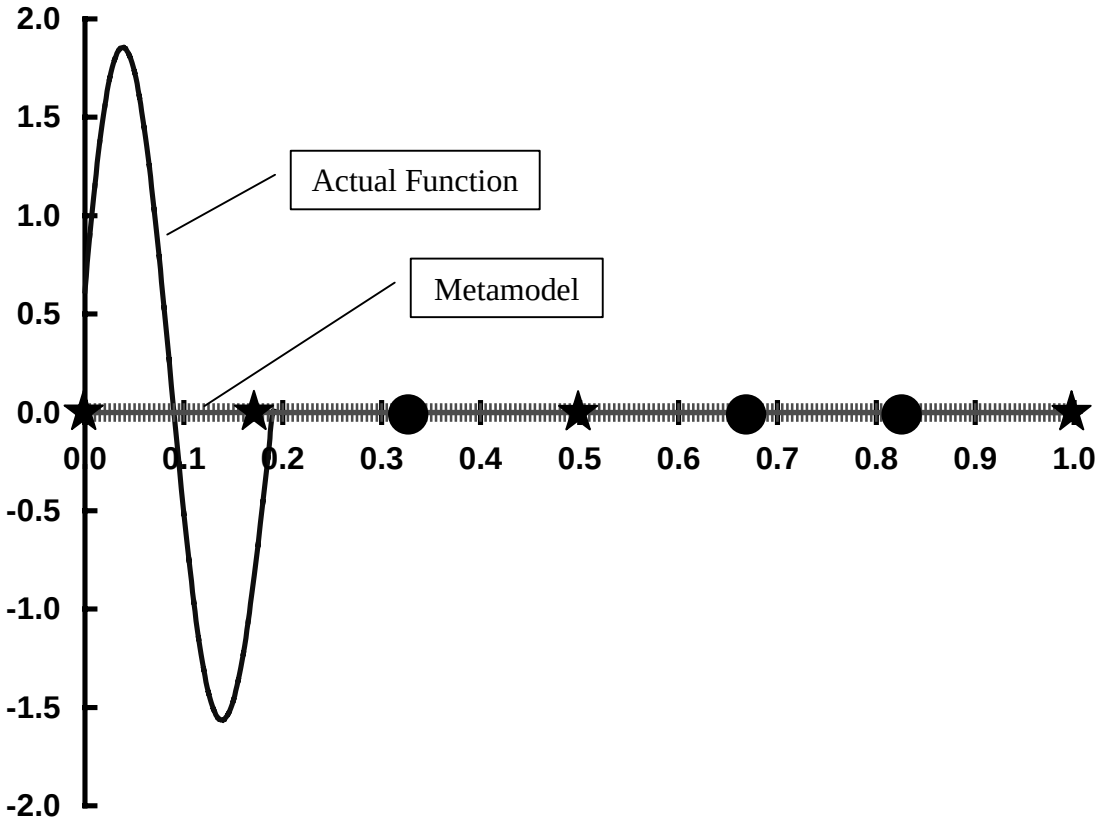


Figure 4.20 Metamodel Developed with 3 Validation Points in Iteration II – Step 3

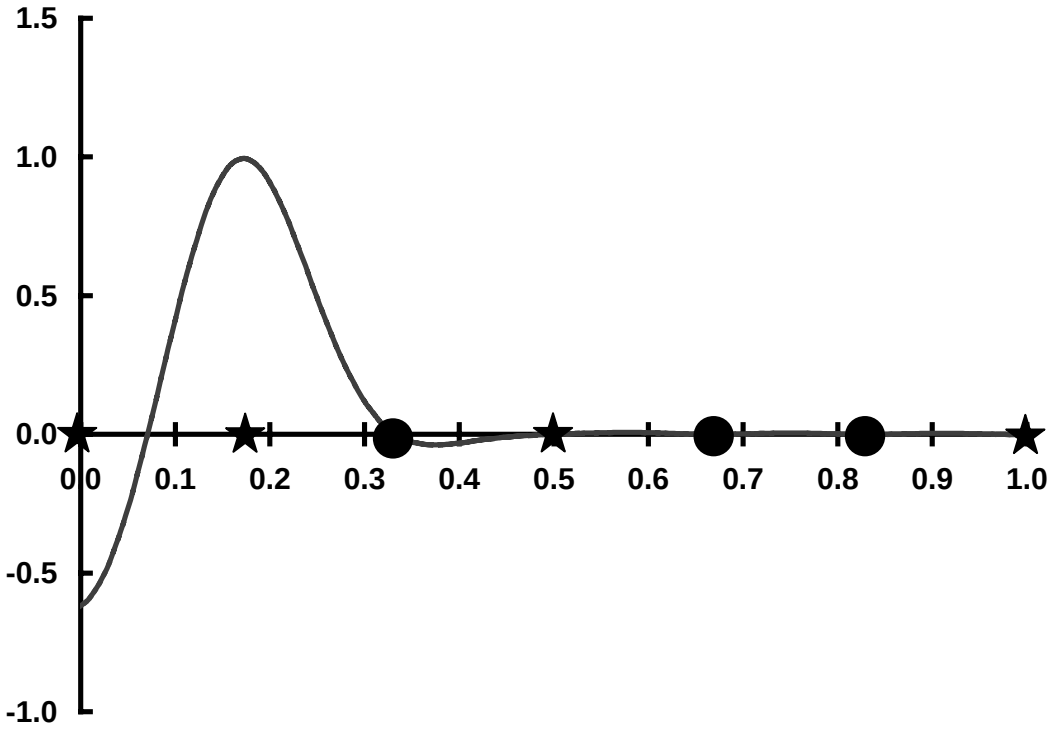


Figure 4.21 Metamodel of Prediction Errors Calculated in Iteration II – Step 3

Table 4.13 New Validation Point Added in the 2nd Iteration

x	0.157	0.252
y	-1.310	0.0

Iteration II – Step 4: Metamodel of Prediction Errors. In this step, a metamodel of prediction errors is developed based on information of prediction errors at 4 data points (all are zero's) and 5 validation points. Prediction errors at validation points

are listed in Table 4.14. The metamodel of prediction errors is illustrated in Figure 4.22. The value of θ for this metamodel is 99.99997. The maximum absolute predicted prediction error is $e_{max} \approx 1.87$.

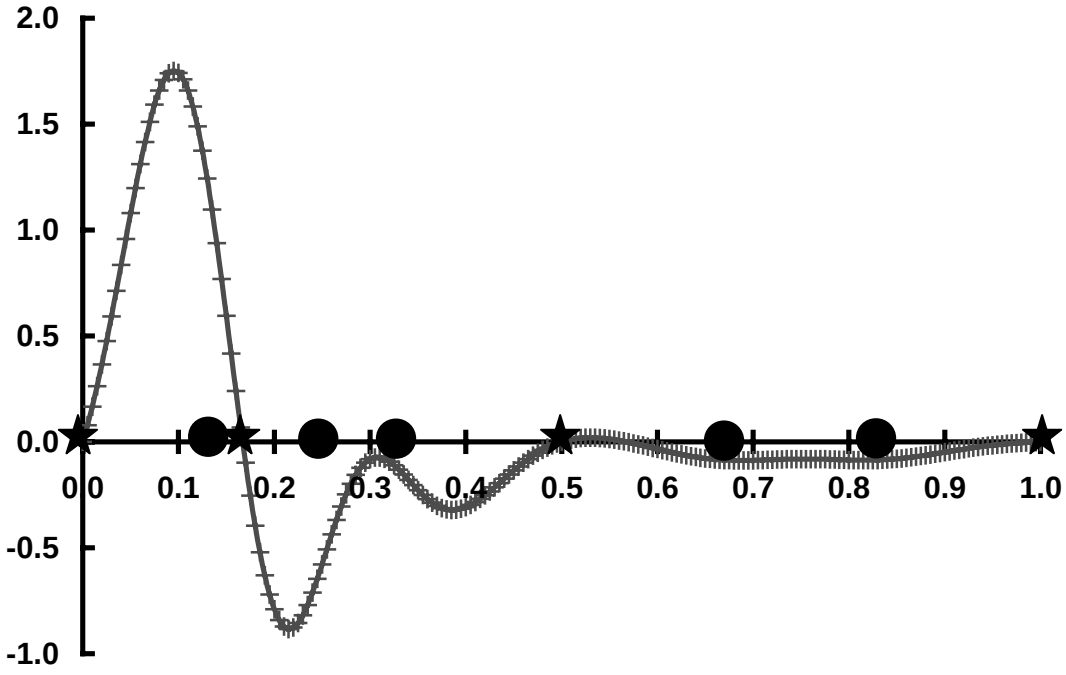


Figure 4.22 Metamodel of Prediction Errors in the 2nd Iteration

Table 4.14 Prediction Errors at 5 Validation Points in the 2nd Iteration

$\boldsymbol{x}$	0.157	0.252	0.333	0.667	0.833
$\boldsymbol{y}_{pred}$	-0.964	-0.55	-0.145	-0.085	-0.085
$\boldsymbol{y}_{actual}$	-1.31	0	0	0	0
$\boldsymbol{y}_{error}$	0.346	-0.55	-0.145	-0.085	-0.085

Iteration II – Step 5: Metamodel Validation. This step is skipped.

Iteration II – Step 6: Formulation of the Adjusted Covariance Matrix. In this iteration we decide to add in $n_{new} = 1$ new data point. In this step, entries of the adjusted covariance matrix are calculated following Formulation II of the SEED method as described in Section 4.5.3.2. The key equations here are Equations (4.27) and (4.34).

Iteration II – Step 7: Identification of New Data Points. By maximizing the determinant of the covariance matrix developed in *Iteration II – Step 7*, we are able to identify the new data point as $x = 0.758$.

Iteration II – Step 8: Updated Metamodel of Responses. Now we have 5 data points, as listed in Table 4.15. A new kriging metamodel is developed with this information and illustrated in Figure 4.23. The value of θ for this metamodel is 99.99987.

Table 4.15 Five Data Points Used in the 2nd Iteration

x	0.0	0.5	1.0	0.167	0.758
y	0.618	0.0	0.0	-0.991	0.0

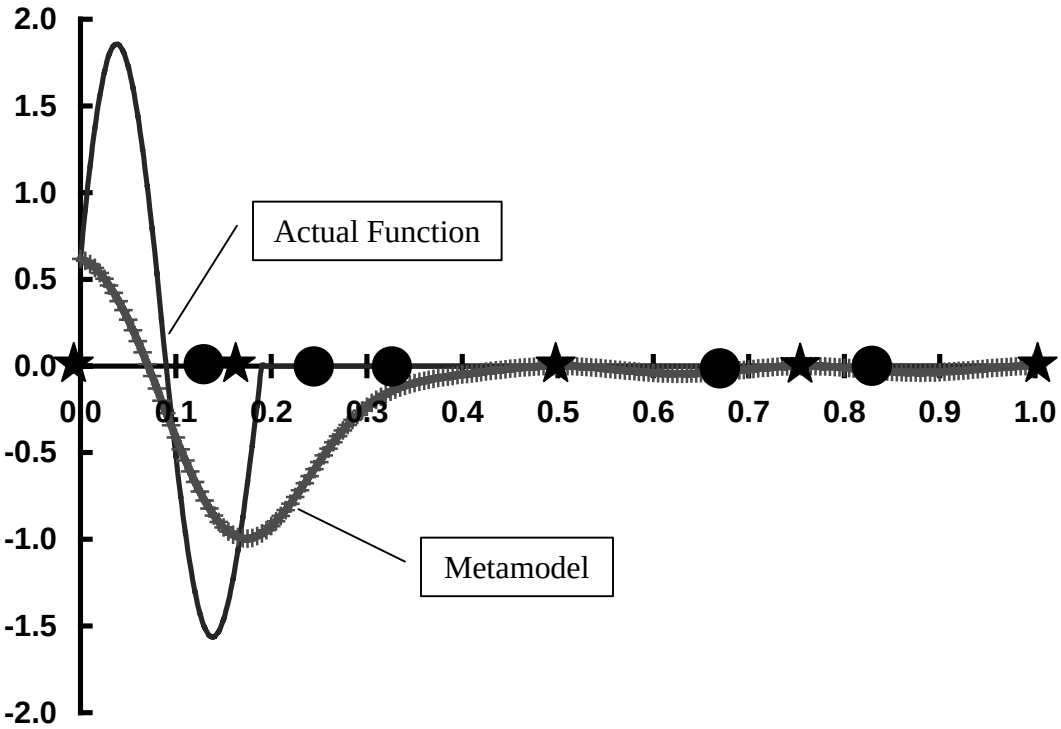


Figure 4.23 Metamodel of Responses with 5 Data Points

Iteration III – Step 3: Identification of New Validation Points. Note that we do not use the accuracy of metamodels as the stopping criterion in this example. Since we have got 10 observed points, we will only add in one more point in this iteration. Following similar approach as in *Iteration II – Step 3*, in this step we first develop metamodel of responses with 5 validation points (as illustrated in Figure 4.24), then prediction errors at 5 data points are observed and a metamodel of prediction errors is built (as illustrated in Figure 4.16).

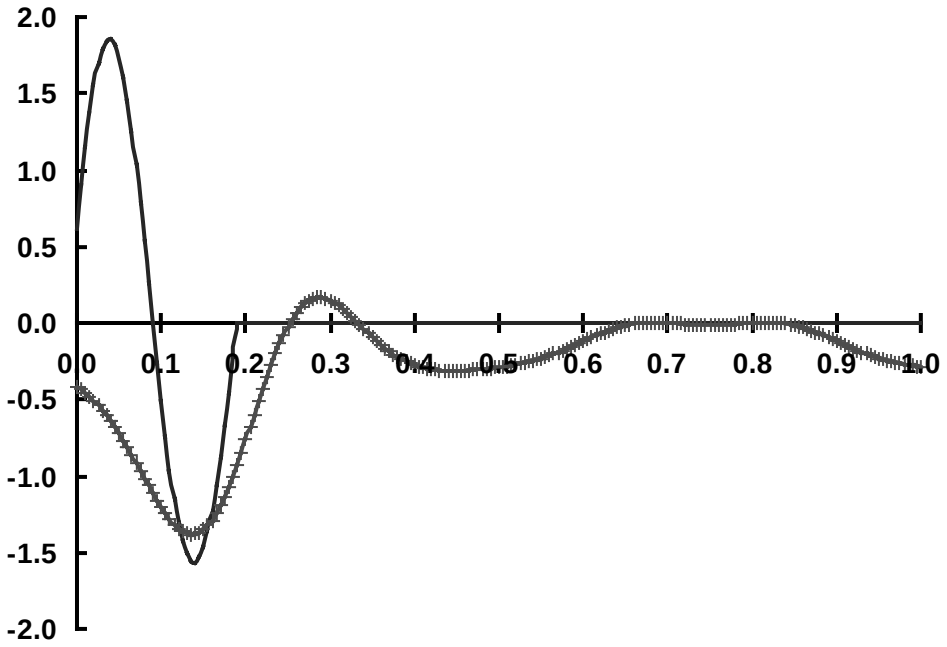


Figure 4.24 Metamodel of Responses Developed in Iteration III – Step 3

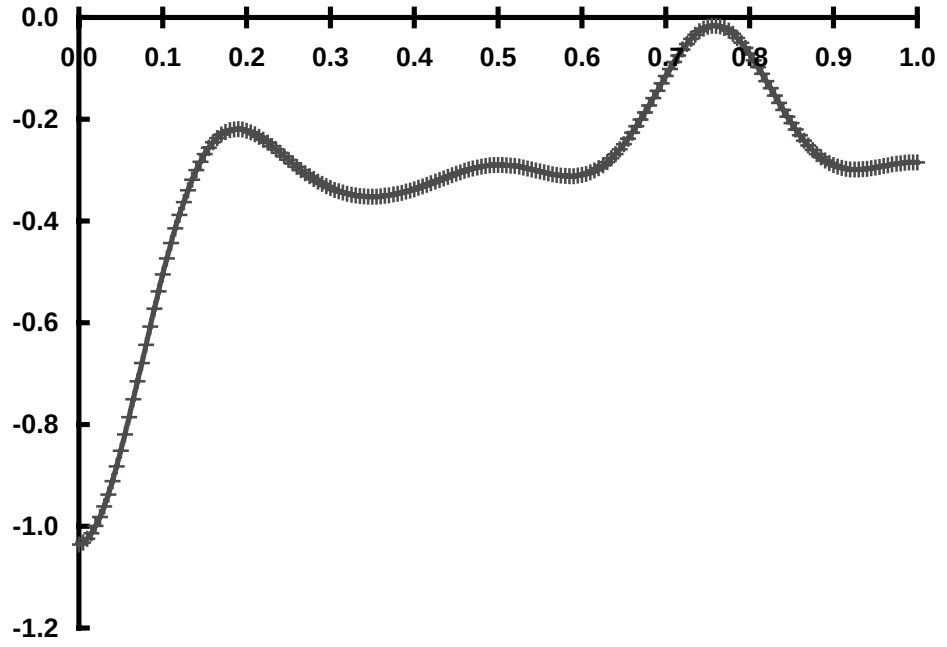


Figure 4.25 Metamodel of Prediction Errors Developed in Iteration III – Step 3

An 11×11 covariance matrix is then formulated, with the first 5 rows and columns corresponding to the validation points, the 6th to 10th rows and columns corresponding to data points, and the last row and column corresponding to the new validation point. Following the same method as used in *Iteration I – Step 6* and *Step 7*, the covariance matrix is adjusted and new validation points are identified, at $x = 0.045$.

Table 4.16 Prediction Errors at Five Validation Points

$\boldsymbol{x}$	0.045	0.157	0.252	0.333	0.667	0.833
$\boldsymbol{y}_{pred}$	0.322	-0.964	-0.539	-0.129	-0.036	-0.027
$\boldsymbol{y}_{actual}$	1.812	-1.31	0	0	0	0
$\boldsymbol{y}_{error}$	-1.49	0.346	-0.539	-0.129	-0.036	-0.027

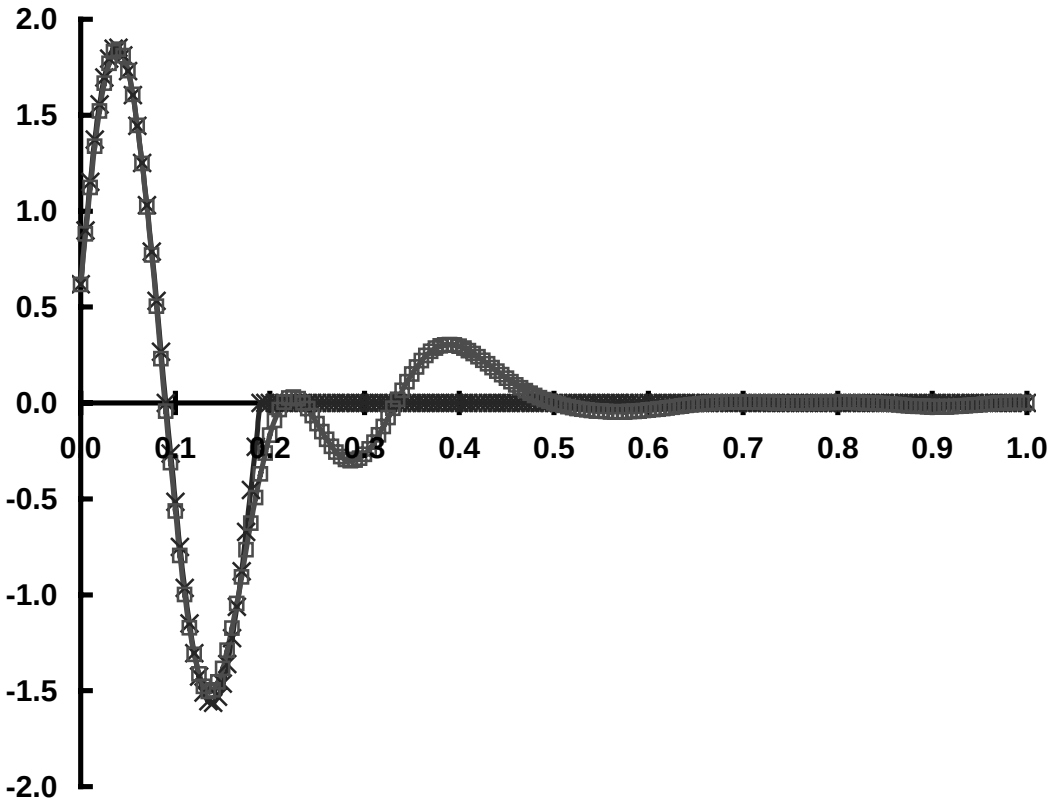


Figure 4.26 Metamodel of Responses with 11 Points (SEED Formulation II)

Since we already got 11 points, the SEED process stopped in this iteration. Prediction errors at 6 validation points are presented in Table 4.16. Same as what we did in Section 4.6.2, a final kriging metamodel is developed based on information at 11 observed points, and illustrated in Figure 4.26. The value of θ for this metamodel is 100. After validating this metamodel with 201 validation points that evenly spread over the design space, we got the maximum absolute prediction error of this metamodel is MAX = 0.622, and the root mean squared error is RMSE = 0.198.

In this section four approaches are used to design experiments and develop kriging metamodels for the single-variable example. In Section 4.6.1, two "single-stage" methods are studied, in one of which all the data points are identified in a single step (Metamodel (I), Figure 4.6), and in the other the data points are added in sequentially but without adjustment based on information from previous experiments (Metamodel (II), Figure 4.7). The SEED method with Formulation I is applied in Section 4.6.2 and the metamodel is illustrated in Figure 4.18. The SEED method with Formulation II is applied in Section 4.6.3 and the metamodel is illustrated in Figure 4.26. Based on studies in this section, the maximum absolute error (MAX) and root mean squared error (RMSE) of the kriging metamodels from different approaches are calculated with information from 201 validation points and listed in Table 4.17. For more details of the information shown in Table 4.17, see discussions in Sections 4.6.1, 4.6.2, and 4.6.3.

Table 4.17 Accuracy of Kriging Metamodels from Different Approaches

	Single-Stage Approach		SEED	
	Points Added at One Time	Points Added Sequentially	Formulation I	Formulation II
	Metamodel I	Metamodel II		
MAX	1.730	1.711	0.371	0.622
RMSE	0.452	0.472	0.113	0.198

In Table 4.17, we see clearly that with the SEED approach, no matter which formulation is used, values of MAX and RMSE of the metamodels are much smaller than those of Metamodels I and II with single-stage approaches. This shows that metamodels with the SEED approach are more effective than those with single-stage approaches. With the SEED approach, data points are allocated in "crucial" regions where there are large expected prediction errors; the information brought in by each new data point is more than that in single-stage experiments. This could also be seen through comparison of Figure 4.6, Figure 4.7, Figure 4.18, and Figure 4.26.

We also observe that designers may meet problem in selecting points at some stages in the method by Currin and co-authors (Data Set II) because there may be two or more points that are equally good with their criteria. In Data Set II, if the number of observed points is not set to be 11 (for example, it could be set to be 12 or 13), designers will not be able to select the "better" set of data points in experimental design. Dilemma in design of experiments will be inevitable. Thus, with single-stage experimental design method, the achievement of accurate metamodels is not guaranteed; the result of experimental design is very sensitive to decisions made by designers in the metamodeling

process. Our study shows the SEED method is robust to decisions made by designers in the metamodeling process; the achievement of accurate metamodels is guaranteed.

The SEED method will be used in RCEM to facilitate efficient development of accurate metamodels for design space exploration. To be specific, the SEED method will replace Processors B, C, D, and E in RCEM, as illustrated in Figure 4.27. This is further discussed in Chapters 5 and 6.

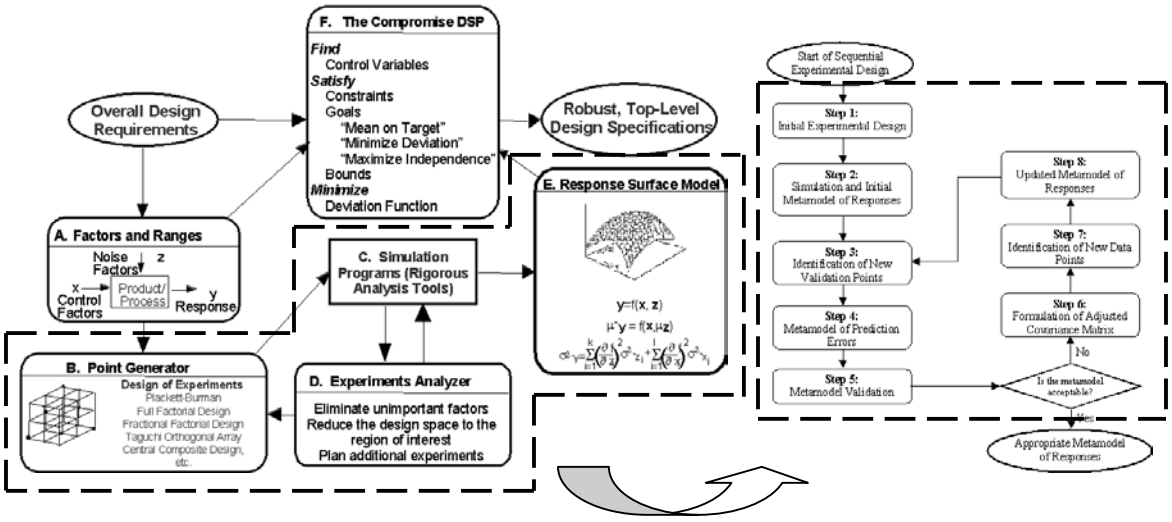


Figure 4.27 Application of SEED in RCEM

4.7 A LOOK BACK AND A LOOK AHEAD

The method of Sequential Exploratory Experimental Design (SEED) is developed in this chapter. The SEED method is demonstrated and verified with a single-variable example. Research in this chapter helps answer Research Question 2 and its sub-questions; the corresponding hypotheses are tested. Research Question 2, its sub-questions, and corresponding hypotheses are listed below.

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

Hypothesis 2: Sequential experiments could be designed through analysis of information from data/validation points and metamodels.

R.Q.2.1: *How to measure the information worth of a point?*

Sub-Hypothesis 2.1: The information worth of a point could be measured with entropy.

R.Q.2.2: *How to select validation points to achieve a sequential design of computer experiments?*

Sub-Hypothesis 2.2: Selection of validation points should follow similar rules for selection of data points; information from validation points could be used as guidance in identifying new data points.

R.Q.2.3: *How to utilize information from previous points and metamodels in identifying new data points?*

Sub-Hypothesis 2.3: Through maximizing entropy (as formulated based on Sub-Hypotheses 1.1 and 1.2) we are able to allocate new data points in the design space that yield maximum potential information.

To answer Research Question 2, the method of Sequential Exploratory Experimental Design (SEED) is developed based on *D*-optimal design and maximum entropy sampling. In this chapter, we verified that with the SEED method, designers are able to add in new data points with large amount of potential information, and thus

accurate metamodels could be achieved efficiently. Information from current data and validation points and metamodels are used as guidance in identifying new data points. Hypothesis 2 is verified; our answer to Research Question 2 is: *Accurate metamodels can be developed through iterations in sequential experimental design with the SEED method, in which information from current data/validation points and metamodels is used as guidance in identifying new data points.*

Research Question 2.1 is answered primarily in Sections 4.3 and 4.4. The application of Bayesian entropy design in SEED in Sections 4.5 and 4.6 supports our idea from Sections 4.3 and 4.4. A clear statement on Research Question 2.1 is presented at the beginning of Section 4.5. Sub-Hypothesis 2.1 is tested; our answer to Research Question 2.1 is: *The entropy criterion could be used to measure the information worth of a new point.*

Research Question 2.2 is studied in developing and verifying the SEED method in Sections 4.5 and 4.6; Sub-Hypothesis 2.2 is tested. The usage of validation points and observation of prediction errors are necessary steps in the SEED method; it provides the foundation for adjusting the covariance matrix, which is the core of the SEED method. In the SEED method, validation points are added sequentially in iterations; as more and more data and validation points are observed, designers are able to develop more and more accurate metamodels for responses and prediction errors. In Section 4.6, different strategies on selecting validation points are applied and studied in the SEED method. Our answer to Research Question 2.2 is: *Validation points should be added in iterations*

in sequential experimental design; information from validation points should be used as guidance in identifying future data points.

Research Question 2.3 is answered and Sub-Hypothesis 2.3 is tested in the development of the SEED method. To be specific, the method of maximum entropy sampling is introduced in Section 4.4; in Section 4.5.2, strategies on how to utilize information from previous points and metamodels are discussed; the mathematical formulations in SEED is developed in Section 4.5.3, which enables designers to design sequential experiments through maximizing entropy; Demonstration and verification is enclosed in Section 4.6. Our answer to Research Question 2.3 is: *Information from current data/validation points and metamodels could be used to build the adjusted covariance matrix; new data points could be identified through maximizing the determinant of the adjusted covariance matrix.*

Chapter 4 is the foundation of research in the following 2 chapters. In the next chapter, the SEED method is further developed and tested with different types of metamodels. In Chapter 5, as a support to the SEED method and the Efficient Robust Concept Exploration Method (to be developed in Chapter 6), research is done on comparison of different types of metamodels, sequential experimental design in irregular design spaces, and metamodel selection along the design timeline. In Chapter 6, ideas from the SEED method will be further developed and used in developing the Efficient Robust Concept Exploration Method (E-RCEM), in which the design process of metamodeling and design space exploration are integrated and efficient exploration of the design space is facilitated.

CHAPTER 5

SEQUENTIAL METAMODELING ALONG THE DESIGN TIMELINE

In Chapter 4, the method of Sequential Exploratory Experimental Design (SEED) is developed and studied with kriging metamodels and a very simple example. In this chapter, studies on SEED are extended with the application of other types of metamodels, i.e., Response Surface (RS) models and Multivariate Adaptive Regression Splines (MARS) models. In this chapter, first we will study the performance of kriging and univariate quintic regression spline (application of MARS in one-dimensional problems) metamodels in response surface prediction in Section 5.2. The application of SEED with MARS metamodels is described and studied in Section 5.3. Then the approach of sequential utilization of RS, kriging, and MARS metamodels along the design timeline is described in Section 5.4. This approach is illustrated with a simple engineering example in Section 5.5. A look back and a look forward are enclosed in Section 5.6. Research questions visited in this chapter are R.Q.2, R.Q.4, their sub- research questions, and R.Q.3.2.

5.1 WHAT IS PRESENTED IN THIS CHAPTER

In Chapter 3 we studied metamodel validation techniques to answer Research Question 1 in this dissertation. In Chapter 4, the method of Sequential Exploratory Experimental Design (SEED) is developed and studied to help answer Research Question 2. In this chapter, the application of SEED is extended with other types of metamodel, i.e., Response Surface (RS) models and Multivariate Adaptive Regression Splines (MARS) models.

Research Question 2, *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*, is revisited in this chapter with the utilization of MARS metamodels. This is specifically done in Section 5.3, where kriging models and MARS models are used together in the application of SEED with a simple example.

The comparison of kriging and regression spline (application of MARS in one-dimensional problems) metamodels is done in Section 5.2, which helps answer Research Question 4.1, *How do different types of metamodels perform in engineering design?*, and Research Question 4.2, *How to select different types of metamodels at different design stages?* Previous studies on RS metamodels and various types of kriging metamodels also contribute to answers to these research questions.

Based on studies in Sections 5.2 and 5.3, an approach is developed in Section 5.4, in which RS, kriging, and MARS metamodels are utilized together to help efficiently and effectively develop acceptable metamodels in engineering design. This approach is then

illustrated through the application with a simple engineering example in Section 5.5. This helps answer Research Question 4, *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?*

In Section 5.6 we revisit research questions and hypotheses discussed in this chapter. Studies in Chapters 3, 4, and 5 build the foundation for research in Chapter 6, in which the Efficient Robust Concept Exploration Method (E-RCEM) is developed and studied to facilitate efficient exploration of the design space for robust solutions.

5.2 A COMPARISON OF KRIGING AND MARS METAMODELS IN RESPONSE PREDICTION

Research questions to be studied in this section are R.Q. 4.1, *How do different types of metamodels perform in engineering design?* and R.Q. 4.2, *How to select different types of metamodels at different design stages?* The comparison of Response Surface (RS) metamodels and kriging metamodels with various types of correlation functions has been done in (Simpson, 1998) and (Lin, 2000), and will not be performed in this dissertation. In order to answer R.Q. 4.1 and R.Q. 4.2, in this section first we observe and analyze the performance of kriging and univariate quintic regression spline (application of MARS in one-dimensional problems) metamodels with space filling experiments in Section 5.2.1. Then in Section 5.2.2, the observation and analysis are extended to cases in which non-space-filling data points are used (which is typical in sequential experiments). The example used in this section is a single-variable function.

5.2.1 An Observation and Analysis on the Performance of Kriging and Univariate Regression Spline Metamodels in Response Prediction with Space-Filling Experiments

In this section, we observe and analyze the performance of kriging metamodels and regression splines in response prediction with space filling data points. The kriging metamodel is developed with the Gaussian correlation function as expressed in Equation (2.14) in Chapter 2. The regression splines are actually applications of MARS in one-dimensional problems. In building the regression splines, we use the implementation of MARS in (Chen, et al., 1999). For regression splines metamodels in this chapter, if not specifically pointed out, the number of maximum basis functions is set to be 50; the number of knots is set equal to the number of data points; the maximum number of splits is set to be 2, which is suitable for two-way interactions and apparently more than enough for the single-variable function. Details of the kriging and regression splines will be described later in this section.

In this study we use a single-variable function taken from (Farhang-Mehr and Azarm, 2002):

$$f(x) = (1 - e^{-2\sqrt{x}}) + 6xe^{-7x}\sin(10x) - 0.2e^{-2000(x-0.25)^2} \\ +60\min(0,|x-0.14|-0.08)^2[\ln(x+0.2)+1.5\sin^2(85x)] \tag{5.1}$$

In our study in this section, the design variable x is set to be within the design space of [0,1]. The actual response surface of Equation (5.1) with x = [0,1] is shown in Figure 5.1. The global minimum happens at x = 0 with y = 0; the global maximum happens around x = 0.165 with y = 0.953.

In Figure 5.1 we see that this single-variable function is very highly nonlinear in the design space of $x \in [0.15,0.35]$, and very flat when x is large. This single-variable function provides a very good platform with which we could compare the performance of kriging metamodels and regression splines. Since there is only one design variable, we could choose to have data points evenly spread over the whole design space of [0,1]. In order to observe how kriging and regression splines works with different number of data points, we choose three different sets of data points, one with 6 data points, another with 12, and the third with 18 data points. We use two software to develop the kriging metamodels; one is the computer program written by Simpson (see, Simpson 1998) and the other is commercial software named iSIGHT. The software used to develop regression splines is from (Chen, et al., 1999).

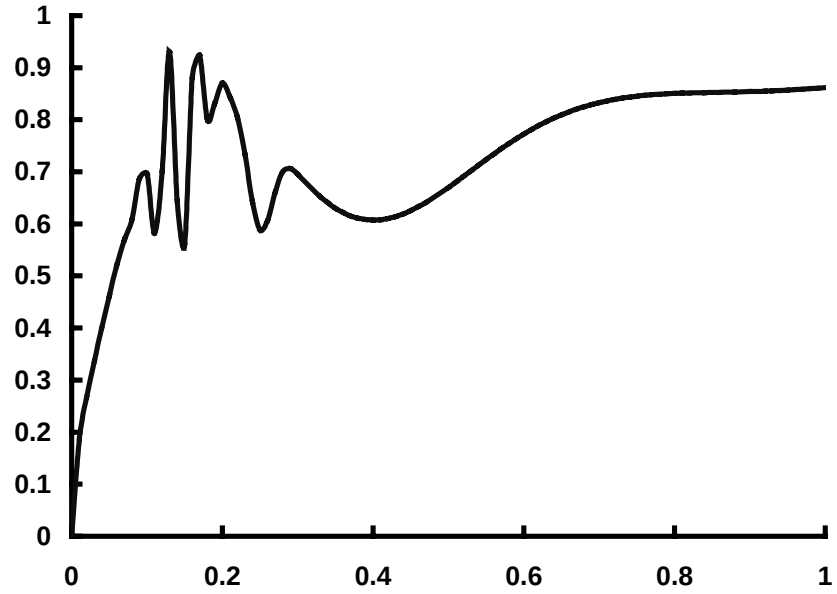


Figure 5.1 A Single-Variable Function

The set of 6 data points is listed in Table 5.1. The corresponding kriging and regression spline metamodels are illustrated in Figure 5.2 and Figure 5.3, respectively. The value of θ for the kriging metamodel is 99.9999. To build the regression spline metamodel, we use 6 knots in the x dimension, which is equal to the number of data points; Backwards deletion is used in this metamodeling. Results of regression splines approximation are saved in the file named qmars.dat, and the content is presented in Appendix B. In Figure 5.2 and Figure 5.3 we see that with 6 data points both kriging and regression spline metamodels performs well in grasping the response surface at regions with very small and large x values. Both of them do not catch the high nonlinearity in $x \in [0.15,0.35]$ since we do not have enough information in this region. The regression spline metamodel is a little superior to the kriging metamodel as we see that there are two "waves" around $x = 0.7$ and 0.9 in Figure 5.2, the kriging metamodel.

Table 5.1 Data Point Set I – 6 Points

x	0.0	0.2	0.4	0.6	0.8	1.0
y	0.0	0.87017	0.60729	0.88250	0.85041	0.86169

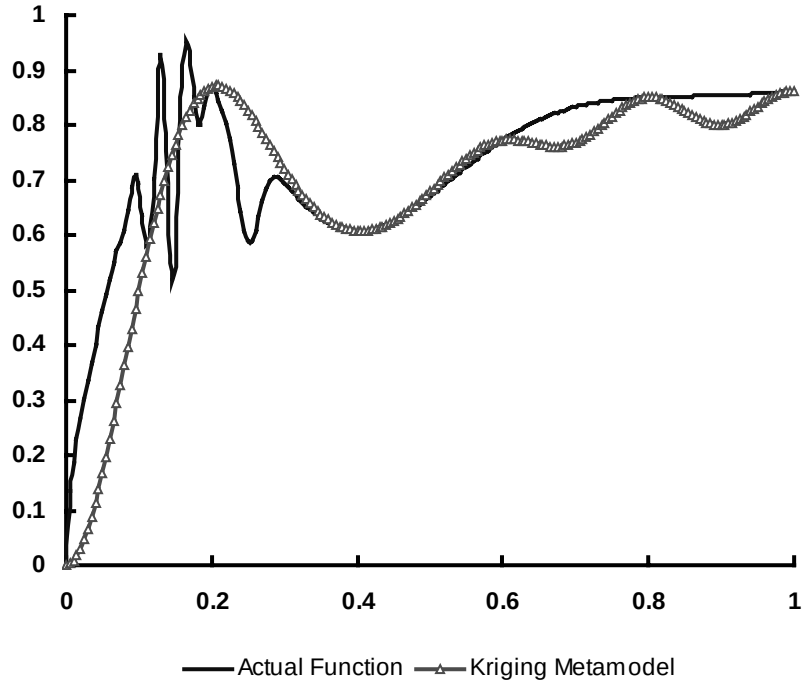


Figure 5.2 Kriging Metamodel with 6 Data Points

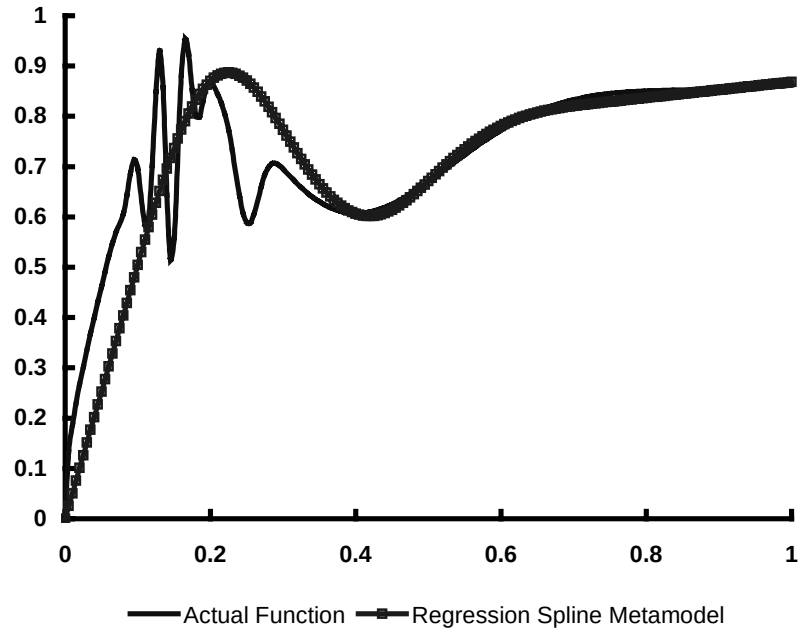


Figure 5.3 Regression Spline Metamodel with 6 Data Points

Now let us see how kriging and regression spline metamodels perform with 12 data points. Similar to that in Table 5.1, we select 12 data points evenly spread over $x \in [0,1.0]$, as listed in Table 5.2. The corresponding kriging metamodel is illustrated in Figure 5.4. The value of θ for the kriging metamodel is 19.49807. The regression spline metamodel is illustrated in Figure 5.5. Details of this regression spline model are presented in Appendix B.

Table 5.2 Data Point Set II – 12 Points

x	0.0	0.090909	0.181818	0.272727	0.363636	0.454545
y	0.0	0.694415	0.794018	0.674573	0.619361	0.628709
x	0.545455	0.636364	0.727273	0.818182	0.909091	1.0
y	0.718714	0.80075	0.840776	0.851326	0.854544	0.861688

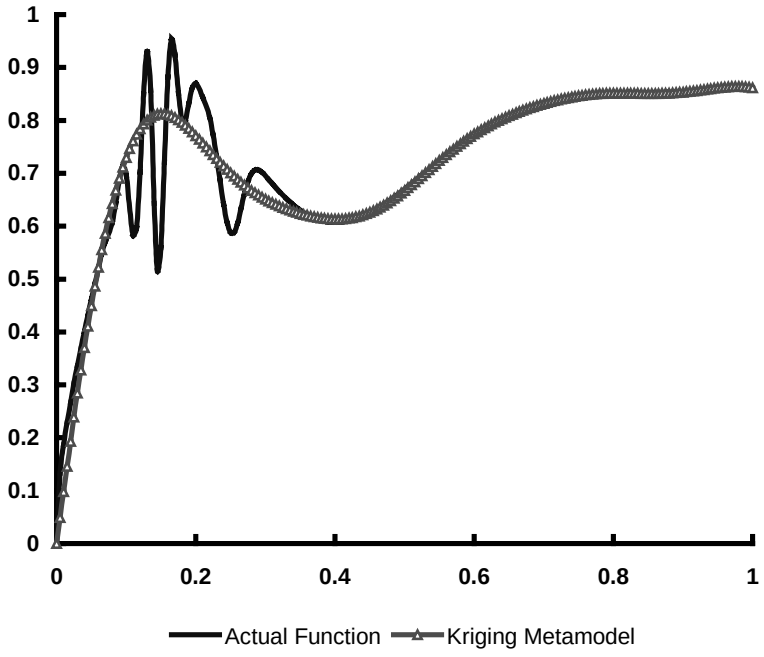


Figure 5.4 Kriging Metamodel with 12 Data Points

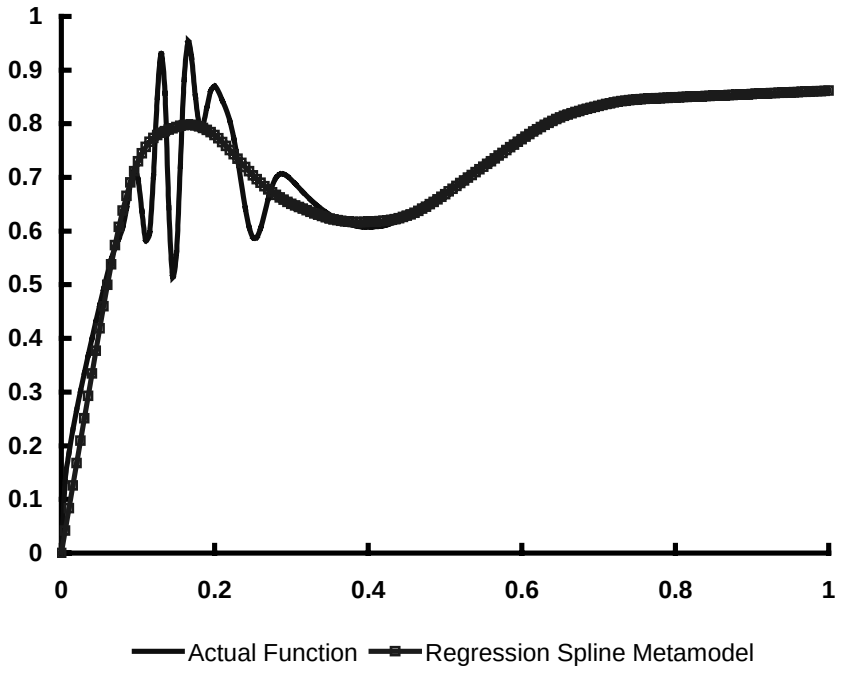


Figure 5.5 Regression Spline Metamodel with 12 Data Points

In Figure 5.4 and Figure 5.5 we see that the kriging and regression spline metamodels perform approximately the same in response prediction. Compared to Figure 5.2 and Figure 5.3 in which only 6 data points are used, the metamodels in Figure 5.4 and Figure 5.5 do not improve much even we used 12 data points. The reason is that all data points are allocated evenly over the design space, thus there are no enough points in the highly nonlinear region. As discussed in Chapter 4, this is the main shortcoming of single-stage experimental design, and the SEED method could help achieve accurate metamodels with relatively fewer data points.

Another thing to be noticed is that the kriging metamodel improves more than the regression spline metamodel after using more data points. This is because that the kriging

metamodel with 6 data points (Figure 5.2) does not perform very well around $x = 0.7$ and 0.9. By using more data points in the flat region (with large x values), the kriging metamodel is able to grasp the fluctuation on the response surface.

It is expected that an accurate kriging metamodel could be developed as long as we have enough data points. Now let us see how kriging and regression spline metamodels perform with 18 data points for the single-variable function. These 18 data points are selected uniformly in [0,1], as listed in Table 5.3. The corresponding kriging and regression spline metamodels are shown in Figure 5.6 and Figure 5.7, respectively. The value of θ is 99.99999683, and this works in this case. Details about the regression spline metamodel are attached in Appendix B. In Figure 5.6 and Figure 5.7 we see that both kriging and regression spline metamodels are more accurate with 18 data points than those with 6 or 12 data points.

Table 5.3 Data Set III – 18 Points

x	0.0	0.058824	0.117647	0.176471	0.235294	0.294118
y	0.0	0.51409	0.642286	0.83149	0.684104	0.702732
x	0.352941	0.411765	0.470588	0.529412	0.588235	0.647059
y	0.627525	0.608298	0.641646	0.701418	0.76189	0.807675
x	0.705882	0.764706	0.823529	0.882353	0.941176	1
y	0.83488	0.847302	0.0851541	0.85343	0.856438	0.861688

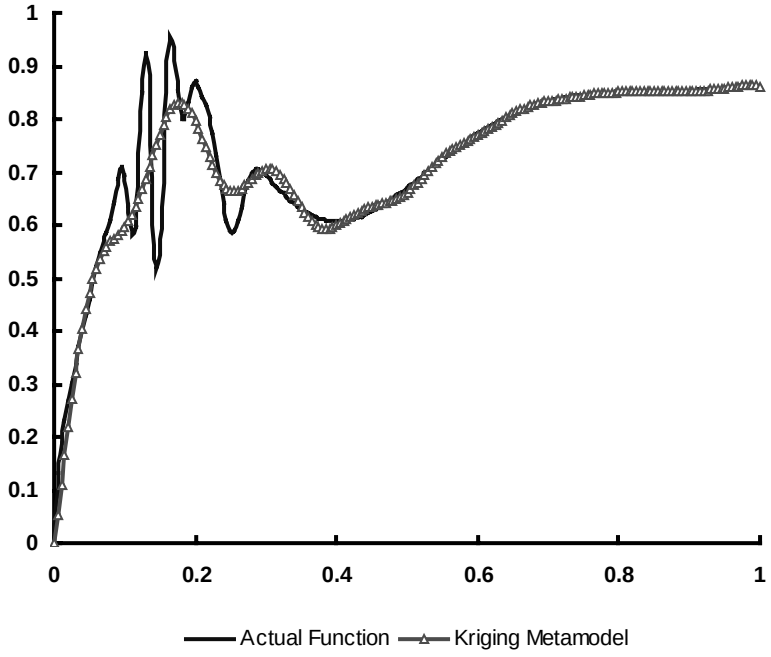


Figure 5.6 Kriging Metamodel with 18 Data Points

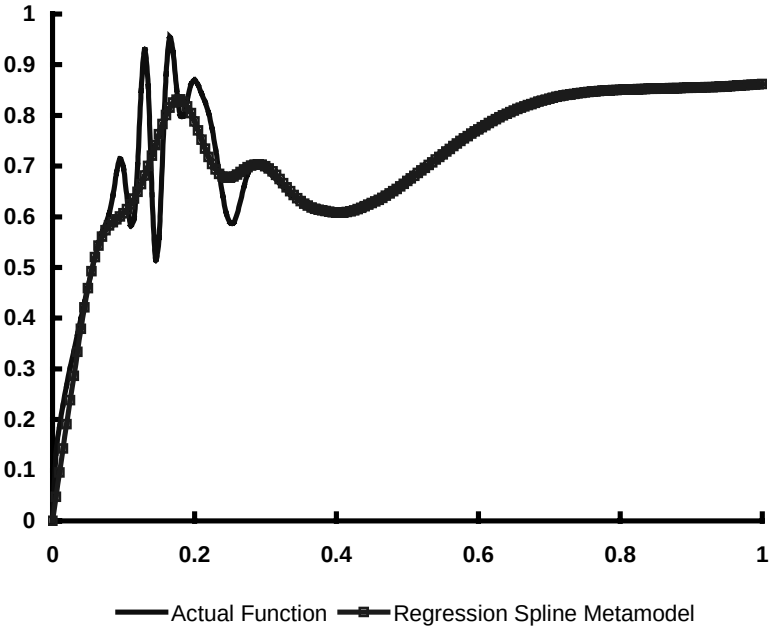


Figure 5.7 Regression Spline Metamodel with 18 Data Points

Now let us see how kriging and regression spline metamodels work with 24 data points. Similar to previous studies, these data points spread over the design space uniformly, as shown in Table 3.4. The corresponding regression spline metamodel is illustrated in Figure 5.8. Details about this regression spline metamodel are presented in Appendix B. As for the kriging metamodel, we got θ = 283.0647. The kriging metamodel is illustrated in Figure 5.9.

Table 3.4 Data Set IV – 24 Points

x	0	0.0434783	0.0869565	0.1304348	0.1739130	0.2173913
y	0	0.422047	0.660364	0.930513	0.866839	0.816847
x	0.2608696	0.3043478	0.3478261	0.3913043	0.4347826	0.4782609
y	0.610096	0.688948	0.632174	0.608023	0.616353	0.64856
x	0.5217391	0.5652174	0.6086957	0.6521739	0.6956522	0.7391304
y	0.693114	0.739403	0.779896	0.810762	0.831379	0.843307
x	0.7826087	0.8260870	0.8695652	0.9130435	0.9565217	1
y	0.849152	0.851638	0.853004	0.854741	0.857589	0.861688

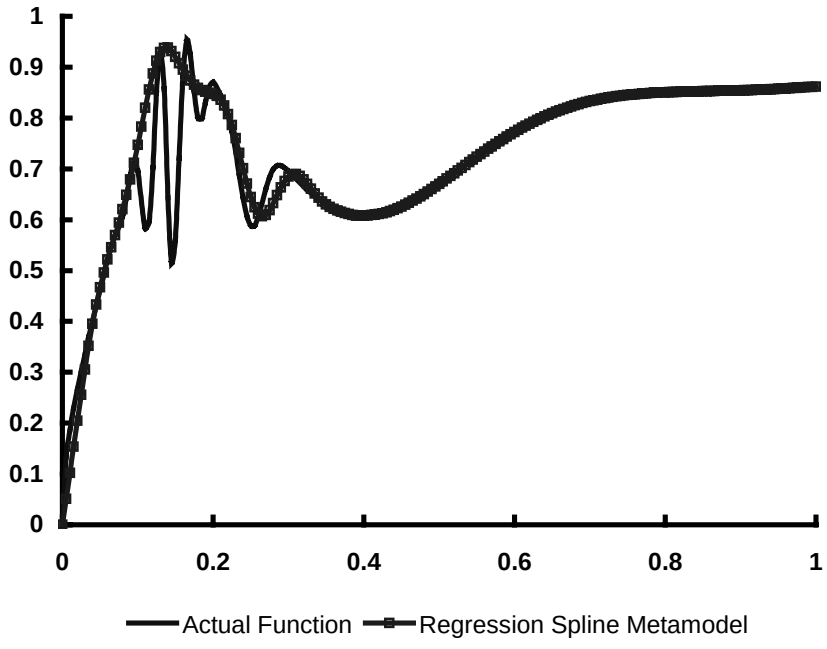


Figure 5.8 Regression Spline Metamodel with 24 Data Points

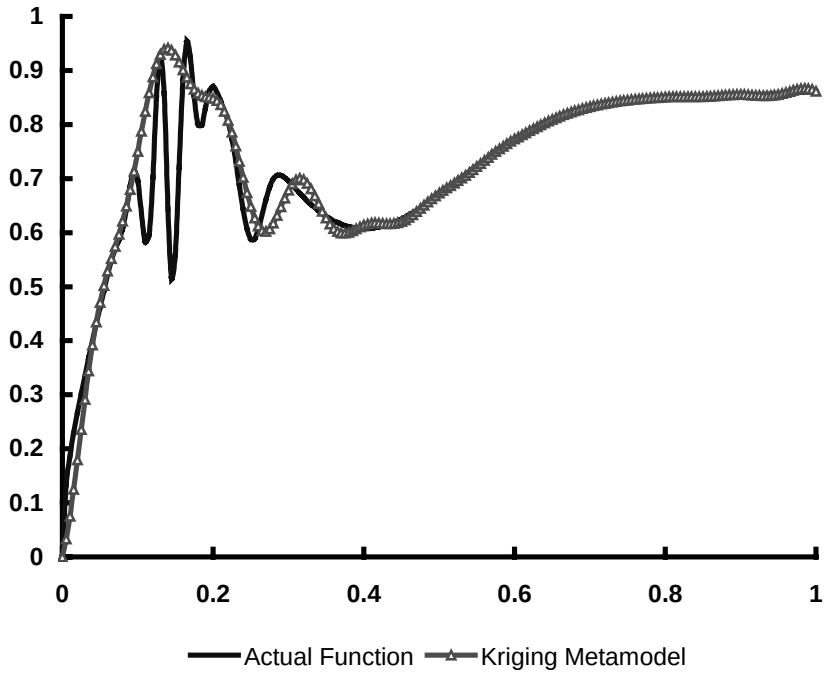


Figure 5.9 Kriging Metamodel with 24 Data Points

Now let us see the last "space-filling" design with 65 data points – all these points are evenly spreading over the design space of [0,1]. Details about these points will not be put here. The regression spline metamodel is illustrated in Figure 5.10. The value of θ in the kriging metamodel is 2099.77433. The corresponding kriging metamodel is illustrated in Figure 5.11. In Figure 5.11 we see that though the kriging metamodel has some fluctuations on the deep slope between [0,0.1] (note that in Figure 5.10 the regression spline metamodel performs well on this slope – the predicted surface is very smooth), it captures the highly nonlinear response surface in [0.1,0.3] better than previous kriging metamodels do.

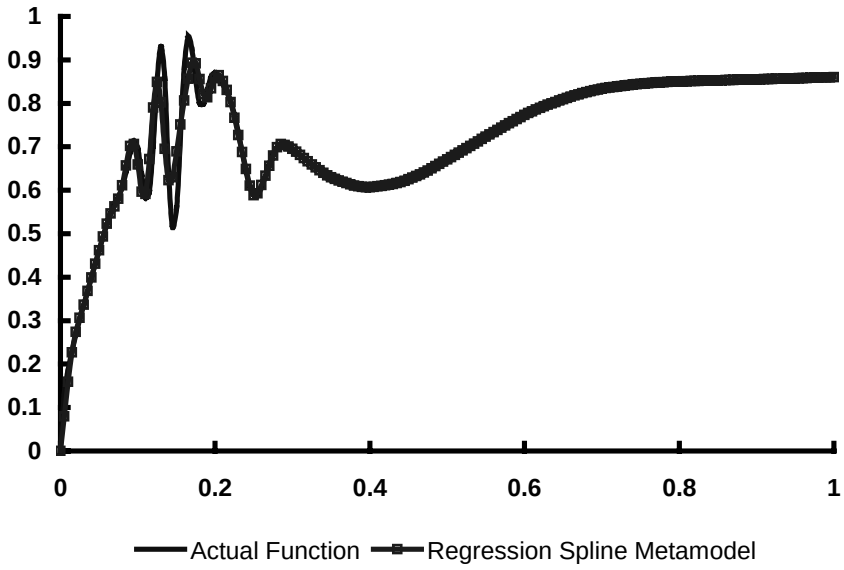


Figure 5.10 Regression Spline Metamodel with 65 Data Points

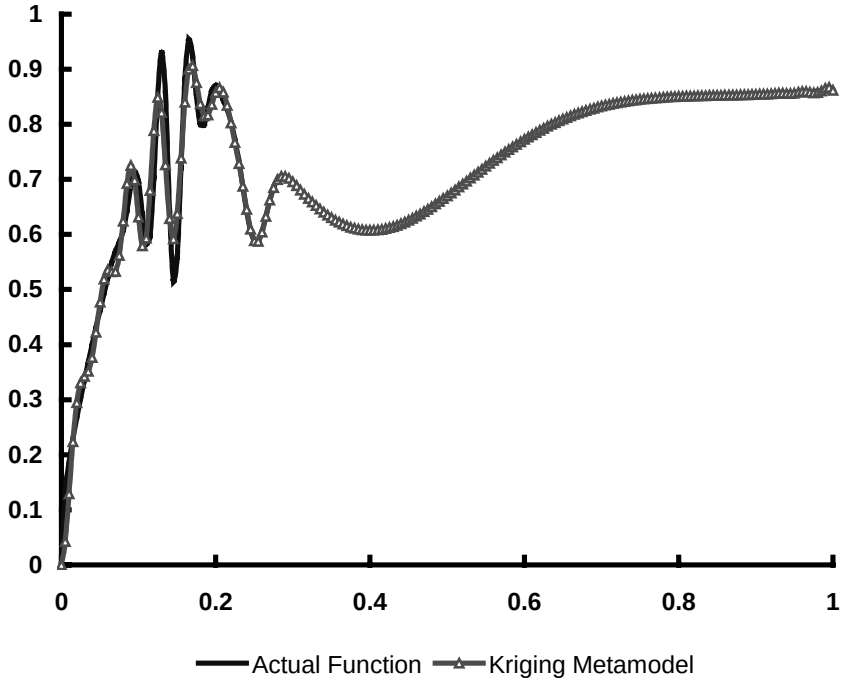


Figure 5.11 Kriging Metamodel with 65 Data Points

Suppose now we want to develop a very accurate metamodel for the single-variable function, thus more data points should be used. This time we decide to use 201 data points evenly spread over [0,1]. The corresponding regression spline metamodel is illustrated in Figure 5.12; details are presented in Appendix B. The kriging metamodel is illustrated in Figure 5.13; the value of θ is 7016.42038.

Based on the examples above, we observe that when the number of data point increases, the value of θ increases too. It has been stated by many researchers (see, e.g., Simpson, 1998; Farhang-Mehr and Azarm, 2002) that a value of 10 to 100 for θ implies very rapid decaying correlation. In our examples, we meet θ values much larger than their suggestions (actually, we had to modify Simpson's code to increase its upper limit on possible θ values). On a highly nonlinear response surface, each of the data points conveys little information at its neighborhood; this yields a large value for θ. When there are few data points, the high nonlinearity on the response surface is not captured and a small θ value could explain the available information. However, when more data points are used, the high nonlinearity on the response surface is sensed and as a result, larger values of θ are needed to reflect this fluctuating surface. In this case, the response surface in [0.1, 0.4] is the focus in our study; its high nonlinearity affects the value of θ.

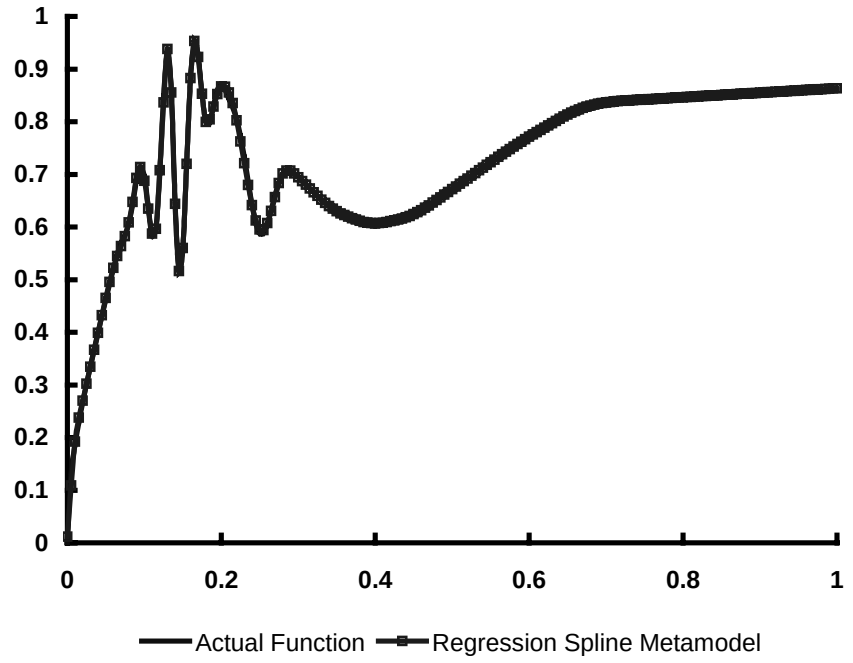


Figure 5.12 Regression Spline Metamodel with 201 Data Points

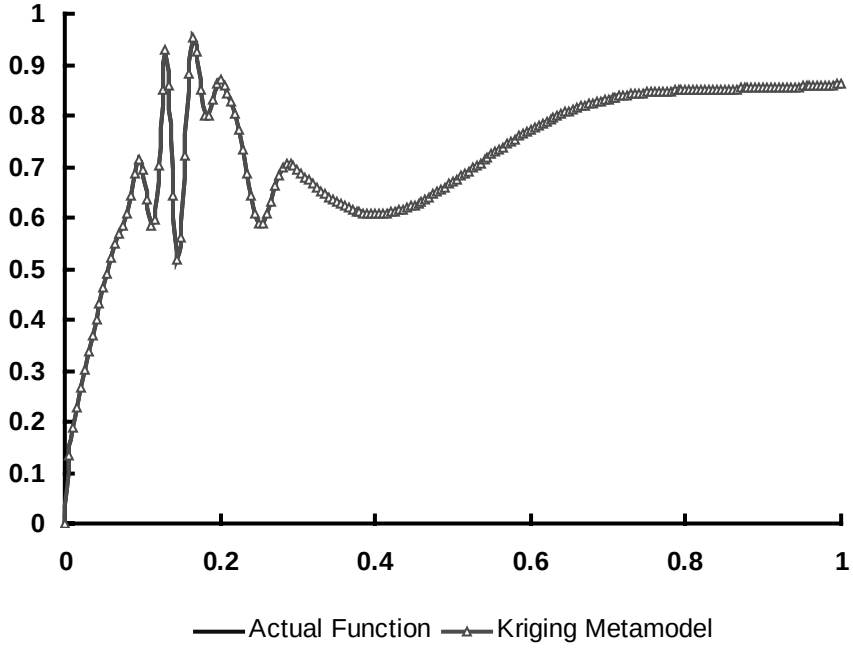


Figure 5.13 Kriging Metamodel with 201 Data Points

Another thing to be noted is the computation time in developing kriging and regression spline metamodels. It is expected that the computation time increase when the number of design variables and that of data points increases. In this example of the single-variable function, it takes little time (<< 1 second) to build the regression spline metamodel with our P4, 196M computer. With Simpson's code, it takes less than 1 second to build kriging metamodels for the single-variable function with fewer than 24 data points; in the case with 65 data points, kriging metamodel fitting costs 2 seconds, and in the case with 201 data points, we spend 135 seconds to fit the kriging metamodel. It seems that the regression spline is a little superior to kriging on saving the computation time.

Observations above support the assertion that more accurate kriging and regression spline metamodels could be developed with more data points spreading over the design space; when enough data points are selected and placed evenly in the design space, a metamodel could be developed as loyal to the actual function as possible. While in the next section, our study shows that the assumption above is not always valid when metamodels are developed with non-space-filling experiments (unevenly spread data points).

5.2.2 An Observation and Analysis on the Performance of Kriging and Regression Spline Metamodels in Response Prediction with Unevenly Spread Data Points

In Section 5.2.1, we studied the performance of kriging and regression spline metamodels with evenly spread data points in the design space. We observe that with

more data points evenly spreading over the design space, more accurate kriging and regression spline metamodels could be developed. Values of θ in kriging metamodels and the computation time to build a kriging metamodel increase as the number of data points increases. In our examples, regression spline metamodels seems a little superior to kriging metamodel since: 1). In some cases (with 8, 24, or 65 data points) the regression spline metamodels are smooth while kriging metamodels have tiny fluctuations on the predicted response surfaces, and 2). As the number of data points increases, it tends to cost much more computation time to build a kriging metamodel than to build a regression spline metamodel.

In this section, we study the performance of kriging and regression spline metamodels when unevenly spread data points are used in metamodeling. As presented in Chapter 4, in sequential exploratory experimental design, new data points are added not to "spread over" the design space, but rather to "reduce predicted prediction errors". It is expected that more data points would be added in regions with high nonlinearity or high prediction errors, thus in sequential experimental design, we may get sets of data points that are not evenly allocated in the design space. It is important to study the performance of kriging and regression spline metamodels with unevenly allocated data points in sequential experiments.

In this study we still use the single-variable function as presented in Equation (5.1). A wise designer (as expected in sequential experimental designs) might use the set of data points as listed in Table 5.5. In Table 5.5 we see that the data points are apparently unevenly allocated in the design space; a large portion of data points are put in

the highly nonlinear region of [0.09, 0.4] (7 data points in [0.09, 0.2]), and only two data points in the flat region of [0.5, 1]. Data points in the region of [0.09, 0.4] are very close to local peaks and bottoms on the highly nonlinear response surface. It is expected that accurate kriging and regression spline metamodels could be developed with this set of data points.

The corresponding regression spline metamodel is illustrated in Figure 5.14; details about this metamodel are presented in Appendix B. However, we meet problems in building the kriging metamodels. Simpson's code gives the value of θ as 2.34233, which is apparently incorrect because predicted response values at data points with the corresponding kriging metamodel are totally different from true values, which should not happen in deterministic kriging (note that prediction errors at data points should be zero). The value of θ from iSIGHT is 99.999999. Predicted response values range from $y \approx -1355$ to $y \approx 550$, while the actual function values are in [0, 0.95]. The corresponding kriging metamodel is illustrated in Figure 5.15; as we see, compared with the kriging metamodel, the actual function is like a horizontal line on the *x*-axis. However, designers may not be able to see this because the predicted response values are the same as true ones (note that designers may only have information at data points).

Table 5.5 Effective Data Set – 13 Points

x	0	0.095	0.11	0.13	0.145	0.165	0.185
y	0	0.71380	0.58279	0.93006	0.51582	0.95305	0.79902
x	0.2	0.25	0.29	0.4	0.67	1.0	
y	0.87017	0.58812	0.70592	0.60728	0.82040	0.86169	

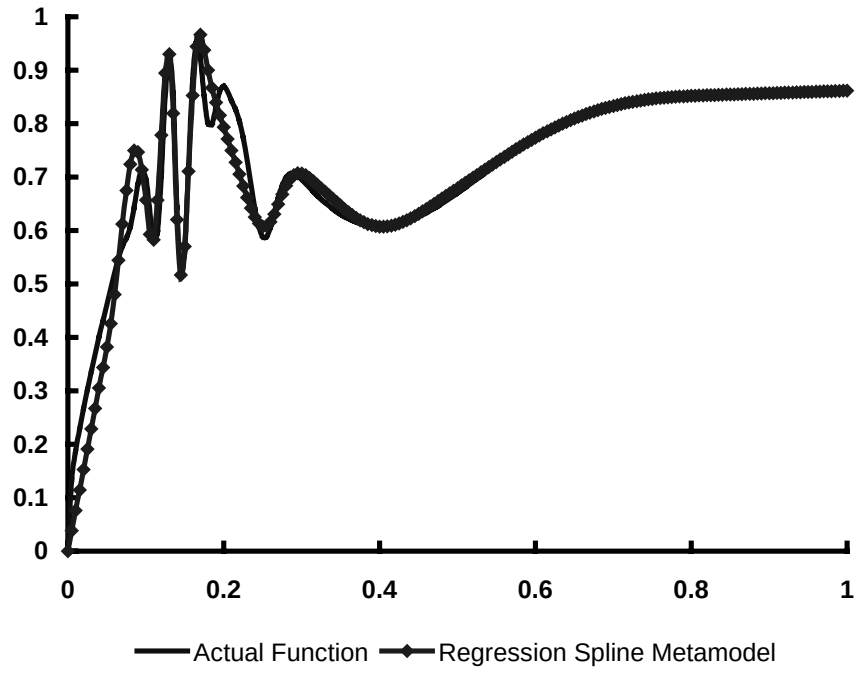


Figure 5.14 Regression Spline Metamodel with 13 Data Points

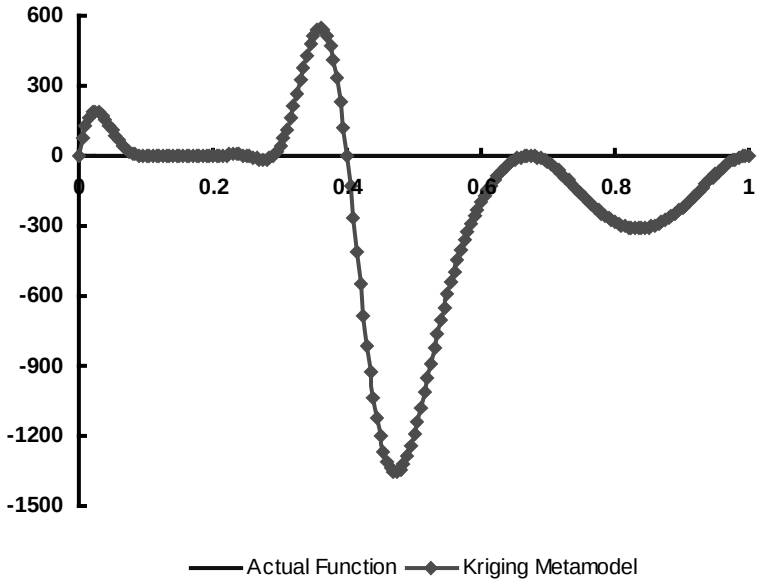


Figure 5.15 Kriging Metamodel with 13 Data Points (θ = 99.999999)

One possible reason for the problem above may be the limitation of algorithms or software; this caught our attention since Simpson's code and iSIGHT yield different results. This implies that the θ values we got might be incorrect. To study this possibility, we develop several kriging metamodels with different θ values and analyze their performance. Kriging metamodels with θ = 50, 500, 1000, and 5000 are illustrated in Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19, respectively. From these figures we see that as values of θ increase, the range of predicted responses decreases and the corresponding metamodel becomes more and more accurate – the highly nonlinear response surface in [0.09, 0.4] is better reflected with large θ values.

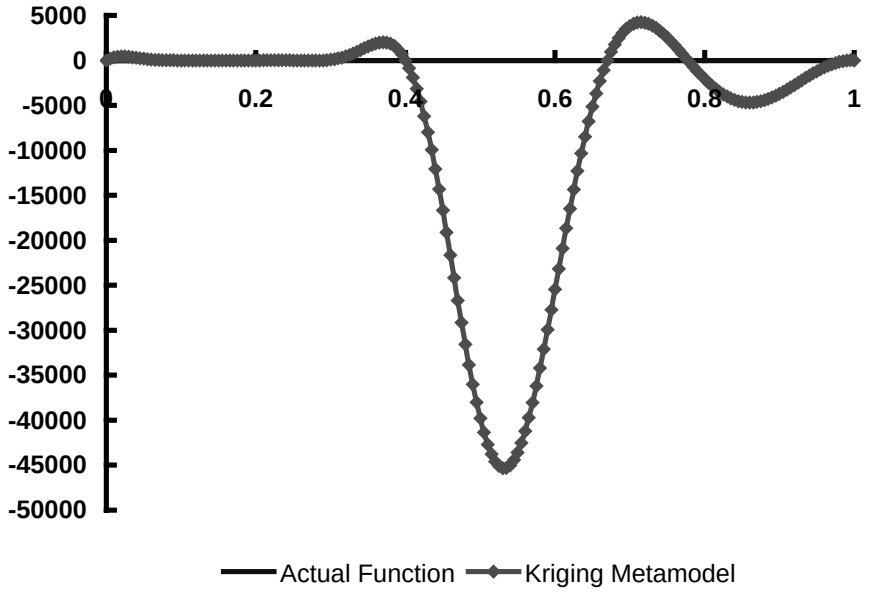


Figure 5.16 Kriging Metamodel with 13 Data Points (θ = 50)

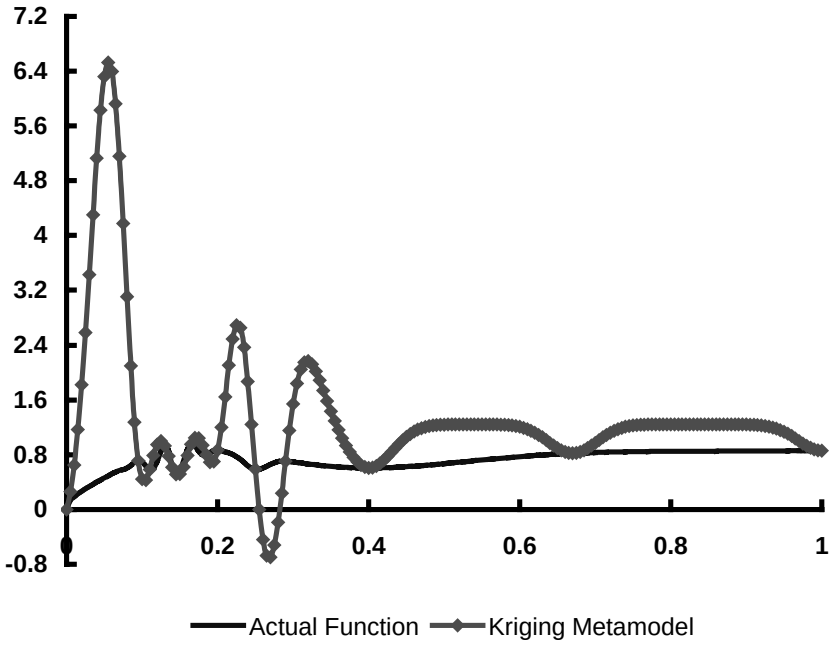


Figure 5.17 Kriging Metamodel with 13 Data Points (θ = 500)

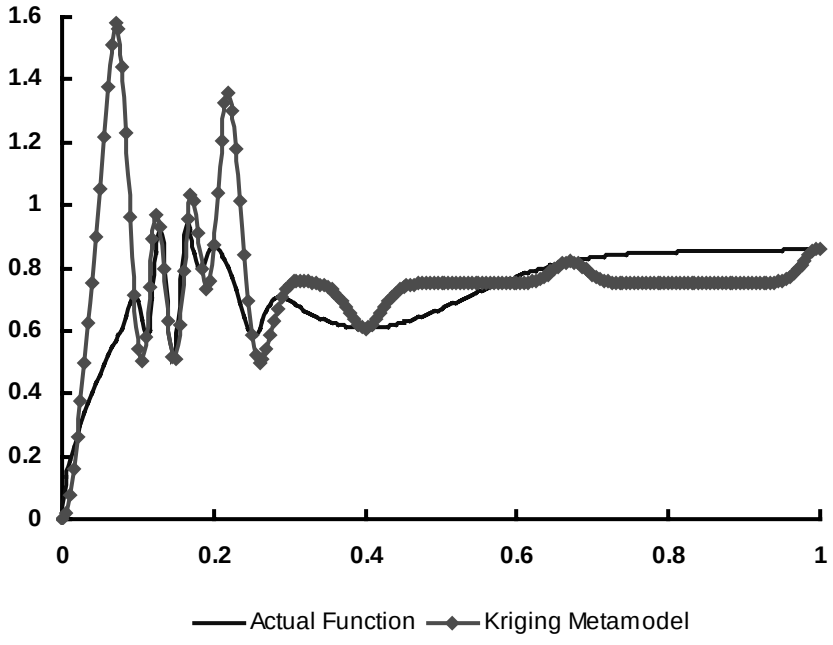


Figure 5.18 Kriging Metamodel with 13 Data Points (θ = 1000)

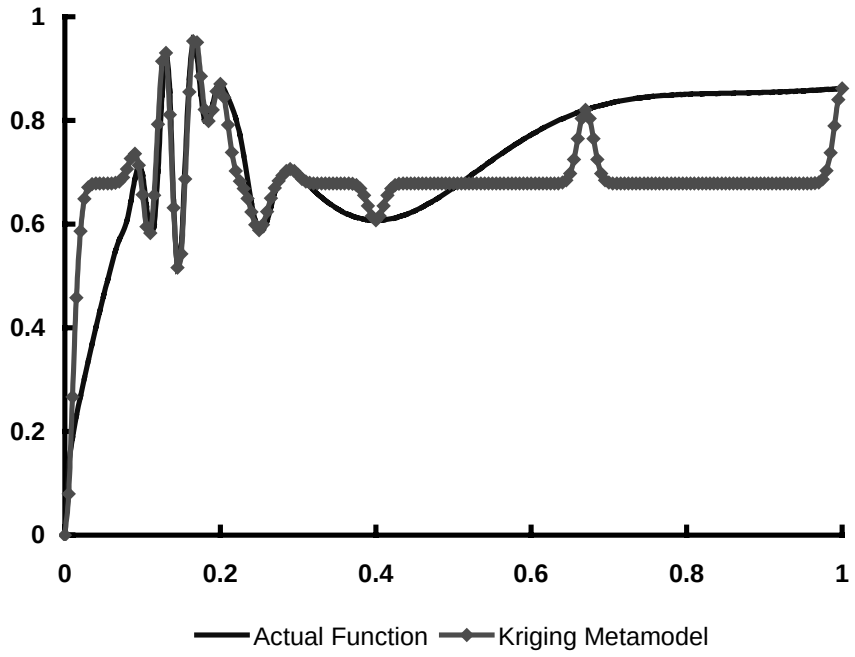


Figure 5.19 Kriging Metamodel with 13 Data Points (θ = 5000)

In Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19, we see that as θ values increase, the predicted response surface becomes flat (horizontal) in [0.5,1] except several sharp peaks at data points in this region. It seems that we are not able to get an acceptable kriging metamodel with the set of data points as listed in Table 5.5. Kriging metamodels do not work well with unevenly allocated data points. Possible drawbacks of kriging software are not the very reason for the unreasonable kriging metamodel in Figure 5.15; kriging metamodeling is constrained by its own limitations.

It is important to know why kriging metamodeling meets problems in modeling this single variable function with data points listed in Table 5.5. The reason lies in the global usage (in one dimension) of θ in the design space. As discussed in Chapter 2 and

Chapter 4, the parameter θ in maximum entropy sampling and kriging metamodeling represents correlation between points in the design space. As the value of θ increases, the correlation between two points becomes weaker and weaker (suppose the distance between these two points is a constant). Thus the value of θ in a highly nonlinear design space should be much larger than that in a flat one. In this sense, the parameter θ could also be viewed as an indicator of how much information one data point could reflects in its neighborhood. A small θ indicates that a data point in the design space reflects much information in its neighborhood; or say, it has great influence on response values in its neighborhood. As stated before, values of 10 to 100 for θ indicate a very rapid decaying correlation between points. In the single-variable function as presented in Equation (5.1) and Figure 5.1, the response surface in [0.09, 0.4] is highly nonlinear and could only be reflected by kriging metamodels with very large θ values (e.g., 2099.77433 as in Figure 5.11). Data points in this region could reflect very little information in their neighborhoods – and this is why we need to put more data points in this region in sequential experiments. The responses surface in [0.5, 1] is very flat and could be modeled by kriging metamodels with very small θ values. Data points in this region could reflect much information in their neighborhoods, thus only a few data points are needed in this region. In kriging, the value of θ is universal in one dimension; the various demands of θ values as discussed above cause dilemma that cannot be compromised in kriging metamodeling with unevenly allocated data points. This is the very reason why

we could not develop acceptable kriging metamodels with the set of data points in Table 5.5.

With large θ values, the highly nonlinear part of the actual response surface could be modeled accurately, while in flat regions where we have few data points, the kriging metamodel tends to rest at its constant β (see Equations (2.18) and (2.20)) and be occasionally dragged to observed values at data points – this is why we see sharp peaks on flat (horizontal) surfaces in Figure 5.17, Figure 5.18, and Figure 5.19.

To solve this problem, one method is to put more data points in the flat region; data points should be very close to each other – a data point should be put where another one's influence demises. This results in an experimental design in which data points are almost evenly allocated in the whole design space. As we see in Section 5.2.1, when "space filling" experiments are used, accurate kriging metamodels could be developed as long as we have enough data points. This selection of evenly allocated data points is not desired in our sequential experimental design because a lot of effort is wasted on data points in flat regions.

In our approach, we recommend replacing kriging with MARS in metamodeling with sequential experiments when necessary. Kriging metamodels are still very useful and could not be totally discarded in our approach because: 1). Kriging metamodels predict the exact values at observed points while MARS metamodels smoothes the data – and this is important in metamodeling with deterministic computer experiments as explained in Chapter 2, and 2). As discussed in Chapter 4, in the SEED method, θ values

from current kriging metamodels could be used in the identification of new data points to help distinguish dimensions with high nonlinearity – more data points could be automatically put in dimensions with large θ values with our modified maximum entropy sampling approach. Thus in conceptual design, when we want to develop metamodels for system responses with sequential experiments, we may use:

- MARS metamodels only. The shortcomings are that the metamodel smoothes the data (so the predicted value at data points may not be accurate), and we may not be able to identify highly nonlinear dimensions quickly.
- Kriging metamodels only. This is when the actual response surface is not very complicated. However, in practice designers do not know how the actual response surface look; and it is very difficult to tell when kriging meets difficulty in the metamodeling process.
- Kriging and MARS metamodels together. Kriging metamodels could be used in early stages of design when the data points are nearly evenly allocated in the design space. As the metamodeling process goes on, we may switch to MARS metamodels whenever a problem is identified. One way to identify problems is to develop both kriging and MARS metamodels and compare their predictions for abnormal performance.

Besides metamodels for system responses, we also develop metamodels for prediction errors in SEED. In SEED processes, metamodels for prediction errors are expected to be more complicated than those for system responses because there are a lot

of points with zero prediction errors mixed with points with positive or negative prediction errors. This could be illustrated with the example in Figure 5.20.

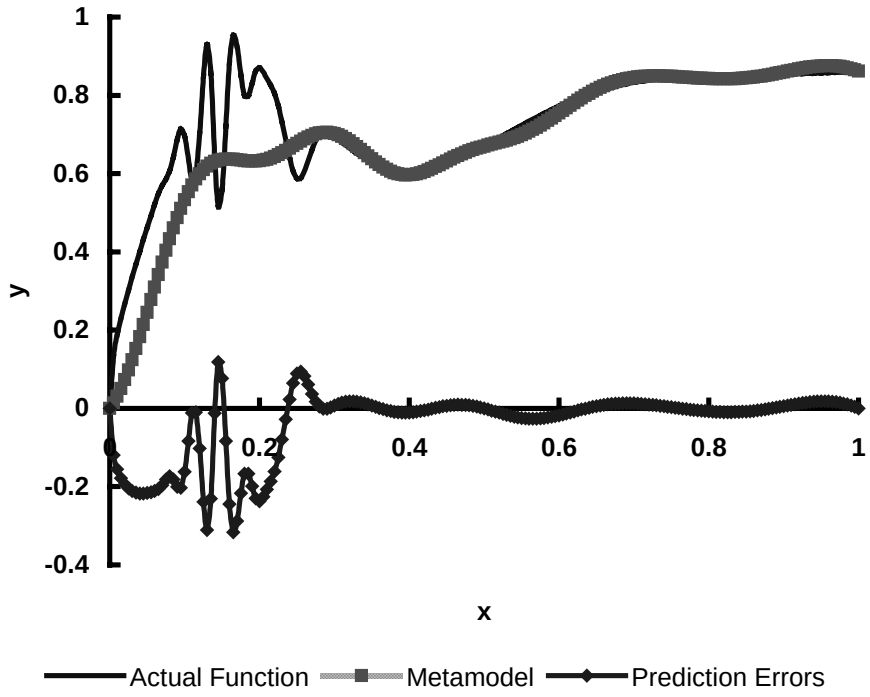


Figure 5.20 An Example of Metamodel and Prediction Errors

The example in Figure 5.20 is obtained when we tried to apply SEED (with kriging metamodels only) in modeling the single-variable function in Equation (5.1). In Figure 5.20 we see that since the actual function is highly nonlinear and the metamodel is flat, the actual response surface for prediction errors is highly nonlinear. 10 validation points are selected; prediction errors at these validation points and 11 data points are listed in Table 5.6. The corresponding regression spline metamodel is shown in Figure 5.21.

Table 5.6 Prediction Errors at 21 Points

x	0.0	0.0542	0.0699	0.1038	0.1133	0.1748	0.1947
y_err	0.0	-0.21447	-0.19659	-0.10392	0.0	-0.21942	-0.22848
x	0.2377	0.2849	0.3557	0.3615	0.4320	0.5	0.5448
y_err	0.0	0.0	0.00361	0.0	0.0	0.0	-0.02285
x	0.5839	0.6362	0.6802	0.7623	0.8838	0.9054	1.0
y_err	-0.02367	0.0	0.01256	0.0	0.0	0.0073	0.0

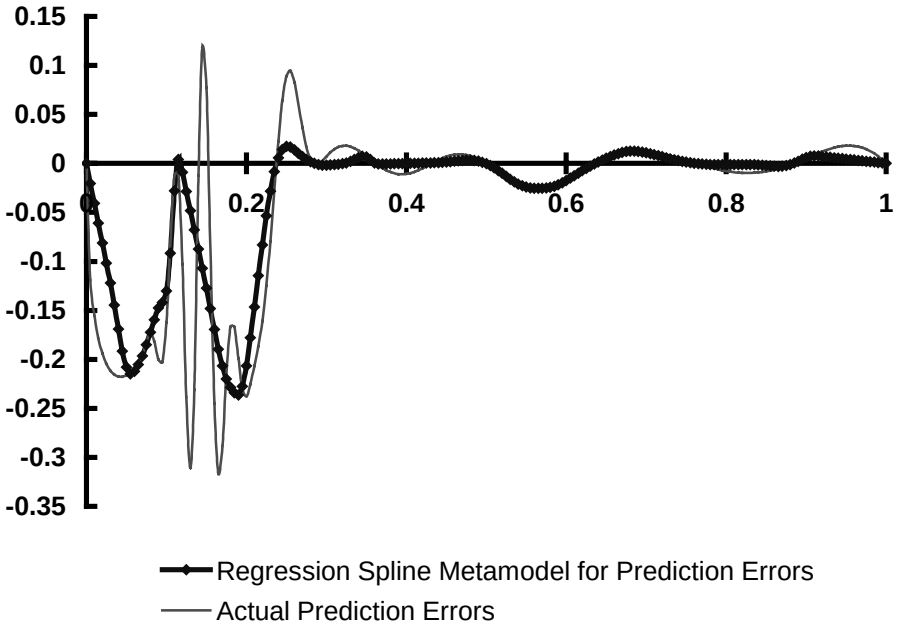


Figure 5.21 Regression Spline Metamodel for Prediction Errors with 21 Points

In Figure 5.21 we see that the regression spline model captures the actual response well, except for the highly nonlinear region where we may add in more validation points in future stages. We failed to build an acceptable kriging metamodel in this case, partly because of the highly nonlinear property of the actual surface of prediction errors which is inherited from the single-variable function, partly because of the mixture of zero's and non-zero's for prediction errors in the design space. For example, we notice that there are

three zero's from x=0.36 to x=0.5, which implies a flat (horizontal) surface, while in other regions the surface may not be flat – this is usual in modeling prediction errors in SEED. When this confliction is intense enough, as shown in this case, we will fail in developing an acceptable kriging metamodel.

5.2.3 An Observation and Analysis on the Performance of Kriging and MARS Metamodels in Response Prediction with Unevenly Spread Data Points

In Section 5.2.2, we compared the performance of kriging and univariate regression spline metamodels with a single-variable function. Our observations show that the regression spline metamodels are more robust to irregular response surfaces while it is difficult to use kriging to model irregular response surface that is highly nonlinear in some regions but flat in other regions. In this section, we will extend this comparison to kriging and MARS metamodels with a two-variable function. The two-variable function is as presented in Equation (5.2).

$$f(x_1, x_2) = (1 - e^{-2\sqrt[4]{x_1^2 + x_2^2}}) + 6\sqrt{x_1^2 + x_2^2}\, e^{-7\sqrt{x_1^2 + x_2^2}} \sin(10\sqrt{x_1^2 + x_2^2}) - 0.2e^{-2000(\sqrt{x_1^2 + x_2^2} - 0.25)^2} + 60\min(0, |\sqrt{x_1^2 + x_2^2} - 0.14| - 0.08)^2 [\ln(\sqrt{x_1^2 + x_2^2} + 0.2) + 1.5\sin^2(85\sqrt{x_1^2 + x_2^2})] \quad (5.2)$$

Equation (5.2) is a modified two-variable version of Equation (5.1); the modification is done by substituting x in Equation (5.1) with $\sqrt{x_1^2 + x_2^2}$. The surface plot and contour plot of this function are illustrated in Figure 5.22 and Figure 5.23. As we see in Figure 5.22

and Figure 5.23 the two-variable function is highly nonlinear with small x_1 and x_2 values and flat with large when x_1 and x_2 are large; the actual response surface is irregular.

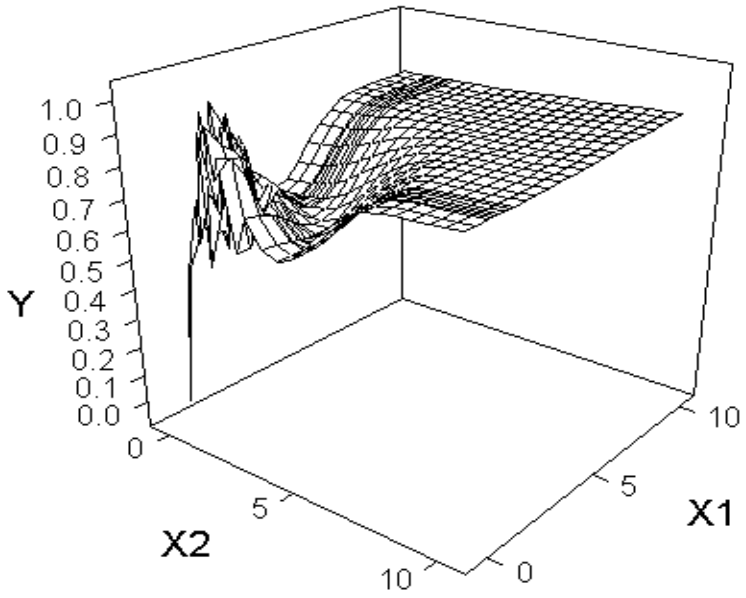


Figure 5.22 Surface Plot of the Two-Variable Function

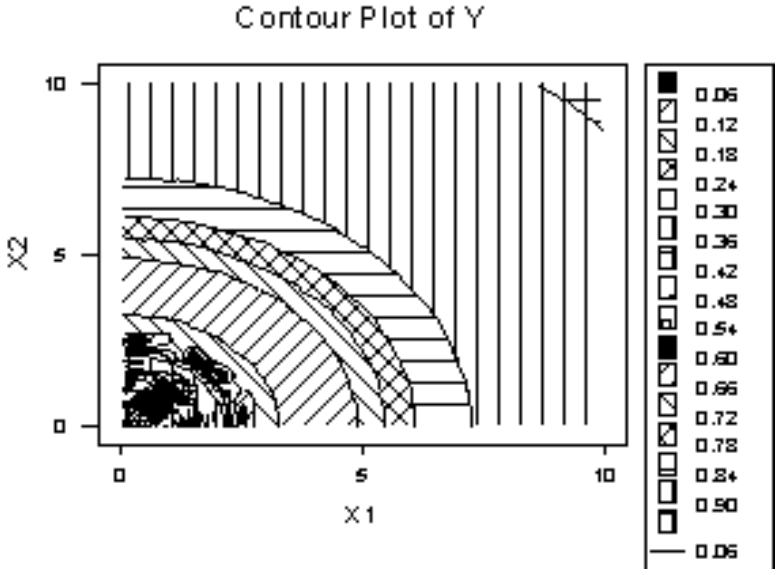


Figure 5.23 Contour Plot of the Two-Variable Function

According to our observations in Section 5.2.2, since the actual response surface is irregular (i.e., highly nonlinear in some regions while flat in others; the property of the response surface changes greatly), it is expected that: 1) it is difficult to get an accurate kriging metamodel for this two-variable function with reasonable number of data points, and 2) it is possible to build an accurate MARS (application of regression splines with multiple variables in this example) metamodel with data points putting at "critical" positions (unevenly spread data points, as developed in SEED).

Similar to what we did in Section 5.2.2, first we act as a "wise" designer here. As a wise designer, we are able to put most data points at critical positions, as expected from a sequential experimental design; in this example, we could achieve this by carefully examining the plots of Figure 5.22 and Figure 5.23. A possible set of data points with 45 data points is listed in Table 5.7.

With the data points in Table 5.7, we are unable to develop a kriging metamodel that is accurate; actually, the kriging metamodel we build, with θ_1 = 2.84701 and θ_2 = 1.97175, does not perform normally. We use 625 points (which spread over the design space, with more being allocated in the region with small x values) to validate the kriging metamodel. In Figure 5.22 and Figure 5.23 we see that the actual response values in the design space are in the range of [0,1], while the values predicted with the kriging metamodel at the validation points are very large (many are over −100,000,000). The reason why we cannot develop an acceptable kriging metamodel in this example lies in the irregularity of the actual response surface, as explained in Section 5.2.2.

Table 5.7 Experiments with 45 Data Points

X_1	X_2	Y	X_1	X_2	Y	X_1	X_2	Y
0	0	0	0.11	0	0.58279	0.2	0	0.87017
0	1	0.86169	0	0.11	0.58279	0	0.2	0.87017
1	0	0.86169	0.078	0.078	0.58112	0.141	0.141	0.87037
1	1	0.90773	0.13	0	0.93006	0.25	0	0.58812
0	0.5	0.67001	0	0.13	0.93006	0	0.25	0.58812
0.5	0	0.67001	0.092	0.092	0.93029	0.177	0.177	0.58758
1	0.5	0.87670	0.145	0	0.51582	0.3	0	0.69537
0.5	1	0.87670	0	0.145	0.51582	0	0.3	0.69537
0.5	0.5	0.83527	0.103	0.103	0.51171	0.212	0.212	0.69563
0.25	0.75	0.84978	0.165	0	0.95305	0.1	0.8	0.85076
0.75	0.25	0.84978	0	0.165	0.95305	0.8	0.1	0.85076
0.75	0.75	0.86900	0.117	0.117	0.95403	0.05	0.3	0.68927
0.05	0	0.46195	0.185	0	0.79902	0.3	0.05	0.68927
0	0.05	0.46195	0	0.185	0.79902	0.165	1	0.86319
0.035	0.035	0.45891	0.131	0.131	0.80017	1	0.165	0.86319

A MARS metamodel is developed with information from the data points listed in Table 5.7. To build this MARS model, we set the number of knots in each dimension as $T = 100$, the maximum number of MARS basis functions in approximation $Mmax = 50$, the maximum number of splits per basis function $maxIA = 2$ to allow two-way interactions. Backwards deletion is allowed in building the MARS metamodel. The MARS metamodel is illustrated in Figure 5.24. In Figure 5.24 we see that the MARS metamodel roughly grasps the irregular response surface. We examined the prediction errors at 625 validation points and get RMSE = 0.0739 and MAX = 0.395 based on Equations (2.7) and (2.9). The value of RMSE is small, which indicates that the overall model fitting is satisfactory and the metamodel grasps the fluctuations of the whole response surface; design space exploration with such a metamodel would probably

successfully lead us to regions in which the design solution lies. The value of MAX is large, which implies that *Local Model Inaccuracy* (see Lin, et al., 1999) exists and it might be difficult to precisely identify the final design solution with this metamodel though we could have been led to the region where the solution lies.

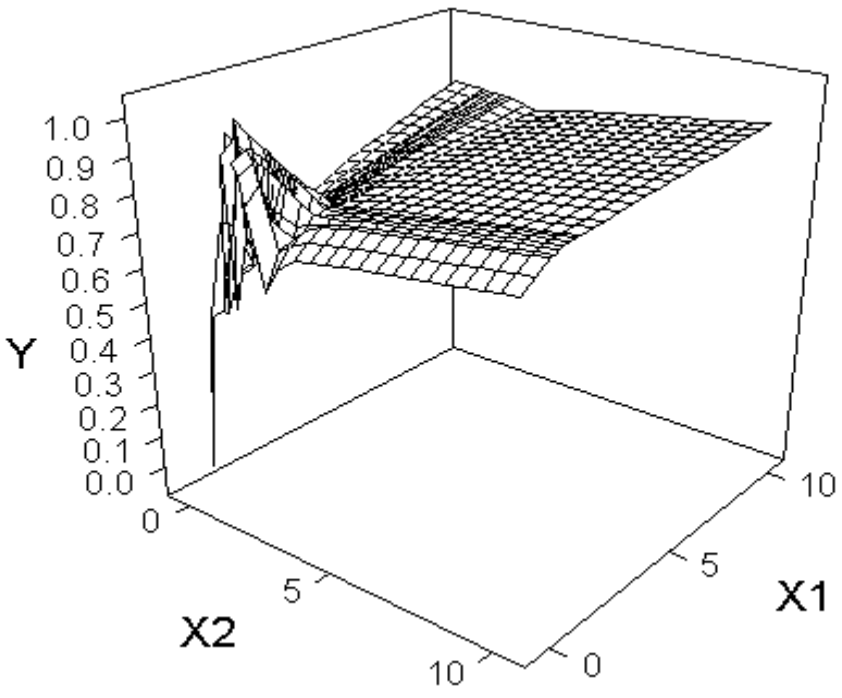


Figure 5.24 MARS Metamodel with 45 Data Points

In this section, through the study of kriging and regression spline metamodels in response prediction, we answered Research Question 4.1; our study shows that regression spline metamodels (specifically, MARS in multi-dimensional cases and univariate regression splines in one-dimensional problems) are more robust to fluctuations on the

response surface than kriging metamodels do. Studies of RS metamodels and different types of kriging metamodels in engineering design are done in (Simpson, 1998) and (Lin, 2000). Research Question 4.2, *How to select different types of metamodels at different design stages?,* is also visited in this section and will be further explored in Section 5.4. Based on the study above, we recommend that: 1). At the beginning of metamodeling processes, we could use kriging metamodels only or both kriging and MARS metamodels; and 2). When more points are selected in a highly nonlinear design space, we should use MARS to model prediction errors and both kriging and MARS to model system responses in SEED. Both kriging and MARS metamodels have their own strong and weak aspects. Thus, in real-world applications we should develop metamodels with both techniques if possible; when we meet problems with one technique, we could always switch to the other one. The application of MARS with SEED is discussed in the next section.

5.3 UTILIZATION OF MARS METAMODELS IN THE SEQUENTIAL EXPLORATORY EXPERIMENTAL DESIGN METHOD

As discussed in Section 5.2, we suggest using MARS in SEED. Since the SEED method was initially developed with kriging metamodels, we need to do some small modifications to have it work smoothly with MARS, which is to be done in this section. The research questions to be visited in this section are R.Q. 2, *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?* and R.Q.4.2, *How to select different types of metamodels at*

different design stages? The utilization of MARS in SEED is discussed in Section 5.3.1, and then demonstrated with a single-variable function in Section 5.3.2.

5.3.1 Utilization of MARS in SEED

To use MARS in SEED brings no significant change to the sequential experimental design method as developed in Chapter 4. The flowchart of SEED remains the same as that in Figure 4.4. Similar to the description in Chapter 4, there are still two ways to formulate the modified covariance matrix (see Section 4.5.3), and mathematical formulations remain the same as Equations (4.27), (4.28), and (4.34).

Since we suggest using MARS to model the prediction errors, it should be noted that the MARS metamodel smoothes the data so the predicted values at data points may not be accurate, thus we may have non-zero predicted prediction errors at data points. This may bring problem when we use kriging metamodels for system responses and MARS for prediction errors – even though the actual prediction errors at data points are zero with kriging, the predicted prediction errors from MARS may be different. Usually this difference is not large, and it should not affect the SEED process a lot. To be safe, careful examinations of this difference are recommended in the SEED process; the cost of this examination is negligible since only simple comparisons are involved.

Another important thing is the selection of values of θ in identifying new points when we use MARS metamodels for system responses. When we use kriging metamodels for system responses (as in Chapter 4), values of θ could be used in the formulation of the covariance matrix (see Equations (4.29) and (4.34)). In multi-variable

problems, this approach helps us identify highly nonlinear dimensions (with large θ values); more new points will be automatically put in these dimensions with SEED as discussed in Chapter 4. When MARS is used to model system responses, it provides no guidance on the selection of θ values for future sampling. In such cases, we could use a universal θ value for all dimensions; usually we set θ as 10 (or larger values) to ensure a rapid decaying correlation between points. In cases where there are already too many data points in the design space, we may need to set extremely large θ values (e.g., 1000) to ensure that the correlation decays fast enough and new points could be identified through maximum entropy sampling (small θ values may result in negative values of determinants of the covariance matrices, which implies that it is not worthwhile to add in new points). Besides the selection of θ values, the selection of values for λ and e_{max} is also very important; this is introduced in Chapter 4, and will be further discussed in this section after the application of MARS and SEED in developing metamodels for one single-variable function; similar to our study in last section, in this single-variable example, we are actually using the univariate quintic regression splines instead of MARS.

Since kriging has some desirable properties (loyal to data, providing guidance on identification of new points, etc.), we may want to keep using kriging to model system responses until it is necessary to switch to MARS. It may be helpful to develop both kriging and MARS metamodels for response surfaces with same data in the design process. Besides the comparison between previous and current metamodels, the comparison between kriging and MARS metamodels could also help identify possible

problems in metamodeling as discussed in Section 5.2. Once such a problem is identified, we will start to use MARS to replace kriging in future metamodeling stages. Designers may also use MARS to model system responses at the very beginning of SEED, as we will illustrate in Section 5.3.2 with the example of a single-variable function.

5.3.2 Example: A Single-Variable Function

In this section we illustrate and study the usage of regression splines in SEED with the single-variable example as used in Section 5.2. In this example, we follow the same steps as presented in Figure 4.4. The method used to formulate the adjusted covariance matrix is as described in Section 4.5.3.1, in which we adjust the covariance matrix without modifying the correlation function. Equations (4.27) and (4.28) are used in the formulation. In this sequential experimental design, we plan to use 4 data points and 5 validation points as initial design, and add in 2 new validation and 2 data points each time. We will stop with total 17 data points, i.e., after 3 iterations. Similar to the example in Chapter 4, metamodel accuracy is not used as the stopping criteria, thus no metamodel validation is needed in Step 4 during the sequential experimental design process. In this example, we decide to use kriging and regression splines to model system responses and regression splines to model prediction errors.

Iteration I – Step 1: Initial Experimental Design. As discussed in Chapter 4, there are many ways to design the initial experiments in SEED. In this example, we

decide to use the single-stage method by Currin, et al. (1991). The initial data points are listed in Table 5.8.

Table 5.8 Initial Experimental Design – 4 Data Points

x	0.0	0.331	0.669	1.0
y	0.0	0.6508	0.8199	0.8617

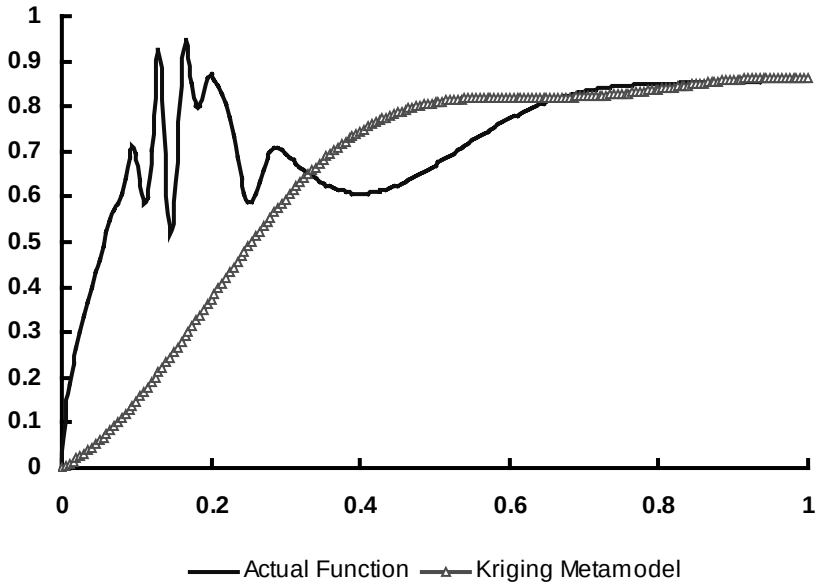


Figure 5.25 Initial Kriging Metamodel with 4 Data Points

Iteration I – Step 2: Simulation and Initial Metamodel of Responses. The corresponding kriging metamodel of system responses is illustrated in Figure 5.25; the value of θ for this metamodel is 7.83290.

Iteration I – Step 3: Identification of New Validation Points. In this step, we only have information from four data points and the initial metamodel developed in *Step 2*. We need to identify validation points for the first iteration. Similar to that in Chapter

4, when no enough information is available, we will use standard maximum entropy sampling to identify validation points. In this step we decide to identify n_{error} = 5 validation points.

A 9×9 covariance matrix is constructed with the first 4 rows and columns corresponding to the 4 data points in Table 5.8. In the formulation of covariance matrices, we set θ = 20, which yields a rapid decaying correlation between points. By maximizing the determinant of this 9×9 covariance matrix we identify 5 validation points as listed in Table 5.9. Six optimization algorithms (same as those used in Chapter 4) are used in iSIGHT to ensure the achievement of global optimum.

Table 5.9 Five New Validation Points in Iteration I

$\boldsymbol{x}$	0.091	0.215	0.5	0.785	0.909
$\boldsymbol{y}_{pred}$	0.1301	0.4091	0.8088	0.8344	0.8604
$\boldsymbol{y}_{actual}$	0.6951	0.8262	0.6700	0.8494	0.8545

Iteration II – Step 4: Metamodel of Prediction Errors. In this step, a metamodel of prediction errors of the kriging metamodel (Figure 5.25) is developed with information from 4 data points and 5 validation points. Prediction errors at these points are listed in Table 5.10. As described in Sections 5.2 and 5.3.1, a regression spline metamodel is developed based on the information in Table 5.10, and the plot of predicted prediction error y_{error} vs. x is drawn in Figure 5.26. The maximum absolute predicted prediction error is about 0.6.

Table 5.10 Prediction Errors at 4 Data Points and 5 Validation Points

x	0.0	0.331	0.669	1.0	
y_{error}	0.0	0.0	0.0	0.0	
x	0.091	0.215	0.5	0.785	0.909
y_{error}	-0.565	-0.4171	0.1388	-0.015	0.0059

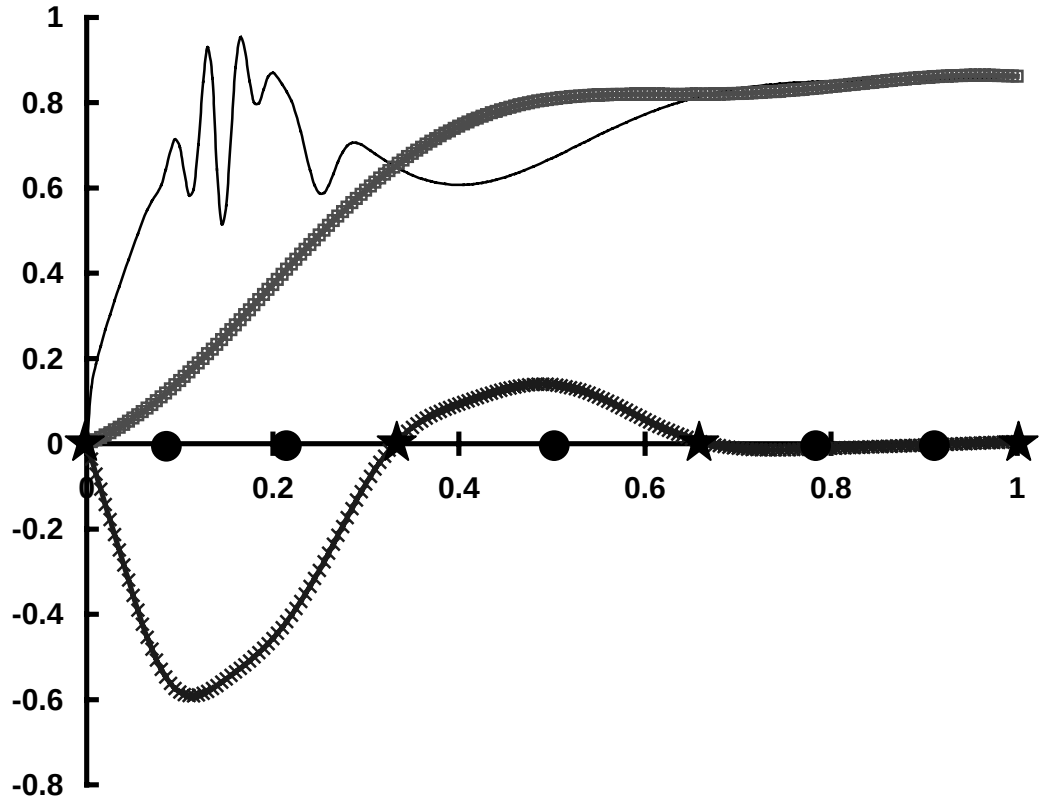


Figure 5.26 Metamodel of Prediction Errors in Iteration I

In Figure 5.26 we see that given the 9 points spreading over the whole design space, the regression spline metamodel grasps the prediction error very well. Peaks and bottoms in [0.09, 0.3] are not precisely captured because we have no enough information

in this region. In the next step, the regression spline metamodel of prediction errors as illustrated in Figure 5.26 will be used in the formulation of the covariance matrix.

Iteration I – Step 5: Metamodel Validation. Similar to the example in Chapter 4, the step of metamodel validation is skipped here. New data points are to be added.

Iteration I – Step 6: Formulation of the Adjusted Covariance Matrix. As introduced before, we plan to add in n_{new} = 2 new data points each time. In this step, entries of the 6×6 adjusted covariance matrix are calculated following Formulation I of the SEED method as described in Section 4.5.3.1. The key equations here are Equations (4.27) and (4.28).

A 6×6 covariance matrix is first built following Equations (4.19) and (4.20), with the first 4 rows and columns corresponding to the six data points that we already have, and the rest 2 rows and columns representing new data points. The value of θ is set to be 7.8329. All processes in this step are similar to those in Chapter 4; the only difference is that we use a C program to calculate predicted prediction errors, e_i, at candidate points with the regression spline metamodel in Figure 5.26 (instead of the FORTRAN program for kriging metamodels).

Iteration I – Step 7: Identification of New Data Points. In this step, we identify two possible new data points through maximizing the determinant of the adjusted covariance matrix developed in *Iteration I – Step 6*, as listed in Table 5.11. This is done in iSIGHT with six optimization techniques as used in Chapter 4.

Since the possible new data points, $x = 0.101$ and $x = 0.503$, are very close to two of the validation points, $x = 0.091$ and $x = 0.5$, we decide not to collect information at $x = 0.101$ and $x = 0.503$; instead, we use $x = 0.091$ and $x = 0.5$ as new data points. This helps avoid clustering of data/validation points and ensures efficiency in sequential experiments.

Table 5.11 Two Possible New Data Points in Iteration I

x	0.101	0.503

Iteration I – Step 8: Updated Metamodel of Responses. Now we have 6 data points, as listed in Table 5.12. New metamodels are developed with information from these data points. The kriging metamodel is illustrated in Figure 5.27; the value of θ is 99.99981.

As a comparison, we develop a regression metamodel with the six data points in Table 5.12, as illustrated in Figure 5.28. Comparing Figure 5.27 and Figure 5.28 we see that the regression spline metamodel performs better than the kriging metamodel. It is apparent that the kriging metamodel in Figure 5.27 does not predict the response surface in [0.6,1] well. In real-world applications, with information from only data points, we may not be able to tell which metamodel is more accurate because we do not know whether the bell-shape curve in [0.6,1] in Figure 5.27 reflects the actual surface or not. However, with information from the validation points, we could figure out which model is better. In this case, based on available information at $x = 0.785$ and $x = 0.909$, we

figure out that the unusual bell-shape curve in [0.6,1] of the kriging metamodel is far from reality; further inspection shows that this unusual bell-shape curve is actually from the large value of θ, which inherits from the highly nonlinear surface in regions with small x values. This problem is similar to the one we discussed in Section 5.2. Since kriging meets difficulty in modeling the object surface, we decide to use regression splines to model both responses and prediction errors in later stages of experimental design.

After finishing *Step 8*, following the flow chart in Figure 4.4, we will go to *Step 3* of Iteration II to add in new validation points to test the current metamodels. The regression spline metamodel in Figure 5.28 will be used in the next iteration.

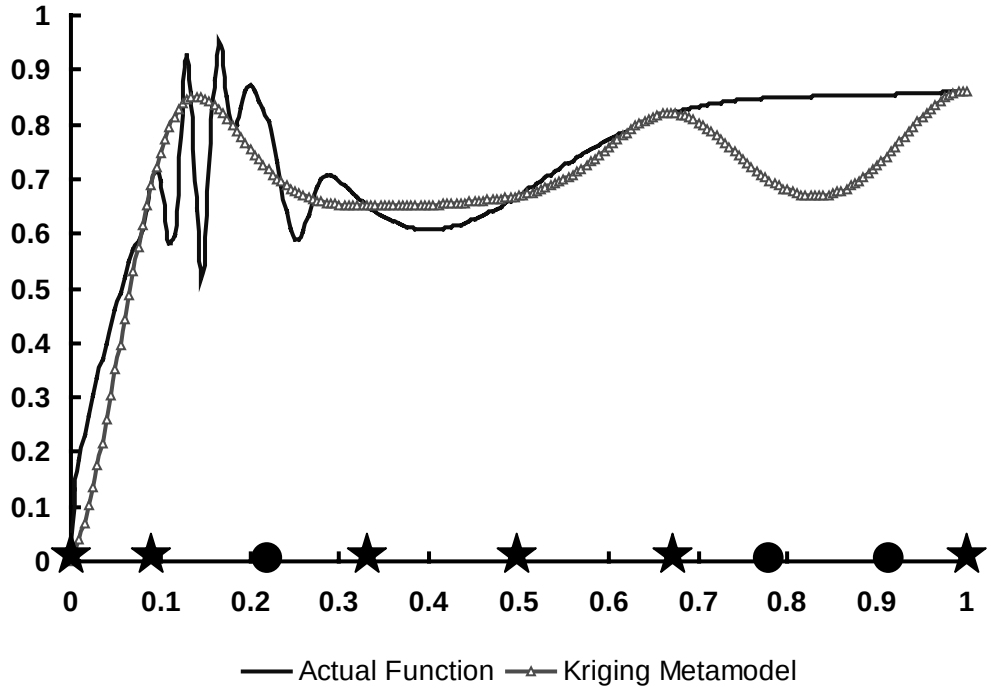


Figure 5.27 Kriging Metamodel with 6 Data Points

Table 5.12 Six Data Points

x	*y*	*x*	*y*	*x*	*y*
0.0	0.0	0.669	0.8199	0.091	0.6951
0.331	0.6508	1.0	0.8617	0.5	0.6700

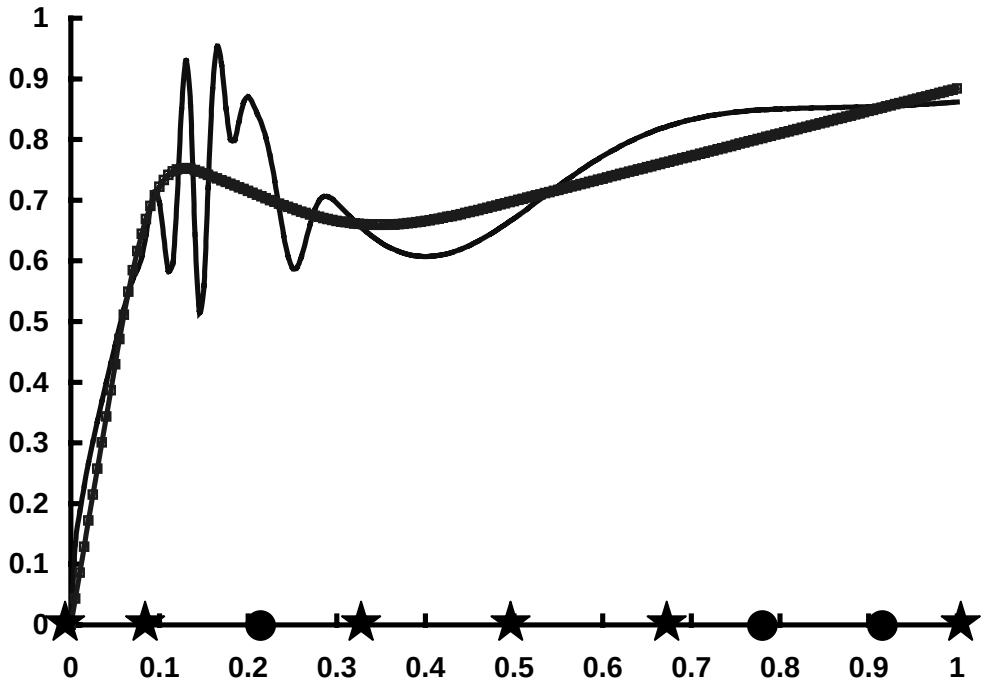


Figure 5.28 Regression Spline Metamodel with 6 Data Points

Iteration II – Step 3: Identification of New Validation Points. Now we have 6 data points and 3 validation points. In this step we decide to add in $n_{new} = 4$ validation points in order to have as many validation points as data points. To identify new validation points, a kriging metamodel of response is first developed with 3 validation points and illustrated in Figure 5.29. Prediction errors of this metamodel at 6 data points are calculated and listed in Table 5.13; a regression spline metamodel of prediction errors

are then developed and illustrated in Figure 5.30. Using the method similar to *Iteration I – Step 6* to *Step 8*, we identify 4 new validation points as listed in Table 5.14.

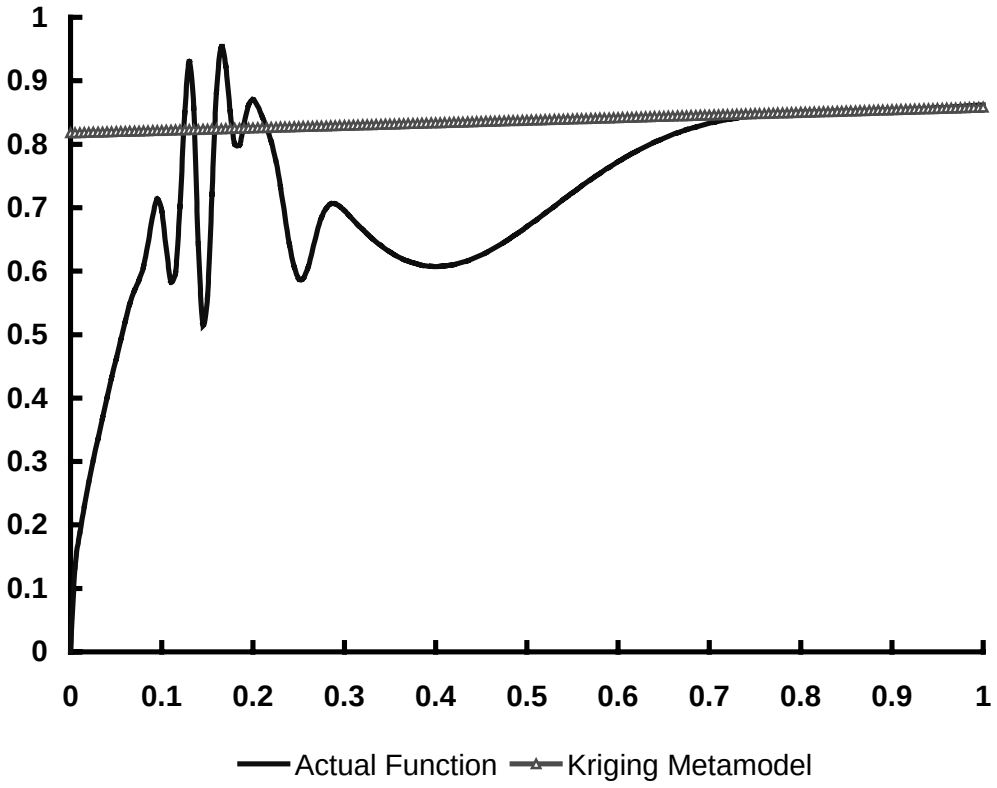


Figure 5.29 Metamodel of Responses Developed in Iteration II – Step 3

Table 5.13 Prediction Errors at 6 Data Points in Iteration II – Step 3

x	0	0.331	0.669	1	0.091	0.5
y_{pred}	0.8178	0.8308	0.8446	0.8582	0.8213	0.8376
y_{actual}	0	0.6508	0.8199	0.8617	0.6951	0.67
y_{error}	0.8178	0.1800	0.0247	-0.0035	0.1262	0.1676

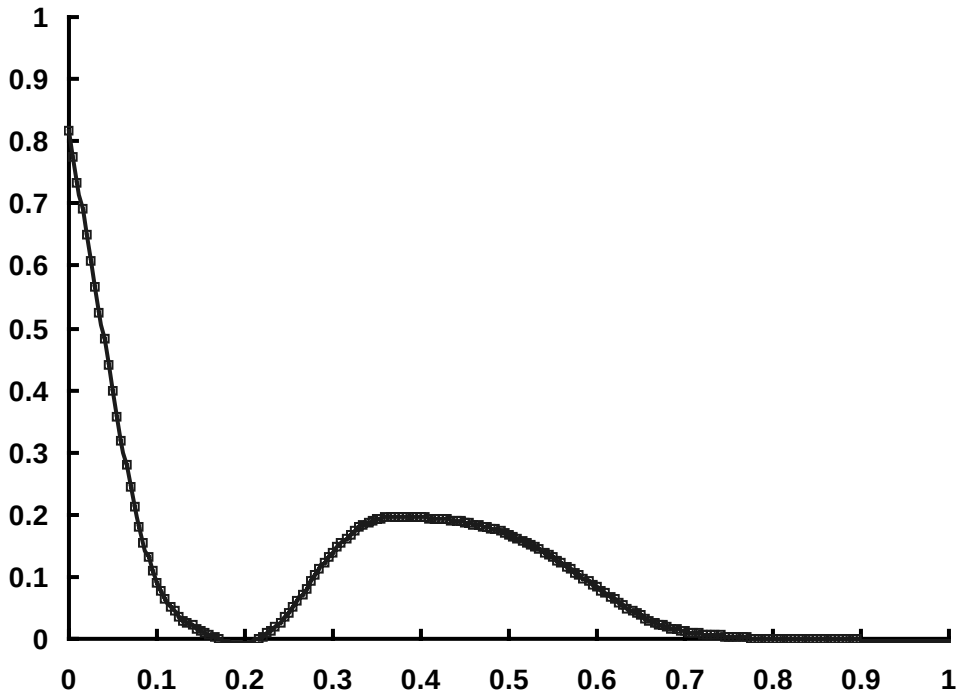


Figure 5.30 Regression Spline Metamodel of Prediction Errors in Iteration II – Step 3

Table 5.14 New Validation Points Added in Iteration II

x	0.026	0.289	0.414	0.582
y_{actual}	0.3091	0.7063	0.6087	0.7560

Iteration II – Step 4: Metamodel of Prediction Errors. Prediction errors at 7 validation points and 6 data points are listed in Table 5.15. A regression spline metamodel of prediction errors is built with information from these 13 points, and illustrated in Figure 5.31. The maximum absolute predicted prediction error is $e_{max} \approx 0.20$.

Table 5.15 Prediction Errors at Observed Points in Iteration II – Step 4

x	0.000	0.091	0.331	0.500	0.669	1.000	
y_{pred}	0.0001	0.6940	0.6600	0.6984	0.7611	0.8839	
y_{actual}	0.0000	0.6951	0.6508	0.6700	0.8199	0.8617	
y_{err}	0.0001	-0.0011	0.0092	0.0284	-0.0588	0.0222	
x	0.026	0.215	0.289	0.414	0.582	0.785	0.909
y_{pred}	0.2233	0.7061	0.6690	0.6691	0.7288	0.8041	0.8501
y_{actual}	0.3091	0.8262	0.7063	0.6087	0.7560	0.8494	0.8545
y_{err}	-0.0858	-0.1201	-0.0373	0.0604	-0.0272	-0.0453	-0.0044

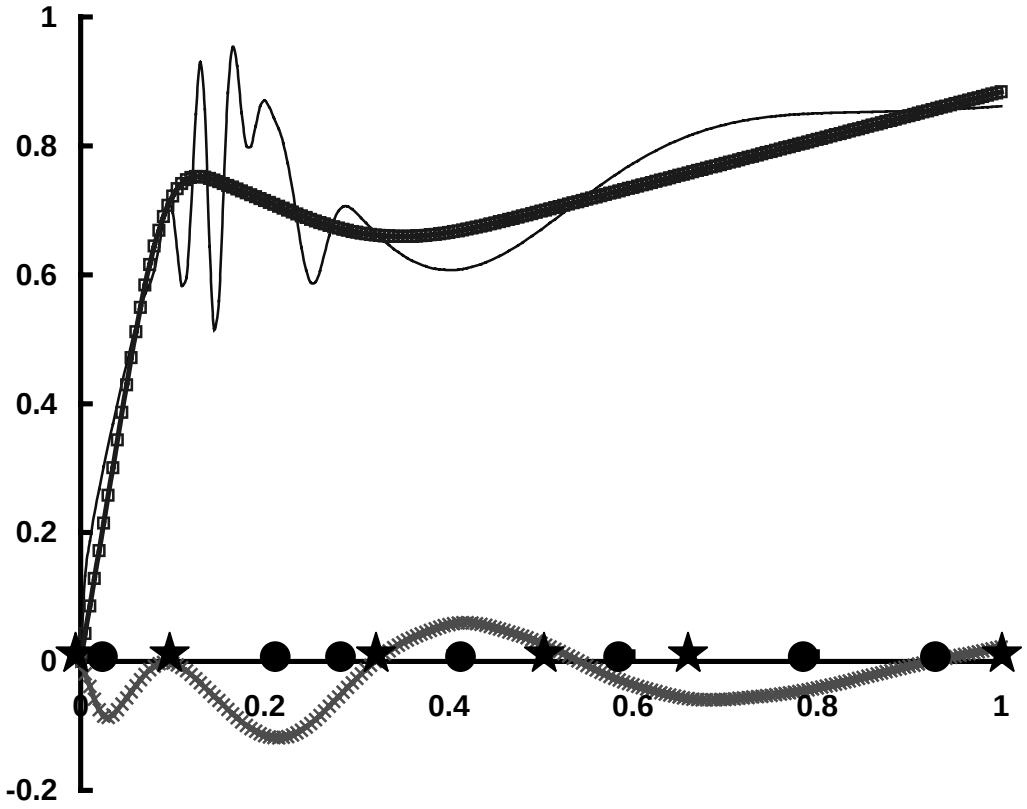


Figure 5.31 Regression Spline Metamodel of Prediction Errors in Iteration II

Iteration II – Step 5: Metamodel Validation. As described before, this step is skipped and we proceed to *Step 6*.

Iteration II – Step 6: Formulation of the Adjusted Covariance Matrix. We need to add in n_{new} = 2 new data points, thus we formulate an 8×8 covariance matrix following the method as used in *Iteration I – Step 6*. The first 6 rows and columns correspond to previous data points, and the rest 2 rows and columns representing new data points. The value of θ is set to be 100.0, which is the limit of Simpson's kriging code.

Iteration II – Step 7: Identification of New Data Points. By maximizing the determinant of the adjusted covariance matrix as built in *Iteration II – Step 6*, we are able to identify 2 possible new data points at x =0.213 and x = 0.833. Since one of the possible new data points, x = 0.213, is very close to one of the validation points, x = 0.215, we decide to use x = 0.215 instead of x = 0.213 as the new data point. New data points added in this step are listed in Table 5.16.

Table 5.16 New Data Points Added in Iteration II

x	0.215	0.833
y	0.8262	0.8519

Iteration II – Step 8: Updated Metamodel of Responses. A new regression spline metamodel is developed with information from the 8 data points as listed in Table 5.17. The regression spline metamodel for responses is illustrated in Figure 5.32.

Table 5.17 Eight Data Points in Iteration II

x	*y*	*x*	*y*
0	0.0000	0.5	0.6700
0.091	0.6951	0.669	0.8199
0.215	0.8262	0.833	0.8519
0.331	0.6508	1	0.8617

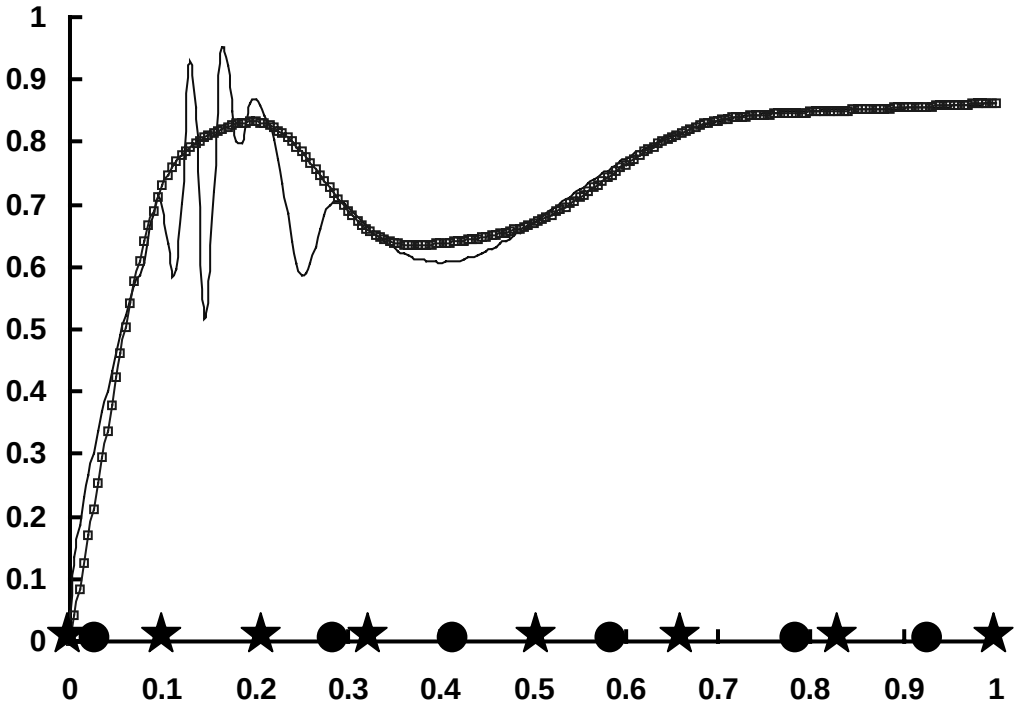


Figure 5.32 Regression Spline Metamodel with 8 Data Points

Iteration III – Step 3: Identification of New Validation Points. Now we have 8 data points and 6 validation points. In this step, we need to add in 3 new validation points. We build a regression spline metamodel of responses with 6 validation points; this metamodel is illustrated in Figure 5.33. Then prediction errors of this metamodel at

8 data points and 6 validation points are calculated and listed in Table 5.18. A regression spline metamodel of prediction errors is then developed and plotted in Figure 5.34.

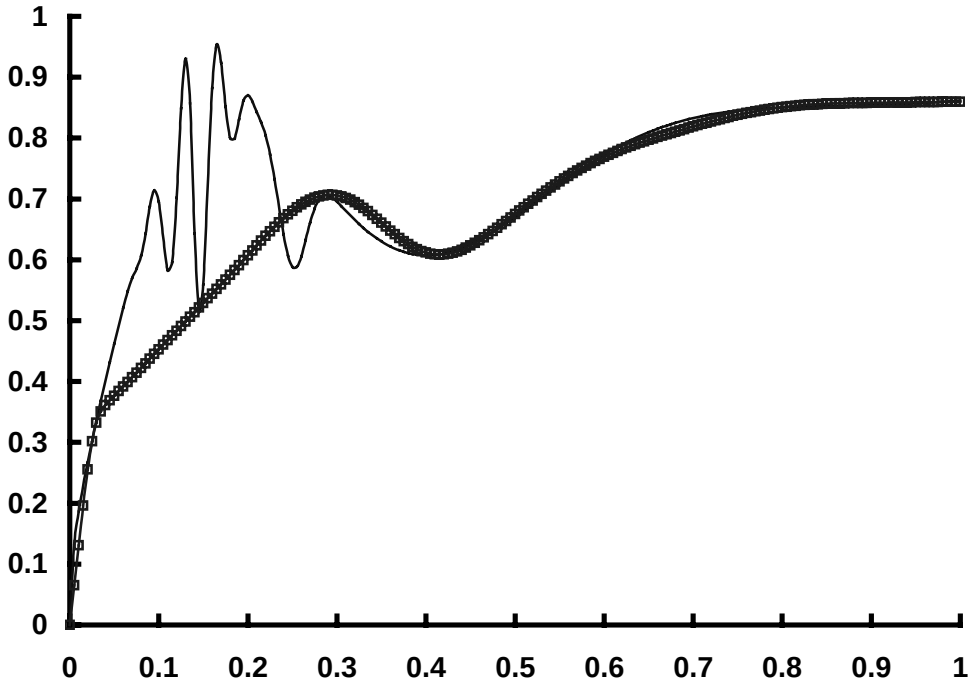


Figure 5.33 Regression Spline Metamodel of Responses Developed in Iteration III – Step 3

Table 5.18 Prediction Errors at Observed Points in Iteration III – Step 3

x	0.000	0.091	0.215	0.331	0.500	0.669	0.833	1.000
y_{pred}	0.2663	0.4162	0.6243	0.6844	0.6800	0.8026	0.8450	0.8879
y_{actual}	0.0000	0.6951	0.8262	0.6508	0.6700	0.8199	0.8519	0.8617
y_{err}	0.2663	-0.2789	-0.2019	0.0336	0.0100	-0.0173	-0.0069	0.0262
x	0.026	0.289	0.414	0.582	0.785	0.909		
y_{pred}	0.3091	0.7064	0.6077	0.7636	0.8326	0.8645		
y_{actual}	0.3091	0.7063	0.6087	0.7560	0.8494	0.8545		
y_{err}	0.0000	0.0001	-0.0010	0.0076	-0.0168	0.0100		

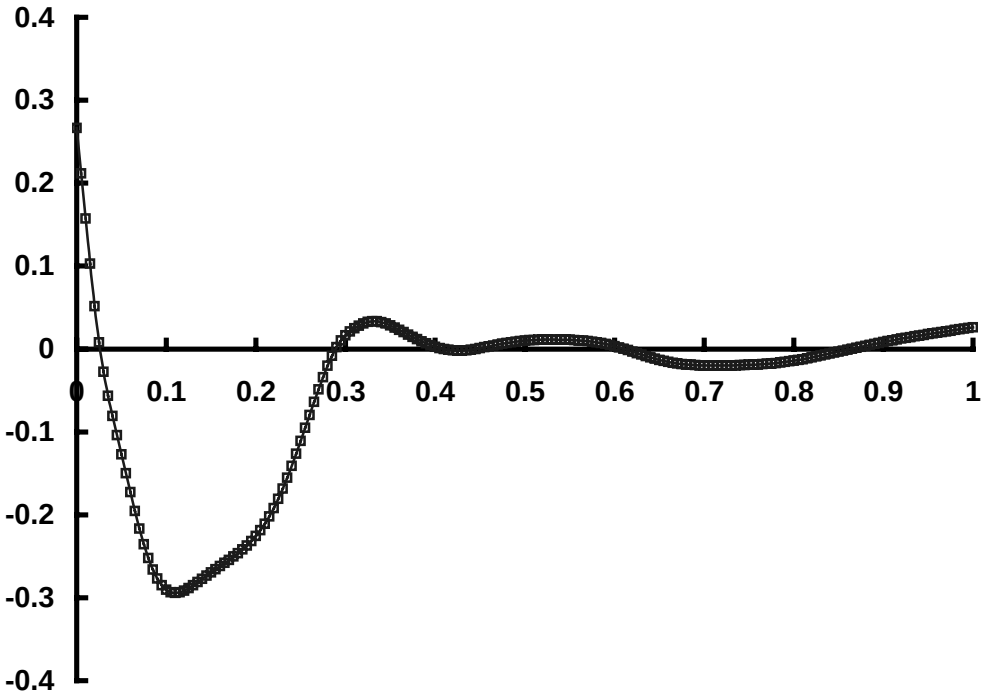


Figure 5.34 Regression Spline Metamodel of Prediction Errors in Iteration III – Step 3

Following similar approach used in *SEED – Steps 6 to 8*, three possible validation points are identified at $x = 0.0$, $x = 0.149$, and $x = 1.0$. Since two of the possible validation points are previously observed as data points, we need to redo the identification of new validation points because we should not convert data points to validation points in the SEED process. A new regression spline metamodel of responses is developed with 6 validation points and 2 validation points ($x = 0.0$ and $x = 1.0$); this metamodel is illustrated in Figure 5.35. Then prediction errors of this metamodel at 8 other data points and 6 validation points are calculated and listed in Table 5.19. A

regression spline metamodel of prediction errors is then developed and plotted in Figure 5.36. Following same processes as in *Iteration II – Step 6* to *Step 8*, three new validation points are identified and listed in Table 5.20.

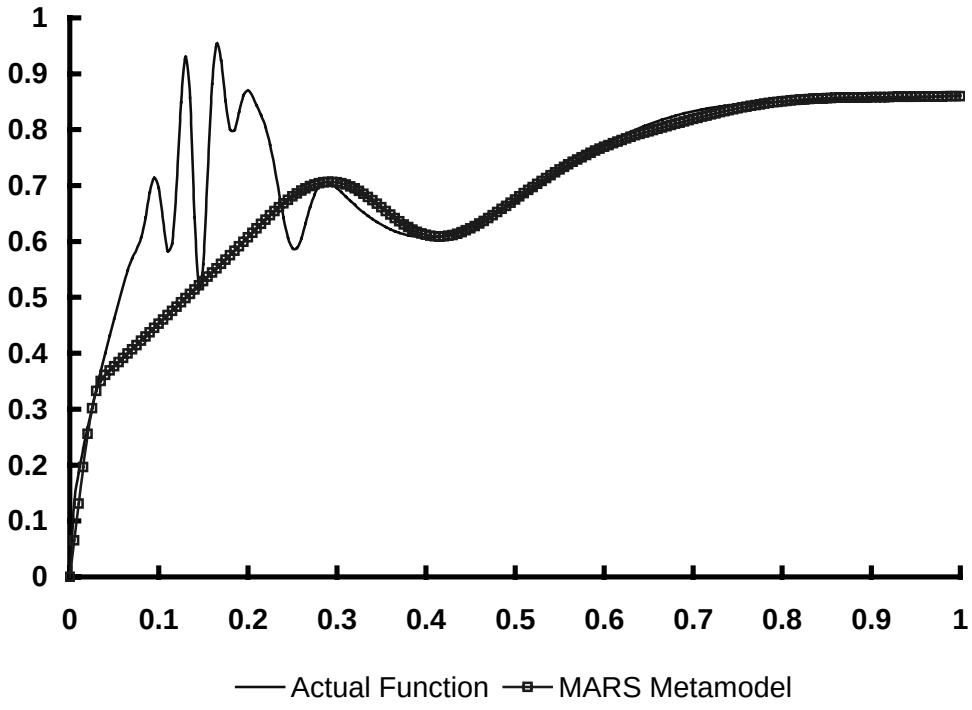


Figure 5.35 New Regression Spline Metamodel of Responses in Iteration III – Step 3

Table 5.19 New Prediction Errors at Observed Points in Iteration III – Step 3

x	0.000	0.091	0.215	0.331	0.500	0.669	0.833	1.000
y_{pred}	0.0000	0.4390	0.6312	0.6841	0.6750	0.8056	0.8550	0.8598
y_{actual}	0.0000	0.6951	0.8262	0.6508	0.6700	0.8199	0.8519	0.8617
y_{err}	0.0000	-0.2561	-0.1950	0.0333	0.0050	-0.0143	0.0031	-0.0019
x	0.026	0.289	0.414	0.582	0.785	0.909		
y_{pred}	0.3091	0.7063	0.6087	0.7562	0.8475	0.8580		
y_{actual}	0.3091	0.7063	0.6087	0.7560	0.8494	0.8545		
y_{err}	0.0000	0.0000	0.0000	0.0002	-0.0019	0.0035		

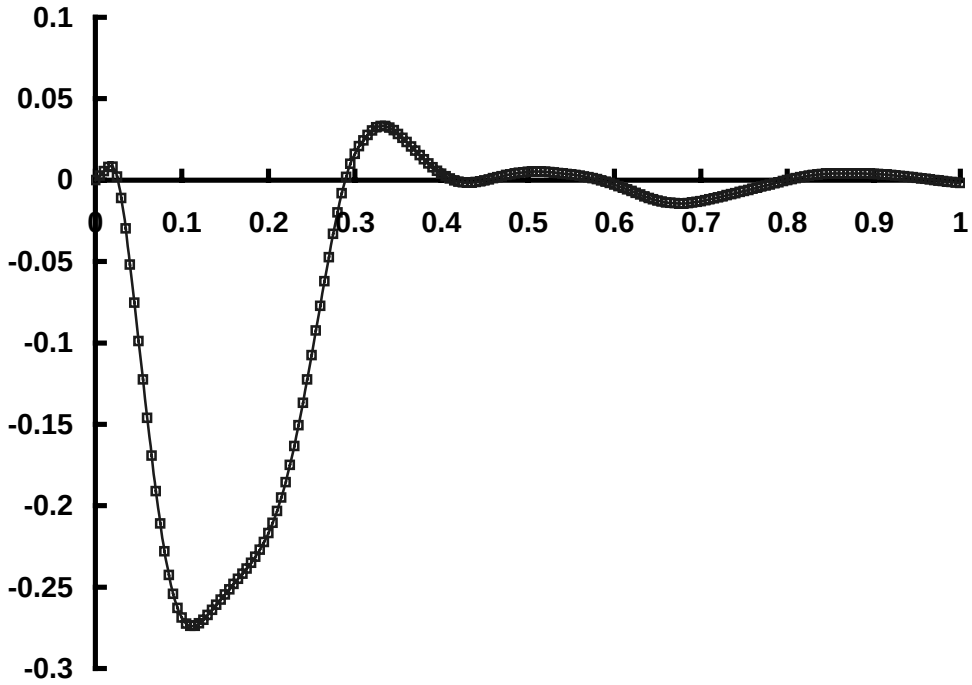


Figure 5.36 New Regression Spline Metamodel of Prediction Errors in Iteration III – Step 3

Table 5.20 New Validation Points Added in Iteration III

x	0.071	0.151	0.243
y	0.5732	0.5857	0.6193

Iteration III – Step 4: Metamodel of Prediction Errors. Now we have 8 data points and 9 validation points. In this step, prediction errors at both data and validation points are used to develop a regression spline metamodel to predict prediction errors in the design space. The observed prediction errors are listed in Table 5.21, and the

corresponding metamodel is illustrated in Figure 5.37. The maximum absolute predicted prediction error is $e_{max} \approx 0.06$.

Table 5.21 Prediction Errors at Data and Validation Points

x	y_{pred}	y_{actual}	y_{err}	x	y_{pred}	y_{actual}	y_{err}
0	0.0000	0	0.0000	0.026	0.2193	0.3091	-0.0898
0.091	0.6951	0.6951	0.0000	0.071	0.5833	0.5732	0.0101
0.215	0.8262	0.8262	0.0000	0.151	0.8107	0.5857	0.2250
0.331	0.6508	0.6508	0.0000	0.243	0.7956	0.6193	0.1763
0.5	0.6699	0.67	-0.0001	0.289	0.7094	0.7063	0.0031
0.669	0.8208	0.8199	0.0009	0.414	0.6398	0.6087	0.0311
0.833	0.8504	0.8519	-0.0015	0.582	0.7444	0.7560	-0.0116
1	0.8624	0.8617	0.0007	0.785	0.8469	0.8494	-0.0025
				0.909	0.8558	0.8545	0.0013

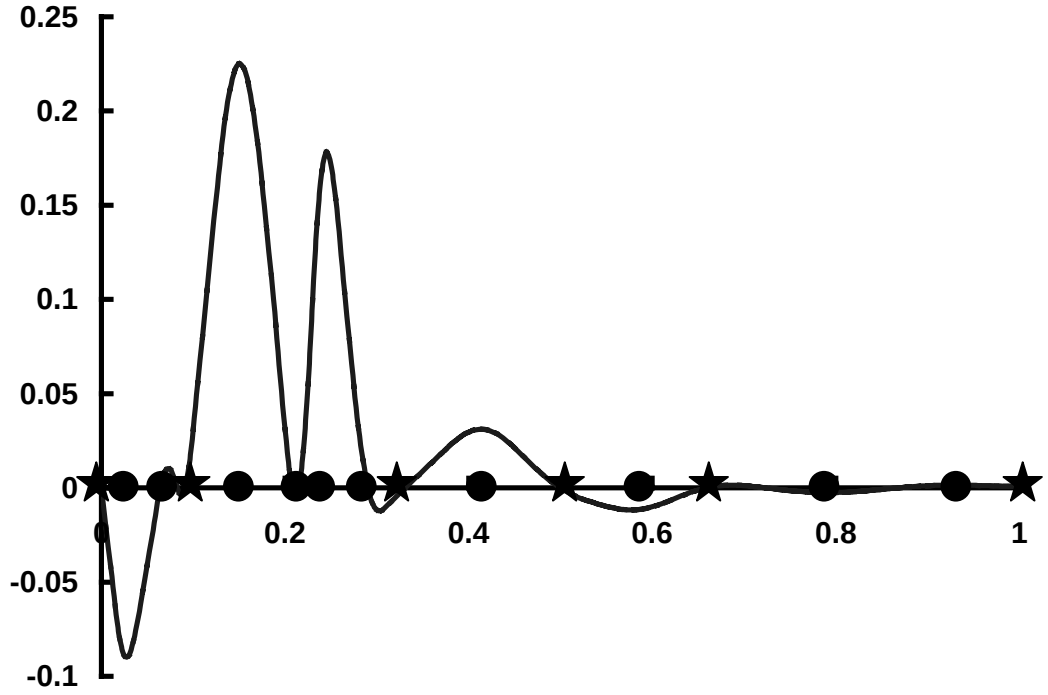


Figure 5.37 Regression Spline Metamodel of Prediction Errors in Iteration III

Iteration III – Step 5: Metamodel Validation. This step is skipped.

Iteration III – Step 6: Formulation of the Adjusted Covariance Matrix. We plan to add in $n_{new} = 2$ new data points. Since after this iteration we plan to get 19 points and stop the SEED process, in this step we will consider the correlation between candidate points and all observed points; note this is different from what we did in Iteration II – Step 6 in which we only considered the correlation between candidate points and data points. To achieve this, we build a 19×19 covariance matrix with the first 8 rows and columns corresponding to the current data points, the 9^{th} to 17^{th} rows and columns corresponding to the validation points, and the last 2 rows and columns corresponding to new data points. The value of θ is set to be 100.0.

Iteration III – Step 7: Identification of New Data Points. By maximizing the determinant of the adjusted covariance matrix as developed in *Iteration III – Step 6*, two new data points are identified and listed in Table 5.22.

Table 5.22 Possible New Data Points in Iteration III

x	0.126	0.254
y	0.8743	0.5871

Iteration III – Step 8: Updated Metamodel of Responses. Now we have 10 data points and 9 validation points as listed in Table 5.23. As stated at the beginning of this section, we stop the SEED process since we have observed 19 points. A final regression spline metamodel of responses is developed and illustrated in Figure 5.38.

Table 5.23 Nineteen Observed Points

x	*y*	*x*	*y*
0	0.0000	0.026	0.3091
0.091	0.6951	0.071	0.5732
0.126	0.8743	0.151	0.5857
0.215	0.8262	0.243	0.6193
0.254	0.5871	0.289	0.7063
0.331	0.6508	0.414	0.6087
0.5	0.6700	0.582	0.7560
0.669	0.8199	0.785	0.8494
0.833	0.8519	0.909	0.8545
1	0.8617		

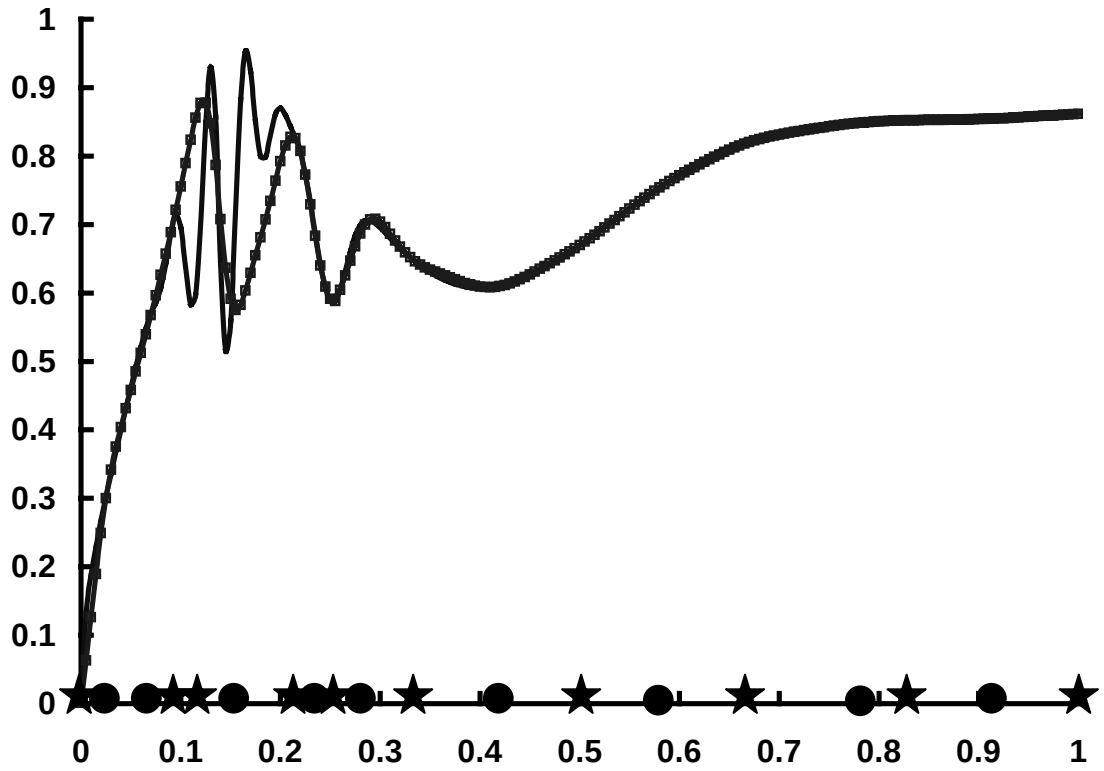


Figure 5.38 Regression Spline Metamodel of Responses with 16 Data Points

As a comparison, a regression spline metamodel of responses is developed with information from 19 evenly-spread data points in [0, 1]. This metamodel is illustrated in Figure 5.39. Comparing the regression spline metamodel in Figure 5.38 and the one in Figure 5.39, we see that in the single-variable example, using the SEED method, we are able to develop a more accurate regression spline metamodel with the same number of data points. More detailed discussions are presented in Section 5.3.3.

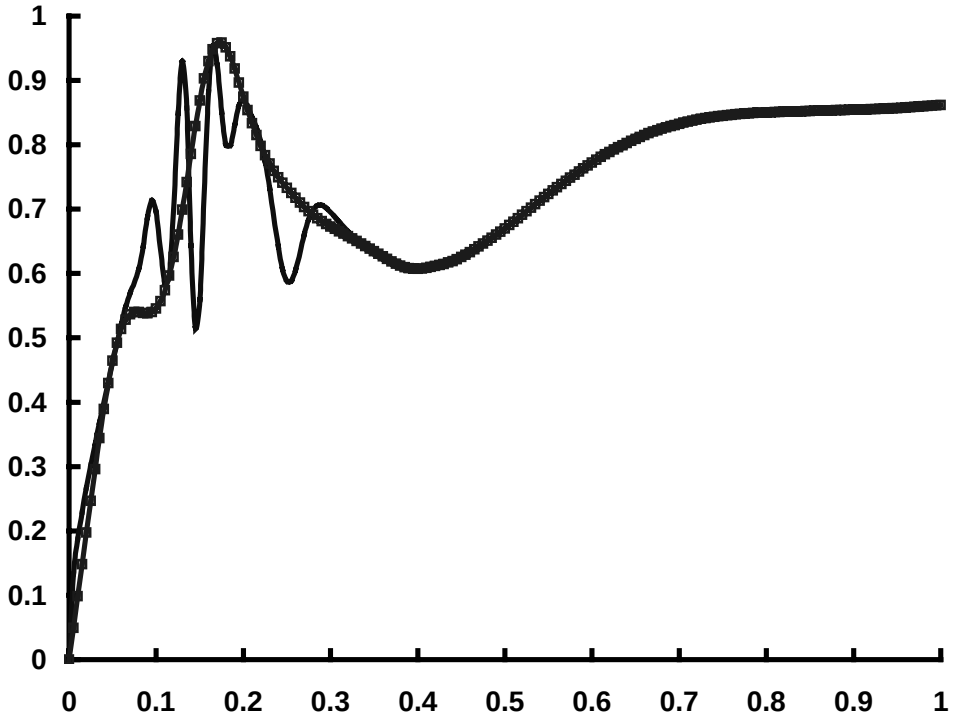


Figure 5.39 Regression Spline Metamodel of Responses with 19 Evenly-Spread Data Points

5.3.3 Discussions on Applications of the SEED method

In this section, we further explored the application of the SEED method with MARS (to be specific, it is actually univariate quintic regression splines in this example) metamodels; this helped answer R.Q.2, *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?* and R.Q.4.2, *How to select different types of metamodels at different design stages?* Our studies show that the SEED method is effective in allocating data points sequentially to obtain an acceptable metamodel; the usage of both kriging and regression spline metamodels provides sufficient flexibility in metamodeling. In Figure 5.38 we see that with only 19 points we are able to grasp the high nonlinearity in [0.2,0.4] satisfactorily. Data points are added in regions where large prediction errors exist; in this case, it is near the lower band of the design space, i.e., [0.09,0.4].

In SEED, both kriging and regression splines could be used to develop metamodels for system responses. Usually we use kriging metamodels in very early stages; regression spline metamodels could be developed to test whether there is abnormal behavior in kriging metamodeling (as discussed in Section 5.2). When kriging meets difficulty in modeling, we should use regression spline metamodels in future stages of metamodeling. To build metamodels for prediction errors, we suggest using regression spline metamodels, as explained with Figure 5.20 and Figure 5.21 in Section 5.2.

The usage of two groups of points (data points and validation points) is very important in SEED. In Section 5.3.2, finally we got a better metamodel with 19 data points (compared to the metamodel with 19 evenly-spread data points) with the SEED

method. In SEED, data and validation points are added sequentially and the information from previous points are used to guide the allocation of future data and validation points. New points are added to decrease information uncertainty and thus should be in regions with large prediction errors. In this way we are able to maximally utilize the available resources and save the computation expense on some expensive computer simulations.

Besides sequential experiments and metamodeling, another option to develop metamodels is to conduct parallel simulation. For example, one may run computer simulation simultaneously on many computers; in this way, large amount of information could be achieved by observing system responses at many data points. In this sense, the parallel computing strategy seems superior to SEED because it is simpler. However, in real-world applications, sequential experiments and metamodeling is necessary because we may not have enough resources. To apply the parallel computation strategy, one may need to have a lot of computers running at the same time – and usually one computer could only afford one simulation (or a few) because the simulation may occupy a lot of resource (memory, CPU time, etc.) in the computer. For example, in a simple industrial case that has 8 design variables (see the vehicle body structural design in Lin, 2000), the designers may need to run at least 64 simulations to develop the metamodels, and maybe another 64 to validate them. To have 64 computer running at the same time may be difficult even in large laboratories. To solve this problem, designers may want to do the experiments sequentially, e.g., run 8 simulations simultaneously at one time and conduct 8 iterations; and this is where SEED is useful – it provides guidance on how to identify future data points in sequential simulations.

Another possibility is to extend the usage of SEED in physical experiments. In some cases we could only do physical experiments because computer simulations are not available. In other cases, computer simulations are used as references: they could give good estimations of system responses, while real-world experiments are needed to validate solutions obtained from computer simulations. Usually these physical experiments are expensive – not only computationally but also monetarily. Examples include crash experiments in designing vehicle bodies, some bio-system experiments, etc. In such cases, SEED could be applied and its advantage is apparent.

The discussion above is closely related to another topic – the cost of applying SEED. To apply SEED, in addition to the simulation expense, a lot of time and effort is spent on formulation of covariance matrices and search of maximum determinants of the matrices. In the formulation of covariance matrices, designers' decisions are involved and human behaviors occupy most time; in the search of maximum determinants, optimization algorithms are used and they usually require some time to get solutions. To minimize time and effort wasted on designers' decisions, a bunch of decision-support tools are to be incorporated into a computer framework, which could be done as we develop, verify, and improve the SEED method. This is a future work for this dissertation. To minimize computation time spent on optimization, we could adopt faster (though may be less effective) optimization algorithms, e.g., the "hiker" method used in (Currin, et al., 1991). It is expected that the application of SEED should be very inexpensive with all supporting tools are ready.

One drawback of SEED is the necessary human decisions in the formulation of adjusted covariance matrices. Actually this is partly from *D*-optimal design and maximum entropy sample, which is the basis of the SEED method: as introduced in Chapter 4, prior distributions are usually needed in such experimental designs. As discussed in Chapter 4 and our studies in this chapter, designers need to select values of θ, λ, and e_{max} in the formulation of the matrices. At the beginning of SEED, we usually set θ=25, λ=2, and e_{max} could be obtained with metamodels of prediction errors.

In cases where kriging metamodels work well, values of θ from previous kriging metamodels could be used in the formulation of covariance matrices in future stages. In cases where kriging meets difficulty, a large value of θ, e.g., θ=100, could be used in the formulation of covariance matrices. In a design space with a few points, when other factors (λ, and e_{max}, etc.) holding constant, as values of θ increases over a very large value, e.g., 100, solutions (new points) tend to spread over the design space instead of being in regions with large errors. This is because that large θ values represent rapid decaying correlations, thus in a design space with only a few points, most regions will be a "desert" with little correlations with current points; in such cases the effect of the adjustment based on prediction errors is usually negligible.

As discussed in Section 4.5.3.1, λ is used to gauge the balance between "spreading over the design space" and "being in regions with large prediction errors". When large values of λ are used, the adjustment based on prediction errors is small and new points tend to spread over the design space. When small values of λ are used, the

adjustment is large and new points tend to be in regions with great prediction errors. In very early experimental design stages, since we do not have many points in the design space and are not very confident on the prediction of prediction errors in the design space, we tend to add in new points that spread over the design space to avoid being misled by the inaccurate information. In later stages, as we have more accurate metamodel and confidence on the prediction of prediction errors, we could use small λ values, e.g., λ=1.5, to force new points to be added in regions with great predicted prediction errors.

The selection of e_{max} also affects the identification of new points. As pointed out in Chapter 4, it may be very difficult to get the exact global maximum absolute predicted prediction error e_{max} in a real-world application with many design variables and responses. When values of e_{max} are much larger than the actual one, the adjustment on the covariance matrices will be too small and thus new points tend to spread over the design space. When values of e_{max} are much smaller than the actual one, the adjustment on the covariance matrices will be too large and new points tend to spread over the design space too because too many candidate points in the design space are affected by this adjustment and those in regions with large prediction errors do not receive more attention compared with others. Usually we use a value of e_{max} that is a bit smaller than the actual maximum absolute prediction error, which generates small regions around points with large prediction errors; in the formulation of adjusted covariance matrices, points in these regions receive the same amount of adjustment. This allows more trade-off in identifying new points and helps avoid clustering of new points with current points, especially in selecting new validation points.

In this section, we revisited R.Q.2 and improved the SEED method by applying MARS in the metamodeling processes. The usage of different types of metamodels in SEED brings great advantage. In the next section, we will go further and explore the utilization of more types of metamodels, i.e., RS, kriging, and MARS, along the design timeline; this work will be closely related with the SEED method.

5.4 AN APPROACH FOR SEQUENTIAL METAMODELING ALONG THE DESIGN TIMELINE

In this section, we plan to answer Research Question 4, *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?* Only RS, kriging, and MARS metamodels are considered in our study in this section. To answer Research Question 4, we have done comparisons among RS, kriging, and MARS metamodels in Section 5.2 and previous studies (see, Simpson, 1998; Lin, 2000; Lin, et al., 2000). In this section, an approach is proposed to incorporate and utilize these metamodels sequentially in accordance with different requirements and goals in different stages of experimental design. The development of this approach also helps answer R.Q. 3.2, *How to reduce the design space with information from previous metamodeling and design space exploration?* This approach is illustrated with a simple engineering problem in Section 5.5.

In this dissertation we focus on the usage of three types of metamodels, the Response Surface (RS) model, kriging model, and Multivariate Adaptive Regression Splines (MARS). Fundamentals of these metamodels are presented in Chapter 2. The

comparison and usage of other types of metamodels in engineering design will be a future work for this dissertation. In (Simpson, 1998), the author compared the performance of RS and kriging metamodels in engineering design. In (Lin, 2000) the author studied the performance of RS and kriging metamodels in robust design. The comparison of various types of kriging metamodels could be found in (Simpson, 1998; Lin, 2000; Lin, et al., 2000). The usage of kriging and MARS metamodels in SEED is studied in Sections 5.2 and 5.3 in this dissertation. Based on previous studies, properties of these metamodels are listed and compared in Table 5.24. Items 1 – 3 in Table 5.24 correspond to the mathematical and computational complexity of metamodels, 4 – 6 corresponding to the accuracy (or the ability of prediction) of different metamodels, and 7 – 9 corresponding to metamodels' relationship with other techniques.

Table 5.24 Plus and Minus of Different Types of Metamodels

	RS (Regression)	**Kriging**	**MARS**
1. Mathematical complexity	Simple	Complicated	Complicated
2. Computation time	Short	Long	Medium
3. Problem size: # of design variables and # of data points	Large, Medium, and Small Problems	Small Problems	Medium and Small Problems
4. Metamodel accuracy	Low	High	High
5. Loyalty to data	No	Yes	No, with very small bias
6. Ability to model irregular surfaces (highly nonlinear or flat in different regions)	No	Yes, but only when with lots of data	Yes
7. Suitable for existing screening techniques	Yes	No	Yes
8. Preference to specific experimental designs	Yes	Yes	No
9. Mathematical connectivity to SEED (adapted maximum entropy sampling)	No	Yes	No

In Table 5.24 we see that the RS metamodel has very apparent advantages and drawbacks. Among the three types of metamodels, the RS model is easiest to develop; its mathematical foundation is simple and the computation time (on both model building and response prediction) is short. Since it is simple, its accuracy is not very satisfactory and it cannot model irregular surfaces that are highly nonlinear or flat in different regions in the design space. Usually the RS metamodels are developed with classical experiments, i.e., fractional factorial designs, CCD, etc. The usage of these experiments and the RS metamodel in the Response Surface Methodology (RSM) provides an effective approach to screen out unimportant design variables – though this technique is primarily suitable for physical experiments that come with random errors.

The kriging metamodel is most difficult to develop because it involves matrix calculations. This sacrifice on computation time enables kriging metamodels to predict response values accurately with sufficient data. One appealing property of the kriging metamodel used in this dissertation is that it is loyal to the existing data, which is suitable for metamodeling with deterministic computer experiments. Previous studies show that kriging works better with space-filling experiments than with classical experiments. Kriging and maximum entropy sampling (the basis of SEED) share the same mathematical foundation, which makes the application of kriging in SEED natural and easy. For example, in SEED, values of θ from previous kriging metamodels could be used in the formulation of covariance matrices in future maximum sampling steps. Major limitations of kriging are: 1) it can only deal with small problems because the computation time on both model building and response prediction increases dramatically

as the numbers of design variables and data points increase, and 2) it cannot model irregular surfaces well, as discussed in Section 5.2.

The MARS metamodel is mathematically complicated but does not require as much computation time as kriging because it does not require matrix calculations. Without strict computation constraints, it is able to deal with more design variables and data points than kriging. It smoothes the data, but the prediction errors at current data points are very small. Our studies show that it works well with both evenly and unevenly spread data points; this is attractive because in the SEED method data points tend not to be evenly spread. As studied in Section 5.2, MARS could model irregular response surfaces, which is also very attractive in metamodeling.

Since different types of metamodels all have their advantages and drawbacks, we propose to develop an approach in which these metamodels are used in different stages of experimental design so that we could take advantage from their strong points and avoid their shortcomings. The incorporation of kriging and MARS metamodels in SEED has already been studied in Sections 5.2 and 5.3, thus in this section, our focus is on the usage of RS metamodels in early stages of experimental design and its incorporation with kriging and MARS metamodels in SEED. Major advantages of the RS metamodel are its simplicity and ability of identifying unimportant design variables. Thus, in very early stages of sequential experimental design, classical experiments and RS metamodels could be used to help reduce the size of the problem by screening out unimportant design variables. As the experimental design evolves, more accurate metamodels are needed and we should use kriging and MARS metamodels to replace the RS metamodel.

The framework of sequential metamodeling is illustrated in Figure 5.40 and Figure 5.41 in different formats. The SEED method, which was presented in Figure 4.4, is treated as an integrated and independent processor in this framework of sequential metamodeling. The RS metamodel is not directly used in the SEED method; instead, it is used before we apply the SEED method to develop accurate metamodels. Thus in this approach the primary goal of using RS metamodels is to reduce the design space by decreasing the number of dimensions of the problem. At early stages of sequential metamodeling, we usually design fractional factorial experiments and develop first-order regression models (RS Metamodels) to gain knowledge of the actually response surface and eliminate unimportant design variables. Then we may augment more data points to construct CCD experiments and second-order RS metamodels may be developed to help grasp more details of the simulation program. We may also skip the development of second-order RS metamodels, going directly to Processor D, in which we apply the SEED method to get accurate kriging or MARS metamodels for system responses. The dash arrows between Processors B and D indicate that the SEED method should call the simulation program occasionally to collect information at new data/validation points in its iterations.

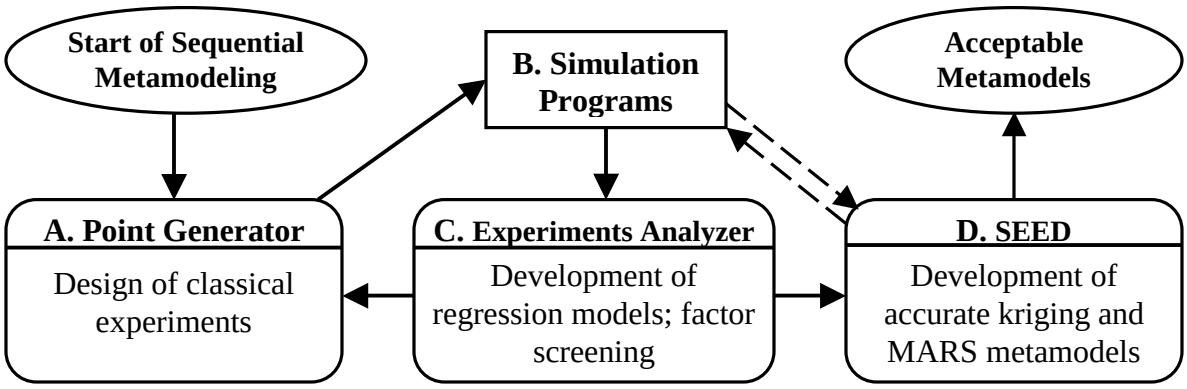


Figure 5.40 Framework of Sequential Metamodeling (I)

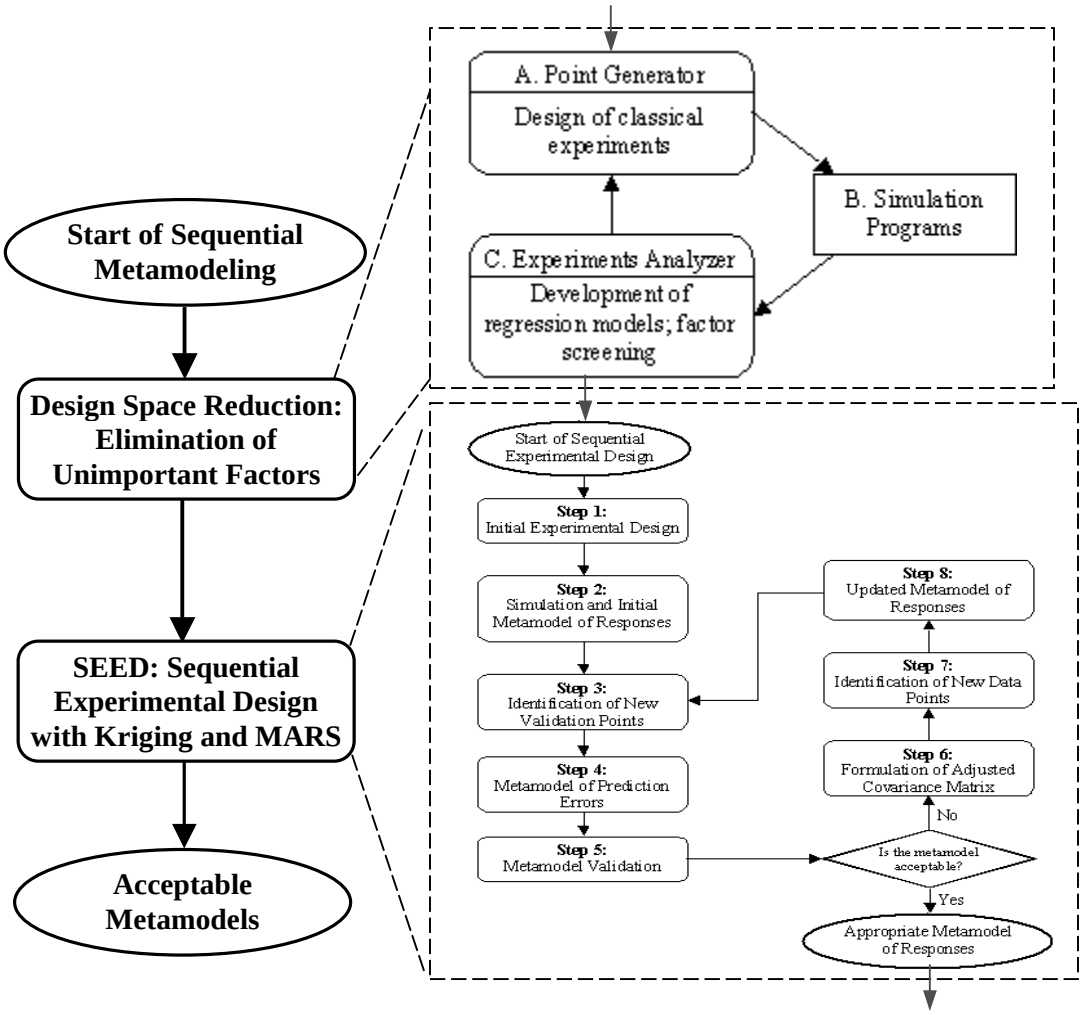


Figure 5.41 Framework of Sequential Metamodeling (II)

The approach for sequential metamodeling helps answer R.Q.4.2 by using different types of metamodels according to different requirements along the design timeline. R.Q.3.2 is also answered in that the RS metamodels are used to help reduce the design space by screening out unimportant design variables. To answer R.Q.3.2 completely, a future work is to develop approaches to reduce the ranges of the design variables.

The approach of sequential metamodeling is introduced and illustrated in Figure 5.40 and Figure 5.41 in this section. Note that in this study we do not consider multiple responses; the extension of this approach to multi-response problems is easy in cases where we have clear ideas on the relative importance of each response. In Section 5.4.2, we will apply this approach in an engineering problem. Further applications of this approach are to be presented in following chapters with more complicated real-world case studies.

5.5 APPLICATION OF SEQUENTIAL METAMODELING: DEVELOPMENT OF METAMODELS IN DESIGNING A PRESSURE VESSEL

In this section, we use the example of design of pressure vessels to illustrate the sequential metamodeling approach as described in Section 5.4. This example is taken from (Li and Chou, 1994; Sandgren, 1990) with some modifications. The cylindrical pressure vessel is shown in Figure 5.42. The shell is made in two halves of rolled steel plate which are joined by two longitudinal welds. Available rolling equipment limits the

length of the shell to 20 ft. The end caps are hemispherical, forged, and welded to the shell. All welds are single-welded butt joints with a backing strip. The material is carbon steel ASME SA 203 grade B.

There are three design variables – radius (R) and length (L) of the cylindrical shell, and the thickness (T) of the cylindrical shell and spherical head, which have the following ranges of interest:

$$10 \text{ in.} \leq R \leq 50 \text{ in.}$$
$$10 \text{ in.} \leq L \leq 100 \text{ in.}$$
$$0.9 \text{ in.} \leq T \leq 1.1 \text{ in.}$$

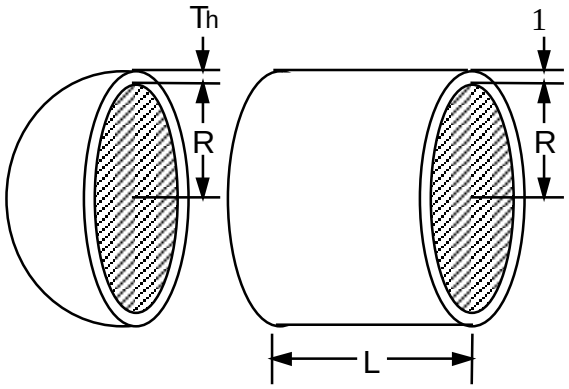


Figure 5.42 Pressure Vessel

The design objectives are to maximize the tank volume and minimize total system cost which is a combination of welding, material, and forming costs. The tank volume is written as:

$$Vol. = \pi R^2 L + \frac{4}{3}\pi R^3 \tag{5.3}$$

The total system cost is given by:

$$Cost = 0.6224RL + 1.7781TR^2 + 3.1661L + 19.84R \tag{5.4}$$

Meanwhile, the constraints which limit the minimal wall thickness T are from the ASME boiler and pressure vessel codes and are given as:

$$1 - 0.0193R \geq 0 \tag{5.5}$$

$$T - 0.00954R \geq 0 \tag{5.6}$$

Another constraint is put on the tank volume:

$$Vol - 1.296\text{E}5 \geq 0 \tag{5.7}$$

Given the ranges of design variables, we see that the first two constraints (Equations (5.5) and (5.6)) are automatically satisfied, thus we will not consider these constraints in our design. The third constraint is only related to one of the system responses, *Vol*. As talked about earlier, the design goals are also only related to *Vol* and *Cost*. Thus, in the metamodeling process, we will only consider metamodels for two system responses, the tank volume *Vol* and the system cost *Cost*. In Section 5.5.1, we will discuss on how to develop metamodels for multiple system responses in our framework of sequential DOE and metamodeling. Appropriate metamodels are then developed in Section 5.5.2, and the design solution is obtained after exploration of the design space.

5.5.1 Development of Metamodels for Multiple Responses in SEED

In our previous studies on SEED, we only considered problems with one response. In this section, our focus is on cases in which metamodels of multiple responses are needed in design.

The identification of important design variables in a multi-response problem has been studied by many researchers, most of which are with response surface metamodels. The identification of important factors is not the focus of our study in this dissertation; for case studies in this dissertation, we use the approach as used in (Ortega, 1998) to identify and screen out unimportant design variables. Our interest is in the design of sequential experiments and development of sequential metamodels (specifically, MARS and kriging metamodels in SEED) in multi-response problems.

Suppose there are n_r system responses for which we need to develop metamodels in the design process. When there is only one response, we could easily calculate the uncertainty associated with the metamodel accuracy following equations and methods described in Chapter 4. In a multi-response problem, there may be trade-offs in the allocation of new data points; different responses (and different metamodels with certain amount of prediction errors) may "drag" candidate points to different directions because candidate points with large prediction errors in one response may be with small prediction errors in another response. To take this trade-off into consideration, we need to modify the equations in SEED as presented in Chapter 4.

Suppose that we could assign the "degrees of importance" for each of the n_r system responses; there are many methods to achieve this, e.g., we could follow the

method as used in Selection Decision Support Problems (see, Mistree, et al., 1994). We use the symbol ρ_k to represent the importance of the kth system response, which satisfies:

$$0 \le \rho_k \le 1, \quad \sum_{k=1}^{n_r} \rho_k = 1, \quad \text{and} \quad k = 1,...,n_r \tag{5.8}$$

Following the constraints as described in Equation (5.8), we could assign larger values of ρ_k to important responses (e.g., safety in some examples). In sequential experimental design, we should pay more attention to these responses; the accuracy of metamodels for these responses is given higher priority.

Note that in Chapter 4, we developed two methods to formulate entries in the adjusted covariance matrix. Core equations for these two methods are Equations (4.27), (4.28) and (4.34). To reflect the relative importance of different responses in sequential experimental design in multi-response problems, we modify Equations (4.27) and (4.28) as below:

$$\sigma_{ij} = \sigma^2 \alpha_i \alpha_j R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right)$$

$$= \sigma^2 \cdot \begin{cases} \left(1 - \frac{1}{\lambda}\sum_{k=1}^{n_r}\left(\rho_k \left|\frac{e_{i,k}}{e_{\max,k}}\right|\right)\right)\left(1 - \frac{1}{\lambda}\sum_{k=1}^{n_r}\left(\rho_k \left|\frac{e_{j,k}}{e_{\max,k}}\right|\right)\right) R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) & when \begin{cases} i \le n,\ j > n \\ i > n,\ j \le n \end{cases} \\ R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) & when \begin{cases} i \le n,\ j \le n \\ i > n,\ j > n \end{cases} \quad i \ne j \\ 1 & when \quad i = j \end{cases} \tag{5.9}$$

and

$$\begin{aligned}\sigma_{ij} &= \sigma^2 R^{adj}\left(x_i, x_j\right) = \sigma^2 \prod_{m=1}^{n_{dv}} \exp\left(-\theta_m^{adj} \left|d_m\right|^2\right) \\ &= \sigma^2 \prod_{m=1}^{n_{dv}} \exp\left(-\beta_i \beta_j \theta_m \left|d_m\right|^2\right) \\ &= \sigma^2 \prod_{m=1}^{n_{dv}} \exp\left(-\left(1+\lambda \sum_{k=1}^{n_r}\left(\rho_k \left|\frac{e_{i,k}}{e_{\max,k}}\right|\right)\right)\left(1+\lambda \sum_{k=1}^{n_r}\left(\rho_k \left|\frac{e_{j,k}}{e_{\max,k}}\right|\right)\right)\theta_m \left|d_m\right|^2\right)\end{aligned} \tag{5.10}$$

Equation (5.9) is used to formulate entries of the adjusted covariance matrix without changing the correlation function (corresponding to Equation (4.28)), and Equation (5.10) is to formulate entries of the adjusted covariance matrix through changing the correlation function (corresponding to Equation (4.32)). Note there are n_r responses and the quantified importance of each response is ρ_k. $e_{max,k}$ is the maximum predicted prediction error of the current metamodel for the kth system response, and $e_{i,k}$ is the predicted prediction error of the current metamodel for the kth system response at point $\boldsymbol{x}_i$. Meanings of other symbols are the same as those for Equations (4.28) and (4.32). Note that in Equations (5.9) and (5.10) we use a single correlation function R, which is not inherit from any previous metamodels. As described in the single-variable, single-objective examples in Chapter 4 and previous sections of Chapter 5, when there is only one system response, values of θ from the previous metamodel could be used in formulation of the covariance matrix in the next sampling iteration. In cases with multiple responses, to be simple, we decide not to adopt this approach; instead, based on information from previous metamodels, the designers arbitrarily set the values of θ in the

correlation function R when formulating the covariance matrix. To develop a more effective approach to address the concerns above is a future work for this dissertation.

Comparing Equations (5.9) and (5.10) to Equations (4.28) and (4.34), we see that the only modification is on the formulations of the adjusting coefficients α_i and β_i. Responses with greater weight ρ_k play more important roles in allocating new data points because more of their prediction errors are reflected in Equations (5.9) and (5.10). There may be other formulations of the entries of the adjusted covariance matrix that help achieve the same goal. In this dissertation, we will only use Equations (5.9) and (5.10); the study and comparison of possible formulations would be one of the future work of this dissertation.

With Equations (5.9) or (5.10) we could build the adjusted covariance matrix; new data points could be identified through maximizing the determinant of the adjusted covariance matrix. In this dissertation, we develop metamodels for all system responses with the same set of data points. This simplifies our method and enables us to focus on the development and verification of the SEED method.

In real-world case studies, it is better to use different sets of data points to develop metamodels for different system responses. However, to use totally different data points in metamodeling requires much more computation time and effort than to use the same set of data points. To solve this problem, a method could be developed based on the usage of data and validation points. As described before, in sequential experimental design, we have information of two sets of points, n_d data points and n_{error} validation points. In each iteration of SEED, a number of new data and validation points are added

to increase the accuracy of metamodels. When there are multiple system responses, we may rearrange points in the large pool of observed points (data points + validation points) and form different sets of data/validation points for different system responses, i.e., for a particular observed point, we may use it as a data point for some responses, and as a validation point for other responses at the same time. An algorithm needs to be developed to help select the set of data points for a particular system response; one possible criterion may be the prediction errors – we should use data points so that the corresponding metamodel's prediction errors at the rest points (validation points) are smallest. This is closely related to the cross-validation method. Here we will not go further in this direction; the development of such an approach is a future work of this dissertation.

5.5.2 Development of Metamodels for System Responses

In this section, we will develop acceptable metamodels for the two system responses, *Vol* and *Cost*. Following the approach described in Section 5.4 (see Figure 5.40 and Figure 5.41), we will first build RS metamodels and screen out unimportant design variables, and then accurate metamodels (MARS or kriging) could be developed with the SEED method.

Since there are only 3 design variables in this example, we need not use the fractional factorial experiments as initial experimental design. The factorial experiments with 8 points, as listed in Table 5.25, are used to help develop first-order RS metamodels.

The center point will not be observed and used to develop the initial metamodel. The design variables are scaled to [−1, 1] when building the RS metamodels.

Table 5.25 Initial Experimental Design with 8 Data Points

R	*L*	*T*	*R_norm*	*L_norm*	*T_norm*	*Vol*	*Cost*
10	10	0.9	−1	−1	−1	7330.38	452.33
50	10	0.9	1	−1	−1	602138.60	5335.59
10	100	0.9	−1	1	−1	35604.72	1297.44
50	100	0.9	1	1	−1	1308996.96	8421.34
10	10	1.1	−1	−1	1	7330.38	487.89
50	10	1.1	1	−1	1	602138.60	6224.64
10	100	1.1	−1	1	1	35604.72	1333.00
50	100	1.1	1	1	1	1308996.96	9310.39

Given the information in Table 5.25, we develop first-order regression model as following:

$$Vol = 488518 + 467050\,R + 183783\,L + 0\,T \tag{5.11}$$

$$Cost = 4108 + 3215\,R + 983\,L + 231\,T \tag{5.12}$$

More details could be found in Appendix B. As introduced in Chapter 2, widely used statistics in Response Surface Methodology (RSM), like MSE, F-statistics, etc., are not suitable in deterministic applications with computer experiments because of the lack of random errors. Only values of R-sq and adjusted R-sq could give some verification of model adequacy, and often this measure is not sufficient (Simpson, et al., 1997). In this example, the values of R-sq and adjusted R-sq for the metamodel of *Vol* are 89.7% and 82.1%, and those for the metamodel of *Cost* are 96.9% and 94.6%, respectively. This

shows that the first-order RS metamodels in Equations (5.11) and (5.12) are somewhat "accurate"; thus we are confident to use it to identify and screen unimportant design variables.

We notice that the design variable *T* has no influence in the metamodel of *Vol* (the coefficient of *T* in Equation (5.11) is zero), and much smaller effect in the metamodel of *Cost*. The coefficient of *T* in Equation (5.12) is about 1/4 and 1/15 of those of *R* and *L*. The main effects plot is shown in Figure 5.43, in which we see clearly that the design variable *T* has little influence on the response *Cost*. Values of *t*-ratio and *p* of *Cost* (see Appendix B) give some reference on how importance a design variable is. The *p*-value for *T* in *Cost* is 0.484, which is not small in a [0,1] range, and much larger than those for *R* and *L*, which are 0.0 and 0.031, respectively. The *t*-ratio for *T* in *Cost* is 0.77, which is much smaller than those for *R* and *L*, which are 10.73 and 3.28, respectively. Since a small *p*-value and a large *t*-ratio imply significant influence of the corresponding design variable on a response, we see that the design variable *T* has smaller influence on *Cost* than *R* and *L* do. To decide whether *T* is unimportant or not, we need to set a confidence level and perform mathematical tests as used in RSM or ANOVA (Analysis of Covariance). However, these tests may not be appropriate in deterministic applications which have no random errors. Thus in this example we will not do mathematical tests.

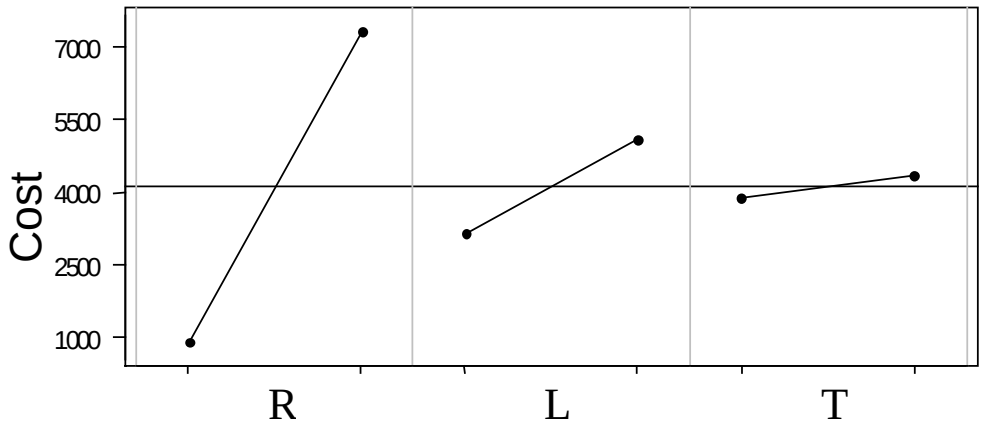


Figure 5.43 Main Effects Plot – Means for *Cost*

Based on our observations above, the design variable *T* is likely to be unimportant since it has no influence on *Vol* and little influence on *Cost*. A first-order RS metamodel of *Cost* is developed without the design variable *T* and shown in Equation (5.13); more details of this metamodeling are presented in Appendix B. Values of R-sq and adjusted R-sq for this RS model are 96.5% and 95.1%, respectively, which are almost the same as those for Equation (5.12). This also implies that the design variable *T* has little effects on *Cost*.

$$Cost = 4108 + 3215\ R + 983\ L \tag{5.13}$$

In future processes of metamodeling and design space exploration, effects from the design variable *T* are omitted and a constant value should be assigned to *T*. In RSM, an unimportant design variable is usually set at the center of its factor range (i.e., a value of zero in the [−1,1] interval). However, in sequential experimental design, in order to save computation time and effort on simulation, we should keep as many current observed points as possible. If the normalized value of the design variable *T* is set as zero, none of the current observed points could be used in future metamodeling process. Thus, we should set the normalized value of *T* as either 1 or −1; in this way we are able to keep 4 observed data points for future use. In Figure 5.43 we see that the main effect of *T* on *Cost* is positive, and our design goal is to minimize *Cost*, so we should set the normalized value of *T* at its lower band −1 (or say, the value of *T* is set as 0.9in.) to help obtain smaller values of *Cost*. Thus, the first 4 data points in Table 5.25 will be kept in future metamodeling processes.

After building first-order RS metamodels and screening out unimportant design variables, we could either build higher-order RS metamodels (more data points are needed to realize CCD experiments) or go directly to the next step in Figure 5.41, the Sequential Exploratory Experimental Design. In this example, since the actual response functions (see Equations (5.3) and (5.4)) are not highly nonlinear, second-order RS metamodels should be acceptable for design space exploration. However, in order to illustrate our sequential experimental design and metamodeling approach, we decide not

to develop second-order RS metamodels and go directly to the SEED process to develop kriging or MARS metamodels for the system responses *Vol* and *Cost*.

As a reference, 3-D plots of *Vol* and *Cost* with respect to the design variables *R* and *L* are presented in Figure 5.44. In Figure 5.44 we see that since the actual function of *Vol* and *Cost* are no more than cubic functions, the exact response surfaces are not highly nonlinear. The response surface of *Cost* is more flat than that of *Vol* because the *Cost* is calculated with a second-order function while *Vol* is calculated with a third-order function. Our next goal in metamodeling is to develop acceptable metamodels to reflect the actual response surfaces in Figure 5.44.

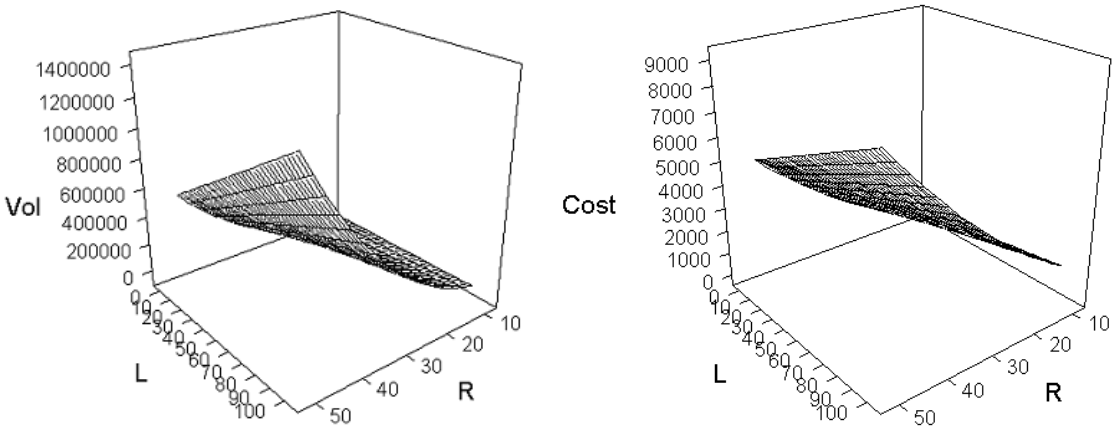


Figure 5.44 Actual Responses of Volume and Cost

In this example, besides the initial experiments with 4 data points and 4 validation points, we plan to add in 4 more data points and 2 more validation points. Thus finally we will have 14 observed points. Similar to our previous examples, to be simple, we will not use the accuracy of metamodels as stopping criteria in the SEED sampling process.

Iteration I – Step 1: Initial Experimental Design. As discussed in Chapter 4, there are many ways to design the initial experiments in SEED. In this example, since we already observed responses at points when developing RS metamodels, we will use these points as our initial experimental design. As discussed earlier, 4 data points could be kept and used in SEED, as listed in Table 5.26. Note that since the design variable *T* has been identified as an unimportant factor and will be set as 0.9 inch in all steps in the SEED method. Note that in kriging and MARS metamodeling in SEED, we normalize design variables to [0,1].

Table 5.26 Initial Experimental Design with 4 Data Points

R	*L*	*R_norm*	*L_norm*	*Vol*	*Cost*
10	10	0	0	7330.38	452.33
50	10	1	0	602138.60	5335.59
10	100	0	1	35604.72	1297.44
50	100	1	1	1308996.96	8421.34

Iteration I – Step 2: Simulation and Initial Metamodel of Responses. Kriging metamodels are developed based on the information in Table 5.26. For the kriging metamodel of *Vol*, we got θ_1=79.44092 and θ_2=0.59025. For the kriging metamodel of *Cost*, we got θ_1=77.00927 and θ_2=0.28594. In this study, if not specifically pointed out, the symbol θ_1 always corresponds to the design variable *R*, and θ_2 corresponds to the design variable *L*. The kriging metamodels are illustrated in Figure 5.45.

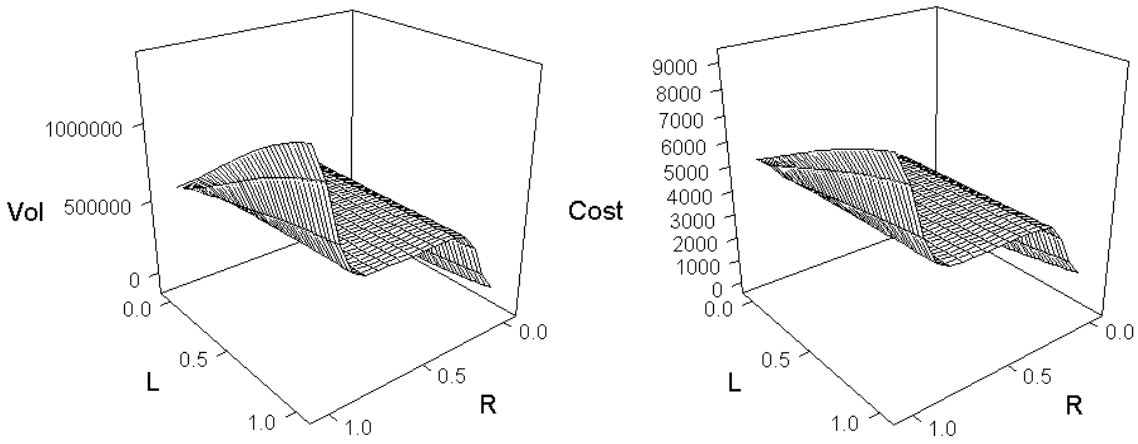


Figure 5.45 Initial Kriging Metamodel for Volume and Cost

Iteration I – Step 3: Identification of New Validation Points. In this step we need to identify 4 validation points. In the first iteration, we only have information from data points and the initial metamodel. Without previous information on metamodel validation in this step, we will add in new points that are as far from current points as possible. Maximum entropy sampling is directly applied without adjustment to the covariance matrix to help identify the validation points. Values of θ in the correlation function R are set as 20 in formulating the covariance matrix. New validation points are listed in Table 5.27.

Note that in Table 5.27 the validation points are not strictly symmetrical to the center point of the design space because of small computational errors in the calculation and optimization of determinants of the covariance matrix. Also, similar to the single-variable example in Chapter 4, in this example, there should be another set of validation points that has the same value of determinant, i.e., are "equally" good in the optimization

of determinants of the covariance matrix. That set of validation points could be easily obtained by switching the values for the two design variables, *R* and *L*. In Chapter 4, we have shown that the SEED method is robust to the selection of points in each step, i.e., no matter which set of points are selected when there are multiple choices, the designers are assured to get acceptable metamodels after multiple iterations. Thus, in this section, we will only consider the case with one possible set of validation points in this step, i.e., the points listed in Table 5.27.

Table 5.27 Four New Validation Points Added in Iteration I

R	***L***	***R_norm***	***L_norm***	***Vol***	***Cost***
30.036	10.108	0.5009	0.0012	142153.31	2260.6
18.968	55.225	0.2242	0.5025	91006.7	1778.9
41.448	55.081	0.7862	0.5009	595538.3	5166.86
29.984	99.982	0.4996	0.9998	395307.47	4216.03

Iteration I – Step 4: Metamodels of Prediction Errors. In this step, prediction errors at both data and validation points are used to develop two metamodels to predict prediction errors for the two system responses across the design space. The prediction errors at data points are zero; prediction errors at validation points are listed in Table 5.28.

Table 5.28 Prediction Errors at Validation Points in Iteration I

R	***L***	***R_norm***	***L_norm***	***Vol_err***	***Cost_err***
30.036	10.108	0.5009	0.0012	346364.4	1616.075
18.968	55.225	0.2242	0.5025	387949.3	2031.313
41.448	55.081	0.7862	0.5009	-93268.6	-1195.64
29.984	99.982	0.4996	0.9998	93210.2	-339.355

We first developed metamodels of prediction errors with MARS, as illustrated in Figure 5.46. As mentioned in Section 5.3, since MARS metamodels smooth the data, when developing MARS metamodels, to be safe it is better to check whether they have big problems in prediction at observed points (though usually the prediction errors of MARS at observed points are very small). In this case, we found that the MARS metamodel of prediction errors for the system response *Vol* is not working as expected. The prediction error at point [1,1] should be about zero since it is one of the data points listed in Table 5.26, while in Figure 5.46 we see that the predicted error at [1,1] with the MARS metamodel is around –900,000. The difference between actual and predicted values is so large that we could not trust the MARS metamodel of prediction errors for *Vol* as illustrated in the left plot of Figure 5.46. A kriging metamodel of prediction errors for *Vol* is developed and illustrated in Figure 5.47. For this kriging metamodel, we got θ_1 = 99.81484 and θ_2 = 0.52487.

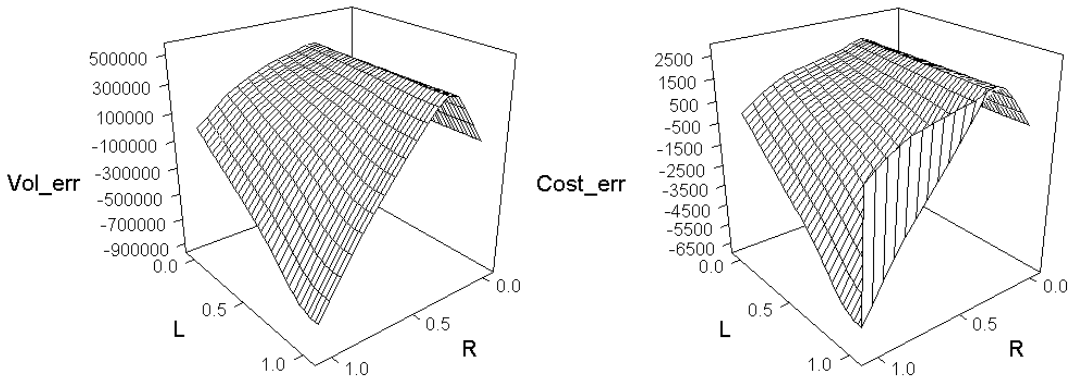


Figure 5.46 MARS Metamodels of Prediction Errors in Iteration I

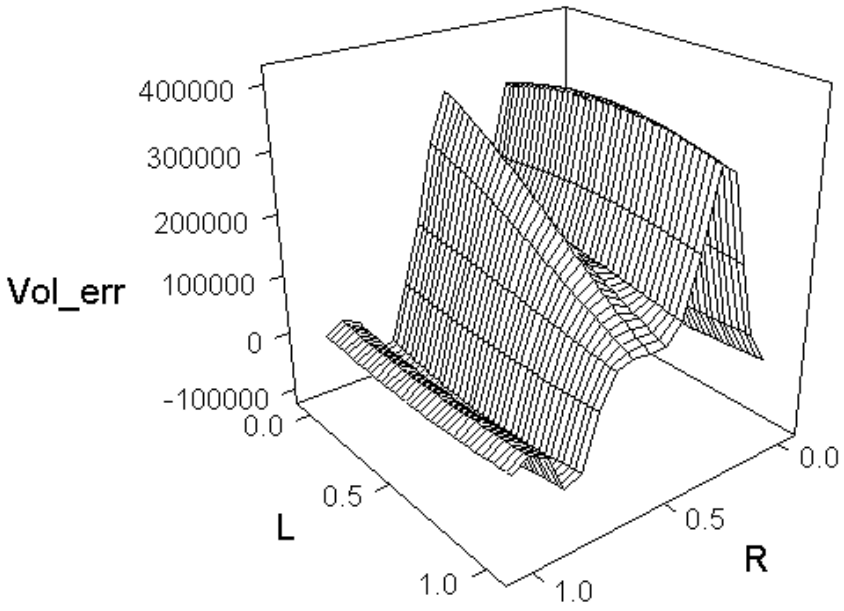


Figure 5.47 Kriging Metamodel of Prediction Errors in Iteration I

As for the MARS metamodel of prediction errors for *Cost*, we do not observe any abnormal features. Thus, in future steps of SEED, we will use the MARS metamodel of prediction errors for *Cost* as illustrated in the right plot of Figure 5.46, and the kriging metamodel of prediction errors for *Vol* as illustrated in Figure 5.47. The maximum absolute predicted prediction errors are $e_{max,vol} \approx 370000$, and $e_{max,cost} \approx 6290$.

Iteration I – Step 5: Metamodel Validation. This step is skipped since we do not use the accuracy of metamodels as the stopping criterion of the SEED method.

Iteration I – Step 6: Formulation of the Adjusted Covariance Matrix. To get more accurate metamodels, we decide to add in $n_{new} = 2$ data points. The 6×6 adjusted covariance matrix is formulated following Equation (5.9). Values of θ'*s* in the

correlation function are set as 20. The two responses, *Vol* and *Cost*, are considered to be equally important, i.e., $\rho_{vol} = \rho_{cost} = 0.5$. The value of λ is set as 2.

To realize the formulation of adjusted covariance matrix with multiple responses, the FORTRAN program used in Chapter 4 is modified and presented in Appendix B.

Iteration I – Step 7: Identification of New Data Points. In this step, by maximizing the determinant of the adjusted covariance matrix as developed in the previous step, two possible new data points are identified and listed in Table 5.29. This is done in iSIGHT; the picture of task organization of this step in iSIGHT is illustrated in Appendix B.

Table 5.29 Two New Data Points Added in Iteration I

R	***L***	***R_norm***	***L_norm***	***Vol***	***Cost***
30.036	79.102	0.5009	0.7678	337736.41	3768.84
30	28.387	0.5	0.2043	193359.69	2655.38

Iteration I – Step 8: Updated Metamodels of Responses. Now we have 6 data points, as listed in Table 5.26 and Table 5.29. Two new kriging metamodels are developed with information from these 6 data points, and illustrated in Figure 5.48. For the kriging metamodel of *Vol*, we got θ_1=1.43075 and θ_2=0.37732. For the kriging metamodel of *Cost*, we got θ_1=0.17743 and θ_2=0.06426. In Figure 5.48 we see that metamodels for both system responses are more accurate than the ones with 4 data points.

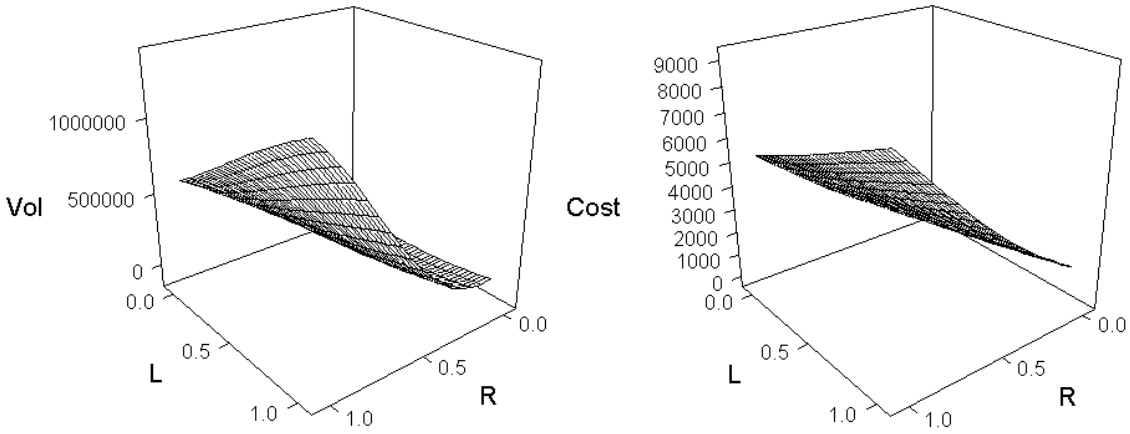


Figure 5.48 Updated Metamodels of Responses with 6 Data Points

Iteration II – Step 3: Identification of New Validation Points. In this step, we need to add in 2 new validation points. Two kriging metamodels are developed for *Vol* and *Cost* based on information from 4 validation points. For the kriging metamodel of *Vol*, we got θ_1=99.60797 and θ_2=99.22609. For the kriging metamodel of *Cost*, we got θ_1=3.17433 and θ_2=0.56250. Plots of these two metamodels are illustrated in Figure 5.49.

Prediction errors of these two metamodels at 6 data points and 4 validation points are calculated and listed in Table 5.30. Two MARS metamodels of prediction errors are then developed with information at 6 data points and 4 validation points, and illustrated in Figure 5.50. From Figure 5.50 and Table 5.30 we see that the MARS metamodel of prediction errors for *Vol* does not perform well; the predicted prediction errors at four validation points are far from zero (since validation points are used to develop kriging metamodels of responses in this step, prediction errors at these points should be zero).

The MARS metamodel of prediction errors for *Cost* works well. This suggests that we should not use MARS metamodel to predict prediction errors for *Vol* in this step. A kriging metamodel is developed to calculate prediction errors for *Vol*, as illustrated in Figure 5.51. For this kriging metamodel, we got θ_1=99.93684 and θ_2=2.00708.

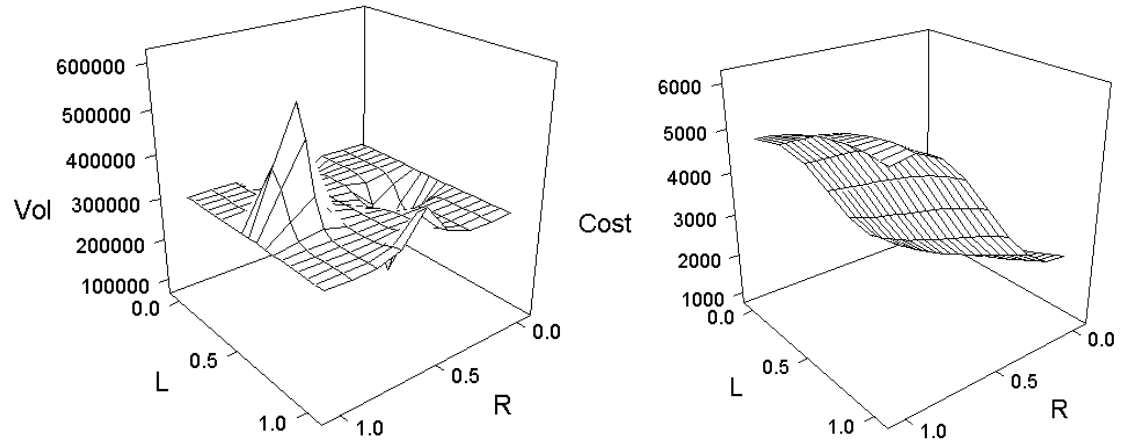


Figure 5.49 Kriging Metamodels of Responses Developed with 4 Validation Points in Iteration II – Step 3

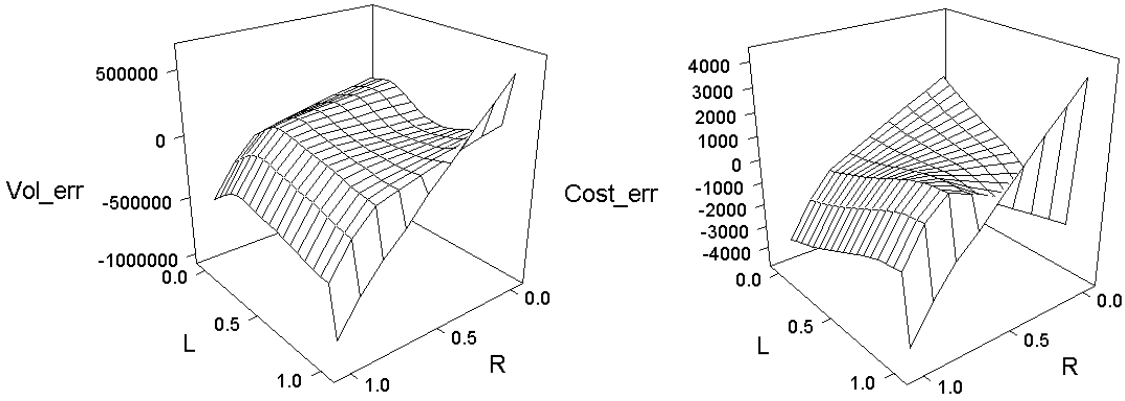


Figure 5.50 MARS Metamodels of Prediction Errors Developed with 6 Data Points and 4 Validation Points in Iteration II – Step 3

Table 5.30 Prediction Errors of MARS Metamodels at Data and Validation Points in Iteration II – Step 3

R	*L*	*R_norm*	*L_norm*	*Vol_err*	*Cost_err*
10.00	10.00	0	0	134822.93	1808.27
50.00	10.00	1	0	-511131.90	-3556.69
10.00	100.00	0	1	559933.58	3869.41
50.00	100.00	1	1	-913689.49	-4205.31
30.04	79.10	0.5009	0.7678	-31268.38	67.38
30.00	28.39	0.5	0.2043	109907.60	-35.50
30.04	10.11	0.5009	0.0012	-103.25	-0.73
18.97	55.23	0.2242	0.5025	-5946.40	-2.20
41.45	55.08	0.7862	0.5009	-8028.14	-1.94
29.98	99.98	0.4996	0.9998	-8582.95	-0.24

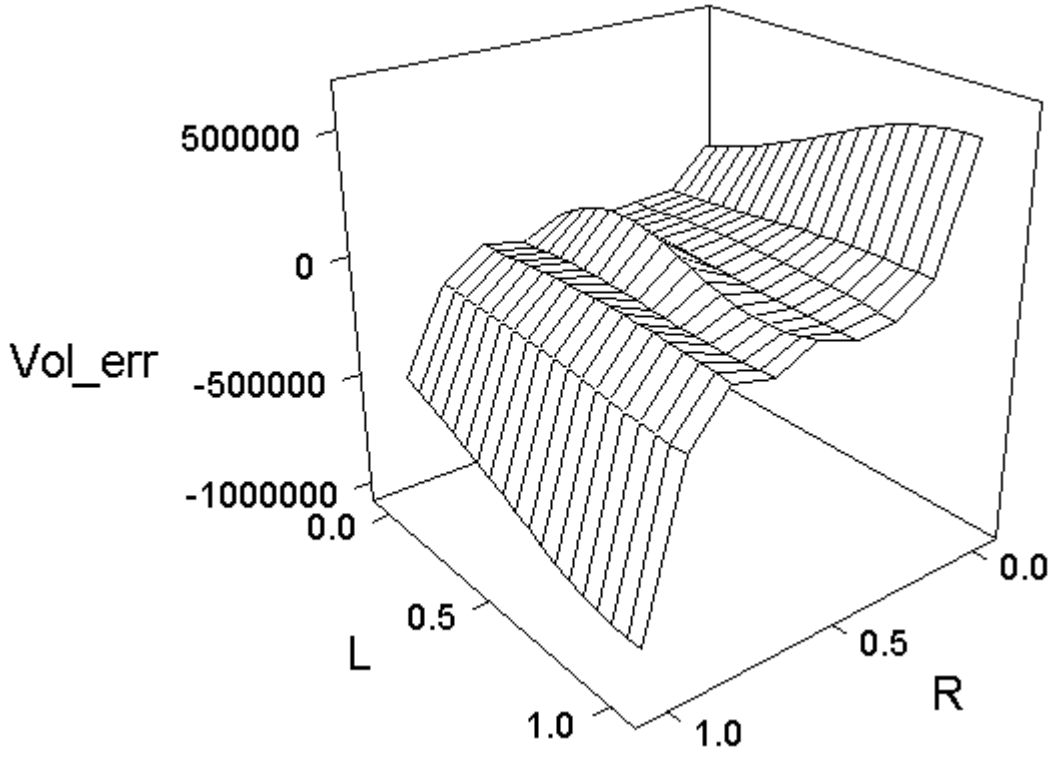


Figure 5.51 Kriging Metamodel of Prediction Errors for Volume in Iteration II – Step 3

To identify 2 new validation points, a 12×12 covariance matrix is built, with the first 6 rows and columns corresponding to the data points, the 7^{th} to 10^{th} rows and columns corresponding to the validation points, and the last two rows and columns corresponding to the candidate points. Then the 12×12 adjusted covariance matrix is formulated following Equation (5.9). Values of θ*'s* in the correlation function are set as 20. The two responses, *Vol* and *Cost*, are considered to be equally important, i.e., ρ_{vol} = ρ_{cost} = 0.5. The value of λ is set as 2. By maximizing the determinant of this adjusted covariance matrix, 2 new validation points are identified and listed in Table 5.31.

Table 5.31 Two New Validation Points Added in Iteration II

R	***L***	***R_n***	***L_n***	***Vol***	***Cost***
17.48	28.15	0.1871	0.2017	49424.63	1231.57
42.38	83.39	0.8096	0.8154	789523.87	6179.38

Iteration II – Step 4: Metamodels of Prediction Errors. Prediction errors of the updated kriging metamodels (Figure 5.48) are zero at data points. Prediction errors at the 6 validation points are listed in Table 5.32. The predicted values of responses are calculated with updated kriging metamodels in Figure 5.48. Note that some of the predicted values are negative, which is apparently wrong since both *Vol* and *Cost* should have positive values.

Table 5.32 Prediction Errors at Validation Points

R_n	*L_n*	*Vol*	*Cost*	*Vol_pred*	*Cost_pred*	*Vol_err*	*Cost_err*
0.5009	0.0012	142153.31	2260.60	171500.83	2347.96	29347.53	87.36
0.2242	0.5025	91006.70	1778.90	-2617.04	1643.15	-93623.74	-135.76
0.7862	0.5009	595538.30	5166.85	686754.27	5184.76	91215.97	17.90
0.4996	0.9998	395307.47	4216.03	415060.77	4297.58	19753.30	81.55
0.1871	0.2017	49424.63	1231.57	-30331.33	572.18	-79755.96	-659.40
0.8096	0.8154	789523.87	6179.38	1131853.75	7483.56	342329.88	1304.19

Two MARS metamodels of predicted errors are developed with information of prediction errors at 6 data points and 6 validation points. These two metamodels are illustrated in Figure 5.52; more details are presented in Appendix B.

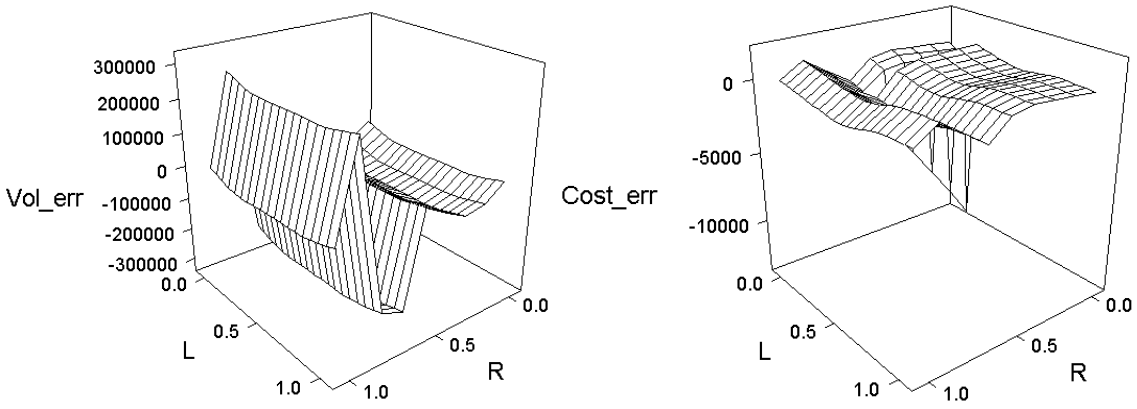


Figure 5.52 MARS Metamodels of Prediction Errors in Iteration II

Similar to what we did in *Iteration I*, here we need to check whether these MARS metamodels work properly at data and validation points; we expect the predicted prediction errors from these metamodels to be very close to those "true" values that we observed. The true and predicted prediction errors are listed in Table 5.33. In Table 5.33

we see that the MARS metamodel of prediction errors for *Cost* does not perform well; the difference between actual and predicted values is very large at several points, e.g., there is a difference of about 1600 at (0.8096, 0.8154). The MARS metamodel of prediction errors for *Vol* performs not well; the difference between actual and predicted values is very large (e.g., a difference of 4627 at (0.5009,0.7678) where the prediction error should be zero), though this difference may seem to be small compared to the huge range of prediction errors of *Vol* (from around –86274 to +94319). Thus, two kriging metamodels of prediction errors for *Vol* and *Cost* are developed and illustrated in Figure 5.53. For the kriging metamodel of prediction errors for *Vol*, we got θ_1=99.99965 and θ_2=6.49084. For the kriging metamodel of prediction errors for *Cost*, we got θ_1=99.99659 and θ_2=17.19953.

Table 5.33 True and Predicted Prediction Errors at Data/Validation Points

R_n	***L_n***	***Vol_err***	***Cost_err***	***Vol_err_pred***	***Cost_err_pred***
0	0	0	0	391.61	1.23
1	0	0	0	-597.57	2.66
0	1	0	0	-363.31	-0.47
1	1	0	0	598.44	-2.64
0.5009	0.7678	0	0	-4627.24	2.76
0.5	0.2043	0	0	-1969.90	-5.70
0.5009	0.0012	29347.53	87.36	30193.33	83.65
0.2242	0.5025	-93623.74	-135.76	-86273.75	-141.00
0.7862	0.5009	91215.97	17.90	94318.65	23.17
0.4996	0.9998	19753.30	81.55	19181.00	85.10
0.1871	0.2017	-79755.96	-659.40	-32350.83	280.32
0.8096	0.8154	342329.88	1304.19	-25824.06	-283.06

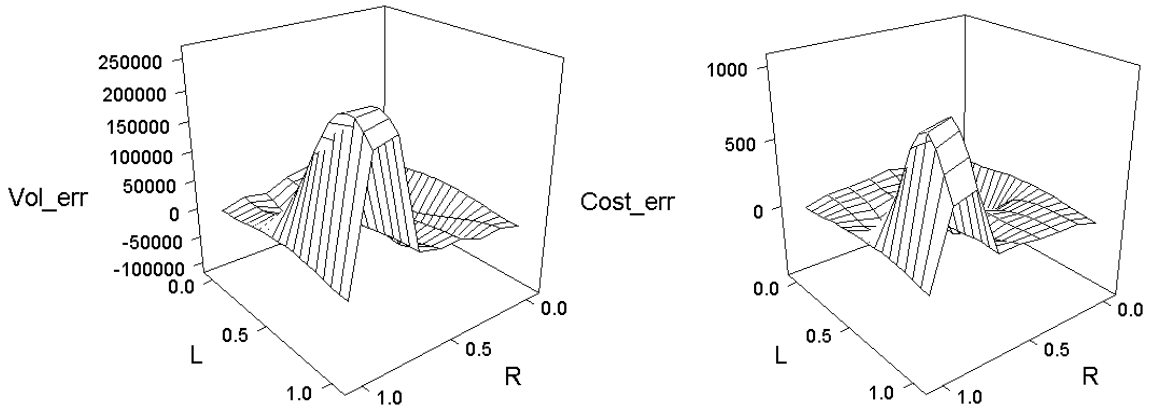


Figure 5.53 Kriging Metamodels of Prediction Errors in Iteration II

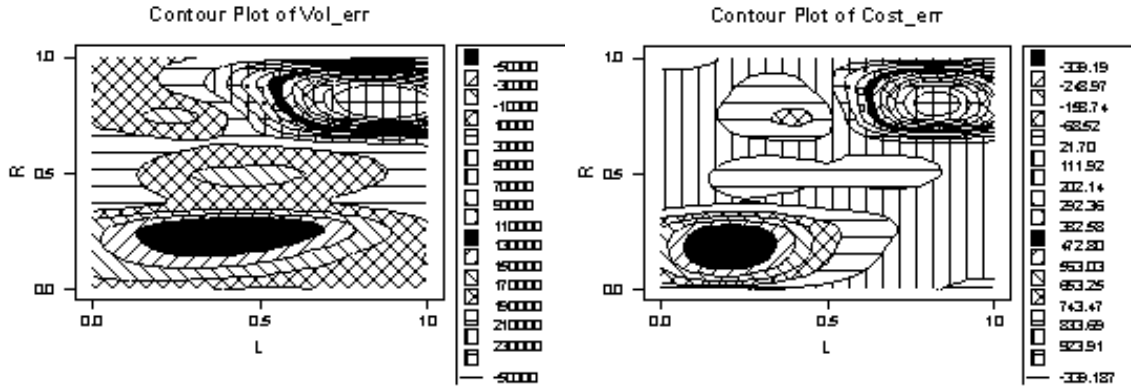


Figure 5.54 Contour Plots of Metamodels of Prediction Errors for *Vol* and *Cost*

Contour plots of kriging metamodels of prediction errors for *Vol* and the MARS metamodel of prediction errors for *Cost* are illustrated in Figure 5.54. These two metamodels are used in future steps to help formulate the adjusted covariance matrix. The maximum absolute prediction errors are $e_{max,vol} \approx 93700$ and $e_{max,cost} \approx 258$.

Iteration II – Step 5: Metamodel Validation. This step is skipped.

Iteration II – Step 6: Formulation of the Adjusted Covariance Matrix. Two new data points are to be added in this iteration. Since this is the last step in this SEED process, a 14×14 adjusted covariance matrix is built following Equation (5.8). Values of θ'*s* in the correlation function are set as 20. The two responses, *Vol* and *Cost*, are considered to be equally important, i.e., $\rho_{vol} = \rho_{cost} = 0.5$. The value of λ is set as 2.

Iteration II – Step 7: Identification of New Data Points. In this step, by maximizing the determinant of the adjusted covariance matrix, two possible new data points are identified and listed in Table 5.34.

Table 5.34 Two New Data Points Added in Iteration II

R	*L*	*R_norm*	*L_norm*	*Vol*	*Cost*
18.08	82.19	0.2021	0.8021	109213.48	2067.43
41.97	27.96	0.7993	0.1996	464482.31	4470.92

Iteration II – Step 8: Updated Metamodels of Responses. Now we have 8 data points and 6 validation points as listed in Table 5.35. Since we already got 14 observed points, the SEED process will stop in this iteration. Final metamodels of *Vol* and *Cost* are developed based on the information in Table 5.35; these metamodels are illustrated in Figure 5.55. For the kriging metamodel of *Vol*, we got θ_1=0.19587 and θ_2=0.00136. For the kriging metamodel of *Cost*, we got θ_1=0.00226 and θ_2=0.00122. As a comparison, two MARS metamodels are also developed for *Vol* and *Cost*, and illustrated in Figure 5.56.

Table 5.35 Observed Points

R	*L*	*R_norm*	*L_norm*	*Vol*	*Cost*
10.00	10.00	0	0	7330.38	452.33
50.00	10.00	1	0	602138.60	5335.59
10.00	100.00	0	1	35604.72	1297.44
50.00	100.00	1	1	1308996.96	8421.34
30.04	79.10	0.5009	0.7678	337697.71	3768.84
30.00	28.39	0.5	0.2043	193359.69	2655.38
18.08	82.19	0.2021	0.8021	109213.48	2067.43
41.97	27.96	0.7993	0.1996	464482.31	4470.92
30.04	10.11	0.5009	0.0012	142153.31	2260.60
18.97	55.23	0.2242	0.5025	91006.70	1778.90
41.45	55.08	0.7862	0.5009	595538.30	5166.85
29.98	99.98	0.4996	0.9998	395307.47	4216.03
17.48	28.15	0.1871	0.2017	49424.63	1231.57
42.38	83.39	0.8096	0.8154	789523.87	6179.38

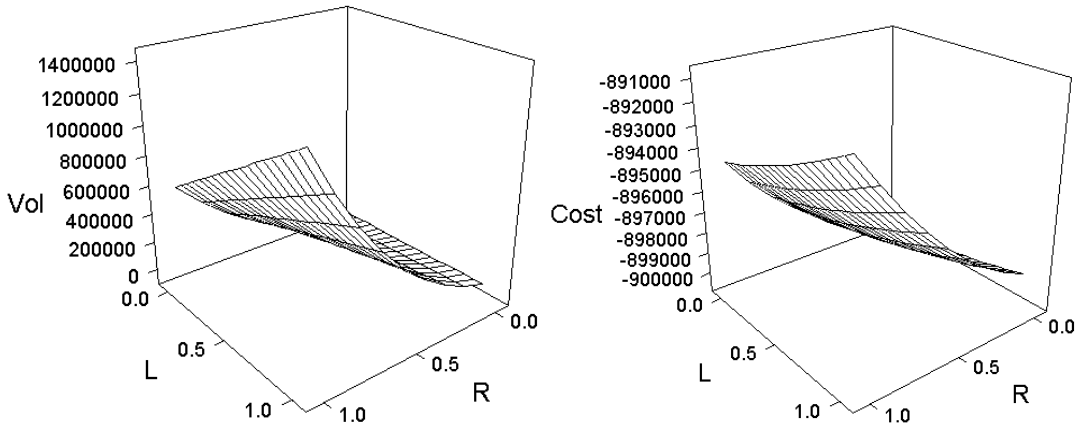


Figure 5.55 Final Kriging Metamodels for *Vol* and *Cost*

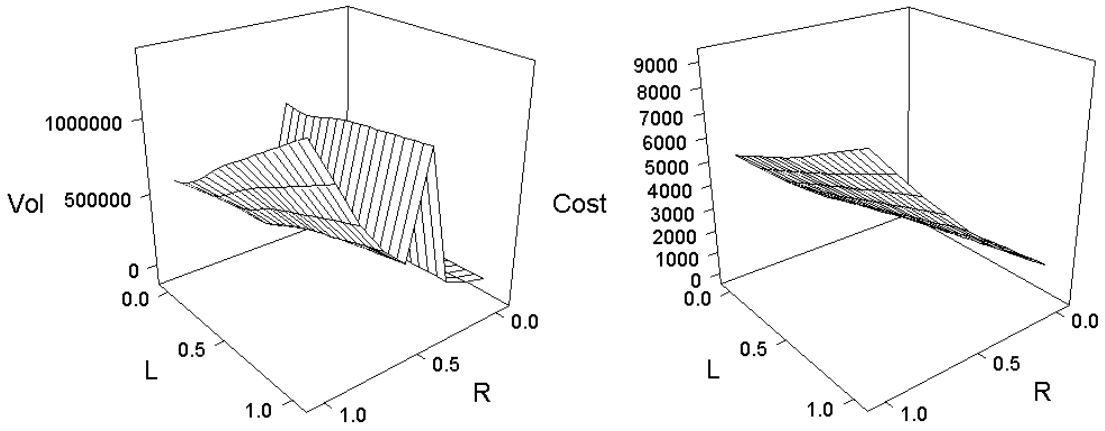


Figure 5.56 Final MARS Metamodels for *Vol* and *Cost*

In Figure 5.55 we see that the kriging metamodel for *Vol* works well, while that for *Cost* is not acceptable. The Predicted values of *Cost* with the kriging metamodel are all negative, which is far away from actual values. As discussed in Section 5.2.1 the kriging algorithm may cause this problem. We should not use the kriging metamodel to predict *Cost* in our later stages of this pressure vessel design. As to the MARS metamodels, we see that the MARS metamodel does not work well in prediction of *Vol* because the predicted values at observed points do not match with the actual values; however, it works well when predicting values of *Cost*. Thus, in this problem, we will use the kriging metamodel to predict responses of *Vol* and the MARS metamodel to predict responses of *Cost*. Contour plots for these two metamodels are illustrated in Figure 5.57. These two metamodels will be used in design space exploration for solutions that satisfy design constraints and achieve design goals as described at the beginning of Section 5.5.

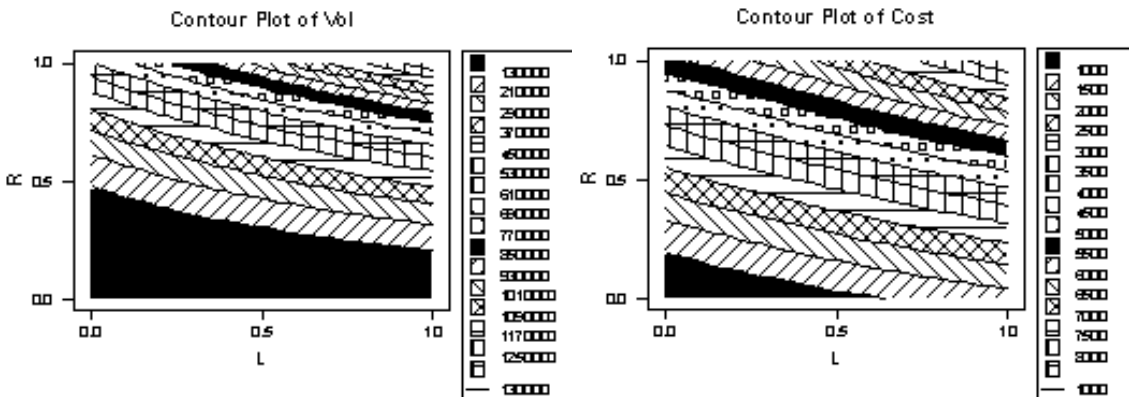


Figure 5.57 Contour Plots of Final Metamodels for *Vol* and *Cost*

The approach of sequential metamodeling and its integration with the SEED method are illustrated in this section. The initial experiments and RS metamodels are used to identify and screen out the unimportant design variable, the wall thickness *T*. Then the SEED method is applied and metamodels of system responses are updated as new data points are added in. Both kriging and MARS are used in developing the metamodels of responses and prediction errors in applying SEED. A very interesting observation is that sometimes the MARS technique does not work well because it does not necessarily predict accurately at observed points. In real-world applications, designers could observe this problem by examining the difference between actual and predicted values at observed points; in such cases, kriging may be used as a remedy to develop the metamodel that met difficulty with the MARS technique. This will be summarized and further discussed later.

5.5.3 Comparison of Metamodels from SEED and Single-Stage Experiments Designs

To verify the strategy of sequential experimental design, we need to compare the above results to that obtained with metamodels developed in a single-stage experimental design. Two single-stage experimental designs are studied, one of which is Latin Hypercubes, and the other is maximum entropy sampling as stated in Currin, et al., 1991 (without adjusting the covariance matrix); both of them have 14 data points, as listed in Table 5.36 and Table 5.37. Kriging and MARS metamodels for both *Vol* and *Cost* are developed with information from Table 5.36 and Table 5.37. For each experimental design and each response, the more accurate metamodel is selected. As a result, MARS metamodels of *Vol* and *Cost* developed with information from Table 5.36, and kriging metamodel of *Vol* and *Cost* developed with information from Table 5.37, are selected and used in our comparisons in this section.

Table 5.36 Single-Stage Maximum Entropy Sampling with 14 Data Points

R_n	*L_n*	*R*	*L*	*Vol*	*Cost*
0	0	10.00	10.00	7330.38	452.33
1	0	50.00	10.00	602138.60	5335.59
0	1	10.00	100.00	35604.72	1297.44
1	1	50.00	100.00	1308996.96	8421.34
0.5009	0.0012	30.04	10.11	142153.31	2260.60
0.2242	0.5025	18.97	55.23	91006.70	1778.90
0.7862	0.5009	41.45	55.08	595538.30	5166.85
0.4996	0.9998	29.98	99.98	395307.47	4216.03
0	0.3141	10.00	38.27	16211.35	717.78
1	0.6927	50.00	72.34	1091779.39	7473.08
0	0.69	10.00	72.10	26839.67	1035.46
1	0.3067	50.00	37.60	818932.06	6281.99
0	0.5022	10.00	55.20	21529.75	876.74
0.6953	0.2135	37.81	29.22	357677.27	3818.25

Table 5.37 Latin Hypercubes with 14 Data Points

R_n	*L_n*	*R*	*L*	*Vol*	*Cost*
0.0	0.9231	10.00	93.08	33430.42	1232.45
0.07692	0.8462	13.08	86.16	55652.71	1507.12
0.1538	0.7692	16.15	79.23	82586.28	1785.27
0.2308	0.0	19.23	10.00	41416.08	1124.82
0.3077	0.4615	22.31	51.54	127071.89	2117.67
0.3846	0.3846	25.38	44.61	158823.44	2380.87
0.4615	1.0	28.46	100.00	351019.21	3948.80
0.5385	0.3077	31.54	37.69	249220.62	3076.95
0.6154	0.1538	34.62	23.84	263500.23	3193.52
0.6923	0.6154	37.69	65.39	516135.63	4762.26
0.7692	0.2308	40.77	30.77	444496.07	4346.80
0.8462	0.5385	43.85	58.47	706271.52	5727.41
0.9231	0.07692	46.92	16.92	549847.47	5002.41
1.0	0.6923	50.00	72.31	1091496.64	7471.85

With information at data points in the single-stage maximum entropy sampling (Table 5.36), we develop the MARS metamodels of *Vol* and *Cost*. With information at data points in the Latin Hypercube design (Table 5.37), we develop the kriging metamodel of *Vol*; the parameters are θ_1=0.03776 and θ_2=0.00252. A MARS metamodel is developed to predict values of *Cost* with information from the Latin Hypercube design.

To compare the accuracy of metamodels from different experimental designs, the values of NRMSE and NMAX are calculated and listed in Table 5.38 based on Equations (2.7) and (2.9). As introduced in Chapter 2, the smaller the values of NRMSE and NMAX, the more accurate the corresponding metamodel is. In the first row of Table 5.38 we see that metamodels for two system responses, *Vol* and *Cost*, are studied. In the second row, designs of experiments (DOE) used are Sequential Exploratory Experimental

Design (SEED), single-stage maximum entropy sampling (S-MES), and Latin Hypercubes (LH).

Table 5.38 Accuracy of Metamodels from Different Experimental Designs

Response	*Vol*			*Cost*		
DOE	SEED	S-MES	LH	SEED	S-MES	LH
NRMSE	0.00009	0.0395	0.0039	0.0058	0.0295	0.0058
NMAX	0.00002	0.1055	0.0051	0.0262	0.0943	0.0262

In Table 5.38 we see that metamodels from SEED is most accurate since they have smallest NRMSE and NMAX values for *Vol* and *Cost*. Single-stage maximum entropy sampling performs better in modeling *Cost* than in modeling *Vol*; it performs worst among these three methods because it has the largest values of NRMSE and NMAX for both *Vol* and *Cost*. The Latin Hypercubes design performs better than single-stage maximum entropy sampling, but worse than the SEED method.

Another issue to be noticed in comparison is the computation and handling expense associated with SEED and the single-stage experimental designs. To design Latin Hypercube experiments is very fast and simple. The SEED method requires a lot of time, most of which is spent on human interference – handling input and output files, transferring information, making decisions, etc. Maximum entropy sampling is very time consuming when there are many design variables and/or when we want to allocate a lot of data points in one step. This is the reason why the authors in (Currin, et al., 1991) developed a method in which data points are identified "sequentially", in which information from previous data points is not used in identifying new points (this is why

we still call that method a “single-stage” method, as explained in Chapter 4). The SEED method is similar to the method in (Currin, et al., 1991) except that information of prediction errors is used in the metamodeling process. The computation expense of SEED is slightly higher than that of the S-MES method, but the difference should not be very significant. The handling expense (due to human interference) of SEED is much higher than that of the S-MES method because human decisions have to be made in the metamodeling process, and human activities are used in transferring information between programs, developing metamodels of prediction errors, etc. The handling expense of SEED could be reduced a lot by building SEED in a computer framework in which transference of information, development of metamodels, etc., are done automatically with the supervision of human beings; human decisions are still needed in some steps but time spent on the decision-making could be minimized by providing a good human-computer interface.

The comparison above shows that with equal number of data points, the SEED method helps achieve more accurate metamodels than single-stage experimental designs do. The reason is that in SEED, information from validation points is taken into account during metamodeling, while in single-stage designs information from validation points is wasted because it is collected and used only after the metamodels are developed.

5.5.4 Exploration of Solutions for the Design of Pressure Vessels

A compromise Decision Support Problem (C-DSP) is built for the design of pressure vessels; design requirements and goals are described at the beginning of Section 5.5. The mathematical formulation of the C-DSP is presented in Figure 5.58. As described in Section 5.5.2, the factor *T* is identified as unimportant and set as 0.9 inch in metamodeling; thus, in C-DSP and design space exploration, it is not regarded as a design variable but a constant with the value of 0.9 inch.

There are three system constraints. The first two system constraints, $g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$, are automatically satisfied in the given design space, as we described at the beginning of Section 5.5; thus in design space exploration, we only need to consider the constraint of $g_3(\boldsymbol{x})$. The metamodel of response for *Vol* as developed in Section 5.5.2 could be used in $g_3(\boldsymbol{x})$.

There are two design goals, one is to maximize the tank volume, and the other is to minimize the cost. The metamodels developed in Section 5.5.2 are used in the C-DSP to replace the simulation code (in this example, Equations (5.3) and (5.4)). In this example, the usage of metamodels does not help reduce computation time and effort; actually, it increases the computation time because the kriging metamodel is more complicated than the simple equations for system responses. However, this is a demonstration of our sequential experimental design and metamodeling approach; saving on computation time and effort is not a goal in this section. In this example, since we want to maximize the tank volume and minimize the cost, we use different formulations of the systems goals. We set $Vol_{\text{target}} = 700{,}000$, and $Cost_{\text{target}} = 3000$.

Given:

System variables *R*, *L*, and their ranges.
System constraints and goals.
T = 0.9 inch.

Find:

- *Values of independent system variables:* Cylinder radius, R, Cylinder length, L
- *Values of deviation variables:* d_i^-, d_i^+, i = 1, 2

Satisfy:

- *System Constraints:*

$g_1(\boldsymbol{x}) = 1 - 0.0193R \geq 0$
$g_2(\boldsymbol{x}) = T - 0.00954R \geq 0$
$g_3(\boldsymbol{x}) = Vol(\boldsymbol{x}) - 1.296E5 \geq 0$

- *System Goals:*

To maximize *Vol*:

$$Vol\,(\boldsymbol{x}) / Vol_{target} + d_1^- - d_1^+ = 1$$

To minimize *Cost*:

$$Cost_{target} / Cost(\boldsymbol{x}) + d_2^- - d_2^+ = 1$$

- *Bounds:*

10 in. ≤ R ≤ 50 in.
10 in. ≤ L ≤ 100 in.
$d_i^-, d_i^+ \geq 0,\ d_i^- \cdot d_i^+ = 0;\ i = 1, 2$

Minimize:

Preemptive deviation function (lexicographic minimum):

$Z = w_1 * d_1^- + w_2 * d_2^-$, where $w_1 = w_2 = 0.5$

Figure 5.58 Mathematical Formulation of C-DSP for Pressure Vessel Design

In Figure 5.58, we show the Archimedean deviation function in which both design goals are equally weighed. To obtain more general knowledge, we may need to study different design scenarios, e.g., Archimedean deviation functions with unequally weighed design goals, preemptive deviation functions, etc. However, in this study, we will only

explore for solutions with the given Archimedean deviation function since our focus here is on the sequential experimental design and metamodeling process, not the acquiesce of solutions for the pressure vessel design. Given the formulation of design goals in the compromise DSP in Figure 5.58, d_1^+ and d_2^+ do not play roles in the Archimedean formulation because their values are always zero before the design goals are achieved.

Solving the compromise DSP in Figure 5.58, we got the solution as presented in Table 5.39. Note that this solution is obtained with metamodels developed in Section 5.5.2. As a comparison, the compromise DSP is re-solved with simulations, i.e., theoretical mathematical functions in Equations (5.3) and (5.4); no metamodel is used in this formulation. The result is also listed in Table 5.39; note that this result could be regarded as the "true" solution based on given design requirements and goals. The compromise DSP are also solved with metamodels developed with single-stage experimental designs as described in Section 5.5.3.

Table 5.39 Design Solutions Obtained by Solving the C-DSP

	R	***L***	***T***	***Vol***	***Cost***	***Vol_pred***	***Cost_pred***	D_1^-	D_2^-
C-DSP with Simulation	44.746	51.625	0.9	699999.90	5693.05	–	–	0.00	0.4730
C-DSP with Metamodels from SEED	44.75	51.62	0.9	700026.14	5693.18	699992.09	5640.52	0.00	0.4681
C-DSP with Metamodels from S-MES	44.76	48.86	0.9	683012.87	5609.31	699996.86	5616.18	0.00	0.4658
C-DSP with Metamodels from LH	46.72	55.32	0.9	806541.79	6203.87	699999.55	6062.94	0.00	0.5052

In Table 5.39 we see that solutions from C-DSP with metamodels are close to that from C-DSP with simulation (the "true" solution), which indicates that the utilization of sequential experiments and metamodels is effective in finding out the design solutions. The sequential metamodeling approach described in Section 5.4 is effective; the application of RS, kriging, and MARS metamodeling techniques is appropriate in the example problem.

Comparing solutions based on metamodels from SEED, S-MES, and LH, we find out that the best solution is achieved with the metamodel from SEED, which is very close (within ±0.01) to the true solution. Metamodels from the Latin Hypercube design perform worst because its design solution is very far from the "true" solution. The solution with metamodels from SEED is closest to that obtained with "actual simulations"; values of responses of *Vol* and *Cost* are also not far from those of the "true" solution. The solution obtained with metamodels from the single-stage maximum entropy sampling is also very close to the "true" solution. Since the actual response surfaces are not highly nonlinear or irregular (note that Equations (5.3) and (5.4) are 2^{nd}-order or 3^{rd}-order questions), difference between solutions from different metamodels is not very huge. In cases with irregular responses, we expect to achieve more accurate metamodels and better solutions with the sequential experimental design method.

Metamodels with LH are generally more accurate than those from S-MES (as presented in Table 5.38), while the solution from metamodels with S-MES is better than that from metamodels with LH. With the SEED method, we got the most accurate metamodels and best solutions. This indicates that a more accurate metamodel may not

necessarily lead to a better solution, as in the case of LH and S-MES; while with more accurate metamodels it is more likely that we will achieve better solutions, as in the case of SEED.

In this chapter, our focus is on studies for R.Q.4, *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?* An approach for sequential metamodeling is developed and illustrated through studies in Sections 5.2, 5.3, 5.4, and 5.5. This approach is closely related to the SEED method developed in Chapter 4. In Chapter 6, a new approach is to be developed to help integrate processes of metamodeling and design space exploration; studies in Chapter 4 (SEED) and this chapter (the approach of Sequential Metamodeling) will serve as the foundation of the proposed research. Summaries of research in this chapter and its connections with studies in future chapters are presented in the following section.

5.6 A LOOK BACK AND A LOOK FORWARD

The research in this chapter is partly based on our studies in Chapter 4. The SEED method as developed in Chapter 4 serve as an important component in the approach that is developed in this chapter. Work in this chapter, together with that in Chapter 4, provides the foundation of studies in the next chapter (Chapter 6), in which an approach is developed to efficiently explore the design space for design solutions through the integration of the processes of metamodeling and design space exploration. Chapters 4, 5, and 6 are the core of this dissertation, which provide the methodological basis for applications in Chapter 7 and 8.

In this chapter, our focus is on studies for R.Q.4, *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?* To answer this research question, we posed two sub-research questions, as listed below:

> ***R.Q.4:*** *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?*
>
> ***R.Q.4.1:*** *How do different types of metamodels perform in engineering design?*
>
> ***R.Q.4.2:*** *How to select different types of metamodels at different design stages?*

R.Q.4.1 is studied and answered in Sections 5.2 and 5.4. A comparison between kriging and MARS metamodels is done in Section 5.2 with some interesting observations. The comparison between RS and kriging metamodels has been done in previous work in (Simpson, 1998) and (Lin, 2000), and comparisons between more types of metamodels could be a future work of this dissertation. In our studies we observe that both kriging and MARS have their strong and weak points; kriging metamodels may not perform appropriately when the properties of the response surface change greatly (i.e., highly nonlinear in some regions while flat in others), and MARS metamodels may meet problems in deterministic applications because they smooth the data and thus the predicted values at data points may not be accurate. A summary on comparison between RS, kriging, and MARS metamodels is presented in Table 5.24 in Section 5.4, before the

development of the approach for sequential metamodeling. This could be viewed as the answer to R.Q.4.1.

Based on the studies in Section 5.2, the SEED method is extended in Section 5.3 by utilizing both kriging and MARS metamodels. This helps answer R.Q.4.2. Kriging and MARS may be appropriate, or, on the other hand, inappropriate, in different situations; thus we recommend that both be used to develop metamodels in sequential experimental design and metamodeling. Designers could make decisions only after building the metamodels and observing their performance. A recommendation on how to use kriging and MARS metamodels is described in Section 5.3.

R.Q.4.2 is further studied and answered in Sections 5.4 and 5.5, in which an approach for sequential metamodeling is developed and illustrated with an engineering example. The framework for the approach of sequential metamodeling is presented in Figure 5.40 and Figure 5.41 in Section 5.4. In Section 5.5, we modified our SEED mathematical formulations introduced in Chapter 4 to account for multiple system responses. An engineering example of pressure vessel design is used to illustrate the approach of sequential metamodeling introduced in Section 5.4 and the handling of multiple responses described in Section 5.5.1.

In our studies in Section 5.5, one interesting observation is that MARS metamodels may work abnormally in response prediction: in some cases the prediction errors of MARS metamodels at observed points are dramatically large, while in our previous studies, MARS metamodels used to have very small prediction errors at observed points though theoretically they do not predict exactly at observed points. In

such cases, kriging metamodels are developed to overcome this shortcoming. This confirms our previous recommendations made in Section 5.3, in which we propose to develop metamodels with both kriging and MARS techniques and designers could select appropriate ones in design.

R.Q.4 is answered based on all studies in Sections 5.2, 5.3, 5.4, and 5.5. Our answer to R.Q.4 is: various types of metamodels could be developed and utilized in the design process following the approach of sequential metamodeling as described in Section 5.4 and 5.5.

Our research in this chapter not only helps answer R.Q.4, but also provides some augments to the SEED method, which is related to R.Q.2:

> ***R.Q.2:*** *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

The approach of sequential metamodeling is developed partly based on the SEED method (which is developed in Chapter 4); to some extent, it could also be viewed as an extension of and augment to the SEED method. We propose to use both kriging and MARS metamodels to handle information from current data/validation points. This is done and illustrated in Section 5.3 and Section 5.5.

The development of the approach for sequential metamodeling also answers Research Question 3.2:

R.Q.3.2: *How to reduce the design space with information from previous metamodeling and design space exploration?*

The usage of RS metamodels at the very early stages of metamodeling helps identify and screen unimportant design variables. This is described in Section 5.4, and illustrated with the pressure vessel design problem in Section 5.5. Another way to reduce the design space is to reduce the ranges of design variables; however, we will not go further on this research and leave it as a future work for this dissertation.

Research in Chapters 4 and 5, i.e., development of the SEED method and the approach for sequential metamodeling, provides the foundation for our work in the following chapter (Chapter 6), in which the Efficient Robust Concept Exploration Method (E-RCEM) is developed to facilitate efficient metamodeling and design space exploration through the integration of these two processes. Chapters 4, 5, and 6 form the core of this dissertation, which provides the methodological foundation of our applications in Chapter 7.

CHAPTER 6

THE EFFICIENT ROBUST CONCEPT EXPLORATION METHOD: INTEGRATION OF PROCESSES OF METAMODELING AND DESIGN SPACE EXPLORATION

In this chapter, our focus is on the development of the Efficient Robust Concept Exploration Method, in which the processes of metamodeling and design space exploration are integrated. Research questions answered in this chapter are R.Q.3 and two of its sub-research questions, R.Q.3.1 and R.Q.3.3. After a discussion on current design and metamodeling processes and the proposal of the integration of these processes in Section 6.1, Research Question 3.1 is answered in Section 6.2 through the study of incorporating design constraints in the metamodeling process. This study is further extended in Section 6.3 where Research Question 3.3 is answered. The Efficient Robust Concept Exploration Method (E-RCEM), which enables designers to develop metamodels and get design solutions efficiently and effectively at early design stages through the integration of metamodeling and design space exploration, is then developed in Section 6.4. An application of the Efficient Robust Concept Exploration Method is presented in Section 6.5.

6.1 PROCESSES OF METAMODELING AND DESIGN SPACE EXPLORATION AT EARLY DESIGN STAGES

As introduced in Chapter 2, the purpose of metamodeling is to develop acceptable metamodels that helps designers integrate multi-disciplinary analysis codes, gain insights into the relationship between inputs and outputs, and then explore the design space efficiently for design solutions in later design stages. Metamodeling is a very important process in early-stage design. Design space exploration is a process in which designers explore the whole design space for solutions that satisfy design constraints and achieve design goals; various optimization techniques could be used in this process, and the usage of metamodels could help save a lot of computation time. Typically, the two processes, metamodeling and design space exploration, are separated and conducted sequentially in applications, as illustrated in Figure 6.1.

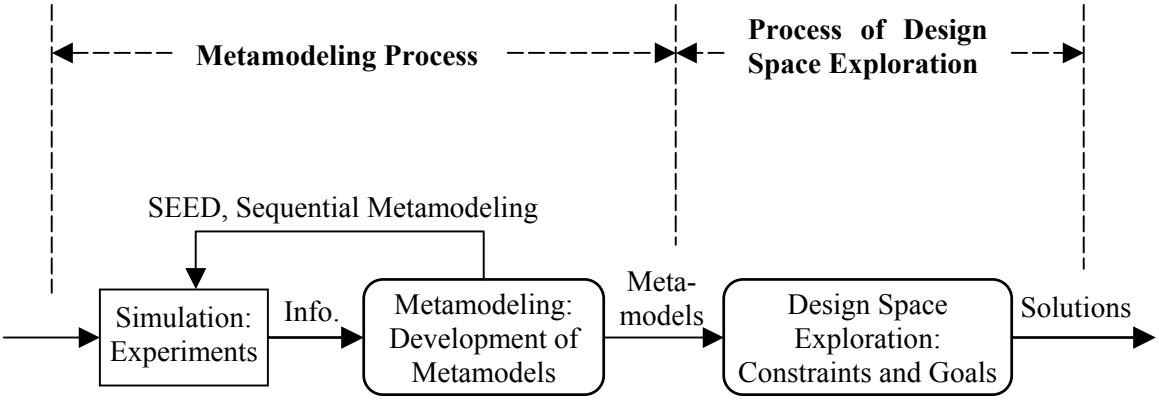


Figure 6.1 Traditional Organization of Processes of Metamodeling and Design Space Exploration

As illustrated in Figure 6.1, given an expensive simulation, usually we should design experiments, then develop metamodels for system responses based on the

information collected at data points. In the SEED method and the sequential metamodeling approach as described in Chapter 4 and Chapter 5, we design experiments and develop metamodels in iterations to ensure that acceptable metamodels be acquired. This is the process of metamodeling. After finishing the metamodeling process, we enter the process of design space exploration, in which the metamodels developed in the previous process are used in the exploration of design solutions. System constraints and goals are considered in the process of design space exploration.

This sequential organization of processes of metamodeling and design space exploration is widely used in engineering design. For example, in many applications of Taguchi's robust design (Taguchi, 1987), physical experiments are first designed and some statistics, e.g., signal-to-noise ratios, are developed (similar to our concept of "metamodels"); then the robust design solutions are found by analyzing the signal-to-noise ratios, which corresponds to the process of design space exploration in Figure 6.1. In the Robust Concept Exploration Method (Figure 6.2), the sequential application of metamodeling and design space exploration is apparent: the process metamodeling is realized in Processors B, C, D, and E, and the process of design space exploration is realized in Processor F.

One advantage of doing metamodeling and design space exploration sequentially lies in its simplicity. The framework is clear and designers need only follow steps to get design solutions, avoiding backward information flows. The objectives in each process are also very clear: in the metamodeling process, the objective is to build acceptable metamodels, and in the process of design space exploration, the objectives are to achieve

design goals and satisfying design constraints. As a result of this clarification of objectives and steps, designers' load is minimized because they need not deal with tangled design requirements and information flows.

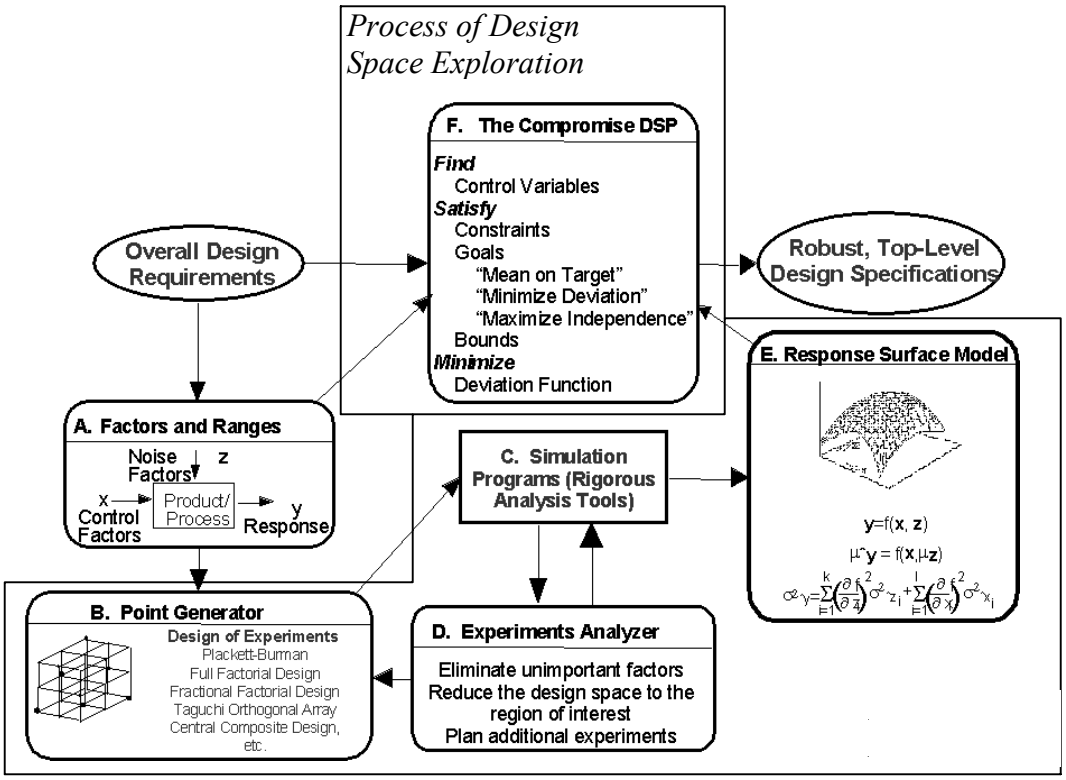


Figure 6.2 The Robust Concept Exploration Method (adapted from Chen, et al., 1996a)

On the other hand, the sequential organization of processes of metamodeling and design space exploration also has its disadvantages, as discussed below.

In the sequential organization of processes of metamodeling and design space exploration, objectives of the two processes need to be clearly defined to ensure the achievement of good design solutions efficiently and effectively. Since the objective of

the metamodel process (to build accurate metamodels) is different from that of the whole design process (to get design solutions that achieve design goals and satisfy design constraints), designers need to define the objective of metamodeling very clearly and carefully. Two questions of importance are: which type of metamodels should be used? How accurate the metamodels should be? The first question is related to the activities that designers plan to do in the process of design space exploration, e.g., some type of metamodels do not perform well in design space exploration of robust solutions (see, Lin, et al., 1999), while others may be so complicated that it may cost a lot of time and effort in design space exploration. The second question is still not well addressed in current literature. It is also related to the first question on types of metamodels. Lin and co-authors (Lin, et al., 1999) discussed on *Local Model Inaccuracy* and its effects on the achievement of design solutions; similar ideas are presented in (Jin, et al., 2001), which states more on the side of types of metamodels used. Designers have to answer these two questions and clarify the objective of the metamodeling process before conducting a successful and efficient design.

It is very possible that designers waste a lot of time, effort, and money on experiments at infeasible points (points outside of the feasible design space) in the metamodeling process since design constraints usually have no influence in design of experiments. Usually, the initial design space used by designers in metamodeling is a multi-dimensional "hyper cubes" with preset ranges for design variables. Most of current widely used DOE techniques, e.g., factorial design, Latin Hypercubes, Orthogonal Arrays, etc., are suitable for such design spaces; they are not suitable for experimental

designs in a feasible design space which is usually irregular as a result of the consideration of various design constraints. Since design constraints are usually not considered in the metamodeling process, it is very possible that designers spend a great deal of time and money on experiments at infeasible points.

When there is no design constraint, (or say, the feasible design space is "regular"), it is still very possible that a lot of time, effort, and money is wasted on experiments at "unimportant" points. Note that the objective of the early-stage design process in Figure 6.1 is to achieve a good design solution; points far from this solution are considered "less important" than those close to the solution. What designers pursue and eventually obtain in the early-stage design process is only the solution, while to achieve this solution a lot of time, effort, and money have to be spent on observations at numerous points in the feasible design space. Among these points some are close to the design solution, while most others not. Since design goals are not considered in the metamodeling process (note that there is no information flow from the process of design space exploration to metamodeling), all points in the design space are considered to be equally important in achieving design goals, and as a result, it may cost a lot to ensure the metamodel accuracy in some local regions that are far from the solution – and this is a waste when we review the design process after obtaining the design solution.

As discussed above, the sequential organization of processes of metamodeling and design space exploration is inefficient and ineffective in the achievement of design solutions, especially when the simulation (or physical experiment) is very expensive. To overcome the discussed shortcomings, we need to develop a method in which the two

processes are integrated. To be specific, first we need to integrate the consideration of design constraints in the metamodeling process, and second we need to integrate the consideration of design goals in metamodeling; in other words, we can also say that we should integrate metamodeling in exploring the design space for solutions that achieves design goals and satisfy design constraints. Another requirement for this integration is that the developed method should be organized clearly so that designers could follow it step by step, without getting lost in the complicated information flow which is expected to come with the integration.

The study of applying design constraints in metamodeling is done in Section 6.2, and that of applying design goals in metamodeling is done in Section 6.3. The Efficient Robust Concept Exploration Method (E-RCEM) is then developed and described as a result of the research in Sections 6.2 and 6.3.

6.2 METAMODELING WITH CONSIDERATION OF DESIGN CONSTRATINTS

There are basically two types of constraints in design space exploration. One is the constraint put on design variables, e.g., in the design of pressure vessel in Chapter 5, we have a constraint associated with the wall thickness T and the radius of the spherical head R (see Equation (5.6)). The other type of constraints is put on the responses, e.g., also in the design of pressure vessel in Chapter 5, a constraint is associated with one of the responses, Vol (see Equation (5.7)). In examples with constraints only associated with design variables, the feasible design space is actually clearly defined though sometimes it is not easy to draw the boundaries. In cases with constraints associated with

responses (and metamodels of these responses), the feasible design space could not be clearly defined in design because of the uncertainty associated with the metamodel; with current metamodel, the boundary of the feasible design space could be drawn but since the metamodel is not 100% accurate, this boundary is "vague" with some degree of uncertainty.

No matter which types of constraints are used in the problem, it is very possible that the feasible design space is not "hyper cubes" as in most experimental designs. Classical experiments, e.g., factorial designs, and some space-filling experiments, e.g., Latin Hypercubes, are most suitable with regular design spaces that we may regard as "hyper cubes". Maximum entropy sampling is still appropriate in dealing with irregular design spaces; quite a lot research has been done (e.g., see Anstreicher, et al., 1996; Vandenberghe, et al., 1998; Lee and Williams, 1999; etc.) to address this. The Constrained D-Optimality Problem (CDOPTP) and the Constrained Maximum-Entropy Sampling Problem (CMESP) are both fundamental problems in experimental design.

In this section, we will not do or follow the theoretical (mathematical) work on Constrained Maximum Entropy Sampling. Instead, using the example of design of pressure vessels in Chapter 5 (with small modifications), we empirically study the application of the SEED method in designing experiments with two types of constraints. In our study, the way to consider constraints is intuitive and direct (without complex mathematical deduction or algorithms); it could be a future work to incorporate previous results of Constrained Maximum Entropy Sampling in the SEED method.

In Section 6.2.1, we will incorporate constraints on design variables in designing sequential experiments. The constraints on design responses are considered in designing sequential experiments in Section 6.2.2.

6.2.1 Sequential Experimental Design and Metamodeling with Consideration of Constraints on Design Variables

In the example of design of pressure vessels in Chapter 5, we identified two important design variables, R and L; in our studies here, only these two variables are considered to facilitate simple applications and illustrations. There were two responses in our studies in Chapter 5, *Vol* and *Cost*; in this section, to be simple we only consider the response of *Vol*. The original design space in the example in Chapter 5 is very small and some of the system constraints (Equations (5.5) and (5.6)) are automatically satisfied; in this section, the original design space is enlarged so that system constraints are active. The ranges of design variables are:

$$25 \text{ in.} \le R \le 150 \text{ in.}$$

$$25 \text{ in.} \le L \le 140 \text{ in.}$$

The system response *Vol* could be calculated (simulated) with the following equation:

$$Vol. = \pi R^2 L + \frac{4}{3}\pi R^3 \tag{6.1}$$

Two system constraints are put on design variables:

$$1 - 0.0193R \ge 0 \tag{6.2}$$

$$T - 0.00954R \ge 0 \tag{6.3}$$

Since the design variable T is not used in this example, Equation (6.3) will not be considered in the study. Instead, according to some customers' requirements (e.g.,

assembling compatibility with other equipments, etc.), we may put other constraints on design variables:

$$L - 0.5R \geq 0 \tag{6.4}$$

$$L - 1.5R \leq 0 \tag{6.5}$$

One system constraint is put on the response:

$$Vol - 1.296\text{E}5 \geq 0 \tag{6.6}$$

In this section we will only study the incorporation of constraints on design variables in designing sequential experiments, the constraint in Equation (6.6) will not be considered. Now our aim is to develop an acceptable metamodel for the system response *Vol* in the feasible design space that is decided by ranges of design variables and three system constraints (Equations (6.2), (6.4), and (6.5)). First, let us have a look at the "feasible" design space constructed based on the factor ranges and constraints in Equations (6.2), (6.4), and (6.5).

The original design space is a 125in. × 115in. rectangular set by the ranges of the two design variables. Three constraints are posed on the design variables and form clear boundaries for the "quasi-feasible" design space; here we use "quasi-" because we do not consider the effect of the constraint posed on the response (Equation (6.6)). The "quasi-feasible" design space is illustrated in Figure 6.3 marked in red shadow. In Figure 6.3 Constraint I corresponds to Equation (6.2); Constraint II corresponds to Equation (6.4); Constraint III corresponds to Equation (6.5). The "quasi-feasible" design space has clear boundaries because the constraints considered in Figure 6.3 are posed on design variables only. At the beginning of metamodeling, we have no information on the responses

(metamodels, simulations, etc.), thus the constraint associated with system responses (Equation (6.6)) is not considered in Figure 6.3. Note that this design space is convex.

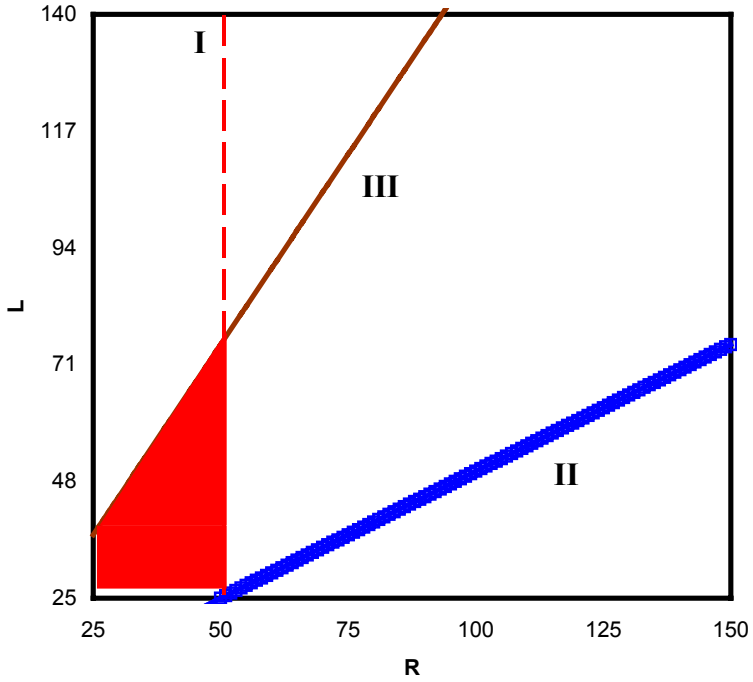


Figure 6.3 Quasi-Feasible Design Space with 3 Constraints on Design Variables

The initial experiments are designed within the quasi-feasible design space in Figure 6.3. The data points in the initial experimental design are listed in Table 6.1. An initial kriging metamodel is developed based on information from these 6 data points. This corresponds to Step 1 and Step 2 of the SEED method. Six validation points are then identified and listed in Table 6.2; this corresponds to Step 3 of the SEED method. Note that in the metamodeling processes in this study, we normalized the initial ranges of design variables to [0, 1]. Also, in these steps we use $\theta_1 = \theta_2 = 20$ to calculate entries of the covariance matrices. To pose constraints in identifying data/validation points is easy

to realize in iSIGHT by eliminating points that do not satisfy the constraints (note that in Chapter 4 and Chapter 5 we applied the SEED method is in iSIGHT); the organizations of tasks, information flows, and the calculation of constraints, etc. in iSIGHT is presented in Appendix C.

Table 6.1 Four Data Points

R	*L*	*R_n*	*L_n*	*Vol*
25	25	0	0	114537.23
51.8125	26.0235	0.2145	0.0089	802104.03
51.8125	77.716	0.2145	0.4584	1238063.89
35.6	53.3935	0.0848	0.2469	401577.67

Table 6.2 Four Validation Points

R	*L*	*R_n*	*L_n*	*Vol*
38.25	25	0.106	0	349322.33
44.7875	39.444	0.1583	0.1256	624889.11
25.125	37.512	0.001	0.1088	140829.54
51.7625	45.47	0.2141	0.178	963685.39

The data points and validation points are illustrated in Figure 6.4. In Figure 6.4, black solid crosses represent data points, and red solid triangular represent validation points. We successfully put data points (and validation points) in the irregular design space through maximum entropy sampling.

After the initial experiments, a kriging metamodel is developed for *Vol* with information from 4 data points in Table 6.1. For this kriging metamodel, we got θ_1 = 27.47586 for the design variable *R*, and θ_2 = 1.55966 for the design variable *L*. The

contour plot of this kriging metamodel is illustrated in Figure 6.5. To test the accuracy of this metamodel, we collected information at 356 points; values of NMAX and NRMSE are calculated, and we get NMAX = 0.111 and NRMSE = 0.051. The prediction errors of this metamodel are zero at data points; prediction errors at validation points are listed in Table 6.3.

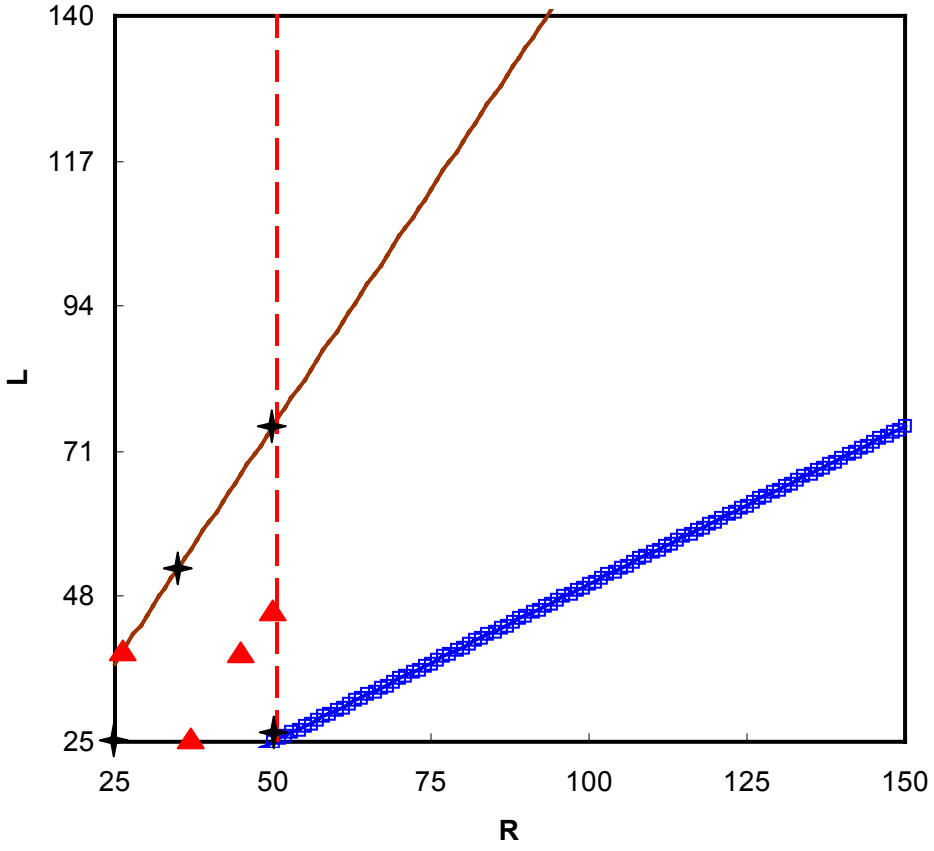


Figure 6.4 Data and Validation Points in the Quasi-Feasible Design Space

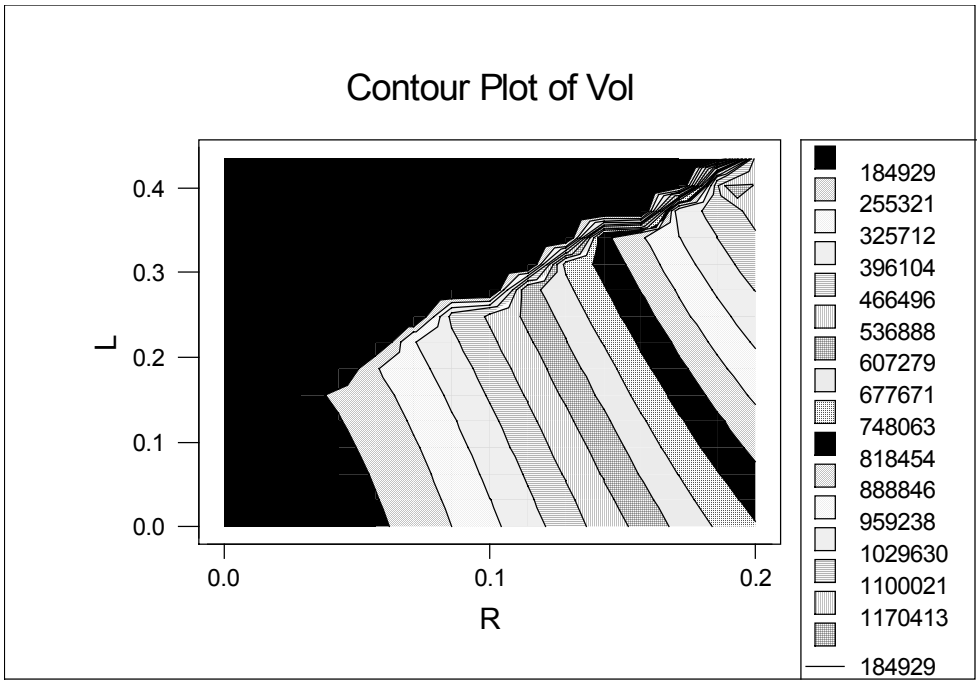


Figure 6.5 Contour Plot of Kriging Metamodels for Volume with 4 Data Points

Table 6.3 Prediction Errors at Validation Points

R_n	*L_n*	*Vol*	*Vol_pred*	*Prediction Error*
0.106	0	349322.33	333084.24	-16238.09
0.1583	0.1256	624889.11	675782.52	50893.40
0.001	0.1088	140829.54	128348.83	-12480.71
0.2141	0.178	963685.39	978560.85	14875.45

Following the steps in SEED, after calculating prediction errors at validation points, the next step is to develop the metamodel of prediction errors. As we did in Chapter 5, a MARS metamodel is developed to predict prediction errors at candidate points; this MARS metamodel of prediction errors is proved to be inappropriate since we found that its predicted values at data and validation points are far from the actual ones.

Thus we develop a kriging metamodel for predicting prediction errors. For this kriging metamodel, we got θ_1 = 999.98841, and θ_2 = 18.08955. The contour plot of this kriging metamodel is illustrated in Figure 6.6.

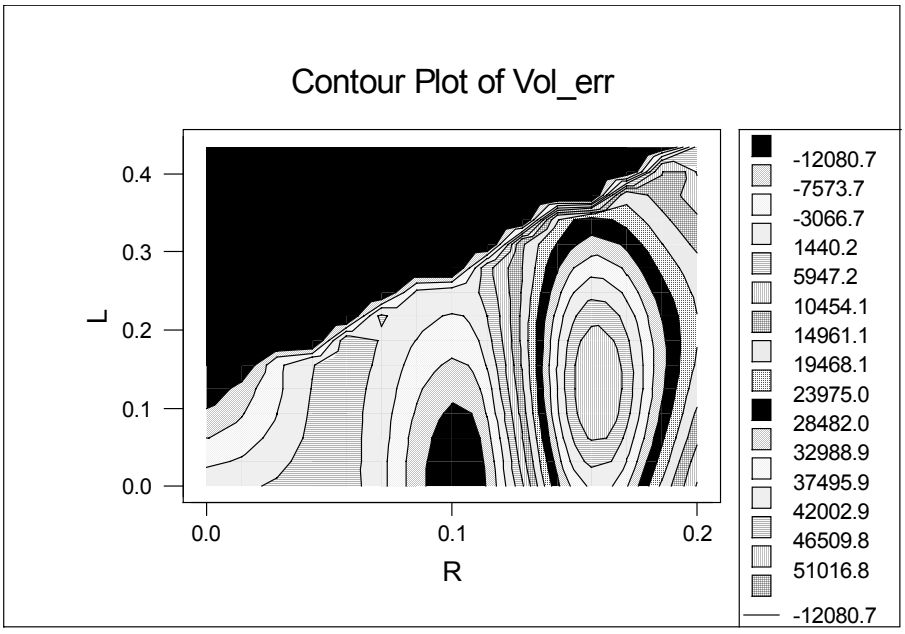


Figure 6.6 Contour Plot of Predicted Prediction Errors (with 4 Data Points and 4 Validation Points)

The maximum absolute value of predicted prediction errors is around 51020. Following the steps in SEED as described in Chapter 4 and Chapter 5, we adjust the covariance matrix and identify 2 new data points, as listed in Table 6.4. The 6 data points and 4 validation points are illustrated in Figure 6.8.

A final kriging metamodel is developed for *Vol* with information from 10 observed points; the contour plot of this metamodel is illustrated in Figure 6.7. For this

kriging metamodel, we got $\theta_1 = 1.77631$, and $\theta_2 = 0.03167$. Based on information from 356 points we test the accuracy of this kriging metamodel and get NMAX = 0.0007 and NRMSE = 0.0003. We see that new data points are successfully identified in the irregular quasi-feasible design space with the SEED method to help obtain most potential information.

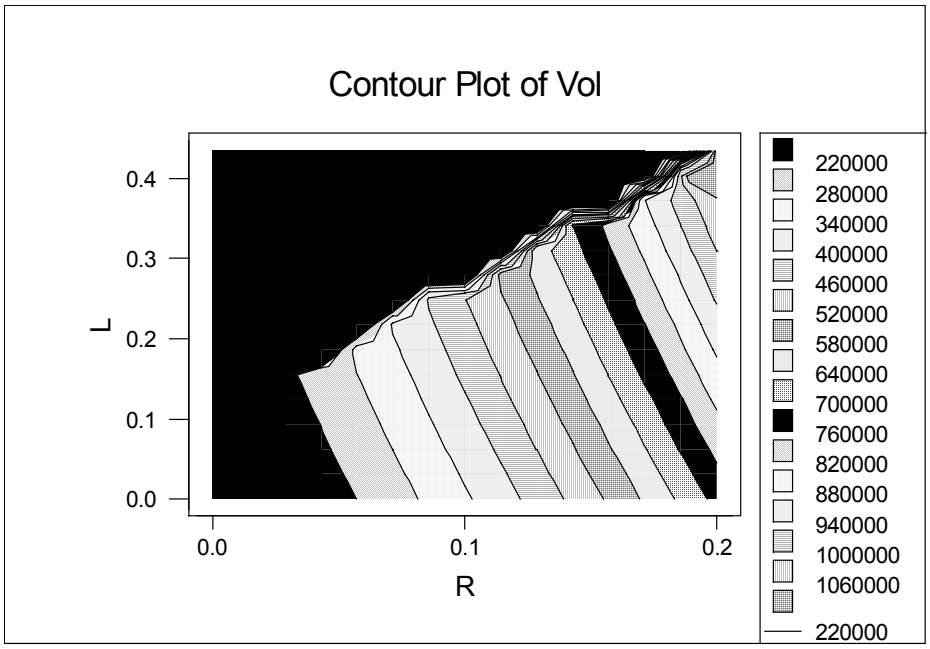


Figure 6.7 Contour Plot of Kriging Metamodel of Vol with 10 Observed Points

Table 6.4 Two New Data Points

R	*L*	*R_n*	*L_n*	*Vol*
45.3625	63.2145	0.1629	0.3323	799661.03
25.9375	38.9035	0.0075	0.1209	155315.80

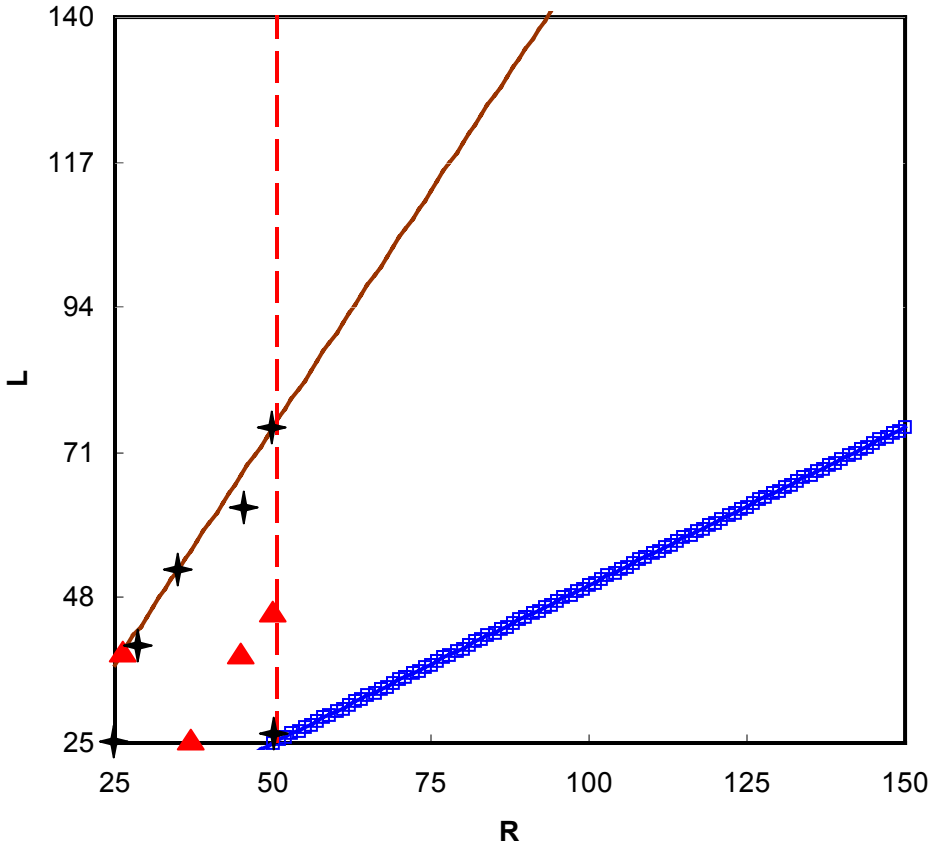


Figure 6.8 Eight Data Points and Six Validation Points

In this practice we see that initial experiments could be designed in an irregular quasi-feasible design space. New validation and data points could be identified following steps in the SEED method. As discussed before, constrained maximum entropy sampling is not the focus of our research in this dissertation, and the study in this section is only a supportive step for our development of the Efficient Robust Concept Exploration Method (E-RCEM) in this chapter. In this sub-section we have shown that we are able to deal with irregular design spaces with constraints on design variables with

the algorithms developed for SEED. In Section 6.2.2, we will go further to study the application of SEED algorithms in problems with constraints on system responses.

6.2.2 Sequential Experimental Design and Metamodeling with Consideration of Constraints on Responses

In this section, we study the application of SEED in irregular design spaces defined by constraints on responses. In cases where constraints are only put on design variables, boundaries of design spaces are clear and fixed; there is no uncertainty associated with the design space. In cases where constraints are put on responses, boundaries of design spaces are vague and subject to change as the metamodels evolve; uncertainty plays an important role here. Boundaries of design spaces tend to be less vague (uncertainty of design spaces reduces) as more and more data points are added and more and more accurate metamodels of system responses are obtained. In the metamodeling process with consideration of system constraints on responses, we should pay attention to the following things in this dissertation:

- With current metamodels we are able to define a small design space (expected to be irregular). However, we need to consider the uncertainty associated with boundaries of this design space.
- In sequential experimental design and metamodeling, metamodels should be developed for responses in the whole (initial) design space instead of the reduced design space, thus more and more accurate boundaries could be identified in the metamodeling process.

- In our studies in this dissertation, we only deal with irregular design spaces that are neither isolated nor concave. In other words, we only study and apply our methods in problems with continuous, convex design spaces. Studies with concave or discrete design spaces may be a future work for this dissertation.

There are many methods to address the uncertainty with boundaries of design spaces in the metamodeling process, with the keywords of "reliability" or "uncertainty" in literature. For example, in (Du and Chen, 2000) and (Du and Chen, 2001), the authors examined several feasibility-modeling techniques and proposed a most probable point (MPP) based importance sampling method for evaluating the feasibility robustness. In (Gu, et al., 2000), the authors investigate how uncertainty propagates through a multidisciplinary system analysis subject to the bias errors associated with the disciplinary design tools and the precision errors in the inputs is undertaken; a method of worst case estimation of uncertainty is then integrated into a robust optimization framework. It is future work of research in this dissertation to incorporate such methods in SEED or develop new methods that suits SEED better. In this section, a preliminary observation is done in addressing boundaries of design space, which simply serves as a support for the development of E-RCEM (the Efficient Robust Concept Exploration Method) in this chapter.

To address the boundaries of an irregular design space, one possible way is to develop confidence intervals for the boundaries, and new boundaries could be identified with a certain confidence level; it is expected that the new boundaries be obtained by pushing current boundaries outwards the design space, thus the new design space should

be a little larger than previous ones (all expected to be irregular). This method is not studied here and may be a future work for this dissertation. Another method to address the uncertainty with boundaries of design spaces is to utilize information from metamodels of prediction errors. In identifying boundaries of the design space, we should consider not only the information from metamodels of responses (e.g., values of *Vol* should be larger than some preset constant), but also information from metamodels of prediction errors; in this case, the prediction errors could be considered a measure (or reflection) of uncertainty. Suppose we have an irregular design space with boundaries identified by calculating response values with metamodels of responses, now we should push the boundaries outwards to new ones whose points safely satisfy the constraints put on responses, even when the effect of prediction errors (absolute prediction errors are recommended) is added to the response values. More and more accurate boundaries could be identified and used after iterations of metamodeling in SEED. Also, this method is not studied and used in this dissertation because it is very likely that concave or discrete design spaces would be developed, which is not in the scope of studies in this dissertation.

In our studies in this section, we simply release the constraints on responses to some extent to hopefully address a good portion of the uncertainty with boundaries of the new design space. With information from data/validation points, the normalized root mean squared error (NRMSE) could be calculated following Equation (2.9) and discussions in Section 2.2.3. As explained in Section 2.2.3, usually NRMSE has a value between 0 and 1 (though in some cases it could be larger), and this could be used as a

reference on how much we should push the boundaries outwards. For example, if the constraint is to have a system response y larger than a fixed constant y_0; the minimum observed response value is y_{min} (suppose $y_{min} \le y_0$) and the value of NRMSE is t%. What we do is to release the constraint by t%, i.e., draw the boundary to satisfy

$$y = y_0 - (y_0 - y_{min}) \cdot t\% \tag{6.7}$$

It should be noted that this method is not theoretically solid because the uncertainty associated with boundaries of the design space should be considered as *Local Model Inaccuracy* (Lin, et al., 1999) and the value of NRMSE is a measure of *Global Model Inaccuracy* (see, Lin, 2000). To avoid this problem, we may use the normalized maximum absolute error (NMAX, Equation (2.7) and discussions in Section 2.2.3) to replace NRMSE. Another way to solve this problem, and which is the most intuitive way, is to calculate the average absolute error (AAE) or root mean squared error (RMSE) for the response with Equations (2.8) or (2.9), and then release the constraint correspondingly. Given the problem statement in the paragraph above, supposing the value of AAE or RMSE is Err, we could draw the new boundary to satisfy:

$$y = y_0 - \text{Err} \tag{6.8}$$

The identified new boundaries with this method are not guaranteed to be accurate; the uncertainty may be under- or over- estimated. However, after iterations of metamodeling in SEED, it is expected that more accurate boundaries could be obtained, which helps yield better results in sequential experimental design and modeling with same effort because of the reduced design space.

Now let us look at the pressure vessel example in Chapter 5. In this section we only consider two design variables R and L and one system response Vol. The original design space is 10in.≤ R ≤50in. and 10in.≤ L ≤100in. In this design space, system constraints put on design variables (Equations (5.5) and (5.6)) are automatically satisfied. Thus we only need to consider the constraint posed on the system response Vol, i.e., Vol – 1.296E5 ≥ 0 (Equation (5.7)). The actual feasible design space is illustrated by shadows in Figure 6.10.

Suppose now we have got 6 data points and 6 validation points, as listed in Table 6.5 (the first 6 rows correspond to data points and last 6 rows correspond to validation points). Now we have a kriging metamodel for the response Vol, as illustrated in Figure 6.9. For the kriging metamodel of Vol, we got θ_1=1.43001 and θ_2=0.37760. The prediction errors of the metamodel at data and validation points are listed in Table 5.41.

Using Equation (2.7) and following descriptions in Section 2.2.3, we get the normalized maximum absolute error with 6 validation points as NMAX = 7.2%. Following Equation (6.7), we decide to release the constraint to Vol – 1.2E5 ≥ 0. We can also calculate the value of root mean squared error with Equation (2.9) and get RMSE = 59507; following Equation (6.8), we can release the constraint to Vol – 7.0E4 ≥ 0. The constraint can also be released to Vol – 7.83E4 ≥ 0 if we use the average absolute error which is AAE = 51274 in this example. Thus, we have several possible new boundaries, which are illustrated in Figure 6.10.

Table 6.5 Initial Experiments – Six Data Points and Six Validation Points

R	*L*	*R_n*	*L_n*	*Vol*
10	10	0	0	7330.38
50	10	1	0	602138.60
10	100	0	1	35604.72
50	100	1	1	1308996.96
30.036	79.102	0.5009	0.7678	337697.71
30	28.387	0.5	0.2043	193359.69
30.036	10.108	0.5009	0.0012	142153.31
18.968	55.225	0.2242	0.5025	91006.70
41.448	55.081	0.7862	0.5009	595538.30
29.984	99.982	0.4996	0.9998	395307.47
10	38.269	0	0.3141	16211.35
50	72.361	1	0.6929	1091920.76

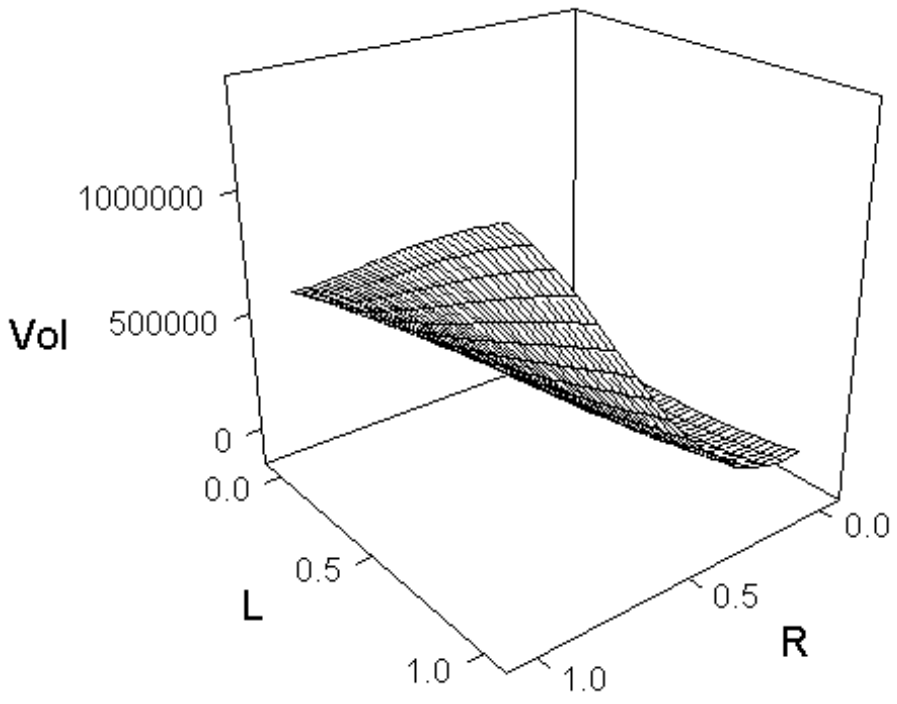


Figure 6.9 Updated Metamodels of Responses with 6 Data Points

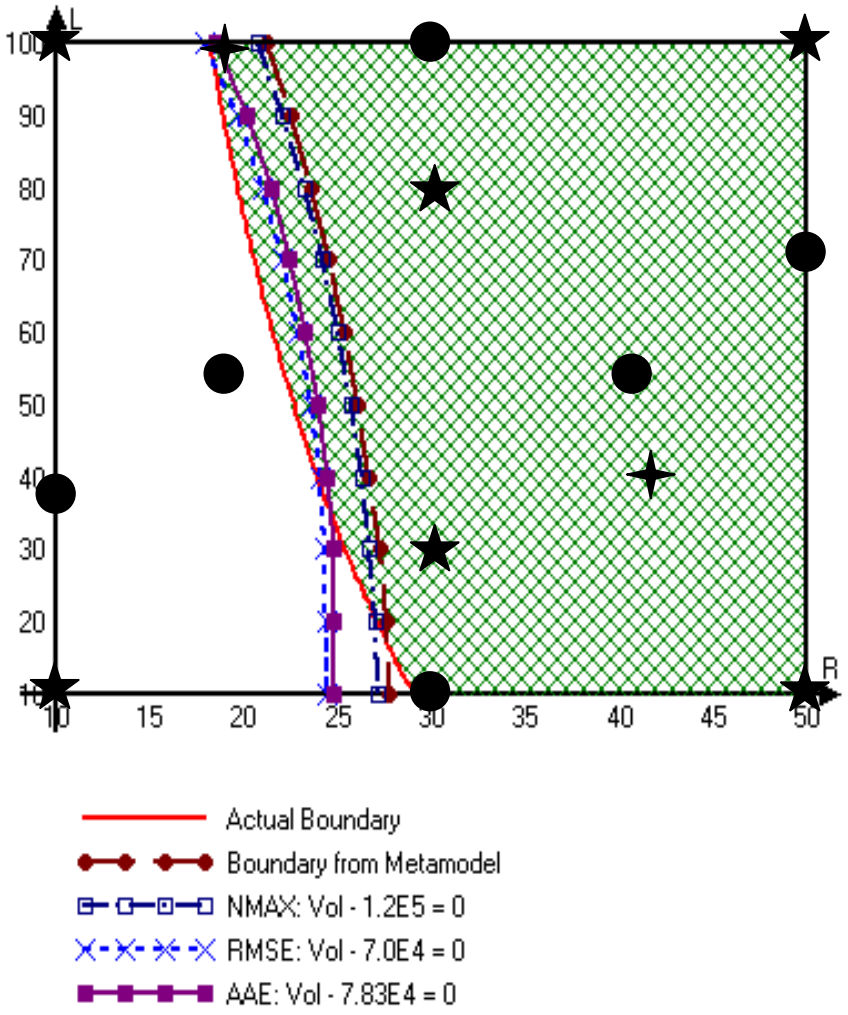


Figure 6.10 The Feasible Design Space and Boundaries

In Figure 6.10 stars represent data points and solid circles represent validation points. In Figure 6.10 we see that boundaries calculated with metamodels do not comply with the actual boundary (the curve on the very left when L=70). After releasing the constraint, new boundaries are still not close to the actual one. It is hard to say which boundary is better than others; in studies in this section, we will use the one calculated

with AAE, i.e., the third curve on the left when L=70, as the left boundary of the constrained design space in this stage of experimental design.

Now we have 6 data points, 6 validation points, metamodels of responses and prediction errors for *Vol*, and an irregular "feasible" design space. To identify 2 new data points, we should follow steps in SEED, adjusting the covariance matrix and doing optimization to maximize the determinant of the covariance matrix. In Figure 6.10 we see that 2 of the data points fall far out of the feasible design space, thus we have only 4 observed data points formulating the 6×6 adjusted covariance matrix. Two new data points are listed in Table 6.6.

Table 6.6 Two New Data Points

R	*L*	*R_n*	*L_n*	*Vol*
44.616	43.768	0.8654	0.3752	645723.3
18.748	99.964	0.2187	0.9996	137986.1

Table 6.7 Two New Data Points Identified When the Constraint on Volume Is Not Considered in SEED

R	*L*	*R_n*	*L_n*	*Vol*
45.308	48.196	0.8827	0.4244	700416.14
18.996	57.79	0.2249	0.531	94225.68

In Figure 6.10 we see that the new data points (represented by solid crosses) are allocated in the feasible design space (the shadowed region). If the constraints of Equation (5.7) have not been considered in the process above, new data points will be identified as in Table 6.7. The second point in Table 6.7, (R, L) = (18.996, 57.79), falls

out of the feasible design space in Figure 6.10. In this example, we see that by considering the constraints on system responses, we avoid locating new points in infeasible design space.

The application of SEED in irregular design spaces with two types of system constraints is preliminarily studied in this section. In Section 6.2.1 and Section 6.2.2 we see that with SEED we are able to save experimental time and effort by locating new points in the feasible design space (typically irregular). In the next section, design goals will be taken into consideration in the SEED process. Then the E-RCEM method will be developed in Section 6.4 based on our observations in Section 6.2 and 6.3.

6.3 METAMODELING WITH CONSIDERATION OF DESIGN GOALS

In order to construct the information flow (or feedback) from the process of design space exploration to the process of metamodeling in Figure 6.1, we need to taken design constraints and goals into consideration in experimental designs and development of metamodels. In Section 6.2, we identified two types of constraints and observed the performance of SEED in irregular design spaces outlined by these constraints. When system constraints are considered, the initial design space (usually a hypercube) is reduced to an irregular one; some boundaries of this feasible design space are clear and fixed, while others are vague and may change when more accurate metamodels are obtained. In either case, the usage of SEED helps reduce experimental time and effort by avoiding locating new points in infeasible regions. In this section, we consider another possible information flow (feedback), i.e., the incorporation of design goals in

metamodeling process, to help achieve design solutions more effectively and efficiently. One existing method, the Efficient Global Optimization (EGO) is briefly introduced in Section 6.3.1, and our practice of metamodeling with consideration of design goals is done in Section 6.3.2.

6.3.1 The Efficient Global Optimization Method

A remarkable and interesting method applying this idea is the Efficient Global Optimization (EGO) developed in (Jones, et al., 1998). The idea of the EGO algorithm is to first fit a metamodel (usually a kriging model) to data collected by evaluating the objective function at a few points. Then, EGO balances between finding the minimum of the surface (assume that the optimization goal is to minimize the response) and improving the approximation by sampling where the prediction error may be high. The prediction error used in EGO follows the equation to calculate prediction mean squared error at any new point:

$$s^2(\boldsymbol{x}^*) = \sigma^2\left[1 - \boldsymbol{r}^T\boldsymbol{R}^{-1}\boldsymbol{r} + \frac{\left(1 - \boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{r}\right)^2}{\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{f}}\right] \tag{6.9}$$

The equation above is the same as Equation (3.7); note that it only has meanings with kriging metamodels. The term $-\boldsymbol{r}^T\boldsymbol{R}^{-1}\boldsymbol{r}$ represents the reduction in prediction error due to the fact that $\boldsymbol{x}^*$ is correlated with the sampled points. The σ^2 here is the same as in Equation (2.21). The term $(1-\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{r})^2/\boldsymbol{f}^T\boldsymbol{R}^{-1}\boldsymbol{f}$ reflects the uncertainty that stems from our not knowing μ exactly, but rather having to estimate it from the data. The prediction error in Equation (6.9) is σ reduced by an amount that depends on how correlated the

new point is to the sampled points. With the stationary assumption as stated in Chapter 4, points far from current observed data points have large prediction mean squared error from Equation (6.9).

In EGO, the expected improvement of an experiment at one new point is a combination of improvement on the optimization goal and improvement on metamodel accuracy, which is calculated with the following equation:

$$E\left[I(x)\right] = \begin{cases} \left(f_{\min} - \hat{y}\right)\Phi\left(\frac{f_{\min} - \hat{y}}{s}\right) + s\phi\left(\frac{f_{\min} - \hat{y}}{s}\right) & if \quad s > 0 \\ 0 & if \quad s = 0 \end{cases} \tag{6.10}$$

In Equation (6.10), $f_{\min}$ is the smallest response value at current data points, $\hat{y}$ is the predicted response value at a candidate point with mean and standard deviation given by the kriging predictor (Equation (2.18)) and its standard error (Equation (6.9)), $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and distribution function, respectively. By exploring for the largest value of the expected improvement, EGO locates the new point where either the predicted value is close to the goal or the prediction standard error is large. More discussion on EGO and its applications can be found in (Schonlau, 1997; Sasena, et al., 2002; etc.).

6.3.2 Incorporation of Design Goals in SEED Metamodeling Processes

The most valuable idea in EGO is the incorporation of optimization and metamodeling processes. However, EGO has its limitations:

1. EGO only works with kriging metamodels since the expected improvement is calculated based on the kriging predictor and standard error.
2. The kriging standard error (Equation (6.9)), which is based on the stationary assumption, is used to reflect prediction errors at a candidate point in EGO. Our previous studies show that this may not be a reliable way to estimate prediction errors.

SEED does not have the shortcomings stated above. In SEED, the stationary assumption is relieved and more accurate estimated prediction errors are obtained in iterations, which is discussed and illustrated in Chapter 4. The improvement and application of SEED with kriging and MARS metamodels is shown in Chapter 5. In this section, we express our method of incorporating design goals in the SEED metamodeling processes.

In SEED, new points are identified in critical regions that are either far from current data points or with large prediction errors. This is achieved by adjusting the covariance matrix with Equation (4.28) or Equation (4.34). Correction coefficients (α_i in Equation (4.28) and β_i in Equation (4.34)) are introduced to address the effect of prediction errors of current metamodels. As a result, weak correlations are given to candidate points with large prediction errors, holding other criteria constant. In a similar way, we can introduce some correction coefficients to represent the effect to design goals in adjusting the covariance matrix. In this chapter, we will perform this study in the

context of SEED – Formulation I, which is based on Equation (4.28). The study with SEED – Formulation II is a topic for future research.

There are many ways to formulate and insert these correction coefficients in the sequential metamodeling process. When both prediction errors and design goals are considered, the adjusted covariance between a candidate point and an existing point can be calculated as:

$$\sigma_{ij}^{adj} = \eta_i \eta_j \alpha_i \alpha_j \sigma_{ij} = \eta_i \eta_j \alpha_i \alpha_j \sigma^2 R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \tag{6.11}$$

In Equation (6.11), α_i is the coefficient to reflect the current metamodel's uncertainty (prediction errors) at point $\boldsymbol{x}_i$, and α_j is the coefficient to reflect the current metamodel's uncertainty at point $\boldsymbol{x}_j$. They are calculated with Equation (4.25). η_i and η_j are coefficients to reflect degrees of achievement of design goals at points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively. Theoretically, η_i and η_j should have values between [0,1). A value close to 0 means that the design goal is almost achieved at the candidate point; while a value close to 1 means that the design goal is hardly achieved. To use Equation (6.11) in SEED is like "pulling" data points to regions with both large prediction errors (effect of α_i) and response values that are close to the design goal (effect of η_i).

Another way is to formulate the adjusted covariance as:

$$\sigma_{ij}^{adj} = \frac{1}{4}\left(\alpha_i + \eta_i\right)\left(\alpha_j + \eta_j\right)\sigma^2 R\left(\left|x_i - x_j\right|\right) \tag{6.12}$$

In Equation (6.12) the two coefficients, α_i and η_i, are added instead of being multiplied as in Equation (6.11). Since α_i and η_i both have values between 0 and 1, their sum is

between 0 and 2; thus a coefficient of 1/4 is added to ensure that the whole coefficient part has values between 0 and 1. When Equation (6.11) is used the designer expects to add new points that "*either* have large prediction errors *or* achieve design goals" because the covariance will be greatly adjusted when either criterion is satisfied. When Equation (6.12) is used the designer expects to add new points that "*both* have large prediction errors *and* achieve design goals" because the covariance will be greatly adjusted only when both criteria is satisfied.

The third way is to modify the coefficient α_i to reflect the effects from both prediction errors and design goals. Note that in Equation (4.28), α_i is calculated with the following equation:

$$\alpha_i = 1 - relative.uncert = 1 - |\frac{e_i}{\lambda e_{\max}}| \tag{6.13}$$

We may change Equation (6.13) to the following one:

$$\alpha_i = 1 - relative.uncert - goal.achievement \tag{6.14}$$

Or,

$$\alpha_i = 1 - relative.uncert \times goal.achievement \tag{6.15}$$

In Equations (6.14) and (6.15) the term *relative.uncert* represents effect from prediction errors, and *goal.achievement* represents effect from design goals.

The methods talked above are those in which the three criteria, "locating points in regions with large prediction errors", "having points spread over the design space", and "locating points in regions where design goals are (almost) achieved", are considered in one formulation of adjusted covariance; and in SEED, the tradeoffs among these three

criteria are done in one step. Another possible method is to do the tradeoffs in separate steps, e.g., first we add in new points to minimize the prediction error, and then we add in new points that achieve design goals better. In this method, the covariance will be adjusted twice in a single iteration, following the equations below:

$$\sigma_{ij}^{adj1} = \alpha_i \alpha_j \sigma_{ij} = \alpha_i \alpha_j \sigma^2 R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \tag{6.16}$$

$$\sigma_{ij}^{adj2} = \eta_i \eta_j \sigma_{ij} = \eta_i \eta_j \sigma^2 R\left(\left|\boldsymbol{x}_i - \boldsymbol{x}_j\right|\right) \tag{6.17}$$

Due to the space and time limit, only one of the above ideas will be studied and used in this dissertation. The method associated with Equations (4.28) and (6.11) will be studied here because it is the simplest formulation. To study all formulations mentioned above and compare their performance is future work to research in this dissertation.

When formulating the coefficient α_i in Equation (6.11) and Equation (6.13), the term *relative.uncert* is calculated using the same method as in Chapter 4:

$$relative.uncert = \left|\frac{e_i}{\lambda e_{\max}}\right| \tag{6.18}$$

When design goals are not considered in metamodeling processes (as in Chapter 4 and Chapter 5), usually we take $\lambda = 2$, thus *relative.uncert* has values between 0 and 0.5. In a similar way, the coefficient η_i is formulated as:

$$\eta_i = 1 - goal.achievement \tag{6.19}$$

To formulate the term *goal.achievement*, we need to satisfy the following requirements:

- *goal.achievement* should have values between 0 and 1.

- *goal.achievement* should be a increasing function of degrees of achievement of design goals, i.e., large values should be assigned to *goal.achievement* at points where design goals are almost achieved.
- In the process of designing sequential experiments, since the information from current metamodels of response values and prediction errors is usually inaccurate, we should balance between "locating points in regions with large prediction errors", "having points spread over the design space", and "locating points in regions where design goals are (almost) achieved". As discussed in Chapter 4, the balance between the first two aims is controlled by the factor λ (see Equation (4.25) or Equation (6.18)). After taking design goals into consideration, new data points may not be those with largest predicted prediction errors with current metamodels or those have long distance from current data points; more points will also be added in regions where design goals are expected to be achieved. More trade-off is needed. This is like "twisting" the data points with three forces, one pulling points to regions with large predicted prediction errors, another to regions far from current data points, and the third to regions where design goals are expected to be achieved. Based on the discussions above, in practice it may be better not to define *goal.achievement* between 0 and 1. Points that almost achieve design goals should not have *goal.achievement* close to 1; otherwise the trade-off will be damaged. As design evolves, more points are observed and more

accurate metamodels are developed; designers intend to decrease the value of γ to focus more on "achieving design goals" in the exploration of new points.

- As will be shown later, a factor γ is introduced (together with λ which is introduced in Chapter 4) to balance the weight of consideration of "prediction errors", "space-filling", and "design goals" in the identification of new data points. For example, in practice, we may use $goal.achievement \in \left[\frac{1}{\gamma}, 1\right]$.

In this dissertation, to calculate *goal.achievement*, we follow formulations of nonlinear design goals in the compromise DSP (Mistree, et al., 1993b). There may be other ways to formulate *goal.achievement*; studies and comparisons on those possible formulations will be future work to this research. In the compromise DSP, objective functions are normalized using a target value for each goal and the deviation from this target value is used to formulate the deviation function. There are two deviation variables, d^- and d^+, for each goal that measure the deviation from the target value. Both deviation variables take on only non-negative values. Nonlinear design goals are formulated as:

$$A_i(x) + d_i^- - d_i^+ = 0 \tag{6.20}$$

where, $d_i^-, d_i^+ \geq 0$ and $d_i^- \cdot d_i^+ = 0$. In Equation (6.20) the target value is absorbed into the definition of the function $A_i(\boldsymbol{x})$.

When the goal is to minimize a response y, first we choose a low target value, T_L, for the response based on experience. Then Equation (6.20) can be formulated as:

$$1-\frac{T_L}{y(x)}+d_i^- - d_i^+ = 0 \text{ , or, } 1-\frac{y_{\max}-y(x)}{y_{\max}-\max\left(y_{\min},T_L\right)}+d_i^- - d_i^+ = 0 \qquad (6.21)$$

In this case, the deviation variable d_i^+ needs to be minimized to achieve minimum values for y.

When the goal is to maximize a response y, first we choose a high target value, T_H, for the response based on experience. Then Equation (6.20) can be formulated as:

$$\frac{y(x)}{T_H}-1+d_i^- - d_i^+ = 0 \text{ , or, } \frac{y(x)-y_{\min}}{\min\left(T_H,y_{\max}\right)-y_{\min}}-1+d_i^- - d_i^+ = 0 \qquad (6.22)$$

In this case, the deviation variable d_i^- needs to be minimized to achieve maximum values for y.

When the goal is to make a response y as close as possible to a preset value, T_S, Equation (6.20) can be formulated as:

$$\frac{y(x)}{T_S}-1+d_i^- - d_i^+ = 0 \qquad (6.23)$$

We see that the target value for the response is achieved exactly when both deviation variables are equal to zero. Therefore, in this case we seek to minimize both d_i^- and d_i^+.

Based on the formulations of nonlinear design goals in compromise DSP as presented above, we formulate *goal.achievement* as below:

$$goal.achievement = \begin{cases} 0 & y_{\max} \le y(x) \\ \frac{1}{\gamma}\cdot\frac{y_{\max}-y(x)}{y_{\max}-\max\left(y_{\min},T_L\right)} & \max\left(T_L,y_{\min}\right)_L < y(x) < y_{\max} \\ \frac{1}{\gamma} & y(x) \le \max\left(T_L,y_{\min}\right) \end{cases} \qquad (6.24)$$

$$goal.achievement = \begin{cases} 0 & y(x) \le y_{\min} \\ \frac{1}{\gamma} \cdot \frac{y(x) - y_{\min}}{\min\left(T_H, y_{\max}\right) - y_{\min}} & y_{\min} < y(x) < \min\left(T_H, y_{\max}\right) \\ \frac{1}{\gamma} & \min\left(T_H, y_{\max}\right) \le y(x) \end{cases} \quad (6.25)$$

$$goal.achievement = \begin{cases} 0 & y(x) \le y_{\min} \\ \frac{1}{\gamma} \cdot \frac{y(x) - y_{\min}}{T_S - y_{\min}} & y_{\min} < y(x) < T_S \\ \frac{1}{\gamma} & y(x) = T_S \\ \frac{1}{\gamma} \cdot \frac{y_{\max} - y(x)}{y_{\max} - y_{\min}} & T_S < y(x) < y_{\max} \\ 0 & y_{\max} \le y(x) \end{cases} \quad (6.26)$$

Equation (6.24) is used when the design goal is to minimize the response. Equation (6.25) is used when the design goal is to maximize the response. Equation (6.26) is used when the design goal is to make the response as close to a preset value as possible. In Equations (6.24), (6.25), and (6.26), y_{max} and y_{min} are the maximum and minimum response values at all observed points, respectively. We can use the maximum and minimum observed response values for y_{max} and y_{min}; sometimes we use values slightly different from observed values to ensure that the *goal.achievement* is appropriately weighted in the adjustment of covariance matrices. The expression $\max\left(T_L, y_{\min}\right)$ is used in Equation (6.24) is to make sure that the adjustment due to achievement of design goals does not fade even when the design target value is not achieved. The term $\min\left(T_H, y_{\max}\right)$ is used in Equation (6.24) because of the same reason. T_L, T_H, and T_S have the same

meanings as in Equations (6.21), (6.22), and (6.23); they are target goal values selected by designers based on experience. As stated before, a coefficient, γ, is used to help balance the tradeoffs among three criteria in identifying new points. It should be noted that the response values (y_{max}, y_{min}, T_L, T_H, and $y(x)$) in Equations (6.24), (6.25), and (6.26) should satisfy $y_{max} > T_L$ and $T_H > y_{min}$; $y(x)$ typically has values larger than T_L or smaller than T_H. In cases where this requirement is not met, designers should make corresponding modifications; usually it is because of inappropriate problem initialization.

Equations (4.28), (6.11), (6.13), (6.19), (6.24), (6.25), and (6.26) will be used in the SEED processes. From Equations (6.13) and (6.19) we see that $\alpha_i \in \left[1-\frac{1}{\lambda},1\right]$, and $\eta_i \in \left[1-\frac{1}{\gamma},1\right]$. When λ and γ are both given values of 2, the adjustment at point x_i, $\alpha_i \eta_i$, is in [0.25, 1]. When λ and γ are both given values of 1.5, $\alpha_i \eta_i$ is in [0.1111, 1].

In this section, we discussed how to take design goals into consideration in the metamodeling processes. Several possible ways are proposed; we focus on one of them and developed detailed mathematical formulations for SEED applications. It should be noted that these mathematical formulations are not necessarily perfect; future research may be needed to study various possible formulations and identify the best or theoretically sound one. Test of the proposed formulations will be done in Section 6.5, as part of the validation for the Efficient Robust Concept Exploration Method (E-RCEM).

By considering design goals in metamodeling processes, we are able to facilitate the information feedback from the process of design space exploration to the process of

metamodeling. It is expected that such information feedback will help designers locate new points in more "critical" regions, i.e., regions where *either* (*both*) prediction errors are large *or* (*and*) design goals are almost achieved. Based on the SEED method in Chapters 4 and 5, experimental designs with constrained design spaces in Section 6.2, and experimental designs with design goals in this section, we develop the Efficient Robust Concept Exploration Method (E-RCEM) as will be discussed in detail in Section 6.4.

6.4 THE EFFICIENT ROBUST CONCEPT EXPLORATION METHOD

The Efficient Robust Concept Exploration Method (E-RCEM) is presented in this section. E-RCEM is developed to integrate the two traditionally separated processes in simulation-approximation-based design, i.e., the process of metamodeling and that of design space exploration. It is expected that this integration will help achieve better design solutions with less time and money spent on expensive experiments and optimization processes.

As discussed in Section 6.1, in traditional early-stage design processes, the information flow is one-way from metamodeling to design space exploration. The two processes are not integrated and have different goals. The purpose of the metamodeling process is to develop accurate metamodels, and that of the design space exploration process is to obtain a satisficing (Mistree, et al., 1993b) design solution based on current metamodels. To achieve a good design solution in the design space exploration process, it is very important to have an accurate metamodel; while in the metamodeling process, it

is very hard to tell how accurate the metamodel should be in order to achieve good design solutions, given that there is no information feedback from the design space exploration process to the metamodeling process in traditional design methods. This conflict leads to different strategies and behaviors in the two processes.

As what we do with SEED, from the viewpoint of metamodeling, designers should make more observations in regions where prediction errors are large. While from the viewpoint of design space exploration, designers should make more observations where design solutions probably lie (i.e., design goals are achieved while design constraints are satisfied) given that the metamodel is accurate. As a result, a lot of time and money is wasted in the metamodeling process in "unimportant regions" (in infeasible regions, or where design goals are hardly achieved) to help achieve more accurate metamodels.

On the other hand, inaccurate metamodels may be misleading in the design space exploration process. Thus designers need to balance between "increasing metamodel accuracy" and "exploring in most-likely-to-succeed regions". This can only be achieved when the two processes, metamodeling and design space exploration, are integrated. In other words, the information feedback flow from design space exploration to metamodeling must be built. This idea of "*metamodeling **for** design space exploration*" leads to the Efficient Robust Concept Exploration Method.

The Efficient Robust Concept Exploration Method (E-RCEM) is developed based on the Robust Concept Exploration Method (RCEM), incorporating several new methods and tools, e.g., the SEED method, metamodeling with irregular design spaces,

metamodeling with consideration of design goals, etc. The infrastructure of E-RCEM is illustrated in Figure 6.11 and Figure 6.12. Comparing Figure 6.11 with Figure 1.8 (the infrastructure for RCEM), we see that E-RCEM inherits RCEM's design process organization. Both RCEM and E-RCEM consists three main phases: Problem Initialization, Metamodeling, and Design Space Exploration; this organization of design process is well illustrated in Figure 6.1. In RCEM, Processor A (Step 1) corresponds to the phase of Problem Initialization; Processors B, C, D, and E (Steps 2 and 3) correspond to the Metamodeling phase; and Processor F (Step 4) corresponds to the phase of Design Space Exploration. In E-RCEM, Processors A and B correspond to the phase of Problem Initialization; the loop with Processors C, D, E, F, and G correspond to the phase of Metamodeling; and the loop with Processors C, D, E, F, G, and H correspond to the phase of Design Space Exploration. The inheritance from RCEM to E-RCEM is apparent.

In Figure 6.12 we see that the phases of metamodeling and design space exploration are not strictly separated. There is a flow back from the compromise DSP to the beginning of the phase of metamodeling. Thus from a viewpoint at a higher level, these two phases are "integrated" in E-RCEM; or in other words, we can say, we are doing "metamodeling ***for*** design space exploration" in E-RCEM.

The implementation of the three phases in E-RCEM is discussed in details in Sections 6.4.1 through 6.4.3. In E-RCEM, we can follow different types of design processes, in which the three phases are organized in different ways. These processes are discussed in Section 6.4.4, while our focus is on the description of the ***integrated design***

process in which the metamodeling process and design space exploration process are integrated.

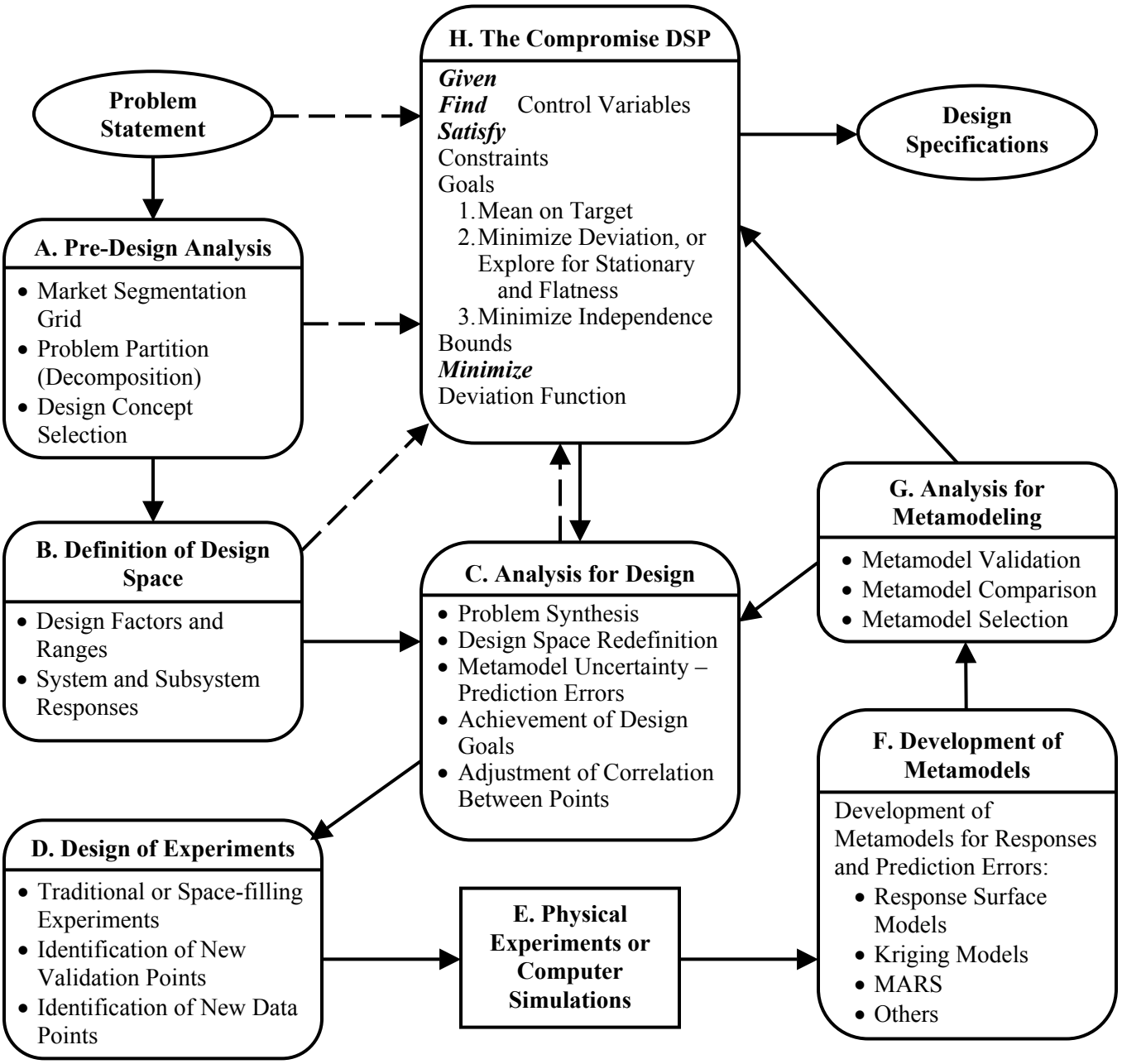


Figure 6.11 Infrastructure of the Efficient Robust Concept Exploration Method (I)

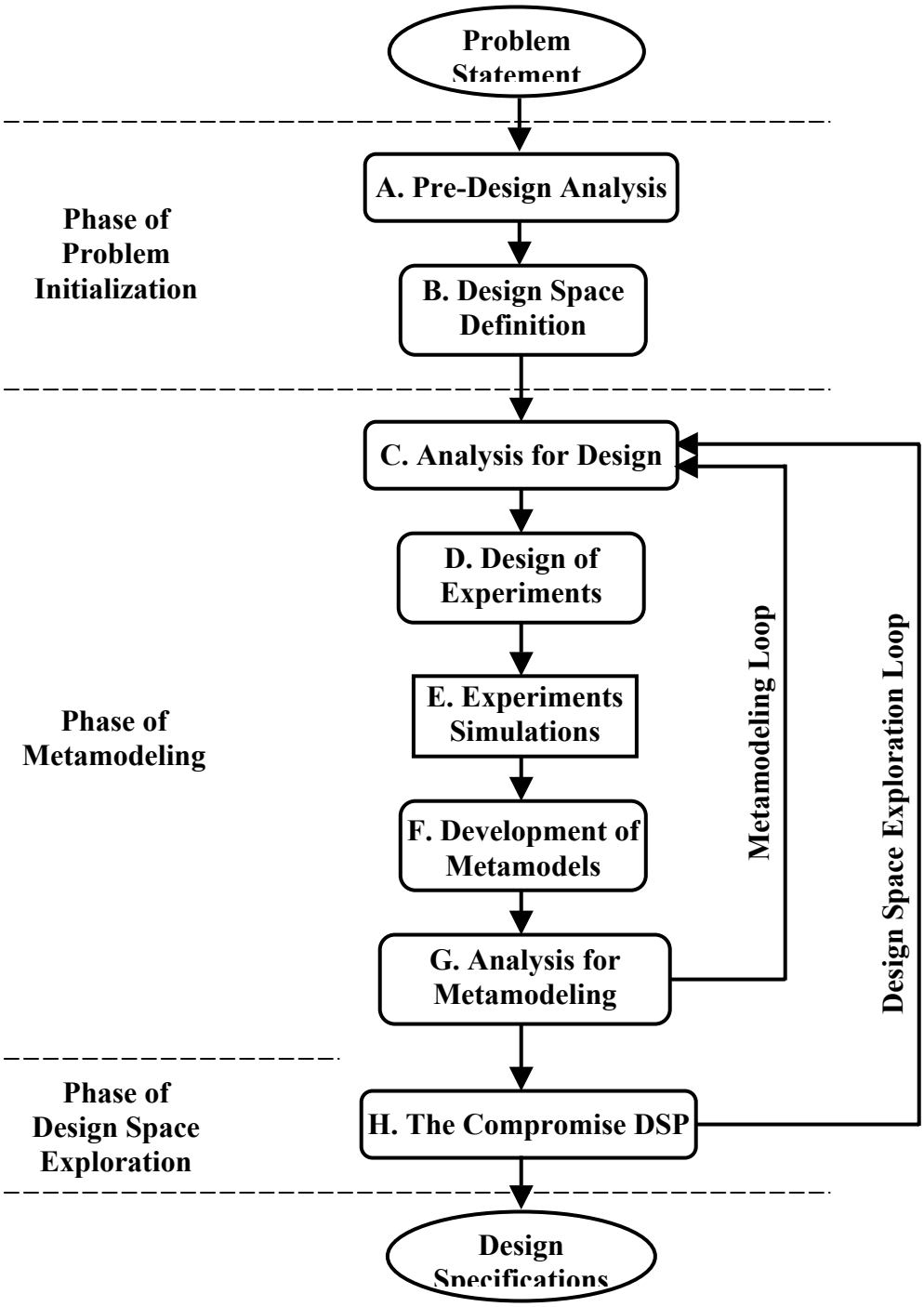


Figure 6.12 Infrastructure of the Efficient Robust Concept Exploration Method (II)

6.4.1 The Phase of Problem Initialization

As shown in Figure 6.13, the phase of Problem Initialization in E-RCEM consists of two processors or steps, A: Pre-Design Analysis and B: Design Space Definition.

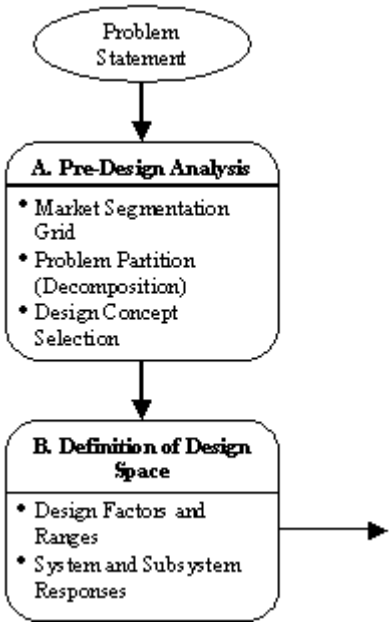


Figure 6.13 Phase I – Problem Initialization

Activities conducted in Processor A are listed in Figure 6.11. The market segmentation grid (Meyer, 1997) is drawn to facilitate identifying leveraging strategies for a product platform. This is inherited from the Product Platform Concept Exploration Method (PPCEM, see Simpson, 1998); this analysis is useful when we are designing product families. Problem partition or decomposition, together with problem synthesis in Processor C, was studied in (Koch, 1998); this analysis is needed when we are dealing with a complex engineering system with coupling subsystems. Design concept selection is necessary when we have several optional design concepts; a most-likely-to-succeed design concept can be selected and further studied in E-RCEM by formulating and

solving a selection DSP (Mistree, et al., 1994). In applications, not all mentioned analyses should be conducted; the implementation varies from case to case.

The design variables and responses are clarified in Processor B. Factors are classified in the following manner. Appropriate ranges for the control and noise factors are identified during this step, and constraints and goal targets for the responses are also identified.

- **Responses** are performance parameters of the system; in the problem formulation, they may be constraints or goals or both and are identified from the overall design requirements and the market segmentation grid.
- **Control factors** are variables which can be freely specified by a designer; settings of the control factors are chosen to minimize the effects of variations in the system while achieving desired performance targets and meeting the necessary constraints. Signal factors are also lumped within control factors since it is often difficult to know, *a priori*, which design variables are control factors and can be used to minimize the sensitivity of the design to noise variations and those which are signal factors and have no influence on the robustness of the system.
- **Noise factors** are parameters over which a designer has no control or which are too difficult or expensive to control.
- **Scale factor** is a factor around which a product platform is leveraged either through vertical scaling, horizontal scaling, or a combination of the two. This is inherited from the PPCEM (Simpson, 1998).

6.4.2 The Phase of Metamodeling

The whole process of sequential metamodeling as discussed in Chapter 5 (Figure 5.40 and Figure 5.41) is applied in this phase. As shown in Figure 6.14, processors involved in this phase are Processor C, D, E, F, and G. The purpose of this phase is to develop acceptable metamodels for the next phase, design space exploration.

The initial design space is redefined in Processor C. In the metamodeling phase, this redefinition of design spaces is done by elimination of unimportant factors, as illustrated in Figure 5.41. The loop of C-D-E-F-G corresponds to the SEED processes as studied and applied in Chapters 4 and 5.

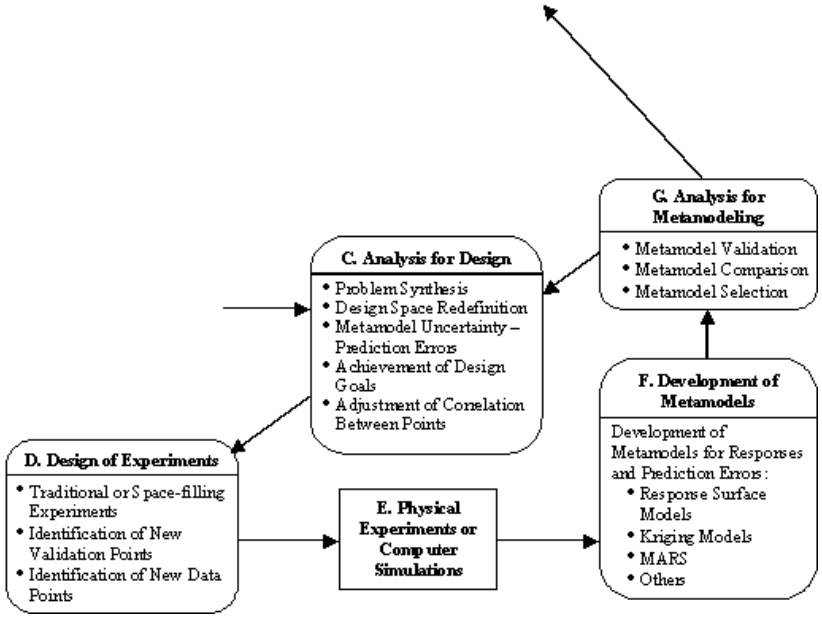


Figure 6.14 Phase II – Sequential Metamodeling

6.4.3 The Phase of Design Space Exploration

As shown in Figure 6.14, processors involved in this phase are Processor C, D, E, F, G, and H. The purpose of this phase is to explore for robust design solutions with acceptable metamodels.

In this phase, the redefinition of design space in Processor C corresponds to our discussions in Sections 6.2, generation of irregular design spaces due to constraints on design variables and system responses. Following the method developed in Section 6.3, design goals are considered in analyses in Processor C. Note that the achievement of design goals is calculated with Equations (4.28), (6.11), (6.13), (6.19), (6.24), (6.25), and (6.26), which come from Processor H, the compromise DSP.

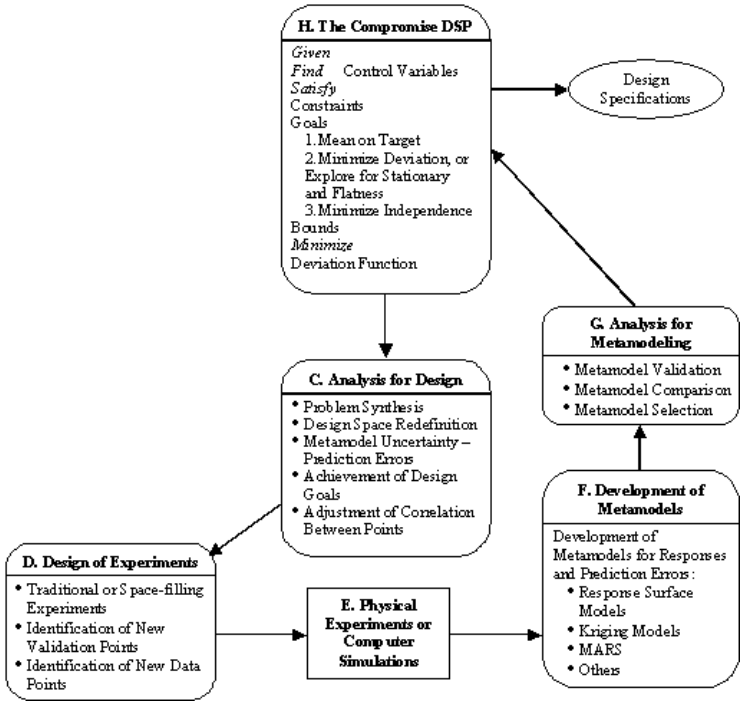


Figure 6.15 The Phase of Design Space Exploration (Integrated Processes of Metamodel and Design Space Exploration)

6.4.4 Different Design Processes in E-RCEM

In E-RCEM, after initialization of the design problem, we can go through the design processes in three ways following designers' different decisions:

1. ***Traditional Process***: develop accurate metamodels then explore for design solutions in the compromise DSP without updating the metamodels. This is a one-way process; thus there is no information feedback from Processor H to C in Figure 6.11 (the design space exploration loop in Figure 6.12 is then removed). Our studies and application in Chapters 4 and 5 follow this way.
2. ***Integrated Design Process***: skip the phase of metamodeling and enter the integrated design processes of metamodeling and design space exploration as illustrated in Figure 6.15. In this case the SEED process is not conducted thus the flow from Processor G to Processor C is removed. This corresponds to the removal of the metamodeling loop in Figure 6.12. This method is usually used with simple problems in which the actual responses are not highly nonlinear or irregular.
3. ***Hybrid Process***: develop metamodels first, and then explore the design space for solutions as well as updating metamodels. The SEED method is applied to ensure acceptable metamodels are obtained. Then the design space exploration loop (in Figure 6.12) is adopted to help update the metamodel and obtain better design solutions. This method is usually used with large-scale problems with highly nonlinear or irregular responses.

In this chapter, since we use simple examples to illustrate the integrated design process in E-RCEM, we will follow the 2nd way as mentioned above. The steps of this integrated design process are explained below:

Step 1 – Problem Initialization. This is the first phase in design, and corresponds to Processors A and B in Figure 6.11 and Figure 6.12.

Step 2 – Initial Experiments and Design Space Reduction. This corresponds to Processor C in Figure 6.11 and Figure 6.12, or the process of elimination of unimportant design factors in Figure 5.41. In this step we use classical experiments and the response surface metamodels to identify important design variables.

Step 3 – Design Space Redefinition. This corresponds to Processor C in Figure 6.11 and Figure 6.12. In this step we identify the feasible design space (usually irregular) due to the constraints on design variables and responses.

Step 4 – Identification of New Validation Points. This corresponds to Processors H, C, and D in Figure 6.11 and Figure 6.12. In this step we identify new validation points using the similar method in SEED; the only difference is that in identifying the new points design goals are considered as well as prediction errors and distances from existing points. After locating new validation points, metamodels of prediction errors are developed and the achievement of design goals at points in the feasible design space are calculated to facilitate the identification of new data points in the next step.

Step 5 – Identification of New Data Points. The covariance matrix is adjusted with information from the metamodels of prediction errors and the achievement of design

goals as obtained in Step 4. New data points are identified by maximizing the determinant of this adjusted covariance matrix. This corresponds to Processor D in Figure 6.11 and Figure 6.12.

Step 6 – Updated Metamodels and Metamodel Selection. This corresponds to Processors E, F, and G in Figure 6.11 and Figure 6.12. New metamodels are developed and the best metamodels are selected in future iterations and steps (as what we did in Chapter 5 with kriging and MARS metamodels).

Step 7 – Analysis of Design. This corresponds to Processors H and C in Figure 6.11 and Figure 6.12. In this step, we either compare the achievement of design goals at new identified points with that at old points (when designers wish to enter another design space exploration iteration), or formulate and solve the compromise DSP for design solutions (when designers wish to finish the design space exploration process). The E-RCEM processes will stop in this step once the stopping criterion is met; otherwise another iteration will start at Step 4. Besides the stopping criteria introduced in Chapter 4, we can also stop when the improvement of achievement of design goals is smaller than some preset value; when this criterion is adopted, the design space exploration process in E-RCEM becomes similar to the EGO, and can be viewed as an optimization algorithm.

In this section we developed the E-RCEM based on RCEM and our studies in Chapter 4, Chapter 5, and Sections 6.1 – 6.3. In order to develop more accurate metamodels with less time and money spent on expensive experiments, sequential metamodeling and the SEED processes are used in E-RCEM to replace the metamodeling

phase in RCEM. Design goals are considered in the metamodeling process, which is expected to help achieving better design solutions with fewer experiments. The actual implementation of each step in the integrated design process is liable to vary from problem to problem. This integration of metamodeling and design space exploration, in which both computer simulations (or physical experiments) and empirical metamodels are used in achieving design solutions, can also be viewed as a new optimization algorithm that is best used in cases with expensive experiments. The integrated design process in E-RCEM will be illustrated with a single-variable function in the next section.

6.5 APPLICATION OF THE E-RCEM METHOD: A SINGLE-VARIABLE EXAMPLE

In this section, we apply the E-RCEM method in the single-variable example similar to that we studied in Chapter 4. The single-variable function is:

$$f(x)=\begin{cases}\dfrac{Sin\left(10\pi\left(x+0.01\right)\right)}{x+0.5} & 0\le x\le 0.19\\ 0 & 0.19<x\le 1\end{cases} \tag{6.27}$$

This function is the same as Equation (4.35) by adding a constant 2 to the response. A graph of this function is shown in Figure 6.16. As we see from the equation and graph, the design space is x = [0, 1]. In this design space, the maximum response value is y = 1.852 at x = 0.04, and the minimum response value is around y = -1.564 at around x = 0.138; the response range is 3.415.

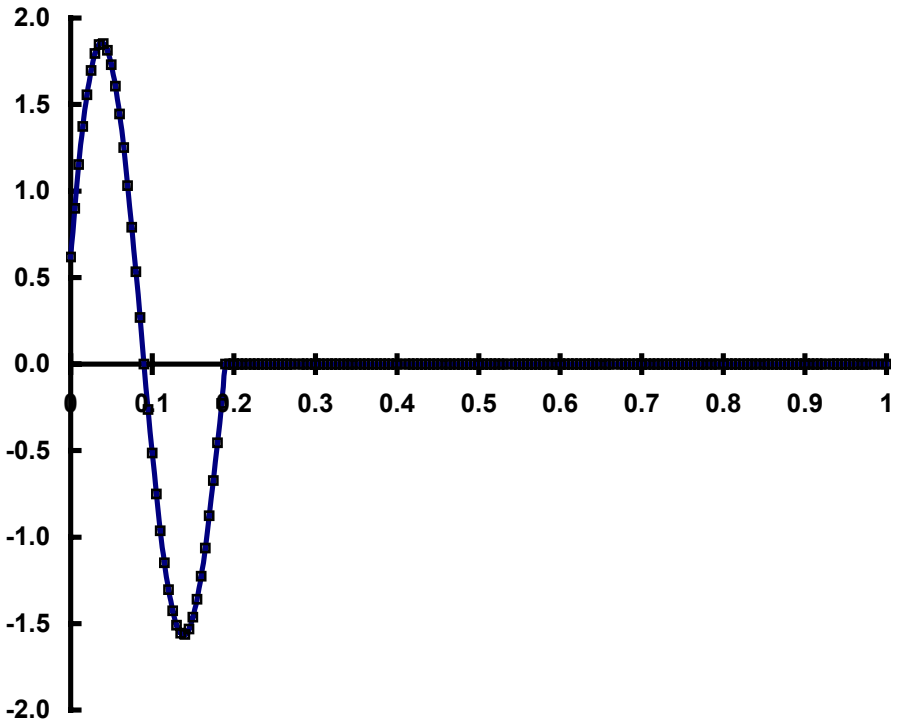


Figure 6.16 A Single-Variable Function

The design goal in this example is to minimize the response $f(x)$. The design target is preset at $T_L = -1.6$, which is unachievable (smaller than the minimum actual response value); thus this design task is actually an optimization problem. There is no constraint put on the design variable or the response. In Chapter 4 following the SEED processes we are able to develop an acceptable metamodel with 11 observed points, which is much more accurate than those developed with single-stage experimental design methods. In this section, the design solution obtained with E-RCEM will be compared to that obtained with SEED. Initially we will have 3 data points and 4 validation points; 4 more points will be added in 3 iterations of the integrated process of metamodeling and

design space exploration. The value of λ in Equations (6.13) and (6.18) is set to be 2.0 throughout the whole design process to balance "space-filling" and "reducing prediction errors" in the exploration for new points. The value of γ will gradually decrease from 2.0 to 1.25 along the timeline. As discussed in Section 6.3, when more accurate metamodels are obtained with more observed points in later iterations of the integrated process of metamodeling and design space exploration, more weight should be put on the "achievement of design goals", instead of "space-filling" or "reducing prediction errors", in the exploration of new points. A small value of γ helps achieve this balance.

Iteration I – Step 1: Problem Initialization. This is done.

Iteration I – Step 2: Initial Experiments and Design Space Reduction. Since there is only one design variable in this example, we do not reduce the design space by screening out unimportant design factors. The initial experiments are the same as that in Chapter 4; the three data points are listed in Table 6.8. The corresponding kriging metamodel is illustrated in Figure 6.17; the value of θ for this kriging metamodel is 98.71232.

Table 6.8 Initial Experiments

x	0.0	0.5	1.0
y	0.618	0.0	0.0

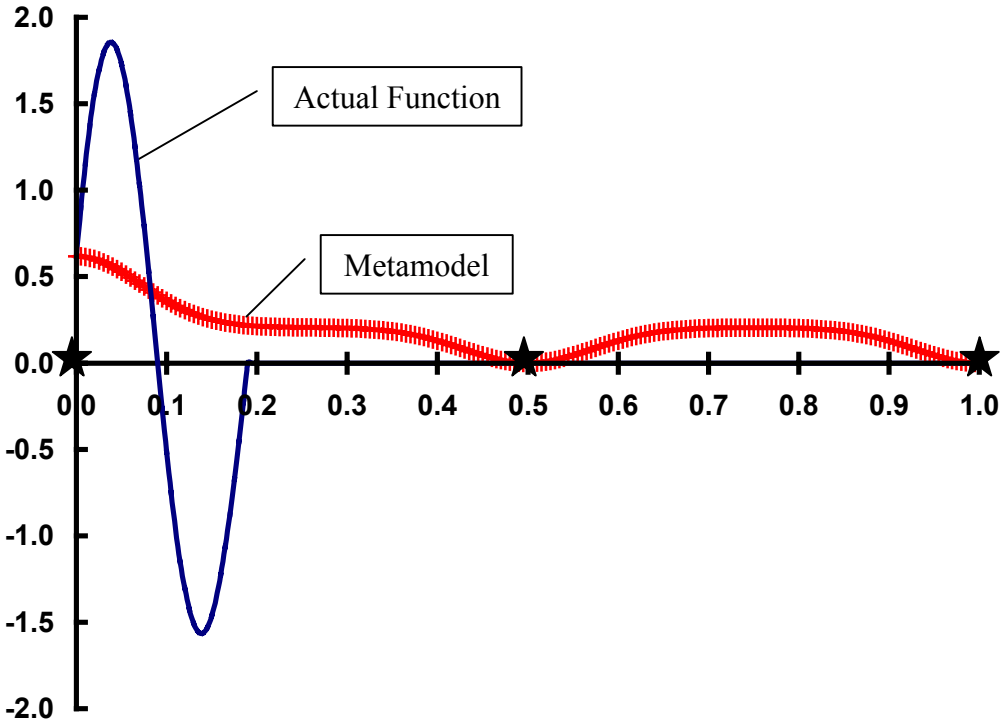


Figure 6.17 Initial Metamodel with 3 Data Points

Iteration I – Step 3: Design Space Redefinition. Since there is no constraint on design variables or responses, the design space is not redefined in this step.

Iteration I – Step 4: Identification of New Validation Points. Four new validation points are identified to be as far from current observed points as possible. These points are listed in Table 6.9. Predicted prediction errors are unavailable thus not considered in this process. The design goal is not considered in identifying new validation points in this step because we do not think the initial metamodel with only 3 data points is accurate enough. As design evolves and more points are observed we will

take the design goal into consideration. This prevents us from being misled to incorrect directions by inaccurate metamodels in very early stages of design.

A kriging metamodel of prediction errors is developed based on the information in Table 6.9. Note that prediction errors at 3 data points are zero. The value of θ is 99.99880. The maximum absolute predicted prediction error, $e_{max} \approx 1.3$, is found through optimization. The predicted prediction error at a candidate point, e_i, will be calculated with the kriging metamodel of prediction errors and used in the formulation of α_i in Equation (6.13). This information is then further used in the adjustment of entries in the covariance matrix in sequential experimental designs.

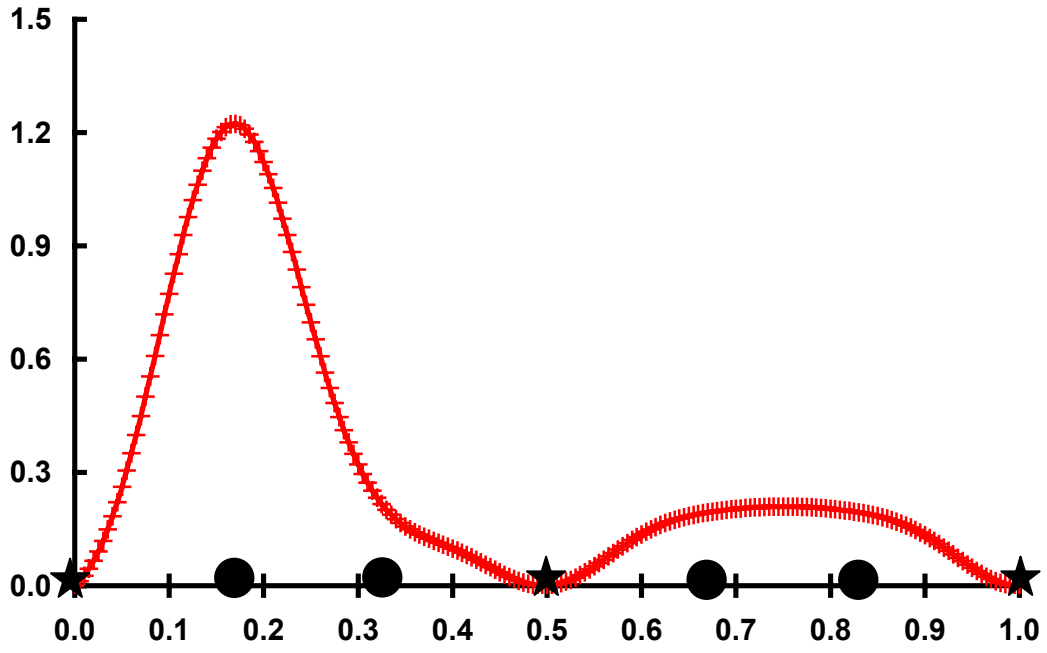


Figure 6.18 Metamodel of Prediction Errors Calculated in Iteration I – Step 4

Table 6.9 Validation Points in the 1st Iteration

x	0.167	0.333	0.667	0.833
y_{pred}	0.232	0.193	0.193	0.193
y_{actual}	-0.991	0.0	0.0	0.0
y_{error}	1.223	0.193	0.193	0.193

Another task in this step is to calculate the *goal.achievement* at candidate points. Since in this problem we want to minimize the response, Equation (6.24) will be used to formulate *goal.achievement*. As discussed in Section 6.3, when using Equation (6.24), we may or may not force *goal.achievement* to be 0 when the predicted response $y(x)$ is larger than or equal to y_{max}. In this case, we force *goal.achievement* to be 0 at points with large predicted response values.

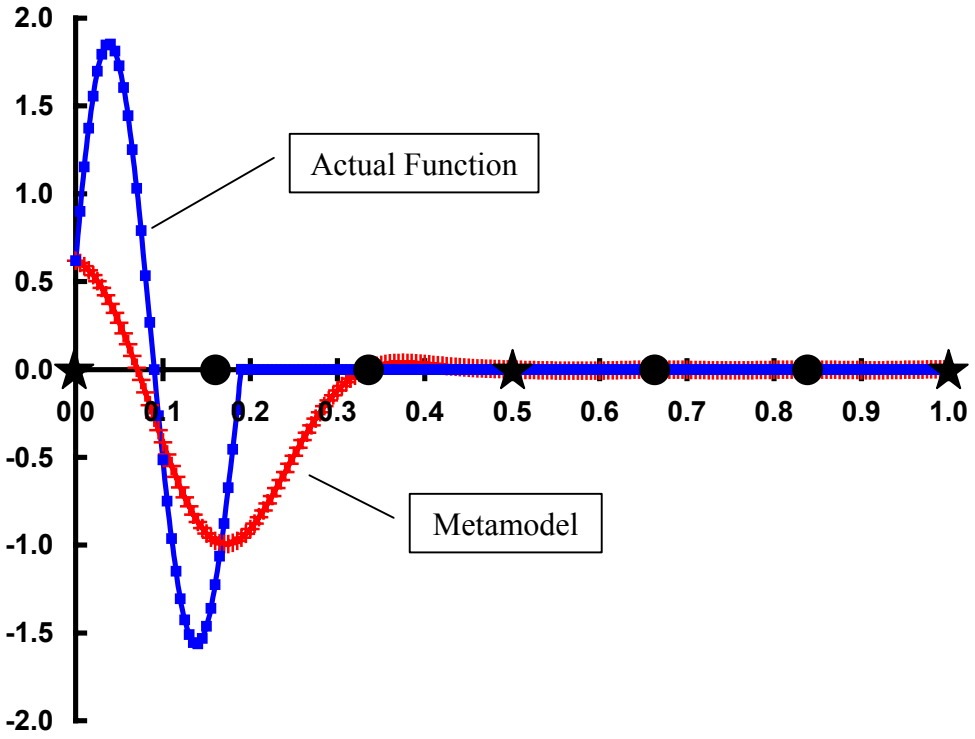


Figure 6.19 Metamodel Developed with 7 Observed Points in Iteration I – Step 4

The value of y_{max} is 0.618 (when $x = 0.0$) and the value of y_{min} is –0.991 (when $x = 0.167$); in this step we use the actual observed response values for y_{max} and y_{min}. As mentioned before, the design target value is set at $T_L = -1.6$. The metamodel used in Equation (6.24), $y(x)$, is developed with all observed points in the feasible design space. Since at the end of the design process we will use all observed points to develop a final metamodel and explore the "final" design solution, it is reasonable to calculate the achievement of design goals based on information from all observed points in intermediate iterations. To predict prediction errors we will have to use two groups of points; however, to calculate the achievement of design goals, we should utilize information from as many points as possible to try not be misled to wrong directions. The kriging metamodel developed with 7 observed points is illustrated in Figure 6.19. The value of θ for this kriging metamodel is 99.99983.

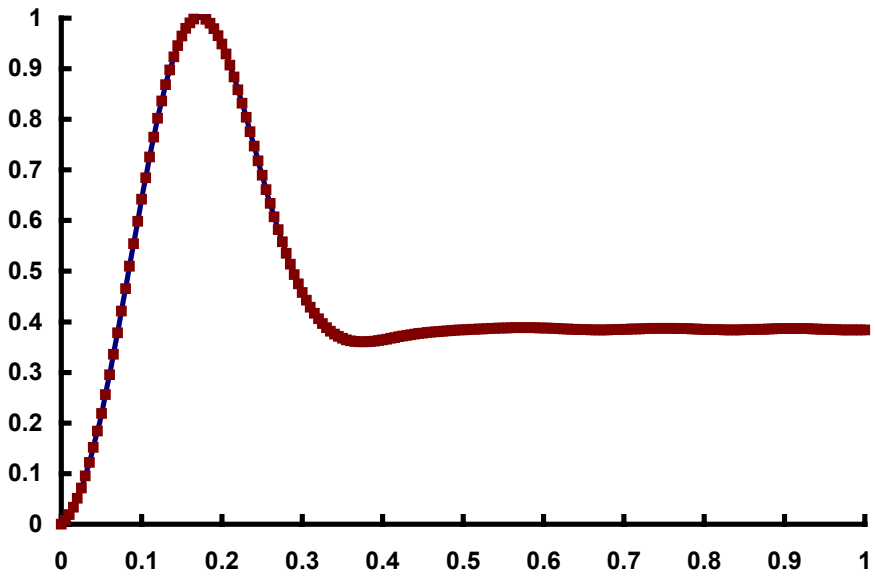


Figure 6.20 Values of *goal.achievement* at Points in the Design Space

A plot of *goal.achievement* at points in the design space is shown in Figure 6.20. In this figure we set $\gamma = 1$; when adjusting the covariance matrix in the next step, we will not set $\gamma = 1$ (we use large values for γ, e.g., $\gamma = 1.5$ or 2), as explained in Section 6.3. In Figure 6.20 we see that when x is around 0.2 we have higher values of *goal.achievement*, which means we are close to achieve the design goal.

Iteration I – Steps 5 and 6: Identification of New Data Points and Updated Metamodels. In this step we need to identify 1 new data point. A 4×4 covariance matrix is developed with the first 3 rows and columns corresponding to the 3 data points, and the last row and column corresponding to the new data point. Entries of this covariance matrix are then adjusted with information from the prediction errors and achievement of design goals. This adjustment is done with Equations (4.28), (6.11), (6.13), (6.19), and (6.24). The values of λ and γ are set as $\lambda = \gamma = 2$. Values of $\alpha_i \gamma_i$ (the amount of adjustment at point x_i) in the design space are illustrated in Figure 6.21. Then the determinant of this adjusted covariance matrix is calculated. The new data point is the one that generates the adjusted covariance matrix with the largest determinant. FORTRAN programs are written to facilitate the formulation and adjustment of the covariance matrices and calculation of determinants of matrices.

The software iSIGHT is used to link the programs (formulation of covariance matrices, calculation of prediction errors, calculation of degrees of achievement of design goals, the adjustment of covariance matrices, and calculation of the determinant). Various optimization techniques (as what we did in Chapter 4) built in iSIGHT are utilized to find out the point with the maximum determinant of the corresponding

adjusted covariance matrix. The possible new data point is identified at $x = 0.177$. Since this point is very close to one of the validation points, $x = 0.167$, we decide to use $x = 0.167$ as the new data point. The four data points are listed in Table 6.10. A new metamodel is developed with 4 data points and illustrated in Figure 6.22. The value of θ for this kriging metamodel is 99.99964. We do not make metamodel comparison and selection in this example because only kriging metamodels are developed.

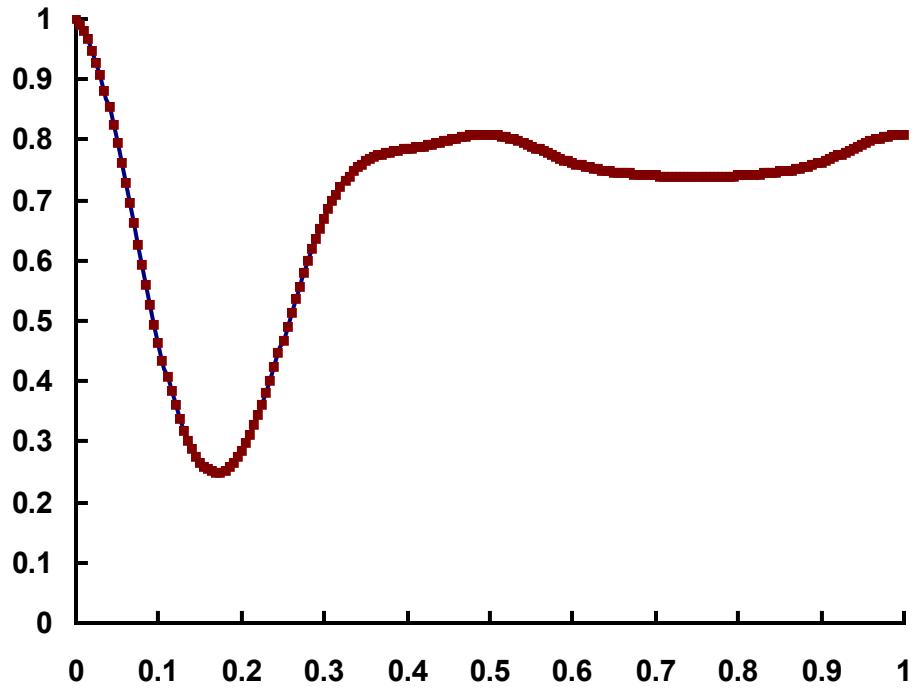


Figure 6.21 Values of $\alpha_i\gamma_i$ at Candidate Points in the Design Space in Iteration I – Step 5

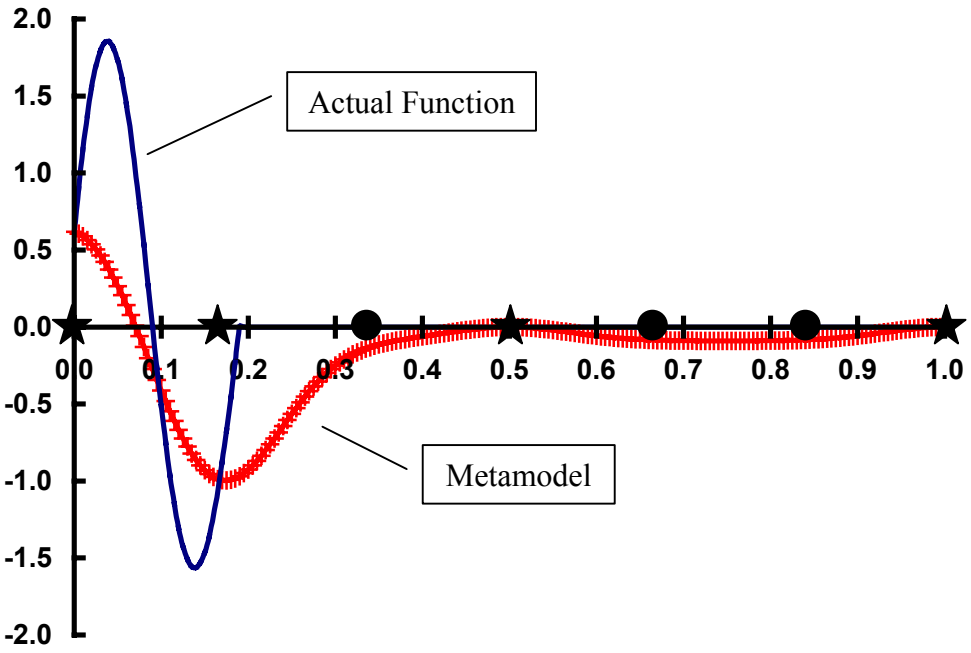


Figure 6.22 Kriging Metamodel of Responses Developed with 4 Data Points

Table 6.10 Four Data Points

x	0.0	0.167	0.5	1.0
y	0.618	-0.991	0.0	0.0

Iteration I – Step 7: Analysis of Design. Since the stopping criterion is not met, we will go to the next iteration of integrated metamodeling and design space exploration process.

Iteration II – Step 4: Identification of New Validation Points. In this step we plan to add in 2 new validation points. Similar to the SEED process, in this step we will switch the roles of data points and validation points. We first develop a metamodel of responses with 3 validation points, which is illustrated in Figure 6.23. Prediction errors

of this metamodel at 4 validation points are listed in Table 6.11. A metamodel of prediction errors is then developed with this information and illustrated in Figure 6.24.

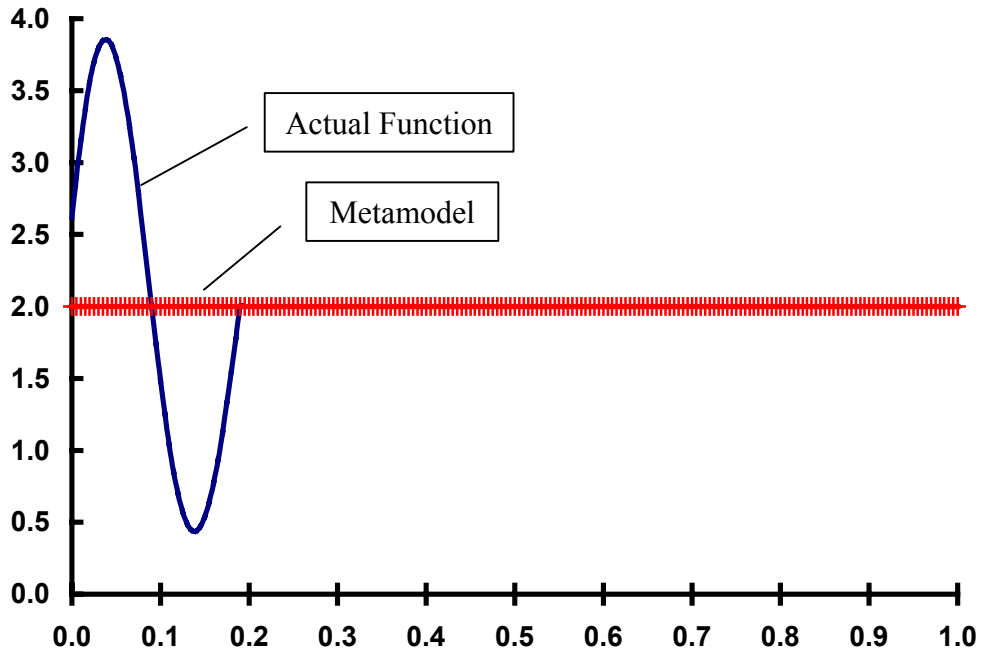


Figure 6.23 Metamodel of Responses Developed with 3 Validation Points in Iteration II – Step 4

Table 6.11 Prediction Errors at 4 Data Points in Iteration II – Step 4

x	0.0	0.167	0.5	1.0
y_{pred}	0.0	0.0	0.0	0.0
y_{actual}	0.618	-0.991	0.0	0.0
y_{error}	-0.618	0.991	0.0	0.0

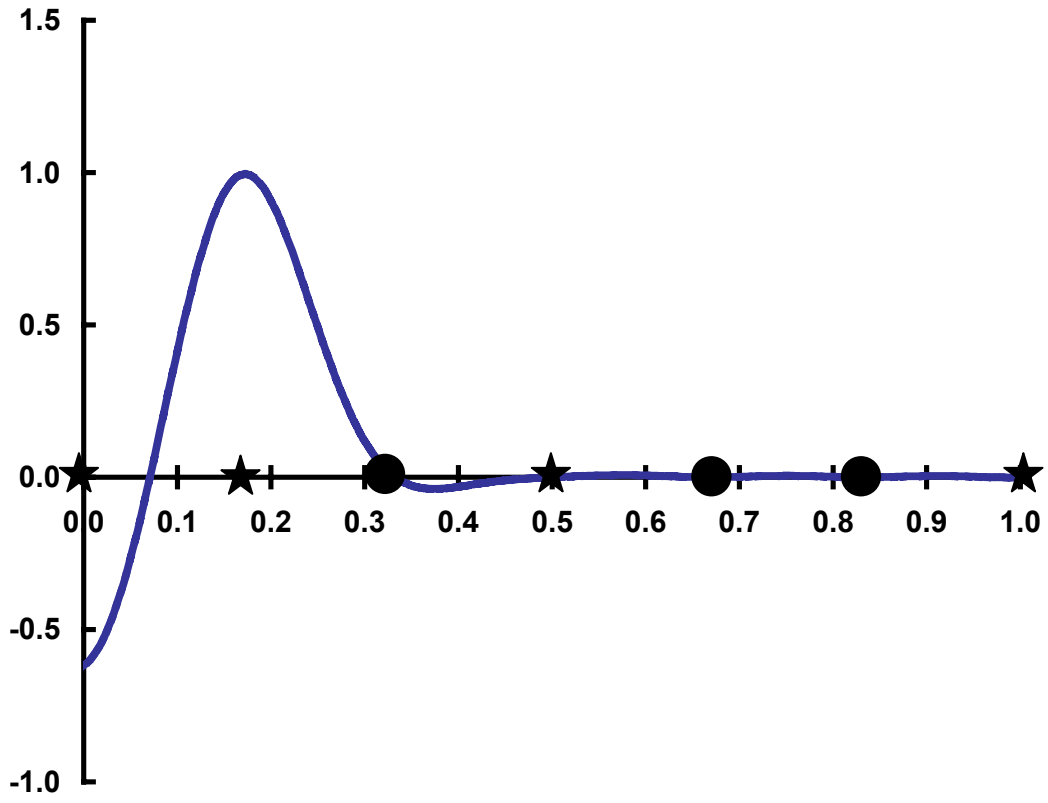


Figure 6.24 Metamodel of Prediction Errors Calculated in Iteration II – Step 4

Since no new point is added in the past steps, the metamodel developed with all observed points is the same as that in Figure 6.19. The values of *goal.achievement* at points in the design space are as illustrated in Figure 6.20. Following the same method as in *Iteration I – Steps 5 and 6*, we identify two new validation points at $x = 0.111$, and $x = 0.243$. All validation points and the prediction errors of the intermediate kriging metamodel (in Figure 6.22) at these points are listed in Table 6.12. Prediction errors at data points are zero.

A kriging metamodel of prediction errors is then developed based on this information and is illustrated in Figure 6.25. The value of θ for this kriging metamodel is 99.99993. The maximum absolute predicted prediction error is $e_{max} \approx 0.63$. This metamodel of prediction errors will be used in the next step to adjust the correlation between points.

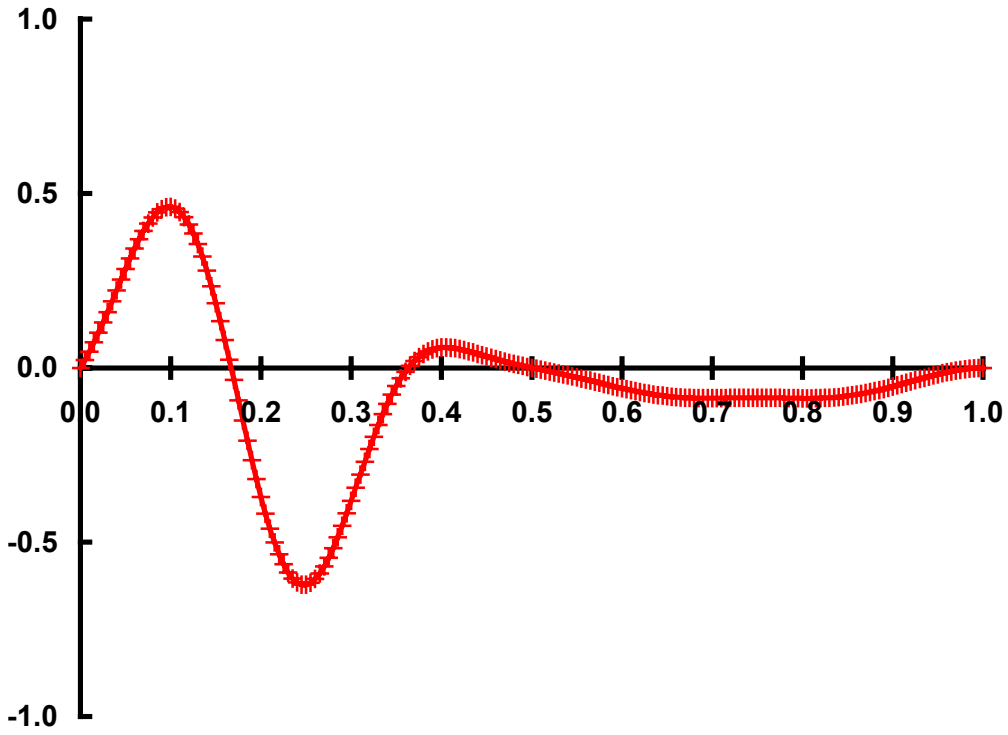


Figure 6.25 Metamodel of Prediction Errors in Iteration II Developed with Information at 9 Observed Points in Iteration II – Step 4

Table 6.12 Validation Points in the 2nd Iteration

x	0.111	0.243	0.333	0.667	0.833
y_{pred}	-0.559	-0.62	-0.145	-0.085	-0.085
y_{actual}	-1.003	0.0	0.0	0.0	0.0
y_{error}	0.444	-0.62	-0.145	-0.085	-0.085

Before going to the next step we need to calculate the degree of achievement of design goals at candidate points in the design space. At this design stage we observed a lot of points with the response value of 0.0. These points spread all over the design space, especially with large x values. This indicates that the actual response function may be flat in most places, with response values close to 0.0. This is useful in our formulation of design goals in the metamodeling process (or say, in the integrated process of metamodeling and design space exploration). Since in this pure minimization example, the response value, 0.0, is far from the target goal value (compared with other observed points), we may set y_{max} as 0.0 instead of the maximum observed response value in following steps. Note that in cases with more design goals (e.g., maximize some other response, robust design goal, etc.), this operation may not be appropriate. In such cases designers need to consider the combined effects from all design goals when trying to set y_{max} or y_{min} at values different from observed ones.

The predicted response value, $y(x)$, is calculated with the metamodel of responses developed with all 9 observed points as illustrated in Figure 6.26. To calculate *goal.achievement*, we set y_{max} = 0.0, y_{min} = -1.15, T_L = -1.6, γ = 1.5. The value of y_{min} is smaller than the observed value, which is –1.003; this is because that from the metamodel of response in Figure 6.26 we observe that the minimum predicted response value is around 0.85. At this stage of design we focus more on the achievement of design goals since we are confident with the metamodel with 9 observed points, thus the value of γ is set at 1.5 instead of 2.0 (note that a smaller value of γ yields larger weight on achievement of design goals). The values of *goal.achievement* at candidate points are

calculated with Equation (6.24). We illustrate the values of *goal.achievement* calculated with $\gamma = 1$ in Figure 6.27.

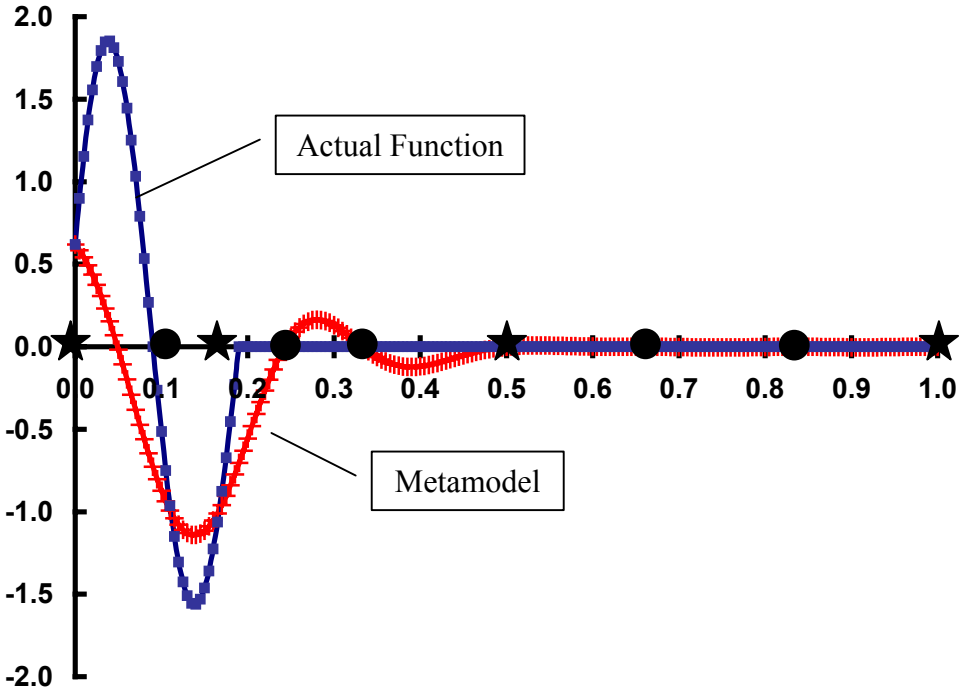


Figure 6.26 Metamodel of Responses Developed with 9 Observed Points in Iteration II – Step 4

Iteration II – Step 5 and 6: Identification of New Data Points and Updated Metamodels. In this step we need to identify 1 new data point. A 5×5 covariance matrix is developed with the first 4 rows and columns corresponding to the 4 data points, and the last row and column corresponding to the new data point. Entries of this covariance matrix are then adjusted with information from the prediction errors and achievement of

design goals. This adjustment is done with Equations (4.28), (6.11), (6.13), (6.19), and (6.24). The values of λ and γ are set as $\lambda = 2$ and $\gamma = 1.5$. The adjustment of entries in the covariance matrices due to prediction errors and achievement of design goals at candidate points, $\alpha_i\gamma_i$, is illustrated in Figure 6.28. Then the determinant of this adjusted covariance matrix is calculated. The new data point is the one that generates the adjusted covariance matrix with the largest determinant. FORTRAN programs are written to facilitate the formulation and adjustment of the covariance matrices and calculation of determinants of matrices.

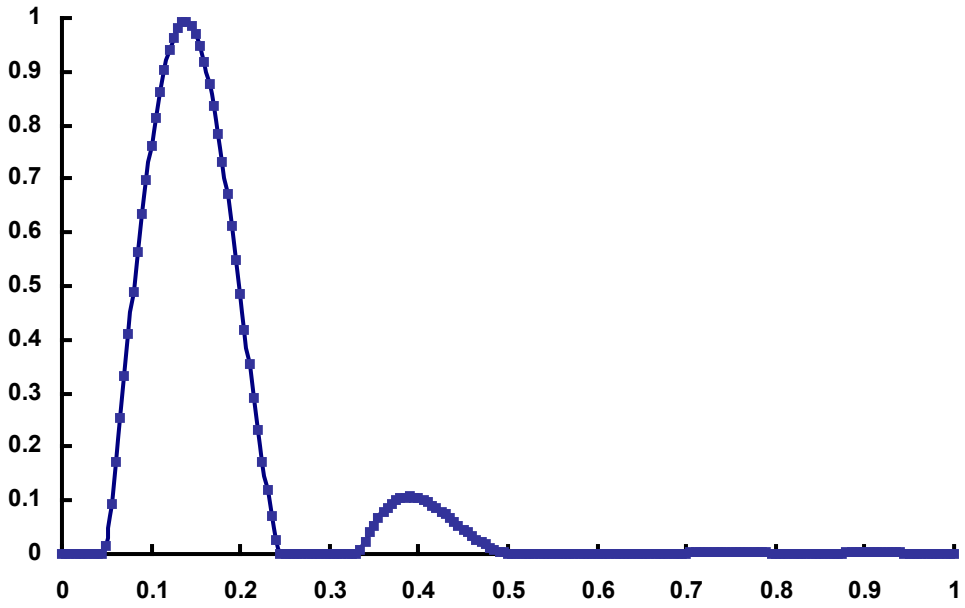


Figure 6.27 Values of *goal.achievement* at Points in the Design Space in Iteration II – Step 4

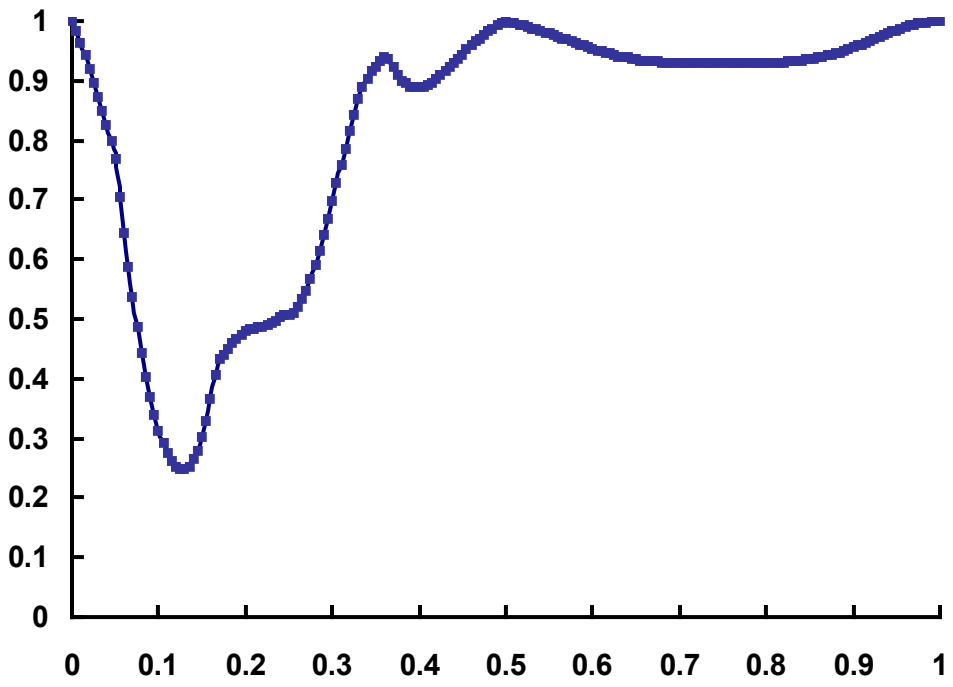


Figure 6.28 Values of $\alpha_i\gamma_i$ at Candidate Points in the Design Space in Iteration II – Step 5

The software iSIGHT is used to link the programs (formulation of covariance matrices, calculation of prediction errors, calculation of degrees of achievement of design goals, the adjustment of covariance matrices, and calculation of the determinant). Various optimization techniques (as what we did in Chapter 4) built in iSIGHT are utilized to find out the point with the maximum determinant of the corresponding adjusted covariance matrix. Organizations and flowcharts of these programs in iSIGHT are presented in Appendix C.

By pursuing the maximum determinant of adjusted covariance matrices, we identify the new data point as $x = 0.131$. Now we have 5 data points as listed in Table

6.13. A kriging metamodel of responses is developed with information at these 5 points and illustrated in Figure 6.29. The value of θ for this kriging metamodel is 99.99985. We do not make metamodel comparison and selection in this example because only kriging metamodels are developed.

Table 6.13 Five Data Points

x	0.0	0.131	0.167	0.5	1.0
y	0.618	-1.522	-0.991	0.0	0.0

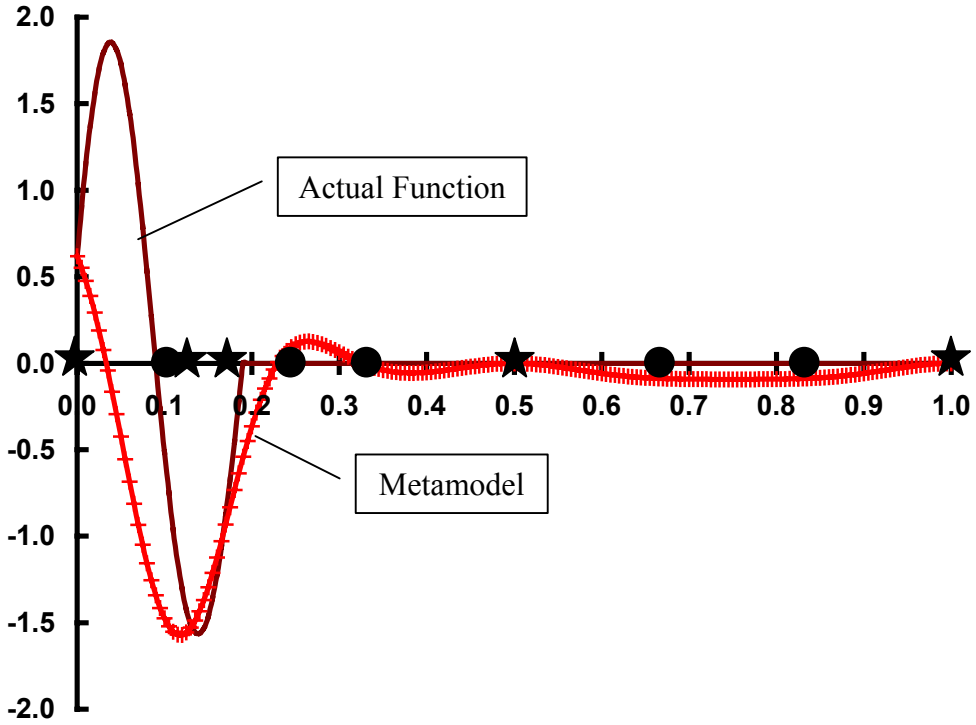


Figure 6.29 Metamodel of Responses Developed with 5 Data Points in Iteration II – Steps 5, 6

Iteration II – Step 7: Analysis of Design. Since the stopping criterion is not met we will enter the next iteration of the integrated process of metamodeling and design space exploration.

Iteration III – Step 4: Identification of New Validation Points. Now we have 5 data points and 5 validation points. In this step we plan to add in 1 new validation point. Similar to what we did in *Iteration II – Step 4*, we switch the roles of data points and validation points in this step. First we need to develop a metamodel of response with 5 validation points. This kriging metamodel is illustrated in Figure 6.30; the value of θ for this metamodel is 99.99982.

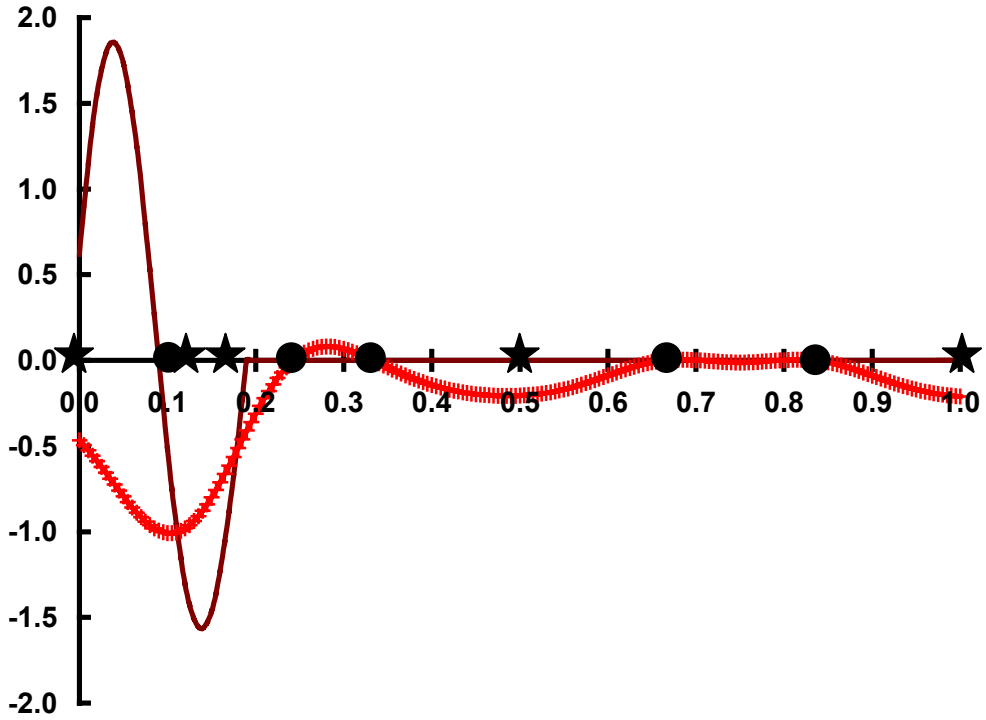


Figure 6.30 Metamodel of Responses Developed with 5 Validation Points in Iteration III – Step 4

Prediction errors of this metamodel at 5 data points are listed in Table 6.14. A kriging metamodel of prediction errors is then developed with information of prediction errors at the data points and validation points. This metamodel is illustrated in Figure 6.31. The value of θ for this metamodel is 100.00. A univariate regression spline metamodel of prediction errors is also developed and illustrated in Figure 6.32.

Table 6.14 Prediction Errors at 5 Data Points in Iteration III – Step 4

x	0.0	0.131	0.167	0.5	1.0
y_{pred}	-0.468	-0.932	-0.644	-0.206	-0.211
y_{actual}	0.618	-1.522	-0.991	0.0	0.0
y_{error}	-1.086	0.59	0.347	-0.206	-0.211

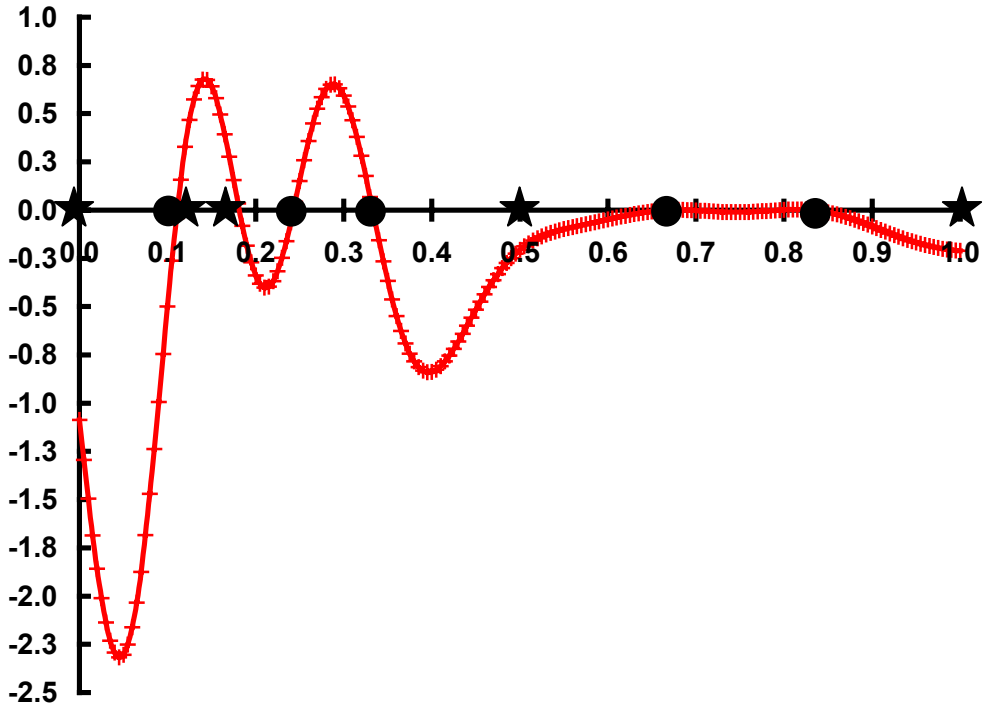


Figure 6.31 Kriging Metamodel of Prediction Errors in Iteration III – Step 4

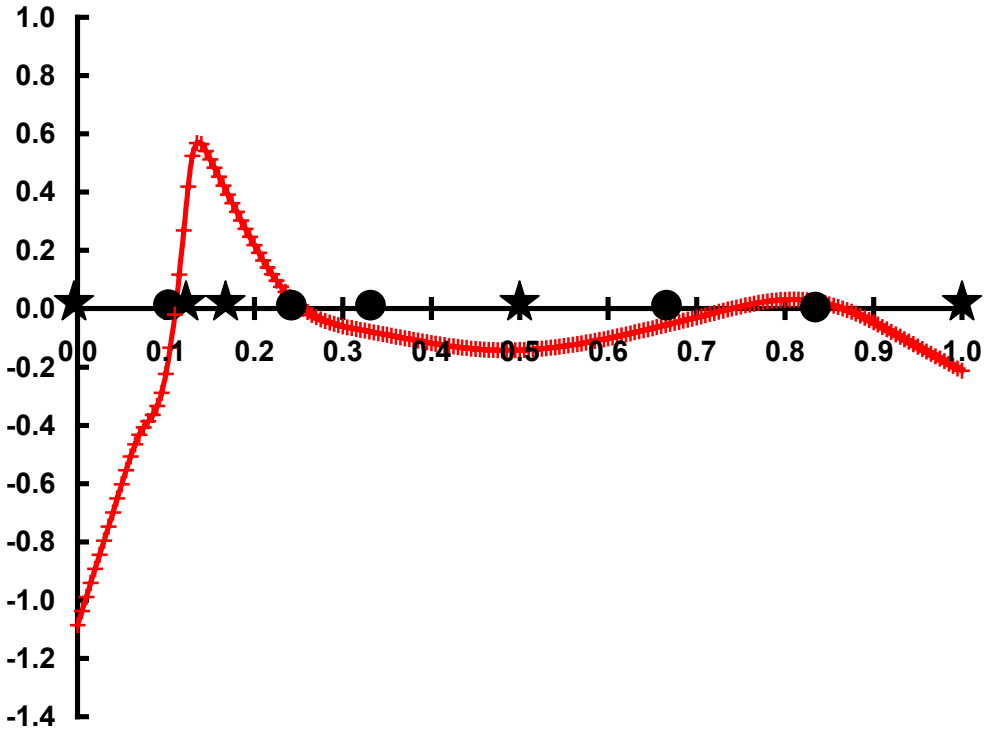


Figure 6.32 Univariate Regression Splines Metamodel of Prediction Errors in Iteration III – Step 4

Comparing Figures Figure 6.31 and Figure 6.32, we observe that the univariate regression splines metamodel is more reliable because it does not have the dramatic fluctuations in unobserved regions as the kriging metamodel (see the peaks or bottoms at x = 0.5, 0.3, and 0.4 in Figure 6.31). Thus in the following steps we will use the univariate regression splines metamodel to calculate prediction errors.

The predicted response value, $y(x)$, can be calculated with the metamodel of responses developed with all 10 observed points as illustrated in Figure 6.33. The value of θ for this metamodel is 100.00. A univariate regression splines metamodel of

responses is also developed and illustrated in Figure 6.34. Comparing Figures Figure 6.33 and Figure 6.34 we see that the kriging metamodel does not work well; the fluctuations around $x = 0.3$ and $x = 0.4$ is abnormal. As having been studied in Chapter 5, kriging cannot model irregular responses well. The kriging metamodel works well in the prediction with very small x values; however, the peak around $x = 0.5$ cannot be validated when the actual response function is unknown and no observation in this region is done. Thus the good performance of this kriging metamodel with small x values is not a systematic solution but just lucky. The univariate regression splines metamodel in Figure 6.34 honestly reflects the response surface based on information from 10 observed points, and will be used in future steps.

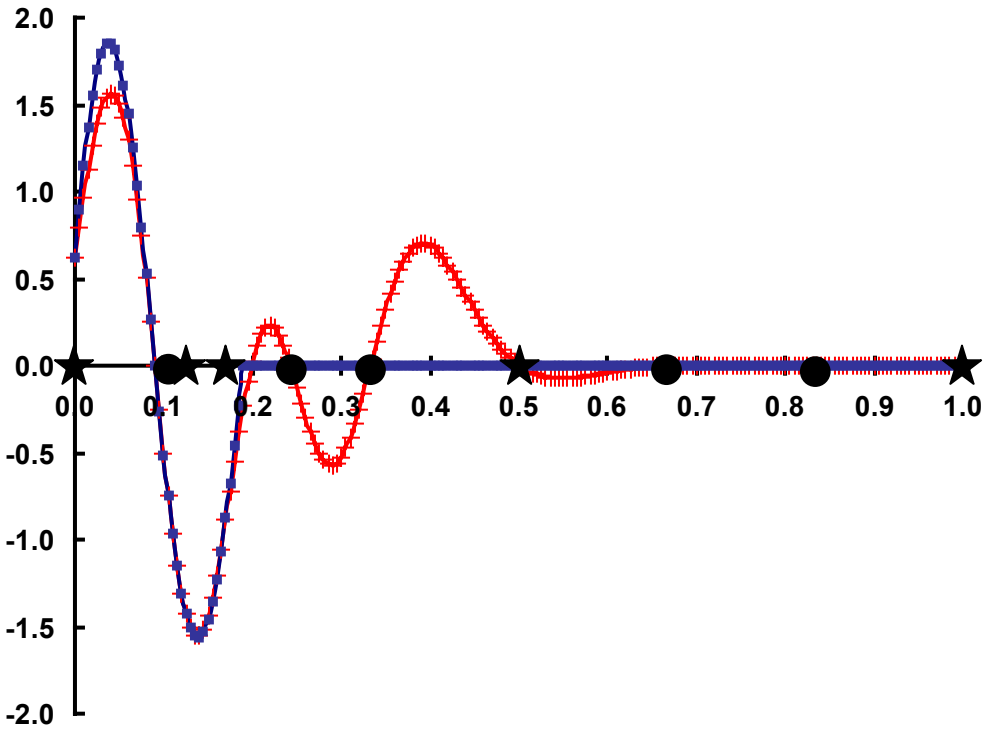


Figure 6.33 Kriging Metamodel of Responses Developed with 10 Observed Points

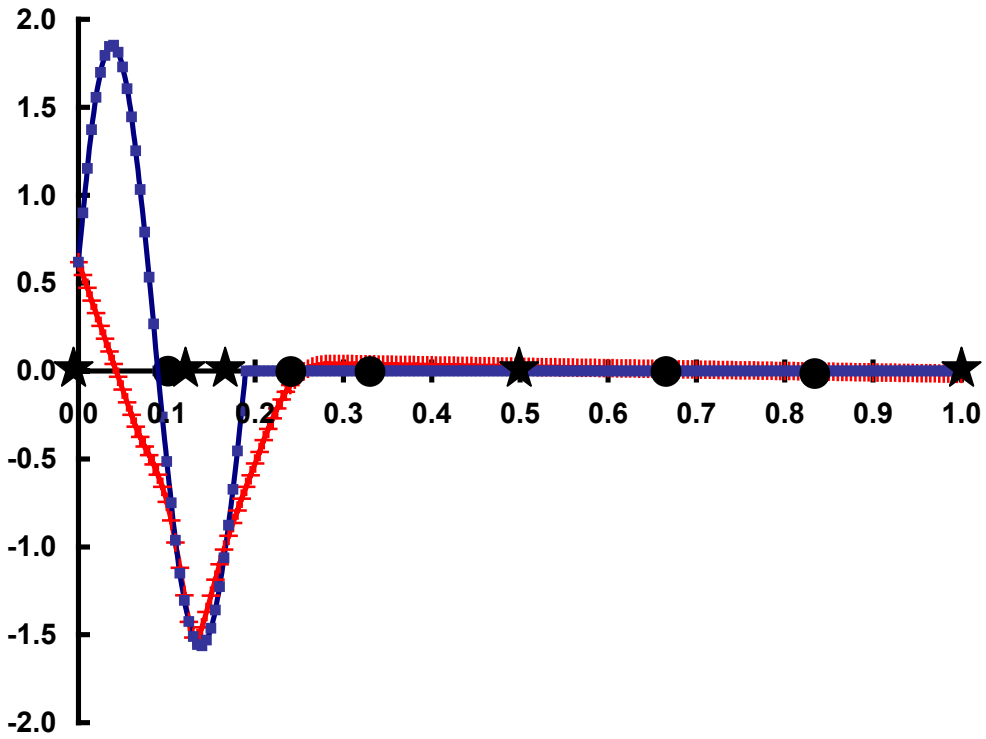


Figure 6.34 Univariate Regression Splines Metamodel of Responses Developed with 10 Observed Points

An 11×11 covariance matrix is developed with the first 5 rows and columns corresponding to the 5 data points, the 6th to 10th rows and columns corresponding to the 5 validation points, and the last row and column corresponding to the new validation point. Entries of this covariance matrix are then adjusted with information from the prediction errors and achievement of design goals. This adjustment is done with Equations (4.28), (6.11), (6.13), (6.19), and (6.24). To calculate *goal.achievement*, we set y_{max} = 0.0, y_{min} = -1.53, T_L = -1.6, γ = 1.25. The value of γ is set as 1.25 because we wish to focus more on the achievement of design goals since we have much confidence

on the accuracy of the metamodel. Values of *goal.achievement* calculated with $\gamma = 1$ are illustrated in Figure 6.35. To calculate *relative.uncert*, we set $\lambda = 2$ and $e_{max} = 0.8$. The adjustment of entries in the covariance matrices due to prediction errors and achievement of design goals at candidate points, $\alpha_i \gamma_i$, is illustrated in Figure 6.36. Then the determinant of this adjusted covariance matrix is calculated. The new data point is the one that generates the adjusted covariance matrix with the largest determinant. FORTRAN programs are written to facilitate the formulation and adjustment of the covariance matrices and calculation of determinants of matrices.

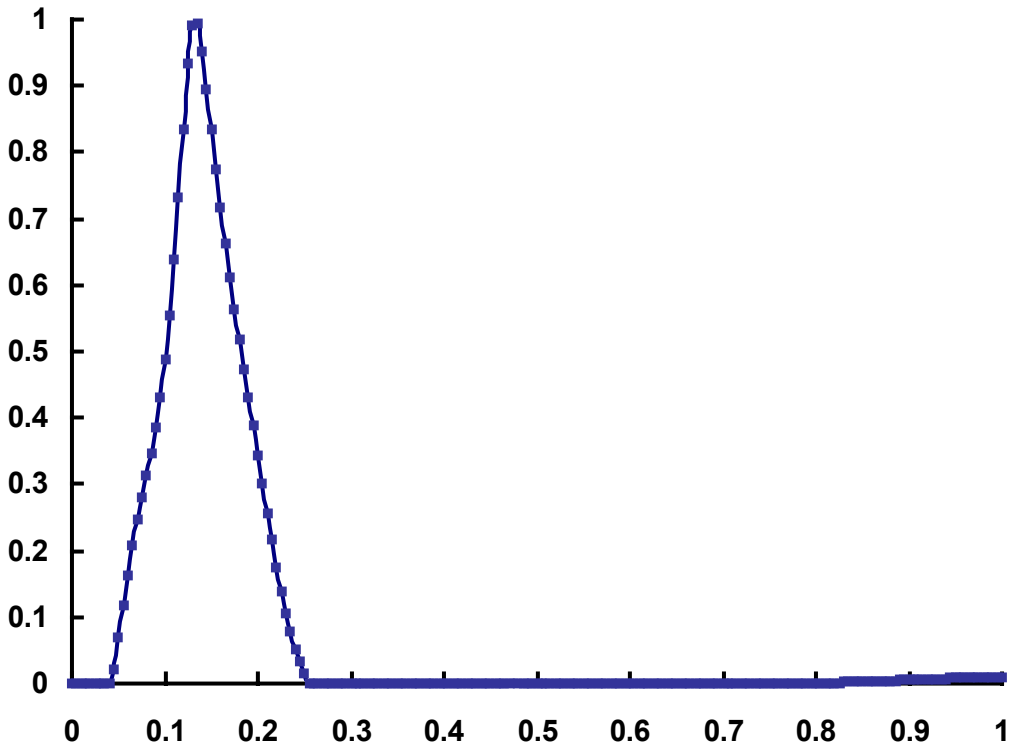


Figure 6.35 Values of *goal.achievement* at Points in the Design Space

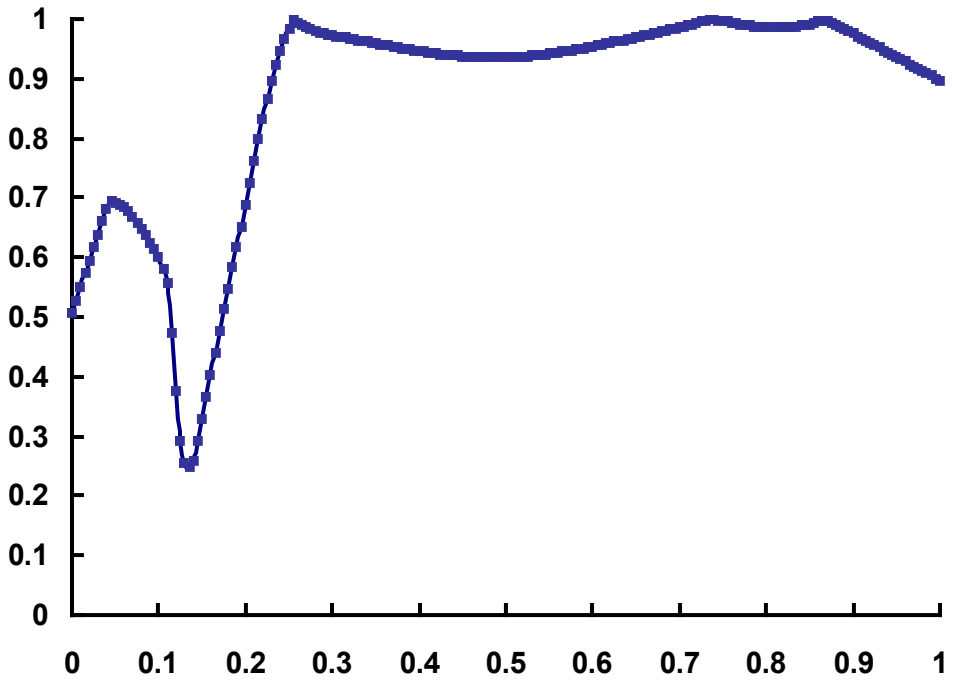


Figure 6.36 Values of $\alpha_i\gamma_i$ at Candidate Points in the Design Space in Iteration III

By pursuing the maximum determinant of adjusted covariance matrices, we identify the new validation point at x = 0.138. Now we have 5 data points and 6 validation points as listed in Table 6.15. Since we have already obtained information at 11 points, the stopping criterion is met and we will stop in this iteration. A univariate regression splines metamodel of responses is developed with information from Table 6.15. This metamodel is illustrated in Figure 6.37. We cannot develop an acceptable kriging metamodel for this example; the reason is explained in our studies in Chapter 5. The univariate regression splines metamodel will be used as the final metamodel for this example.

Table 6.15 Eleven Observed Points

Data Points	***x***	0.0	0.131	0.167	0.5	1.0	
	y	0.618	-1.522	-0.991	0.0	0.0	
Validation Points	***x***	0.111	0.138	0.243	0.333	0.667	0.833
	y	-1.003	-1.564	0.0	0.0	0.0	0.0

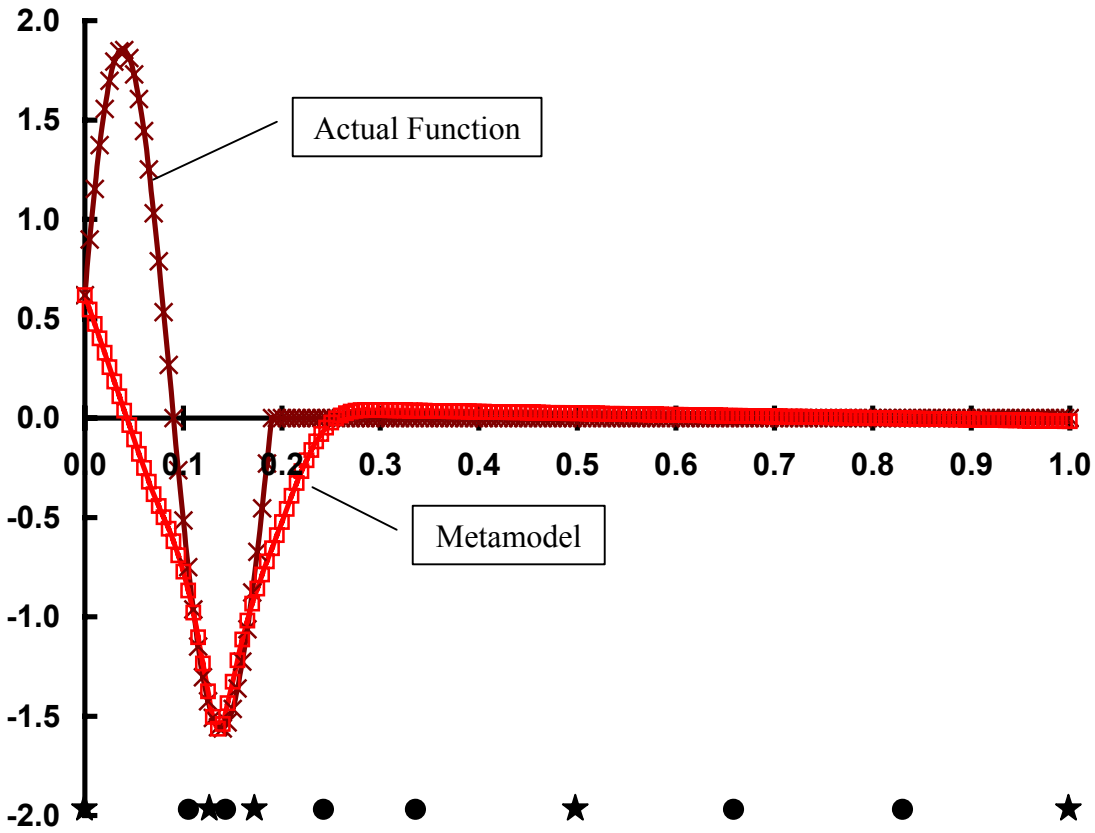


Figure 6.37 Final Univariate Regression Splines Metamodel of Responses Developed with Information at 11 Observed Points

In Figure 6.37 we see that the final univariate regression splines metamodel is not very accurate around $x = 0.04$; it does not grasp the bell shape in this region where the

global maximum response lies. Four points are clustered in the region [0.1, 0.17], while others scatter in the whole design space. As a result, the regression splines metamodel performs well around $x = 0.14$ where the global minimum response lies.

As a comparison, in Chapter 4 we developed a kriging metamodel with the SEED method; the points are listed in Table 4.11 and the metamodel is illustrated in Figure 4.18. In that experimental design, the points are not clustered in the region where design goals are achieved (or almost achieved). Instead, more points are located in regions with large prediction errors; thus the peak around $x = 0.04$ and the bottom around $x = 0.14$ are observed and grasped. The values of root mean squared error (RMSE) and maximum absolute error (MAX) for both metamodels are calculated with information from 201 points and listed in Table 6.16. In Table 6.16 we see that the metamodel from SEED has much smaller values of RMSE and MAX, which supports our impression that the metamodel from SEED is more accurate than the metamodel from E-RCEM.

Table 6.16 RMSE and MAX for Metamodels from SEED and E-RCEM

	Metamodel from SEED Formulation (I)	**Metamodel from Integrated Design Process in E-RCEM**
RMSE	0.113	0.432
MAX	0.371	1.847

In the E-RCEM method, we focus more on the achievement of design goals. In this example, the prediction error still affects the location of new points but its influence is not strong enough to drag our attention from the "bottom" region to the "peak" region on the actual response surface. As a result, the metamodel developed with SEED method

in Chapter 4 is more accurate than the one developed in the integrated process of metamodeling and design space exploration in this chapter. However, a more accurate metamodel does not ensure a better design solution. In Table 6.17 we see that the minimum response of the metamodel developed in E-RCEM is at $x = 0.138$ with the predicted response value of $y = -1.564$, which is the same as the true minimum. This is better than the solution obtained with the metamodel developed with SEED, which is at $x = 0.136$ with the predicted response value of $y = -1.506$. The solutions are subject to round off errors within ±0.0005. In E-RCEM more points are observed around the design solution so we are able to obtain a metamodel that is more accurate in the region of interest. This metamodel may not perform well in "unimportant" regions (e.g., in this example, the region around $x = 0.04$ and that with large x values), but its local accuracy in the "important" region helps obtain a better solution with information from the same number of experiments.

Table 6.17 Minimum Response Values in the Single-Variable Example

	Actual Function	**Metamodel from SEED Formulation (I)**	**Metamodel from the Integrated Design Process in E-RCEM**
x_{min}	0.138	0.136	0.138
y_{min} (predicted)	N/A	−1.506	−1.564
y_{min} (true)	−1.564	−1.560	−1.564

The computational expense of the integrated design process in E-RCEM is only slightly higher than SEED. Only one more program (the one to calculate predicted

response values) is called in the optimization iterations for maximum determinants of the covariance matrices, and a little more calculations are added in the adjustment of covariance matrices. There is no much human interaction in the integrated design process once SEED is implemented. After successfully implementation of SEED in the automatic running mode in the future, it requires little effort to realize an automatic integrated design process in E-RCEM.

From the viewpoint of metamodeling, the traditional process with SEED is better than the integrated design process in E-RCEM because it yields a more accurate metamodel in the whole design space; while from the viewpoint of design space exploration, the integrated design process in E-RCEM is better than the traditional process with SEED because it yields a metamodel with higher local accuracy in critical regions and thus possibly a better design solution. In cases with expensive computer or physical experiments, both the traditional process with SEED and the integrated process in E-RCEM help develop better metamodels with less time and money, and thus ensure better design solutions than traditional experimental designs and design space exploration approaches. When design goals are not well defined at the beginning of design (e.g., in some cases the relative priorities of design goals may change greatly during the design phrase) and it is hard to address this uncertainty, designers may prefer to use SEED to develop globally accurate metamodels. When design goals are clearly defined, designers may prefer to use the integrated design process of metamodeling and design space exploration in E-RCEM to achieve better design solutions faster. In most cases where design goals are defined but still subject to small changes in the future, designers may

prefer to use SEED first to achieve an acceptable metamodel, then use the integrated design process in E-RCEM to explore for new experimental points and design solutions.

6.6 A LOOK BACK AND A LOOK AHEAD

The Efficient Robust Concept Exploration Method (E-RCEM) is developed in this chapter. The integrated design process in E-RCEM is demonstrated and verified with a single-variable example. Research in this chapter helps answer Research Question 3 and its sub-questions; the corresponding hypotheses are tested. Research Question 3, its sub-questions, and corresponding hypotheses are listed below.

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

Hypothesis 3: The processes of metamodeling and robust design space exploration could be integrated through building the information flow from C-DSP to the metamodeling cycle in the Robust Concept Exploration Method.

R.Q.3.1: *How to design sequential experiments with consideration of design constraints?*

Sub-Hypothesis 3.1: Consideration of design constraints could be incorporated in the metamodeling process through construction irregular design spaces.

R.Q.3.3: *How to do sequential metamodeling with consideration of design goals?*

Sub-Hypothesis 3.3: Design goals can be taken into consideration in metamodeling by formulating influential factors with the compromise DSP and using them in maximum entropy sampling.

To answer Research Question 3, the Efficient Robust Concept Exploration Method (E-RCEM) is developed based on the Robust Concept Exploration Method (RCEM) and the method of Sequential Exploratory Experimental Design (SEED). In this chapter, we verified that with the integrated design process in E-RCEM, designers are able to incorporate considerations of metamodel accuracy and achievement of design goals in the experimental design and metamodeling process. New points are identified in regions where design goals are to be achieved or large prediction errors exist. With this integrated design process in E-RCEM (or the *metamodeling **for** design space exploration* approach), designers are able to achieve better design solutions with less time and money spent on expensive computer or physical experiments. Hypothesis 3 is verified; our answer to Research Question 3 is: *Better design solutions can be achieved with fewer experiments by integrating the processes of metamodeling and design space exploration; this integrated design process is realized in E-RCEM, in which information about metamodel uncertainty and achievement of design goals is used as guidance in identifying new points in sequential metamodeling*.

Research Question 3.1 is answered primarily in Section 6.2. Sequential metamodeling with constraints on design variables is studied in Section 6.2.1, and sequential metamodeling with constraints on responses is studied in Section 6.2.2. In this section we show that design constraints can be taken into consideration in the SEED method and the integrated design process in E-RCEM. After taking design constraints into consideration, the design space is usually irregular; with SEED or E-RCEM, new points will be identified only in the reduced irregular feasible design space, and this helps

save time and money spent on experiments wasted in infeasible regions. Our answer to Research Question 3.1 is: *Design constraints can be taken into consideration to define an irregular design space, and SEED or E-RCEM can be used to identify new points in the reduced irregular feasible design space*.

Research Question 3.3 is studied and answered in Section 6.3. Based on the compromise DSP, the degree of achievement of design goals at candidate points can be formulated and scaled in [0,1]; a value close to 0 means that design goals are hardly achieved, and a value close to 1 means that design goals are almost achieved at this point. Usually we preset a target value for the design goal, and once this target value is met or exceeded, we set the degree of achievement of design goals to be 1. This quantitative expression of degree of achievement of design goals can be used in the adjustment of covariance matrices in maximum entropy sampling, and "drag" new points to regions where design goals are met or almost met. Our answer to Research Question 3.3 is: *The degree of achievement of design goals at a particular point can be quantitatively formulated with the compromise DSP and used as an influential factor in SEED or E-RCEM*.

The Efficient Robust Concept Exploration Method (E-RCEM) is developed in Section 6.4. There are three ways to organize design processes in E-RCEM: the ***Traditional Process*** (SEED $\rightarrow$ design space exploration), the ***Integrated Design Process*** (SEED + design space exploration), and the ***Hybrid Process*** (traditional $\rightarrow$ integrated). The traditional process has already been studied and implemented in Chapters 4 and 5, thus in this section we describe the integrated design process in detail. A single-variable

example is presented in Section 6.5, implementing the integrated design process in E-RCEM. It is shown that with the integrated design process in E-RCEM, better design solution is achieved than that obtained with the traditional process.

Research in Chapter 6 is built on that in Chapters 4 and 5, and should be viewed from a higher level. In Chapters 4 and 5 we focus on the metamodeling process, while in this chapter we consider the whole design process in the early stages: problem initialization, metamodeling, and design space exploration. The E-RCEM is an integrated robust design method developed for efficient and effective identification of design solutions at early stages. It can also be viewed as or has the potential to be developed to a new optimization algorithm or heuristic.

The SEED method introduced in Chapter 4, the sequential metamodeling strategy studied in Chapter 5, and the integrated design process in E-RCEM developed in Chapter 6 will be implemented in Chapter 7 with a more complicated engineering problem.

CHAPTER 7

ENGINEERING APPLICATION: DESIGN OF UNIT CELLS FOR LINEAR CELLULAR ALLOYS

In this chapter, the method of Sequential Exploratory Experimental Design (SEED), sequential metamodeling, and the Efficient Robust Concept Exploration Method (E-RCEM), are applied in the engineering application of design of unit cells for linear cellular alloys (LCA). The results are compared with that from the existing approximation-based design method in the Systems Realization Laboratory (i.e., RCEM without loops in metamodeling and information feedback from design space exploration to metamodeling). Research questions visited in this chapter are R.Q.2, R.Q.3, R.Q.4 and their sub-research questions. After an introduction to the thermal topological design of unit cells for linear cellular materials in Section 7.1, the design problem is defined in Section 7.2 and the traditional design method of RCEM is applied in Section 7.3. The SEED method and sequential metamodeling approach is applied in Section 7.4. The integrated design process in E-RCEM is applied in Section 7.5. Comparisons and discussions are presented in Section 7.6.

7.1 BACKGROUND OF DESIGN OF LINEAR CELLULAR ALLOYS

The thermal topological design of unit cells for linear cellular alloys (LCA) is used in this chapter as a case study with which we compare the performance of SEED, E-RCEM and traditional robust design methods like RCEM. This design example is taken from studies in (Seepersad, et al., 2003). The background of linear cellular alloys, topology design, the example, and the finite element model and simulation are introduced in this section.

7.1.1 Topology Design

In topology design designers simultaneously adjust both the external shape *and* the number and shape of internal boundaries for a given 2D or 3D domain and associated boundary conditions and design objectives (Eschenauer and Olhoff, 2001; Rozvany, 2001). Vastly different topologies can be obtained from an arbitrary initial domain with topological design techniques. Important properties like compliance, stiffness, strength, eigenfrequencies, convective coefficients, and other properties sensitive to material arrangement can be tailored through the adjustment of the topology of a structure. It is possible to distribute material strategically, resulting in lightweight structures with desirable properties. Emerging manufacturing processes (e.g., additive fabrication and processing of cellular materials) facilitate the fabrication of structures with nearly arbitrary topologies.

In Topology Design the following question is to be addressed: How can material be distributed efficiently in a given design region to tailor properties that are sensitive to

material distribution (e.g., compliance, stiffness, strength, convection, etc.)? In topology optimization nothing is known about structure or shape a priori; the shape and number of discontinuities (i.e., voids) are determined during the course of topology optimization. A typical topology design approach, as proposed by Carolyn Conner Seepersad in her PhD proposal involves the following steps:

Step 1 - Establish design requirements, objectives, and domain.

Step 2 - Divide domain into finite elements.

Step 3 - Assign density variable to each finite element (ρ_i).

Step 4 - Modify density variables according to solution (optimization) algorithm. Small density values for an element imply that the element is empty (i.e., part of a hole). Large density values imply solid material.

Step 5 - Calculate effective properties of structure.

A. Select penalization power, p>3. The penalization power penalizes intermediate densities and encourages convergence to regions of solid (full density) and void (minimum density).

B. Calculate effective properties in each element. For example, a stiffness matrix (K) for an element becomes: $K_i = \rho_i^p K_{solid}$

C. Calculate effective properties for the structure.

Step 6 - Return to Step 4 until convergence is achieved.

Explorations of the appropriate topology are to be incorporated in our research as a future direction. However, in this dissertation, our focus is on the synthesis of design processes involving mechanical and material design.

The computational model for topology design used in the example in this chapter stems from a 99 line MATLAB® code for compliance minimization of statically loaded

structures, developed by Ole Sigmund from the Department of Solid Mechanics at the Technical University of Denmark (Sigmund, 2001). The code was intended for engineering education and contains both a mesh independency filter and a finite element code. A number of simplifying assumptions are made to reduce the code complexity. For example, the design domain is modeled as a rectangle and is discretized using square finite elements, as indicated in Figure 7.1. Element and node numbering proceeds on a column-by-column basis, starting in the upper left corner. The aspect ratio of the structure to be optimized is determined by the number of horizontal (*nelx*) and vertical (*nely*) elements as specified by the user.

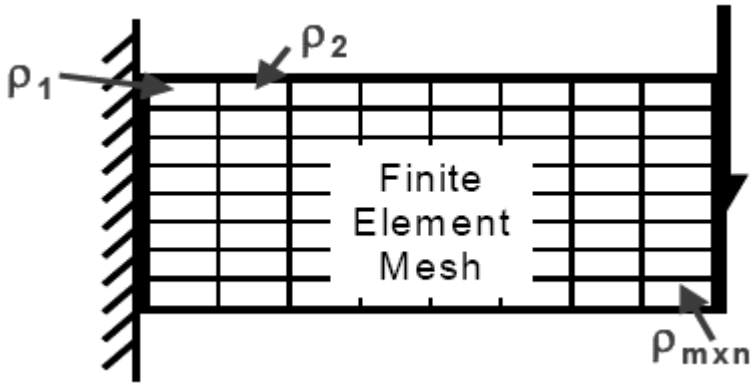


Figure 7.1 Dividing the Cantilever Beam Design Domain into Finite Elements (Choi and Fernandez, 2003)

The chosen implementation of topology optimization within this algorithm is based on the "power law approach" or SIMP approach (Solid Isotropic Material with Penalization), where properties are assumed constant within each element and design

variables are the element relative densities. For more information about this topology optimization problem, please refer to (Sigmund, 2001).

7.1.2 Linear Cellular Alloys

Linear Cellular Alloys (see Figure 7.2) are metallic cellular materials with a constant cross section, fabricated through a process developed by the Lightweight Structures Group at Georgia Tech (Seepersad, et al., 2003). The process combines extrusion of ceramic slurry, composed of metal oxides and water through a die, allowing for the achievement of quasi-arbitrary two-dimensional cellular topologies. Extrusion of the ceramic is followed by exposure to thermal and chemical treatments that cure the composites. The inherent advantage in producing materials using this process is the ability to tailor properties of the resulting structure such as the effective moduli of elasticity and conductivity by altering the topologies of the cells. Structures may be composed of either periodically repeating unit cells or functionally graded, non-uniform cells of various topologies.

Linear or two-dimensional cellular materials are particularly suitable for multifunctional applications that require not only structural performance but also lightweight thermal or energy absorption capabilities. LCAs are superior to those of metallic foams with equivalent densities. For example, LCAs exhibit greater in-plane stiffness and strength and out-of-plane specific energy absorption than stochastic metal foams (Evans, et al., 2001; Hayes, et al., 2001). LCAs are advantageous as heat exchangers due to larger surface area density and lower pressure drop – two factors that

compensate for lower heat transfer coefficients for laminar forced convection than for turbulent forced convection in stochastic metal foams with comparable relative densities (Lu, 1999). Accordingly, LCAs have potential for use in applications such as actively cooled supersonic aircraft skins or engine combustor liners (Seepersad, et al., 2002).

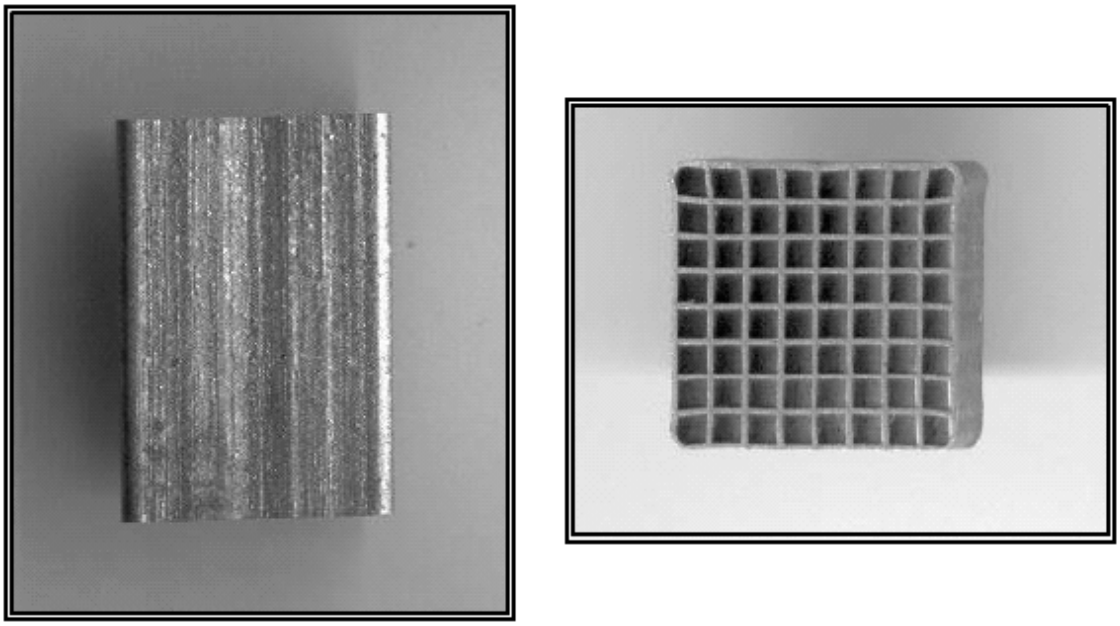


Figure 7.2 Square-Cell Linear Cellular Alloy (Hayes, et al., 2001)

In addition, the manufacturing process for linear cellular alloys facilitates the fabrication of multi-functional cellular materials. Powder slurries are extruded through a die and then exposed to thermal and chemical treatments in a process developed by the Lightweight Structures Group at Georgia Tech (Cochran, et al., 2000). Extruded metallic cellular structures can be produced with nearly arbitrary two-dimensional cellular topologies limited only by paste flow and die manufacturability. Wall thicknesses and

cell diameters as small as fifty microns and several hundred microns, respectively, have been manufactured (Church, et al., 2001).

As presented in (Seepersad, et al., 2003), several authors have reported multifunctional analyses of two-dimensional cellular materials. "Torquato and coauthors establish cross-property bounds on the thermal conductivities of periodic hexagonal, triangular, and square cells in terms of elastic properties and vice versa (Torquato, et al., 1998). Gu and coauthors present analytical models and dimensionless indices that enable simultaneous evaluation of structural and heat transfer performance of periodic hexagonal, square, and triangular cells (Gu, et al., 2001). Structural performance is measured in terms of the effective shear modulus while a corrugated wall model (Lu, 1999) is recommended for heat transfer. The non-dimensional indices include a thermal performance index—the ratio of total heat transfer rate to pressure drop—and a thermomechanical index formulated by multiplying the thermal index by the ratio of shear modulus to the modulus of elasticity of the solid material. Both Gu and coauthors and Evans and coauthors (Evans, et al., 2001) employ the indices to evaluate the performance of periodic triangular, square, and hexagonal topologies for thermomechanical applications. Hayes and coauthors use theoretical estimates and physical experiments to evaluate several thermal and mechanical characteristics of LCAs, including total heat transfer rate, elastic properties, initial plastic buckling strengths, and in-plane and out-of-plane compressive strength, collapse behavior, and energy absorption for both quasi-static and dynamic loading (Hayes, et al., 2001). The steady state heat transfer rate is evaluated for periodic square cells using a finite difference approach that is

more rigorous than closed-form estimates because it accounts for three-dimensional temperature distribution throughout the LCA and the convective fluid. The finite difference approach can accommodate functionally graded cell topologies, although Hayes and coauthors did not leverage this capability" (Seepersad, et al., 2003).

In (Seepersad, et al., 2003), Seepersad and co-authors design multifunctional, two-dimensional cellular structures for applications that require both structural and thermal performance. While others have focused primarily on *analysis* of the structural and thermal properties of cellular materials, the authors adopt a *design* perspective; given a set of rigorous analytical models, their emphasis is on *synthesis* of cellular designs and identification of superior design regions. The example used in this chapter is modified from their studies in the referenced paper.

7.1.3 Convectively Cooled Heat Sink for a Computer Chip

LCAs are potentially well suited for heat exchanger applications, including compact electronic cooling devices and ultralight, actively cooled, aerospace structures. Unlike most heat exchangers, however, the two-dimensional cells that dissipate heat via conduction and convection also have desirable structural properties.

The LCA example considered in this chapter is that of a convectively cooled heat sink for a computer chip. A sample schematic of the structure is given in Figure 7.3. The general requirements for a CPU heat sink are that it 1) remove enough heat from the chip so as to ensure steady state operation and 2) withstand the relatively high compressive forces exerted by clamps used to attached the heat sink to the chip as tightly as possible

(see Figure 7.4). With this in mind, it is important to note that constant temperature at the chip interface is desired in this investigation. Although, it may seem more intuitive to model constant heat flux instead, the idea is to design a heat sink that is capable of removing enough heat to keep the chip below 1) its maximum operating temperature or 2) its melting temperature (in the case of potential over-clocking).

In Figure 7.3, the device has fixed overall width (W), depth (D), and height (H) of 25 mm, 75 mm, and 25 mm, respectively. It is insulated on the left, right, and bottom sides and is subjected to a heat source at constant temperature, T_s, on the top face. The mechanism for heat dissipation is forced convection via air with entry temperature, T_{in}, and total mass flowrate $\dot{M}$. The flowrate is variable, but it is linked to the available pressure head through a representative characteristic fan curve, illustrated in Fig, 2. Steady state, incompressible laminar flow is assumed. The solid material in the device is copper. The thermal conductivity, k_s, of copper samples fabricated with the thermo-chemical LCA extrusion process has been measured to be 363 W/m-K [5].

In (Seepersad, et al., 2003), the LCA is composed exclusively of rectangular cells, but the size, shape, and number of cells are permitted to vary in a graded manner. Each row of cells may assume a different height, h_i, and each column a different width, w_i. The only restriction on cell height and width is that the cells must fit within the external dimensions with sufficient remaining space for vertical cell walls of variable thickness, t_h, and horizontal walls of variable thickness, t_v. The numbers of cells in the horizontal and vertical directions are designated N_h and N_v, respectively. The goal for the example in

(Seepersad, et al., 2003) is to achieve desirable values for two objectives: (1) overall rate of steady state heat transfer and (2) overall structural elastic stiffness of the structure.

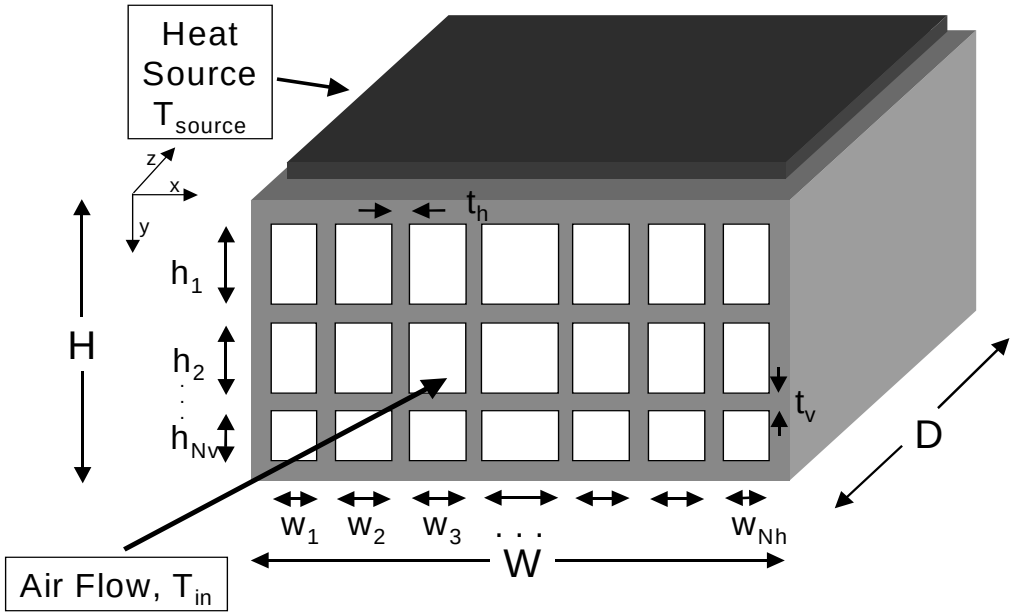


Figure 7.3 Compact, Forced Convection Heat Exchanger with Graded Rectangular LCAs (Seepersad, et al., 2003)

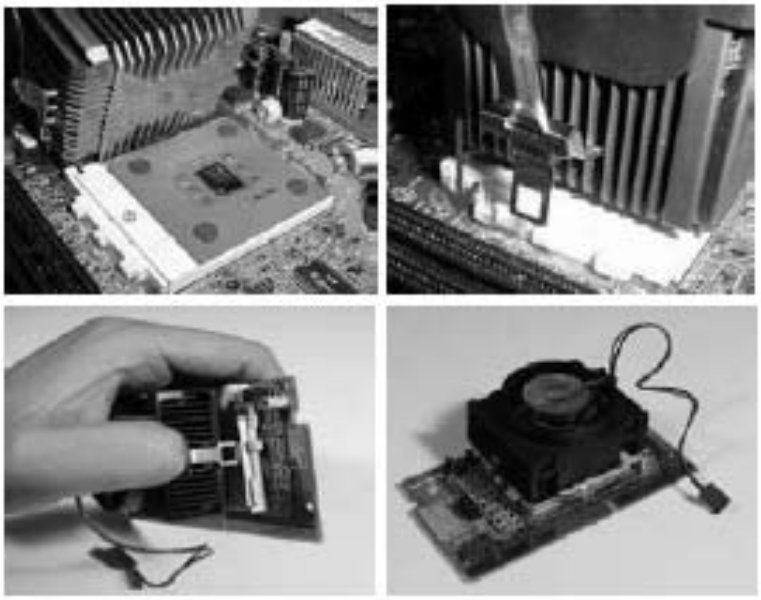


Figure 7.4 Steps Involved in CPU/Heat Sink Assembly (Choi and Fernandez, 2003)

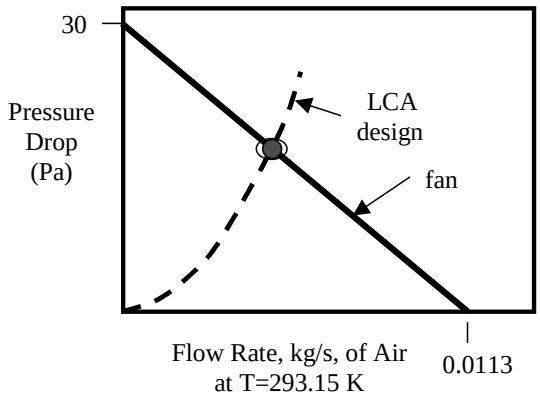


Figure 7.5 Characteristics Fan Curve (Seepersad, et al., 2003)

7.1.4 Finite Element Modeling and Computer Simulation

As stated before, typically, topology design and optimization involve the general steps outlined in Figure 7.6. As indicated, every change in geometry requires renewed analysis to evaluate system performance with regard to desired objectives. Considering that such changes in geometry also require the recalculation of temperature dependent (i.e., inlet, outlet, and bulk) properties such as fluid viscosity μ, convective coefficient $\boldsymbol{h}$, Prandtl Number $\boldsymbol{Pr}$, Reynolds Number $\boldsymbol{Re}$, Hydraulic Diameter $\boldsymbol{D_h}$, etc. and the reevaluation of potentially huge stiffness matrices computational expense is considerable. This is especially true when a number of different software applications are involved.

Through an adaptation of the 99 line topology optimization algorithm, developed by Ole Sigmund and extended by Carolyn Conner Seepersad, as described in Section 7.1.1, Finite Element Thermal and Structural analysis has been developed and successfully deployed in MATLAB.

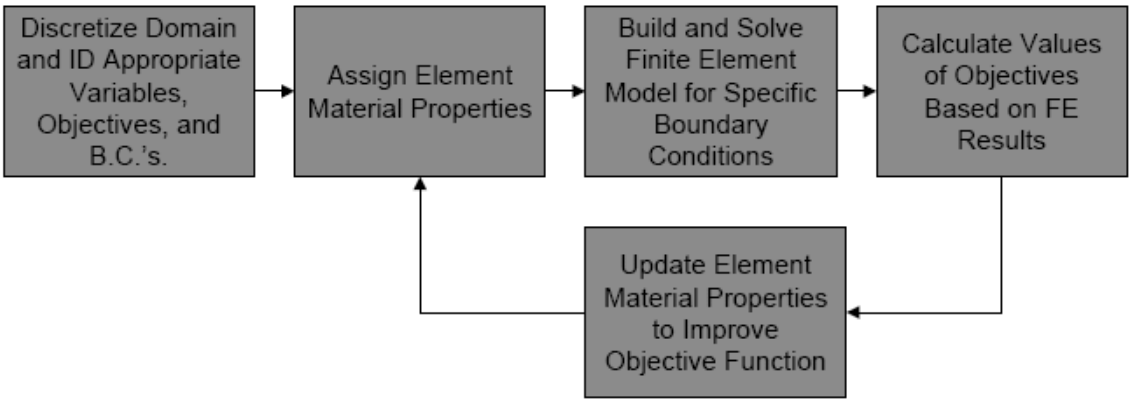


Figure 7.6 General Step for Topology Design and Optimization (Adapted from Choi and Fernandez, 2003)

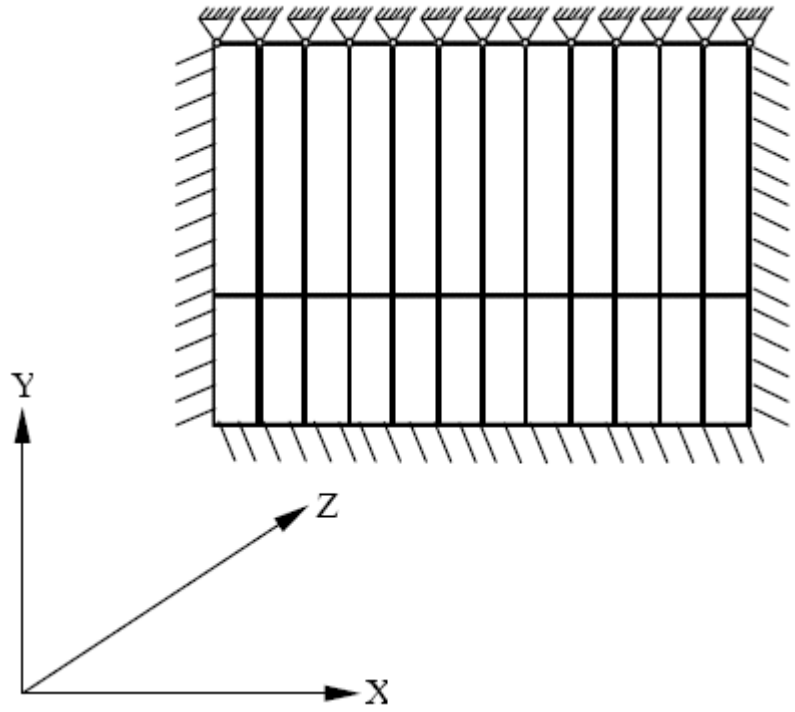


Figure 7.7 FEA Boundary Conditions (Adapted from Choi and Fernandez, 2003)

The boundary conditions and coordinate system used for the thermal and structural finite element analysis are provided in Figure 7.7. A number of simplifying assumptions are made in implementing the thermal and structural analysis for multi-objective topology design. The fluid temperature difference between inlet and outlet is assumed to be very small.

In this chapter, the thermal analysis model is different from that used in (Seepersad, et al., 2003). The simulation used in this chapter is not very accurate since it is not the focus of studies in this dissertation; simulations with low-fidelity are used here because the low cost enables us to observe thousands of points to illustrate the effectiveness of the SEED and E-RCEM methods in this chapter. In the next section, this simulation and structural and thermal analysis will be used to construct the example problem to be used in this chapter.

7.2 EXAMPLE PROBLEM: DESIGN OF UNIT CELLS FOR LINEAR CELLULAR ALLOYS

The example problem of design of unit cells for linear cellular alloys is defined in this section. Structural and thermal models introduced in Section 7.1 will be used as simulations in this example. The example here is slightly different from that in (Seepersad, et al., 2003) because our focus is to illustrate and verify the SEED and E-RCEM methods instead of exploring topology designs.

In this chapter, the convectively cooled heat sink for a computer chip in Figure 7.3 is used as the example problem. Steady state, incompressible laminar flow of air is

assumed. The temperature of the inlet flow is a constant, T_{in} = 293K. The temperature of the heat source, which is put on the top of the LCA device as illustrated in Figure 7.3, is considered to be a constant of T_{source} = 373K. The thermal conductivity is set as 365W/mK, which is that of the copper samples fabricated with the thermal-chemical LCA extrusion process in (Church, et al., 2001). The depth of the device is set as D = 0.075m. The width and height of the device is $W = H$. The number of cells in the horizontal and vertical directions is set as $N_h = N_v = N = 8$. The overall structure of this LCA is defined and will not change in our example. Identical rectangular cells are used, with $h_1 = h_2 = \ldots = h_{nv} = w_1 = w_2 = \ldots = w_{nh} = w$ in Figure 7.3. The wall thickness is set as $t_v = t_h = t$. The relationship between wall thickness, t, and cell size, w, follows Equation (7.1):

$$W = t \cdot (N+1) + w \cdot N = 9t + 8w \tag{7.1}$$

where as introduced earlier, N is the number of cells in the vertical or horizontal direction, and W is the width of the device and in this example, $W = H$. The total area of the cross section that the working fluid (air) passes the device is:

$$A_f = N^2 \cdot w^2 = (W - 9t)^2 \tag{7.2}$$

And the area of the cross section that is filled with solid materials is:

$$A_s = A - A_f = W \cdot H - N^2 \cdot w^2 = W^2 - (W - 9t)^2 \tag{7.3}$$

In this example we consider 3 design variables, as described below:

- Wall thickness, t. The wall thickness is used as a control factor in our example. The ranges for t is 0.0002m ≤ t ≤ 0.0008m.

- Width of the device, *W*. The width of the device is a control factor. The ranges for *W* is 0.015m ≤ *W* ≤ 0.035m.
- Fluid velocity, *V*, or total mass flow rate, $\dot{M}$. Air is the working fluid. We assume that 1) the fluid velocity is the same at any places in the device, and 2) as a noise factor, the air temperature does not change greatly so that a constant density of air, ρ_f, can be used, we have the relationship between the fluid velocity and total mass flow rate as:

$$\dot{M} = \rho_f \cdot A_f \cdot V \tag{7.4}$$

Thus, in this example, we need to use only one of the two variables. The density of air at 20 ^{o}C is ρ_f = 1.205 kg/m^3. The mass flow rate, $\dot{M}$, is used as one of the control factors. In this example, we set the boundaries for $\dot{M}$ as 0.0005kg/s ≤ $\dot{M}$ ≤ 0.003kg/s.

There are three system constraints in this design:

- In this example we assume to have steady state, incompressible laminar flow in the LCA device. Typically, a flow is laminar when the Reynolds number is smaller than 2300, and this is the second constraint in this example:

$$R_e < 2300 \qquad (\textit{Constraint I})$$

where the Reynolds number, R_e, is calculated with Equation (7.5).

$$R_e = \frac{D_h \cdot V \cdot \rho_f}{\mu_f} \tag{7.5}$$

where D_h is the hydraulic diameter, V is the fluid velocity, ρ_f is the density of air, and μ_f is the fluid dynamic viscosity. In this example, the dynamic viscosity is calculated with the following equation:

$$\mu_f = 0.0000001 \times \left(0.4415 \cdot T_{average} + 51.638\right) \tag{7.6}$$

where $T_{average}$ is the average fluid temperature and in this example, we take the fluid inlet temperature as the average temperature:

$$T_{average} = T_{in} \tag{7.7}$$

In Equation (7.5), the hydraulic diameter D_h is calculated with the following equation:

$$D_h = \frac{4A_f}{WettedPerimeter} \tag{7.8}$$

In this example, since we use $N_h = N_v = 8$ square cells along the horizontal and vertical directions, Equation (7.8) can be simplified to:

$$D_h = W - 9t \tag{7.9}$$

This constraint is very important in our example because the simulation code in Section 7.1.4 is developed specifically for cases with laminar flows; the results may not be valid when the flow is developing or turbulent. This constraint puts limits on wall thickness, t, and fluid velocity, V. The fluid inlet temperature, T_{in}, also affects the value of Reynolds number.

- As illustrated in Figure 7.5, there is a constraint associated with the pressure drop and mass flow rate. It is required that we must design the LCA device

with pressure drop and mass flow rate (along the LCA curve) smaller than those at the cross point in Figure 7.5. This can be expressed as:

$$\Delta P \leq 30 - 2663.35\dot{M} \qquad (\textit{Constraint II})$$

where $\dot{M}$ can be calculated with Equation (7.4). The pressure drop in a horizontal, steady state flow in a duct can be calculated with the following equation:

$$\Delta P = f \cdot \frac{D}{D_h} \cdot \frac{\rho_f \cdot V^2}{2} \qquad (7.10)$$

where f is the friction coefficient, D is the length of duct or pipe (in this example, the depth of the LCA device), ρ_f is the density of air, and V is the fluid velocity. In this example, to be simple, we use the hydraulic diameter, D_h, to calculate the pressure drop; it should be noted that to be accurate, the equivalent diameter, D_e, should be used since LCA is a rectangular duct. For fully developed laminar flow the friction coefficient depends only on the Reynolds Number, *Re*, and can be expressed as:

$$f = \frac{64}{R_e} \qquad (7.11)$$

Pressure drop for non-laminar flows is not considered in this example because design solutions with such flows are not considered due to Constraint I.

- Performance requirements include a constraint on the volume fraction of the unit cell and goals for the elastic properties of the cellular material. The

volume fraction (or portion of the unit cell occupied by solid material), *vf*, is limited to at most 30% by the manufacturing process.

$$vf = \frac{A_s}{A} = \frac{A - A_f}{A} = 1 - \frac{\left(W - 9 \cdot t\right)^2}{W^2} \leq 30\% \qquad \text{(}\textit{Constraint III}\text{)}$$

There are three design goals considered in this example:

- Maximize the total heat transfer rate, *Q*. With the finite element model as introduced in Section 7.1.4, we are able to calculate the exit temperatures of the fluid in each cell, and the total rate of steady state heat transfer is then calculated by a summation over all the cells (Incropera and DeWitt):

$$Q = \sum_{i}^{n\ cells} \dot{m}_{cell_i} c_{p_{ave}} \left(T_{exit_i} - T_{in}\right) \qquad (7.12)$$

The total heat transfer rate *Q* is directly obtained from the simulation code. We desire to maximize the heat transfer rate of the LCA device to cool down the computer chip. In this example, we formulate this goal in the compromise DSP as:

$$\frac{Q - Q_{\min}}{Q_{target} - Q_{\min}} - 1 + d_1^- + d_1^+ = 0 \qquad (7.13)$$

In Equation (7.13), *Q* is the total heat transfer rate at the current point, Q_{min} is the minimum observed total heat transfer rate, and Q_{target} is the target value for the heat transfer rate, which is set as 20W. Note that there are different ways to formulate the goal (normalize the responses) in the compromise DSP, as

described in Chapter 6. The deviation variables, d_1^- and d_1^+, satisfy the following requirements:

$$d_1^-, d_1^+ \geq 0 \quad and \quad d_1^- \cdot d_1^+ = 0 \tag{7.14}$$

To maximize *Q*, in the compromise DSP we need to minimize the deviation variable d_1^-.

- Minimize the compliance, *J*. Compliance is the measurement of softness as opposed to stiffness of a material. It is the reciprocal of Young's modulus or the inverse of the stiffness matrix. In this example, we use a simulation code to calculate the compliance of the LCA device. Since we want to maximize the stiffness of the device, in this example we minimize the compliance. In the compromise DSP, this goal is formulated as:

$$1 - \frac{J_{\max} - J}{J_{\max} - J_{target}} + d_2^- - d_2^+ = 0 \tag{7.15}$$

where J_{max} is the maximum observed compliance, and J_{target} is the target value for this goal, which we set as J_{target} = 0.0015m/N. To minimize the compliance *J*, we need to minimize the deviation variable d_2^+ in the compromise DSP.

- Minimize the device weight. Since the material is selected as copper and the depth of the device is fixed, this goal is the same as minimizing the cross-section area that is filled with solid materials, A_s, as calculated in Equation (7.3). . In the compromise DSP, this goal is formulated as:

$$1-\frac{A_{s\max}-A_s}{A_{s\max}-A_{s\,target}}+d_3^- - d_3^+ = 0 \qquad (7.16)$$

where A_{smax} is the maximum observed value of A_s, and $A_{starget}$ is the target value for A_s, which we set as $A_{starget}$ = 0.00025m^2. To minimize the compliance J, we need to minimize the deviation variable d_3^+ in the compromise DSP.

Response contour plots are presented below. All plots are drawn with information from 1573 points evenly spread over in the whole design space. The contour plots of Q (total heat transfer rate) versus t (wall thickness) & W (device width), t & $Mdot$ (mass flow rate), and W & $Mdot$ are illustrated in Figure 7.8, Figure 7.9, and Figure 7.10, respectively. The contour plots of J (compliance) versus t and W is illustrated in Figure 7.11. The contour plots of the cross-section area of solid materials (A_s) versus t and W is illustrated in Figure 7.12. From the plots we see that these responses are not highly nonlinear or highly irregular.

LCA heat exchangers with desirable structural and thermal properties are designed for the boundary conditions summarized in Table 7.1. Design is guided with the use of the compromise DSP in Figure 7.13. *Given* a set of boundary conditions and techniques for analyzing non-periodic LCA heat exchangers, the objective is to find the values of the set of design variables that satisfy the set of constraints and bounds and achieve the targets for one or more goals as closely as possible. After formulation of the compromise DSP for the LCA design problem, design solutions can be achieved using the design automation and exploration software of iSIGHT®. When the computation is not

expensive to run we may link the simulation code to iSIGHT to obtain the actual solution. This actual design solution is listed in Table 7.2.

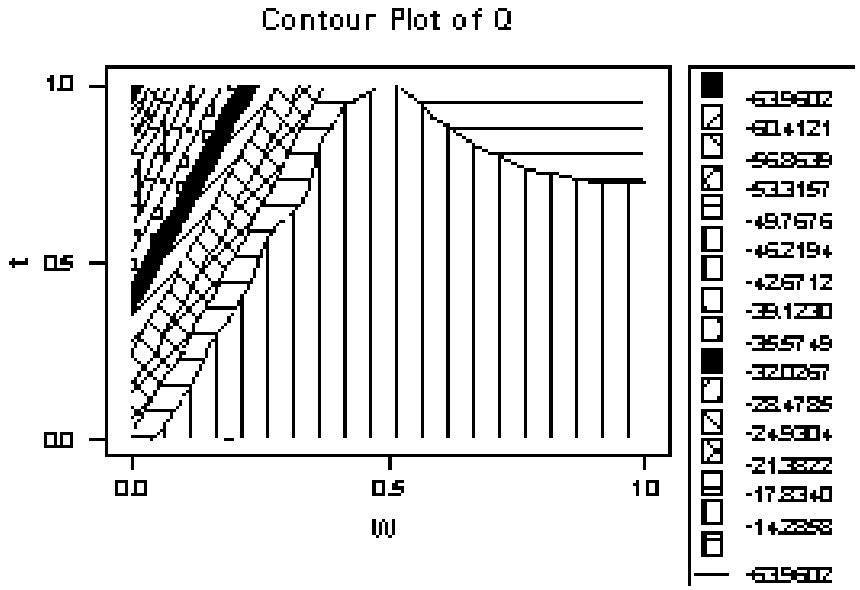


Figure 7.8 Contour Plot of Heat Transfer Rate (*Q*) vs. Wall Thickness (*t*) and Device Width (*W*)

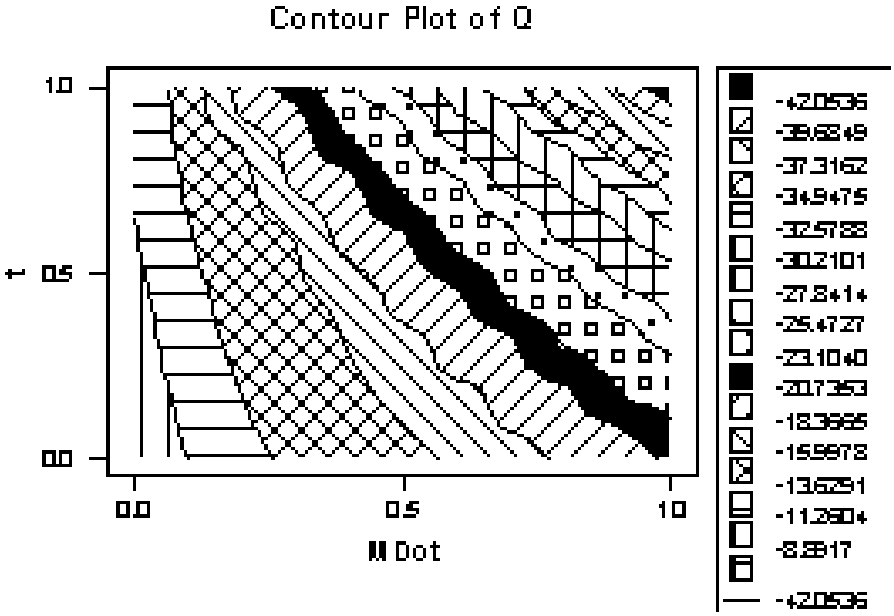


Figure 7.9 Contour Plot of Heat Transfer Rate (*Q*) vs. Wall Thickness (*t*) and Mass Flow Rate (*MDot*)

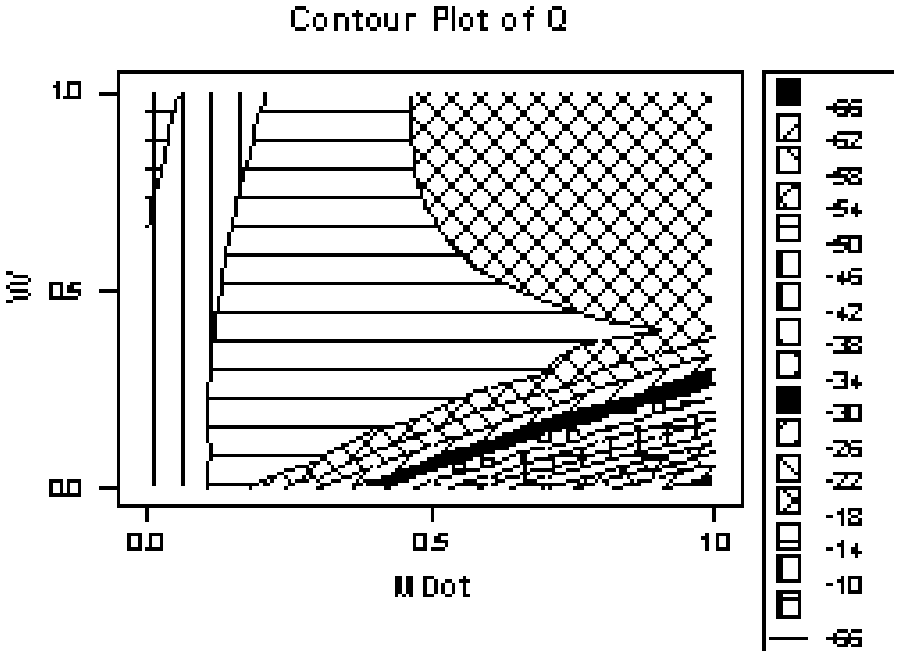


Figure 7.10 Contour Plot of Heat Transfer Rate (*Q*) vs. Device Width (*W*) and Mass Flow Rate (*MDot*)

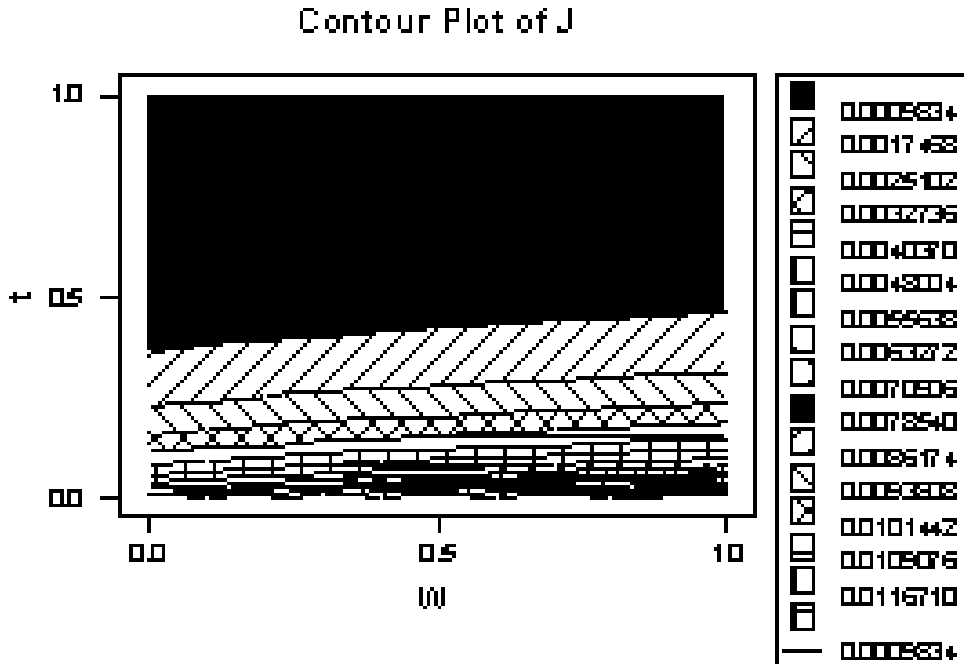


Figure 7.11 Contour Plot of Compliance (*J*) vs. Wall Thickness (*t*) and Width (*W*)

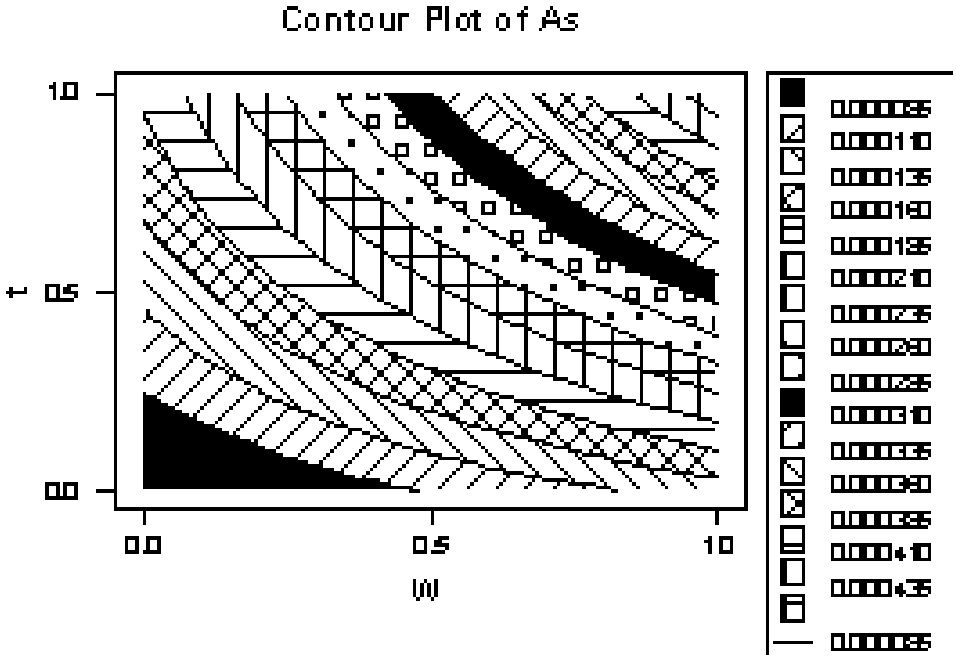


Figure 7.12 Contour Plot of Cross-Section Area of Solid Materials (A_s) vs. Wall Thickness (t) and Device Width (W)

Table 7.1 Boundary Conditions for Design

Structure Width (W), Height (H)	$W = H$
Structure Depth (D)	0.075m
Heat Source Temperature (T_{source})	373K
Fluid Inlet Temperature (T_{in})	293K
Working Fluid	Air
Working Fluid Density	1.205 kg/m^3
LCA Structure	64 square cells
Wall Thickness (t)	Variable, $t_v = t_h$
Thermal Conductivity of Solid Materials (k_s)	363 W/mK
Fluid Velocity (V)	Variable, tied to mass flow rate and pressure drop

Given

Control Factors and Noise Factors:

Three control factors.

Models:

Simulations for thermal and structural analyses.

Assumption:

Steady-state uncompressible laminar flow of air.
The average temperature of air in the device equals to the inlet temperature.
Air density is a fixed value of 1.205 kg/m^3.
All walls have same thickness.
All square cells have identical sizes.

Find

System Variables:

- Mass flow rate ($\dot{M}$) kg/s
- Device Width (W) m
- Wall thickness (t) m

Deviation Variables:

The under and over achievement of the goal of maximizing total heat transfer rate: d_1^-, d_1^+.
The under and over achievement of the goal of minimizing compliance: d_2^-, d_2^+.
The under and over achievement of the goal of minimizing device weight: d_3^-, d_3^+.

Satisfy

System Constraints:

Laminar flow: $R_e < 2300$

Fan curve: $\Delta P \le 30 - 2663.35\dot{M}$

Volume fraction: $vf = \frac{A_s}{A} = \frac{A - A_f}{A} = 1 - \frac{(W - N \cdot t)^2}{W^2} \le 30\%$

System Goals:

System Performance:

Maximize heat transfer rate Q:

$$\frac{Q - Q_{\min}}{Q_{target} - Q_{\min}} - 1 + d_1^- + d_1^+ = 0$$

Minimize compliance J:

$$1 - \frac{J_{\max} - J}{J_{\max} - J_{target}} + d_2^- - d_2^+ = 0$$

Minimize weight:

$$1 - \frac{A_{s\max} - A_s}{A_{s\max} - A_{s target}} + d_3^- - d_3^+ = 0$$

Variable Bounds:

$0.0005 \le \dot{M} \le 0.003$ kg/s
$0.015 \le W \le 0.035$ m
$0.0002 \le t \le 0.0008$ m
$d_i^-, d_i^+ \ge 0$; $d_i^- \cdot d_i^+ = 0$.

Minimize

Deviation Function:

$Z = w_1 \cdot d_1^- + w_2 \cdot d_2^+ + w_3 \cdot d_3^+$. $w_1 = w_2 = w_3 = 1$.

Figure 7.13 Compromise DSP for LCA Unit Design

Table 7.2 Actual Design Solution Obtained with Simulation Codes

	Values
Mass flow rate, *Mdot* (kg/s)	0.00129
Device width, *W* (m)	0.0348
Wall thickness, *t* (m)	0.00042
Mdot_normalized	0.316
W_normalized	0.99
t_normalized	0.3667
Reynolds number, R_e	2297.61
Volume fraction, *vf*	0.2054
$30 - 2663.35\dot{M} - \Delta P$	26.5141
Area of solid materials, A_s (m^2)	0.000249
Heat transfer rate, *Q* (W)	-15.59
Compliance, *J* (m/N)	0.00139
$Z = d_1^- + d_2^+ + d_3^+$	0.31489

7.3 EXPLORATION OF DESIGN SOLUTIONS WITH RCEM

In this section, the compromise DSP in Figure 7.13 is solved with the Robust Concept Exploration Method (RCEM). Since the problem has been defined in Section 7.2, our first step in this section is to design experiments and develop metamodels for responses.

There are three design goals and three constraints in the compromise DSP. The cross-section area of solid materials (A_s), the volume fraction (*vf*), the Reynolds number (R_e), and the pressure drop (ΔP) are easy to get with simple equations, thus we will not develop metamodel for them. The total heat transfer rate (Q) and compliance (J) are obtained from the finite element simulation, and need to be modeled.

There are three design variables as stated in Section 7.2. In this example, we do not perform any screening experimental design to identify unimportant design variables.

Thus a single-stage experimental design is needed to select data points. It is recommended in (iSIGHT, 2003) that to ensure the achievement of acceptable metamodels at least 3*n* data points should be used in cases with *n* design variables. Thus in this example we use a Latin Hypercube design with 30 data points; values of design variables at these points are normalized to [0,1] and listed in Table D.1 in Appendix D.1.1. Total heat transfer rate and compliance are observed by running simulations at these points. Note that the total heat transfer rate is negative because the heat is transferred from the device to the air; in the compromise DSP, we multiply these values with –1 so that we maximize positive values for *Q*.

With information from Table D.1 in Appendix D.1.1, two kriging metamodels are developed for *Q* and *J*, respectively. Values of θ for these kriging metamodels are listed in Table 7.3. In Table 7.3, θ_1 corresponds to the design variable of mass flow rate ($\dot{M}$ or *Mdot*), θ_2 corresponds to the device width (*W*), and θ_3 corresponds to the wall thickness (*t*). In this chapter, if not particularly pointed out, we always use these denotations. Contour plots of *Q* and *J* calculated from kriging metamodels versus *t*, *W*, and *Mdot* are illustrated in Figure 7.14,Figure 7.15, Figure 7.16, and Figure 7.17, respectively. Comparing plots with those in Figure 7.8, Figure 7.9, Figure 7.10, and Figure 7.11, we see that the kriging metamodel for compliance, *J*, is acceptable, while that for total heat transfer rate, *Q*, does not capture the actual responses very well. MARS metamodels are also developed and the model files, qmars.dat, are presented in Appendix D.1.2. The kriging metamodels are more accurate then the MARS metamodels in this example.

Without comparison to actual responses, we decide to use kriging metamodels in solving the compromise DSP because it gives more reasonable predictions (predicted response ranges from MARS are too large compared to what we observed with 30 data points).

Table 7.3 Values of θ for Kriging Metamodels of Q and J

	θ_1	θ_2	θ_3
Q	2.75193	9.41537	8.58675
J	0.01004	0.00627	10.87170

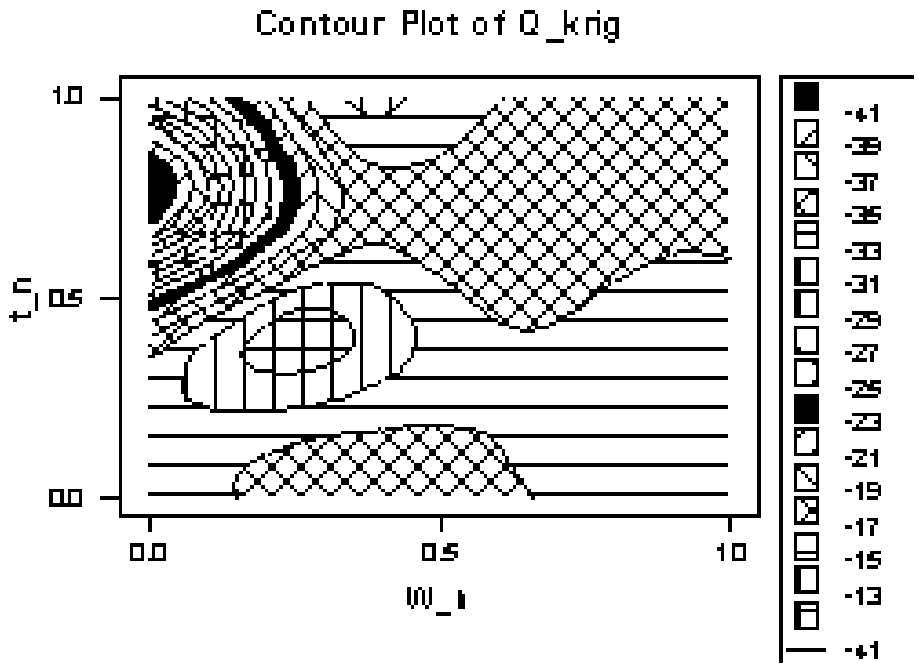


Figure 7.14 Kriging Metamodel of Total Heat Transfer Rate Q with 30 Data Points

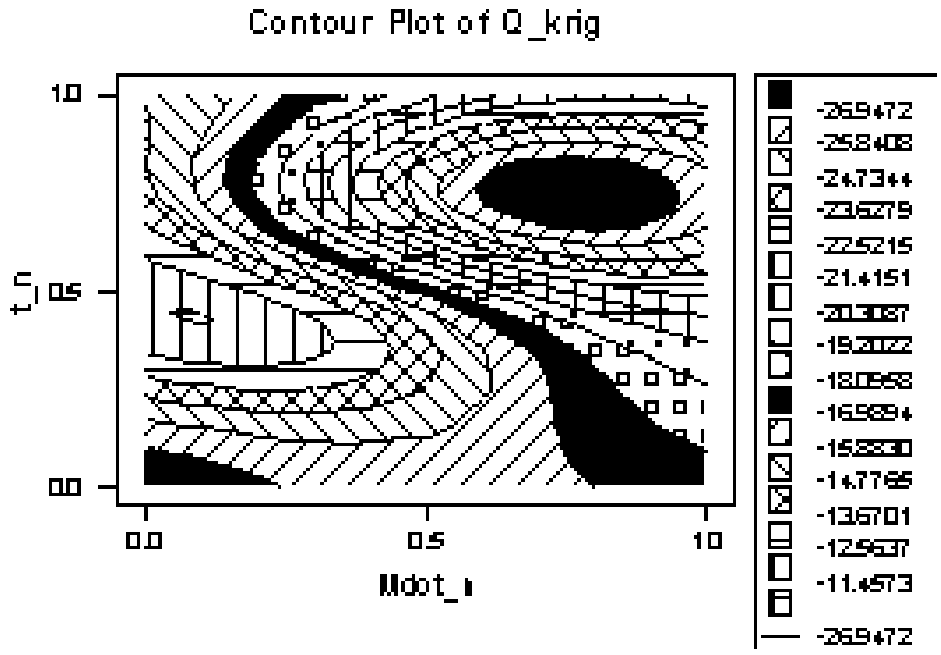


Figure 7.15 Kriging Metamodel of Total Heat Transfer Rate *Q* with 30 Data Points

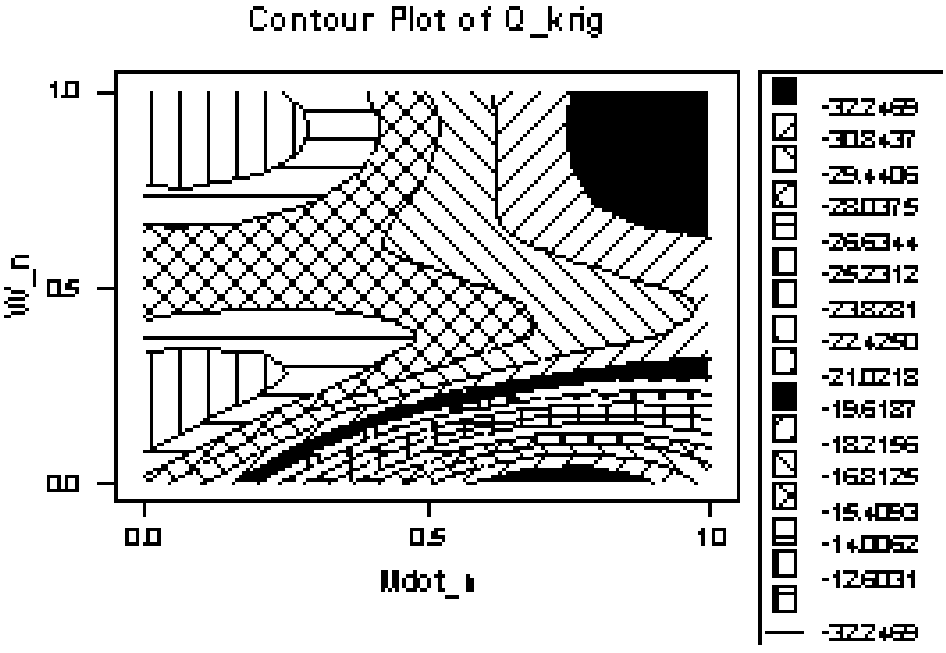


Figure 7.16 Kriging Metamodel of Total Heat Transfer Rate *Q* with 30 Data Points

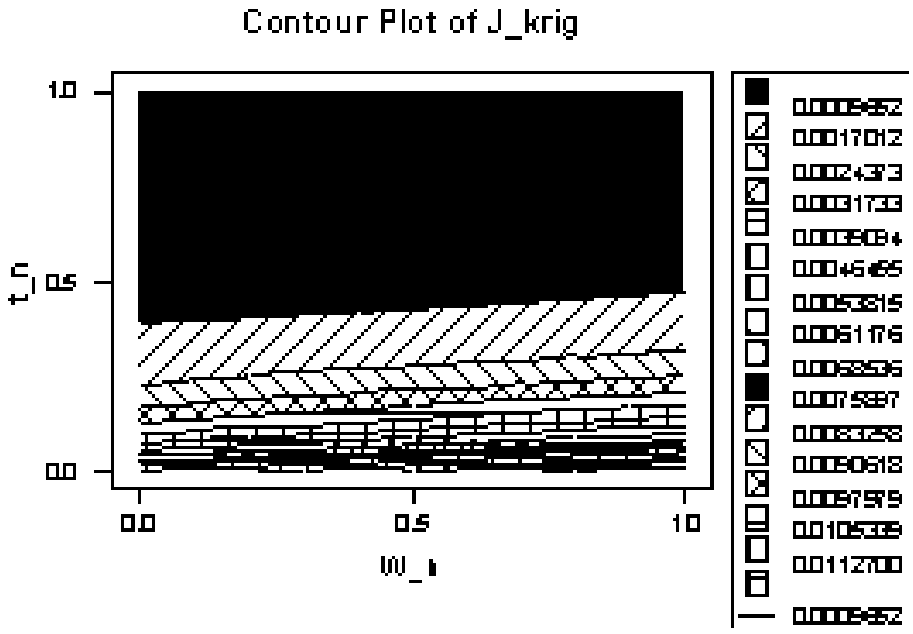


Figure 7.17 Kriging Metamodel of Compliance *J* with 30 Data Points

After developing the metamodels for responses, design solutions can be achieved using the design automation and exploration software of iSIGHT®. We link the metamodels directly in iSIGHT® to explore for solutions for the compromise DSP in Figure 7.13. The implementation of C-DSP in iSIGHT® is illustrated in Appendix D.1.3. The solution is presented in Table 7.4. In Table 7.4 we see that the solution obtained with RCEM and single-stage metamodeling has small values of *Mdot*, and medium value of *W* and *t*. Constraint I ($R_e < 2300$) is active. Both Design Goals II (minimizing compliance) and III (minimizing area) are met. The predicted objective function value at this solution is 0.35357, while the actual value is 0.37351. The solution obtained with 30 data points from Latin Hypercube design is far from the actual solution listed in Table 7.2.

Table 7.4 The Design Solution Obtained with RCEM – 30 LH Experiments

	Predicted Value	Actual Value
Mass flow rate, *Mdot* (kg/s)	0.00097	
Device width, *W* (m)	0.0278	
Wall thickness, *t* (m)	0.00051	
Mdot_normalized	0.1875	
W_normalized	0.6406	
t_normalized	0.5708	
Reynolds number, R_e	2300	
Volume fraction, *vf*	0.29995	
$30 - 2663.35\dot{M} - \Delta P$	27.30	
Area of solid materials, A_s (m^2)	0.00023	
Heat transfer rate, *Q* (W)	−15.05	−14.77
Compliance, *J* (m/N)	0.00078	0.00080
$Z = d_1^- + d_2^+ + d_3^+$	0.35357	0.37351

In order to have more comparisons, a Latin Hypercube design with 40 data points is also used in our study. This experimental design and corresponding response values are listed in Table D.2 in Appendix D.1.4. Two kriging metamodels of responses for *Q* and *J* are developed with this information; values of θ for these kriging metamodels are listed in Table 7.5. The design solution obtained with these metamodels is listed in Table 7.6. From Table 7.6 we see that though information is collected from more points than that in Table 7.4, the solution becomes further from the actual one. One possible reason is that in this example the metamodels developed with 30 data points are more accurate than those developed with 40 data points.

Table 7.5 Values of θ for Kriging Metamodels of *Q* and *J*

	θ_1	θ_2	θ_3
Q	5.71612	10.49253	7.49590
J	0.00238	0.00614	10.31701

Table 7.6 The Design Solution Obtained with RCEM – 40 LH Experiments

	Predicted Value	Actual Value
Mass flow rate, *Mdot* (kg/s)	0.0005	
Device width, *W* (m)	0.0201	
Wall thickness, *t* (m)	0.00036	
$Mdot\_{normalized}$	0.0	
$W\_{normalized}$	0.2532	
$t\_{normalized}$	0.2707	
Reynolds number, R_e	1644.18	
Volume fraction, *vf*	0.29872	
$30 - 2663.35\dot{M} - \Delta P$	28.44	
Area of solid materials, A_s (m^2)	0.00012	
Heat transfer rate, *Q* (W)	−20.05	−11.21
Compliance, *J* (m/N)	0.00165	0.00167
$Z = d_1^- + d_2^+ + d_3^+$	0.01429	0.64419

Table 7.7 Root Mean Squared Errors of Metamodels Developed in RCEM

	Metamodels with LH 30 Points		Metamodels with LH 40 Points	
	Q	***J***	***Q***	***J***
RMSE	9.2047	0.0003433	8.3527	0.000175
NRMSE	8.94%	3.00%	8.11%	1.53%

To compare the accuracy of the metamodels, root mean squared errors (RMSE) are calculated with Equation (2.34) and listed in Table 7.7. The normalized root mean squared errors (NRMSE) are calculated by dividing RMSE with the observed response range; it gives the impression of how large the RMSE is compared with possible response changes. All the information is calculated with information from 1573 evenly spread points in the whole design space. The smaller RMSE (or NRMSE) is, the more accurate the corresponding metamodel. Typically when NRMSE is smaller than 10% we consider the metamodel to be acceptable (Simpson, 1998). In Table 7.7 we see that metamodels developed with 40 data points are more accurate than those developed with 30 data

points, which is opposite to what we have expected in the analysis in the past paragraph. Thus the question here is that: *why we cannot obtain better design solutions with more accurate metamodels?* This question will be answered in the discussion in Section 7.6, after we have applied and studied SEED and E-RCEM in Sections 7.4 and 7.5.

7.4 EXPLORATION OF DESIGN SOLUTIONS WITH SEED IN RCEM

In this section, the SEED method is used to facilitate sequential identification of data points and develop accurate metamodels for design space exploration. After defining the design problem in Section 7.2, here we design the sequential experiments following the methods and steps described in Chapters 4 and 5. We plan to start with 8 data points and 8 validation points, then add in 3 data points or validation points each time. We will stop this sequential experimental design process once 28 points (which is fewer than that used in RCEM in Section 7.3) are observed, i.e., in Iteration III – Step 3. We expect to develop more accurate metamodels and also achieve better design solutions with fewer observed points using the SEED method.

Iteration I – Step 1: Initial Experimental Design. Eight data points are identified at the "corners" of the hypercube, as listed in Table 7.8. This is actually a full factorial experimental design.

Iteration I – Step 2: Simulation and Initial Metamodels of Responses. We run the simulation codes and get response values of Q and J at eight data points. Kriging metamodels are developed with this information, and values of θ are listed in Table 7.9. Since at this very early stage of metamodeling, it is unlikely that kriging may behave

abnormally (as discussed in Chapter 5), we decide not to develop MARS metamodels and use kriging as the initial metamodels for responses. The contour plot of *Q* versus *t* and *Mdot* is illustrated in Figure 7.18; more contour plots are presented in Appendix D.2.1.

Table 7.8 Initial Experimental Design with 8 Data Points

Mdot	*W*	*t*	*Mdot_n*	*W_n*	*t_n*	*Q*	*J*
0.0005	0.015	0.0002	0	0	0	-11.01	0.00749
0.0005	0.015	0.0008	0	0	1	-14.37	0.00022
0.0005	0.035	0.0002	0	1	0	-6.65	0.01167
0.0005	0.035	0.0008	0	1	1	-9.56	0.00027
0.003	0.015	0.0002	1	0	0	-42.24	0.00749
0.003	0.015	0.0008	1	0	1	-109.66	0.00022
0.003	0.035	0.0002	1	1	0	-19.86	0.01167
0.003	0.035	0.0008	1	1	1	-23.03	0.00027

Table 7.9 Values of θ for the Initial Kriging Metamodels

	θ_1	θ_2	θ_3
Q	78.19556	3.11212	0.80947
J	0.00100	0.20111	71.32491

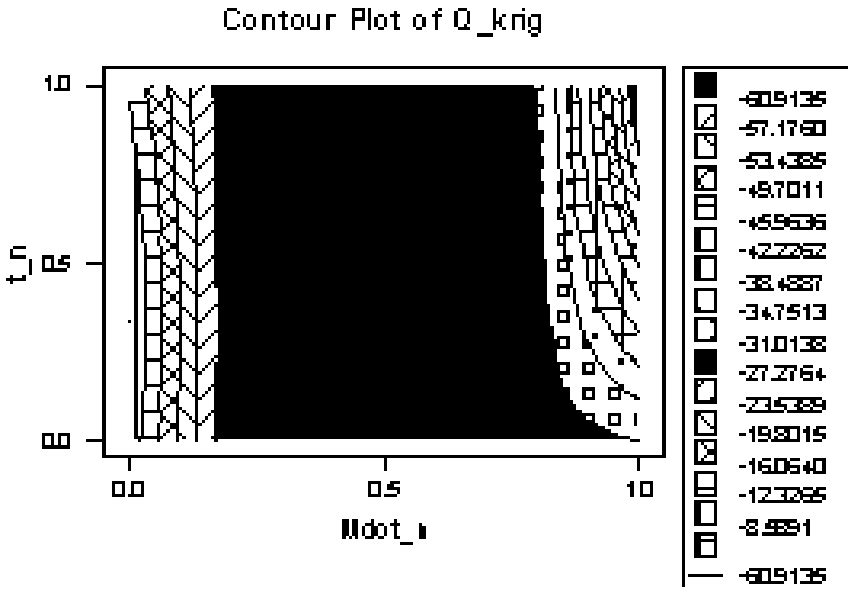


Figure 7.18 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Initial Kriging Metamodel with 8 Data Points)

Iteration I – Step 3: Identification of New Validation Points. In this step we need to identify 8 validation points. Since no information of prediction errors is known in this iteration, we do not adjust the covariance matrix and identify new validation points to spread over the whole design space. A 16×16 covariance matrix is build with the first 8 rows and columns corresponding to 8 data points that we have, and the last 8 rows and columns corresponding to the 8 validation points that we need to identify. In the formulation of this covariance matrix, we set $\theta_1 = 78.19556$, $\theta_2 = 3.11212$, $\theta_3 = 71.32491$. For each design variable we use the larger one (in columns) in Table 7.9. By maximizing the determinant of the covariance matrix, we identify 8 validation points for the first iteration as listed in Table 7.10.

Table 7.10 Eight New Validation Points Identified in Iteration I

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00222	0.0235	0.00045	0.6865	0.4227	0.4100	-17.64	0.00098
0.00125	0.0350	0.00035	0.3008	1.0	0.2559	-14.85	0.00232
0.00239	0.0150	0.00065	0.7573	0.0	0.7573	-70.69	0.00034
0.00300	0.0250	0.00050	1.0	0.5	0.5	-18.94	0.00076
0.00053	0.0203	0.00041	0.0111	0.2663	0.3472	-11.79	0.00118
0.00175	0.0250	0.00080	0.5	0.5	1.0	-18.53	0.00025
0.00175	0.0250	0.00020	0.5	0.5	0.0	-15.92	0.00990
0.00146	0.0341	0.00057	0.3834	0.9532	0.6156	-17.53	0.00061

Iteration I – Step 4: Metamodel of Prediction Errors. Prediction errors of the initial kriging metamodels (Table 7.9) at the validation points are listed in Table 7.11. Prediction errors at data points are zero. Two kriging metamodels of prediction errors are

then developed for heat transfer rate and compliance. The values of θ are listed in Table 7.12. The maximum absolute prediction error is about 50 for *Q*, and 0.0078 for *J*.

Table 7.11 Prediction Errors at 8 Validation Points

Mdot_n	*W_n*	*t_n*	*Q*	*J*	*Q_pred*	*J_pred*	*Q_err*	*J_err*
0.6865	0.4227	0.4100	-17.64	0.00098	-20.11	0.00491	-2.47	0.00393
0.3008	1.0	0.2559	-14.85	0.00232	-20.39	0.00491	-5.54	0.00259
0.7573	0.0	0.7573	-70.69	0.00034	-20.73	0.00491	49.96	0.00457
1.0	0.5	0.5	-18.94	0.00076	-21.13	0.00491	-2.19	0.00415
0.0111	0.2663	0.3472	-11.79	0.00118	-21.56	0.00486	-9.77	0.00368
0.5	0.5	1.0	-18.53	0.00025	-22.03	0.00427	-3.5	0.00402
0.5	0.5	0.0	-15.92	0.00990	-22.53	0.00208	-6.61	-0.0078
0.3834	0.9532	0.6156	-17.53	0.00061	-23.03	0.00027	-5.5	-0.0003

Table 7.12 Values of θ for Kriging Metamodels of Prediction Errors in Iteration I

	θ_1	θ_2	θ_3
Q_err	31.38770	14.31849	0.00395
J_err	7.72797	0.00100	32.19186

Iteration I – Step 5: Metamodel Validation. This step is skipped.

Iteration I – Step 6: Formulation of the Adjusted Covariance Matrix. We need to add in 3 data points. An 11×11 covariance matrix is formulated, with the first 8 rows and columns corresponding to current data points, and the last 3 rows and columns corresponding to new data points. Then the prediction errors calculated from metamodels developed in Iteration I – Step 4 are used to calculate correcting coefficients following Equation (5.9). In the formulation of this covariance matrix, we set θ_1 = 78.19556, θ_2 = 14.31849, θ_3 = 71.32491. The two responses, *Q* and *J*, are considered to be equally important, i.e., $\rho_Q = \rho_J = 0.5$ in Equation (5.9). The value of λ is 2.

Iteration I – Step 7: Identification of New Data Points. In this step, by maximizing the determinant of the adjusted covariance matrix as developed in the previous step, 3 possible new data points are identified and listed in Table 7.13.

Table 7.13 Four New Data Points Identified in Iteration I

Mdot	*W*	*t*	*Mdot_n*	*W_n*	*t_n*	*Q*	*J*
0.00175	0.0250	0.0005	0.5	0.5	0.5	-17.49	0.00076
0.00058	0.0321	0.00043	0.0333	0.8556	0.3769	-9.58	0.00126
0.00204	0.0237	0.00027	0.6143	0.4333	0.1167	-16.69	0.00405

Iteration I – Step 8: Updated Metamodels of Responses. Now we have 11 data points and 8 validation points. Two new kriging metamodels of responses are developed with information from the data points. The values of θ are listed in Table 7.14. Contour plots of responses are presented in Appendix D.2.1.

Table 7.14 Values of θ for Kriging Metamodels of Responses with 12 Data Points

	θ_1	θ_2	θ_3
Q	1.76888	1.16104	0.48836
J	0.00100	0.24056	84.62898

Iteration II – Step 3: Identification of New Validation Points. In this step, we need to add in 3 new validation points. Two kriging metamodels are developed for *Q* and *J* based on information from 8 validation points. The values of θ are listed in Table 7.15. Prediction errors of these metamodels at data points are listed in Table 7.16.

Kriging metamodels of prediction errors are developed with information at 19 points. The values of θ are listed in Table 7.17. The maximum absolute prediction error is about 60 for Q, and 0.003 for J. To identify 3 new validation points, a 22×22 covariance matrix is formulated with the first 19 rows and columns corresponding to observed points and the last 3 rows and columns corresponding to new validation points. In the formulation of this covariance matrix, we set θ_1 = 0.83202, θ_2 = 23.19746, θ_3 = 25.33245. Then prediction errors calculated from metamodels in Table 7.17 are used to adjust entries of the covariance matrix. By maximizing the determinant of this adjusted covariance matrix, 3 new validation points are identified and listed in Table 7.18.

Table 7.15 Values of θ for Kriging Metamodels of Responses with 8 Validation Points

	θ_1	θ_2	θ_3
Q	0.01269	23.19746	0.00692
J	0.00103	0.00100	6.24017

Table 7.16 Prediction Errors at 11 Data Points

Mdot_n	***W_n***	***t_n***	***Q***	***J***	***Q_pred***	***J_pred***	***Q_err***	***J_err***
0	0	0	-11.01	0.00749	-70.07	0.00981	-59.06	0.00232
0	0	1	-14.37	0.00022	-70.27	0.00025	-55.90	0.00003
0	1	0	-6.65	0.01167	-14.83	0.00985	-8.18	-0.00182
0	1	1	-9.56	0.00027	-14.99	0.0003	-5.43	0.00003
1	0	0	-42.24	0.00749	-70.50	0.00995	-28.26	0.00246
1	0	1	-109.66	0.00022	-70.69	0.0002	38.97	-0.00002
1	1	0	-19.86	0.01167	-14.95	0.00999	4.91	-0.00168
1	1	1	-23.03	0.00027	-15.11	0.00025	7.92	-0.00002
0.5	0.5	0.5	-17.49	0.00076	-17.21	0.00073	0.28	-0.00003
0.0333	0.8556	0.3769	-9.58	0.00126	-23.48	0.00109	-13.90	-0.00017
0.6143	0.4333	0.1167	-16.69	0.00405	-16.96	0.00583	-0.27	0.00178

Table 7.17 Values of θ for Kriging Metamodels of Prediction Errors in Iteration II – Step 3

	θ_1	θ_2	θ_3
Q	0.83202	7.31764	0.32241
J	0.00160	0.63128	25.33245

Table 7.18 Three New Validation Points Identified in Iteration II

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.003	0.0294	0.00031	0.9998	0.7204	0.1767	-19.52	0.00314
0.00053	0.0290	0.00067	0.0123	0.7001	0.7850	-11.22	0.00039
0.0005	0.0210	0.00065	0.0015	0.2976	0.7563	-12.91	0.00037

Table 7.19 Prediction Errors at 11 Validation Points

Mdot_n	***W_n***	***t_n***	***Q***	***J***	***Q_pred***	***J_pred***	***Q_err***	***J_err***
0.6865	0.4227	0.41	-17.64	0.00098	-30.54	0.00097	-12.90	-0.00001
0.3008	1	0.2559	-14.85	0.00232	-3.31	0.00252	11.54	0.00020
0.7573	0	0.7573	-70.69	0.00034	-77.40	0.00306	-6.71	0.00272
1	0.5	0.5	-18.94	0.00076	-50.45	0.00076	-31.51	0.00000
0.0111	0.2663	0.3472	-11.79	0.00118	-11.70	0.00170	0.09	0.00052
0.5	0.5	1	-18.53	0.00025	-27.21	0.00009	-8.68	-0.00016
0.5	0.5	0	-15.92	0.0099	-9.02	0.00993	6.90	0.00003
0.3834	0.9532	0.6156	-17.53	0.00061	-3.76	0.00247	13.77	0.00186
0.9998	0.7204	0.1767	-19.52	0.00314	-27.53	0.00274	-8.01	-0.00040
0.0123	0.7001	0.785	-11.22	0.00039	-11.72	0.00303	-0.50	0.00264
0.0015	0.2976	0.7563	-12.91	0.00037	-12.74	0.00306	0.17	0.00269

Iteration II – Step 4: Metamodels of Prediction Errors. The prediction errors of metamodels in Iteration I – Step 8 at 11 validation points are calculated and listed in Table 7.19. Prediction errors at 11 data points are zero. Two kriging metamodels of prediction errors are developed with this information and the values of θ are listed in

Table 7.20. The observed maximum absolute prediction error for Q is around 30, and that for J is around 0.003.

Table 7.20 Values of θ for Kriging Metamodels of Prediction Errors in Iteration II – Step 4

	θ_1	θ_2	θ_3
Q	23.92013	0.00100	7.91005
J	0.00100	0.24284	72.55745

Iteration II – Step 5: Metamodel Validation. This step is skipped.

Iteration II – Step 6: Formulation of the Adjusted Covariance Matrix. We need to identify 3 new data points in this iteration. The adjusted covariance matrix is formulated with the same method as described in Iteration I – Step 6. In the formulation of this covariance matrix, we set θ_1 = 23.92013, θ_2 = 1.16104, θ_3 = 84.62898. The two responses, Q and J, are considered to be equally important, i.e., $\rho_Q = \rho_J = 0.5$ in Equation (5.9). The value of λ is 2.

Iteration II – Step 7: Identification of New Data Points. By maximizing the determinant of the adjusted covariance matrix formulated in Step 6, three new data points are identified and listed in Table 7.21.

Table 7.21 Four New Data Points Identified in Iteration II

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00082	0.0157	0.00064	0.1276	0.0344	0.7252	-14.65	0.00036
0.00298	0.0245	0.00044	0.9925	0.4751	0.3961	-18.58	0.00106
0.00182	0.0321	0.00074	0.5290	0.8567	0.9059	-19.76	0.00031

Iteration II – Step 8: Updated Metamodels of Responses. Since we will stop the SEED process after identifying 2 more new points in Iteration III – Step 3, and then the final metamodels of responses will be developed with information from all observed points, we do not need to update metamodels of responses in this step.

Iteration III – Step 3: Identification of New Validation Points. In this step, we need to add in 3 new validation points. Two kriging metamodels are developed for Q and J based on information from 11 validation points. The values of θ are listed in Table 7.22. Prediction errors of these metamodels at data points are listed in Table 7.23. Prediction errors of these metamodels at validation points are zero.

Kriging metamodels of prediction errors are developed with information at 25 points. The values of θ are listed in Table 7.24. The observed maximum absolute prediction error is about 60 for Q, and 0.0022 for J. To identify 3 new validation points, a 28×28 covariance matrix is formulated with the first 25 rows and columns corresponding to observed points and the last 3 rows and columns corresponding to new validation points. In the formulation of this covariance matrix, we set $\theta_1 = 0.46716$, $\theta_2 = 11.95818$, $\theta_3 = 17.40336$. Then prediction errors calculated from metamodels in Table 7.24 are used to adjust entries of the covariance matrix. By maximizing the determinant of this adjusted covariance matrix, 3 new validation points are identified and listed in Table 7.25.

Table 7.22 Values of θ for Kriging Metamodels of Responses with 12 Validation Points

	θ_1	θ_2	θ_3
Q	0.01937	11.95818	0.00382
J	0.00100	0.00426	5.67512

Table 7.23 Prediction Errors at 14 Data Points in Iteration III – Step 3

Mdot_n	***W_n***	***t_n***	***Q***	***J***	***Q_pred***	***J_pred***	***Q_err***	***J_err***
0	0	0	-11.01	0.00749	-68.51	0.00977	-57.50	0.00228
0	0	1	-14.37	0.00022	-69.33	0.00031	-54.96	0.00009
0	1	0	-6.65	0.01167	-13.87	0.01032	-7.22	-0.00135
0	1	1	-9.56	0.00027	-13.79	0.00023	-4.23	-0.00004
1	0	0	-42.24	0.00749	-70.36	0.00947	-28.12	0.00198
1	0	1	-109.66	0.00022	-71.16	0.00028	38.50	0.00006
1	1	0	-19.86	0.01167	-17.45	0.01002	2.41	-0.00165
1	1	1	-23.03	0.00027	-17.37	0.00021	5.66	-0.00006
0.5	0.5	0.5	-17.49	0.00076	-17.21	0.00083	0.28	0.00007
0.0333	0.8556	0.3769	-9.58	0.00126	-17.45	0.00155	-7.87	0.00029
0.6143	0.4333	0.1167	-16.69	0.00405	-16.86	0.00478	-0.17	0.00073
0.1276	0.0344	0.7252	-14.65	0.00036	-60.53	0.00033	-45.88	-0.00003
0.9925	0.4751	0.3961	-18.58	0.00106	-18.88	0.00098	-0.30	-0.00008
0.529	0.8567	0.9059	-19.76	0.00031	-20.75	0.00037	-0.99	0.00006

Table 7.24 Values of θ for Kriging Metamodels of Prediction Errors in Iteration III – Step 3

	θ_1	θ_2	θ_3
Q	0.46716	3.81693	0.20295
J	0.00880	0.53704	17.40336

Table 7.25 Three New Validation Points Identified in Iteration III

Mdot	*W*	*t*	*Mdot_n*	*W_n*	*t_n*	*Q*	*J*
0.0005	0.035	0.00068	0.0	1.0	0.7935	-8.92	0.0004
0.003	0.035	0.00046	1.0	1.0	0.4309	-21.65	0.00109
0.0005	0.025	0.00062	0.0	0.5	0.7	-11.68	0.00044

Now since we have already obtained 28 points (14 data points and 14 validation points), the SEED process stops in this iteration. Information of responses at the 28 points is listed in Table D.3 in Section D.2.4. Two kriging metamodels are developed for Q and J based on information from these 28 points. The values of θ are listed in Table 7.26. Contour plots of responses calculated with the kriging metamodels are illustrated in Figure 7.19, Figure 7.20, Figure 7.21, and Figure 7.22.

With the kriging metamodels in Table 7.26, we solve the compromise DSP in iSIGHT. The solution obtained in this section is listed in Table 7.27. We see that this solution is closer to the actual solution (in Table 7.2) than those obtained in Section 7.3. Constraint I is active, while other constraints are not. The design goals associated with J and A_f are achieved.

Table 7.26 Values of θ for Kriging Metamodels of Responses Developed with SEED

	θ_1	θ_2	θ_3
Q	1.12370	2.69722	0.39064
J	0.00100	0.09403	13.84511

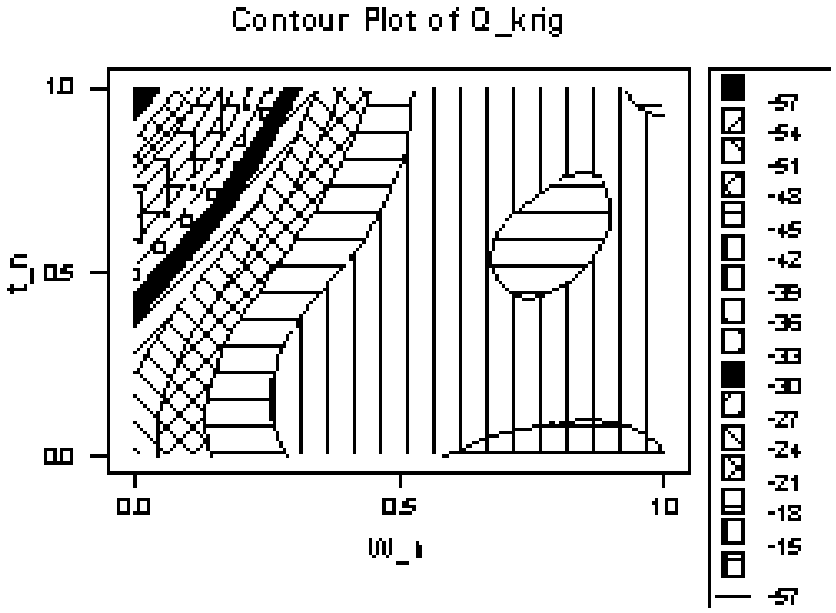


Figure 7.19 Contour Plot of the Kriging Metamodel for Heat Transfer Rate (*Q*) with Respect to Device Width (*W*) and Wall Thickness (*t*) Developed with SEED

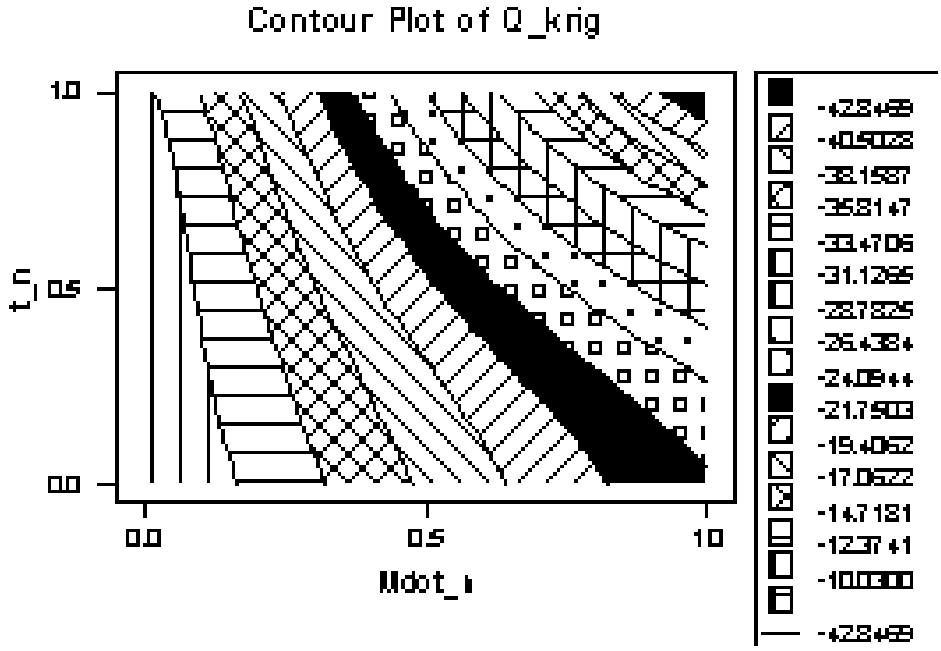


Figure 7.20 Contour Plot of the Kriging Metamodel for Heat Transfer Rate (*Q*) with Respect to Wall Thickness (*t*) and Mass Flow Rate (Mdot) Developed with SEED

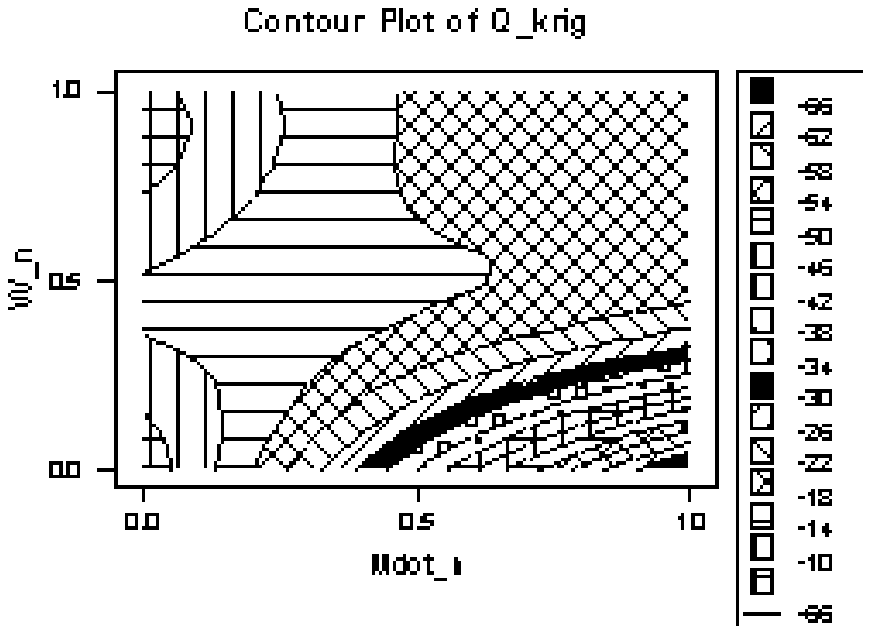


Figure 7.21 Contour Plot of the Kriging Metamodel for Heat Transfer Rate (*Q*) with Respect to Device Width (*W*) and Mass Flow Rate (Mdot) Developed with SEED

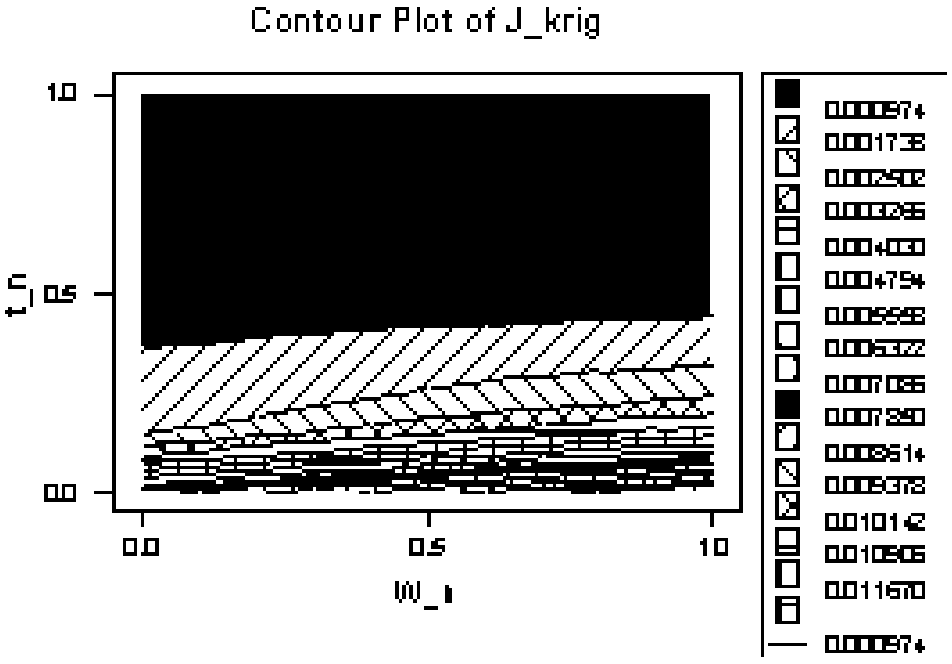


Figure 7.22 Contour Plot of the Kriging Metamodel for Compliance (*J*) with Respect to Device Width (*W*) and Wall Thickness (*t*) Developed with SEED

Table 7.27 The Design Solution Obtained with SEED

	Predicted Value	Actual Value
Mass flow rate, *Mdot* (kg/s)	0.00113	
Device width, *W* (m)	0.0316	
Wall thickness, *t* (m)	0.00048	
Mdot_normalized	0.2535	
W_normalized	0.8284	
t_normalized	0.4694	
Reynolds number, R_e	2300	
Volume fraction, *vf*	0.25578	
$30 - 2663.35\dot{M} - \Delta P$	26.91	
Area of solid materials, A_s (m^2)	0.00025	
Heat transfer rate, *Q* (W)	−16.61	−15.30
Compliance, *J* (m/N)	0.00089	0.00093
$Z = d_1^- + d_2^+ + d_3^+$	0.24180	0.35620

Root mean square error (RMSE) of metamodels are calculated with information from 1573 points, and listed in Table 7.28. Values of RMSE and NRMSE for *Q* in Table 7.28 are much smaller than those in Table 7.7. Values of RMSE and NRMSE for *J* in Table 7.28 are between those of the two metamodels for *J* with 30 or 40 points in Table 7.7. Generally speaking, we are able to develop more accurate metamodels and achieve better design solutions with fewer observed points in the design space. Further discussion and analyses will be done in Section 7.6, after the application of E-RCEM in Section 7.5.

Table 7.28 Root Mean Squared Errors of Metamodels Developed in RCEM

	Metamodels with 28 Points Identified with SEED	
	Q	***J***
RMSE	4.4767	0.0002304
NRMSE	4.35%	2.01%

7.5 EXPLORATION OF DESIGN SOLUTIONS WITH E-RCEM

In Section 7.4, we apply the SEED method in the design of unit cells for linear cellular alloys; in this example we show that more accurate metamodels and better design solutions can be achieved with fewer experiments using the SEED method in the metamodeling process. In cases with very expensive computer simulations or physical experiments, using the SEED method helps save significant amount of time or money, and ensures a better solution as a starting point for design in later design stages. In this section, we will apply the E-RCEM method to realize an integrated process of metamodeling and design space exploration in the LCA design. As shown in Chapter 6, E-RCEM ensures the identification of most-likely-to-succeed regions in the metamodeling process and the development of metamodels with better local accuracy in such critical regions. Uncertainty of global metamodel accuracy is addressed to avoid being misled to wrong directions in the integrated process of metamodeling and design space exploration, but global metamodel accuracy is not pursued or guaranteed. In the integrated design process in E-RCEM, new points are added sequentially in regions with large metamodel uncertainty and/or better achievement of design goals. In this LCA design example, we expect to achieve a solution closer to the true solution identified in Section 7.2 (Table 7.2) with fewer observed points using the integrated design process in E-RCEM than with RCEM (Section 7.3) or SEED (Section 7.4).

As described in Section 6.4.4, there are three possible ways in implementing E-RCEM: the traditional process, the integrated design process, and the hybrid process. In the traditional process, designers develop acceptable metamodels and explore for design

solutions in two separated processes; there are no information feedbacks from the process of design space exploration to metamodeling. The application of RCEM and SEED in Sections 7.3 and 7.4 follows this way. In the integrated design process, prediction errors, achievement of design goals, and satisfaction of design constraints are considered simultaneously in the identification of new points. In the hybrid process, acceptable metamodels are first developed, and then more points are added following the integrated design process. In this section, we adopt the integrated design process in LCA design.

In E-RCEM, the integrated process of metamodeling and design space exploration is realized by introducing the link (information feedback) from the compromise DSP to design of experiments, as illustrated in Figure 6.11. This information feedback includes two types of information, one of which is associated with design goals, and the other associated with design constraints. The consideration of design constraints and identification of points in irregular design spaces are discussed in Section 6.2. From the viewpoint of design space exploration, infeasible regions in the design space are not "critical" and designers should not waste time or money on experiments in these regions. The feasible design space may be much smaller than the original design space (which is usually a hypercube); to identify points in such small design spaces help save experimental expense and achieve better design solutions. However, the feasible design space may not have clear boundaries (when the constraints are associated with responses for which we need to build metamodels), or the boundaries may be difficult to identify and illustrate (when designers have a lot of design constraints in a multi-variable, multi-

response problem). E-RCEM helps address this concern and facilitate more **efficient** designs of experiments in irregular feasible design spaces.

The consideration of design constraints helps identify feasible design spaces, which gives metamodeling a good start because "absolute uncritical" regions are removed. Design goals are then taken into consideration with prediction errors to help identify critical regions in the feasible design space. Critical regions are those in which design goals are achieved or nearly achieved and/or prediction errors are large with current metamodels. By adding more points in critical regions in iterations, designers are able to develop metamodels with better local prediction performance and thus achieve better design solutions than using traditional design methods like RCEM.

In a multi-variable, multi-response, and multi-objective design case, the feasible design space is constructed with boundaries from design variables and design constraints, and the covariance matrix will be adjusted with information from both prediction errors and the achievement of design goals. Based on the research in Chapters 5 and 6, entries of the adjusted covariance matrix should be formulated with the equations below:

$$\sigma_{ij}^{adj} = \sigma^2 \cdot \begin{cases} \alpha_i \alpha_j \eta_i \eta_j R\left(\left|x_i - x_j\right|\right) & when \begin{cases} i \le n,\ j > n \\ i > n,\ j \le n \\ R\left(\left|x_i - x_j\right|\right) \ne 1 \end{cases} \\ R\left(\left|x_i - x_j\right|\right) & when \begin{cases} i \le n,\ j \le n \\ i > n,\ j > n \end{cases} \\ 1 & when \begin{cases} i \le n,\ j > n \\ i > n,\ j \le n \\ R\left(\left|x_i - x_j\right|\right) = 1 \end{cases} \end{cases} \tag{7.17}$$

where α_i and α_j are the coefficient to reflect the current metamodel's uncertainty (prediction errors) at point $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, η_i and η_j are coefficients to reflect degrees of achievement of design goals at points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, respectively. In multi-variable, multi-response, and multi-objective cases, the coefficient α_i (or α_j) is formulated with the following equation:

$$\alpha_i = 1 - relative.uncert = 1 - \sum_{k=1}^{n_r} \left(\frac{\rho_k}{\lambda_k} \left| \frac{e_{i,k}}{e_{\max,k}} \right| \right) \tag{7.18}$$

where *relative.uncert* stands for the measurement of relative uncertainty at the candidate point, n_r is the number of responses for which we need to develop metamodels, $e_{i,k}$ is the predicted prediction error of the metamodel for the k^{th} response at the candidate point, $e_{max,k}$ is the maximum absolute error observed with the k^{th} response (from current observations or from predictions with the metamodel), ρ_k is the weight designers assigned to the k^{th} response in metamodeling, and λ_k is the coefficient to balance "minimizing prediction errors" and "spread over the design space" for the k^{th} response in the identification of new points. Usually we set:

$$\lambda_1 = \lambda_2 = \cdots = \lambda_{n_r} = 2 \tag{7.19}$$

and

$$\rho_1 = \rho_2 = \cdots = \rho_{n_r} = \frac{1}{n_r} \tag{7.20}$$

so that it satisfies that:

$$\sum_{k=1}^{n_r} \rho_k = 1 \tag{7.21}$$

The coefficients, η_i and η_j, are formulated with the following equation:

$$\eta_i = 1 - total.goal.achievement \tag{7.22}$$

where *total.goal.achievement* is the measurement of degrees that the design goals are achieved at the candidate point. In multi-objective cases, *goal.achievement* can be formulated with Equation (7.23):

$$total.goal.achievement = \sum_{k=1}^{n_g} w_k \cdot goal.achievement_k \tag{7.23}$$

where $goal.achievement_k$ is the measurement of degrees that the k^{th} design goal is achieved at the candidate point, n_g is the number of design goals involved, and w_k is the weight assigned to the k^{th} design goal. Usually, the formulation of *total.goal.achievement* should be the same as that of the deviation function, *z*, in the compromise DSP (Figure 7.13). Thus, the formulation of $goal.achievement_k$ follows Equations (6.24), (6.25), and (6.26) in Chapter 6.

In this section, we will follow the integrated design process in E-RCEM as described in Section 6.4.4, starting with 6 data points and 6 validation points, and ending with 20 observed points. Each time we plan to add in 2 new data or validation points, thus the integrated design process will stop in Iteration III – Step 4.

Step 1: Problem Initialization. This step is finished in Section 7.2.

Steps 2 and 3: Initial Experiments, Design Space Reduction, and Design Space Redefinition. In this example we do not screen out unimportant design variables. The initial design space is defined in Section 7.2, Figure 7.13, and is refined in this step by considering design constraints.

In this example, all design constraints are associated with design variables only, thus the design space is fixed and clear. After examination of design boundaries, it can be shown that Constraint II is satisfied at all points in the design space, thus it will not be studied and taken into consideration in this section. Boundaries from Constraints I and III are illustrated in Figure 7.23.

As shown in Figure 7.23, the initial design space is cubic; the surface in red (dark color in black-white printouts) is the boundary calculated from Constraint I; the surface in green (light color in black-white printouts) is that from Constraint III. Note that Figure 7.23 is just an illustration and the boundaries on design variables do not strictly follow those in Figure 7.13, the compromise DSP. Constraints are not satisfied at points below the boundaries contain points. The two design constraints separate the initial design space into four regions, and the one above both boundaries is the feasible design space, as illustrated in Figure 7.23.

It should be noted that design constraints could be easily accounted in E-RCEM without much expense; the analysis in the above paragraphs is for illustration only. The initial experiments are designed with the maximum entropy sampling method. All 6 data points are constrained in the feasible design space; the points and corresponding response values are illustrated in Table 7.29. Based on this information, initial kriging metamodels

are developed for the two responses, Q and J. Values of θ for the kriging metamodels are listed in Table 7.30. The contour plot of the metamodel for total heat transfer rate (Q) versus wall thickness (t) and mass flow rate (*Mdot*) is illustrated in Figure 7.24. More contour plots are presented in Appendix D.3. Comparing Figure 7.24 with Figure 7.9 we see that the initial metamodel is not accurate at all.

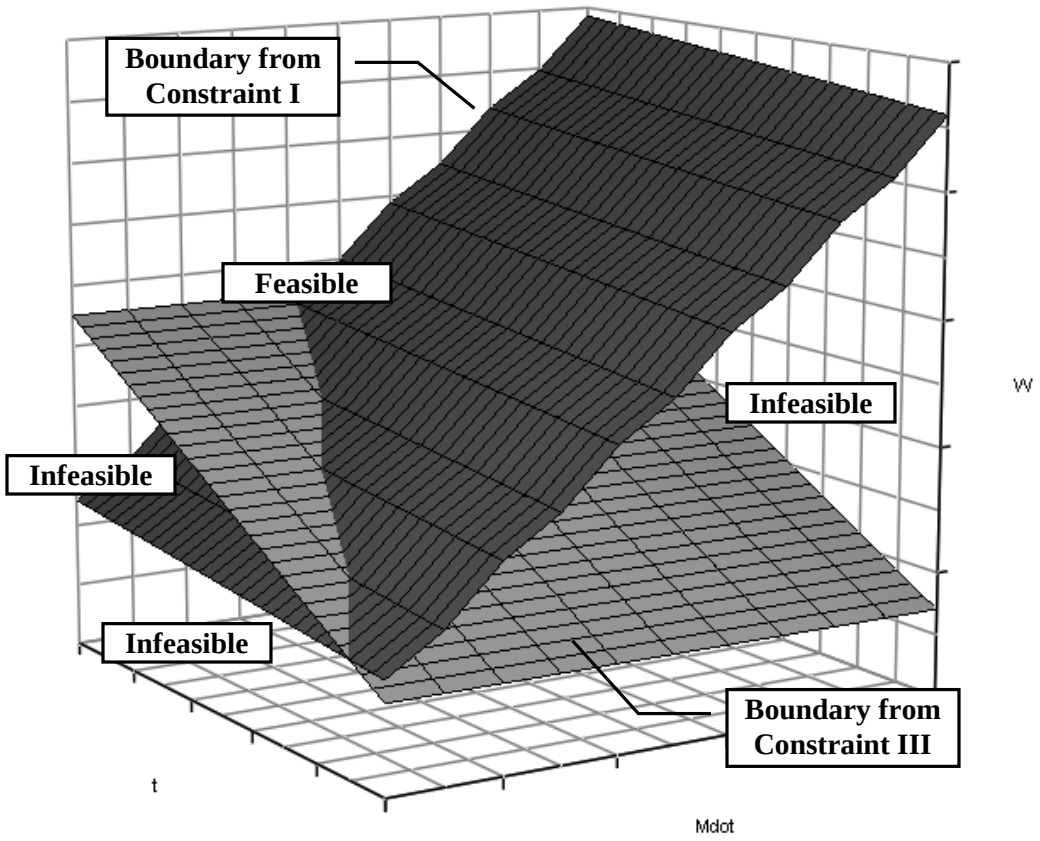


Figure 7.23 Boundaries from Constraints I and III in LCA Design

Table 7.29 Initial Experiments with 6 Data Points in E-RCEM

Mdot	*W*	*t*	*Mdot_n*	*W_n*	*t_n*	*Q*	*J*
0.00052	0.0348	0.00020	0.0072	0.9875	0.0028	-6.93	0.01164
0.00102	0.0266	0.00020	0.2067	0.5794	0.0013	-13.24	0.01022
0.00130	0.0347	0.00038	0.3201	0.9873	0.3026	-15.41	0.00184
0.00054	0.0217	0.00039	0.0147	0.3341	0.3169	-11.48	0.00139
0.00055	0.0152	0.00020	0.0195	0.0118	0.0073	-11.34	0.00754
0.00051	0.0308	0.00055	0.0023	0.7907	0.5806	-9.63	0.00065

Table 7.30 Values of θ for Initial Kriging Metamodels of Responses in E-RCEM

	θ_1	θ_2	θ_3
Q	20.06353	0.94352	0.28535
J	0.00100	0.14830	15.84192

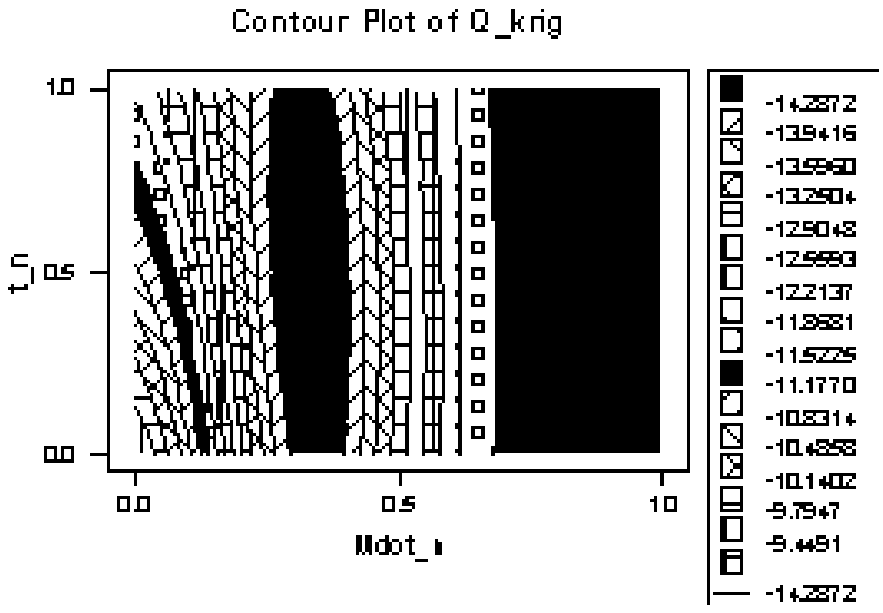


Figure 7.24 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Initial Kriging Metamodel with 6 Data Points)

Iteration I – Step 4: Identification of New Validation Points. Six new validation points are identified in the feasible design space. There is no information

about prediction errors, and design goals are not considered in the identification of new validation points in the first iteration. These points are added to be as far from existing data points as possible. A 12×12 covariance matrix is formatted with the first 6 columns and rows corresponding to existing data points and the last 6 columns and rows corresponding to new validation points. Then the new validation points are identified through maximization of the determinant of the covariance matrix. This is done in iSIGHT, following the same process as in Iteration I – Step 3 of the SEED method in Section 7.4 (see Figure D.15 in Appendix D.2.3). The new validation points and corresponding response values are listed in Table 7.31.

Table 7.31 Six Validation Points Identified in Iteration I – Step 4 in E-RCEM

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00057	0.0278	0.00024	0.0267	0.6424	0.0646	-9.41	0.00614
0.00107	0.0337	0.00059	0.2287	0.934	0.6533	-15.23	0.00056
0.00062	0.0266	0.00041	0.0479	0.5787	0.3527	-11.39	0.00133
0.00067	0.0332	0.00030	0.0668	0.9111	0.1586	-9.68	0.00351
0.00066	0.0240	0.00033	0.0626	0.4476	0.2195	-11.84	0.00231
0.00095	0.0268	0.00027	0.1798	0.5896	0.1209	-13.32	0.00430

Prediction errors of the initial metamodels (Table 7.30) at validation points are then calculated and listed in Table 7.32. Prediction errors at data points are zero. Kriging metamodels of prediction errors are then developed based on the information at 12 observed points. Values of θ for these metamodels are listed in Table 7.33. The observed maximum absolute error for Q is about 1.14, and that for J is about 0.00315.

Table 7.32 Prediction Errors of Initial Metamodels at 6 Validation Points

Mdot_n	*W_n*	*t_n*	*Q*	*J*	*Q_krig*	*J_krig*	*Q_err*	*J_err*
0.0267	0.6424	0.0646	-9.41	0.00614	-9.32	0.00922	0.09	0.00308
0.2287	0.934	0.6533	-15.23	0.00056	-14.09	0.00129	1.14	0.00073
0.0479	0.5787	0.3527	-11.39	0.00133	-10.75	0.00089	0.64	-0.00044
0.0668	0.9111	0.1586	-9.68	0.00351	-8.66	0.00666	1.02	0.00315
0.0626	0.4476	0.2195	-11.84	0.00231	-11.21	0.00364	0.63	0.00133
0.1798	0.5896	0.1209	-13.32	0.00430	-12.81	0.00727	0.51	0.00297

Table 7.33 Values of θ for Kriging Metamodels of Prediction Errors Developed with Information at Observed 12 Points in Iteration I – Step 4

	θ_1	θ_2	θ_3
Q	27.09014	0.37486	0.23548
J	6.91046	0.01056	99.99883

Iteration I – Steps 5 and 6: Identification of New Data Points and Updated Metamodels of Responses. In this step we plan to add in two new data points. A 14×14 covariance matrix is formulated with the first 6 rows and columns corresponding to the data points, the 7th to 12th rows and columns corresponding to the validation points, and the last 2 rows and columns corresponding to the new data points. In this formulation, we set θ_1 = 27.09014, θ_2 = 0.94352, and θ_3 = 99.99883, which are the largest values of θ's in metamodels of responses (Table 7.30) and those of prediction errors (Table 7.33).

To adjust entries of the covariance matrix, we need to have information of prediction errors and achievement of design goals. Values of the coefficients, α_i and α_j, are calculated with Equation (7.18). In this calculation, we have n_r = 2, $\rho_1 = \rho_2$ = 0.5, $\lambda_1 = \lambda_2$ = 2, $e_{max,1}$ = 1.14, $e_{max,2}$ = 0.00315.

Values of η_i and η_j are calculated with Equations (7.22) and (7.23). There are n_g = 3 design goals with the same weights, i.e., $w_1 = w_2 = w_3 = 1$. The first design goal is to maximize the total heat transfer rate, Q (here we multiply it with –1 which makes the response values positive). We calculate the degree of achievement of the 1st design goal, $goal.achievement_1$, with Equation (6.25). The target value is $T_{1,H} = 20$; we set $y_{1,max} = 16.0$, $y_{1,min} = 6.0$, and $\gamma_1 = 2$. The second design goal is to minimize the compliance, J. We calculate the degree of achievement of the 2nd design goal, $goal.achievement_2$, with Equation (6.24). The target value is $T_{2,L} = 0.0015$; we set $y_{2,max} = 0.012$, $y_{2,min} = 0.00056$, and $\gamma_2 = 2$. The third design goal is to minimize the cross-section area for solid materials, A_s. We calculate the degree of achievement of the 3rd design goal, $goal.achievement_3$, with Equation (6.24). The target value is $T_{3,L} = 0.00025$; we set $y_{3,max} = 0.00046$, $y_{3,min} = 0.00005$, and $\gamma_3 = 2$.

After the calculation of correction coefficients, we adjust entries of the covariance matrix with Equation (7.17). By maximizing the determinant of the adjusted covariance matrix, 2 new data points are identified and listed in Table 7.34. The FORTRAN code used to formulate the adjusted covariance matrix is presented in Appendix D.3.2, and the implementation of the new-point-identification process is illustrated in Appendix D.3.3.

Table 7.34 Two New Data Points Identified in Iteration I – Step 5 in E-RCEM

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00134	0.0350	0.00032	0.3355	1	0.1927	-15.23	0.00299
0.00127	0.0348	0.00049	0.306	0.9884	0.4764	-15.89	0.00092

Now we have 8 data points and 6 validation points. New kriging metamodels are developed for responses Q and J. Values of θ for the kriging metamodels are listed in Table 7.35. The contour plot of the metamodel for total heat transfer rate (Q) versus wall thickness (t) and mass flow rate (*Mdot*) is illustrated in Figure 7.25. More contour plots are presented in Appendix D.3. Comparing with Figure 7.25 we see that the initial metamodel is not accurate at all.

Table 7.35 Values of θ for Kriging Metamodels of Responses Developed with 8 Data Points in Iteration I – Step 6

	θ_1	θ_2	θ_3
Q	3.46015	0.45474	0.14310
J	0.00100	0.10223	16.22954

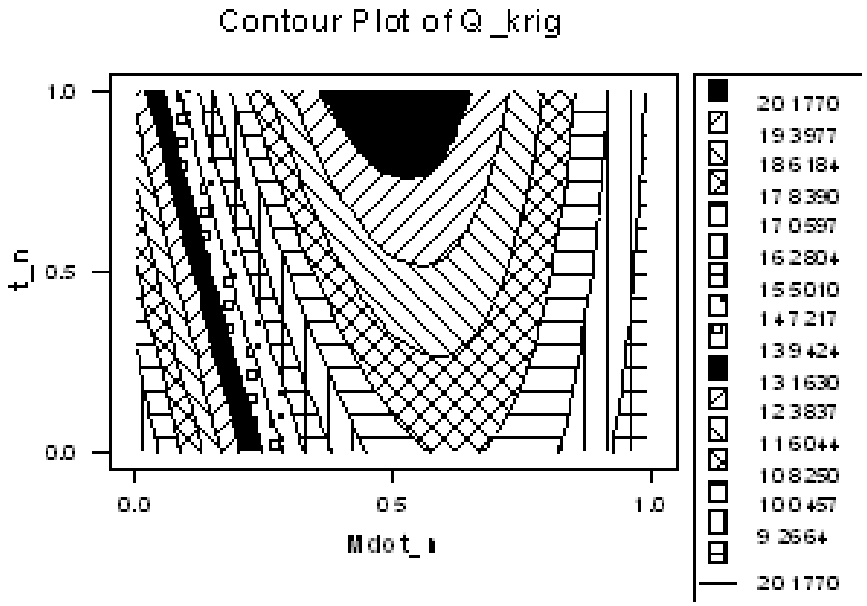


Figure 7.25 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 8 Data Points)

Iteration I – Step 7: Analysis of Design. Since the stopping criterion is not satisfied we will go to the next iteration of the integrated process of metamodeling and design space exploration.

Iteration II – Step 4: Identification of New Validation Points. In this step we plan to add in 2 new validation points. In this step, we temporarily switch the roles of data points and validation points, and new validation points are identified to bring maximum possible potential information about the actual response and achievement of design goals.

Kriging metamodels are developed for responses *Q* and *J* with 6 validation points. Values of θ for the kriging metamodels are listed in Table 7.36. The contour plot of the metamodel for total heat transfer rate (*Q*) versus wall thickness (*t*) and mass flow rate (*Mdot*) is illustrated in Figure 7.26. More contour plots are presented in Appendix D.3.

Prediction errors of these metamodels of responses (Table 7.36) at 8 data points are calculated and listed in Table 7.37. Prediction errors at 6 validation points are zero. Then kriging metamodels of prediction errors are developed with this information; values of θ for these kriging metamodels are listed in Table 7.38. The observed maximum absolute error for *Q* is about 1.30, and that for *J* is about 0.0038.

Table 7.36 Values of θ for Kriging Metamodels of Responses Developed with 6 Validation Points in Iteration II – Step 4

	θ_1	θ_2	θ_3
Q	2.36289	0.21077	0.10405
J	0.09658	0.00100	8.72353

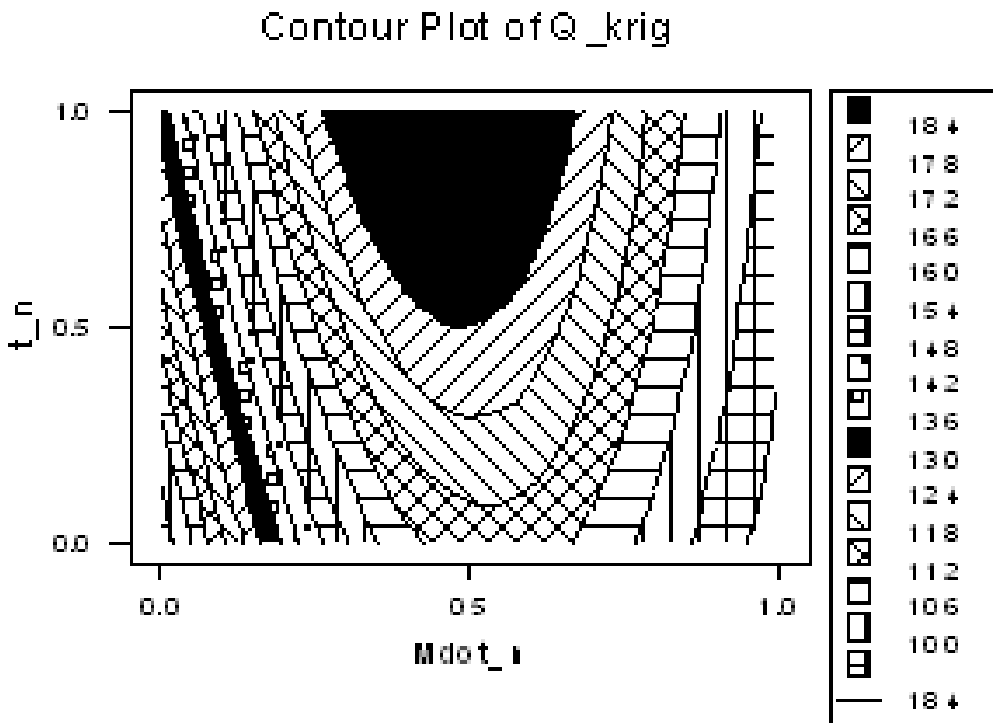


Figure 7.26 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 6 Validation Points)

Table 7.37 Prediction Errors of Metamodels at 8 Data Points in Iteration II – Step 4

Mdot_n	***W_n***	***t_n***	***Q***	***J***	***Q_krig***	***J_krig***	***Q_err***	***J_err***
0.0072	0.9875	0.0028	-6.93	0.01164	-7.45	0.00786	-0.52	-0.00378
0.2067	0.5794	0.0013	-13.24	0.01022	-13.40	0.00762	-0.16	-0.00260
0.3201	0.9873	0.3026	-15.41	0.00184	-15.28	0.00119	0.13	-0.00065
0.0147	0.3341	0.3169	-11.48	0.00139	-11.77	0.00146	-0.29	0.00007
0.0195	0.0118	0.0073	-11.34	0.00754	-12.64	0.00773	-1.30	0.00019
0.0023	0.7907	0.5806	-9.63	0.00065	-10.33	0.00089	-0.70	0.00024
0.3355	1	0.1927	-15.23	0.00299	-15.05	0.00239	0.18	-0.00060
0.306	0.9884	0.4764	-15.89	0.00092	-15.72	0.00108	0.17	0.00016

Table 7.38 Values of θ for Kriging Metamodels of Prediction Errors Developed with 14 Points in Iteration II – Step 4

	θ_1	θ_2	θ_3
Q	30.92156	3.99810	0.23290
J	0.95488	1.05725	99.99985

A 16×16 covariance matrix is formulated with the first 8 rows and columns corresponding to 8 data points, the 9th to the 14th rows and columns corresponding to 6 validation points, and the last 2 rows and columns corresponding to 2 new validation points. In the formulation of this covariance matrix, we set $\theta_1 = 30.92156$, $\theta_2 = 3.99810$, and $\theta_3 = 99.99985$.

Then entries of the covariance matrix are adjusted with similar methods to that used in Iteration I – Step 5. In the adjustment we have $n_r = 2$, $\rho_1 = \rho_2 = 0.5$, $\lambda_1 = \lambda_2 = 2$, $e_{max,1} = 1.30$, $e_{max,2} = 0.0038$; $n_g = 3$, $w_1 = w_2 = w_3 = 1$, $T_{1,H} = 20$, $y_{1,max} = 16.0$, $y_{1,min} = 6.0$, $T_{1,H} = 0.0015$, $y_{2,max} = 0.012$, $y_{2,min} = 0.00056$, $T_{3,L} = 0.00025$, $y_{3,max} = 0.00046$, $y_{3,min} = 0.00005$, and $\gamma_1 = \gamma_2 = \gamma_3 = 2$. By maximizing the determinant of the adjusted covariance matrix, 2 new validation points are identified and listed in Table 7.39.

Table 7.39 Two New Validation Points Identified in Iteration II – Step 4

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00078	0.0207	0.00020	0.1134	0.2837	0.0002	-12.27	0.00896
0.00132	0.0350	0.00038	0.3265	1	0.2924	-15.50	0.00184

Iteration II – Steps 5 and 6: Identification of New Data Points and Updated Metamodels of Responses. In this step we plan to add in 2 new data points. To formulate and adjust entries of the covariance matrix, we need information about prediction errors and achievement of design goals. Prediction errors of the metamodels of responses developed in Iteration I – Step 6 (in Table 7.35) at 8 validation points are listed in Table 7.40. Prediction errors at 8 data points are zero. The observed maximum absolute error for Q is about 0.78, and that for J is about 0.00131. Based on this information, kriging metamodels of prediction errors are developed, and values of θ for these kriging metamodels are listed in Table 7.41.

Table 7.40 Prediction Errors of Metamodels at 8 Validation Points Calculated in Iteration II – Step 5

Mdot_n	*W_n*	*t_n*	*Q*	*J*	*Q_krig*	*J_krig*	*Q_err*	*J_err*
0.0267	0.6424	0.0646	-9.41	0.00614	-9.39	0.00745	0.02	0.00131
0.2287	0.934	0.6533	-15.23	0.00056	-15.04	0.00134	0.19	0.00078
0.0479	0.5787	0.3527	-11.39	0.00133	-11.09	0.00156	0.30	0.00023
0.0668	0.9111	0.1586	-9.68	0.00351	-8.90	0.00385	0.78	0.00034
0.0626	0.4476	0.2195	-11.84	0.00231	-11.59	0.00154	0.25	-0.00077
0.1798	0.5896	0.1209	-13.32	0.00430	-13.08	0.00452	0.24	0.00022
0.1134	0.2837	0.0002	-12.27	0.00896	-12.41	0.00902	-0.14	0.00006
0.3265	1	0.2924	-15.50	0.00184	-15.47	0.00187	0.03	0.00003

Table 7.41 Values of θ for Kriging Metamodels of Prediction Errors Developed with 16 Points in Iteration II – Step 5

	θ_1	θ_2	θ_3
Q	53.44009	4.84689	99.99936
J	99.99713	99.99965	2.55718

An 18×18 covariance matrix is formulated, with the first 8 rows and columns corresponding to 8 data points, the 9th to the 16th rows and columns corresponding to 8 validation points, and the last 2 rows and columns corresponding to the new data points. In the formulation of this covariance matrix, we set θ_1 = 99.99713, θ_2 = 99.99965, and θ_3 = 99.99936.

Then entries of the covariance matrix are adjusted with similar methods to that used in Iteration I – Step 5. In the adjustment we have n_r = 2, $\rho_1 = \rho_2$ = 0.5, $\lambda_1 = \lambda_2$ = 2, $e_{max,1}$ = 0.78, $e_{max,2}$ = 0.00131; n_g = 3, $w_1 = w_2 = w_3$ = 1, $T_{1,H}$ = 20, $y_{1,max}$ = 16.0, $y_{1,min}$ = 6.0, $T_{1,H}$ = 0.0015, $y_{2,max}$ = 0.012, $y_{2,min}$ = 0.00056, $T_{3,L}$ = 0.00025, $y_{3,max}$ = 0.00046, $y_{3,min}$ = 0.00005, and $\gamma_1 = \gamma_2 = \gamma_3$ = 1.5. By maximizing the determinant of the adjusted covariance matrix, 2 new validation points are identified and listed in Table 7.42.

Table 7.42 Two New Data Points Identified in Iteration II – Step 5

Mdot	***W***	***t***	***Mdot_n***	***W_n***	***t_n***	***Q***	***J***
0.00056	0.0347	0.00044	0.0241	0.9868	0.4026	-8.67	0.00122
0.00063	0.0179	0.00031	0.0506	0.1438	0.183	-12.32	0.00238

Now we have 10 data points and 8 validation points. Since we will stop in Iteration III – Step 4, which is the next iteration, and final metamodels will be developed with all data and validation points, we do not need to update the metamodels of responses in this step.

Iteration II – Step 7: Analysis of Design. Since the stopping criterion is not satisfied we will go to the next iteration of the integrated process of metamodeling and

design space exploration. The analysis of achievement of design goals is not done here in this example; it will be done after we finish the E-RCEM process in the next step.

Iteration III – Step 4: Identification of New Validation Points. In this step we plan to add in 2 new validation points. In this step, we temporarily switch the roles of data points and validation points, and new validation points are identified to bring maximum possible potential information about the actual response and achievement of design goals.

Kriging metamodels are developed for responses *Q* and *J* with 8 validation points. Values of θ for the kriging metamodels are listed in Table 7.43. The contour plot of the metamodel for total heat transfer rate (*Q*) versus wall thickness (*t*) and mass flow rate (*Mdot*) is illustrated in Figure 7.27. More contour plots are presented in Appendix D.3.

Prediction errors of these metamodels of responses (Table 7.43) at 10 data points are calculated and listed in Table 7.44. Prediction errors at 8 validation points are zero. Then kriging metamodels of prediction errors are developed with this information; values of θ for these kriging metamodels are listed in Table 7.45. The observed maximum absolute error for *Q* is about 0.35, and that for *J* is about 0.0027.

Table 7.43 Values of θ for Kriging Metamodels of Responses Developed with 6 Validation Points in Iteration II – Step 4

	θ_1	θ_2	θ_3
Q	2.36289	0.21077	0.10405
J	0.09658	0.00100	8.72353

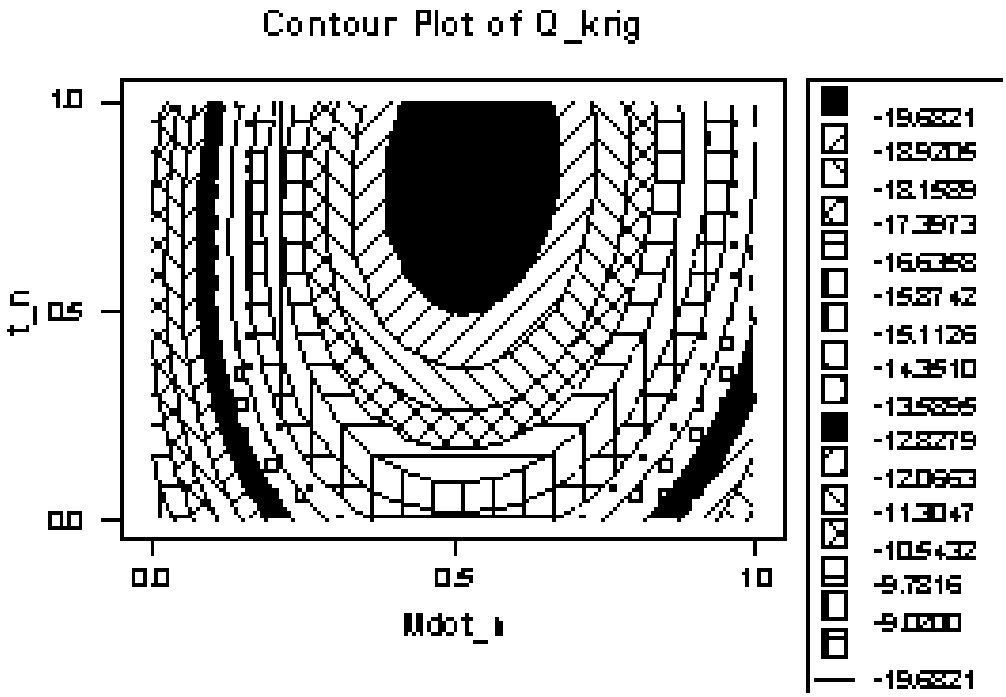


Figure 7.27 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 6 Validation Points)

Table 7.44 Prediction Errors of Metamodels at 8 Data Points in Iteration II – Step 4

Mdot_n	*W_n*	*t_n*	*Q*	*J*	*Q_krig*	*J_krig*	*Q_err*	*J_err*
0.0072	0.9875	0.0028	-6.93	0.01164	-7.20	0.00896	-0.27	-0.00268
0.2067	0.5794	0.0013	-13.24	0.01022	-12.91	0.00899	0.33	-0.00123
0.3201	0.9873	0.3026	-15.41	0.00184	-15.52	0.00176	-0.11	-0.00008
0.0147	0.3341	0.3169	-11.48	0.00139	-11.44	0.00142	0.04	0.00003
0.0195	0.0118	0.0073	-11.34	0.00754	-11.39	0.00852	-0.05	0.00098
0.0023	0.7907	0.5806	-9.63	0.00065	-9.46	0.0006	0.17	-0.00005
0.3355	1	0.1927	-15.23	0.00299	-14.93	0.00294	0.30	-0.00005
0.306	0.9884	0.4764	-15.89	0.00092	-16.17	0.00091	-0.28	-0.00001
0.0241	0.9868	0.4026	-8.67	0.00122	-9.02	0.00128	-0.35	0.00006
0.0506	0.1438	0.183	-12.32	0.00238	-12.59	0.00273	-0.27	0.00035

Table 7.45 Values of θ for Kriging Metamodels of Prediction Errors Developed with 14 Points in Iteration II – Step 4

	θ_1	θ_2	θ_3
Q	0.00102	99.99939	63.99243
J	0.52949	0.79194	56.07930

A 20×20 covariance matrix is formulated with the first 10 rows and columns corresponding to 10 data points, the 11^{th} to the 18^{th} rows and columns corresponding to 8 validation points, and the last 2 rows and columns corresponding to 2 new validation points. In the formulation of this covariance matrix, we set $\theta_1 = 2.36289$, $\theta_2 = 99.99939$, and $\theta_3 = 63.99243$.

Then entries of the covariance matrix are adjusted with similar methods to that used in Iteration I – Step 5. In the adjustment we have $n_r = 2$, $\rho_1 = \rho_2 = 0.5$, $\lambda_1 = \lambda_2 = 2$, $e_{max,1} = 0.35$, $e_{max,2} = 0.0027$; $n_g = 3$, $w_1 = w_2 = w_3 = 1$, $T_{1,H} = 20$, $y_{1,max} = 16.0$, $y_{1,min} = 6.0$, $T_{1,H} = 0.0015$, $y_{2,max} = 0.012$, $y_{2,min} = 0.00056$, $T_{3,L} = 0.00025$, $y_{3,max} = 0.00046$, $y_{3,min} = 0.00005$, and $\gamma_1 = \gamma_2 = \gamma_3 = 1.25$. Note that in this iteration, the value of γ in this iteration (which is 1.25) is smaller than those used in Iteration II – Step 5 (which is 1.5) or Iteration I – Step 5 (which is 2.0). This is because that as we have more knowledge of the actual responses and more confidence on the metamodel, more emphasis is put on the achievement of design goals in identifying new points. By maximizing the determinant of the adjusted covariance matrix, 2 new validation points are identified and listed in Table 7.46.

Table 7.46 Two New Validation Points Identified in Iteration III – Step 4

Mdot	*W*	*t*	*Mdot_n*	*W_n*	*t_n*	*Q*	*J*
0.00112	0.0305	0.00041	0.2462	0.7728	0.343	-14.94	0.00141
0.00121	0.0311	0.00020	0.2838	0.8058	0.0006	-13.94	0.01104

Now we have observed totally 20 points (10 data points and 10 validation points), the integrated process of metamodeling and design space exploration in E-RCEM will stop in this iteration. Steps 5 and 6 in this iteration are skipped because we do not plan to add in more points. Thus, we will directly enter Step 7, the analysis of design.

Iteration III – Step 7: Analysis of Design. All 20 observed points are listed in Table 7.47. All these points are in the feasible design space. In Table 7.47 we list not only the response values but also the values of deviation variables and the deviation function. When necessary we can select the point with the minimum value of the deviation function z from Table 7.47, and take it as the design solution to be used in the future design. In Table 7.47 we see that the smallest value of the deviation function at all observed points is $z = 0.35381$, at the point of $Mdot\_n = 0.3265$, $W\_n = 1.0$, $t\_n = 0.2924$.

Better design solutions can be found by exploring the feasible design space with metamodels developed with all observed points. The final kriging metamodels are developed for Q and J with information from Table 7.47; values of θ for these kriging metamodels are listed in Table 7.48. Contour plots of the responses versus design variables are illustrated in Figure 7.28, Figure 7.29, Figure 7.30, and Figure 7.31. Comparing Figure 7.28, Figure 7.29, Figure 7.30, Figure 7.31 with Figure 7.8, Figure 7.9,

Figure 7.10, and Figure 7.11, we see that the kriging metamodels of responses developed with the integrated design process in E-RCEM is not globally accurate.

Table 7.47 All Points Identified in the Integrated Process in E-RCEM

Mdot_n	*W_n*	*t_n*	*Q*	*J*	*As*	d_1^-	d_2^+	d_3^+	*z*
0.0072	0.9875	0.0028	-6.93	0.01164	0.00012	0.93357	0.96571	0.00000	1.89929
0.2067	0.5794	0.0013	-13.24	0.01022	0.00009	0.48286	0.83048	0.00000	1.31333
0.3201	0.9873	0.3026	-15.41	0.00184	0.00023	0.32786	0.03238	0.00000	0.36024
0.0147	0.3341	0.3169	-11.48	0.00139	0.00014	0.60857	0.00000	0.00000	0.60857
0.0195	0.0118	0.0073	-11.34	0.00754	0.00005	0.61857	0.57524	0.00000	1.19381
0.0023	0.7907	0.5806	-9.63	0.00065	0.00028	0.74071	0.00000	0.14485	0.88556
0.3355	1	0.1927	-15.23	0.00299	0.00019	0.34071	0.14190	0.00000	0.48262
0.306	0.9884	0.4764	-15.89	0.00092	0.00029	0.29357	0.00000	0.17851	0.47209
0.0241	0.9868	0.4026	-8.67	0.00122	0.00026	0.80929	0.00000	0.04354	0.85282
0.0506	0.1438	0.183	-12.32	0.00238	0.00009	0.54857	0.08381	0.00000	0.63238
0.0267	0.6424	0.0646	-9.41	0.00614	0.00012	0.75643	0.44190	0.00000	1.19833
0.2287	0.934	0.6533	-15.23	0.00056	0.00033	0.34071	0.00000	0.37951	0.72023
0.0479	0.5787	0.3527	-11.39	0.00133	0.00018	0.61500	0.00000	0.00000	0.61500
0.0668	0.9111	0.1586	-9.68	0.00351	0.00017	0.73714	0.19143	0.00000	0.92857
0.0626	0.4476	0.2195	-11.84	0.00231	0.00013	0.58286	0.07714	0.00000	0.66000
0.1798	0.5896	0.1209	-13.32	0.00430	0.00012	0.47714	0.26667	0.00000	0.74381
0.1134	0.2837	0.0002	-12.27	0.00896	0.00007	0.55214	0.71048	0.00000	1.26262
0.3265	1	0.2924	-15.50	0.00184	0.00023	0.32143	0.03238	0.00000	0.35381
0.2462	0.7728	0.343	-14.94	0.00141	0.00021	0.36143	0.00000	0.00000	0.36143
0.2838	0.8058	0.0006	-13.94	0.01104	0.00011	0.43286	0.90857	0.00000	1.34143

Table 7.48 Values of θ for Final Kriging Metamodels of Responses Developed with 20 Points in Iteration III – Step 7

	θ_1	θ_2	θ_3
Q	27.64666	0.36657	0.32412
J	0.13692	0.47800	63.94798

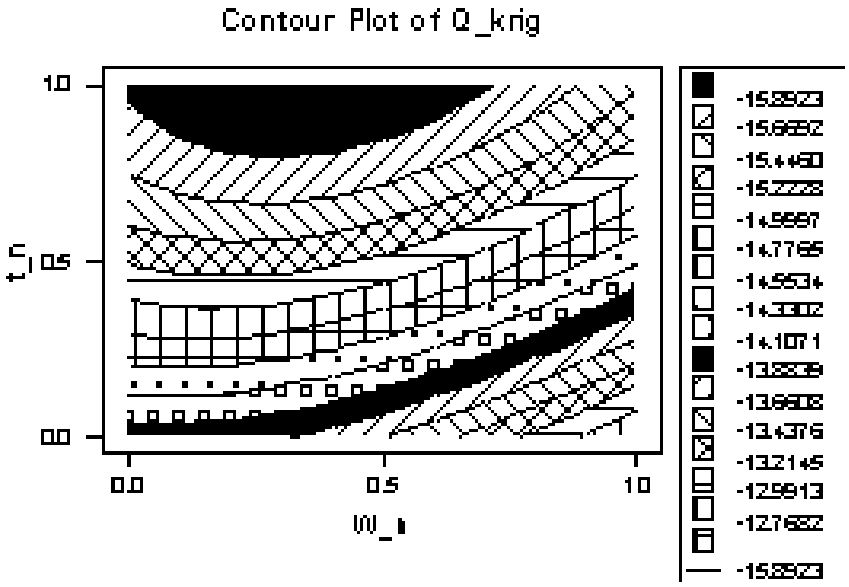


Figure 7.28 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Device Width (Kriging Metamodel with 20 Points)

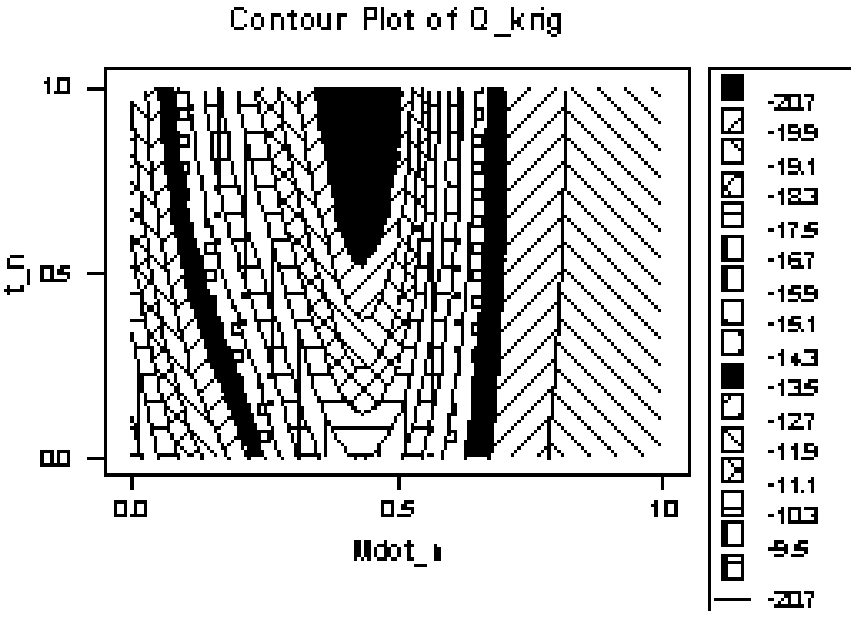


Figure 7.29 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 20 Points)

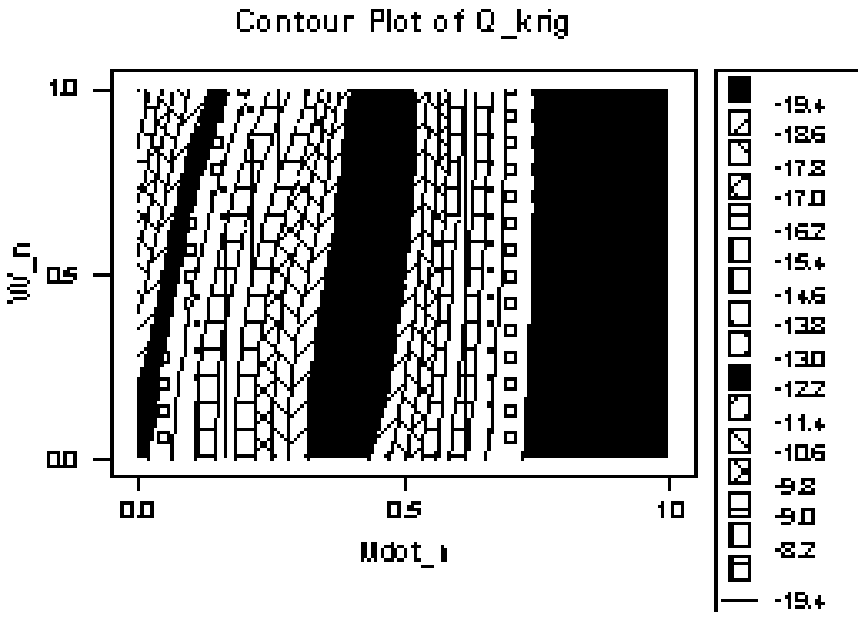


Figure 7.30 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 20 Points)

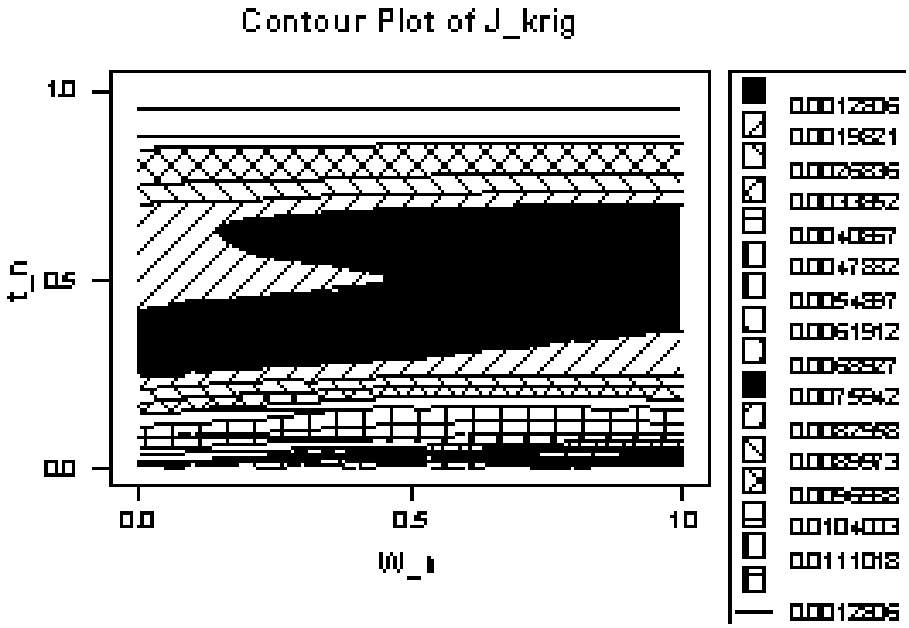


Figure 7.31 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 20 Points)

Root mean square error (RMSE) of metamodels are calculated with information from 1573 points, and listed in Table 7.49. Values of RMSE and NRMSE for *Q* and *J* in Table 7.49 are much larger than those in Table 7.7 and Table 7.28, which supports our observation that the metamodels of responses developed in this section is not as accurate as those developed with RCEM or SEED. The design solution is obtained by solving the compromise DSP in Figure 7.13, and listed in Table 7.50.

Table 7.49 Root Mean Squared Errors of Metamodels Developed in RCEM

	Metamodels with 28 Points Identified with SEED	
	Q	***J***
RMSE	15.1906	0.0016713
NRMSE	14.75%	14.60%

Table 7.50 The Design Solution Obtained with the Integrated Design Process in E-RCEM

	Predicted Value	**Actual Value**
Mass flow rate, *Mdot* (kg/s)	0.00130	
Device width, *W* (m)	0.0350	
Wall thickness, *t* (m)	0.00043	
Mdot*_*normalized	0.3183	
W*_*normalized	0.9991	
t*_*normalized	0.3810	
Reynolds number, R_e	2300.00	
Volume fraction, *vf*	0.20836	
$30 - 2663.35\dot{M} - \Delta P$	26.50	
Area of solid materials, A_s (m^2)	0.00025	
Heat transfer rate, *Q* (W)	–15.72	–15.69
Compliance, *J* (m/N)	0.00124	0.00131
$Z = d_1^- + d_2^+ + d_3^+$	0.30545	0.33587

Usually a metamodel is acceptable when the value of NRMSE is smaller than 5%. The final metamodels developed in this section have values of NRMSE around 15%, which is unacceptable from the viewpoint of metamodeling. However, in the integrated design process of E-RCEM, to achieve a globally accurate metamodel is not the goal; E-RCEM aims at identifying most-likely-to-succeed regions in the feasible design space, building locally accurate metamodels, and achieving robust design solutions with little time or money spent on expensive computer simulations or physical experiments. The achievement of better design solutions is the goal of E-RCEM, which is the same for all designs. Acceptable metamodels help realize this goal, but they are not the goal. A metamodel with higher global fidelity does not ensure a better design solution; in other words, a metamodel with lower global fidelity may lead to a better design solution. It is not surprising to see that the final metamodels developed in E-RCEM are not as accurate as those developed with RCEM and SEED, while a fair comparison should only be done on the achievement of good design solutions. More analysis will be done in Section 7.6.

7.6 A COMPARISON AND DISCUSSION ON RCEM, SEED, AND THE INTEGRATED DESIGN PROCESS IN E-RCEM

Solutions are obtained for the LCA unit design with RCEM, the traditional process with SEED in E-RCEM, and the integrated design process in E-RCEM in Sections 7.3, 7.4, and 7.5, respectively. In this section, comparisons are done on the performance of these three methods, and then recommendations are given on how to use them in design.

7.6.1 Comparison of Performance of Metamodels on Response Prediction

First we compare the performance of the three methods in response prediction. Based on information from 1573 evenly spread points in the whole design space, values of RMSE and NRMSE are calculated for metamodels of responses developed in Sections 7.3, 7.4, and 7.5 using Equation (2.34) and listed in Table 7.51.

Table 7.51 Root Mean Squared Errors of Metamodels Developed in RCEM, SEED, and the Integrated Design Process in E-RCEM – Comparison in the Whole Design Space

Response	***Q***				***J***			
Method	**RCEM**		**SEED**	**E-RCEM**	**RCEM**		**SEED**	**E-RCEM**
# Points	30	40	28	20	30	40	28	20
RMSE	9.2047	8.3527	4.4767	15.1906	0.0003433	0.000175	0.0002304	0.0016713
NRMSE	8.94%	8.11%	4.35%	14.75%	3.00%	1.53%	2.01%	14.60%

In Table 7.51 we see that the metamodels developed in E-RCEM have largest values of RMSE and NRMSE, which indicates that they are most inaccurate among all metamodels. For the response *Q*, the metamodel developed in SEED is the most accurate one. For the response *J*, the metamodel developed with 40 points in RCEM is most accurate, but is only slightly better than the one developed in SEED. In Table 7.51 we also see that when being compared in the whole design space, the metamodels developed with 40 points in RCEM are more accurate than those developed with 30 points in RCEM, which is reasonable. From the viewpoint of global metamodel accuracy, the metamodels developed in SEED are best because they perform better (or nearly as well as) than other metamodels in response prediction with fewer observed points (except E-RCEM, which uses fewer points than SEED in this example). This is apparent when we

compare the metamodels from SEED with those with 30 points from RCEM: metamodels from SEED have smaller values of NRMSE for both *Q* and *J* than those from RCEM though fewer points are used in SEED. This observation proves that through the identification of regions with large prediction errors, more globally accurate metamodels can be developed with fewer observed points sequentially added with the SEED method. It is not surprising to see that metamodels developed in SEED perform best in response prediction in the whole design space.

A comparison is done among these metamodels on response prediction in the feasible design space. Prediction errors at 159 points evenly spread in the feasible design space are observed and used to calculate values of RMSE and NRMSE. The results are listed in Table 7.52.

Table 7.52 Root Mean Squared Errors of Metamodels Developed in RCEM, SEED, and the Integrated Design Process in E-RCEM – Comparison in the Feasible Design Space

Response	***Q***				***J***			
Method	**RCEM**		**SEED**	**E-RCEM**	**RCEM**		**SEED**	**E-RCEM**
# Points	30	40	28	20	30	40	28	20
RMSE	4.6587	6.5112	1.0886	0.2128	0.0006787	0.0002631	0.0003457	0.0002355
NRMSE	49.35%	68.97%	11.53%	2.25%	6.10%	2.36%	3.12%	2.12%

In Table 7.52 we see that metamodels developed in E-RCEM perform much better in the feasible design space than they do in the whole design space; their values of NRMSE are a little larger than 2%, which are much smaller than those in Table 7.51. All metamodels from RCEM and SEED perform much worse in the feasible design space

than in the whole design space. Among all metamodels, those developed in E-RCEM are most accurate in the feasible design space.

An interesting observation is that the metamodels developed with 40 points in RCEM are more accurate than those with 30 points when being compared in the whole design space, but not as accurate as those with 30 points in the feasible design space. This is because that more of the 40 points are put in the infeasible design space than those of the 30 points. The metamodels developed in SEED perform worse in the feasible design space than in the whole design space because there is high nonlinearity in the infeasible design space and as a result, many points are added sequentially in these regions to help grasp the nonlinearity. Even so, the metamodels developed in SEED are still much more accurate than those developed in RCEM. Note that the values of NRMSE of the RCEM metamodels for *Q* are about 50%, which means that the root mean squared error of these metamodels is about half of the actual response range; Metamodels with such large prediction errors can not be trusted in design. The metamodel from SEED is much better with an error bound of about 10%.

If the values of NRMSE calculated with observations in the whole design space (Table 7.51) are used to judge whether a metamodel is acceptable or not (using 5% or 10% as the criterion), the results are: 1). Metamodels from SEED are acceptable because their values of NRMSE are smaller than 5%; 2). Metamodels from RCEM are acceptable or nearly acceptable because their values of NRMSE are smaller than 5% for *J* and 10% for *Q*; and 3). Metamodels from E-RCEM are unacceptable because their values of NRMSE are larger than 10%.

If the values of NRMSE calculated with observations in the feasible design space (Table 7.52) are used to judge whether a metamodel is acceptable or not (using 5% or 10% as the criterion), the results are: 1). Metamodels from SEED are unacceptable because their values of NRMSE for *Q* are dramatically larger than 10%; 2). Metamodels from RCEM are acceptable or nearly acceptable because their values of NRMSE are smaller than 5% for *J* and only slightly larger than 10% for *Q*; and 3). Metamodels from E-RCEM are acceptable because their values of NRMSE are smaller than 5%.

The judgments based on local metamodel accuracy in the feasible design space are very different from that based on global metamodel accuracy in the whole design space. Since the final design solution is obtained through exploration of the feasible design space, we conclude that the metamodel accuracy in the feasible design space is a more reliable criterion than that calculated with observations in the whole design space. This is further proved by studies in Section 7.6.2.

7.6.2 Comparison of Performance of Metamodels in Sequential Design Space Exploration

In this section we compare the performance of metamodels in design space exploration, in other words, we compare the design solutions obtained with metamodels developed in RCEM, SEED, and E-RCEM. The actual design solution is obtained in Section 7.2 with original simulations. Design solutions from all methods are listed in Table 7.53. Note that in Table 7.53, RCEM (I) stands for the solution obtained with metamodels developed with 30 points in RCEM, and RCEM (II) stands for that with metamodels developed with 40 points in RCEM. As described in Section 7.5, there are

two ways to identify the design solution in E-RCEM: 1). Final metamodels of responses are developed with information at all observed points, and then the compromise DSP is solved to identify the design solution, or 2). The design solution can be selected from the observed points because the exploration of design solutions has already been incorporated in the sequential metamodeling process through the formulation of design goals and constraints in the compromise DSP in E-RCEM. The solution obtained in the first way in E-RCEM is represented by E-RCEM (II) and that obtained in the second way is represented by E-RCEM (I) in Table 7.53

Table 7.53 The Design Solutions Obtained with Simulations, RCEM, SEED, and the Integrated Design Process in E-RCEM

	Actual Solution	**RCEM**		**SEED**	**E-RCEM**	
		(I)	**(II)**		**(I)**	**(II)**
# Points Observed	**–**	**30**	**40**	**28**	**20**	**20**
Mass flow rate, *Mdot* (kg/s)	0.00129	0.00097	0.0005	0.00113	0.00132	0.00130
Device width, *W* (m)	0.0348	0.0278	0.0201	0.0316	0.0350	0.0350
Wall thickness, *t* (m)	0.00042	0.00051	0.00036	0.00048	0.00038	0.00043
$Mdot_{normalized}$	0.316	0.1875	0.0	0.2535	0.3265	0.3183
$W_{normalized}$	0.99	0.6406	0.2532	0.8284	1.0	0.9991
$t_{normalized}$	0.3667	0.5708	0.2707	0.4694	0.2924	0.3810
Reynolds number, R_e	2297.61	2300	1644.18	2300	2300	2300
Volume fraction, *vf*	0.2054	0.29995	0.29872	0.25578	0.18588	0.20836
Constraint II	26.5141	27.30	28.44	26.91	26.44	26.50
Area of solid materials, A_s (m^2)	0.000249	0.00023	0.00012	0.00025	0.00023	0.00025
Heat transfer rate, *Q* (W)	-15.59	–14.77	–11.21	–15.30	–15.50	–15.69
Compliance, *J* (m/N)	0.00139	0.00080	0.00167	0.00093	0.00184	0.00131
$Z = d_1^- + d_2^+ + d_3^+$	0.31489	0.37351	0.64419	0.35620	0.35381	0.33587

In Table 7.53 we see that the solutions obtained from E-RCEM are closer to the actual design solution than those obtained from RCEM or SEED; the solutions also have smaller values of the deviation function, which means that they achieve design goals better than those obtained with RCEM or SEED. This verifies that better design solutions can be achieved with fewer observed points in the integrated design process in E-RCEM than in the traditional process used in RCEM and SEED.

The solution of E-RCEM (II), which is obtained with the final metamodels of responses developed with 20 points in E-RCEM, is better than that of E-RCEM (I), which is selected among the 20 observed points in E-RCEM. Thus when the expense on metamodel building and design space exploration is affordable, designers had better explore for design solutions with final metamodels of responses developed with all observed points. When the expense is not affordable (e.g., in cases with a lot of design variables, responses, constraints, and goals, the computation expense on design space exploration may be very high even with cheap-to-run metamodels), designers can skip the step of solving the compromise DSP and select the design solution from the observe points.

The solution obtained with SEED (the traditional process in E-RCEM) is better than those obtained with RCEM but worse than those obtained with E-RCEM. The solution obtained with 30 points is better than that obtained with 40 points in RCEM, which seems a little unexpected because we are not able to get a better design solution with more points observed in the design space in this example.

Here we can relate this comparison to that on metamodel accuracy in Section 7.6.1. Metamodels from E-RCEM perform worst on response prediction when being compared in the whole design space, but they help achieve the best design solutions. The metamodels developed with 40 points in RCEM performs second best on response prediction when being compared in the whole design space, but the solution obtained with these metamodels is the worst of all. This indicates that there is no clear positive correlation between the global metamodel accuracy and the performance in design space exploration. In other words, to obtain metamodels that perform well in response prediction in the whole design space does not ensure the achievement of good design solutions, while better design solutions could be achieved with metamodels with less global metamodel accuracy.

On the other hand, the metamodel accuracy in the feasible design space, as presented in Table 7.52, does have a positive correlation with the metamodels' performance in design space exploration. The metamodels from E-RCEM, which are the most accurate when, being compared in the feasible design space, facilitate the achievement of best design solutions of all. The worst design solution is obtained with the metamodels developed with 40 points in RCEM that perform worst on response prediction when being compared in the feasible design space. Metamodels from SEED are second best on response prediction in the feasible design space, and the solution obtained with SEED is also second best to the ones obtained with E-RCEM. Metamodels developed with 30 points in RCEM perform better in response prediction in the feasible

design space and thus help achieve a design solution than those developed with 40 points in RCEM.

The observations above suggests that when judging whether a metamodel is acceptable or not, we should examine the accuracy of metamodels in the feasible design space instead of the whole design space. This conclusion is intuitive and reasonable. However, there may not always be positive correlation between the metamodel accuracy in feasible design spaces and the performance in design space exploration. In the single-variable example in Chapter 6, the feasible design space is the same as the whole design space because there is not system constraint. In that example, a better design solution is achieved with a metamodel developed in E-RCEM than with that developed in SEED, though the metamodel developed in E-RCEM is not as accurate as that developed in SEED. Thus, when judging whether a metamodel is acceptable or not, we should focus on the *local metamodel accuracy in critical regions*, which is measured in regions where design goals are achieved or nearly achieved, instead of global metamodel accuracy, which is measured in the whole design space or feasible design space. In the LCA unit design example, because responses in the relatively small feasible design space are not highly nonlinear, we do not need to identify smaller critical regions and can view the feasible design space as a whole critical region. In this case, our conclusion holds valid because the metamodels that are most accurate in the critical region, which are developed in E-RCEM, facilitate the achievement of best design solutions. The judgment of whether metamodels are acceptable can be done in Step 7 of the integrated design process in E-RCEM, when the stopping criterion is to obtain good design solutions or accurate

metamodels, instead of being given a preset of maximum number of observed points as in the examples in Chapters 4 to 7 in this dissertation.

With E-RCEM and SEED, we are able to achieve better design solutions as well as save a lot of expense on simulations in this LCA design example. In cases with expensive experiments, this reduction of experimental expense is very valuable. In E-RCEM and SEED, we have additional expense on the calculation of prediction errors and achievement of design goals, the formulation of adjusted covariance matrix, the calculation of the determinant of the matrices, and the optimization to find out the matrix with the maximum determinant value. These expenses can be categorized into two categories:

- Manually operational expense. This includes the initialization of input, output, and parameter files for FORTRAN or C codes used in the E-RCEM or SEED process, the organization of analysis codes in iSIGHT, and the documentation of experimental and analysis results.
- Computational expense. This includes the computational time spent on the FORTRAN or C codes in E-RCEM or SEED and the optimization process in iSIGHT.

To build the E-RCEM and SEED processes in an automated framework, which means the exclusion of manual operations will help save a large portion of expense spent in the examples in this dissertation. This is future research for this dissertation. As for the computational expense, most time and effort is spent on the optimization to find the matrix with maximum value of the determinant. This expense can be reduced by using

appropriate optimization algorithms in iSIGHT. The E-RCEM and SEED processes are not as complicated or intensive as they appear in Sections 7.4 and 7.5. Some of the information in these sections is for illustration only and thus unnecessary in the sequential metamodeling and exploration process (e.g., the contour plots), and some can be easily managed within an automated framework (e.g., the documentation of information at data and validation points).

7.6.3 Selection of the Most Suitable Methods in Design: RCEM, SEED, or the Integrated Design Process in E-RCEM

RCEM is best used in cases with very cheap experiments and/or when the response surface is flat. When the expense of experiments or simulations is low, designers are able to collect information at a lot of data points and develop very accurate metamodels without adopting sequential metamodeling and design space exploration strategies. With cheap-to-run computer simulation models (no physical experiments are involved), designers even do not need to develop metamodels; the simulation codes can be linked in iSIGHT or similar software, and optimal (or robust) solutions can be found with optimization techniques. In some cases the experimental expense may be high, but based on experience, designers may select very small design spaces in which the responses are not nonlinear; RCEM is better in such cases because acceptable metamodels can be developed with very few data points, and thus there is no need to adopt a sequential strategy.

As described in this dissertation, SEED and E-RCEM are best used when: 1) the computer simulations or physical experiments are expensive to conduct, and/or 2)

designers expect (or are not sure) that the responses are nonlinear in the given design space. In such cases, SEED helps achieve more accurate metamodels in the whole design space with fewer experiments (or simulations) than the RCEM method. E-RCEM helps achieve more accurate metamodels in critical regions and thus obtain better design solutions with fewer experiments or simulations than SEED or RCEM.

The consideration of design constraints in the integrated design process of E-RCEM can also be used in SEED (or say, the traditional process of E-RCEM in which the information flow from the process of metamodeling to the process of design space exploration is one-way). In this way, designers are able to develop accurate metamodels in the feasible design space with SEED, without wasting time or money on experiments in infeasible design spaces. The corresponding metamodels may be more accurate in the feasible design space; however, they still may not be as accurate in critical regions, and the corresponding solutions may not be as good as those obtained with metamodels from the integrated design process in E-RCEM.

In cases with clearly defined design goals, the integrated design process in E-RCEM is better than the traditional process of SEED. Otherwise, SEED is better because it helps achieve more accurate metamodels in the feasible design space. SEED is more robust to changes of the design goals in later design stages because the accurate metamodels ensure the achievement of good design solutions no matter how the design goals are changed. The integrated design process in E-RCEM is not as robust as SEED, because the “critical” regions may change as design goals change; the current metamodels

may not be acceptable in new critical regions after design goals are changed, and thus may lead to design solutions that are not as good as those obtained with SEED.

Besides SEED (the traditional process in E-RCEM), the integrated design process in E-RCEM, we can also adopt the hybrid process in E-RCEM, as introduced in Section 6.4.4. In most cases where design goals are defined but still subject to small changes in the future, designers may prefer to the hybrid process in E-RCEM, in which SEED is first used to achieve an acceptable metamodel, then the integrated design process in E-RCEM is adopted to explore for new experimental points and design solutions.

The methods of SEED and E-RCEM give designers more design freedom to deal with limited resources in early design stages. Engineers are able to design and utilize expensive physical experiments or computer simulations in design. Previously, expensive physical experiments are usually used to verify the final design solution, but seldom used to assist the design from early stages; complicated simulations are discouraged to avoid high computational expense. With the SEED and E-RCEM methods, engineers can utilize expensive physical experiments in early design stages with relatively low total cost, and are allowed to develop time-consuming but high-fidelity computer simulations without worrying about their utilities. Engineers are also given the freedom of defining and exploring a large design space without worrying about the nonlinearity and irregularity of responses. They do not need carefully study the responses and conservatively define the design space (as small as possible) before design – they usually do these based on experience – and this experience, or previous information, is

not needed in SEED or E-RCEM, because engineers are able to grasp maximum information with limited available resources.

7.7 A LOOK BACK AND A LOOK AHEAD

In this chapter, the methods of RCEM, SEED (the traditional process in E-RCEM), and the integrated design process in E-RCEM are applied and compared with the example of unit design for an LCA device. Research Questions 2 and 3 are answered and the corresponding hypotheses are verified. These research questions and hypotheses are listed below:

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

Hypothesis 2: Sequential experiments could be designed through analysis of information from data/validation points and metamodels.

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

Hypothesis 3: The processes of metamodeling and robust design space exploration could be integrated through building the information flow from C-DSP to the metamodeling cycle in the Robust Concept Exploration Method.

The LCA design problem is described in Sections 7.1 and 7.2; the problem is initiated, the compromise DSP is formulated, and the actual design solution is obtained with the original computer simulation models in Section 7.2. Single-stage experimental designs are applied and the solutions are obtained with RCEM in Section 7.3. The SEED method (traditional process in E-RCEM) is applied in Section 7.4. The integrated design process in E-RCEM is applied in Section 7.5. The metamodels developed in Section 7.3, 7.4, and 7.5 are compared on their performance in response prediction and achievement of design solutions in Section 7.6. With the LCA design example we observe that more accurate metamodels are developed and better design solutions are achieved with fewer observed points in the methods of SEED and E-RCEM than in RCEM.

Sequential experiments can be designed with SEED through the analysis of information from data and validation points and previous metamodels. Prediction errors are used to adjust entries of the covariance matrices; by maximizing the determinant of the adjusted covariance matrix, new points are identified in regions with fewer observed points and/or large expected prediction errors. Thus after iterations in SEED, more points are allocated at "critical" locations to reduce prediction errors; as a result, the final metamodels are more accurate than those developed with single-stage experimental designs in which information of responses from previous observations is not used as guidance in the identification of new points. SEED ensures the achievement of metamodels that are accurate in the whole design space (or the feasible design space when design constraints are considered in the sequential metamodeling process).

The processes of metamodeling and design space exploration are integrated in the integrated design process in E-RCEM. The information flow from the compromise DSP to metamodeling is built in E-RCEM and this information feedback helps engineers find "critical" regions, or regions of interest, and allocate more points in such regions to ensure the achievement of good design solutions. The "critical" regions in the integrated design process of E-RCEM are those in which design goals are achieved or nearly achieved with current metamodels, and those in which we have large uncertainty with current metamodels. This criterion of "critical" in E-RCEM is broader than that in SEED, which defines the "critical" regions as those in which we have large uncertainty with current metamodels.

SEED and E-RCEM are superior to RCEM in cases with expensive experiments and nonlinear responses; they give engineers more freedom in design with limited resources. However, the additional, relatively high expense in the complicated sequential experimental design process brings trouble for the application of SEED and E-RCEM. To build an automated computer framework for SEED and E-RCEM helps reduce the complexity and expense, and this will be a future direction for research in this dissertation. The methods of SEED and E-RCEM, their plus and minus, their applications, and possible future improvements are further discussed in Chapter 8.

CHAPTER 8

CLOSURE

Through this chapter significant issues addressed in this dissertation are recapitulated. In this dissertation, the Sequential Exploratory Experimental Design (SEED) and the Efficient Robust Concept Exploration Method (E-RCEM) are developed and verified through the study of several single- or two- variable examples and an industrial application of LCA design. Metamodel evaluation, comparison, and selection are also studied as preliminary research for the development of the two proposed methods. SEED and E-RCEM give engineers more freedom in design; they facilitate the development of acceptable metamodels for irregular responses with limited computational or monetary resources and the achievement of satisficing design solutions in a large design space with expensive physical or computer experiments. Our study in this dissertation is brought to a close in this chapter. In Section 8.1, closure is sought by returning to the research questions posed in Chapter 1 and reviewing the answers that have been offered. Then, the resulting contributions are discussed in Section 8.2. Limitations of the research and future work are then described in Section 8.3.

8.1 ANSWERING THE RESEARCH QUESTIONS

As stated in Chapter 1, the principal objective in this dissertation is to develop systematic yet flexible methods that facilitate the development of acceptable metamodels and achievement of satisficing design solutions with limited resources. With the proposed methods engineers can fully utilize expensive physical or computer experiments to grasp important properties of design responses in early design stages. This helps avoid possible expensive re-design processes and thus reduce the development time for new products. The key research question is proposed to motivate our study in this dissertation:

> *How to explore the design space efficiently and effectively for satisficing solutions by employing sequential metamodeling and design space exploration techniques in accordance with the changing design information along the design timeline in early design stages?*

In Section 1.3.2, based on the key question two research questions, Research Questions 1, 2, 3, and 4 are posed for investigation in this dissertation, each of which corresponds to a category of techniques to be studied, developed, and utilized. The four research questions are:

R.Q.1: *How to validate a metamodel with deterministic computer experiments?*

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

R.Q.4: *How to utilize different types of metamodels along the design timeline in accordance with the changing design information? (How to do sequential metamodeling to achieve robust design solutions?)*

Research Question 1 is about metamodel validation techniques with deterministic computer experiments. Research Question 2 is about the sequential identification of data points. Research Question is about the integration of metamodeling and design space exploration processes. Research Question 4 is about metamodel comparison and selection. The relations between research questions, and research questions and proposed methods in this dissertation are presented in Figure 2.5. To address these questions, research hypotheses are introduced and identified in support of achieving the principal objective for the dissertation. In this dissertation, according to the four questions and the corresponding hypothesis, first we prove that the leave-one-out cross-validation is inappropriate and proposed approaches to validate the accuracy of metamodels; then the SEED method is developed based on maximum entropy sampling techniques; metamodel

comparison and selection are studied to improve the SEED method; finally, E-RCEM is developed based on SEED and the compromise DSP to integrate the processes of metamodeling and design space exploration. The elaboration and verification of the research questions and hypothesis provide the context in which the research work has proceeded.

R.Q. 2 and R.Q. 3 are the most important research questions, which lead to the development of SEED and E-RCEM in Chapters 4 and 6, respectively. Researches for R.Q. 1 and R.Q. 4 provide supporting tools for the development and improvement of SEED and E-RCEM.

As described in Section 1.3.2, each of the four research questions is divided into several secondary research questions. Then the corresponding sub-hypotheses are proposed. The secondary research questions operate at a lower level of abstraction in comparison to the research questions posed earlier. In the rest of this section we answer the research questions through revisiting and summarizing our work for the secondary research questions.

8.1.1 Answering Research Question R.Q.1

The first research question, R.Q.1, leads to studies of metamodel validation techniques with deterministic computer experiments, which is a preliminary research for the development of SEED and E-RCEM. This research question is separated into two supporting research questions leading to two studies, one of which is to prove the inappropriateness of the currently widely used method, leave-one-out cross-validation, in deterministic applications, and the other is to develop new approaches of metamodel

validation. Research Question 1, two supporting research questions, and corresponding hypotheses are:

R.Q.1: *How to validate a metamodel with deterministic computer experiments?*

Hypothesis 1: Information from either previous additional validation points is needed in testing the accuracy of a metamodel with deterministic computer experiments.

R.Q.1.1: *Is leave-one-out cross-validation a suitable method of metamodel validation with computer experiments?*

Sub-Hypothesis 1.1: Leave-one-out cross-validation is not an appropriate method of metamodel validation with deterministic computer experiments.

R.Q.1.2: *How to test the accuracy a metamodel in deterministic applications?*

Sub-Hypothesis 1.2: The accuracy of a metamodel could be validated through examining prediction errors at additional validation points.

To answer Research Question 1 and test Hypothesis 1 two tasks need to be accomplished, one is the theoretical study of the inappropriateness of leave-one-out cross-validation in metamodel evaluation, and the other is the development of approaches to test metamodels' accuracy with information from additional validation points. These correspond to studies for the supporting research questions and sub-hypotheses.

Research Question 1.1 and Sub-Hypothesis 1.1 are studied in Sections 3.2 and 3.3. In Section 3.2 with two single-variable examples we observe that leave-one-out cross-validation is insufficient in metamodel validation because it is actually a

measurement for degrees of insensitivity of a metamodel to lost information at its data points, while an insensitive metamodel is not necessarily an accurate one. After careful examination, we point out that there are two causes for this insensitivity: clustering or inappropriately correlated data points. To design space-filling experiments with a sufficient number of data points is one way to prevent an inaccurate and insensitive model, while this cannot assure the validity of the leave-one-out cross-validation method, and this is opposite to our idea of sequential experimental design and may result in great waste of time or money on unnecessary experiments, which will increase the time of bringing new products to market. Our conclusion is verified through empirical study in Section 3.3. Sub-Hypothesis 1.1 is tested and Research Question 1.1 is answered: *Leave-one-out cross-validation is not an appropriate method to validate the accuracy of metamodels.*

Research Question 1.2 and Sub-Hypothesis 1.2 are studied in Section 3.4, in which approaches are proposed for engineers to test the accuracy of metamodels. Several methods are described to help engineers gain insight into the performance of metamodels over the whole design space. Information from additional validation points is utilized in these approaches. The sub-hypothesis is tested and Research Question 1.2 is answered: *The accuracy of metamodels can be tested with information from additional validation points with the developed approaches.*

After answering the supporting research questions and test the sub-hypotheses, we are able to answer Research Question 1. We verify that leave-one-out cross-validation is theoretically inappropriate in metamodel validation and information at

additional validation points are needed. Several preliminary approaches are proposed for engineers to utilize this information of prediction errors at validation points to validate the accuracy of metamodels.

8.1.2 Answering Research Question R.Q. 2

To answer Research Question 2 we focus on the development of accurate metamodels with a sequential experimental design strategy. Three secondary research questions and their corresponding hypotheses are posed:

R.Q.2: *How to design sequential computer experiments (how to select data and validation points sequentially) to get an accurate metamodel?*

Hypothesis 2: Sequential experiments could be designed through analysis of information from data/validation points and metamodels.

R.Q.2.1: *How to measure the information worth of a point?*

Sub-Hypothesis 2.1: The information worth of a point could be measured with entropy.

R.Q.2.2: *How to select validation points to achieve a sequential design of computer experiments?*

Sub-Hypothesis 2.2: Selection of validation points should follow similar rules for selection of data points; information from validation points could be used as guidance in identifying new data points.

R.Q.2.3: *How to utilize information from previous points and metamodels in identifying new data points?*

Sub-Hypothesis 2.3: Through maximizing entropy (as formulated based on Sub-Hypotheses 1.1 and 1.2) we are able to allocate new data points in the design space that yield maximum potential information.

To answer Research Question 2, the method of Sequential Exploratory Experimental Design (SEED) is developed based on *D*-optimal design and maximum entropy sampling. The development of SEED is the foundation of research for Research Questions 3 and 4. To develop the SEED method, we need to accomplish the following tasks: definition and identification of "critical regions" and "information potential of points", consideration of "information potential" in the identification of data points, and selection of validation points. These are done in Chapters 4 and 5.

In Chapters 4 and 5, we verified that with the SEED method, designers are able to add in new data points in the design space with large amount of potential information, and thus accurate metamodels could be achieved efficiently. Information from current data and validation points and metamodels are used as guidance in identifying new data points. Hypothesis 2 is verified; our answer to Research Question 2 is: *Accurate metamodels can be developed through iterations in sequential experimental design with the SEED method, in which information from current data/validation points and metamodels is used as guidance in identifying new data points.*

Research Question 2.1 is answered primarily in Sections 4.3 and 4.4. The application of Bayesian entropy design in SEED in Sections 4.5 and 4.6 supports our idea from Sections 4.3 and 4.4. A clear statement on Research Question 2.1 is presented at the beginning of Section 4.5. Sub-Hypothesis 2.1 is tested; our answer to Research

Question 2.1 is: *The entropy criterion could be used to measure the information worth of a new point.*

Research Question 2.2 is answered in developing and verifying the SEED method in Sections 4.5 and 4.6; Sub-Hypothesis 2.2 is tested. The usage of validation points and observation of prediction errors are necessary steps in the SEED method; it provides the foundation for adjusting the covariance matrix, which is the core of the SEED method. In the SEED method, validation points are added sequentially in iterations; as more and more data and validation points are observed, designers are able to develop more and more accurate metamodels for responses and prediction errors. In Section 4.6, different strategies on selecting validation points are applied and studied in the SEED method. Our answer to Research Question 2.2 is: *Validation points should be added in iterations in sequential experimental design; information from validation points should be used as guidance in identifying future data points.*

Research Question 2.3 is answered and Sub-Hypothesis 2.3 is tested in the development of the SEED method. To be specific, the method of maximum entropy sampling is introduced in Section 4.4; in Section 4.5.2, strategies on how to utilize information from previous points and metamodels are discussed; the mathematical formulations in SEED is developed in Section 4.5.3, which enables designers to design sequential experiments through maximizing entropy; Demonstration and verification is enclosed in Section 4.6. Our answer to Research Question 2.3 is: *Information from current data/validation points and metamodels could be used to build the adjusted*

covariance matrix; new data points could be identified through maximizing the determinant of the adjusted covariance matrix.

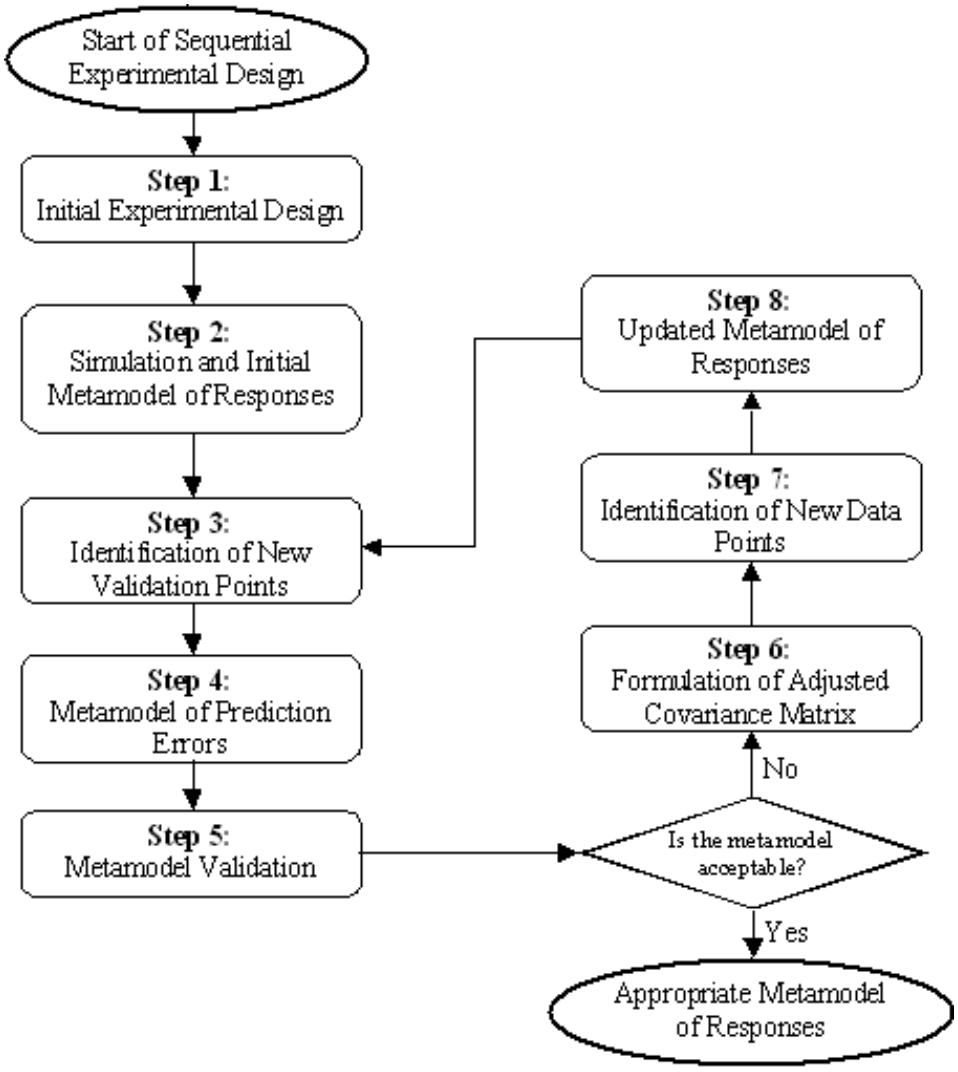


Figure 8.1 Flowchart of the Sequential Exploratory Experimental Design Method

The flowchart of the SEED method is illustrated in Figure 8.1. The SEED method can be used to replace the metamodeling process in RCEM, as illustrated in

Figure 8.2. Also, as shown in Chapter 6, the application of SEED method in E-RCEM helps form the traditional process and hybrid process of the E-RCEM method. In Chapters 4, 5, and 7, with several simple examples and a multivariable, multi-response example, it is shown that more globally accurate metamodels can be developed with fewer experiments and better design solutions can be achieved with the SEED method than with traditional methods (such as RCEM). In cases with expensive experiments and/or irregular responses, SEED helps designers grasp important response properties in the whole (or feasible) design space with low cost, and thus enable engineers to fully utilize the approximation-based design strategy and reduce the time of introducing new products to the market.

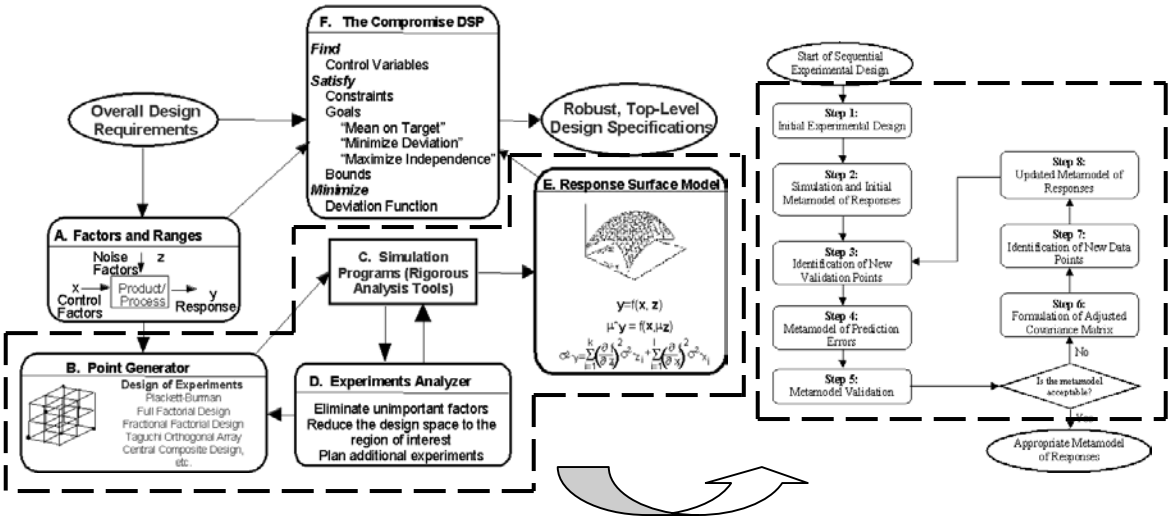


Figure 8.2 Application of SEED in RCEM

8.1.3 Answering Research Question R.Q. 3

To answer Research Question 3 we study the integration of processes of metamodeling and design space exploration. Three secondary research questions and their corresponding hypotheses are posed. The research question, supporting research questions, and corresponding hypotheses are:

R.Q.3: *How to integrate the processes of metamodeling and robust design space exploration?*

Hypothesis 3: The processes of metamodeling and robust design space exploration could be integrated through building the information flow from C-DSP to the metamodeling cycle in the Robust Concept Exploration Method.

R.Q.3.1: *How to design sequential experiments with consideration of design constraints?*

Sub-Hypothesis 3.1: Consideration of design constraints could be incorporated in the metamodeling process through construction irregular design spaces.

R.Q.3.2: *How to reduce the design space with information from previous metamodeling and design space exploration?*

Sub-Hypothesis 3.2: Design space could be reduced through analysis of the information from previous metamodels.

R.Q.3.3: *How to do sequential metamodeling with consideration of design goals?*

Sub-Hypothesis 3.3: Design goals can be taken into consideration in metamodeling by formulating influential factors with the compromise DSP and using them in maximum entropy sampling.

To answer Research Question 3, the Efficient Robust Concept Exploration Method (E-RCEM) is developed in Chapter 6, and the screening of unimportant design variables is built in the SEED method in Chapter 5. The integrated design process in E-RCEM is demonstrated and verified with a single-variable example in Chapter 6 and in the LCA unit design in Chapter 7.

E-RCEM is developed through the conduction of two tasks: consideration of design constraints in metamodeling, and consideration of design goals in metamodeling. E-RCEM is developed based on the Robust Concept Exploration Method (RCEM), the method of Sequential Exploratory Experimental Design (SEED), and the Compromise Decision Support Problems (C-DSP). In Chapters 6 and 7, we verified that with the integrated design process in E-RCEM, designers are able to incorporate considerations of metamodel accuracy and achievement of design goals in the experimental design and metamodeling process. New points are identified in regions where design goals are to be achieved or large prediction errors exist. With this integrated design process in E-RCEM (or the *metamodeling **for** design space exploration* approach), designers are able to achieve better design solutions with less time and money spent on expensive computer or physical experiments. Hypothesis 3 is verified; our answer to Research Question 3 is: *Better design solutions can be achieved with fewer experiments by integrating the processes of metamodeling and design space exploration; this integrated design process is realized in E-RCEM, in which information about metamodel uncertainty and achievement of design goals is used as guidance in identifying new points in sequential metamodeling*.

Research Question 3.1 is answered primarily in Section 6.2. Sequential metamodeling with constraints on design variables is studied in Section 6.2.1, and sequential metamodeling with constraints on responses is studied in Section 6.2.2. In this section we show that design constraints can be taken into consideration in the SEED method and the integrated design process in E-RCEM. After taking design constraints into consideration, the design space is usually irregular; with SEED or E-RCEM, new points will be identified only in the reduced irregular feasible design space, and this helps save time and money spent on experiments wasted in infeasible regions. Our answer to Research Question 3.1 is: *Design constraints can be taken into consideration to define an irregular design space, and SEED or E-RCEM can be used to identify new points in the reduced irregular feasible design space*.

Research Question 3.2 is answered in Section 5.4 and illustrated in Section 5.5. The usage of RS metamodels at the very early stages of metamodeling helps identify and screen unimportant design variables. Another way to reduce the design space is to reduce the ranges of design variables, which is not studied and incorporated with SEED in researches in this dissertation; thus this study here is preliminary and future research is needed to improve the design space reduction approaches that are built in the methods of SEED and E-RCEM. Our answer to Research Question 3.2 is: *The design space can be reduced by eliminating unimportant design variables through the analysis of information from previous data points and metamodels*.

Research Question 3.3 is answered in Section 6.3. Based on the compromise DSP, the degree of achievement of design goals at candidate points can be formulated

and scaled in [0,1]; a value close to 0 means that design goals are hardly achieved, and a value close to 1 means that design goals are almost achieved at this point. Usually we preset a target value for the design goal, and once this target value is met or exceeded, we set the degree of achievement of design goals to be 1. This quantitative expression of degree of achievement of design goals can be used in the adjustment of covariance matrices in maximum entropy sampling, and "drag" new points to regions where design goals are met or almost met. Our answer to Research Question 3.3 is: *The degree of achievement of design goals at a particular point can be quantitatively formulated with the compromise DSP and used as an influential factor in SEED or E-RCEM*.

The flowchart for the Efficient Robust Concept Exploration Method is illustrated in Figure 8.3 and Figure 8.4. With the integrated design process in E-RCEM, engineers identify points sequentially in regions of interest, thus are able to develop metamodels with more local accuracy in critical regions and achieve better design solutions than SEED and RCEM. In cases with expensive experiments and irregular responses, E-RCEM helps achieve efficient and effective designs with affordable cost, which may not be accomplished with traditional approximation-based design methods such as RCEM.

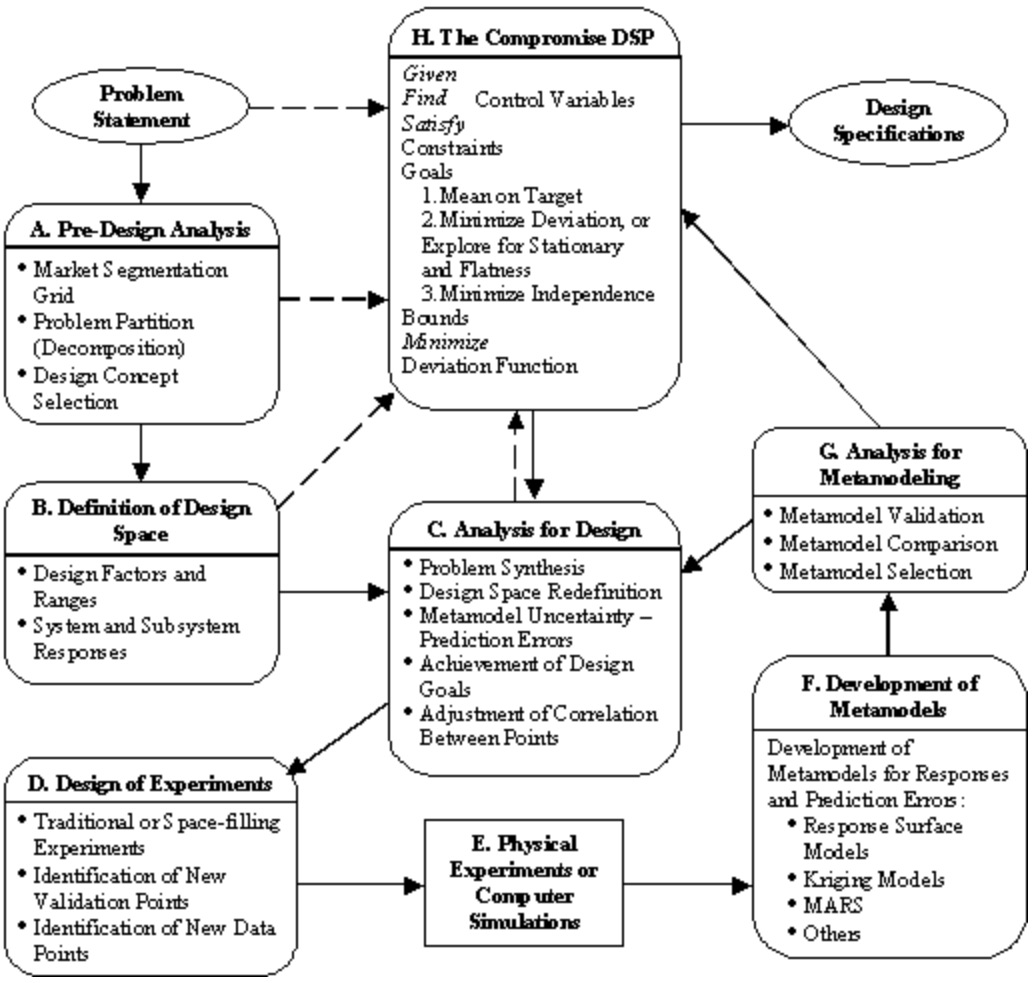


Figure 8.3 Flowchart of the Efficient Robust Concept Exploration Method (I)

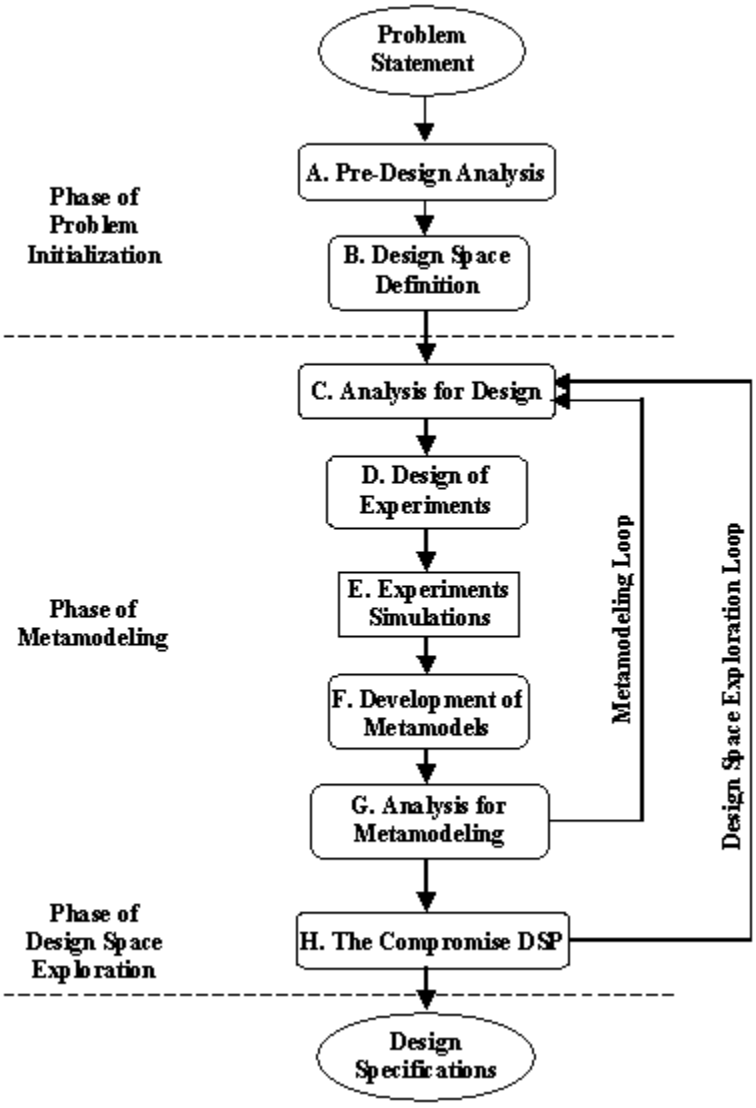


Figure 8.4 Flowchart of the Efficient Robust Concept Exploration Method (II)

8.1.4 Answering Research Question R.Q. 4

To answer Research Question 4 we study the comparison and selection of different types of metamodels in the SEED and E-RCEM processes. Three secondary

research questions and their corresponding hypotheses are posed. The research question, supporting research questions, and corresponding hypotheses are:

R.Q.4: *How to utilize different types of metamodels along the design timeline in accordance with the changing design information?*

Hypothesis 4: Different types of metamodels should be used at different design stages in accordance with different requirements of design.

R.Q.4.1: *How do different types of metamodels perform in engineering design?*

Sub-Hypothesis 4.1: Different types of metamodels have their strong and weak points.

R.Q.4.2: *How to select different types of metamodels at different design stages?*

Sub-Hypothesis 4.2: As design evolves, more complicated types of metamodels should be used to help yield good approximations with more computation time and efforts.

In this dissertation we consider three types of metamodels, the response surface (RS) metamodels, kriging, and multivariate adaptive regression splines (MARS). R.Q.4.1 is studied and answered in Sections 5.2 and 5.4. A comparison between kriging and MARS metamodels is done in Section 5.2 with some interesting observations. The comparison between RS and kriging metamodels has been done in previous work in (Simpson, 1998) and (Lin, 2000), and comparisons between more types of metamodels could be a future work of this dissertation. In our studies we observe that both kriging

and MARS have their strong and weak points; kriging metamodels may not perform appropriately when the properties of the response surface change greatly (i.e., highly nonlinear in some regions while flat in others), and MARS metamodels may meet problems in deterministic applications because they smooth the data and thus the predicted values at data points may not be accurate. Hypothesis 4.1 is tested, and as an answer to Research Question 4.1, a summary on comparison between RS, kriging, and MARS metamodels is presented in Table 8.1. Particularly, in Table 8.1 we see that *MARS works better than kriging in modeling irregular responses, while kriging has a native mathematical connectivity to SEED that MARS lacks*.

Table 8.1 Plus and Minus of Different Types of Metamodels

	RS (Regression)	**Kriging**	**MARS**
1. Mathematical complexity	Simple	Complicated	Complicated
2. Computation time	Short	Long	Medium
3. Problem size: # of design variables and # of data points	Large, Medium, and Small Problems	Small Problems	Medium and Small Problems
4. Metamodel accuracy	Low	High	High
5. Loyalty to data	No	Yes	No, with very small bias
6. Ability to model irregular responses (highly nonlinear or flat in different regions)	No	Yes, but only when with lots of data	Yes
7. Suitable for existing screening techniques	Yes	No	Yes
8. Preference to specific experimental designs	Yes	Yes	No
9. Mathematical connectivity to SEED (adapted maximum entropy sampling)	No	Yes	No

Based on the studies in Section 5.2, the SEED method is extended in Section 5.3 by utilizing both kriging and MARS metamodels. This helps answer R.Q.4.2. Kriging and MARS may be appropriate, or, on the other hand, inappropriate, in different situations; thus we recommend that both be used to develop metamodels in sequential experimental design and metamodeling. Designers could make decisions only after building the metamodels and observing their performance. A recommendation on how to use kriging and MARS metamodels is described in Section 5.3.

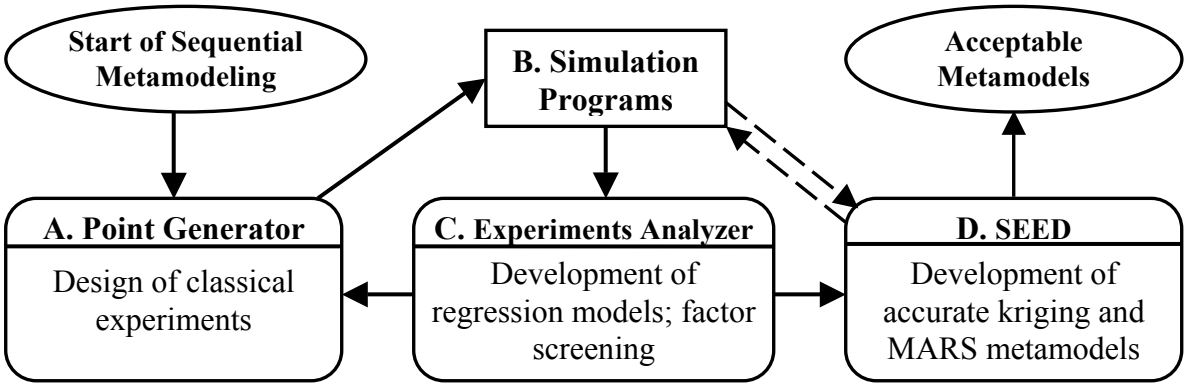


Figure 8.5 Framework of Sequential Metamodeling

R.Q.4.2 is further studied and answered in Sections 5.4 and 5.5, in which an approach for sequential metamodeling is developed and illustrated with an engineering example. The framework for the approach of sequential metamodeling is presented in Figure 8.5. This sequential metamodeling approach is incorporated in the method of E-RCEM in Chapter 6. Hypothesis 4.2 is tested and Research Question 4.2 is answered: *Response surface metamodels should be used at the beginning of design to help gain knowledge of responses and screen unimportant design variables; MARS and kriging*

should be used in later stages with SEED to help gain accurate interpretations of irregular responses.

Answers to research questions, tasks, and verification of hypotheses are presented in this section. This discussion leads to the research contributions of this dissertation, which will be summarized in the next section.

8.2 ACHIEVEMENTS: REVIEW OF RESEARCH CONTRIBUTIONS

The expected contributions of this dissertation have been stated in Section 1.3.3 and Section 2.1. Here is a revisit of the achievements and contributions of the research in this dissertation.

Contributions Related to the Sequential Exploratory Experimental Design Method:

- The development of the SEED method for sequential experimental design. The SEED method is developed in Chapter 4 and then improved with the utility of various types of metamodels in Chapter 5. The SEED method facilitates the development of globally accurate metamodels with limited number of observations in the whole design space. Its utility has been verified with several examples.
- An approach to calculate and incorporate prediction errors in the identification of regions of interest and data points. This is done in Section 4.5.2. The

usage of two groups of points to calculate and incorporate prediction errors in design is an original work of this dissertation.

- Two approaches to modify the mathematical formulations of entries of the covariance matrix in maximum entropy sampling. This is done in Section 4.5.3. With the two developed approaches, the information of prediction errors can be mathematically taken into consideration in the identification of new points, thus this work helps solid the idea of sequential experimental design.

Contributions Related to the Efficient Robust Concept Exploration Method:

- The development of the Efficient Robust Concept Exploration Method. E-RCEM is developed in Chapter 6 and further improved with multi-variable and multi-response examples in Chapter 7. E-RCEM facilitates efficient and effective design space exploration for robust design solutions.
- The integration of traditionally separated processes of metamodeling and design space exploration. The idea of consideration of design constraints and design goals in the metamodeling process is innovative, and is realized in E-RCEM based on the SEED algorithm and the compromise DSP. This is the core of E-RCEM, and studied throughout Chapters 6 and 7.
- An approach to consider design constraints and design goals in the identification of regions of interest and new data points. This is done in

Sections 6.2 and 6.3. This work supports the integration of processes of metamodeling and design space exploration.

- A preliminary design space exploration heuristic for designers with expensive physical and computer experiments. The integrated design process in E-RCEM (one of the three possible ways to apply E-RCEM, as stated in Section 6.4.4) can be viewed as a design space exploration heuristic for cases with expensive experiments. Although only examples with computer experiments are used in this dissertation, it is expected that E-RCEM is also suitable for designs with expensive physical experiments.

Contributions Related to Metamodel Evaluation:

- A study shows that leave-one-out cross-validation is theoretically inappropriate for metamodel validation. This is done in Section 3.2. This is an original work of this dissertation. This conclusion is also supported with empirical studies in Section 3.3.
- Preliminary approaches for engineers to validate metamodels' accuracy with information at additional validation points. This is done in Section 3.4. The developed approaches, though may be complicated and somewhat non-solid in applications, help designers gain knowledge of the responses and support designers' decisions in metamodel validation.

Contributions Related to Metamodel Comparison and Selection:

- A comparison between kriging and MARS and an observation of kriging's limitation in modeling irregular responses. This is done in Section 5.2. Previously kriging and MARS are only compared with space-filling experiments. The comparison of kriging and MARS with unevenly spread points from sequential experiments in this dissertation is original.
- An approach in which three types of metamodels are used sequentially along the design timeline to facilitate effective and efficient exploration of satisficing design solutions. In Section 5.3 recommendations are made on how to use MARS and kriging in sequential experimental design. In Section 5.4, a sequential metamodeling approach is proposed in which response surface models, kriging, and MARS are utilized in the metamodeling process.

The value of these contributions lies in the worth to be either an addition to the fundamental knowledge of the field or a new and better interpretation of the facts already known. Based on this criterion contributions of this dissertation are classified and listed in Table 8.2. The most important contributions of this dissertation are the development of the methods of SEED and E-RCEM, which are all original in this dissertation. These contributions represent an addition to the fundamental knowledge of the field.

Some of the other contributions, e.g., the comparison of kriging and MARS in modeling irregular responses, and the verification that leave-one-out cross-validation is theoretically inappropriate for metamodel validation, are also original in this dissertation and represent an addition to the fundamental knowledge. As to other contributions,

previous work is available, while the studies in this dissertation are from different viewpoints.

Table 8.2 Contributions of Studies in this Dissertation

Contributions	Addition to the Fundamental Knowledge	Better Interpretation of Existing Ideas
SEED	Yes. A new method for sequential experimental design and metamodeling	Information theory
Calculation and incorporation of prediction errors in metamodeling	Yes	
Mathematical formulations to adjust entries of the covariance matrix	Yes	D-optimal design Maximum entropy sampling
E-RCEM	Yes. A method with integrated processes of metamodeling and design space exploration	
Integration of processes of metamodeling and design space exploration	Yes	
Incorporation of design goals and constraints in metamodeling	Yes	
A preliminary optimization heuristic for engineers	Yes	Efficient Global Optimization
Verification of the inappropriateness of leave-one-out cross-validation	Yes	Simpson, 1998
Preliminary approaches to validate metamodels		Simpson, 1998; Jones, et al., 1998
A comparison of kriging and MARS in SEED	Yes	
An approach to utilize three types of metamodels along the design timeline		Response Surface Methodology

With the methods of SEED and E-RCEM, engineers are able to develop more accurate metamodels for irregular responses with limited resources and achieve better design solutions in a large design space with expensive computer or physical experiments than they do with traditional methods like RCEM. This gives engineers the freedom of using expensive experiments to analyze irregular responses in large design spaces in early design stage, thus enables the full utilization of the approximation-based design strategy in industrial applications. However, there are several limitations in our studies in this dissertation. In the next sections, after identifying the limitations of our work for this dissertation and summarizing observations in our study, we point out some possible directions for future work.

8.3 CRITICAL REVIEW

Answers to research questions are summarized in Section 8.1 and the contributions of studies in this dissertation are listed in Section 8.2. In this section, insights obtained from the studies, limitations of the developed methods, and recommendations on how to use the methods in design are presented in four sub-sections corresponding to four research questions in this dissertation. Studies of metamodel evaluation are summarized in Section 8.3.1. Studies of metamodel comparison and selection are summarized in Section 8.3.2. Studies of Sequential Exploratory Experimental Design (SEED) are summarized in Section 8.3.3. Studies of the integrated

design process in the Efficient Robust Concept Exploration Method (E-RCEM) are summarized in Section 8.3.4.

8.3.1 Metamodel Evaluation

The outcome of studies of metamodel validation is documented in Chapter 3. First we studied the performance of leave-one-out cross-validation method in validating metamodels with deterministic computer experiments. With several simple functions we illustrated that cross-validation is an insufficient method, thus to use additional validation points becomes essential in metamodel validation. Then we describe some preliminary methods on how to utilize the information from additional validation points.

The reason why leave-one-out cross-validation is insufficient in metamodel validation is that it is actually a measurement for degrees of insensitivity of a metamodel to lost information at its data points, while an insensitive metamodel is not necessarily accurate. There are two causes for this insensitivity: clustering or inappropriately correlated data points. To design space-filling experiments with a sufficient number of data points is one way to prevent an inaccurate and insensitive model, while this cannot assure the validity of the leave-one-out cross-validation method. We recommend starting with space filling experimental designs in the development of metamodels in engineering applications.

The conclusion here does not mean that previous applications with leave-one-out cross-validation are necessarily wrong. When the original actual function is not highly nonlinear (or the design space is not very large) and there are enough data points spreading all over the design space, the danger of having clustering or inappropriately

correlated data sets is small. However, the success of leave-one-out cross-validation in those examples is dependent on particular cases; real-world applications are usually more complicated and cannot meet the requirements mentioned above. Thus to use additional validation points are necessary in metamodel validation.

Though one important benefit of using metamodels is to save expenses on experiments, to add in additional validation points, which eventually increases time and effort on computer simulations, does not hurt the importance of metamodeling very much. First, in computer experiments, moderate increases of computational expenses are usually affordable with fast computers in a distributed design environment. Second, to use metamodels not only helps us save experimental expenses but also integrates simulation codes from different disciplines to give insight into the relationships between input variables and output responses. Third, and maybe the most important, with the SEED method and the E-RCEM method developed in this dissertation, designers are able to do more observations in the design space with relatively low expense, which makes it possible to utilize validation points with expensive experiments.

Several methods are proposed in Chapter 3 to help engineers gain insight into the performance of metamodels over the whole design space. However, these methods are not very solid and sometimes they are too complex to use; future studies on metamodel valuation are needed. One unsolved problem in model validation is how to select validation points, e.g., how many validation points should be used, and how to allocate these points. Validation points are identified and used in the methods of SEED and E-RCEM; however, the selection of validation points in these methods are for the

achievement of better metamodels, not for the validation of the current metamodel. Thus the development of new strategies to validate a metamodel (either with or without validation points) is future work of this dissertation. Currently, without better methods, we recommend the method with Equations (3.8) and (3.9) because of its simplexity.

8.3.2 Metamodel Comparison and Selection

The outcome of studies of metamodel comparison and selection is documented in Chapter 5. First we studied the performance of kriging and adaptive regression splines in modeling the actual responses with unevenly located data points. Our observations show that in cases with irregular responses (highly nonlinear in some regions while flat elsewhere) and unevenly located data points (usually a result from sequential experimental designs), kriging may work abnormally. The reason lies in the universal usage of a constant value of θ in one dimension in our kriging algorithm; designers meet difficulty when trying to model highly nonlinear surfaces and flat surfaces with the same θ. Univariate or multivariate adaptive regression splines metamodels perform well in modeling irregular responses.

Although kriging might not be appropriate in cases with irregular responses, it has some desirable properties that adaptive regression splines metamodels do not have. First, it yields the exact true value at data points, while adaptive regression splines may have small deviations. This is important in deterministic computer applications. Second, in SEED, values of θ from previous kriging metamodels can be used as a reference in formulating and adjusting the covariance matrices. With this information designers are

able to distinguish design variables with high uncertainties in response prediction, and thus more future points will be automatically identified in these dimensions with the SEED process. When adaptive regression splines metamodels are used we cannot get such information as easily as with kriging.

Based on these observations, we propose to utilize both kriging and adaptive regression splines in SEED. Usually both kriging and adaptive regression splines are used to develop metamodels; careful examinations for abnormal performance are necessary. We prefer to use kriging metamodels when abnormal behaviors are not detected.

The implementation of kriging and adaptive regression splines with SEED is also studied in Chapter 5. It is illustrated that with SEED, designers are able to develop accurate kriging or adaptive regression splines metamodels. In the examples, kriging is first used to develop a metamodel of responses then replace by the univariate (or multivariate) adaptive regression splines because of abnormal performance in the design space. The univariate (or multivariate) adaptive regression splines metamodels work well in modeling both responses and prediction errors.

A limitation of the studies of metamodel comparison and selection is that only three types of metamodels are studied in this dissertation. Other types of metamodels, e.g., the artificial neural networks (ANN) or wavelets, need to be studied and compared with the kriging and MARS metamodels in sequential metamodeling processes.

A sequential metamodeling approach is proposed in Chapter 5, which incorporates the factor-screening techniques in the Response Surface Methodology

(RSM) and the SEED method. The research surface metamodels (regression polynomials) are used to identify and screen unimportant design variables, and then the SEED method is applied to help develop accurate metamodels. It should be noted that the design space reduction approach here is very preliminary; the development of new methods to reduce the number and ranges of design variables should be future work for this dissertation. The methods of SEED and E-RCEM, together with other references (e.g., the fuzzy c-Means clustering technique), can be used in the development of such design-space-reduction techniques.

8.3.3 Sequential Exploratory Experimental Design

One of our main contributions is the development of the Sequential Exploratory Experimental Design (SEED) method in Chapter 4. SEED is based on Bayesian entropy sampling by removing the stationary assumption and introducing correction factors in the calculation of correlations between points. With the SEED method new points are allocated in "crucial" regions (which are with large prediction errors) and as a result more accurate metamodels can be developed with limited number of observed points. In cases with computer experiments, SEED helps save time and effort spent on expensive computer simulations. Though SEED was initially developed for designing computer experiments, it can also be applied in physical experiments and may bring considerable monetary benefits.

To develop the SEED method, the inappropriateness of "locating new points in regions with more local optimums" is illustrated, and the criterion of expected prediction errors is proposed and applied to help identify regions of interest where candidate points

are expected to have more potential information. In order to calculate the expected prediction errors, an approach is proposed in which two groups of points are used to help grasp the information of responses and modeling errors. The usage of two groups of points also facilitates the selection of appropriate sets of points when only a portion of data points are required in building the final metamodel.

Leave-one-out cross-validation is widely used to model prediction errors. We claim that the "cross-validated prediction errors" do not necessarily reflect "actual prediction errors", but leave-one-out cross-validation can still be used in SEED, especially when there are strict limits on the number of total observed points. The application of leave-one-out cross-validation in SEED is future work for this dissertation.

On relaxing the stationary assumption, two methods are proposed to adjust the covariance matrix, as stated in Section 4.5.3. Prediction errors are taken into consideration in the mathematical formulation of entries of the covariance matrix. There may be different ways to incorporate expected prediction errors in sampling (formulation of entries of the covariance matrix).

Kriging metamodels are used in Chapter 4 to illustrate the SEED method. However, the SEED method is not developed for kriging and can be used with other types of metamodels. The MARS metamodels are applied in SEED in Chapter 5. More types of metamodels will be studied and applied in SEED, which is future work of this dissertation.

In sequential experimental design, more time and effort is spent on the comprehensive steps and iterations with SEED than with single-stage experiments. To

develop an automated SEED routine with little human interface will help save significant expense on human operations. In SEED, most computation time is spent on the entropy optimization steps (Steps 3 and 7); adopting faster local optimization techniques (e.g., as in Currin, et al., 1991) helps save computation time. Future work is needed in studying the computational efficiency of the SEED method.

In the SEED method, the numbers of initial data points and validation points and those of new points added in each iteration are determined arbitrarily by the designers. This decision may be based on previous knowledge of the responses in the design space. When previous knowledge of the responses is unavailable, we recommend starting with a factorial (or fractional factorial) experimental design. The central point may be added to help designers grasp more information at the beginning of the SEED process. In the examples in this dissertation, the number of new points added in each iteration is set to be the same as the number of design variables (n_v) or one less than the number of design variables ($n_v - 1$). However, enough number (at least 2 or 3) of iterations in SEED should be ensured so that information at previous points can be fully utilized; this affects designers' decisions on how many initial points and new points should be used in SEED. Future theoretical or empirical studies are needed to compare different strategies and also observe the flexibility of the SEED method.

The mathematical formulations in SEED in this dissertation are not necessarily perfect. Values of parameters λ and θ in SEED are determined by designers. When kriging metamodels are used, values of θ from kriging metamodels can be used in the formulation of entries of the covariance matrix in later iterations. When no kriging

metamodels are developed (e.g., at the beginning of the SEED process), a large number can be assigned to θ; in such cases usually we set all θ's as 10. The parameter λ is used to balance the considerations of "relative distance between the candidate point and current points" and "prediction errors at the candidate point" in the formulation of entries of the adjusted covariance matrix. Usually we set λ as 2; as design develops and more accurate metamodels are obtained, we can use smaller values for λ, e.g., 1.5. Future theoretical or empirical studies are needed to compare different strategies and also observe the flexibility of the SEED method.

8.3.4 The Efficient Robust Concept Exploration Method

The Efficient Robust Concept Exploration Method (E-RCEM) is developed in Chapter 6 based on RCEM and SEED. The E-RCEM method can be used in three ways, the traditional process, the integrated design process, and the hybrid process. In the traditional process the two processes of metamodeling and design space exploration is separated, and the E-RCEM method becomes the SEED method because globally accurate metamodels are pursued. In the integrated design process of the E-RCEM method, regions of interest are those with fewer points, large prediction errors, and also points where design goals are achieved and constraints are satisfied. The two processes of metamodeling and design space exploration are integrated; in other words, we realize a process of metamodeling ***for*** design space exploration. In the integrated design process, the focus is to achieve a good design solution; globally accurate metamodels are not pursued. The hybrid process is a combination of the traditional process and the

integrated design process, and is recommended in complicated applications with multiple design variables and responses. In the hybrid process, the traditional process is first used to help develop metamodels with acceptable accuracy, and then the integrated design process is applied to help achieve design solutions efficiently and effectively.

From the viewpoint of metamodeling, the traditional process of SEED is better than the integrated design process in E-RCEM because it yields a more accurate metamodel in the whole design space; while from the viewpoint of design space exploration, the integrated design process in E-RCEM is better than the traditional process of SEED because it yields a metamodel with higher local accuracy in critical regions and thus possibly a better design solution. In cases with expensive computer or physical experiments, both the traditional process with SEED and the integrated process in E-RCEM help develop better metamodels with less time and money, and thus ensure better design solutions than traditional experimental designs and design space exploration approaches. When design goals are not well defined at the beginning of design (e.g., in some cases the relative priorities of design goals may change greatly during the design phrase) and it is hard to address this uncertainty, designers may prefer to use SEED to develop globally accurate metamodels. When design goals are clearly defined, designers may prefer to use the integrated design process of metamodeling and design space exploration in E-RCEM to achieve better design solutions faster. In most cases where design goals are defined but still subject to small changes in the future, designers may prefer to use SEED first to achieve an acceptable metamodel, then use the integrated design process in E-RCEM to explore for new experimental points and design solutions.

In the integrated design process in E-RCEM, the correction parameter γ is introduced to balance the consideration of design goals, prediction errors, and relative distances. The selection of γ is arbitrary in this dissertation; usually we set it as 2 at the beginning of the E-RCEM process, and as design develops, smaller values of γ (e.g., 1.5) may be adopted. Future studies are needed on the determination of values for γ, as well as other parameters inherited from SEED.

Design constraints are considered in the metamodeling process in E-RCEM, thus designers usually deal with irregular feasible design spaces. Only convex design spaces are considered in this dissertation. This provides a reference for the identification of the initial design space in engineering design. However, in complicated cases with a lot of design variables, responses, and constraints, it may be difficult to identify and use the feasible design space as the initial design space. In such cases designers can use a hypercube design space that is large enough to enclose all possible-to-succeed regions based on designers' previous knowledge. To develop formal methods to define the initial design space and re-define (design space shift and reduction) the design space is future work for this dissertation.

8.4 FUTURE WORK

In carrying out the research that has led to the contributions reviewed in the previous section, many lessons have been learned. The first is that there is always no end for research. The more we study, the more we learn what we need to learn. Though from

some aspects we could say that the study in this dissertation is complete by itself, we could always find limitations here and there in our research; this awareness of limitations, most possibly, leads to future improvements and achievements. Thus in this section we list our possible future work below after having identified the limitations of our study in Section 8.3.

A Flexible Computer Framework to Realize SEED and E-RCEM

Processes in the method of SEED and E-RCEM are complicated for engineers who lack knowledge of maximum entropy sampling. The initialization and realization of the SEED or E-RCEM processes in iSIGHT require tedious manual operations. These two factors mentioned above limit the application of SEED and E-RCEM in academic research and industrial applications. An automated computer framework to realize the SEED and E-RCEM process will solve the two problems and ensure the utility of the methods developed in this dissertation.

As illustrated in Chapters 4, 5, 6, and 7, automated processes of single steps in SEED and E-RCEM have been realized in iSIGHT. However, the formulation of initial input and output files for these steps is still done manually in this dissertation. It is not technically difficult to realize an automated initialization process to formulate the information flow between computer codes used in steps of the SEED and E-RCEM method.

In addition to the automated process, a user-friendly interface is also desired to make SEED and E-RCEM easy to implement. It is also desired that this automated, user-

friendly system should run in a distributed environment, in which engineers from different geological locations can work together during the SEED and E-RCEM design processes.

An Design Space Exploration Heuristic for Engineers with Expensive Experiments

The E-RCEM method developed in this dissertation has great potential to be developed into an optimization heuristic for engineers to use in real-world industrial applications. Current the development of new products industrial applications is still much dependent on designers' experience, partly because of the lack of effective yet efficient analytical, synthetic, and optimization tools for design in early stages. As discussed in Chapter 1 and Section 2.1, designers' freedom is confined and the approximation-based design strategy is not fully utilized in industrial applications because of the expensive experiments, large design spaces, and irregular responses. The method of E-RCEM addresses these problems and facilitates the fast and effective analysis of responses and helps achieve satisficing design solutions with very few runs of the expensive analysis codes (or physical experiments).

E-RCEM is a preliminary design space exploration heuristic that is suitable for engineers in industrial applications. We illustrated its utility with the LCA unit design in Chapter 7. However, there are still many aspects of E-RCEM that can be improved or modified. Besides the automated computer framework, work is needed on the comparison and refinement of the mathematical formulations in E-RCEM. As discussed in Section 6.3.2, based on our idea of incorporating degrees of achievement of design

goals in metamodeling, several possible mathematical formulations are proposed to adjust entries of the covariance matrix in the identification of new points, and finally we adopt only one of them and apply in the E-RCEM process. Future research is needed to study possible mathematical formulations, not limited to those presented in this dissertation, and find out the best one (or ones) with either sound theoretical foundation or good empirical results. Another topic to be considered is the application of E-RCEM in cases with discrete or concave design spaces. The performance of E-RCEM in such cases is not studied in this dissertation, and it is expected that modifications and improvements of E-RCEM be needed in such problems.

It is expected that an optimization heuristic can be developed based on E-RCEM. The proposed optimization heuristic will facilitate the study and achievement of good solutions for engineers with complex responses and expensive experiments in industrial applications.

Design Space Reduction

There are two ways to reduce a design space, one is to screen out unimportant design variables (reduce the dimensionality), and the other is to reduce the ranges of design variables. In this dissertation, the factor-screening technique in the Response Surface Methodology is adopted and used in the sequential metamodeling approach in Chapter 5. However, this technique is only suitable for response surface models, and lacks theoretical foundations in deterministic computer experiments. Thus future studies

are needed on how to reduce a design space, and how to incorporate the proposed approach in the SEED and E-RCEM processes.

As described in the last paragraph, there are two directions in the study of design space reduction. To identify and remove unimportant design variables, methods are developed in (Myer and Montgomery, 1995; Box and Draper, 1969; Balabanov, et al., 1999; Giunta, et al., 1996; Welch, et al., 1992), which can be used as a basis for the proposed research. Kriging and MARS metamodels also provide qualitative information of the relative importance of design variables, thus it is possible to develop an approach for the identification of unimportant design variables within the sequential metamodeling process.

An alternative way to reduce the design space is to reduce the ranges of design variables. Chen and her co-authors developed heuristics to lead the surface refinement to a smaller design space (Chen, et al., 1997). The adaptive RSM (ARSM) method is developed to systematically reduces the size of the design space by discarding portions of it that correspond to objective function values larger than a given threshold value at each modeling-optimization iteration (Wang, 2001; Wang, 2003). Move limit strategies or trust regions are often used to identify "meaningful" design spaces (Wujek and Renaud, 1998a; Wujeck and Renuad, 1998b; Alexandrov, et al., 1998; Rodriguez, et al., 1997). Wang and Simpson propose an intuitive methodology to systematically reduce the design space to a relatively small region by incorporating the fuzzy c-Means clustering technique in the metamodeling process (Wang and Simpson, 2004). All these provide good foundations for our proposed research on design space reduction; I expect the

improvement and incorporation of some of the methods above (e.g., the fuzzy clustering design space reduction approach) with SEED and E-RCEM.

Comparison and Utilization of More Types of Metamodels

In this dissertation only three types of metamodels, the response surface (RS) model, kriging, and MARS, are studied and applied in the SEED and E-RCEM methods. A future research direction is to study the performance of other types of metamodels, e.g., the artificial neural networks (ANN) and wavelets, in metamodeling and design space exploration with SEED and E-RCEM.

Metamodel Evaluation Methods

New approaches are needed to evaluate the metamodels with deterministic computer experiments since the preliminary metamodel validation approaches developed in Chapter 3 are not very solid and easy to use. To validate the metamodels with additional points, engineers should decide the number and location of these validation points. This should be accomplished with the improvement and application of SEED and E-RCEM.

To validate the metamodels without additional points, criteria must be developed to distinguish "good" metamodels from "bad" ones. One possible criterion is the "smoothness" of the responses. Approaches to quantify such criteria are needed.

Application of Cross-Validation in SEED and E-RCEM

Leave-one-out cross-validation is proved to be inappropriate as a method to validate the accuracy of metamodels in Chapter 3 in this dissertation; however, it helps designers judge whether a metamodel is robust to the lost of information due to removal of particular data points. This means that if a metamodel has small leave-one-out cross-validation errors, its performance in response prediction will not be greatly affected by removing any of its data points.

In SEED and E-RCEM, two groups of points are used to test and supplement each other. Prediction errors are calculated with this information and then entries of the covariance matrix are adjusted. It is possible that the leave-one-out cross-validation errors can be used to adjust the entries of the covariance matrix; in such cases only one group of points are needed and a lot of operational and computational expense can be saved. One possible shortcoming of such a strategy is that large leave-one-out cross-validation errors tend to appear close to existing data points. To use k-folder cross-validation may be helpful to avoid such problems; in fact, the two-group-point strategy used in this dissertation is a specific situation of the k-folder cross-validation. Future studies are needed on the possible utilization of cross-validation in SEED and E-RCEM.

Improvement of SEED and E-RCEM

The mathematics used SEED and E-RCEM in this dissertation is not perfect. As discussed in Chapters 4 and 6, there are many ways to adjust entries of the covariance matrix, while we only adopted and tested a few of them in this dissertation. Future

research is needed on the theoretical and empirical studies of and comparisons between these possible methods.

Values of some important parameters (θ, λ, and γ) in SEED and E-RCEM are arbitrarily selected. The original design space, the number of initial data points and that of new points added in each iteration, are also arbitrarily set. Recommendations are given in this dissertation but more empirical studies are needed, not only to provide better suggestions but also to test the flexibility of the SEED and E-RCEM methods.

The stopping criterion is another research topic. In this dissertation we use the total number of observed points as the stopping criterion. Solid metamodel evaluation approaches are desired to test the accuracy of metamodels, and thus may be used as a stopping criterion. In E-RCEM, it is also possible to stop the integrated design process by testing the existence of "cluster" in the sequential identification of data points; as more data points are identified in E-RCEM and we are approaching the critical region with the actual design solution, new identified points tend to cluster, and this may lead to an effective stopping criterion.

Uncertainty of Design Goals and Constraints

To apply the integrated design process in E-RCEM, one premise is that the design goals and constraints should be clear and fixed (or with small uncertainty). If the design constraints and goals are changed during the metamodeling and design space exploration process, the actual design solution will change and thus the identified data points may not still be in "critical" regions. When great uncertainty of the design goals and constraints

exists, we recommend the SEED method instead of the integrated design process in E-RCEM because the globally accurate metamodels developed from SEED are robust to the changes of design goals and constraints in the process of design space exploration.

When the uncertainty of design goals and constraints is not expected to be great, the E-RCEM method may still be used. In such cases, methods to measure, model, and control this uncertainty are needed. This is future work of this dissertation.

Concave and Discrete Design Spaces

Only examples with continuous and convex design spaces are used in this dissertation. However, in real-world industrial applications, due to the complex design constraints, the feasible design spaces are usually concave and/or discrete. It is an interesting yet difficult research direction to study sequential metamodeling and design space exploration in such cases. Design with concave design spaces studied in (Mistree, et al., 1993b) can be very helpful in this proposed research.

Study and Application of SEED and E-RCEM in Large-Scale Engineering Problems

In this dissertation, the SEED and E-RCEM methods are developed, verified, and illustrated with relatively simple examples. In large-scale real-world applications, the design process can be described as multi-variable, multi-response, and multi-objective. How do SEED and E-RCEM perform in cases with hundred or thousands of design variables, responses, or design objectives? What modifications or improvements should be done to SEED and E-RCEM to ensure an effective and efficient design in such cases?

To apply and improve SEED and E-RCEM in large-scale industrial problems is one possible future direction for research in the vein of studies in this dissertation.

Possible interesting and hot applications of SEED and E-RCEM include biomechanical devices, energy and environment analysis, homeland security cases, and medicine, etc. For example, as discussed in (), in biomedical studies, besides enabling physicians to devise better treatments for individual patients, simulation-based engineering methods could enable medical device manufacturers to predict the performance of their devices in virtual patients prior to deployment in human trials. Current physical and animal testing procedures (now used prior to human trials) have significant limitations in representing variations in human anatomy and physiology. Virtual prototyping of medical devices could be conducted by simulating the deployment of alternate device-designs in a group of virtual patients representing the range of conditions likely to be encountered. SEED and E-RCEM provide great utilities in such applications.

APPENDIX A

SEQUENTIAL EXPLORATORY EXPERIMENTAL DESIGN: CODES AND ORGANIZATION OF PROCESSES

This appendix is intended to supplement the development of the SEED method in Chapter 4. The computer codes written to support the SEED method is presented in Section A.1. The organization of the point-identification process of SEED is illustrated in Sections A.2 and A.3.

A.1 EXPLORATION OF DESIGN SOLUTIONS WITH RCEM

The FORTRAN programs used in SEED in Sections 4.6.2 and 4.6.3 are enclosed in this section. To formulate the covariance matrix we use covmat.f and covdata.params.h; the input and output filenames are specified in covdata.params.h. To adjust entries of the covariance matrix we use altcov.f and altcov.params.h. To calculate the determinant of the covariance matrix we use detcov.f and detcov.params.h.

Covmat.f (Formulation I):

```
************************************************************************
*
      program covmat
*
*  This program invokes calculation of the correlation matrix given
*        information of points and values of theta.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Yao Lin 26 March 2003 / Tim Simpson, 25 Feburary 1998
*
************************************************************************
*
* Input files:
* ------------
*  covdata.params.h - parameter file, specifying numdv, numsamp, fprefix
*  .sam             - x's of sample points
*  .gau.fit         - thetas
*
* Output files:
* -------------
*  .cov             - correlation matrix
*
* Variables:
* ----------
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*   numsamp = number of data samples from which the correlation matrix
*               is calculated
*
* Local Variables:
* ----------------
*   DOUBLE PRECISION
*   ----------------
*   xmat     = numdv x numsamp of sample site locations, scaled [0,1]
*
*   INTEGER
*   -------
```

```
*
**********************************************************************

      integer numdv,numsamp
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix, e.g., in the
C  one-variable problem: numdv=1,numsamp=5,fprefix='step1'
C
      include 'covdata.params.h'

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   dummy2,thetaray(1,numdv),theta(numdv)
      integer i,j,dummy,lenstr
      character*16 ftitle
      character*20 deckfile,fitsfile,outfile


C
C  open necessary .sam, .fit, and .cov files based on 'fprefix' name,
C  e.g., in the one-variable problem:
C         step1.sam, step1.gau.fit, step1.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.sam'
       fitsfile=ftitle(1:lenstr) // '.gau.fit'
       outfile=ftitle(1:lenstr) // '.cov'

       open(21,file=deckfile,status='old')
       open(22,file=fitsfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,fitsfile,outfile
       print *, numdv,numsamp
C
C  initialize xmat and theta arrays
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (xmat(i,j),j=1,numdv)
      close(21)

      print *
      write(6,*) 'Reading in theta parameters...'
      do 20 i=1,1
        read(22,*) dummy,(thetaray(i,j),j=1,numdv),dummy2
        write(6,1000) dummy,(thetaray(i,j),j=1,numdv)
 1000   format(i2,8f9.5)
 20   continue
      close(22)

      do 50 j=1,numdv
          theta(j)=thetaray(1,j)
 50     continue
        write(6,1002) (theta(j),j=1,numdv)
 1002     format(8f9.5)


C
C  call subroutine to calculate the correlation matrix
```

```
C
C  input:  xmat, theta, numsamp, numdv
C
C  output: R - the correlation matrix
C

         call cormat (xmat,cov,numsamp,numdv,theta)

C
C  write predicted values to specified .cov file
C
      do 90 i=1,numsamp
        write(27,79) (cov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 90   continue
      close(27)

      print *
      write(6,*) 'Correlation matrix written to specified .cov file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'covdata.params.h'.
*
*  With this known, the .sam, .gau.fit, and .cov suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Yao Lin, 3/26/2003; Tim Simpson, 2/15/1998
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

*********************************************************************
*
      subroutine cormat (xmat,cov,numsamp,numdv,theta)
*
*
```

```
*  This subroutine calculates the correlation matrix and its inverse
*
*  Original code developed by:
*  Yao Lin 26 March 2003 /
*        Tim Simpson 15 February 1998 / Tony Giunta, 12 May 1997
*
**********************************************************************
*
* Inputs:
* -------
*   DOUBLE PRECISION:
*   -----------------
*   xmat,theta
*
*   INTEGER:
*   --------
*   numdv,numsamp
*
* Outputs:
* --------
*   DOUBLE PRECISION:
*   -----------------
*   cov - the correlation matrix.
*
*
**********************************************************************
C
C  passed variables
C
      integer numdv,numsamp

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   theta(numdv),R
C
C  local variables
C
      integer i,j
C
C  calculate terms in the correlation matrix
C
      do 300 i = 1,numsamp
        do 305 j = i,numsamp
          if( i .eq. j ) then
            cov(i,j) = 1.0d0
          else
C
C  call subroutine to compute spatial correlation function for xmat
C
C  input:  xmat, theta, numdv, numsamp, i, j
C
C  output: R
C
            call scfxmat(R,xmat,theta,numdv,numsamp,i,j)
            cov(i,j) = R
            cov(j,i) = cov(i,j)
          endif
 305    continue
 300  continue
      end

C**********************************************************************
C
      subroutine scfxmat(R,xmat,theta,numdv,numsamp,i,j)
C
```

```
C     Origin: Tim Simpson       Date:  February 11, 1998
C     Modified: Yao Lin         Date:  March 26, 2003
C
C     subroutine to compute spatial correlation function (scf) for
C     correlation matrix; NOT to compute scf for r_xhat.
C
C  Output:
C  -------
C    R = value of correlation function between two sample points,
C          given theta
C
C  Input:
C  ------
C    xmat = matrix of sample points
C    theta = array of theta values
C    i,j = i_th and j_th elements of correlation matrix for which
C           correlation function is being computed
C
C  All variables except R are unchanged upon exiting
C
C*********************************************************************
C
C  passed variables
C
      integer i,j,numdv,numsamp
      double precision R,xmat(numsamp,numdv),theta(numdv)
C
C  local variables
C
      double precision sum,thetadist,dist
      integer k

      sum=0.0d0
      do 120 k = 1,numdv
          dist = ABS(xmat(i,k)-xmat(j,k))
          sum = sum + theta(k)*((dist)**2)
  120      continue
         R = exp( -1.0d0*sum )

         return
         end
```

Covdata.params.h (Formulation I):

```
C*********************************************************
C                                                        *
C  Parameter input file for 'covmat'                  *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for calculating the covariance
C          matrix and its determinant
C

      parameter (numdv=1,numsamp=11,fprefix='suit3valid')

C
C  numdv = # design variables
C  numsamp = # samples in data set
C
```

```
C  fprefix = prefix of titles of files to opened/used
C
C*********************************************************
```

Covmat.f (Formulation II):

```
************************************************************************
*
      program covmat
*
*  This program invokes calculation of the correlation matrix given
*        information of points and values of theta.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Yao Lin 26 March 2003 / Tim Simpson, 25 Feburary 1998
*
************************************************************************
*
* Input files:
* ------------
*  covdata.params.h - parameter file, specifying numdv, numsamp, fprefix
*  .sam             - x's of sample points
*  .gau.fit         - thetas
*
* Output files:
* -------------
*  .cov             - correlation matrix
*
* Variables:
* ----------
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*   numsamp = number of data samples from which the correlation matrix
*               is calculated
*   errmax = maximum predicted prediction error
*   lambda = safety coefficient
*
* Local Variables:
* ----------------
*   DOUBLE PRECISION
*   ----------------
*   xmat     = numdv x numsamp of sample site locations, scaled [0,1]
*
*   INTEGER
*   -------
*
************************************************************************

      integer numdv,numsamp,numold
      double precision lambda,errmax
      character*16 fprefix,fprefixe
C
C  include parameter settings for numdv,numsamp,fprefix, e.g., in the
C  one-variable problem: numdv=1,numsamp=5,fprefix='step1'
C
      include 'covdata.params.h'

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   dummy2,thetaray(1,numdv),theta(numdv),errpred(numsamp)
```

```
      integer i,j,dummy,lenstr
      character*16 ftitle
      character*20 deckfile,fitsfile,outfile,errpredfile


C
C  open necessary .sam, .fit, and .cov files based on 'fprefix' name,
C  e.g., in the one-variable problem:
C         step1.sam, step1.gau.fit, step1.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.sam'
       fitsfile=ftitle(1:lenstr) // '.gau.fit'
       outfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefixe,lenstr)
       ftitle=fprefixe
       errpredfile=ftitle(1:lenstr) // '.out'

       open(21,file=deckfile,status='old')
       open(22,file=fitsfile,status='old')
       open(23,file=errpredfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,fitsfile,outfile
       print *, numdv,numsamp
C
C  initialize xmat and theta arrays
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (xmat(i,j),j=1,numdv)
      close(21)

      print *
      write(6,*) 'Reading in theta parameters...'
      do 20 i=1,1
        read(22,*) dummy,(thetaray(i,j),j=1,numdv),dummy2
        write(6,1000) dummy,(thetaray(i,j),j=1,numdv)
 1000   format(i2,8f9.5)
 20   continue
      close(22)

      print *
      write(6,*) 'Reading in and calculating errpred...'
      do 30 i=1,numsamp
         if (i.le.numold) then
            errpred(i)=0.0
         else
            read(23,*) errpred(i)
         endif
         if (abs(errpred(i)).gt.(errmax)) then
            errpred(i)=errmax
         endif
 30   continue
      close(23)

      print *
      do 50 j=1,numdv
          theta(j)=thetaray(1,j)
```

```
 50     continue
        write(6,*) 'theta'
        write(6,1002) (theta(j),j=1,numdv)
 1002     format(8f9.5)


C
C  call subroutine to calculate the correlation matrix
C
C  input:  xmat, theta, numsamp, numdv
C
C  output: R - the correlation matrix
C

         call cormat (xmat,cov,numsamp,numdv,theta,
     &         errpred,errmax,lambda)

C
C  write predicted values to specified .cov file
C
      do 90 i=1,numsamp
        write(27,79) (cov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 90   continue
      close(27)

      print *
      write(6,*) 'Correlation matrix written to specified .cov file'

      stop
      end

**********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'covdata.params.h'.
*
*  With this known, the .sam, .gau.fit, and .cov suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Yao Lin, 3/26/2003; Tim Simpson, 2/15/1998
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
```

```
      end if
      return
      end

*****************************************************************************
*
      subroutine cormat (xmat,cov,numsamp,numdv,theta,
     &           errpred,errmax,lambda)
*
*
*  This subroutine calculates the alternated correlation matrix (by
*       changing values of theta between any two points,
*       and the inverse of the alternated correlation matrix
*
*  Original code developed by:
*  Yao Lin 26 March 2003 /
*        Tim Simpson 15 February 1998 / Tony Giunta, 12 May 1997
*
*****************************************************************************
*
* Inputs:
* -------
*   DOUBLE PRECISION:
*   -----------------
*   xmat,theta,errpred
*
*   INTEGER:
*   --------
*   numdv,numsamp
*
* Outputs:
* --------
*   DOUBLE PRECISION:
*   -----------------
*   cov - the correlation matrix.
*
*
*****************************************************************************
C
C  passed variables
C
      integer numdv,numsamp

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   theta(numdv),R,errpred(numsamp),errmax,lambda
C
C  local variables
C
      integer i,j
C
C  calculate terms in the correlation matrix
C
      do 300 i = 1,numsamp
        do 305 j = i,numsamp
          if( i .eq. j ) then
            cov(i,j) = 1.0d0
          else
C
C  call subroutine to compute spatial correlation function for xmat
C
C  input:  xmat, theta, numdv, numsamp, i, j
C
C  output: R
C
```

```
            call scfxmat(R,xmat,theta,numdv,numsamp,i,j,
     &                   errpred,errmax,lambda)
            cov(i,j) = R
            cov(j,i) = cov(i,j)
          endif
 305    continue
 300  continue
      end

C*********************************************************************
C
      subroutine scfxmat(R,xmat,theta,numdv,numsamp,i,j,
     &                   errpred,errmax,lambda)
C
C     Origin: Tim Simpson       Date:  February 11, 1998
C     Modified: Yao Lin         Date:  March 26, 2003
C
C     subroutine to compute spatial correlation function (scf) for
C     correlation matrix; NOT to compute scf for r_xhat.
C
C  Output:
C  -------
C    R = value of correlation function between two sample points,
C          given theta
C
C  Input:
C  ------
C    xmat = matrix of sample points
C    theta = array of theta values
C    i,j = i_th and j_th elements of correlation matrix for which
C           correlation function is being computed
C    errpred = predicted prediction error at points
C    errmax = maximum predicted prediction error
C    lambda = safety coefficient
C
C  All variables except R are unchanged upon exiting
C
C*********************************************************************
C
C  passed variables
C
      integer i,j,numdv,numsamp
      double precision R,xmat(numsamp,numdv),theta(numdv),
     &       errpred(numsamp),errmax,lambda
C
C  local variables
C
      double precision sum,thetadist,dist,alttheta(numdv)
      integer k

      sum=0.0d0
      do 120 k = 1,numdv
        dist = ABS(xmat(i,k)-xmat(j,k))
        alttheta(k)=theta(k)*(1+lambda*abs(errpred(i))
     &      /errmax)*(1+lambda*abs(errpred(j))/errmax)
          sum = sum + alttheta(k)*((dist)**2)
  120      continue
      R = exp( -1.0d0*sum )

      write(6,1003) (alttheta(1))
 1003 format(8f9.5)
         return
         end
```

Covdata.params.h (Formulation II):

```
C*********************************************************
C                                                        *
C  Parameter input file for 'covmat'                  *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for calculating the covariance
C          matrix and its determinant
C

      parameter (numdv=1,numsamp=4,numold=3,
     &           fprefix='suit1newp',
     &           fprefixe='errpred1_1.gau',
     &           errmax=1.23,lambda=2)

C
C  numdv = # design variables
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C*********************************************************
```

Suit3valid.sam:

```
       0
       0.167
       0.5
       0.75
       1
       0.122
       0.235
       0.333
       0.667
       0.833
       0.0472747809891277
```

Suit3valid.gau.fit:

```
       1      63.78181      -16.55119
```

Altcov.f (Formulation I in Section 4.6.2):

```
************************************************************************
*
      program altcov
*
*  This program calculates the alternated correlation matrix, given the
```

```
*       initial correlation matrix and predicted prediction errors at
*       possible new data points.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
*
************************************************************************
*
* Input files:
* ------------
*  altcov.params.h - parameter file, specifying numdv, numsamp,
*                       errmax, lambda, fprefix, fprefix2, fprefixnew
*  fprefix.cov     - initial correlation matrix
*  fprefix2.out    - predicted prediction errors at possible new data points
*
* Output files:
* -------------
*  fprefixnew.cov  - alternated correlation matrix
*
* Variables:
* ----------
*  inicov     = the initial correlation matrix
*  newcov     = the alternated correlation matrix
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*  numsamp = number of data samples from which the correlation matrix
*               is calculated
*
* Local Variables:
* ----------------
*  DOUBLE PRECISION
*  ----------------
*  errpred = the predicted prediction errors associated with each data
*            and possible new data points
*
************************************************************************

      integer numsamp
      double precision lambda,errmax
      character*16 fprefix,fprefix2,fprefixnew
C
C  include parameter settings for numdv,numsamp,fprefix,fprefix2,fprefixnew,
C  errmax, lambda, e.g., in the one-variable problem, for the first step:
C  numdv=1,numsamp=8,fprefix='step1newp',fprefix2='errpred1',
C  fprefixnew='step1altnewp',errmax=0.50,lambda=2.0
C
      include 'altcov.params.h'

      double precision inicov(numsamp,numsamp),newcov(numsamp,numsamp),
     &       errpred(numsamp)
      integer i,j,lenstr
      character*16 ftitle
      character*20 deckfile,deckfile2,outfile

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.cov'
```

```
       call getlen(fprefix2,lenstr)
       ftitle=fprefix2
       deckfile2=ftitle(1:lenstr) // '.out'

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.cov'

       open(21,file=deckfile,status='old')
       open(23,file=deckfile2,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,deckfile2,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (inicov(i,j),j=1,numsamp)
      close(21)

C
C  initialize errpred
C
      print *
      write(6,*) 'Reading in and calculating errpred...'
      do 20 i=1,numsamp
         if (i.le.numold) then
            errpred(i)=0.0
         else
            read(23,*) errpred(i)
         endif
         if (abs(errpred(i)).gt.(errmax)) then
            errpred(i)=errmax
         endif
 20   continue
      close(23)

C
C  calculate the alternated correlation matrix
C
      do 30 i=1,numsamp
         do 40 j=i,numsamp
         if (i.eq.j) then
            newcov(i,j)=1.0
         elseif (((i.gt.numold).AND.(j.le.numold)).OR.
     &             ((i.le.numold).AND.(j.gt.numold))) then
         newcov(i,j)=inicov(i,j)*(1-abs(errpred(i)/lambda/errmax))
     &   *(1-abs(errpred(j)/errmax/lambda))
         newcov(j,i)=newcov(i,j)
         else
         newcov(i,j)=inicov(i,j)
         newcov(j,i)=newcov(i,j)
         endif
 40   continue
 30   continue

C
C  write alternated correlation matrix into specified .cov file
C
```

```
      do 50 i=1,numsamp
        write(27,79) (newcov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 50   continue
      close(27)

      print *
      write(6,*) 'Alternated correlation matrix written to .cov file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end
```

Altcov.params.h:

```
C*********************************************************
C                                                        *
C  Parameter input file for 'altcov'                     *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for dace modeling software
C
```

```
      parameter ( numdv=1,numsamp=11,numold=10,
     &          fprefix='suit3valid',fprefix2='errpred3_1.gau',
     &          fprefixnew='suit3altvalid',errmax=1.5,
     &          lambda=2.0 )

C
C  numdv = # design variables
C  numsamp = # samples in data set
C  numold = # old data points in the data set
C
C  fprefix = prefix of titles of file that stores the initial
C            correlation matrix for both old and possible new
C            data points
C
C  fprefix2 = prefix of titles of file that stores the
C             predicted prediction errors at possible new
C             data points
C
C  fprefixnew = prefix of titles of file that stores the
C               alternated correlation matrix for both old and
C               possible new data points, with prediction errors
C               at these points considered
C
C  errmax = maximum value of the absolute predicted prediction error
C
C  lambda = coefficient used to gauge the adjustment to initial
C           correlation matrix
C*********************************************************************
```

Detcov.f:

```
C*********************************************************************
C
      program detcov
C
C  This program calculates the determinant given a matrix.  Particularly,
C       in SEED, it is used to calculate the determinant of the
C       correlation matrix.
C
C  Updated by: Yao Lin, March 26, 2003
C
C  Original code developed by:
C  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
C
C*********************************************************************
C
C Input files:
C ------------
C  detcov.params.h - parameter file, specifying numdv, numsamp,
C                        coedet, fprefix
C  .cov             - correlation matrix
C
C Output files:
C -------------
C  .det             - determinant of the correlation matrix
C
C Variables:
C ----------
C   cov     = the input correlation matrix for which we calculate
C             determinant
C
C Parameter Variables (to be specified by user in dace.params.h):
```

```
C -------------------------------------------------
C   numsamp = number of data samples from which the correlation matrix
C               is calculated
C
C Local Variables:
C ----------------
C   DOUBLE PRECISION
C   ----------------
C   work     = vector of length 'numsamp' used as temporary storage
C   invmat   = inverse of the correlation matrix (numsamp x numsamp)
C
C   INTEGER
C   -------
C   ipvt    = vector of length 'numsamp' of pivot locations
C
C**********************************************************************

      integer numsamp
      double precision coedet
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix
C
      include 'detcov.params.h'

C**********************************************************************
C
C  include LINPACK routines used to find determinant of correlation matrix
C
C**********************************************************************

C      include 'dgefa.f'
C      include 'dgedi.f'

C**********************************************************************

      double precision cov(numsamp,numsamp),work(numsamp),
     &       dummy2,detR,det(2),rcond,z(numsamp)
      integer i,j,ipvt(numsamp),dummy,lenstr,info
      character*16 ftitle
      character*20 deckfile,outfile
      err=0.0000
C
C  open necessary .cov and .det files based on 'fprefix' name,
C  e.g., step1.cov, step1.det
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.cov'
       outfile=ftitle(1:lenstr) // '.det'

       open(21,file=deckfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,outfile
       print *, numsamp
C
C  initialize cov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
```

```
 10     read (21,*) (cov(i,j),j=1,numsamp)
      close(21)

C
C     Start to calculate the determinant of the correlation matrix;
C        initialization.
C
      do 307 i=1,numsamp
        work(i)=0.0d0
        ipvt(i)=0
 307  continue

C
C     If there is any error in the calculation in DGEFA (singular matrix),
C        this program will set the determinant to 0.
C
      call dgeco(cov,numsamp,numsamp,ipvt,rcond,z)
      if( rcond .eq. 0 ) then
          write(27,78) err
 78   format(10(f13.5,1x))
          close(27)
          go to 1000
      endif
C
C In DGEDI, last flag is: 1 (inverse only), 10 (Det only), 11 (both)
C
      call dgedi(cov, numsamp, numsamp, ipvt, det, work, 10)
      detR=det(1)*10.0d0**det(2)
      detR=coedet*detR

C
C  write predicted values to specified .det file
C
      write(27,79) detR
 79   format(10(f13.5,1x))
      close(27)

      print *
      write(6,*) detR
1000  write(6,*) 'Coefficient*Determinant written to .det file'

      stop
      end

**********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      character*1 blank
```

```
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

      subroutine dgeco(a,lda,n,ipvt,rcond,z)
      integer lda,n,ipvt(1)
      double precision a(lda,1),z(1)
      double precision rcond
c
c     dgeco factors a double precision matrix by gaussian elimination
c     and estimates the condition of the matrix.
c
c     if  rcond  is not needed, dgefa is slightly faster.
c     to solve  a*x = b , follow dgeco by dgesl.
c     to compute  inverse(a)*c , follow dgeco by dgesl.
c     to compute  determinant(a) , follow dgeco by dgedi.
c     to compute  inverse(a) , follow dgeco by dgedi.
c
c     on entry
c
c        a       double precision(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        rcond   double precision
c                an estimate of the reciprocal condition of  a .
c                for the system  a*x = b , relative perturbations
c                in  a  and  b  of size  epsilon  may cause
c                relative perturbations in  x  of size  epsilon/rcond .
c                if  rcond  is so small that the logical expression
c                           1.0 + rcond .eq. 1.0
c                is true, then  a  may be singular to working
c                precision.  in particular,  rcond  is zero  if
c                exact singularity is detected or the estimate
c                underflows.
```

```
c
c        z       double precision(n)
c                a work vector whose contents are usually unimportant.
c                if  a  is close to a singular matrix, then  z  is
c                an approximate null vector in the sense that
c                norm(a*z) = rcond*norm(a)*norm(z) .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     linpack dgefa
c     blas daxpy,ddot,dscal,dasum
c     fortran dabs,dmax1,dsign
c
c     internal variables
c
      double precision ddot,ek,t,wk,wkm
      double precision anorm,s,dasum,sm,ynorm
      integer info,j,k,kb,kp1,l
c
c
c     compute 1-norm of a
c
      anorm = 0.0d0
      do 10 j = 1, n
         anorm = dmax1(anorm,dasum(n,a(1,j),1))
   10 continue
c
c     factor
c
      call dgefa(a,lda,n,ipvt,info)
c
c     rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
c     estimate = norm(z)/norm(y) where  a*z = y  and  trans(a)*y = e .
c     trans(a)  is the transpose of a .  the components of  e  are
c     chosen to cause maximum local growth in the elements of w  where
c     trans(u)*w = e .  the vectors are frequently rescaled to avoid
c     overflow.
c
c     solve trans(u)*w = e
c
      ek = 1.0d0
      do 20 j = 1, n
         z(j) = 0.0d0
   20 continue
      do 100 k = 1, n
         if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
         if (dabs(ek-z(k)) .le. dabs(a(k,k))) go to 30
            s = dabs(a(k,k))/dabs(ek-z(k))
            call dscal(n,s,z,1)
            ek = s*ek
   30    continue
         wk = ek - z(k)
         wkm = -ek - z(k)
         s = dabs(wk)
         sm = dabs(wkm)
         if (a(k,k) .eq. 0.0d0) go to 40
            wk = wk/a(k,k)
            wkm = wkm/a(k,k)
         go to 50
   40    continue
            wk = 1.0d0
```

```
            wkm = 1.0d0
   50       continue
            kp1 = k + 1
            if (kp1 .gt. n) go to 90
               do 60 j = kp1, n
                  sm = sm + dabs(z(j)+wkm*a(k,j))
                  z(j) = z(j) + wk*a(k,j)
                  s = s + dabs(z(j))
   60          continue
               if (s .ge. sm) go to 80
                  t = wkm - wk
                  wk = wkm
                  do 70 j = kp1, n
                     z(j) = z(j) + t*a(k,j)
   70             continue
   80          continue
   90       continue
            z(k) = wk
  100    continue
         s = 1.0d0/dasum(n,z,1)
         call dscal(n,s,z,1)
c
c        solve trans(l)*y = w
c
         do 120 kb = 1, n
            k = n + 1 - kb
            if (k .lt. n) z(k) = z(k) + ddot(n-k,a(k+1,k),1,z(k+1),1)
            if (dabs(z(k)) .le. 1.0d0) go to 110
               s = 1.0d0/dabs(z(k))
               call dscal(n,s,z,1)
  110       continue
            l = ipvt(k)
            t = z(l)
            z(l) = z(k)
            z(k) = t
  120    continue
         s = 1.0d0/dasum(n,z,1)
         call dscal(n,s,z,1)
c
         ynorm = 1.0d0
c
c        solve l*v = y
c
         do 140 k = 1, n
            l = ipvt(k)
            t = z(l)
            z(l) = z(k)
            z(k) = t
            if (k .lt. n) call daxpy(n-k,t,a(k+1,k),1,z(k+1),1)
            if (dabs(z(k)) .le. 1.0d0) go to 130
               s = 1.0d0/dabs(z(k))
               call dscal(n,s,z,1)
               ynorm = s*ynorm
  130       continue
  140    continue
         s = 1.0d0/dasum(n,z,1)
         call dscal(n,s,z,1)
         ynorm = s*ynorm
c
c        solve  u*z = v
c
         do 160 kb = 1, n
            k = n + 1 - kb
            if (dabs(z(k)) .le. dabs(a(k,k))) go to 150
```

```
            s = dabs(a(k,k))/dabs(z(k))
            call dscal(n,s,z,1)
            ynorm = s*ynorm
  150    continue
         if (a(k,k) .ne. 0.0d0) z(k) = z(k)/a(k,k)
         if (a(k,k) .eq. 0.0d0) z(k) = 1.0d0
         t = -z(k)
         call daxpy(k-1,t,a(1,k),1,z(1),1)
  160 continue
c     make znorm = 1.0
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
      ynorm = s*ynorm
c
      if (anorm .ne. 0.0d0) rcond = ynorm/anorm
      if (anorm .eq. 0.0d0) rcond = 0.0d0
      return
      end


      subroutine dgedi(a,lda,n,ipvt,det,work,job)
      integer lda,n,ipvt(1),job
      double precision a(lda,1),det(2),work(1)
C
C     dgedi computes the determinant and inverse of a matrix
C     using the factors computed by dgeco or dgefa.
C
C     on entry
C
C        a       double precision(lda, n)
C                the output from dgeco or dgefa.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
C
C        ipvt    integer(n)
C                the pivot vector from dgeco or dgefa.
C
C        work    double precision(n)
C                work vector.  contents destroyed.
C
C        job     integer
C                = 11   both determinant and inverse.
C                = 01   inverse only.
C                = 10   determinant only.
C
C     on return
C
C        a       inverse of original matrix if requested.
C                otherwise unchanged.
C
C        det     double precision(2)
C                determinant of original matrix if requested.
C                otherwise not referenced.
C                determinant = det(1) * 10.0**det(2)
C                with  1.0 .le. dabs(det(1)) .lt. 10.0
C                or  det(1) .eq. 0.0 .
C
C     error condition
C
C        a division by zero will occur if the input factor contains
```

```
C        a zero on the diagonal and the inverse is requested.
C        it will not occur if the subroutines are called correctly
C        and if dgeco has set rcond .gt. 0.0 or dgefa has set
C        info .eq. 0 .
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,dswap
C     fortran dabs,mod
C
C     internal variables
C
      double precision t
      double precision ten
      integer i,j,k,kb,kp1,l,nm1
C
C
C     compute determinant
C
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0d0
         det(2) = 0.0d0
         ten = 10.0d0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
C           ...exit
            if (det(1) .eq. 0.0d0) go to 60
   10       if (dabs(det(1)) .ge. 1.0d0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0d0
            go to 10
   20       continue
   30       if (dabs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0d0
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
C
C     compute inverse(u)
C
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0d0/a(k,k)
            t = -a(k,k)
            call dscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0d0
               call daxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
C
C        form inverse(u)*inverse(l)
C
```

```
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0d0
  110       continue
            do 120 j = kp1, n
               t = work(j)
               call daxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call dswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end

      subroutine daxpy(n,da,dx,incx,dy,incy)
C
C     constant times a vector plus a vector.
C     uses unrolled loops for increments equal to one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C        code for unequal increments or equal increments
C          not equal to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
C
C        code for both increments equal to 1
C
C
C        clean-up loop
C
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
```

```
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end

      subroutine  dscal(n,da,dx,incx)
C
C     scales a vector by a constant.
C     uses unrolled loops for increment equal to one.
C     jack dongarra, linpack, 3/11/78.
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision da,dx(*)
      integer i,incx,m,mp1,n,nincx
C
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dx(i) = da*dx(i)
   10 continue
      return
C
C        code for increment equal to 1
C
C
C        clean-up loop
C
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end

      subroutine  dswap (n,dx,incx,dy,incy)
C
C     interchanges two vectors.
C     uses unrolled loops for increments equal one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C       code for unequal increments or equal increments not equal
```

```
C         to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dx(ix)
        dx(ix) = dy(iy)
        dy(iy) = dtemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
C
C       code for both increments equal to 1
C
C
C       clean-up loop
C
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
        dtemp = dx(i + 1)
        dx(i + 1) = dy(i + 1)
        dy(i + 1) = dtemp
        dtemp = dx(i + 2)
        dx(i + 2) = dy(i + 2)
        dy(i + 2) = dtemp
   50 continue
      return
      end


      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)
C
C     dgefa factors a double precision matrix by gaussian elimination.
C
C     dgefa is usually called by dgeco, but it can be called
C     directly with a saving in time if  rcond  is not needed.
C     (time for dgeco) = (1 + 9/n)*(time for dgefa) .
C
C     on entry
C
C        a       double precision(lda, n)
C                the matrix to be factored.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
```

```
C
C     on return
C
C        a       an upper triangular matrix and the multipliers
C                which were used to obtain it.
C                the factorization can be written  a = l*u  where
C                l  is a product of permutation and unit lower
C                triangular matrices and  u  is upper triangular.
C
C        ipvt    integer(n)
C                an integer vector of pivot indices.
C
C        info    integer
C                = 0  normal value.
C                = k  if  u(k,k) .eq. 0.0 .  this is not an error
C                     condition for this subroutine, but it does
C                     indicate that dgesl or dgedi will divide by zero
C                     if called.  use  rcond  in dgeco for a reliable
C                     indication of singularity.
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,idamax
C
C     internal variables
C
      double precision t
      integer idamax,j,k,kp1,l,nm1
C
C
C     gaussian elimination with partial pivoting
C
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
C
C        find l = pivot index
C
         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
C
C        zero pivot implies this column already triangularized
C
         if (a(l,k) .eq. 0.0d0) go to 40
C
C           interchange if necessary
C
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
C
C           compute multipliers
C
            t = -1.0d0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
C
C           row elimination with column indexing
```

```
C
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d0) info = n
      return
      end

      integer function idamax(n,dx,incx)
C
C     finds the index of element having max. absolute value.
C     jack dongarra, linpack, 3/11/78.
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dmax
      integer i,incx,ix,n
C
      idamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      ix = 1
      dmax = dabs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(dabs(dx(ix)).le.dmax) go to 5
         idamax = i
         dmax = dabs(dx(ix))
    5    ix = ix + incx
   10 continue
      return
C
C        code for increment equal to 1
C
   20 dmax = dabs(dx(1))
      do 30 i = 2,n
         if(dabs(dx(i)).le.dmax) go to 30
         idamax = i
         dmax = dabs(dx(i))
   30 continue
      return
      end

      double precision function dasum(n,dx,incx)
c
c     takes the sum of the absolute values.
c     jack dongarra, linpack, 3/11/78.
```

```
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dtemp
      integer i,incx,m,mp1,n,nincx
c
      dasum = 0.0d0
      dtemp = 0.0d0
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dtemp = dtemp + dabs(dx(i))
   10 continue
      dasum = dtemp
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,6)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dabs(dx(i))
   30 continue
      if( n .lt. 6 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,6
        dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
     &  + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
   50 continue
   60 dasum = dtemp
      return
      end

      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
```

```
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     & dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
```

Detcov.params.h:

```
C************************************************************
C                                                           *
C  Parameter input file for 'detcov'                        *
C    Author: Yao Lin                                        *
C      Date: 3/26/2003                                      *
C                                                           *
C************************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=1,numsamp=11,fprefix='suit3altvalid',
     &          coedet=1e4 )
C
C  numdv = # design variables
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C  coedet = when the value of determinant is very small,
C          this coefficient is used to magnify the value.
C************************************************************
```

A.2 IMPLEMENTATION OF SEED (FORMULATION I) IN ISIGHT IN SECTION 4.6.2

Figures presented in this section illustrate how the SEED method (with Formulation I) is implemented in iSIGHT. The organization of tasks in Iteration I – Step 3 is shown in Figure A.1. The organization of tasks in Iteration I – Step 7 is shown in Figure A.2.

In Iteration I – Step 3, since the covariance matrix is not adjusted, there are only two simulation codes used in iSIGHT, Covmat and Detcov. In Iteration I – Step 7, with information from metamodels of prediction errors, we use four simulation codes in iSIGHT, i.e., Covmat, Errpred, Altcov, and Detcov. Covmat is used to formulate the covariance matrix, Errpred are metamodels to predict prediction errors, Altcov is used to adjust entries of the covariance matrix, and Detcov is used to calculate the determinant.

The parameter and input/output files for the component Covmat in iSIGHT are:

- Input files: Inputfilename1.sam (containing $n_s + n_{new}$ data points)
  Inputfilename1.gau.fit
- Output file: Outputfilename1.cov
- Parameter file: Covmat.params.h

The parameter and input/output files for the component Errpred in iSIGHT are:

- Input file: Inputfilename2.npt (containing $n_s + n_{new}$ data points)
- Output file: Outputfilename2.gau.out
- Parameter files: Dace.params.h
  Inputfilename2.dek
  Inputfilename2.gau.fit

The parameter and input/output files for the component Altcov in iSIGHT are:

- Input files: Outputfilename1.cov
  Outputfilename2.gau.out
- Output file: AltOutputfilename1.cov
- Parameter file: Altcov.params.h

The parameter and input/output files for the component Detcov in iSIGHT are:

- Input file: AltOutputfilename1.cov
- Output file: AltOutputfilename1.det
- Parameter file: Detcov.params.h

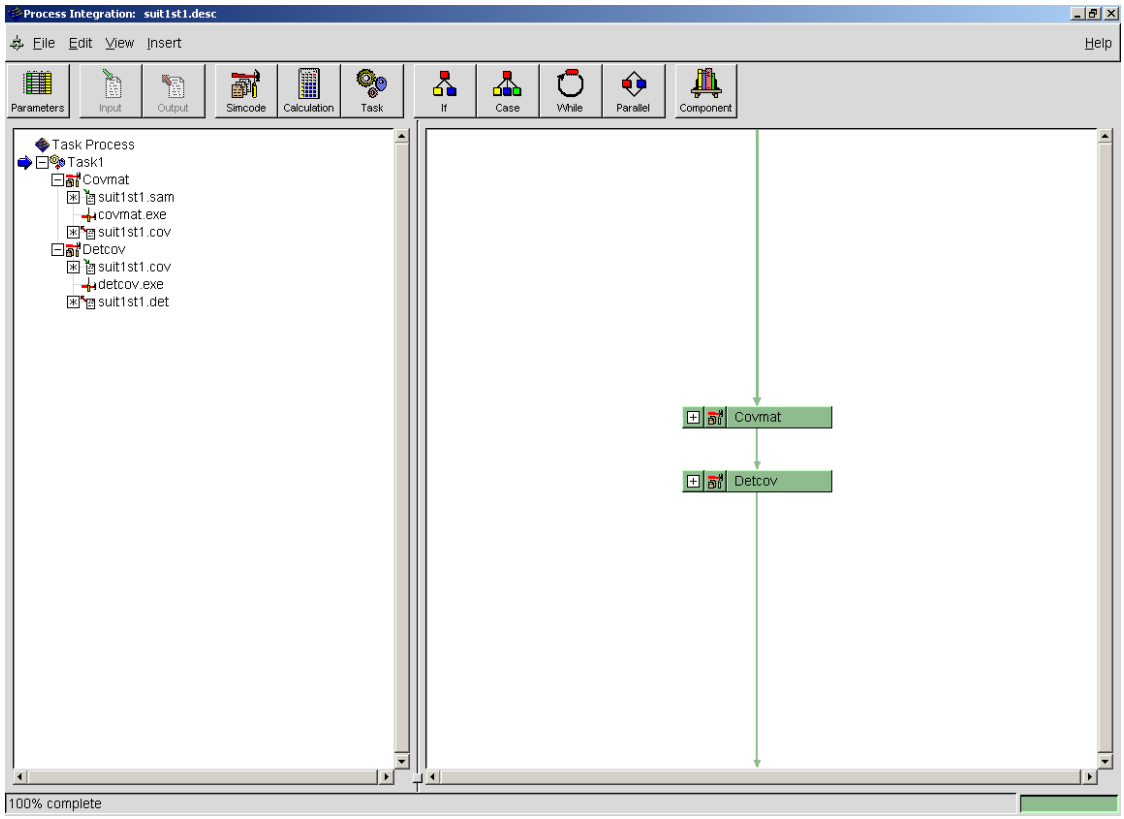


Figure A.1 Implementation of SEED in iSIGHT – Iteration I, Step 3

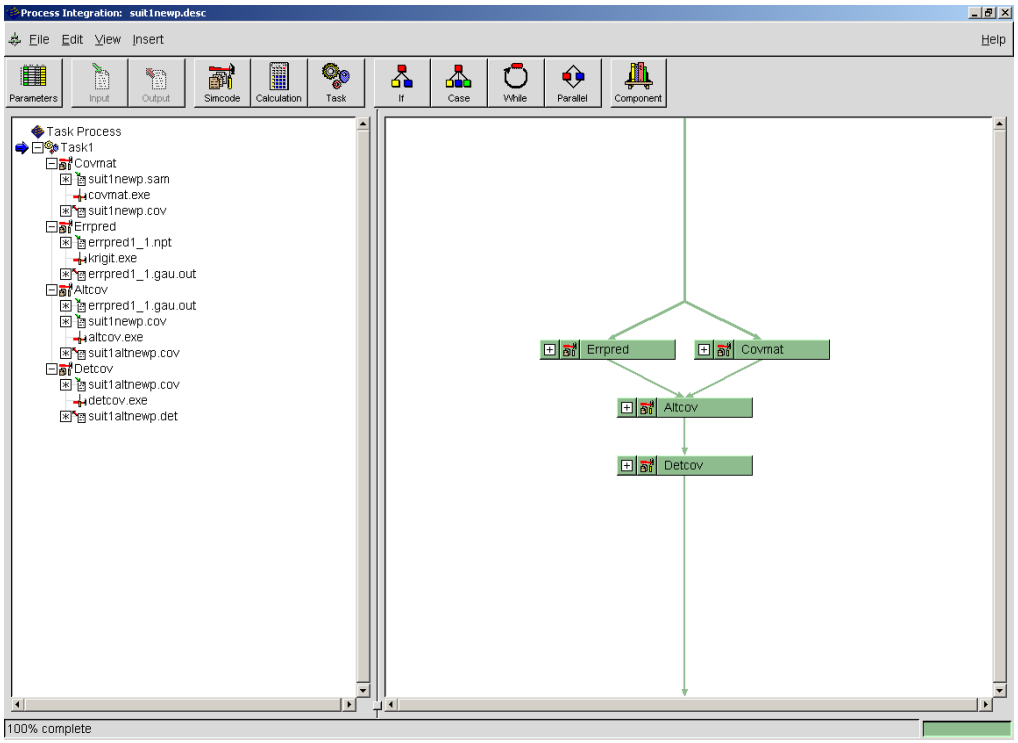


Figure A.2 Implementation of SEED (Formulation I) in iSIGHT – Iteration I, Step 7

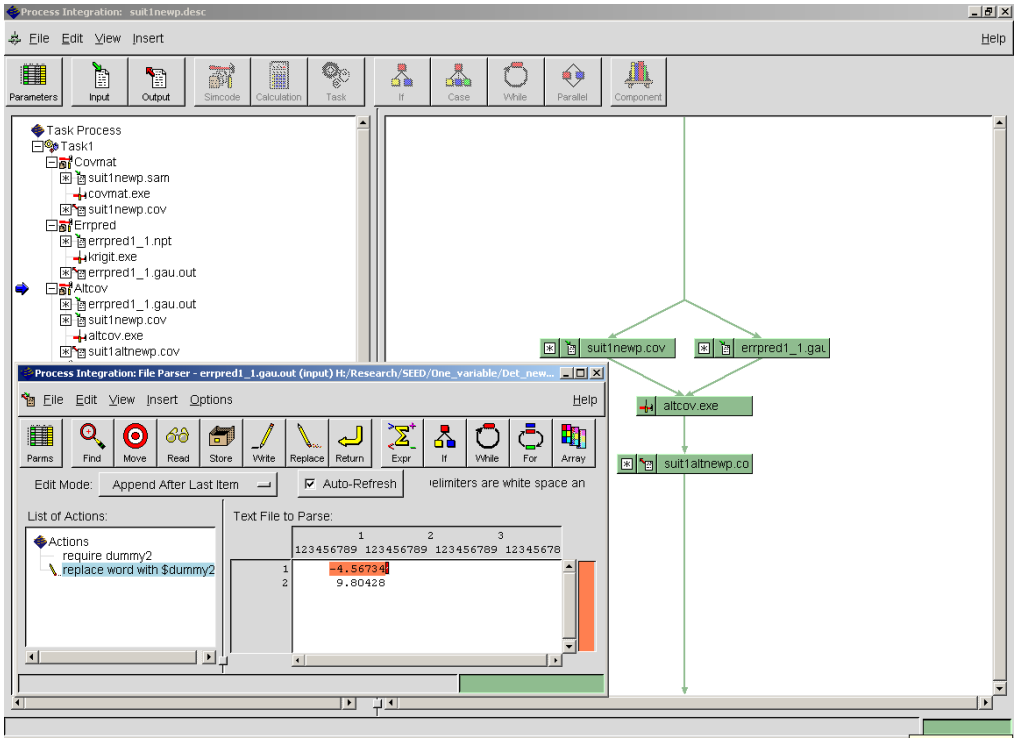


Figure A.3 File Parsing in iSIGHT (Formulation I) – Iteration I, Step 7

A.3 IMPLEMENTATION OF SEED (FORMULATION II) IN ISIGHT IN SECTION 4.6.3

Figures presented in this section illustrate how the SEED method (with Formulation II) is implemented in iSIGHT. The organization of tasks in Iteration I – Step 7 is shown in Figure A.4. In Iteration I – Step 7, with information from metamodels of prediction errors, we use three simulation codes in iSIGHT, i.e., Covmat, Errpred, and Detcov. Covmat is used to formulate the adjusted covariance matrix, Errpred are metamodels to predict prediction errors, and Detcov is used to calculate the determinant.

The parameter and input/output files for the component Errpred in iSIGHT are:

- Input file: Inputfilename2.npt (containing $n_s + n_{new}$ data points)
- Output file: Outputfilename2.gau.out
- Parameter files: Dace.params.h
  Inputfilename2.dek
  Inputfilename2.gau.fit

The parameter and input/output files for the component Covmat in iSIGHT are:

- Input files: Inputfilename1.sam (containing $n_s + n_{new}$ data points)
  Inputfilename2.gau.out
- Output file: Outputfilename1.cov
- Parameter file: Covmat.params.h
  Inputfilename1.gau.fit

The parameter and input/output files for the component Detcov in iSIGHT are:

- Input file: Outputfilename1.cov
- Output file: Outputfilename1.det
- Parameter file: Detcov.params.h

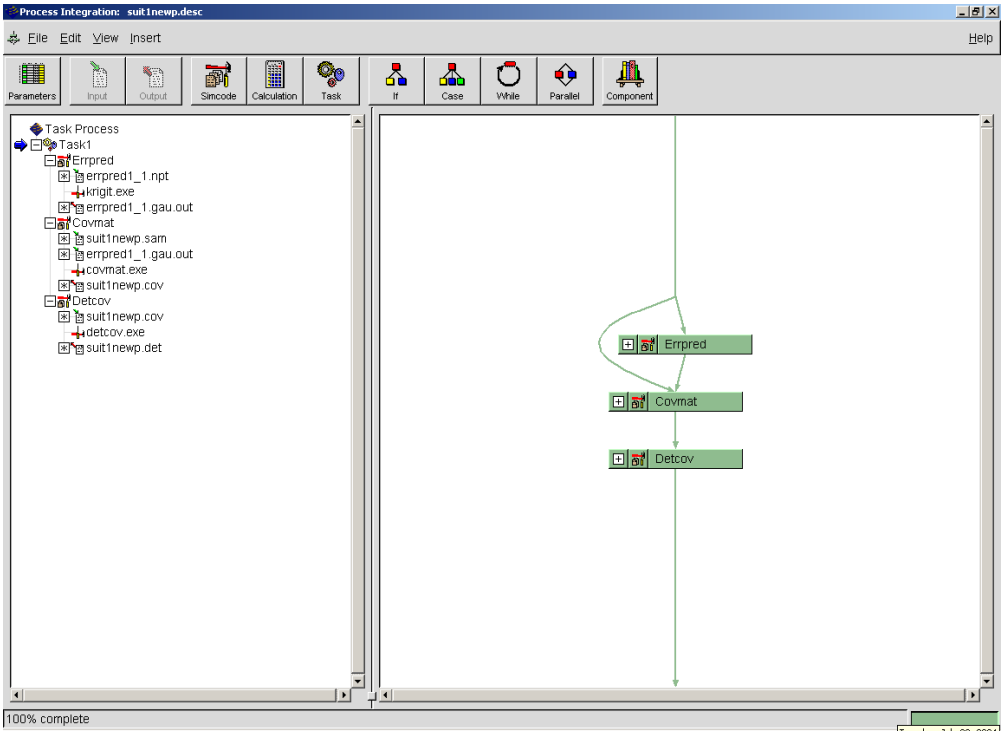


Figure A.4 Implementation of SEED (Formulation II) in iSIGHT – Iteration I, Step 7

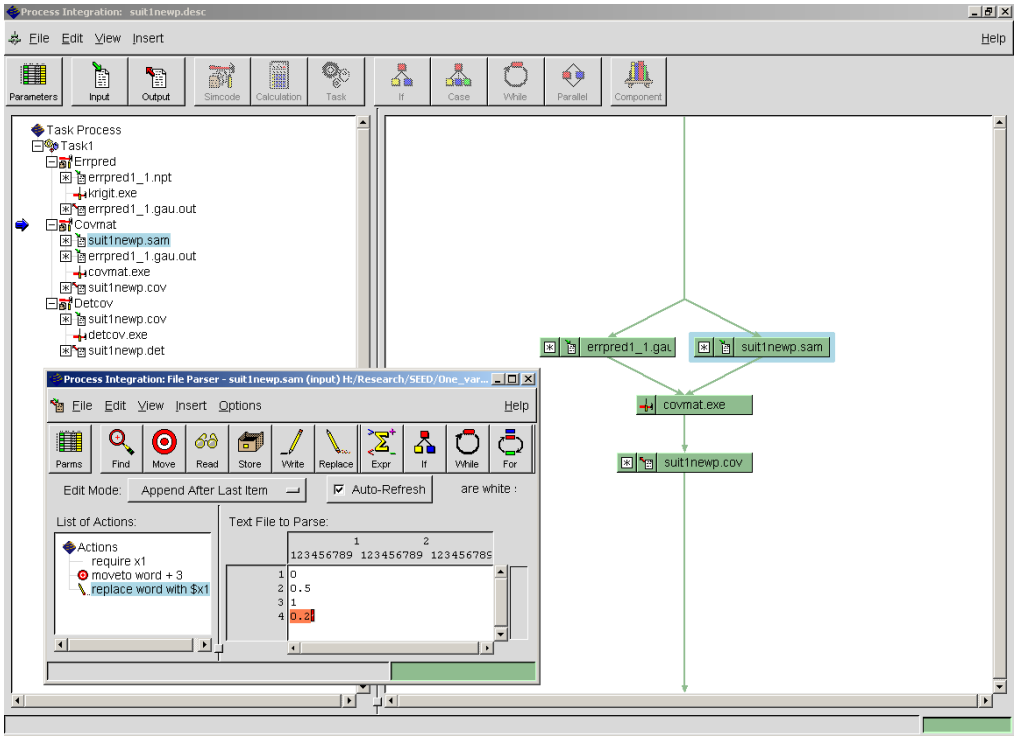


Figure A.5 File Parsing in iSIGHT (Formulation II) – Iteration I, Step 7

APPENDIX B

METAMODEL COMPARISON, SELECTION, AND SEQUENTIAL METAMODELING

This appendix is intended to supplement the study of different types of metamodels and the development of the sequential metamodeling approach in Chapter 5. Supporting materials for studies in Section 5.2 are presented in Section B.1. Supporting materials for studies in Section 5.3 are presented in Section B.2. Supporting materials for studies in Section 5.5 are presented in Section B.3.

B.1 COMPARISON OF KRIGING AND MARS METAMODELS

The regression splines metamodels developed for the single-variable function in Section 5.2 are presented here. These metamodels are developed with the computer codes written by Dr. Victoria Chen. Only the files qmars.dat are presented.

QMARS.dat (6 Data Points):

```
  1   4
     0.500000000000000
     0.500000000000000
  1   1   1   1
     0.742514189859821     -3.782619142636539      0.078497782391407      1.852499728306856     -
0.592897867942160
 -1   1     -0.8000     -0.6000     -0.4000
  1   1     -0.8000     -0.6000     -0.4000
 -1   1     -0.4000     -0.2000      0.0000
 -1   1      0.0000      0.2000      0.5000
```

QMARS.dat (12 Data Points):

```
  1   8
     0.500000000000000
     0.500000000000000
  1   1   1   1   1   1   1   1
     0.804456738034367     -3.931699607694008      0.031300702570219      0.335180979819468     -
1.092464920190181     -0.166540234100570      0.465258270418457     -0.314632853306543
0.519951496439981
 -1   1     -0.909090909     -0.818181818     -0.7272727270
  1   1     -0.909090909     -0.818181818     -0.7272727270
 -1   1     -0.363636363     -0.272727272     -0.1818181810
 -1   1     -0.727272727     -0.636363636     -0.5454545450
 -1   1      0.363636363      0.454545454      0.5909090905
 -1   1     -0.181818181     -0.090909090      0.0454545465
 -1   1      0.090909091      0.272727272      0.3636363630
 -1   1     -0.545454545     -0.454545454     -0.3636363630
```

QMARS.dat (18 Data Points):

```
  1  15
     0.499999996500000
     0.499999996500000
  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
     0.769341991216719     -4.268794934636631      0.049066635497016      1.583636167930717
0.442740915755735     -3.974759760066522     -0.161586860187488      2.348742252830790     -
1.331732197131855     -0.071742968334441      0.687898025469788      0.227592654584590     -
0.140027007826507     -0.127598133593445      0.030241893070942     -0.009966859336800
 -1   1     -0.941176470588235     -0.882352941176471     -0.823529411764706
  1   1     -0.941176470588235     -0.882352941176471     -0.823529411764706
 -1   1     -0.823529411764706     -0.764705882352941     -0.705882352941176
 -1   1     -0.235294117647059     -0.176470588235294     -0.117647058823529
```

```
-1   1     -0.705882352941176     -0.647058823529412     -0.588235294117647
-1   1      0.235294117647059      0.294117647058823      0.352941176470588
-1   1     -0.588235294117647     -0.529411764705882     -0.470588235294118
-1   1     -0.470588235294118     -0.411764705882353     -0.352941176470588
-1   1      0.470588235294118      0.529411764705882      0.588235294117647
-1   1     -0.352941176470588     -0.294117647058823     -0.235294117647059
-1   1     -0.117647058823529     -0.058823529411765      0.029411764705882
-1   1      0.058823529411765      0.176470588235294      0.235294117647059
-1   1      0.352941176470588      0.411764705882353      0.470588235294118
-1   1      0.764705882352941      0.882352941176471      0.941176470588235
-1   1      0.588235294117647      0.647058823529412      0.735294117647059
```

QMARS.dat (65 Data Points):

```
    1   26
        0.500000000000000
        0.500000000000000
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
        0.814748875040033        3.549664011717383        0.024521568179330
0.430340401178658       -9.134154080734954       -0.193694588044647       -
4.892401712101362        5.960830477768092       -1.570290211352681       -
0.438853117041401       -1.976226001655917      -23.619604594495684
17.710681769470721       19.282322168852357      -11.949572131038920       -
2.657872090330002        6.967264825385147       -4.752009346821804       -
1.643991566805389        0.230258560128408       -0.217743722561814
0.431372119205884       -0.083341921457411        0.077691137470582
0.158336056791558        0.296070952762389        0.081009268717310
 -1    1       -0.859375000000000       -0.843750000000000       -
0.828125000000000
  1    1       -0.859375000000000       -0.843750000000000       -
0.828125000000000
 -1    1       -0.265625000000000       -0.218750000000000       -
0.171875000000000
 -1    1       -0.671875000000000       -0.656250000000000       -
0.640625000000000
 -1    1        0.296875000000000        0.375000000000000
0.453125000000000
 -1    1       -0.984375000000000       -0.968750000000000       -
0.953125000000000
 -1    1       -0.515625000000000       -0.500000000000000       -
0.484375000000000
 -1    1       -0.703125000000000       -0.687500000000000       -
0.671875000000000
 -1    1       -0.421875000000000       -0.406250000000000       -
0.382812500000000
 -1    1       -0.578125000000000       -0.562500000000000       -
0.546875000000000
 -1    1       -0.765625000000000       -0.750000000000000       -
0.734375000000000
 -1    1       -0.734375000000000       -0.718750000000000       -
0.703125000000000
 -1    1       -0.796875000000000       -0.781250000000000       -
0.765625000000000
 -1    1       -0.828125000000000       -0.812500000000000       -
0.796875000000000
 -1    1       -0.453125000000000       -0.437500000000000       -
0.421875000000000
 -1    1       -0.640625000000000       -0.625000000000000       -
0.609375000000000
 -1    1       -0.609375000000000       -0.593750000000000       -
0.578125000000000
```

```
 -1    1       -0.906250000000000       -0.875000000000000       -
0.859375000000000
 -1    1       -0.171875000000000       -0.125000000000000       -
0.093750000000000
 -1    1        0.078125000000000        0.218750000000000
0.296875000000000
 -1    1       -0.359375000000000       -0.312500000000000       -
0.265625000000000
 -1    1        0.453125000000000        0.531250000000000
0.648437500000000
 -1    1       -0.953125000000000       -0.937500000000000       -
0.914062500000000
 -1    1       -0.093750000000000       -0.062500000000000       -
0.015625000000000
 -1    1       -0.484375000000000       -0.468750000000000       -
0.453125000000000
 -1    1       -0.546875000000000       -0.531250000000000       -
0.515625000000000
```

QMARS.dat (201 Data Points):

```
    1   26
        0.500000000000000
        0.500000000000000
    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
1    1    1    1    1    1    1    1    1    1
        0.784491961559237        2.905122792062056        0.042860015545419
0.427947997651711       -3.760823714142479       -0.382420644801848
6.611500445700711       12.432798782937748      -15.005696487610892      -
10.259740596893510       12.252316879797418       -0.894864223631819       -
3.011698787841257      -20.865240202107969       -6.460929739742117      -
10.056190560758544       18.419131225604275        9.859346092078775       -
3.588518685680111       -1.418510120164459       -2.306032232475570
0.358212982095669      -10.642612994172065        0.442563362726601
5.410178765440429        9.941634063124194       -0.089060082973616
 -1    1       -0.860000000000000       -0.840000000000000       -
0.825000000000000
  1    1       -0.860000000000000       -0.840000000000000       -
0.825000000000000
 -1    1       -0.260000000000000       -0.210000000000000       -
0.155000000000000
 -1    1       -0.665000000000000       -0.660000000000000       -
0.652500000000000
 -1    1        0.260000000000000        0.350000000000000
0.485000000000000
 -1    1       -0.535000000000000       -0.500000000000000       -
0.470000000000000
 -1    1       -0.705000000000000       -0.700000000000000       -
0.692500000000000
 -1    1       -0.735000000000000       -0.730000000000000       -
0.725000000000000
 -1    1       -0.690000000000000       -0.680000000000000       -
0.675000000000000
 -1    1       -0.775000000000000       -0.770000000000000       -
0.762500000000000
 -1    1       -0.430000000000000       -0.420000000000000       -
0.405000000000000
 -1    1       -0.585000000000000       -0.570000000000000       -
0.547500000000000
 -1    1       -0.755000000000000       -0.740000000000000       -
0.735000000000000
 -1    1       -0.990000000000000       -0.980000000000000       -
0.965000000000000
```

```
 -1    1       -0.825000000000000       -0.810000000000000       -
0.795000000000000
 -1    1       -0.715000000000000       -0.710000000000000       -
0.705000000000000
 -1    1       -0.650000000000000       -0.640000000000000       -
0.625000000000000
 -1    1       -0.620000000000000       -0.600000000000000       -
0.585000000000000
 -1    1       -0.930000000000000       -0.880000000000000       -
0.860000000000000
 -1    1       -0.470000000000000       -0.440000000000000       -
0.430000000000000
 -1    1       -0.155000000000000       -0.100000000000000       -
0.017500000000000
 -1    1       -0.675000000000000       -0.670000000000000       -
0.665000000000000
 -1    1       -0.365000000000000       -0.310000000000000       -
0.260000000000000
 -1    1       -0.795000000000000       -0.780000000000000       -
0.775000000000000
 -1    1       -0.725000000000000       -0.720000000000000       -
0.715000000000000
 -1    1        0.035000000000000        0.170000000000000
0.260000000000000
```

QMARS.dat (13 Data Points):

```
  1  11
     0.280000000000000
     0.500000000000000
  1   1   1   1   1   1   1   1   1   1   1
     0.818148952861391    -10.592469590022956      0.024055246830793      1.141505735575954    -
19.190225735854213     36.236320885414720    -33.743756350529544     19.536054858076000
4.098478120463438    -2.546356064901451    -0.465685671146863     1.709644342307768
 -1   1    -0.465000000000000    -0.370000000000000    -0.355000000000000
  1   1    -0.392500000000000    -0.370000000000000    -0.355000000000000
 -1   1     0.130000000000000     0.240000000000000     0.405000000000000
 -1   1    -0.250000000000000    -0.230000000000000    -0.210000000000000
 -1   1    -0.285000000000000    -0.270000000000000    -0.250000000000000
 -1   1    -0.320000000000000    -0.300000000000000    -0.285000000000000
 -1   1    -0.355000000000000    -0.340000000000000    -0.320000000000000
 -1   1    -0.125000000000000    -0.060000000000000    -0.020000000000000
 -1   1    -0.020000000000000     0.020000000000000     0.080000000000000
 -1   1     0.510000000000000     0.780000000000000     1.110000000000000
 -1   1    -0.210000000000000    -0.190000000000000    -0.160000000000000
```

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 13, 13, 50, 2, 1
v 1 count[v] 13 levels
T set to p-2 (11).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 7 value 0.026000
v 1 t 2 knot[v][t] 2 value 0.091000
v 1 t 3 knot[v][t] 8 value 0.215000
v 1 t 4 knot[v][t] 9 value 0.289000
v 1 t 5 knot[v][t] 3 value 0.331000
v 1 t 6 knot[v][t] 10 value 0.414000
```

```
v 1 t 7 knot[v][t] 4 value 0.500000
v 1 t 8 knot[v][t] 11 value 0.582000
v 1 t 9 knot[v][t] 5 value 0.669000
v 1 t 10 knot[v][t] 12 value 0.785000
v 1 t 11 knot[v][t] 13 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000192
m 0 v 1 t 1 I 33.889683071877734 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 6 I 40.171443343286093 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 9 I 65.506933558794287 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 3 I 89.350140883724464 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 2 I 57.490695569612832 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 5 I 3.907723530036343 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 10 I 1.184287438418364 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 8 I 0.253932969820230 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 7 I 1.099315592147945 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 11 I 0.025664992346307 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 4 I 0.000647036211367 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 4 I 0.000000000000000 zero 1 1 2 0 M=24, onM=12
For N=13, onM=12, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000000841432161 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12
  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000000841469089
quintic lof_bst is 0.000429884029849
quintic lof_bst without penalty is lof*0.005917159763314=0.000002543692484
m 1 split 1 cov 1 knots  -0.868000  -0.842000  -0.803000 s -1
m 2 split 1 cov 1 knots  -0.868000  -0.842000  -0.777000 s 1
m 3 split 1 cov 1 knots  -0.149000  -0.066000   0.020000 s -1
m 4 split 1 cov 1 knots   0.357000   0.444000   0.560000 s -1
m 5 split 1 cov 1 knots  -0.588000  -0.464000  -0.348000 s -1
m 6 split 1 cov 1 knots  -0.777000  -0.712000  -0.614500 s -1
m 7 split 1 cov 1 knots  -0.348000  -0.232000  -0.149000 s -1
m 8 split 1 cov 1 knots   0.560000   0.676000   0.800000 s -1
m 9 split 1 cov 1 knots   0.188000   0.270000   0.357000 s -1
m 10 split 1 cov 1 knots   0.020000   0.106000   0.188000 s -1
m 11 split 1 cov 1 knots   0.800000   0.924000   1.015000 s -1
```

B.2 UTILIZATION OF DIFFERENT TYPES OF METAMODELS IN SEED

The regression splines metamodels developed for the single-variable function in Section 5.3 are presented here. Only the files qmars.dat are presented here.

REGRESSION SPLINE Metamodel of Prediction Errors in Iteration I – Step 4 (with 4 data points and 5 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 9, 9, 30, 2, 1
v 1 count[v] 9 levels
T set to p-2 (7).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 5 value 0.091000
v 1 t 2 knot[v][t] 6 value 0.215000
v 1 t 3 knot[v][t] 2 value 0.331000
v 1 t 4 knot[v][t] 7 value 0.500000
v 1 t 5 knot[v][t] 3 value 0.669000
v 1 t 6 knot[v][t] 8 value 0.785000
v 1 t 7 knot[v][t] 9 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000278
m 0 v 1 t 1 I 2185.235914437491400 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 3 I 1779.564047921492600 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 2 I 211.881913968762090 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 4 I 70.600513313779658 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 5 I 70.390066900327525 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 1.153538313945575 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 7 I 0.992432478524021 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 7 I 0.000000000000000 zero 1 1 2 0 M=16, onM=8
For N=9, onM=8, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000893189952348 with J_bst:
  1  2  3  4  5  6  7  8
  1  1  1  1  1  1  1  0
lof_bst= 0.000482844283989 with J_bst:
  1  2  3  4  5  6  7  8
  1  1  1  1  1  1  0  0
linear lof_bst is 0.000482844295849
quintic lof_bst is 0.000223574353830
quintic lof_bst without penalty is lof*0.049382716049383=0.000011040708831
m 1 split 1 cov 1 knots  -0.909000  -0.818000  -0.694000 s -1
m 2 split 1 cov 1 knots  -0.909000  -0.818000  -0.694000 s 1
m 3 split 1 cov 1 knots  -0.454000  -0.338000  -0.169000 s -1
m 4 split 1 cov 1 knots  -0.694000  -0.570000  -0.454000 s -1
m 5 split 1 cov 1 knots  -0.169000   0.000000   0.169000 s -1
m 6 split 1 cov 1 knots   0.169000   0.338000   0.591500 s -1
```

REGRESSION SPLINE Metamodel of Responses in Iteration I – Step 8 (6 data points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 6, 6, 50, 2, 1
v 1 count[v] 6 levels
T set to p-2 (4).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 5 value 0.091000
v 1 t 2 knot[v][t] 2 value 0.331000
v 1 t 3 knot[v][t] 6 value 0.500000
v 1 t 4 knot[v][t] 3 value 0.669000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000417
m 0 v 1 t 1 I 4803.027518465813000 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 2 I 67.979517786175649 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 4 I 18.226084000257934 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 3 I 28.470572680609678 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 3 I 0.000000000000000 zero 1 1 2 0 M=10, onM=5
For N=6, onM=5, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.010935660807123 with J_bst:
  1  2  3  4  5
  1  1  1  0  1
lof_bst= 0.007004537701781 with J_bst:
  1  2  3  4  5
  1  1  1  0  0
linear lof_bst is 0.007004537701936
quintic lof_bst is 0.007265056347063
quintic lof_bst without penalty is lof*0.111111111111111=0.000807228483007
m 1 split 1 cov 1 knots  -0.772667  -0.681667  -0.545167 s -1
m 2 split 1 cov 1 knots  -0.772667  -0.681667  -0.441667 s 1
m 3 split 1 cov 1 knots  -0.441667  -0.201667   0.158333 s -1
```

REGRESSION SPLINE Metamodel of Prediction Errors in Iteration II – Step 3 (with 3 data points and 6 validation points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 9, 9, 50, 2, 1
v 1 count[v] 9 levels
T set to p-2 (7).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 5 value 0.091000
v 1 t 2 knot[v][t] 7 value 0.215000
v 1 t 3 knot[v][t] 2 value 0.331000
v 1 t 4 knot[v][t] 6 value 0.500000
v 1 t 5 knot[v][t] 3 value 0.669000
v 1 t 6 knot[v][t] 8 value 0.785000
v 1 t 7 knot[v][t] 9 value 0.909000
```

```
Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000278
m 0 v 1 t 1 I 5237.207545019747200 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 4 I 82.908493275196250 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 5 I 63.546597755261850 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 2 I 75.747176648623253 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 3 I 82.351740441337313 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 1.394399483913649 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 7 I 0.026953414014063 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 7 I 0.000000000000000 zero 1 1 2 0 M=16, onM=8
For N=9, onM=8, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000024258092213 with J_bst:
  1  2  3  4  5  6  7  8
  1  1  1  1  1  1  1  0
linear lof_bst is 0.000024258116828
quintic lof_bst is 0.000041249572296
quintic lof_bst without penalty is lof*0.012345679012346=0.000000509253979
m 1 split 1 cov 1 knots  -0.909000  -0.818000  -0.694000 s -1
m 2 split 1 cov 1 knots  -0.909000  -0.818000  -0.694000 s 1
m 3 split 1 cov 1 knots  -0.169000   0.000000   0.169000 s -1
m 4 split 1 cov 1 knots   0.169000   0.338000   0.454000 s -1
m 5 split 1 cov 1 knots  -0.694000  -0.570000  -0.454000 s -1
m 6 split 1 cov 1 knots  -0.454000  -0.338000  -0.169000 s -1
m 7 split 1 cov 1 knots   0.454000   0.570000   0.744000 s -1
```

REGRESSION SPLINE Metamodel of Responses in Iteration II – Step 8 (with 8 data points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 8, 8, 50, 2, 1
v 1 count[v] 8 levels
T set to p-2 (6).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 2 value 0.091000
v 1 t 2 knot[v][t] 3 value 0.215000
v 1 t 3 knot[v][t] 4 value 0.331000
v 1 t 4 knot[v][t] 5 value 0.500000
v 1 t 5 knot[v][t] 6 value 0.669000
v 1 t 6 knot[v][t] 7 value 0.833000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000313
m 0 v 1 t 1 I 5364.836771030441900 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 4 I 86.810040802686899 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 5 I 54.955163539854851 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 2 I 75.668742279855266 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 3 I 82.411207787488465 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 0.849650086301520 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 6 I 0.000000000000000 zero 1 1 2 0 M=14, onM=7
For N=8, onM=7, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000679720795882 with J_bst:
```

```
  1  2  3  4  5  6  7
  1  1  1  1  1  1  0
linear lof_bst is 0.000679720812870
quintic lof_bst is 0.000030046700078
quintic lof_bst without penalty is lof*0.015625000000000=0.000000469479689
m 1 split 1 cov 1 knots  -0.818750  -0.727750  -0.603750 s -1
m 2 split 1 cov 1 knots  -0.818750  -0.727750  -0.603750 s 1
m 3 split 1 cov 1 knots  -0.078750   0.090250   0.259250 s -1
m 4 split 1 cov 1 knots   0.259250   0.428250   0.681750 s -1
m 5 split 1 cov 1 knots  -0.603750  -0.479750  -0.363750 s -1
m 6 split 1 cov 1 knots  -0.363750  -0.247750  -0.078750 s -1
```

REGRESSION SPLINE Metamodel of Responses in Iteration III – Step 3 (with 8 data points and 6 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 6, 6, 50, 2, 1
v 1 count[v] 6 levels
T set to p-2 (4).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 2 value 0.289000
v 1 t 2 knot[v][t] 3 value 0.414000
v 1 t 3 knot[v][t] 4 value 0.582000
v 1 t 4 knot[v][t] 5 value 0.785000

Min/Max x-values:
v 1 min 0.026000 max 0.909000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000417
m 0 v 1 t 1 I 1970.567559425040800 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 2 I 75.588720887641529 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 3 I 29.659951455774380 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 4 I 6.802131324235522 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 4 I 0.000000000000000 zero 1 1 2 0 M=10, onM=5
For N=6, onM=5, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.004081281493572 with J_bst:
  1  2  3  4  5
  1  1  1  1  0
linear lof_bst is 0.004081281503242
quintic lof_bst is 0.002639744126298
quintic lof_bst without penalty is lof*0.027777777777778=0.000073326225731
m 1 split 1 cov 1 knots  -0.777652  -0.479804  -0.338241 s -1
m 2 split 1 cov 1 knots  -0.692148  -0.479804  -0.338241 s 1
m 3 split 1 cov 1 knots  -0.338241  -0.196678  -0.006418 s -1
m 4 split 1 cov 1 knots  -0.006418   0.183843   0.469234 s -1
```

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 14, 14, 50, 2, 1
```

```
v 1 count[v] 14 levels
T set to p-2 (12).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 9 value 0.026000
v 1 t 2 knot[v][t] 2 value 0.091000
v 1 t 3 knot[v][t] 3 value 0.215000
v 1 t 4 knot[v][t] 10 value 0.289000
v 1 t 5 knot[v][t] 4 value 0.331000
v 1 t 6 knot[v][t] 11 value 0.414000
v 1 t 7 knot[v][t] 5 value 0.500000
v 1 t 8 knot[v][t] 12 value 0.582000
v 1 t 9 knot[v][t] 6 value 0.669000
v 1 t 10 knot[v][t] 13 value 0.785000
v 1 t 11 knot[v][t] 7 value 0.833000
v 1 t 12 knot[v][t] 14 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 2 I 1104.034446933376800 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 5 I 641.727010256131280 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 1 I 68.699732858564118 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 3 I 27.511075114669328 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 4 I 35.085611876807420 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 10 I 18.609337638162899 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 6 I 2.143573078245412 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 7 I 1.766371130643947 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 9 I 0.704470670560657 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 8 I 0.783995247638147 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 12 I 0.025600117142965 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 11 I 0.001473995613301 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 11 I 0.000000000000000 zero 1 1 2 0 M=26, onM=13
For N=14, onM=13, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000002063615280 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000002063753677
quintic lof_bst is 0.000005698330870
quintic lof_bst without penalty is lof*0.005102040816327=0.000000029073117
m 1 split 1 cov 1 knots  -0.832143  -0.767143  -0.669643 s -1
m 2 split 1 cov 1 knots  -0.832143  -0.767143  -0.643143 s 1
m 3 split 1 cov 1 knots  -0.329143  -0.287143  -0.224143 s -1
m 4 split 1 cov 1 knots  -0.923143  -0.897143  -0.858143 s -1
m 5 split 1 cov 1 knots  -0.643143  -0.519143  -0.445143 s -1
m 6 split 1 cov 1 knots  -0.445143  -0.371143  -0.329143 s -1
m 7 split 1 cov 1 knots   0.504857   0.620857   0.744857 s -1
m 8 split 1 cov 1 knots  -0.204143  -0.121143  -0.035143 s -1
m 9 split 1 cov 1 knots  -0.035143   0.050857   0.132857 s -1
m 10 split 1 cov 1 knots   0.301857   0.388857   0.504857 s -1
m 11 split 1 cov 1 knots   0.132857   0.214857   0.301857 s -1
m 12 split 1 cov 1 knots   0.744857   0.868857   0.959857 s -1
```

REGRESSION SPLINE Metamodel of Responses II in Iteration III – Step 3 (with 6 data points and 2 data points and 6 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
```

```
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 8, 8, 50, 2, 1
v 1 count[v] 8 levels
T set to p-2 (6).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 3 value 0.026000
v 1 t 2 knot[v][t] 4 value 0.289000
v 1 t 3 knot[v][t] 5 value 0.414000
v 1 t 4 knot[v][t] 6 value 0.582000
v 1 t 5 knot[v][t] 7 value 0.785000
v 1 t 6 knot[v][t] 8 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000313
m 0 v 1 t 1 I 6301.198254006109900 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 2 I 310.957987054077080 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 3 I 57.583263028843767 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 5 I 50.498017544867572 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 4 I 10.043555448854978 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 0.026299124785490 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 6 I 0.000000000000000 zero 1 1 2 0 M=14, onM=7
For N=8, onM=7, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000021039316562 with J_bst:
  1  2  3  4  5  6  7
  1  1  1  1  1  1  0
linear lof_bst is 0.000021039355873
quintic lof_bst is 0.000155486029099
quintic lof_bst without penalty is lof*0.015625000000000=0.000002429469205
m 1 split 1 cov 1 knots  -0.975250  -0.949250  -0.910250 s -1
m 2 split 1 cov 1 knots  -0.975250  -0.949250  -0.686250 s 1
m 3 split 1 cov 1 knots  -0.686250  -0.423250  -0.298250 s -1
m 4 split 1 cov 1 knots  -0.298250  -0.173250  -0.005250 s -1
m 5 split 1 cov 1 knots   0.365750   0.568750   0.783750 s -1
m 6 split 1 cov 1 knots  -0.005250   0.162750   0.365750 s -1
```

REGRESSION SPLINE Metamodel of Prediction Errors II in Iteration III – Step 3 (with 6 data points, 2 data points, and 6 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 14, 14, 50, 2, 1
v 1 count[v] 14 levels
T set to p-2 (12).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 9 value 0.026000
v 1 t 2 knot[v][t] 2 value 0.091000
v 1 t 3 knot[v][t] 3 value 0.215000
v 1 t 4 knot[v][t] 10 value 0.289000
v 1 t 5 knot[v][t] 4 value 0.331000
v 1 t 6 knot[v][t] 11 value 0.414000
v 1 t 7 knot[v][t] 5 value 0.500000
v 1 t 8 knot[v][t] 12 value 0.582000
v 1 t 9 knot[v][t] 6 value 0.669000
v 1 t 10 knot[v][t] 13 value 0.785000
v 1 t 11 knot[v][t] 7 value 0.833000
```

```
v 1 t 12 knot[v][t] 14 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 2 I 339.173680209830590 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 5 I 467.274744487073750 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 3 I 40.935521600356182 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 1 I 33.590584833269411 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 4 I 30.608378933262543 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 7.133305614126354 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 9 I 0.872305796962173 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 11 I 0.643541172744865 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 7 I 0.826719505032901 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 8 I 0.143614638849335 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 12 I 0.049901920769187 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 10 I 0.000054162747859 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 10 I 0.000000000000000 zero 1 1 2 0 M=26, onM=13
For N=14, onM=13, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000000075828957 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000000075954298
quintic lof_bst is 0.000000399706621
quintic lof_bst without penalty is lof*0.005102040816327=0.000000002039319
m 1 split 1 cov 1 knots  -0.832143  -0.767143  -0.669643 s -1
m 2 split 1 cov 1 knots  -0.832143  -0.767143  -0.643143 s 1
m 3 split 1 cov 1 knots  -0.329143  -0.287143  -0.224143 s -1
m 4 split 1 cov 1 knots  -0.643143  -0.519143  -0.445143 s -1
m 5 split 1 cov 1 knots  -0.923143  -0.897143  -0.858143 s -1
m 6 split 1 cov 1 knots  -0.445143  -0.371143  -0.329143 s -1
m 7 split 1 cov 1 knots  -0.204143  -0.121143  -0.035143 s -1
m 8 split 1 cov 1 knots   0.301857   0.388857   0.519357 s -1
m 9 split 1 cov 1 knots   0.552857   0.716857   0.792857 s -1
m 10 split 1 cov 1 knots  -0.035143   0.050857   0.132857 s -1
m 11 split 1 cov 1 knots   0.132857   0.214857   0.301857 s -1
m 12 split 1 cov 1 knots   0.792857   0.868857   0.959857 s -1
```

Intermediate REGRESSION SPLINE Metamodel of Responses (with 17 data points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 17, 17, 50, 2, 1
v 1 count[v] 17 levels
T set to p-2 (15).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 9 value 0.026000
v 1 t 2 knot[v][t] 10 value 0.071000
v 1 t 3 knot[v][t] 2 value 0.091000
v 1 t 4 knot[v][t] 11 value 0.151000
v 1 t 5 knot[v][t] 3 value 0.215000
v 1 t 6 knot[v][t] 12 value 0.243000
v 1 t 7 knot[v][t] 13 value 0.289000
v 1 t 8 knot[v][t] 4 value 0.331000
v 1 t 9 knot[v][t] 14 value 0.414000
v 1 t 10 knot[v][t] 5 value 0.500000
```

```
v 1 t 11 knot[v][t] 15 value 0.582000
v 1 t 12 knot[v][t] 6 value 0.669000
v 1 t 13 knot[v][t] 16 value 0.785000
v 1 t 14 knot[v][t] 7 value 0.833000
v 1 t 15 knot[v][t] 17 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000147
m 0 v 1 t 2 I 7269.432967947485800 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 9 I 75.293730914431393 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 5 I 93.431176329404067 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 12 I 111.320725496105170 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 4 I 70.560647626696408 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 6 I 126.027225668510250 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 3 I 64.635104556370621 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 1 I 63.947382383630810 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 7 I 50.840791638568675 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 8 I 2.945047030187512 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 13 I 1.568282759622121 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 10 I 0.316523842043537 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 11 I 1.173101721484663 zero 1 1 2 1 M=27, onM=14
m 0 v 1 t 15 I 0.025330077475867 zero 1 1 2 1 M=29, onM=15
m 0 v 1 t 14 I 0.001812327231944 zero 1 1 2 1 M=31, onM=16
m 0 v 1 t 14 I 0.000000000000000 zero 1 1 2 0 M=32, onM=16
For N=17, onM=16, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000003080989086 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000003098407401
quintic lof_bst is 0.000004917793013
quintic lof_bst without penalty is lof*0.003460207612457=0.000000017016585
m 1 split 1 cov 1 knots  -0.739353  -0.694353  -0.674353 s -1
m 2 split 1 cov 1 knots  -0.724353  -0.694353  -0.674353 s 1
m 3 split 1 cov 1 knots  -0.091353  -0.008353   0.077647 s -1
m 4 split 1 cov 1 knots  -0.470353  -0.406353  -0.378353 s -1
m 5 split 1 cov 1 knots   0.414647   0.501647   0.617647 s -1
m 6 split 1 cov 1 knots  -0.594353  -0.534353  -0.470353 s -1
m 7 split 1 cov 1 knots  -0.378353  -0.350353  -0.308353 s -1
m 8 split 1 cov 1 knots  -0.674353  -0.654353  -0.624353 s -1
m 9 split 1 cov 1 knots  -0.810353  -0.784353  -0.745353 s -1
m 10 split 1 cov 1 knots  -0.304353  -0.258353  -0.216353 s -1
m 11 split 1 cov 1 knots  -0.216353  -0.174353  -0.111353 s -1
m 12 split 1 cov 1 knots   0.617647   0.733647   0.857647 s -1
m 13 split 1 cov 1 knots   0.077647   0.163647   0.245647 s -1
m 14 split 1 cov 1 knots   0.245647   0.327647   0.414647 s -1
m 15 split 1 cov 1 knots   0.857647   0.981647   1.072647 s -1
```

REGRESSION SPLINE Metamodel of Prediction Errors in Iteration III – Step 4 (with 8 data points and 9 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 17, 17, 50, 2, 1
v 1 count[v] 17 levels
T set to p-2 (15).
```

```
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 9 value 0.026000
v 1 t 2 knot[v][t] 10 value 0.071000
v 1 t 3 knot[v][t] 2 value 0.091000
v 1 t 4 knot[v][t] 11 value 0.151000
v 1 t 5 knot[v][t] 3 value 0.215000
v 1 t 6 knot[v][t] 12 value 0.243000
v 1 t 7 knot[v][t] 13 value 0.289000
v 1 t 8 knot[v][t] 4 value 0.331000
v 1 t 9 knot[v][t] 14 value 0.414000
v 1 t 10 knot[v][t] 5 value 0.500000
v 1 t 11 knot[v][t] 15 value 0.582000
v 1 t 12 knot[v][t] 6 value 0.669000
v 1 t 13 knot[v][t] 16 value 0.785000
v 1 t 14 knot[v][t] 7 value 0.833000
v 1 t 15 knot[v][t] 17 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000147
m 0 v 1 t 4 I 282.133557720369200 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 8 I 156.279296509230140 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 1 I 121.718667887611250 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 5 I 31.115971263869707 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 6 I 148.141265141702890 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 7 I 54.196704995797312 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 3 I 28.193625034420087 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 2 I 9.020780410731158 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 11 I 1.573669673836520 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 9 I 6.461532051825494 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 10 I 0.996209068593577 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 12 I 0.496281852568706 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 13 I 0.080533179549255 zero 1 1 2 1 M=27, onM=14
m 0 v 1 t 15 I 0.024430331786970 zero 1 1 2 1 M=29, onM=15
m 0 v 1 t 14 I 0.001454009081724 zero 1 1 2 1 M=31, onM=16
m 0 v 1 t 14 I 0.000000000000000 zero 1 1 2 0 M=32, onM=16
For N=17, onM=16, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000002471841746 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000002489316544
quintic lof_bst is 0.000000078125069
quintic lof_bst without penalty is lof*0.003460207612457=0.000000000270329
m 1 split 1 cov 1 knots  -0.594353  -0.534353  -0.470353 s -1
m 2 split 1 cov 1 knots  -0.594353  -0.534353  -0.470353 s 1
m 3 split 1 cov 1 knots  -0.216353  -0.174353  -0.111353 s -1
m 4 split 1 cov 1 knots  -0.810353  -0.784353  -0.745353 s -1
m 5 split 1 cov 1 knots  -0.470353  -0.406353  -0.378353 s -1
m 6 split 1 cov 1 knots  -0.378353  -0.350353  -0.308353 s -1
m 7 split 1 cov 1 knots  -0.304353  -0.258353  -0.216353 s -1
m 8 split 1 cov 1 knots  -0.674353  -0.654353  -0.624353 s -1
m 9 split 1 cov 1 knots  -0.739353  -0.694353  -0.674353 s -1
m 10 split 1 cov 1 knots   0.245647   0.327647   0.414647 s -1
m 11 split 1 cov 1 knots  -0.091353  -0.008353   0.077647 s -1
m 12 split 1 cov 1 knots   0.077647   0.163647   0.245647 s -1
m 13 split 1 cov 1 knots   0.414647   0.501647   0.617647 s -1
m 14 split 1 cov 1 knots   0.617647   0.733647   0.857647 s -1
m 15 split 1 cov 1 knots   0.857647   0.981647   1.072647 s -1
```

Final Regression Spline Metamodel of Responses in Iteration III – Step 8 (with 17 points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 19, 19, 50, 2, 1
v 1 count[v] 19 levels
T set to p-2 (17).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 11 value 0.026000
v 1 t 2 knot[v][t] 12 value 0.071000
v 1 t 3 knot[v][t] 2 value 0.091000
v 1 t 4 knot[v][t] 3 value 0.126000
v 1 t 5 knot[v][t] 13 value 0.151000
v 1 t 6 knot[v][t] 4 value 0.215000
v 1 t 7 knot[v][t] 14 value 0.243000
v 1 t 8 knot[v][t] 5 value 0.254000
v 1 t 9 knot[v][t] 15 value 0.289000
v 1 t 10 knot[v][t] 6 value 0.331000
v 1 t 11 knot[v][t] 16 value 0.414000
v 1 t 12 knot[v][t] 7 value 0.500000
v 1 t 13 knot[v][t] 17 value 0.582000
v 1 t 14 knot[v][t] 8 value 0.669000
v 1 t 15 knot[v][t] 18 value 0.785000
v 1 t 16 knot[v][t] 9 value 0.833000
v 1 t 17 knot[v][t] 19 value 0.909000

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000132
m 0 v 1 t 2 I 7168.542631419148200 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 11 I 205.645631822369320 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 4 I 151.450532648740650 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 5 I 314.886298981557500 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 6 I 95.209332861043208 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 7 I 260.087296180865560 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 14 I 77.475774020079555 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 9 I 71.272630255421504 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 1 I 62.989582271940449 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 8 I 15.840252941947378 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 10 I 2.944981081412627 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 15 I 1.568282702010469 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 3 I 0.998537593401555 zero 1 1 2 1 M=27, onM=14
m 0 v 1 t 12 I 0.316523281034097 zero 1 1 2 1 M=29, onM=15
m 0 v 1 t 13 I 1.173099698582701 zero 1 1 2 1 M=31, onM=16
m 0 v 1 t 17 I 0.025330077333144 zero 1 1 2 1 M=33, onM=17
m 0 v 1 t 16 I 0.001812327000035 zero 1 1 2 1 M=35, onM=18
m 0 v 1 t 16 I 0.000000000000000 zero 1 1 2 0 M=36, onM=18
For N=19, onM=18, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000003443458390 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.000003464607892
quintic lof_bst is 0.000005476011637
quintic lof_bst without penalty is lof*0.002770083102493=0.000000015169007
m 1 split 1 cov 1 knots  -0.691316  -0.646316  -0.626316 s -1
m 2 split 1 cov 1 knots  -0.676316  -0.646316  -0.626316 s 1
m 3 split 1 cov 1 knots  -0.043316   0.039684   0.125684 s -1
m 4 split 1 cov 1 knots  -0.571316  -0.536316  -0.511316 s -1
```

```
m 5 split 1 cov 1 knots  -0.511316  -0.486316  -0.448816 s -1
m 6 split 1 cov 1 knots  -0.422316  -0.358316  -0.330316 s -1
m 7 split 1 cov 1 knots  -0.330316  -0.302316  -0.291316 s -1
m 8 split 1 cov 1 knots   0.462684   0.549684   0.665684 s -1
m 9 split 1 cov 1 knots  -0.245316  -0.210316  -0.168316 s -1
m 10 split 1 cov 1 knots  -0.762316  -0.736316  -0.697316 s -1
m 11 split 1 cov 1 knots  -0.291316  -0.280316  -0.263816 s -1
m 12 split 1 cov 1 knots  -0.168316  -0.126316  -0.063316 s -1
m 13 split 1 cov 1 knots   0.665684   0.781684   0.905684 s -1
m 14 split 1 cov 1 knots  -0.626316  -0.606316  -0.576316 s -1
m 15 split 1 cov 1 knots   0.125684   0.211684   0.293684 s -1
m 16 split 1 cov 1 knots   0.293684   0.375684   0.462684 s -1
m 17 split 1 cov 1 knots   0.905684   1.029684   1.120684 s -1
```

Single-Stage Experiments and Corresponding Regression Spline Metamodel of Responses (with 19 points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 1, 0, 19, 19, 50, 2, 1
v 1 count[v] 19 levels
T set to p-2 (17).
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 2 value 0.055556
v 1 t 2 knot[v][t] 3 value 0.111111
v 1 t 3 knot[v][t] 4 value 0.166667
v 1 t 4 knot[v][t] 5 value 0.222222
v 1 t 5 knot[v][t] 6 value 0.277778
v 1 t 6 knot[v][t] 7 value 0.333333
v 1 t 7 knot[v][t] 8 value 0.388889
v 1 t 8 knot[v][t] 9 value 0.444444
v 1 t 9 knot[v][t] 10 value 0.500000
v 1 t 10 knot[v][t] 11 value 0.555556
v 1 t 11 knot[v][t] 12 value 0.611111
v 1 t 12 knot[v][t] 13 value 0.666667
v 1 t 13 knot[v][t] 14 value 0.722222
v 1 t 14 knot[v][t] 15 value 0.777778
v 1 t 15 knot[v][t] 16 value 0.833333
v 1 t 16 knot[v][t] 17 value 0.888889
v 1 t 17 knot[v][t] 18 value 0.944444

Min/Max x-values:
v 1 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000132
m 0 v 1 t 2 I 6380.515450402902400 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 7 I 480.547397715893790 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 3 I 533.166835128732490 zero 1 1 2 1 M=7, onM=4
m 0 v 1 t 1 I 282.444647093684690 zero 1 1 2 1 M=9, onM=5
m 0 v 1 t 12 I 210.694453772197110 zero 1 1 2 1 M=11, onM=6
m 0 v 1 t 4 I 51.861511452135410 zero 1 1 2 1 M=13, onM=7
m 0 v 1 t 5 I 5.817543824950742 zero 1 1 2 1 M=15, onM=8
m 0 v 1 t 8 I 8.042719105064752 zero 1 1 2 1 M=17, onM=9
m 0 v 1 t 14 I 1.236193941396714 zero 1 1 2 1 M=19, onM=10
m 0 v 1 t 11 I 0.952642915238477 zero 1 1 2 1 M=21, onM=11
m 0 v 1 t 13 I 0.253713461247394 zero 1 1 2 1 M=23, onM=12
m 0 v 1 t 9 I 0.130664484272747 zero 1 1 2 1 M=25, onM=13
m 0 v 1 t 10 I 0.079337049073314 zero 1 1 2 1 M=27, onM=14
```

```
m 0 v 1 t 6 I 0.066123847745227 zero 1 1 2 1 M=29, onM=15
m 0 v 1 t 17 I 0.024999987650532 zero 1 1 2 1 M=31, onM=16
m 0 v 1 t 15 I 0.001079991352833 zero 1 1 2 1 M=33, onM=17
m 0 v 1 t 16 I 0.002399987137368 zero 1 1 2 1 M=35, onM=18
m 0 v 1 t 16 I 0.000000000000000 zero 1 1 2 0 M=36, onM=18
For N=19, onM=18, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000002052000131 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1
lof_bst= 0.000001653000115 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0
linear lof_bst is 0.000001653096778
quintic lof_bst is 0.000000533730253
quintic lof_bst without penalty is lof*0.011080332409972=0.000000005913909
m 1 split 1 cov 1 knots  -0.833333  -0.777778  -0.722222 s -1
m 2 split 1 cov 1 knots  -0.833333  -0.777778  -0.722222 s 1
m 3 split 1 cov 1 knots  -0.277778  -0.222222  -0.166667 s -1
m 4 split 1 cov 1 knots  -0.722222  -0.666667  -0.611111 s -1
m 5 split 1 cov 1 knots  -0.944444  -0.888889  -0.833333 s -1
m 6 split 1 cov 1 knots   0.277778   0.333333   0.388889 s -1
m 7 split 1 cov 1 knots  -0.611111  -0.555556  -0.500000 s -1
m 8 split 1 cov 1 knots  -0.500000  -0.444444  -0.388889 s -1
m 9 split 1 cov 1 knots  -0.166667  -0.111111  -0.055556 s -1
m 10 split 1 cov 1 knots   0.500000   0.555556   0.638889 s -1
m 11 split 1 cov 1 knots   0.166667   0.222222   0.277778 s -1
m 12 split 1 cov 1 knots   0.388889   0.444444   0.500000 s -1
m 13 split 1 cov 1 knots  -0.055556   0.000000   0.055556 s -1
m 14 split 1 cov 1 knots   0.055556   0.111111   0.166667 s -1
m 15 split 1 cov 1 knots  -0.388889  -0.333333  -0.277778 s -1
m 16 split 1 cov 1 knots   0.722222   0.888889   0.944444 s -1
```

B.3 EXPLORATION OF DESIGN SOLUTIONS WITH SEED

All supporting materials and documents for studies in Section 5.5 are presented here. The model files for MARS metamodels are listed in Section B.3.1. The RS metamodels of responses developed in Section 5.5.2 are listed in Section B.3.2. FORTRAN codes of the SEED method in the multi-response problem are presented in Section B.3.3. The implementation of SEED in iSIGHT is illustrated in Section B.3.4.

B.3.1 MARS Metamodels Developed in Design of the Pressure Vessels

MARS Metamodel of Prediction Errors for Volume in Iteration II – Step 3 (with 6 data points and 4 validation points):
The responses should be multiplied by 1k.
qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 10, 10, 50, 3, 1
v 1 count[v] 7 levels
v 2 count[v] 8 levels
T set to p-2 (5).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 8 value 0.224200
v 1 t 2 knot[v][t] 10 value 0.499600
v 1 t 3 knot[v][t] 6 value 0.500000
v 1 t 4 knot[v][t] 5 value 0.500900
v 1 t 5 knot[v][t] 9 value 0.786200

v 2 t 1 knot[v][t] 7 value 0.001200
v 2 t 2 knot[v][t] 6 value 0.204300
v 2 t 3 knot[v][t] 8 value 0.502500
v 2 t 4 knot[v][t] 5 value 0.767800
v 2 t 5 knot[v][t] 10 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000250
m 0 v 1 t 5 I 11657676488.734486000000000 zero 1 1 2 1 M=3, onM=2
m 1 v 2 t 5 I 1418662870.428122500000000 zero 1 1 2 1 M=5, onM=4
m 0 v 2 t 5 I 812614130.099333880000000 zero 1 1 2 1 M=7, onM=6
m 0 v 2 t 2 I 41187877.130973093000000 zero 1 1 2 1 M=9, onM=7
m 1 v 2 t 3 I 47085000.369751297000000 zero 1 1 2 1 M=11, onM=8
m 0 v 2 t 4 I 6859787.339947069100000 zero 1 1 2 1 M=13, onM=9
m 0 v 2 t 4 I 0.000000000000000 zero 1 1 2 0 M=14, onM=9
```

```
For N=10, onM=9, lof_all= 1.#INF00000000000
Alg3
lof_bst= 6859.801602373312600 with J_bst:
  1  2  3  4  5  6  7  8  9
  1  1  1  1  1  1  1  1  0
linear lof_bst is 6859.801648299139700
quintic lof_bst is 4832.855340449985300
quintic lof_bst without penalty is lof*0.010000000000000=48.328553404499864
m 1 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 2 split 1 cov 1 knots   0.249340   0.570040   0.783840 s 1
m 3 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 3 split 2 cov 2 knots   0.507000   1.004300   1.004500 s -1
m 4 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 4 split 2 cov 2 knots   1.004000   1.004300   1.004500 s 1
m 5 split 1 cov 2 knots   0.208800   1.004300   1.004500 s -1
m 6 split 1 cov 2 knots   1.004000   1.004300   1.004500 s 1
m 7 split 1 cov 2 knots  -0.791000  -0.586700  -0.280250 s -1
m 8 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 8 split 2 cov 2 knots  -0.492800   0.009700   0.507000 s -1
```

MARS Metamodel of Prediction Errors for Cost in Iteration II – Step 3 (with 6 data points and 4 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 10, 10, 50, 3, 1
v 1 count[v] 7 levels
v 2 count[v] 8 levels
T set to p-2 (5).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 8 value 0.224200
v 1 t 2 knot[v][t] 10 value 0.499600
v 1 t 3 knot[v][t] 6 value 0.500000
v 1 t 4 knot[v][t] 5 value 0.500900
v 1 t 5 knot[v][t] 9 value 0.786200

v 2 t 1 knot[v][t] 7 value 0.001200
v 2 t 2 knot[v][t] 6 value 0.204300
v 2 t 3 knot[v][t] 8 value 0.502500
v 2 t 4 knot[v][t] 5 value 0.767800
v 2 t 5 knot[v][t] 10 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000250
m 0 v 1 t 5 I 422716224254.840150000000000 zero 1 1 2 1 M=3, onM=2
m 1 v 2 t 5 I 50602287774.648834000000000 zero 1 1 2 1 M=5, onM=4
m 0 v 2 t 5 I 8253259926.488828700000000 zero 1 1 2 1 M=7, onM=6
m 0 v 2 t 4 I 14910739.498952884000000 zero 1 1 2 1 M=9, onM=7
m 0 v 2 t 2 I 24541784.041385669000000 zero 1 1 2 1 M=11, onM=8
m 0 v 1 t 4 I 2643745.749466502600000 zero 1 1 2 1 M=13, onM=9
m 0 v 1 t 4 I 0.000000000000000 zero 1 1 2 0 M=14, onM=9
For N=10, onM=9, lof_all= 1.#INF00000000000
Alg3
lof_bst= 2643.756863027096600 with J_bst:
```

```
  1  2  3  4  5  6  7  8  9
  1  1  1  1  1  1  1  1  0
linear lof_bst is 2643.761193983883000
quintic lof_bst is 198.886293763002160
quintic lof_bst without penalty is lof*0.010000000000000=1.988862937630022
m 1 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 2 split 1 cov 1 knots   0.249340   0.570040   0.783840 s 1
m 3 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 3 split 2 cov 2 knots   0.004500   1.004300   1.004500 s -1
m 4 split 1 cov 1 knots  -0.216160   0.570040   0.783840 s -1
m 4 split 2 cov 2 knots   1.004000   1.004300   1.004500 s 1
m 5 split 1 cov 2 knots   0.772300   1.004300   1.004500 s -1
m 6 split 1 cov 2 knots   1.004000   1.004300   1.004500 s 1
m 7 split 1 cov 2 knots  -0.023200   0.540300   0.772300 s -1
m 8 split 1 cov 2 knots  -0.791000  -0.586700  -0.280250 s -1
```

MARS Metamodel of Prediction Errors for Vol in Iteration II – Step 4 (with 6 data points and 6 validation points):
The responses should be multiplied by 1k.
qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 12, 12, 50, 3, 1
v 1 count[v] 9 levels
v 2 count[v] 10 levels
T set to p-2 (7).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 11 value 0.187100
v 1 t 2 knot[v][t] 8 value 0.224200
v 1 t 3 knot[v][t] 10 value 0.499600
v 1 t 4 knot[v][t] 6 value 0.500000
v 1 t 5 knot[v][t] 5 value 0.500900
v 1 t 6 knot[v][t] 9 value 0.786200
v 1 t 7 knot[v][t] 12 value 0.809600

v 2 t 1 knot[v][t] 7 value 0.001200
v 2 t 2 knot[v][t] 11 value 0.201700
v 2 t 3 knot[v][t] 6 value 0.204300
v 2 t 4 knot[v][t] 8 value 0.502500
v 2 t 5 knot[v][t] 5 value 0.767800
v 2 t 6 knot[v][t] 12 value 0.815400
v 2 t 7 knot[v][t] 10 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000208
m 0 v 1 t 7 I 621225387227118.620000000000000 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 6 I 563769260568713.000000000000000 zero 1 1 2 1 M=5, onM=3
m 0 v 1 t 2 I 143945428054018.940000000000000 zero 1 1 2 1 M=7, onM=4
m 0 v 2 t 3 I 7161540023987.430700000000000 zero 1 1 2 1 M=9, onM=6
m 0 v 2 t 6 I 2680587765498.110400000000000 zero 1 1 2 1 M=11, onM=7
m 9 v 1 t 5 I 304802422458.856750000000000 zero 1 1 2 1 M=13, onM=9
m 7 v 1 t 5 I 63536933803.304916000000000 zero 1 1 2 1 M=15, onM=11
m 8 v 1 t 3 I 0.000001552017532 zero 1 1 2 1 M=17, onM=11
```

```
For N=12, onM=11, lof_all= 1.#INF00000000000
Alg3
lof_bst= 35267651.525962584000000 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11
  1  1  1  1  1  1  1  1  1  1  0
lof_bst= 19061327.902506381000000 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11
  1  1  1  1  1  1  1  1  1  0  0
linear lof_bst is 19285221.624043379000000
quintic lof_bst is 319236200.270192330000000
quintic lof_bst without penalty is lof*0.027777777777778=8867672.229727564400000

m 1 split 1 cov 1 knots   0.594383   0.617783   0.652883 s -1
m 2 split 1 cov 1 knots   0.594383   0.617783   0.808183 s 1
m 3 split 1 cov 1 knots   0.008983   0.570983   0.594383 s -1
m 4 split 1 cov 1 knots  -0.777217  -0.553017  -0.216717 s -1
m 5 split 1 cov 2 knots  -0.794633  -0.590333  -0.283883 s -1
m 6 split 1 cov 2 knots  -0.794633  -0.590333   0.020767 s 1
m 7 split 1 cov 2 knots   0.020767   0.631867   0.816467 s -1
m 8 split 1 cov 2 knots  -0.183533   0.631867   0.816467 s -1
m 8 split 2 cov 1 knots  -0.500517   0.000383   0.499483 s -1
m 9 split 1 cov 2 knots  -0.183533   0.631867   0.816467 s -1
m 9 split 2 cov 1 knots  -0.500517   0.000383   0.499483 s 1
```

MARS Metamodel of Prediction Errors for Cost in Iteration II – Step 4 (with 6 data points and 6 validation points):

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 12, 12, 50, 3, 1
v 1 count[v] 9 levels
v 2 count[v] 10 levels
T set to p-2 (7).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 11 value 0.187100
v 1 t 2 knot[v][t] 8 value 0.224200
v 1 t 3 knot[v][t] 10 value 0.499600
v 1 t 4 knot[v][t] 6 value 0.500000
v 1 t 5 knot[v][t] 5 value 0.500900
v 1 t 6 knot[v][t] 9 value 0.786200
v 1 t 7 knot[v][t] 12 value 0.809600

v 2 t 1 knot[v][t] 7 value 0.001200
v 2 t 2 knot[v][t] 11 value 0.201700
v 2 t 3 knot[v][t] 6 value 0.204300
v 2 t 4 knot[v][t] 8 value 0.502500
v 2 t 5 knot[v][t] 5 value 0.767800
v 2 t 6 knot[v][t] 12 value 0.815400
v 2 t 7 knot[v][t] 10 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000208
m 0 v 1 t 7 I 6993899343.225333200000000 zero 1 1 2 1 M=3, onM=2
m 0 v 1 t 6 I 10478966729.657942000000000 zero 1 1 2 1 M=5, onM=3
```

```
m 3 v 2 t 2 I 2961799047.788775900000000 zero 1 1 2 1 M=7, onM=5
m 3 v 2 t 3 I 683032866.399371030000000 zero 1 1 2 1 M=9, onM=6
m 0 v 2 t 4 I 98909051.808867946000000 zero 1 1 2 1 M=11, onM=8
m 1 v 2 t 5 I 2757207.572606816400000 zero 1 1 2 1 M=13, onM=10
m 0 v 2 t 5 I 185920.509375761990000 zero 1 1 2 1 M=15, onM=11
m 0 v 2 t 5 I 0.000000000000000 zero 1 1 2 0 M=16, onM=11
For N=12, onM=11, lof_all= 1.#INF00000000000
Alg3
lof_bst= 53.417769237333374 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11
  1  1  1  1  1  1  1  1  1  0  1
linear lof_bst is 428.554112511562890
quintic lof_bst is 2094.049693607996700
quintic lof_bst without penalty is lof*0.006944444444444=14.542011761166643
m 1 split 1 cov 1 knots   0.594383   0.617783   0.652883 s -1
m 2 split 1 cov 1 knots   0.594383   0.617783   0.808183 s 1
m 3 split 1 cov 1 knots  -0.215217   0.570983   0.594383 s -1
m 4 split 1 cov 1 knots  -0.215217   0.570983   0.594383 s -1
m 4 split 2 cov 2 knots  -0.797233  -0.595533  -0.592933 s -1
m 5 split 1 cov 1 knots  -0.215217   0.570983   0.594383 s -1
m 5 split 2 cov 2 knots  -0.599433  -0.595533  -0.592933 s 1
m 6 split 1 cov 1 knots  -0.215217   0.570983   0.594383 s -1
m 6 split 2 cov 2 knots  -0.592933  -0.590333  -0.586433 s -1
m 7 split 1 cov 2 knots  -0.496433   0.006067   0.271367 s -1
m 8 split 1 cov 2 knots  -0.391883   0.006067   0.271367 s 1
m 9 split 1 cov 1 knots   0.594383   0.617783   0.652883 s -1
m 9 split 2 cov 2 knots  -0.026833   0.536667   0.768867 s -1
m 10 split 1 cov 2 knots   0.271367   0.536667   0.768867 s -1
```

Final MARS Metamodel of Responses for Vol (with 8 data points and 6 validation points): The responses should be multiplied by 1k.

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 14, 14, 50, 3, 1
v 1 count[v] 11 levels
v 2 count[v] 12 levels
T set to p-2 (9).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 13 value 0.187100
v 1 t 2 knot[v][t] 7 value 0.202100
v 1 t 3 knot[v][t] 10 value 0.224200
v 1 t 4 knot[v][t] 12 value 0.499600
v 1 t 5 knot[v][t] 6 value 0.500000
v 1 t 6 knot[v][t] 5 value 0.500900
v 1 t 7 knot[v][t] 11 value 0.786200
v 1 t 8 knot[v][t] 8 value 0.799300
v 1 t 9 knot[v][t] 14 value 0.809600

v 2 t 1 knot[v][t] 9 value 0.001200
v 2 t 2 knot[v][t] 8 value 0.199600
v 2 t 3 knot[v][t] 13 value 0.201700
v 2 t 4 knot[v][t] 6 value 0.204300
v 2 t 5 knot[v][t] 10 value 0.502500
v 2 t 6 knot[v][t] 5 value 0.767800
v 2 t 7 knot[v][t] 7 value 0.802100
v 2 t 8 knot[v][t] 14 value 0.815400
```

```
v 2 t 9 knot[v][t] 12 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 6 I 13964451127484424.000000000000000 zero 1 1 2 1 M=3, onM=2
m 2 v 2 t 2 I 3041069350555118.500000000000000 zero 1 1 2 1 M=5, onM=4
m 0 v 2 t 1 I 336570851093858.440000000000000 zero 1 1 2 1 M=7, onM=6
m 6 v 1 t 7 I 53706319880319.062000000000000 zero 1 1 2 1 M=9, onM=8
m 6 v 1 t 3 I 14911004903154.381000000000000 zero 1 1 2 1 M=11, onM=9
m 6 v 1 t 9 I 13637584247.136438000000000 zero 1 1 2 1 M=13, onM=10
m 0 v 1 t 4 I 538625124.816079740000000 zero 1 1 2 1 M=15, onM=11
m 13 v 2 t 5 I 24754069.548083205000000 zero 1 1 2 1 M=17, onM=13
m 0 v 2 t 6 I 0.001997870607497 zero 1 1 2 1 M=19, onM=13
For N=14, onM=13, lof_all= 1.#INF00000000000
Alg3
lof_bst= 15745.356707442419000 with J_bst:
  1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  0  1
lof_bst= 9372.505216306077300 with J_bst:
1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  0  0
linear lof_bst is 26676.326120175963000
quintic lof_bst is 4180792285.069392700000000
quintic lof_bst without penalty is lof*0.020408163265306=85322291.53202842200000
0
m 1 split 1 cov 1 knots  -0.000914   0.000386   0.002336 s -1
m 2 split 1 cov 1 knots  -0.000914   0.000386   0.499486 s 1
m 3 split 1 cov 1 knots  -0.276314   0.000386   0.285686 s 1
m 3 split 2 cov 2 knots  -0.798529  -0.600129  -0.302529 s -1
m 4 split 1 cov 1 knots  -0.276314   0.000386   0.285686 s 1
m 4 split 2 cov 2 knots  -0.798529  -0.600129   0.200271 s 1
m 5 split 1 cov 2 knots  -0.998129  -0.996929  -0.995129 s -1
m 6 split 1 cov 2 knots  -0.998129  -0.996929   0.001871 s 1
m 7 split 1 cov 2 knots  -0.998129  -0.996929  -0.798529 s 1
m 7 split 2 cov 1 knots   0.285686   0.570986   0.594386 s -1
m 8 split 1 cov 2 knots  -0.998129  -0.996929  -0.798529 s 1
m 8 split 2 cov 1 knots   0.535886   0.570986   0.594386 s 1
m 9 split 1 cov 2 knots  -0.998129  -0.996929  -0.798529 s 1
m 9 split 2 cov 1 knots  -0.777214  -0.553014  -0.276314 s -1
m 10 split 1 cov 2 knots  -0.998129  -0.996929  -0.798529 s 1
m 10 split 2 cov 1 knots   0.594386   0.617786   0.652886 s -1
m 11 split 1 cov 1 knots  -0.501814  -0.002214  -0.000914 s -1
```

Final MARS Metamodel of Responses for Cost (with 8 data points and 6 validation points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 14, 14, 50, 3, 1
v 1 count[v] 11 levels
v 2 count[v] 12 levels
T set to p-2 (9).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 13 value 0.187100
v 1 t 2 knot[v][t] 7 value 0.202100
```

```
v 1 t 3 knot[v][t] 10 value 0.224200
v 1 t 4 knot[v][t] 12 value 0.499600
v 1 t 5 knot[v][t] 6 value 0.500000
v 1 t 6 knot[v][t] 5 value 0.500900
v 1 t 7 knot[v][t] 11 value 0.786200
v 1 t 8 knot[v][t] 8 value 0.799300
v 1 t 9 knot[v][t] 14 value 0.809600

v 2 t 1 knot[v][t] 9 value 0.001200
v 2 t 2 knot[v][t] 8 value 0.199600
v 2 t 3 knot[v][t] 13 value 0.201700
v 2 t 4 knot[v][t] 6 value 0.204300
v 2 t 5 knot[v][t] 10 value 0.502500
v 2 t 6 knot[v][t] 5 value 0.767800
v 2 t 7 knot[v][t] 7 value 0.802100
v 2 t 8 knot[v][t] 14 value 0.815400
v 2 t 9 knot[v][t] 12 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 6 I 562685817939.216190000000000 zero 1 1 2 1 M=3, onM=2
m 0 v 2 t 5 I 78881631025.950470000000000 zero 1 1 2 1 M=5, onM=4
m 3 v 1 t 8 I 12798869263.519903000000000 zero 1 1 2 1 M=7, onM=6
m 4 v 1 t 9 I 1677706339.461287700000000 zero 1 1 2 1 M=9, onM=8
m 0 v 1 t 3 I 123697458.871668280000000 zero 1 1 2 1 M=11, onM=9
m 9 v 2 t 3 I 535854.928281072290000 zero 1 1 2 1 M=13, onM=11
m 0 v 2 t 8 I 24580.374692539062000 zero 1 1 2 1 M=15, onM=12
m 0 v 2 t 4 I 83.420381156008816 zero 1 1 2 1 M=17, onM=13
m 0 v 2 t 4 I 0.000000000000000 zero 1 1 2 0 M=18, onM=13
For N=14, onM=13, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.116793045816546 with J_bst:
1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 0.117325671701268
quintic lof_bst is 60.439011743304562
quintic lof_bst without penalty is lof*0.005102040816327=0.308362304812778
m 1 split 1 cov 1 knots  -0.276314   0.000386   0.415436 s -1
m 2 split 1 cov 1 knots  -0.276314   0.000386   0.499486 s 1
m 3 split 1 cov 2 knots  -0.496829   0.005671   0.318571 s -1
m 4 split 1 cov 2 knots  -0.463679   0.005671   0.318571 s 1
m 5 split 1 cov 2 knots  -0.295129   0.005671   0.456871 s -1
m 5 split 2 cov 1 knots   0.022086   0.597186   0.607486 s -1
m 6 split 1 cov 2 knots  -0.295129   0.005671   0.456871 s -1
m 6 split 2 cov 1 knots   0.581736   0.597186   0.607486 s 1
m 7 split 1 cov 2 knots  -0.295129   0.005671   0.503171 s 1
m 7 split 2 cov 1 knots   0.607486   0.617786   0.633236 s -1
m 8 split 1 cov 2 knots  -0.295129   0.005671   0.503171 s 1
m 8 split 2 cov 1 knots   0.607486   0.617786   0.808186 s 1
m 9 split 1 cov 1 knots  -0.777214  -0.553014  -0.276314 s -1
m 10 split 1 cov 1 knots  -0.777214  -0.553014  -0.216714 s -1
m 10 split 2 cov 2 knots  -0.797629  -0.595929  -0.295129 s -1
m 11 split 1 cov 1 knots  -0.777214  -0.553014  -0.216714 s -1
m 11 split 2 cov 2 knots  -0.797629  -0.595929  -0.295129 s 1
m 12 split 1 cov 2 knots   0.318571   0.631471   0.816071 s -1
```

MARS Metamodel of Responses for Vol with Currin's Method (with 14 data points): The responses should be multiplied by 1k.

qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 14, 14, 50, 3, 1
v 1 count[v] 7 levels
v 2 count[v] 12 levels
T set to p-2 (5).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 6 value 0.224200
v 1 t 2 knot[v][t] 8 value 0.499600
v 1 t 3 knot[v][t] 5 value 0.500900
v 1 t 4 knot[v][t] 14 value 0.695300
v 1 t 5 knot[v][t] 7 value 0.786200

v 2 t 1 knot[v][t] 5 value 0.001200
v 2 t 2 knot[v][t] 12 value 0.306700
v 2 t 3 knot[v][t] 13 value 0.502200
v 2 t 4 knot[v][t] 11 value 0.690000
v 2 t 5 knot[v][t] 8 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 4 I 21153270704470560.000000000000000 zero 1 1 2 1 M=3, onM=2
m 2 v 2 t 2 I 2871077576538964.000000000000000 zero 1 1 2 1 M=5, onM=4
m 0 v 2 t 1 I 226329607510409.160000000000000 zero 1 1 2 1 M=7, onM=6
m 6 v 1 t 1 I 93250906819148.141000000000000 zero 1 1 2 1 M=9, onM=8
m 0 v 2 t 2 I 20436006647032.305000000000000 zero 1 1 2 1 M=11, onM=9
m 0 v 1 t 5 I 2510813171017.603500000000000 zero 1 1 2 1 M=13, onM=10
m 1 v 2 t 4 I 0.153360606386285 zero 1 1 2 1 M=15, onM=11
m 0 v 2 t 5 I 0.085556872945454 zero 1 1 2 1 M=17, onM=12
m 0 v 2 t 5 I 0.071798856600192 zero 1 1 2 1 M=19, onM=12
For N=14, onM=12, lof_all= 0.000012327228356
Alg3
linear lof_bst is 864.330273024335720
quintic lof_bst is 615672484.325671430000000
quintic lof_bst without penalty is lof*0.005102040816327=3141186.144518731600000

m 1 split 1 cov 1 knots  -0.262729   0.432571   0.523471 s -1
m 2 split 1 cov 1 knots   0.296221   0.432571   0.523471 s 1
m 3 split 1 cov 1 knots  -0.024479   0.432571   0.737271 s 1
m 3 split 2 cov 2 knots  -0.652614  -0.347114   0.036186 s -1
m 4 split 1 cov 1 knots  -0.024479   0.432571   0.737271 s 1
m 4 split 2 cov 2 knots  -0.652614  -0.347114   0.036186 s 1
m 5 split 1 cov 2 knots  -0.959314  -0.958114  -0.956314 s -1
m 6 split 1 cov 2 knots  -0.959314  -0.958114  -0.652614 s 1
m 7 split 1 cov 2 knots  -0.959314  -0.958114  -0.652614 s 1
m 7 split 2 cov 1 knots  -0.733829  -0.509629  -0.173329 s -1
m 8 split 1 cov 2 knots  -0.959314  -0.958114  -0.652614 s 1
m 8 split 2 cov 1 knots  -0.733829  -0.509629  -0.038529 s 1
m 9 split 1 cov 2 knots  -0.652614  -0.347114   0.111136 s -1
m 10 split 1 cov 1 knots   0.523471   0.614371   0.750721 s -1
m 11 split 1 cov 1 knots  -0.038529   0.432571   0.737271 s -1
m 11 split 2 cov 2 knots   0.036186   0.419486   0.729486 s -1
m 12 split 1 cov 2 knots   0.345986   1.039086   1.039286 s -1
```

MARS Metamodel of Responses for Cost with Currin's Method (with 14 data points): qmars.dat

```
Parameter file is data/marsparm.dat.
X data file is data/x.dat.
Y data file is data/y.dat.
Output file is data/qmars.dat.
circle,n,p,T,N,Mmax,maxIA,alg3
0, 2, 0, 14, 14, 50, 3, 1
v 1 count[v] 7 levels
v 2 count[v] 12 levels
T set to p-2 (5).
Warning: Knots distributed asymmetrically over levels of covariate 2.
Knots based on scaled/actual x-values:
v 1 t 1 knot[v][t] 6 value 0.224200
v 1 t 2 knot[v][t] 8 value 0.499600
v 1 t 3 knot[v][t] 5 value 0.500900
v 1 t 4 knot[v][t] 14 value 0.695300
v 1 t 5 knot[v][t] 7 value 0.786200

v 2 t 1 knot[v][t] 5 value 0.001200
v 2 t 2 knot[v][t] 12 value 0.306700
v 2 t 3 knot[v][t] 13 value 0.502200
v 2 t 4 knot[v][t] 11 value 0.690000
v 2 t 5 knot[v][t] 8 value 0.999800

Min/Max x-values:
v 1 min 0.000000 max 1.000000
v 2 min 0.000000 max 1.000000
mars.qls
EPS2 0.0000000002500 eps3 0.0000000000179
m 0 v 1 t 4 I 858846602267.893920000000000 zero 1 1 2 1 M=3, onM=2
m 0 v 2 t 2 I 64489801799.024834000000000 zero 1 1 2 1 M=5, onM=4
m 1 v 2 t 5 I 13010850810.562563000000000 zero 1 1 2 1 M=7, onM=6
m 4 v 1 t 1 I 1859203636.146642200000000 zero 1 1 2 1 M=9, onM=8
m 3 v 1 t 3 I 66559694.730898231000000 zero 1 1 2 1 M=11, onM=10
m 0 v 1 t 5 I 116668403.846223890000000 zero 1 1 2 1 M=13, onM=11
m 0 v 2 t 3 I 0.839298763277442 zero 1 1 2 1 M=15, onM=12
m 0 v 2 t 5 I 0.032245321227659 zero 1 1 2 1 M=17, onM=13
m 0 v 2 t 5 I 0.000000036194794 zero 1 1 2 1 M=19, onM=13
For N=14, onM=13, lof_all= 1.#INF00000000000
Alg3
lof_bst= 0.000045143582274 with J_bst:
1  2  3  4  5  6  7  8  9 10 11 12 13
  1  1  1  1  1  1  1  1  1  1  1  1  0
linear lof_bst is 1.296527187277450
quintic lof_bst is 651.881924728576340
quintic lof_bst without penalty is lof*0.005102040816327=3.325928187390695
m 1 split 1 cov 1 knots  -0.262729   0.432571   0.523471 s -1
m 2 split 1 cov 1 knots   0.296221   0.432571   0.523471 s 1
m 3 split 1 cov 2 knots  -0.653814  -0.347114  -0.151614 s -1
m 4 split 1 cov 2 knots  -0.640364  -0.347114  -0.151614 s 1
m 5 split 1 cov 1 knots   0.238171   0.432571   0.724171 s -1
m 5 split 2 cov 2 knots   0.345986   1.039086   1.039286 s -1
m 6 split 1 cov 1 knots   0.238171   0.432571   0.724171 s -1
m 6 split 2 cov 2 knots   1.038786   1.039086   1.039286 s 1
m 7 split 1 cov 2 knots  -0.653814  -0.347114   0.345986 s 1
m 7 split 2 cov 1 knots  -0.733829  -0.509629  -0.232929 s -1
m 8 split 1 cov 2 knots  -0.653814  -0.347114   0.345986 s 1
m 8 split 2 cov 1 knots  -0.733829  -0.509629  -0.232929 s 1
m 9 split 1 cov 2 knots  -0.653814  -0.347114   0.112936 s -1
m 9 split 2 cov 1 knots  -0.232929   0.043771   0.238171 s -1
m 10 split 1 cov 2 knots  -0.653814  -0.347114   0.112936 s -1
m 10 split 2 cov 1 knots  -0.232929   0.043771   0.238171 s 1
```

```
m 11 split 1 cov 1 knots   0.523471   0.614371   0.750721 s -1
m 12 split 1 cov 2 knots  -0.151614   0.043886   0.337136 s -1
```

B.3.2 Response Surface Metamodels Developed in Section 5.5.2

Regression Analysis for Vol versus R, L, and T:
The regression equation is
Vol = 488518 + 467050 R + 183783 L + 0 T

Predictor	Coef	Stdev	t-ratio	p
Constant	488518	84823	5.76	0.005
R	467050	84823	5.51	0.005
L	183783	84823	2.17	0.096
T	0	84823	0.00	1.000

s = 239916 R-sq = 89.7% R-sq(adj) = 82.1%

Analysis of Variance

SOURCE	DF	SS	MS	F	p
Regression	3	2.01530E+12	6.71766E+11	11.67	0.019
Error	4	2.30238E+11	57559535616		
Total	7	2.24553E+12			

SOURCE	DF	SEQ SS
R	1	1.74509E+12
L	1	2.70210E+11
T	1	0

Regression Analysis for Cost versus R, L, and T:
The regression equation is
Cost = 4108 + 3215 R + 983 L + 231 T

Predictor	Coef	Stdev	t-ratio	p
Constant	4107.8	299.7	13.71	0.000
R	3215.2	299.7	10.73	0.000
L	982.7	299.7	3.28	0.031
T	231.2	299.7	0.77	0.484

s = 847.7 R-sq = 96.9% R-sq(adj) = 94.6%

Analysis of Variance

SOURCE	DF	SS	MS	F	p
Regression	3	90851440	30283814	42.14	0.002
Error	4	2874456	718614		
Total	7	93725896			

SOURCE	DF	SEQ SS
R	1	82698160
L	1	7725829
T	1	427452

Regression Analysis for Cost versus R and L:

The regression equation is

Cost = 4108 + 3215 R + 983 L

Predictor	Coef	Stdev	t-ratio	p
Constant	4107.8	287.3	14.30	0.000
R	3215.2	287.3	11.19	0.000
L	982.7	287.3	3.42	0.019

s = 812.6 R-sq = 96.5% R-sq(adj) = 95.1%

Analysis of Variance

SOURCE	DF	SS	MS	F	p
Regression	2	90423984	45211992	68.46	0.000
Error	5	3301908	660382		
Total	7	93725888			

SOURCE	DF	SEQ SS
R	1	82698160
L	1	7725829

B.3.3 FORTRAN Programs Used in SEED in Section 5.5

The FORTRAN program of altcov.f and altcov.params.h used in SEED in Section 5.5 are enclosed in this section. The programs of altcov.f and altcov.params.h are used to

adjust entries of the covariance matrix. Other programs used in the integrated process in SEED are the same as those presented in Appendix A.

Altcov.f:

```
************************************************************************
*
      program altcov
*
*  This program calculates the alternated correlation matrix, given the
*       initial correlation matrix and predicted prediction errors at
*       possible new data points.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
*
************************************************************************
*
* Input files:
* ------------
*  altcov.params.h - parameter file, specifying numdv, numsamp,
*                       errmax, lambda, fprefix, fprefix2, fprefixnew
*  fprefix.cov     - initial correlation matrix
*  fprefix2.out    - predicted prediction errors at possible new data points
*
* Output files:
* -------------
*  fprefixnew.cov  - alternated correlation matrix
*
* Variables:
* ----------
*  inicov     = the initial correlation matrix
*  newcov     = the alternated correlation matrix
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*  numsamp = number of data samples from which the correlation matrix
*              is calculated
*
* Local Variables:
* ----------------
*  DOUBLE PRECISION
*  ----------------
*  errpred = the predicted prediction errors associated with each data
*            and possible new data points
*
************************************************************************

      integer numsamp,numdv,numold
      double precision lambda,errmax1,errmax2
      character*20 fprefix,fprefix2,fprefix3,fprefixnew
C
C  include parameter settings for numdv,numsamp,fprefix,fprefix2,fprefixnew,
C  errmax, lambda, e.g., in the one-variable problem, for the first step:
C  numdv=1,numsamp=8,fprefix='step1newp',fprefix2='errpred1',
C  fprefixnew='step1altnewp',errmax=0.50,lambda=2.0
```

```
C
      include 'altcov.params.h'

      double precision inicov(numsamp,numsamp),newcov(numsamp,numsamp),
     &       errpred1(numsamp),errpred2(numsamp)
      integer i,j,lenstr
      character*16 ftitle
      character*20 deckfile,deckfile2,deckfile3,outfile

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefix2,lenstr)
       ftitle=fprefix2
       deckfile2=ftitle(1:lenstr) // '.out'

       call getlen(fprefix3,lenstr)
       ftitle=fprefix3
       deckfile3=ftitle(1:lenstr) // '.out'

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.cov'

       open(21,file=deckfile,status='old')
       open(23,file=deckfile2,status='old')
       open(24,file=deckfile3,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,deckfile2,deckfile3,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (inicov(i,j),j=1,numsamp)
      close(21)

C
C  initialize errpred
C

      print *
      write(6,*) 'Reading in and calculating errpred 1...'
      do 25 i=1,numsamp
         if (i.le.numold) then
            errpred1(i)=0.0
         else
            read(24,*) errpred1(i)
         endif
         if (abs(errpred1(i)).gt.(errmax1)) then
            errpred1(i)=errmax1
         endif
 25   continue
      close(24)
```

```
      print *
      write(6,*) 'Reading in and calculating errpred 2...'
      do 20 i=1,numsamp
         if (i.le.numold) then
            errpred2(i)=0.0
         else
            read(23,*) errpred2(i)
         endif
         if (abs(errpred2(i)).gt.(errmax2)) then
            errpred2(i)=errmax2
         endif
 20   continue
      close(23)


C
C  calculate the alternated correlation matrix
C
      do 30 i=1,numsamp
         do 40 j=i,numsamp
         if (i.eq.j) then
            newcov(i,j)=1.0
         elseif (((i.gt.numold).AND.(j.le.numold)).OR.
     &             ((i.le.numold).AND.(j.gt.numold))) then
         newcov(i,j)=inicov(i,j)*(1-1/lambda*(0.5*abs
     &   (errpred1(i)/errmax1)+0.5*abs(errpred2(i)/errmax2)))
     &   *(1-1/lambda*(0.5*abs(errpred1(j)/errmax1)+
     &   0.5*abs(errpred2(j)/errmax2)))
         newcov(j,i)=newcov(i,j)
         else
         newcov(i,j)=inicov(i,j)
         newcov(j,i)=newcov(i,j)
         endif
 40   continue
 30   continue

C
C  write alternated correlation matrix into specified .cov file
C
      do 50 i=1,numsamp
        write(27,79) (newcov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 50   continue
      close(27)

      print *
      write(6,*) 'Alternated correlation matrix written to .cov file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
```

```
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end
```

Altcov.params.h

```
C*********************************************************
C                                                        *
C  Parameter input file for 'altcov'                     *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=2,numsamp=14,numold=12,
     &          fprefix='ch5pvit2newp',fprefix2='double1.gau',
     &          fprefix3='errpred2_2.gau',
     &          fprefixnew='ch5pvit2altnewp',errmax1=342400,
     &          errmax2=1310,
     &          lambda=2.0 )

C
C  numdv = # design variables
C  numsamp = # samples in data set
C  numold = # old data points in the data set
C
C  fprefix = prefix of titles of file that stores the initial
C            correlation matrix for both old and possible new
C            data points
C
C  fprefix2 = prefix of titles of file that stores the
C             predicted prediction errors at possible new
C             data points
C
C  fprefixnew = prefix of titles of file that stores the
C               alternated correlation matrix for both old and
C               possible new data points, with prediction errors
C               at these points considered
C
```

```
C  errmax = maximum value of the absolute predicted prediction error
C
C  lambda = coefficient used to gauge the adjustment to initial
C           correlation matrix
C***********************************************************************
```

B.3.4 Implementation of SEED in iSIGHT in Section 5.5

Figures presented in this section illustrate how the SEED method is implemented in iSIGHT. The organization of tasks in Iteration II – Step 7 is shown in Figure B.1.

In Iteration I – Step 5, with information from metamodels of prediction errors, we use five simulation codes in iSIGHT, i.e., Covmat, KrigErrpred, MARSErrpred, Altcov, and Detcov. Covmat is used to formulate the covariance matrix, KrigErrpred and MARSErrpred are metamodels to predict prediction errors, Altcov is used to adjust entries of the covariance matrix, and Detcov is used to calculate the determinant.

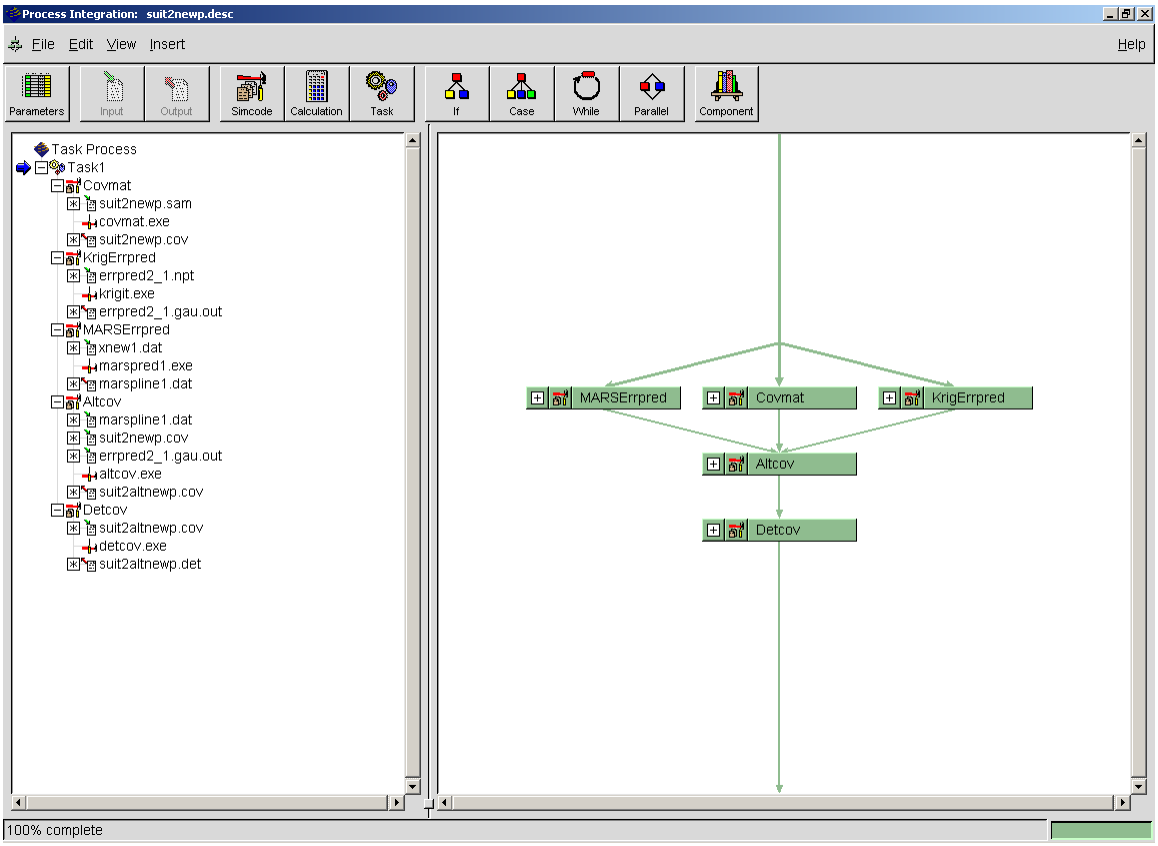


Figure B.1 Implementation of E-RCEM in iSIGHT – Iteration II, Step 7

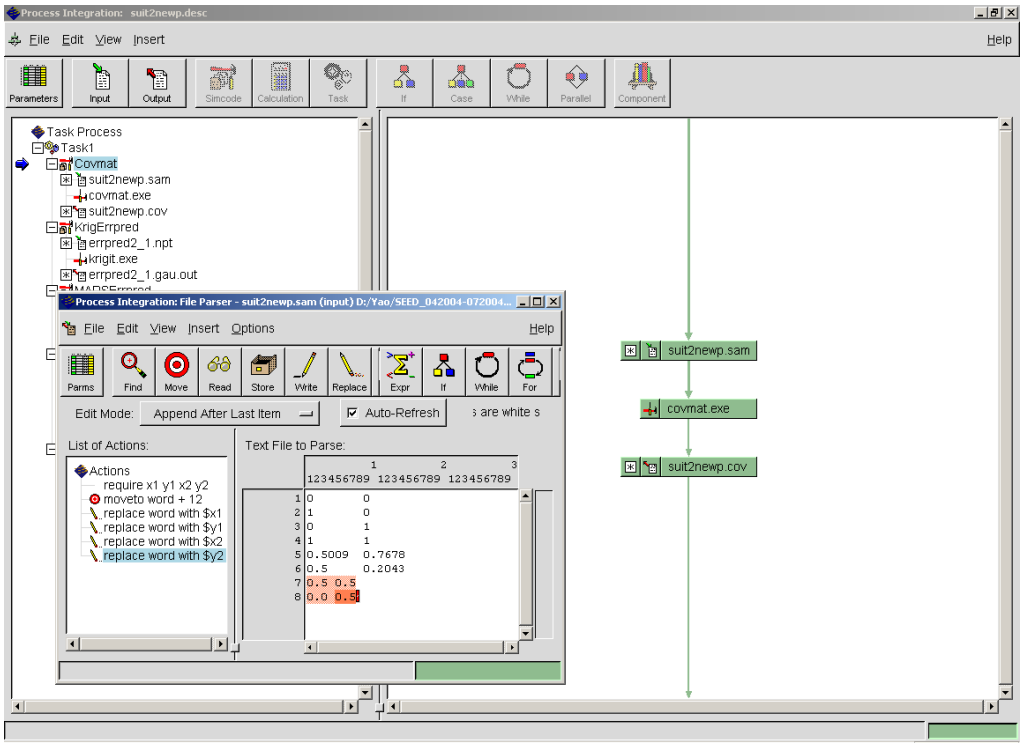


Figure B.2 Input Mapping for Covmat.f in SEED – Iteration II, Step 7

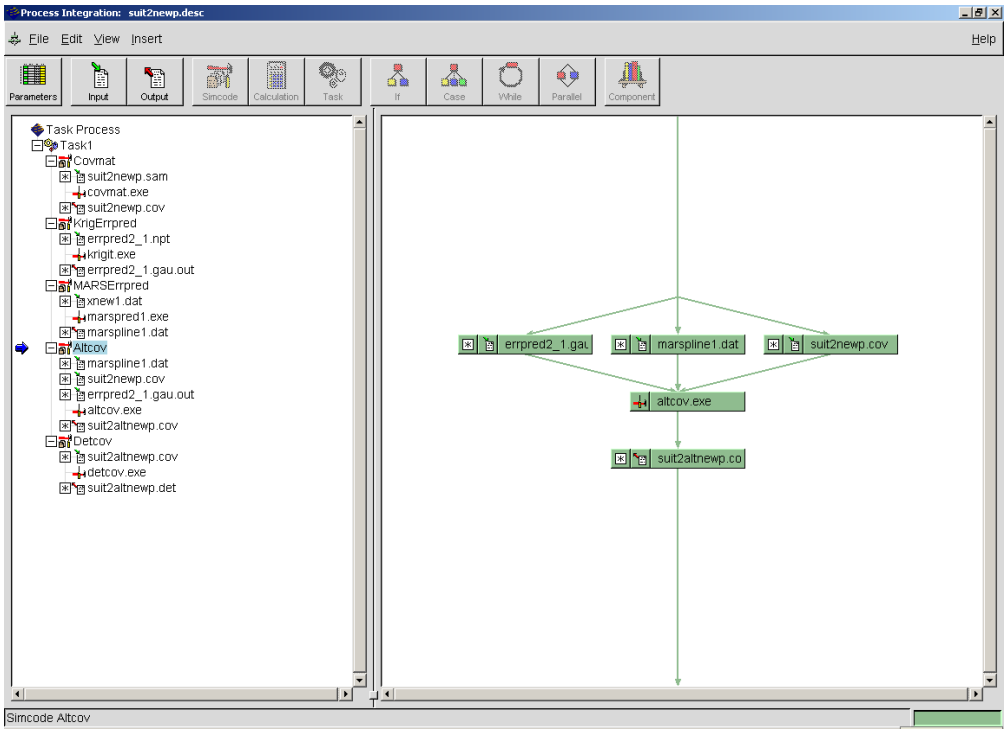


Figure B.3 Organization of Input and Output for Altcov.f in SEED – Iteration II, Step 7

APPENDIX C

SUPPORTING MATERIALS FOR THE INTEGRATED PROCESSES OF METAMODELING AND DESIGN SPACE EXPLORATION IN E-RCEM

This appendix is intended to supplement the development of the E-RCEM method in Chapter 6. The computer codes to incorporate design goals and constraints in the metamodeling process are presented in Section C.1. The organization of the E-RCEM method in iSIGHT with a single-variable example in Section 6.5 is illustrated in Section C.2.

C.1 FORTRAN PROGRAMS TO INCORPORATE DESIGN GOALS IN METAMODELING

The FORTRAN programs to incorporate design goals in metamodeling in Sections 6.3 and 6.5 are listed in this section. To formulate the covariance matrix we use covmat.f and covdata.params.h; the input and output filenames are specified in covdata.params.h. To adjust entries of the covariance matrix we use altcov.f and altcov.params.h. To calculate the determinant of the covariance matrix we use detcov.f and detcov.params.h.

Covmat.f:

```
************************************************************************
*
      program covmat
*
*  This program invokes calculation of the correlation matrix given
*        information of points and values of theta.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Yao Lin 26 March 2003 / Tim Simpson, 25 Feburary 1998
*
************************************************************************
*
* Input files:
* ------------
*  covdata.params.h - parameter file, specifying numdv, numsamp, fprefix
*  .sam             - x's of sample points
*  .gau.fit         - thetas
*
* Output files:
* -------------
*  .cov             - correlation matrix
*
* Variables:
* ----------
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*   numsamp = number of data samples from which the correlation matrix
*               is calculated
*
* Local Variables:
* ----------------
*   DOUBLE PRECISION
```

```
*   ----------------
*   xmat     = numdv x numsamp of sample site locations, scaled [0,1]
*
*   INTEGER
*   -------
*
************************************************************************

      integer numdv,numsamp
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix, e.g., in the
C  one-variable problem: numdv=1,numsamp=5,fprefix='step1'
C
      include 'covdata.params.h'

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   dummy2,thetaray(1,numdv),theta(numdv)
      integer i,j,dummy,lenstr
      character*16 ftitle
      character*20 deckfile,fitsfile,outfile


C
C  open necessary .sam, .fit, and .cov files based on 'fprefix' name,
C  e.g., in the one-variable problem:
C         step1.sam, step1.gau.fit, step1.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.sam'
       fitsfile=ftitle(1:lenstr) // '.gau.fit'
       outfile=ftitle(1:lenstr) // '.cov'

       open(21,file=deckfile,status='old')
       open(22,file=fitsfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,fitsfile,outfile
       print *, numdv,numsamp
C
C  initialize xmat and theta arrays
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (xmat(i,j),j=1,numdv)
      close(21)

      print *
      write(6,*) 'Reading in theta parameters...'
      do 20 i=1,1
        read(22,*) dummy,(thetaray(i,j),j=1,numdv),dummy2
        write(6,1000) dummy,(thetaray(i,j),j=1,numdv)
 1000   format(i2,8f9.5)
 20   continue
      close(22)

      do 50 j=1,numdv
          theta(j)=thetaray(1,j)
 50     continue
        write(6,1002) (theta(j),j=1,numdv)
```

```
 1002     format(8f9.5)


C
C  call subroutine to calculate the correlation matrix
C
C  input:  xmat, theta, numsamp, numdv
C
C  output: R - the correlation matrix
C

         call cormat (xmat,cov,numsamp,numdv,theta)

C
C  write predicted values to specified .cov file
C
      do 90 i=1,numsamp
        write(27,79) (cov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 90   continue
      close(27)

      print *
      write(6,*) 'Correlation matrix written to specified .cov file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'covdata.params.h'.
*
*  With this known, the .sam, .gau.fit, and .cov suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Yao Lin, 3/26/2003; Tim Simpson, 2/15/1998
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end
```

```
************************************************************************
*
      subroutine cormat (xmat,cov,numsamp,numdv,theta)
*
*
*  This subroutine calculates the correlation matrix and its inverse
*
*  Original code developed by:
*  Yao Lin 26 March 2003 /
*        Tim Simpson 15 February 1998 / Tony Giunta, 12 May 1997
*
************************************************************************
*
* Inputs:
* -------
*   DOUBLE PRECISION:
*   -----------------
*   xmat,theta
*
*   INTEGER:
*   --------
*   numdv,numsamp
*
* Outputs:
* --------
*   DOUBLE PRECISION:
*   -----------------
*   cov - the correlation matrix.
*
*
************************************************************************
C
C  passed variables
C
      integer numdv,numsamp

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   theta(numdv),R
C
C  local variables
C
      integer i,j
C
C  calculate terms in the correlation matrix
C
      do 300 i = 1,numsamp
        do 305 j = i,numsamp
          if( i .eq. j ) then
            cov(i,j) = 1.0d0
          else
C
C  call subroutine to compute spatial correlation function for xmat
C
C  input:  xmat, theta, numdv, numsamp, i, j
C
C  output: R
C
            call scfxmat(R,xmat,theta,numdv,numsamp,i,j)
            cov(i,j) = R
            cov(j,i) = cov(i,j)
          endif
 305    continue
 300  continue
      end
```

```
C*****************************************************************
C
      subroutine scfxmat(R,xmat,theta,numdv,numsamp,i,j)
C
C     Origin: Tim Simpson       Date:  February 11, 1998
C     Modified: Yao Lin         Date:  March 26, 2003
C
C     subroutine to compute spatial correlation function (scf) for
C     correlation matrix; NOT to compute scf for r_xhat.
C
C  Output:
C  -------
C    R = value of correlation function between two sample points,
C          given theta
C
C  Input:
C  ------
C    xmat = matrix of sample points
C    theta = array of theta values
C    i,j = i_th and j_th elements of correlation matrix for which
C           correlation function is being computed
C
C  All variables except R are unchanged upon exiting
C
C*****************************************************************
C
C  passed variables
C
      integer i,j,numdv,numsamp
      double precision R,xmat(numsamp,numdv),theta(numdv)
C
C  local variables
C
      double precision sum,thetadist,dist
      integer k

      sum=0.0d0
      do 120 k = 1,numdv
          dist = ABS(xmat(i,k)-xmat(j,k))
          sum = sum + theta(k)*((dist)**2)
  120      continue
         R = exp( -1.0d0*sum )

        return
        end
```

Covdata.params.h:

```
C*********************************************************
C                                                        *
C  Parameter input file for 'covmat'                     *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for calculating the covariance
C          matrix and its determinant
C

      parameter (numdv=1,numsamp=11,fprefix='suit3valid')
```

```
C
C  numdv = # design variables
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C*******************************************************
```

Altcov.f:

```
************************************************************************
*
      program altcov
*
*  This program calculates the alternated correlation matrix, given the
*       initial correlation matrix and predicted prediction errors at
*       possible new data points.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
*
************************************************************************
*
* Input files:
* ------------
*  altcov.params.h - parameter file, specifying numdv, numsamp,
*                       errmax, lambda, fprefix, fprefix2, fprefixnew
*  fprefix.cov     - initial correlation matrix
*  fprefix2.out    - predicted prediction errors at possible new data points
*
* Output files:
* -------------
*  fprefixnew.cov  - alternated correlation matrix
*
* Variables:
* ----------
*  inicov     = the initial correlation matrix
*  newcov     = the alternated correlation matrix
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*  numsamp = number of data samples from which the correlation matrix
*              is calculated
*
* Local Variables:
* ----------------
*  DOUBLE PRECISION
*  ----------------
*  errpred = the predicted prediction errors associated with each data
*            and possible new data points
*
************************************************************************

      integer numsamp
      double precision lambda,errmax,gamma,TargetH,TargetL,TargetS
      double precision y1max,y1min,yconstant
      character TargetType
      character*16 fprefix,fprefix2,fprefixnew,fprefix3
C
```

```
C  include parameter settings for numdv,numsamp,fprefix,fprefix2,fprefixnew,
C  errmax, lambda, e.g., in the one-variable problem, for the first step:
C  numdv=1,numsamp=8,fprefix='step1newp',fprefix2='errpred1',
C  fprefixnew='step1altnewp',errmax=0.50,lambda=2.0
C
      include 'altcov.params.h'

      double precision inicov(numsamp,numsamp),newcov(numsamp,numsamp),
     &       errpred(numsamp),goalachieve(numsamp),responsey1(numsamp),
     &       response,goalachievement
      integer i,j,lenstr
      character*16 ftitle
      character*20 deckfile,deckfile2,deckfile3,outfile

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefix2,lenstr)
       ftitle=fprefix2
       deckfile2=ftitle(1:lenstr) // '.dat'

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefix3,lenstr)
       ftitle=fprefix3
       deckfile3=ftitle(1:lenstr) // '.dat'


       open(21,file=deckfile,status='old')
       open(23,file=deckfile2,status='old')
       open(25,file=deckfile3,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,deckfile2,deckfile3,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (inicov(i,j),j=1,numsamp)
      close(21)

C
C  initialize errpred
C
      print *
      write(6,*) 'Reading in and calculating errpred...'
      do 20 i=1,numsamp
         if (i.le.numold) then
            errpred(i)=0.0
         else
         read(23,*) errpred(i)
         endif
         if (abs(errpred(i)).gt.(errmax)) then
```

```
            errpred(i)=errmax
         endif
 20    continue
       close(23)

       print *
       write(6,*)
      &  'Reading in responses and calculating goal.achievement...'
       do 60 i=1,numsamp
          read(25,*) responsey1(i)
          response=responsey1(i)+yconstant
          if (TargetType.eq.'H') then
             call Hgoalachievecal(goalachievement,TargetH,
      &      response,y1max,y1min,gamma)
             goalachieve(i)=goalachievement
          else if (TargetType.eq.'L') then
             call Lgoalachievecal(goalachievement,TargetL,
      &      response,y1max,y1min,gamma)
             goalachieve(i)=goalachievement
          else if (TargetType.eq.'S') then
             call Sgoalachievecal(goalachievement,TargetS,
      &      response,y1max,y1min,gamma)
             goalachieve(i)=goalachievement
          endif
 60    continue
       close(25)

C
C  calculate the alternated correlation matrix
C
       do 30 i=1,numsamp
          do 40 j=i,numsamp
          if (i.eq.j) then
             newcov(i,j)=1.0
          elseif (((i.le.numold).AND.(j.le.numold)).OR.
      &             ((i.gt.numold).AND.(j.gt.numold))) then
          newcov(i,j)=inicov(i,j)
          newcov(j,i)=newcov(i,j)
          elseif (((i.le.numold).AND.(j.gt.numold)).OR.
      &             ((i.gt.numold).AND.(j.le.numold))) then
          if (inicov(i,j).eq.1) then
          newcov(i,j)=inicov(i,j)
          newcov(j,i)=newcov(i,j)
          elseif (inicov(i,j).lt.1) then
          newcov(i,j)=inicov(i,j)
      &   *(1-abs(errpred(i)/lambda/errmax))
      &   *(1-goalachieve(i))
      &   *(1-abs(errpred(j)/errmax/lambda))
      &   *(1-goalachieve(j))
          newcov(j,i)=newcov(i,j)
          endif
          endif
 40    continue
 30    continue

C
C  write alternated correlation matrix into specified .cov file
C
       do 50 i=1,numsamp
         write(27,79) (newcov(i,j),j=1,numsamp)
 79      format(10(f13.5,1x))
 50    continue
       close(27)
```

```
      print *
      write(6,*) 'Alternated correlation matrix written to .cov file'

      stop
      end

*****************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

*****************************************************************
*
      subroutine Hgoalachievecal(goalachievement,TargetH,
     &      response,y1max,y1min,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      double precision goalachievement,TargetH,response
```

```
      double precision y1max,y1min,gamma

      if (response.le.y1min) then
         goalachievement=0.00000000
      else if (response.ge.min(TargetH,y1max)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(response-y1min)/
     &    (min(TargetH,y1max)-y1min)/gamma
      endif

      return
      end


*************************************************************************
*
      subroutine Lgoalachievecal(goalachievement,TargetL,
     &      response,y1max,y1min,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*************************************************************************
*
      double precision goalachievement,TargetL,response
      double precision y1max,y1min,gamma

      if (response.ge.y1max) then
         goalachievement=0.0000000000
      else if (response.le.max(TargetL,y1min)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(y1max-response)/
     &    (y1max-max(y1min,TargetL))/gamma
      endif

      return
      end

*************************************************************************
*
      subroutine Sgoalachievecal(goalachievement,TargetS,
     &      response,y1max,y1min,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
```

```
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      double precision goalachievement,TargetS,response
      double precision y1max,y1min,gamma

      if (response.ge.y1max) then
         goalachievement=0.00000000
      else if (response.le.y1min) then
         goalachievement=0.00000000
      else if (response.eq.TargetS) then
         goalachievement=1.0/gamma
      else if (response<TargetS.AND.response>y1min) then
          goalachievement=(response-y1min)/(TargetS-y1min)/gamma
      else if (response>TargetS.AND.response<y1max) then
          goalachievement=(response-TargetS)/(y1max-TargetS)/gamma
      endif

      return
      end
```

Altcov.params.h:

```
C*********************************************************
C                                                        *
C  Parameter input file for 'altcov'                     *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=1,numsamp=11,numold=10,
     &          fprefix='suit3valid',fprefix2='marspline1',
     &          fprefixnew='suit3altvalid',
     &          fprefix3='marspline',
     &          errmax=1.1,lambda=2.0,
     &          y1max=0.0,y1min=-1.45,TargetL=-1.6,
     &          TargetH=-1.0,TargetS=-1.0,
     &          TargetType='L',
     &          yconstant=0.0,
     &          gamma=1.25)

C
C  numdv = # design variables
C  numsamp = # samples in data set
C  numold = # old data points in the data set
C
C  fprefix = prefix of titles of file that stores the initial
C            correlation matrix for both old and possible new
C            data points
C
C  fprefix2 = prefix of titles of file that stores the
C             predicted prediction errors at possible new
C             data points
C
C  fprefix3 = prefix of titles of file that stores the
```

```
C             predicted response values at all points
C
C
C  fprefixnew = prefix of titles of file that stores the
C               alternated correlation matrix for both old and
C               possible new data points, with prediction errors
C               at these points considered
C
C  errmax = maximum value of the absolute predicted prediction error
C
C  lambda = coefficient used to gauge the adjustment to initial
C           correlation matrix
C
C*********************************************************************
```

Detcov.f:

```
C*********************************************************************
C
      program detcov
C
C  This program calculates the determinant given a matrix.  Particularly,
C       in SEED, it is used to calculate the determinant of the
C       correlation matrix.
C
C  Updated by: Yao Lin, March 26, 2003
C
C  Original code developed by:
C  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
C
C*********************************************************************
C
C Input files:
C ------------
C  detcov.params.h - parameter file, specifying numdv, numsamp,
C                       coedet, fprefix
C  .cov             - correlation matrix
C
C Output files:
C -------------
C  .det             - determinant of the correlation matrix
C
C Variables:
C ----------
C   cov     = the input correlation matrix for which we calculate
C             determinant
C
C Parameter Variables (to be specified by user in dace.params.h):
C ---------------------------------------------------
C   numsamp = number of data samples from which the correlation matrix
C               is calculated
C
C Local Variables:
C ----------------
C   DOUBLE PRECISION
C   ----------------
C   work     = vector of length 'numsamp' used as temporary storage
C   invmat   = inverse of the correlation matrix (numsamp x numsamp)
C
```

```
C   INTEGER
C   -------
C   ipvt    = vector of length 'numsamp' of pivot locations
C
C**********************************************************************

      integer numsamp
      double precision coedet
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix
C
      include 'detcov.params.h'

C**********************************************************************
C
C  include LINPACK routines used to find determinant of correlation matrix
C
C**********************************************************************

C      include 'dgefa.f'
C      include 'dgedi.f'

C**********************************************************************

      double precision cov(numsamp,numsamp),work(numsamp),
     &       dummy2,detR,det(2),rcond,z(numsamp)
      integer i,j,ipvt(numsamp),dummy,lenstr,info
      character*16 ftitle
      character*20 deckfile,outfile
      err=0.0000
C
C  open necessary .cov and .det files based on 'fprefix' name,
C  e.g., step1.cov, step1.det
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.cov'
       outfile=ftitle(1:lenstr) // '.det'

       open(21,file=deckfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,outfile
       print *, numsamp
C
C  initialize cov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (cov(i,j),j=1,numsamp)
      close(21)

C
C     Start to calculate the determinant of the correlation matrix;
C        initialization.
C
      do 307 i=1,numsamp
        work(i)=0.0d0
        ipvt(i)=0
 307  continue
```

```
C
C     If there is any error in the calculation in DGEFA (singular matrix),
C        this program will set the determinant to 0.
C
      call dgeco(cov,numsamp,numsamp,ipvt,rcond,z)
      if( rcond .eq. 0 ) then
          write(27,78) err
 78   format(10(f13.5,1x))
          close(27)
          go to 1000
      endif
C
C In DGEDI, last flag is: 1 (inverse only), 10 (Det only), 11 (both)
C
      call dgedi(cov, numsamp, numsamp, ipvt, det, work, 10)
      detR=det(1)*10.0d0**det(2)
      detR=coedet*detR

C
C  write predicted values to specified .det file
C
      write(27,79) detR
 79   format(10(f13.5,1x))
      close(27)

      print *
      write(6,*) detR
1000  write(6,*) 'Coefficient*Determinant written to .det file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
```

```
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

      subroutine dgeco(a,lda,n,ipvt,rcond,z)
      integer lda,n,ipvt(1)
      double precision a(lda,1),z(1)
      double precision rcond
c
c     dgeco factors a double precision matrix by gaussian elimination
c     and estimates the condition of the matrix.
c
c     if  rcond  is not needed, dgefa is slightly faster.
c     to solve  a*x = b , follow dgeco by dgesl.
c     to compute  inverse(a)*c , follow dgeco by dgesl.
c     to compute  determinant(a) , follow dgeco by dgedi.
c     to compute  inverse(a) , follow dgeco by dgedi.
c
c     on entry
c
c        a       double precision(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        rcond   double precision
c                an estimate of the reciprocal condition of  a .
c                for the system  a*x = b , relative perturbations
c                in  a  and  b  of size  epsilon  may cause
c                relative perturbations in  x  of size  epsilon/rcond .
c                if  rcond  is so small that the logical expression
c                           1.0 + rcond .eq. 1.0
c                is true, then  a  may be singular to working
c                precision.  in particular,  rcond  is zero  if
c                exact singularity is detected or the estimate
c                underflows.
c
c        z       double precision(n)
c                a work vector whose contents are usually unimportant.
c                if  a  is close to a singular matrix, then  z  is
c                an approximate null vector in the sense that
c                norm(a*z) = rcond*norm(a)*norm(z) .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
```

```
c
c     linpack dgefa
c     blas daxpy,ddot,dscal,dasum
c     fortran dabs,dmax1,dsign
c
c     internal variables
c
      double precision ddot,ek,t,wk,wkm
      double precision anorm,s,dasum,sm,ynorm
      integer info,j,k,kb,kp1,l
c
c
c     compute 1-norm of a
c
      anorm = 0.0d0
      do 10 j = 1, n
         anorm = dmax1(anorm,dasum(n,a(1,j),1))
   10 continue
c
c     factor
c
      call dgefa(a,lda,n,ipvt,info)
c
c     rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
c     estimate = norm(z)/norm(y) where  a*z = y  and  trans(a)*y = e .
c     trans(a)  is the transpose of a .  the components of  e  are
c     chosen to cause maximum local growth in the elements of w  where
c     trans(u)*w = e .  the vectors are frequently rescaled to avoid
c     overflow.
c
c     solve trans(u)*w = e
c
      ek = 1.0d0
      do 20 j = 1, n
         z(j) = 0.0d0
   20 continue
      do 100 k = 1, n
         if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
         if (dabs(ek-z(k)) .le. dabs(a(k,k))) go to 30
            s = dabs(a(k,k))/dabs(ek-z(k))
            call dscal(n,s,z,1)
            ek = s*ek
   30    continue
         wk = ek - z(k)
         wkm = -ek - z(k)
         s = dabs(wk)
         sm = dabs(wkm)
         if (a(k,k) .eq. 0.0d0) go to 40
            wk = wk/a(k,k)
            wkm = wkm/a(k,k)
         go to 50
   40    continue
            wk = 1.0d0
            wkm = 1.0d0
   50    continue
         kp1 = k + 1
         if (kp1 .gt. n) go to 90
            do 60 j = kp1, n
               sm = sm + dabs(z(j)+wkm*a(k,j))
               z(j) = z(j) + wk*a(k,j)
               s = s + dabs(z(j))
   60       continue
            if (s .ge. sm) go to 80
               t = wkm - wk
```

```
               wk = wkm
               do 70 j = kp1, n
                  z(j) = z(j) + t*a(k,j)
   70          continue
   80       continue
   90    continue
         z(k) = wk
  100 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
c
c     solve trans(l)*y = w
c
      do 120 kb = 1, n
         k = n + 1 - kb
         if (k .lt. n) z(k) = z(k) + ddot(n-k,a(k+1,k),1,z(k+1),1)
         if (dabs(z(k)) .le. 1.0d0) go to 110
            s = 1.0d0/dabs(z(k))
            call dscal(n,s,z,1)
  110    continue
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
  120 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
c
      ynorm = 1.0d0
c
c     solve l*v = y
c
      do 140 k = 1, n
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
         if (k .lt. n) call daxpy(n-k,t,a(k+1,k),1,z(k+1),1)
         if (dabs(z(k)) .le. 1.0d0) go to 130
            s = 1.0d0/dabs(z(k))
            call dscal(n,s,z,1)
            ynorm = s*ynorm
  130    continue
  140 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
      ynorm = s*ynorm
c
c     solve  u*z = v
c
      do 160 kb = 1, n
         k = n + 1 - kb
         if (dabs(z(k)) .le. dabs(a(k,k))) go to 150
            s = dabs(a(k,k))/dabs(z(k))
            call dscal(n,s,z,1)
            ynorm = s*ynorm
  150    continue
         if (a(k,k) .ne. 0.0d0) z(k) = z(k)/a(k,k)
         if (a(k,k) .eq. 0.0d0) z(k) = 1.0d0
         t = -z(k)
         call daxpy(k-1,t,a(1,k),1,z(1),1)
  160 continue
c     make znorm = 1.0
      s = 1.0d0/dasum(n,z,1)
```

```
      call dscal(n,s,z,1)
      ynorm = s*ynorm
c
      if (anorm .ne. 0.0d0) rcond = ynorm/anorm
      if (anorm .eq. 0.0d0) rcond = 0.0d0
      return
      end


      subroutine dgedi(a,lda,n,ipvt,det,work,job)
      integer lda,n,ipvt(1),job
      double precision a(lda,1),det(2),work(1)
C
C     dgedi computes the determinant and inverse of a matrix
C     using the factors computed by dgeco or dgefa.
C
C     on entry
C
C        a       double precision(lda, n)
C                the output from dgeco or dgefa.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
C
C        ipvt    integer(n)
C                the pivot vector from dgeco or dgefa.
C
C        work    double precision(n)
C                work vector.  contents destroyed.
C
C        job     integer
C                = 11   both determinant and inverse.
C                = 01   inverse only.
C                = 10   determinant only.
C
C     on return
C
C        a       inverse of original matrix if requested.
C                otherwise unchanged.
C
C        det     double precision(2)
C                determinant of original matrix if requested.
C                otherwise not referenced.
C                determinant = det(1) * 10.0**det(2)
C                with  1.0 .le. dabs(det(1)) .lt. 10.0
C                or  det(1) .eq. 0.0 .
C
C     error condition
C
C        a division by zero will occur if the input factor contains
C        a zero on the diagonal and the inverse is requested.
C        it will not occur if the subroutines are called correctly
C        and if dgeco has set rcond .gt. 0.0 or dgefa has set
C        info .eq. 0 .
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,dswap
```

```
C     fortran dabs,mod
C
C     internal variables
C
      double precision t
      double precision ten
      integer i,j,k,kb,kp1,l,nm1
C
C
C     compute determinant
C
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0d0
         det(2) = 0.0d0
         ten = 10.0d0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
C           ...exit
            if (det(1) .eq. 0.0d0) go to 60
   10       if (dabs(det(1)) .ge. 1.0d0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0d0
            go to 10
   20       continue
   30       if (dabs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0d0
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
C
C     compute inverse(u)
C
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0d0/a(k,k)
            t = -a(k,k)
            call dscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0d0
               call daxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
C
C        form inverse(u)*inverse(l)
C
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0d0
  110       continue
            do 120 j = kp1, n
               t = work(j)
```

```
               call daxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call dswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end

      subroutine daxpy(n,da,dx,incx,dy,incy)
C
C     constant times a vector plus a vector.
C     uses unrolled loops for increments equal to one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C        code for unequal increments or equal increments
C          not equal to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
C
C        code for both increments equal to 1
C
C
C        clean-up loop
C
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end

      subroutine  dscal(n,da,dx,incx)
C
C     scales a vector by a constant.
C     uses unrolled loops for increment equal to one.
C     jack dongarra, linpack, 3/11/78.
```

```
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision da,dx(*)
      integer i,incx,m,mp1,n,nincx
C
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dx(i) = da*dx(i)
   10 continue
      return
C
C        code for increment equal to 1
C
C
C        clean-up loop
C
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end

      subroutine  dswap (n,dx,incx,dy,incy)
C
C     interchanges two vectors.
C     uses unrolled loops for increments equal one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C       code for unequal increments or equal increments not equal
C         to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dx(ix)
        dx(ix) = dy(iy)
        dy(iy) = dtemp
        ix = ix + incx
```

```
        iy = iy + incy
   10 continue
      return
C
C       code for both increments equal to 1
C
C
C       clean-up loop
C
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
        dtemp = dx(i + 1)
        dx(i + 1) = dy(i + 1)
        dy(i + 1) = dtemp
        dtemp = dx(i + 2)
        dx(i + 2) = dy(i + 2)
        dy(i + 2) = dtemp
   50 continue
      return
      end


      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)
C
C     dgefa factors a double precision matrix by gaussian elimination.
C
C     dgefa is usually called by dgeco, but it can be called
C     directly with a saving in time if  rcond  is not needed.
C     (time for dgeco) = (1 + 9/n)*(time for dgefa) .
C
C     on entry
C
C        a       double precision(lda, n)
C                the matrix to be factored.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
C
C     on return
C
C        a       an upper triangular matrix and the multipliers
C                which were used to obtain it.
C                the factorization can be written  a = l*u  where
C                l  is a product of permutation and unit lower
C                triangular matrices and  u  is upper triangular.
C
C        ipvt    integer(n)
C                an integer vector of pivot indices.
```

```
C
C        info    integer
C                = 0  normal value.
C                = k  if  u(k,k) .eq. 0.0 .  this is not an error
C                     condition for this subroutine, but it does
C                     indicate that dgesl or dgedi will divide by zero
C                     if called.  use  rcond  in dgeco for a reliable
C                     indication of singularity.
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,idamax
C
C     internal variables
C
      double precision t
      integer idamax,j,k,kp1,l,nm1
C
C
C     gaussian elimination with partial pivoting
C
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
C
C        find l = pivot index
C
         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
C
C        zero pivot implies this column already triangularized
C
         if (a(l,k) .eq. 0.0d0) go to 40
C
C           interchange if necessary
C
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
C
C           compute multipliers
C
            t = -1.0d0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
C
C           row elimination with column indexing
C
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
```

```
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d0) info = n
      return
      end

      integer function idamax(n,dx,incx)
C
C     finds the index of element having max. absolute value.
C     jack dongarra, linpack, 3/11/78.
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dmax
      integer i,incx,ix,n
C
      idamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      ix = 1
      dmax = dabs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(dabs(dx(ix)).le.dmax) go to 5
         idamax = i
         dmax = dabs(dx(ix))
    5    ix = ix + incx
   10 continue
      return
C
C        code for increment equal to 1
C
   20 dmax = dabs(dx(1))
      do 30 i = 2,n
         if(dabs(dx(i)).le.dmax) go to 30
         idamax = i
         dmax = dabs(dx(i))
   30 continue
      return
      end

      double precision function dasum(n,dx,incx)
c
c     takes the sum of the absolute values.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dtemp
      integer i,incx,m,mp1,n,nincx
c
      dasum = 0.0d0
      dtemp = 0.0d0
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
```

```
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dtemp = dtemp + dabs(dx(i))
   10 continue
      dasum = dtemp
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,6)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dabs(dx(i))
   30 continue
      if( n .lt. 6 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,6
        dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
     &  + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
   50 continue
   60 dasum = dtemp
      return
      end

      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
```

```
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     & dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
```

Detcov.params.h:

```
C*************************************************************
C                                                            *
C  Parameter input file for 'detcov'                         *
C    Author: Yao Lin                                         *
C      Date: 3/26/2003                                       *
C                                                            *
C*************************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=1,numsamp=11,fprefix='suit3altvalid',
     &          coedet=1e4 )
C
C  numdv = # design variables
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C  coedet = when the value of determinant is very small,
C          this coefficient is used to magnify the value.
C
C*************************************************************
```

C.2 IMPLEMENTATION OF E-RCEM IN ISIGHT IN SECTION 5.5

Figures presented in this section illustrate how the SEED method is implemented in iSIGHT. The organization of tasks in Iteration I – Step 7 is shown in Figure C.1. In Iteration I – Step 7, with information from metamodels of prediction errors, we use five simulation codes in iSIGHT, i.e., Covmat, Errpred, Response, Altcov, and Detcov. Covmat is used to formulate the covariance matrix, Errpred is the metamodel to predict

prediction errors, Response is the metamodel to predict response values, Altcov is used to adjust entries of the covariance matrix, and Detcov is used to calculate the determinant.

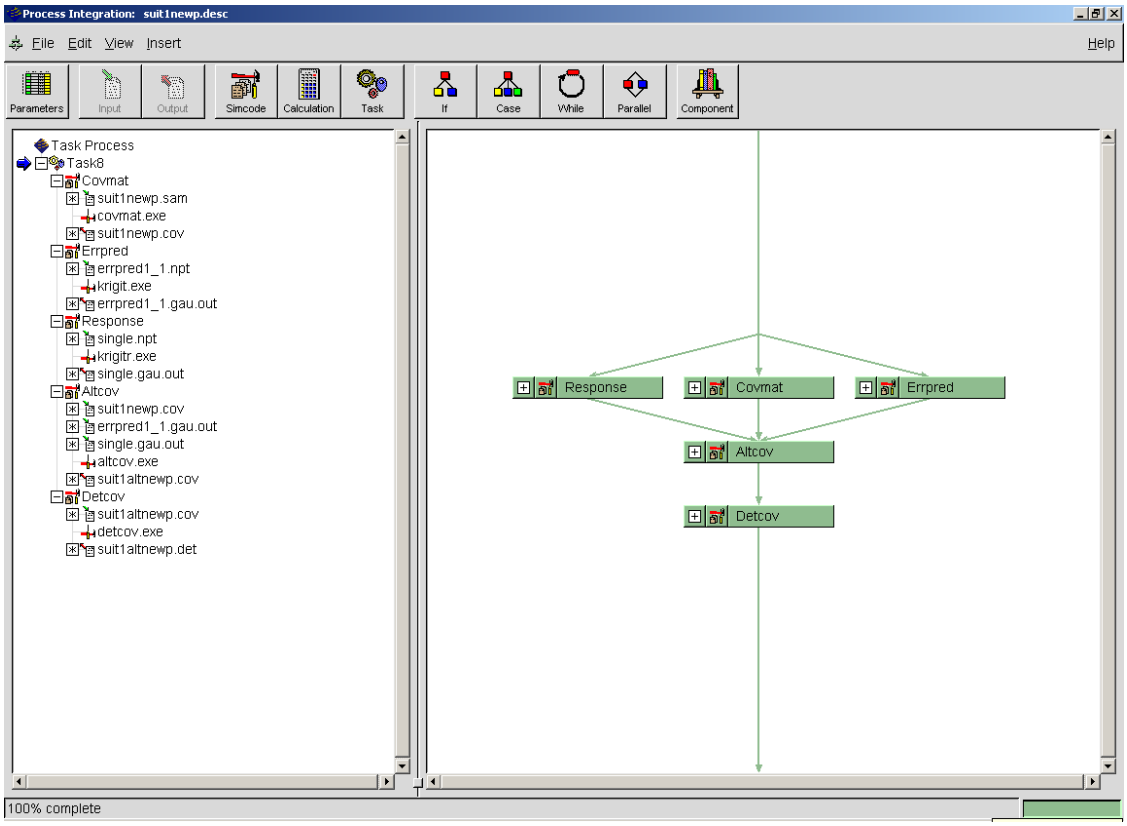


Figure C.1 Implementation of E-RCEM in iSIGHT – Iteration I, Step 7

APPENDIX D

DESIGN OF UNIT CELLS FOR LINEAR CELLULAR ALLOYS: EXPERIMENTS, SIMULATION RESULTS, PROGRAMS, METAMODELS, AND PLOTS

This appendix is intended to supplement the application of SEED and E-RCEM methods in designing unit cells for linear cellular alloys in Chapter 7. The experimental designs, simulation results, metamodels, and plots developed in Section 7.3 are presented in Section D.1. Supporting materials for the application of SEED (Section 7.4) and E-RCEM (Section 7.5) are enclosed in Sections D.2 and D.3, respectively.

D.1 EXPLORATION OF DESIGN SOLUTIONS WITH RCEM

In this section we collect supporting materials for studies in Section 7.3.

D.1.1 Latin Hypercube Design with 30 Data Points

Table D.1 Latin Hypercube Design – 30 Data Points Used in RCEM in Section 7.3

Mdot (kg/s)	W (m)	t (m)	Mdot_n	W_n	t_n	Q (W)	J (m/N)
0.0005	0.0219	0.0008	0	0.3448	1	-13.64	0.00024
0.00059	0.0171	0.00061	0.03448	0.1034	0.6897	-13.80	0.00041
0.00067	0.0309	0.00078	0.06897	0.7931	0.9655	-13.19	0.00028
0.00076	0.0233	0.00041	0.1034	0.4138	0.3448	-13.27	0.00126
0.00084	0.035	0.00045	0.1379	1	0.4138	-12.01	0.00116
0.00093	0.0295	0.00053	0.1724	0.7241	0.5517	-14.45	0.00070
0.00102	0.0247	0.00072	0.2069	0.4828	0.8621	-16.31	0.00031
0.0011	0.0205	0.00068	0.2414	0.2759	0.7931	-16.01	0.00034
0.00119	0.0191	0.00032	0.2759	0.2069	0.2069	-14.59	0.00225
0.00128	0.0198	0.00057	0.3103	0.2414	0.6207	-15.88	0.00050
0.00136	0.0288	0.0007	0.3448	0.6897	0.8276	-17.76	0.00035
0.00145	0.0322	0.0002	0.3793	0.8621	0	-15.14	0.01122
0.00153	0.0184	0.00051	0.4138	0.1724	0.5172	-15.77	0.00065
0.00162	0.0267	0.00055	0.4483	0.5862	0.5862	-17.69	0.00061
0.00171	0.015	0.0003	0.4828	0	0.1724	-14.53	0.00239
0.00179	0.0212	0.00076	0.5172	0.3103	0.931	-17.43	0.00026
0.00188	0.0157	0.00063	0.5517	0.03448	0.7241	-51.85	0.00037
0.00197	0.0164	0.00039	0.5862	0.06897	0.3103	-15.31	0.00123
0.00205	0.0281	0.00066	0.6207	0.6552	0.7586	-19.29	0.00040
0.00214	0.0226	0.00028	0.6552	0.3793	0.1379	-16.63	0.00356
0.00222	0.0274	0.00024	0.6897	0.6207	0.06897	-17.53	0.00610
0.00231	0.0178	0.00022	0.7241	0.1379	0.03448	-15.26	0.00627
0.0024	0.026	0.00049	0.7586	0.5517	0.4828	-18.67	0.00082
0.00248	0.0343	0.00074	0.7931	0.9655	0.8966	-21.73	0.00032
0.00257	0.0302	0.00026	0.8276	0.7586	0.1034	-18.80	0.00507
0.00266	0.0253	0.00037	0.8621	0.5172	0.2759	-18.31	0.00172
0.00274	0.0316	0.00034	0.8966	0.8276	0.2414	-19.87	0.00241
0.00283	0.0329	0.00043	0.931	0.8966	0.3793	-20.76	0.00128
0.00291	0.0336	0.00059	0.9655	0.931	0.6552	-21.75	0.00056
0.003	0.024	0.00047	1	0.4483	0.4483	-18.51	0.00088

D.1.2 MARS Metamodel of Responses Developed with 30 LH Experiments

Qmars.dat for Total Heat Transfer Rate *Q*:

```
   3  28
      0.499998333333333       0.499998333333333       0.499998333333333
      0.500000000000000       0.500000000000000       0.500000000000000
   1   1   1   1   1   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
1   2
     -0.339870036111432         -12.773472901115134         -19.863793680013131         -
46.524074229003666           35.691924559997979           13.394144037678506         -
1681.581052686991800         -184.515789535457710        -1198.261206478433200         -
6.568937120765133           -49.284612979016075            87.083329828075918
12.650147584392984           63.973544584529279           49.554777358052647         -
344.571366217510730           94.750374353128009            7.178640037521877         -
52.280639764008647            56.193684386954324          -109.039095462713310
220.209200073127250           30.784057093017406           69.569559469737314         -
50.105432708334561           -6.049454931256710            1.041273408861595
49.915412264619803     353.571628624746100
 -1   1     -0.448296666666667       0.103403333333333       0.551703333333333
  1   1     -0.448296666666667       0.103403333333333       0.551703333333333
 -1   3      0.413803333333333       0.448203333333333       0.499803333333333
  1   3      0.413803333333333       0.448203333333333       0.724103333333333
 -1   3     -0.310296666666667       0.379403333333333       0.413803333333333
  1   3      0.327803333333333       0.379403333333333       0.413803333333333
 -1   2     -0.724096666666667      -0.448196666666667      -0.241296666666667
  1   3      0.327803333333333       0.379403333333333       0.413803333333333
  1   2     -0.724096666666667      -0.448196666666667      -0.241296666666667
  1   1     -0.448296666666667       0.103403333333333       0.517203333333333
 -1   2     -0.862096666666667      -0.724196666666667      -0.517346666666667
  1   1     -0.448296666666667       0.103403333333333       0.517203333333333
  1   2     -0.862096666666667      -0.724196666666667      -0.517296666666667
 -1   3      0.413803333333333       0.448203333333333       0.499803333333333
 -1   2      0.620703333333334       0.724203333333333       0.758703333333333
 -1   3      0.413803333333333       0.448203333333333       0.499803333333333
  1   2      0.672453333333333       0.724203333333333       0.758703333333333
 -1   1     -0.448296666666667       0.103403333333333       0.517203333333333
 -1   2      0.103503333333333       0.379403333333333       0.689703333333333
 -1   1     -0.448296666666667       0.103403333333333       0.517203333333333
  1   2      0.103503333333333       0.379403333333333       0.689703333333333
 -1   3     -0.310296666666667       0.379403333333333       0.413803333333333
 -1   2      0.758703333333333       0.793203333333333       0.844953333333333
 -1   3     -0.310296666666667       0.379403333333333       0.413803333333333
  1   2      0.758703333333333       0.793203333333333       0.896603333333333
 -1   1     -0.034496666666667       0.103403333333333       0.310253333333333
 -1   3     -0.655196666666667      -0.310396666666667       0.000003333333333
 -1   1     -0.034496666666667       0.103403333333333       0.310253333333333
```

```
 1   3    -0.655196666666667    -0.310396666666667     0.000003333333333
 1   1    -0.034496666666667     0.103403333333333     0.551703333333333
-1   3     0.000003333333333     0.310403333333333     0.344903333333333
 1   1    -0.034496666666667     0.103403333333333     0.551703333333333
 1   3     0.258653333333333     0.310403333333333     0.344903333333333
 1   3     0.413803333333333     0.448203333333333     0.724103333333333
-1   2    -0.241296666666667    -0.034396666666667     0.000003333333333
 1   3     0.413803333333333     0.448203333333333     0.724103333333333
 1   2    -0.085996666666666    -0.034396666666667     0.000003333333333
-1   3     0.344903333333333     0.379403333333333     0.431153333333333
-1   1    -0.586196666666667    -0.172396666666667    -0.034496666666667
-1   3     0.344903333333333     0.379403333333333     0.431153333333333
 1   1    -0.379246666666667    -0.172396666666667    -0.034496666666667
-1   1    -0.448296666666667     0.103403333333333     0.517203333333333
-1   2    -0.241396666666667    -0.172396666666667    -0.068896666666667
-1   3     0.413803333333333     0.448203333333333     0.499803333333333
-1   2     0.000003333333333     0.034403333333333     0.086003333333333
-1   3    -0.310296666666667     0.379403333333333     0.413803333333333
-1   2     0.275803333333334     0.517203333333334     0.620703333333334
-1   2    -0.655196666666667    -0.310396666666667     0.206803333333333
 1   2    -0.413896666666667    -0.310396666666667    -0.241396666666667
 1   1     0.879253333333333     0.931003333333333     0.965503333333333
```

Qmars.dat for Compliance *J*:

```
   3  27
      0.499998333333333       0.499998333333333       0.499998333333333
      0.500000000000000       0.500000000000000       0.500000000000000
   1   1   2   2   1   2   2   1   1   1   1   2   2   2   2   1   1   2   2   2   2   2   2   2   2   2
      0.001466407963489                -0.000192918064502                -0.001245038659639
0.009567743769789       0.144035227182990       0.016246182925700      -0.010048035980419
-0.032223805341409               0.008670185519562               0.000142024396652      -
0.000806115246652       0.000436531100816      -0.010530678740980      -0.065287694525146
0.007554843266843      -0.000978118081569      -0.015116318652959       0.000102440282160
0.000677603505063      -0.015011756040837      -0.000459155980944       0.000184045816162
0.000115606777776      -0.000320462363114       0.001366597092628       0.000152516546845
-0.000160190632074      -0.000211840744665
-1   3    -0.827596666666667    -0.655196666666667    -0.517296666666667
 1   3    -0.827596666666667    -0.655196666666667    -0.517296666666667
-1   3    -0.827596666666667    -0.655196666666667    -0.517296666666667
-1   2     0.344803333333334     0.517203333333334     0.758603333333334
-1   3    -0.827596666666667    -0.655196666666667    -0.517296666666667
 1   2     0.344803333333334     0.517203333333334     0.758603333333334
-1   3    -0.137896666666667    -0.103396666666667    -0.051646666666667
```

-1	3	-0.241396666666667	-0.103396666666667	-0.034496666666667
-1	2	0.344803333333334	0.517203333333334	0.758603333333334
-1	3	-0.241396666666667	-0.103396666666667	-0.034496666666667
1	2	0.344803333333334	0.517203333333334	0.758603333333334
-1	3	-0.517296666666667	-0.379396666666667	-0.275896666666667
-1	3	0.310303333333333	0.448203333333333	0.655053333333333
-1	2	-0.448296666666667	0.103403333333333	0.551703333333333
1	2	-0.448296666666667	0.103403333333333	0.551703333333333
-1	3	-0.827596666666667	-0.655196666666667	-0.413796666666667
-1	1	0.344803333333333	0.448203333333333	0.603303333333333
-1	3	-0.827596666666667	-0.655196666666667	-0.413796666666667
1	1	0.344803333333333	0.448203333333333	0.689603333333333
-1	2	-0.448296666666667	0.103403333333333	0.137903333333333
-1	3	-0.034496666666667	0.034403333333333	0.137753333333333
-1	2	-0.448296666666667	0.103403333333333	0.137903333333333
1	3	-0.034496666666667	0.034403333333333	0.517203333333333
-1	3	-0.275896666666667	-0.172396666666667	-0.137896666666667
-1	3	0.034503333333333	0.172403333333333	0.310303333333333
1	3	-0.413796666666667	-0.172396666666667	0.413803333333333
-1	1	0.689603333333333	0.931003333333333	0.965503333333333
1	3	-0.413796666666667	-0.172396666666667	0.413803333333333
1	1	0.879253333333333	0.931003333333333	0.965503333333333
1	3	-0.827596666666667	-0.655196666666667	-0.413796666666667
-1	1	-0.379296666666667	0.241403333333333	0.344803333333333
1	3	-0.827596666666667	-0.655196666666667	-0.413796666666667
1	1	0.086303333333334	0.241403333333333	0.344803333333333
-1	2	-0.448296666666667	0.103403333333333	0.551703333333333
-1	1	-0.172396666666667	-0.034396666666667	0.172603333333333
-1	2	-0.448296666666667	0.103403333333333	0.551703333333333
1	1	-0.172396666666667	-0.034396666666667	0.482803333333333
1	3	-0.517296666666667	-0.379396666666667	-0.241396666666667
-1	2	0.137903333333333	0.172403333333334	0.224153333333334
1	3	-0.517296666666667	-0.379396666666667	-0.241396666666667
1	2	0.137903333333333	0.172403333333334	0.344803333333334
1	2	-0.448296666666667	0.103403333333333	0.551703333333333
-1	1	-0.655196666666667	-0.310396666666667	-0.172396666666667
1	2	-0.448296666666667	0.103403333333333	0.551703333333333
1	1	-0.517396666666667	-0.310396666666667	-0.172396666666667

D.1.3 Formulating and Solving C-DSP in iSIGHT

The compromise DSP is formulated in Figure 7.13. To solve this compromise DSP, we use the automation and exploration software iSIGHT. Presented below are plots

illustrating the implementation of iSIGHT in solving the compromise DSP in Section 7.3. The overall organization of tasks of C-DSP in iSIGHT is illustrated in Figure D.1 The file parsing process for *Q* and the calculation of the design goal are illustrated in Figure D.2 and Figure D.3, respectively. In Figure D.1, the simulation codes Q and J are kriging metamodels to predict response values at the current point; the simulation code Constraints is a model to calculate all 3 design constraints (as described in Section 7.2) and the value of A_f (cross-section area of the cells). *Q*, *J*, and A_f are then used to calculate the deviation variables.

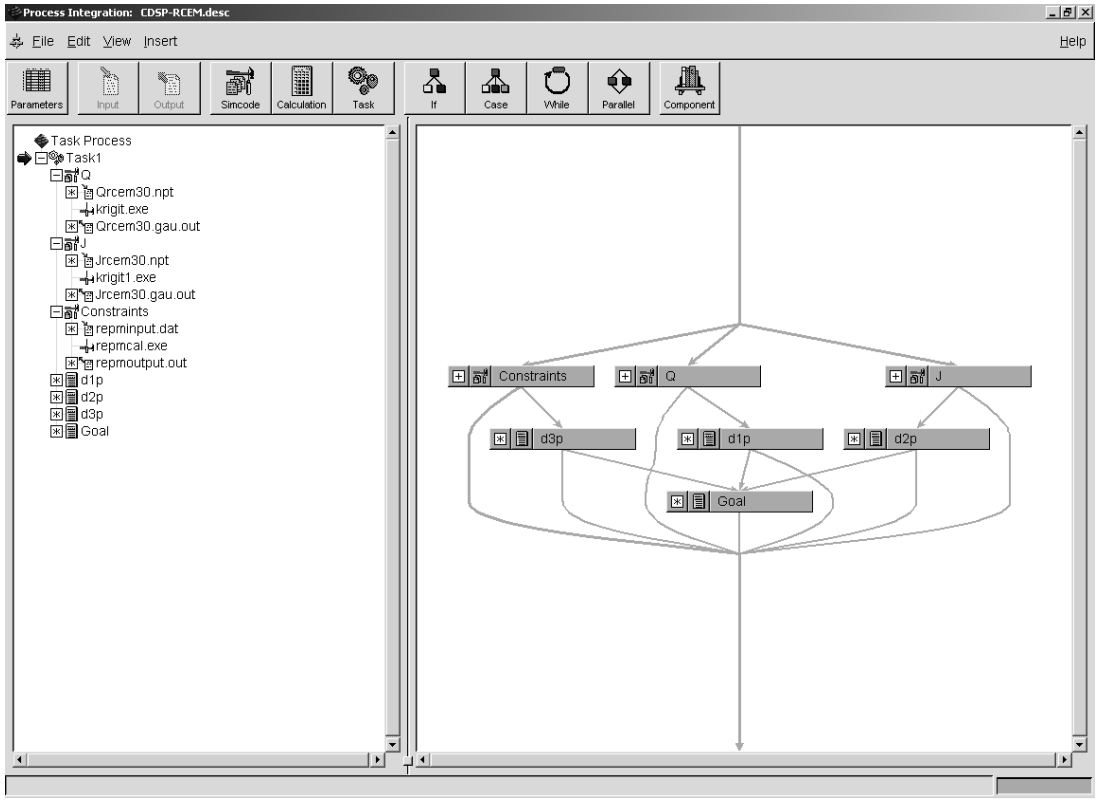


Figure D.1 Solving C-DSP in iSIGHT – Overall Organization of Tasks

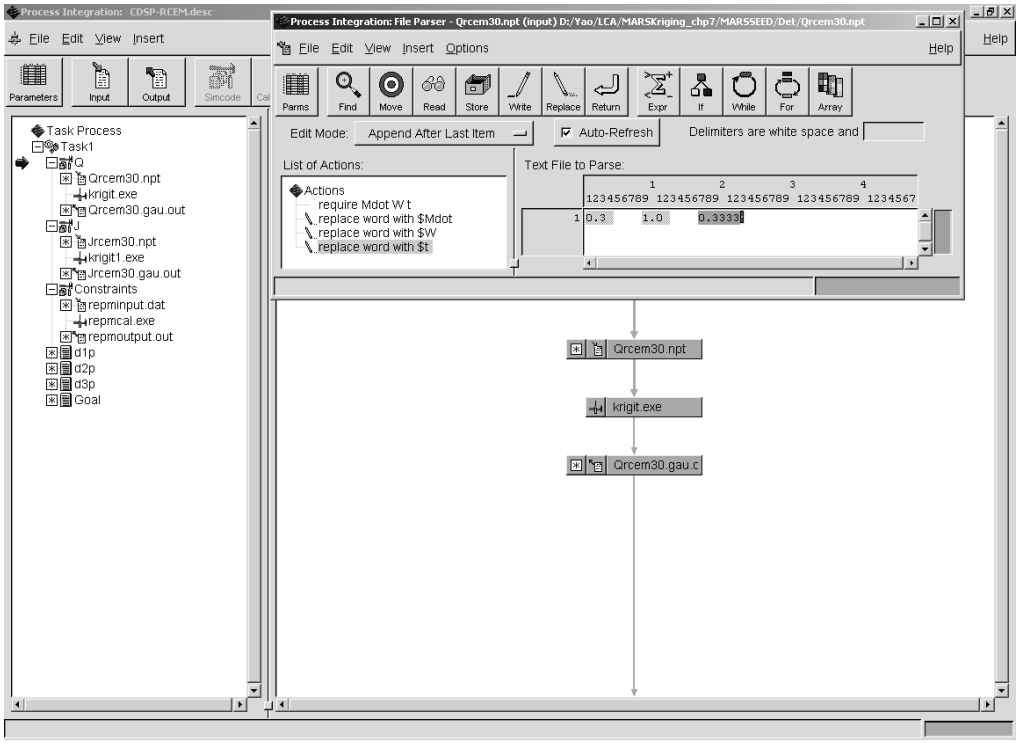


Figure D.2 Solving C-DSP – File Parsing for Input

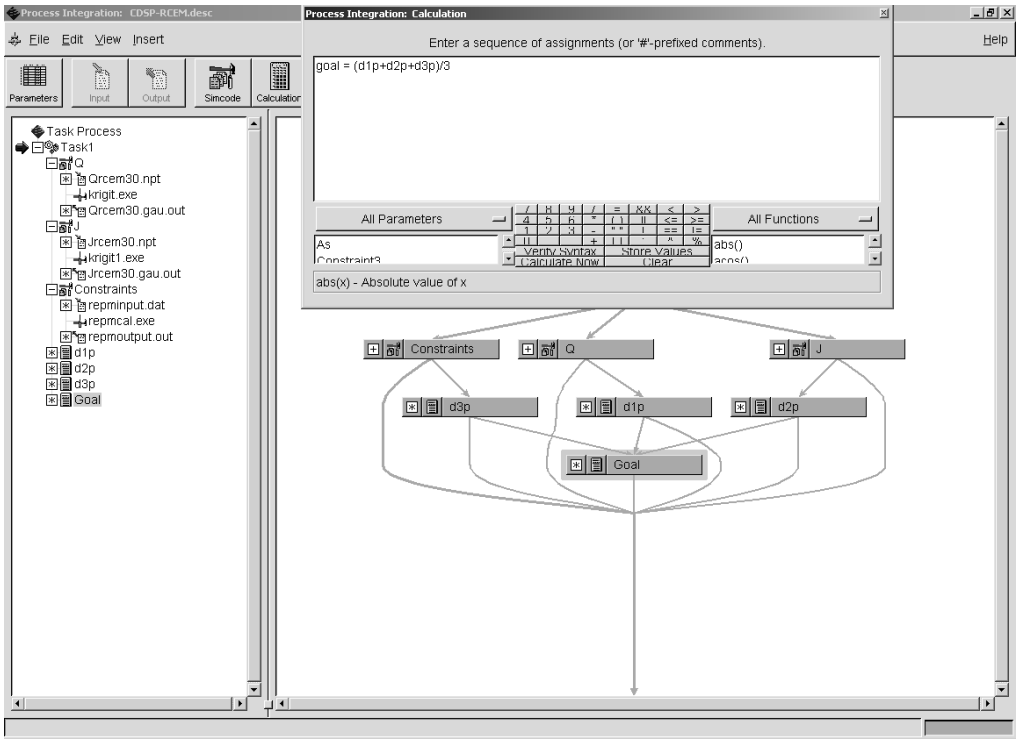


Figure D.3 Solving C-DSP in iSIGHT – Calculation of the Design Goal

D.1.4 Latin Hypercube Design with 40 Data Points

Table D.2 Latin Hypercube Design – 40 Data Points Used in RCEM in Section 7.3

Mdot (kg/s)	W (m)	t (m)	Mdot_n	W_n	t_n	Q (W)	J (m/N)
0.0005	0.03141	0.000646	0	0.8205	0.7436	-9.85	0.00043
0.000564	0.019616	0.000769	0.02564	0.2308	0.9487	-14.32	0.00025
0.000628	0.02218	0.000615	0.05128	0.359	0.6923	-13.61	0.00044
0.000692	0.01859	0.000477	0.07692	0.1795	0.4615	-13.58	0.00077
0.000757	0.018076	0.000446	0.1026	0.1538	0.4103	-13.74	0.00090
0.000821	0.026282	0.000754	0.1282	0.5641	0.9231	-15.34	0.00028
0.000885	0.020128	0.000523	0.1538	0.2564	0.5385	-14.67	0.00063
0.000949	0.02423	0.000662	0.1795	0.4615	0.7692	-15.70	0.00038
0.001013	0.020642	0.000431	0.2051	0.2821	0.3846	-14.74	0.00104
0.001077	0.02577	0.0008	0.2308	0.5385	1	-16.96	0.00025
0.001141	0.028334	0.000262	0.2564	0.6667	0.1026	-14.29	0.00484
0.001205	0.035	0.000569	0.2821	1	0.6154	-15.89	0.00062
0.001269	0.028846	0.000385	0.3077	0.6923	0.3077	-15.73	0.00164
0.001333	0.015	0.000677	0.3333	0	0.7949	-46.55	0.00031
0.001398	0.032948	0.000338	0.359	0.8974	0.2308	-15.89	0.00248
0.001462	0.022692	0.0004	0.3846	0.3846	0.3333	-16.07	0.00133
0.001526	0.016026	0.000492	0.4103	0.05128	0.4872	-15.18	0.00067
0.00159	0.029358	0.000415	0.4359	0.7179	0.359	-17.23	0.00134
0.001654	0.02782	0.000354	0.4615	0.641	0.2564	-16.98	0.00203
0.001718	0.021666	0.0002	0.4872	0.3333	0	-15.35	0.00919
0.001782	0.017564	0.000738	0.5128	0.1282	0.8974	-46.78	0.00027
0.001846	0.031924	0.0006	0.5385	0.8462	0.6667	-19.19	0.00052
0.00191	0.023718	0.000231	0.5641	0.4359	0.05128	-16.25	0.00636
0.001974	0.029872	0.000785	0.5897	0.7436	0.9744	-19.98	0.00027
0.002039	0.033974	0.000308	0.6154	0.9487	0.1795	-18.34	0.00330
0.002103	0.015513	0.000554	0.641	0.02564	0.5897	-51.65	0.00050
0.002167	0.017052	0.000708	0.6667	0.1026	0.8462	-55.53	0.00029
0.002231	0.025256	0.000462	0.6923	0.5128	0.4359	-18.16	0.00095
0.002295	0.032436	0.000369	0.7179	0.8718	0.2821	-19.30	0.00193
0.002359	0.019102	0.000631	0.7436	0.2051	0.7179	-43.92	0.00039
0.002423	0.016538	0.000508	0.7692	0.07692	0.5128	-48.67	0.00063
0.002487	0.030898	0.000323	0.7949	0.7949	0.2051	-19.19	0.00275
0.002551	0.024744	0.000246	0.8205	0.4872	0.07692	-17.45	0.00539
0.002616	0.026794	0.000723	0.8462	0.5897	0.8718	-19.85	0.00031

0.00268	0.021154	0.000538	0.8718	0.3077	0.5641	-17.52	0.00059
0.002744	0.030384	0.000692	0.8974	0.7692	0.8205	-21.02	0.00036
0.002808	0.034488	0.000292	0.9231	0.9744	0.1538	-20.22	0.00385
0.002872	0.027308	0.000585	0.9487	0.6154	0.641	-19.86	0.00053
0.002936	0.023206	0.000215	0.9744	0.4103	0.02564	-17.17	0.00768
0.003	0.033462	0.000277	1	0.9231	0.1282	-20.28	0.00443

D.2 EXPLORATION OF DESIGN SOLUTIONS WITH RCEM

All supporting materials and documents for studies in Section 7.4 are presented here. Contours plots of metamodels of responses (initial metamodels, metamodels of responses in Iteration I – Step 8 and Iteration II – Step 3) are illustrated in Section D.2.1. FORTRAN codes of SEED are presented in Section D.2.2. The implementation of SEED in iSIGHT is illustrated in Section D.2.3. Twenty-eight points identified from SEED and their corresponding response values are listed in Section D.2.4.

D.2.1 Contour Plots of Metamodels of Responses

Contour plots illustrated below are drawn with predicted values from the kriging metamodels of responses in Section 7.4. Contour plots of metamodels of prediction errors are not drawn. The contour plots are drawn using Minitab® with default parameters.

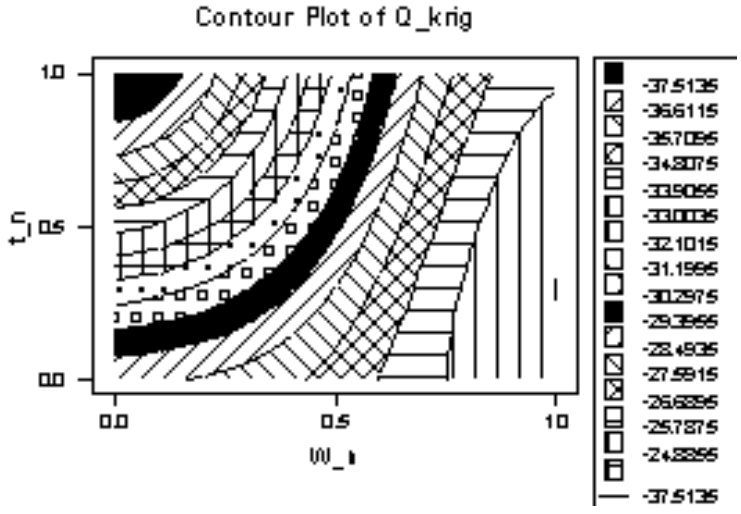


Figure D.4 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Device Width (Initial Kriging Metamodel with 8 Data Points)

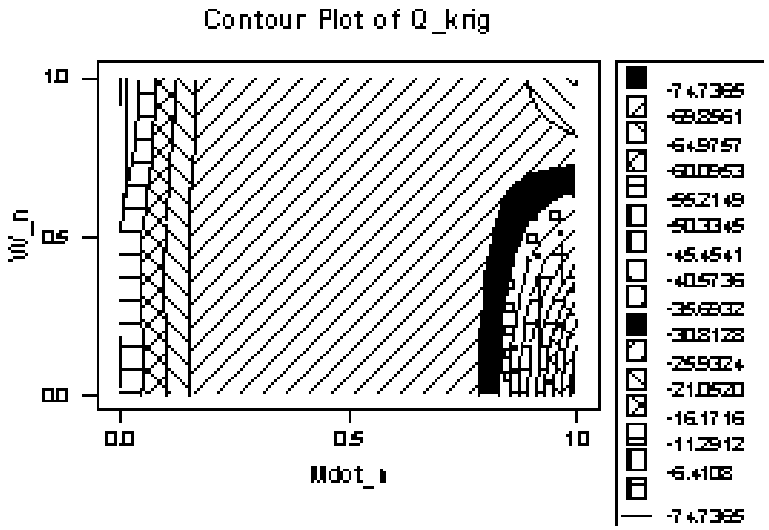


Figure D.5 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Initial Kriging Metamodel with 8 Data Points)

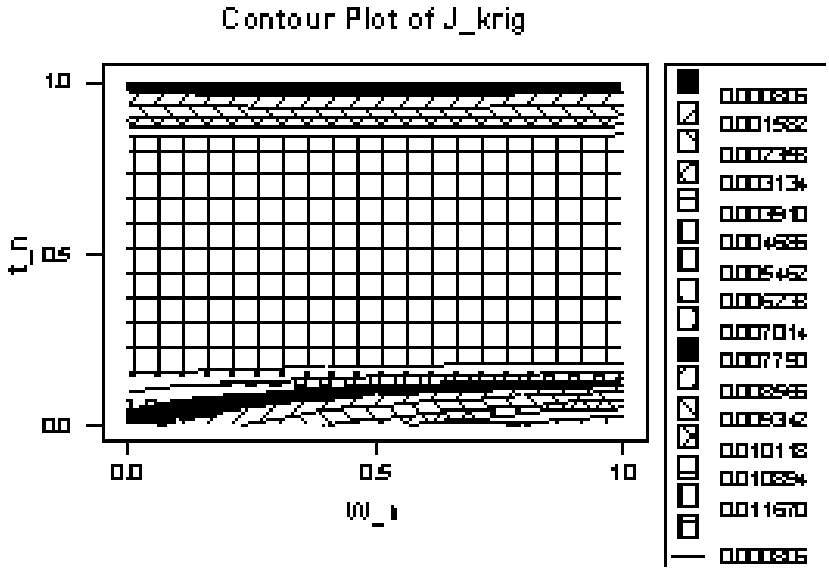


Figure D.6 Contour Plot of Compliance vs. Device Width and Wall Thickness (Initial Kriging Metamodel with 8 Data Points)

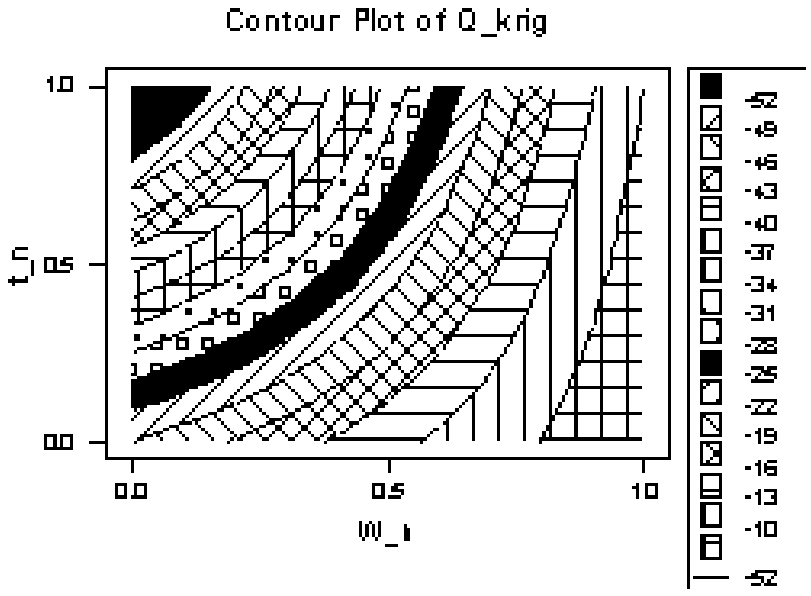


Figure D.7 Contour Plot of Heat Transfer Rate vs. Device Width and Wall Thickness (Kriging Metamodel with 11 Data Points – Iteration I, Step 8)

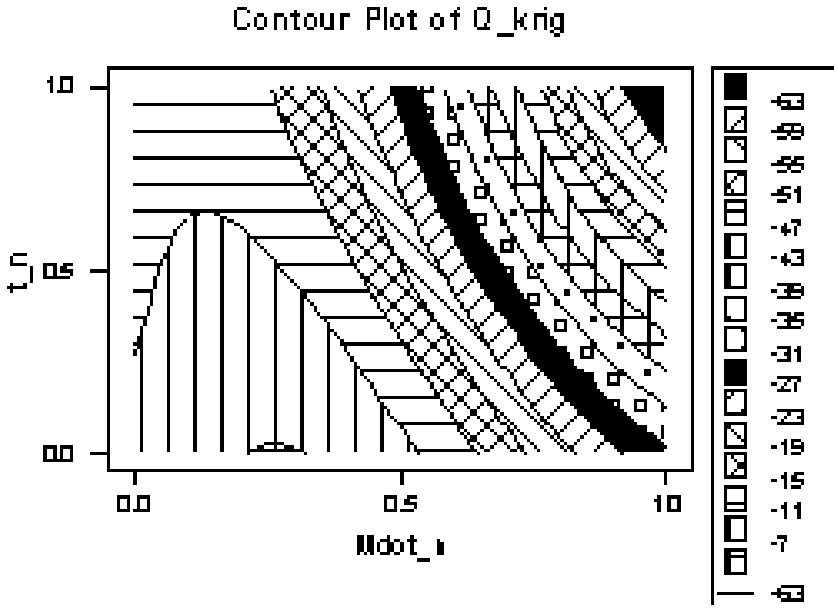


Figure D.8 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 11 Data Points – Iteration I, Step 8)

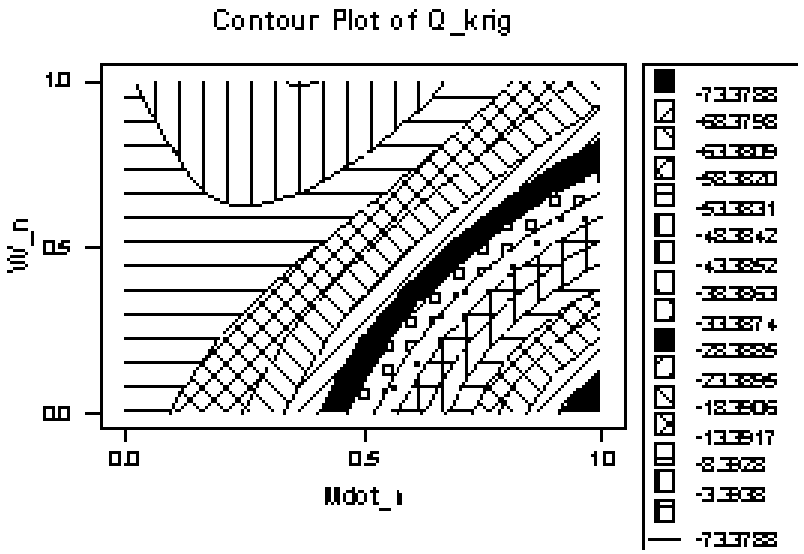


Figure D.9 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 11 Data Points – Iteration I, Step 8)

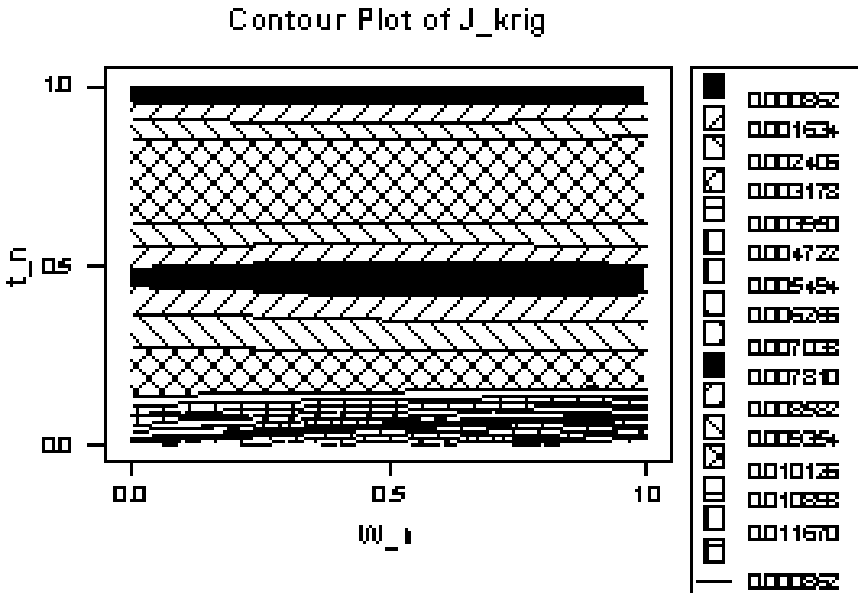


Figure D.10 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 11 Data Points – Iteration I, Step 8)

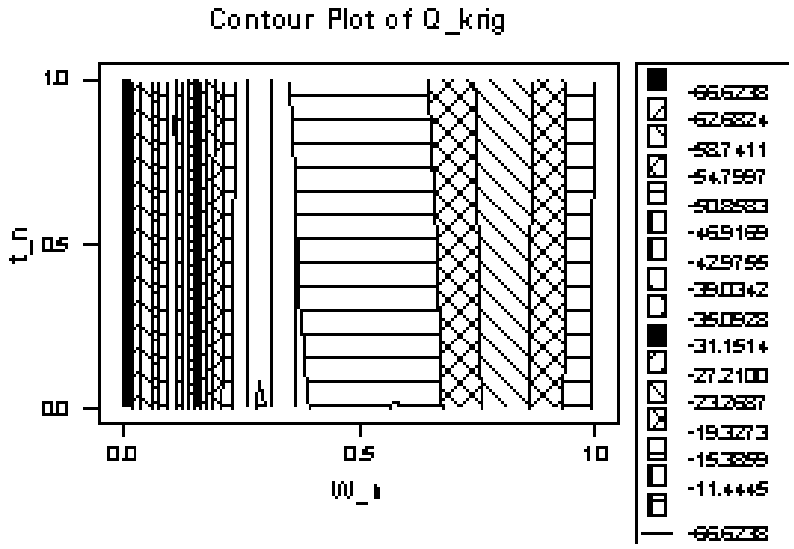


Figure D.11 Contour Plot of Heat Transfer Rate vs. Device Width and Wall Thickness (Kriging Metamodel with 8 Validation Points – Iteration II, Step 3)

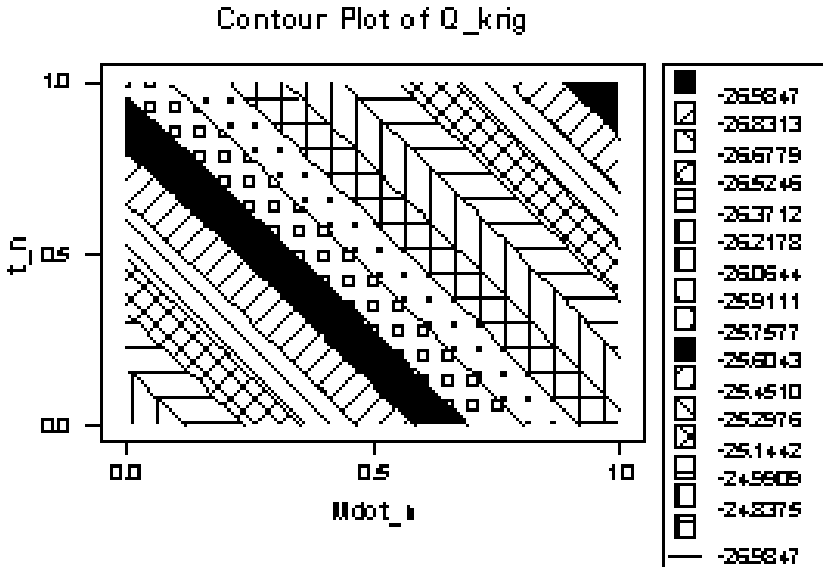


Figure D.12 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 8 Validation Points – Iteration II, Step 3)

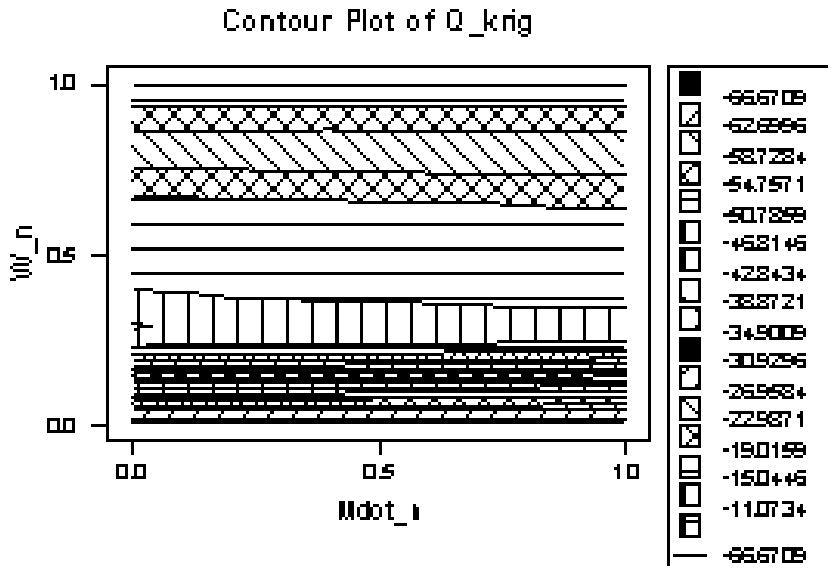


Figure D.13 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 8 Validation Points – Iteration II, Step 3)

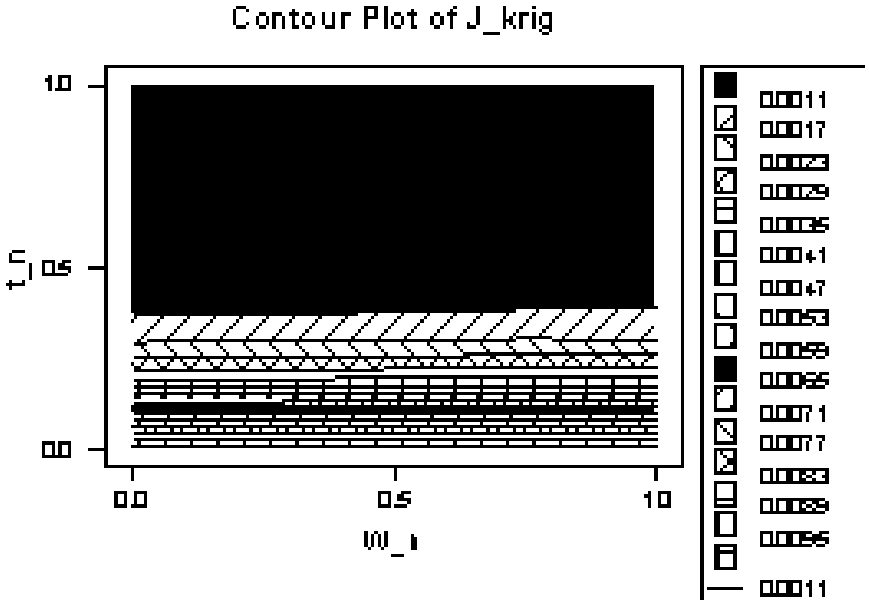


Figure D.14 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 8 Validation Points – Iteration II, Step 3)

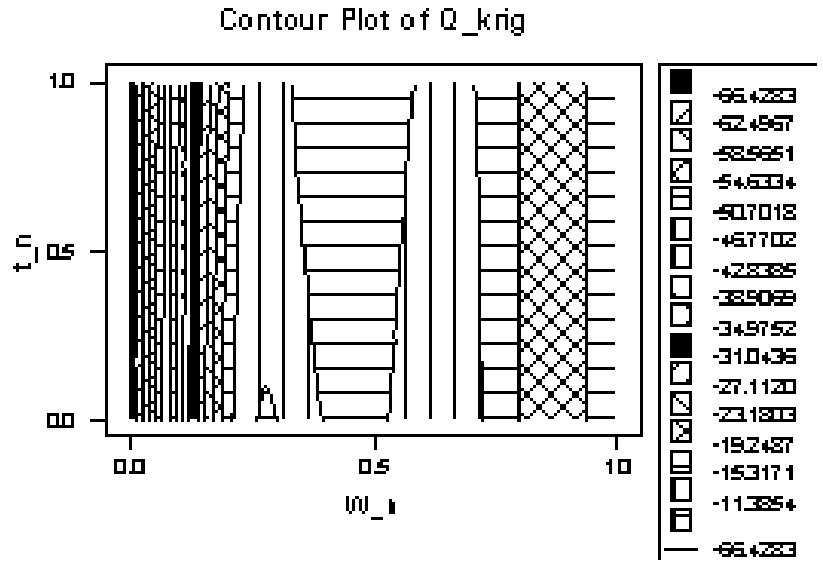


Figure D.15 Contour Plot of Heat Transfer Rate vs. Device Width and Wall Thickness (Kriging Metamodel with 11 Validation Points – Iteration III, Step 3)

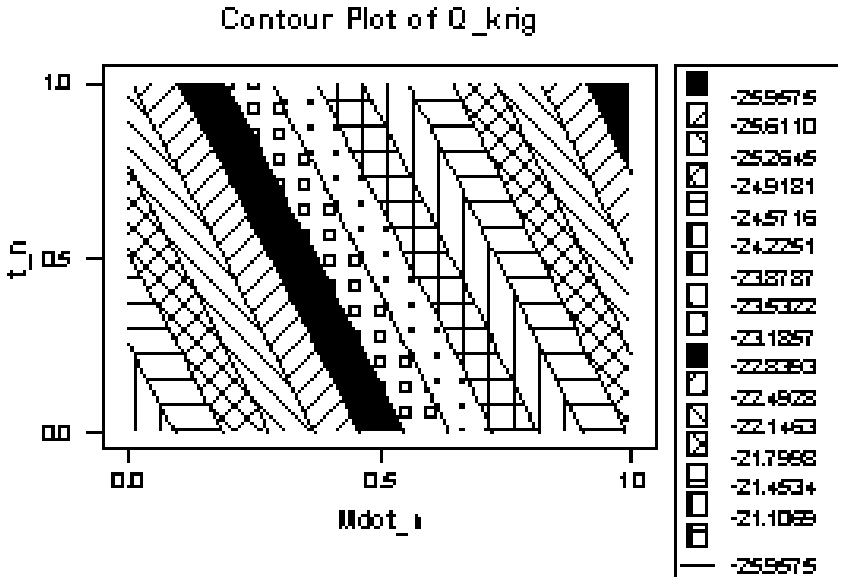


Figure D.16 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Mass Flow Rate (Kriging Metamodel with 11 Validation Points – Iteration III, Step 3)

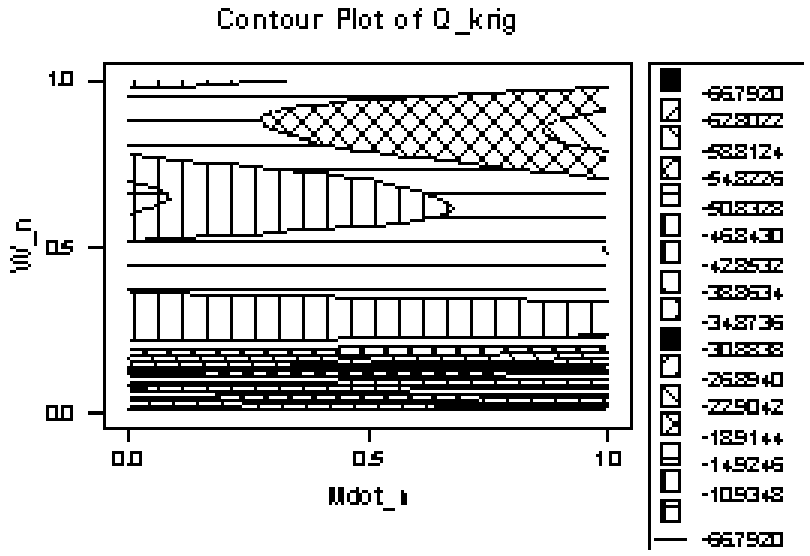


Figure D.17 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 11 Validation Points – Iteration III, Step 3)

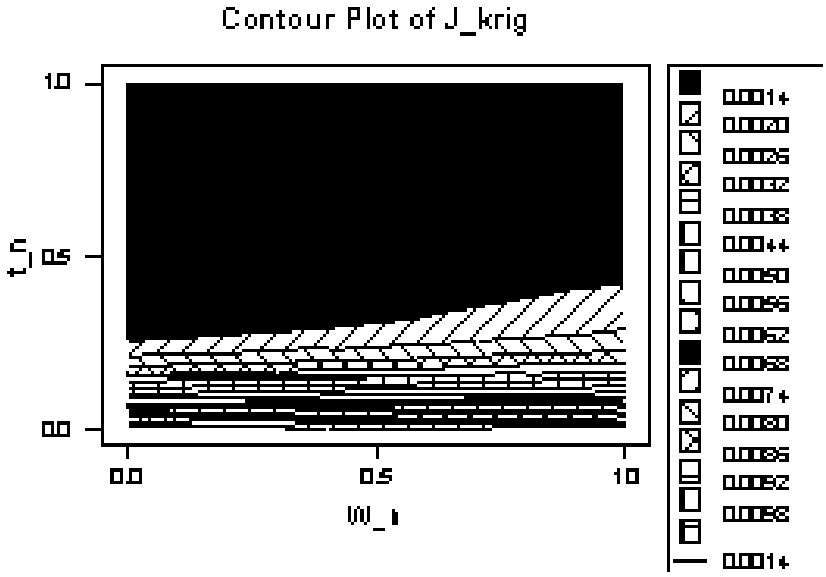


Figure D.18 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 11 Validation Points – Iteration III, Step 3)

D.2.2 FORTRAN Programs Used in SEED in Section 7.4

The FORTRAN programs used in SEED, Iteration III – Step 3, in Section 7.4 are enclosed in this section. To formulate the covariance matrix we use covmat.f and covdata.params.h; the input and output filenames are specified in covdata.params.h. To adjust entries of the covariance matrix we use altcov.f and altcov.params.h. To calculate the determinant of the covariance matrix we use detcov.f and detcov.params.h.

Covmat.f:

```
************************************************************************
*
      program covmat
*
*  This program invokes calculation of the correlation matrix given
*        information of points and values of theta.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Yao Lin 26 March 2003 / Tim Simpson, 25 Feburary 1998
*
************************************************************************
*
* Input files:
* ------------
*  covdata.params.h - parameter file, specifying numdv, numsamp, fprefix
*  .sam             - x's of sample points
*  .gau.fit         - thetas
*
* Output files:
* -------------
*  .cov             - correlation matrix
*
* Variables:
* ----------
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*   numsamp = number of data samples from which the correlation matrix
*               is calculated
*
* Local Variables:
* ----------------
*   DOUBLE PRECISION
*   ----------------
*   xmat     = numdv x numsamp of sample site locations, scaled [0,1]
*
*   INTEGER
*   -------
*
************************************************************************
```

```
      integer numdv,numsamp
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix, e.g., in the
C  one-variable problem: numdv=1,numsamp=5,fprefix='step1'
C
      include 'covdata.params.h'

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   dummy2,thetaray(1,numdv),theta(numdv)
      integer i,j,dummy,lenstr
      character*16 ftitle
      character*20 deckfile,fitsfile,outfile


C
C  open necessary .sam, .fit, and .cov files based on 'fprefix' name,
C  e.g., in the one-variable problem:
C        step1.sam, step1.gau.fit, step1.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.sam'
       fitsfile=ftitle(1:lenstr) // '.gau.fit'
       outfile=ftitle(1:lenstr) // '.cov'

       open(21,file=deckfile,status='old')
       open(22,file=fitsfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,fitsfile,outfile
       print *, numdv,numsamp
C
C  initialize xmat and theta arrays
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (xmat(i,j),j=1,numdv)
      close(21)

      print *
      write(6,*) 'Reading in theta parameters...'
      do 20 i=1,1
        read(22,*) dummy,(thetaray(i,j),j=1,numdv),dummy2
        write(6,1000) dummy,(thetaray(i,j),j=1,numdv)
 1000   format(i2,8f9.5)
 20   continue
      close(22)

      do 50 j=1,numdv
          theta(j)=thetaray(1,j)
 50     continue
        write(6,1002) (theta(j),j=1,numdv)
 1002     format(8f9.5)


C
C  call subroutine to calculate the correlation matrix
C
```

```
C  input:  xmat, theta, numsamp, numdv
C
C  output: R - the correlation matrix
C

         call cormat (xmat,cov,numsamp,numdv,theta)

C
C  write predicted values to specified .cov file
C
      do 90 i=1,numsamp
        write(27,79) (cov(i,j),j=1,numsamp)
 79     format(10(f13.5,1x))
 90   continue
      close(27)

      print *
      write(6,*) 'Correlation matrix written to specified .cov file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'covdata.params.h'.
*
*  With this known, the .sam, .gau.fit, and .cov suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Yao Lin, 3/26/2003; Tim Simpson, 2/15/1998
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

*********************************************************************
*
      subroutine cormat (xmat,cov,numsamp,numdv,theta)
*
*
```

```
*  This subroutine calculates the correlation matrix and its inverse
*
*  Original code developed by:
*  Yao Lin 26 March 2003 /
*        Tim Simpson 15 February 1998 / Tony Giunta, 12 May 1997
*
**********************************************************************
*
* Inputs:
* -------
*   DOUBLE PRECISION:
*   -----------------
*   xmat,theta
*
*   INTEGER:
*   --------
*   numdv,numsamp
*
* Outputs:
* --------
*   DOUBLE PRECISION:
*   -----------------
*   cov - the correlation matrix.
*
*
**********************************************************************
C
C  passed variables
C
      integer numdv,numsamp

      double precision xmat(numsamp,numdv),cov(numsamp,numsamp),
     &   theta(numdv),R
C
C  local variables
C
      integer i,j
C
C  calculate terms in the correlation matrix
C
      do 300 i = 1,numsamp
        do 305 j = i,numsamp
          if( i .eq. j ) then
            cov(i,j) = 1.0d0
          else
C
C  call subroutine to compute spatial correlation function for xmat
C
C  input:  xmat, theta, numdv, numsamp, i, j
C
C  output: R
C
            call scfxmat(R,xmat,theta,numdv,numsamp,i,j)
            cov(i,j) = R
            cov(j,i) = cov(i,j)
          endif
 305    continue
 300  continue
      end

C**********************************************************************
C
      subroutine scfxmat(R,xmat,theta,numdv,numsamp,i,j)
```

```
C
C     Origin: Tim Simpson       Date:  February 11, 1998
C     Modified: Yao Lin         Date:  March 26, 2003
C
C     subroutine to compute spatial correlation function (scf) for
C     correlation matrix; NOT to compute scf for r_xhat.
C
C  Output:
C  -------
C    R = value of correlation function between two sample points,
C          given theta
C
C  Input:
C  ------
C    xmat = matrix of sample points
C    theta = array of theta values
C    i,j = i_th and j_th elements of correlation matrix for which
C           correlation function is being computed
C
C  All variables except R are unchanged upon exiting
C
C*********************************************************************
C
C  passed variables
C
      integer i,j,numdv,numsamp
      double precision R,xmat(numsamp,numdv),theta(numdv)
C
C  local variables
C
      double precision sum,thetadist,dist
      integer k

      sum=0.0d0
      do 120 k = 1,numdv
          dist = ABS(xmat(i,k)-xmat(j,k))
          sum = sum + theta(k)*((dist)**2)
  120      continue
         R = exp( -1.0d0*sum )

        return
        end
```

Covdata.params.h:

```
C*********************************************************
C                                                        *
C  Parameter input file for 'covmat'                  *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for calculating the covariance
C          matrix and its determinant
C

      parameter (numdv=3,numsamp=28,fprefix='suit3valid')

C
```

```
C  numdv = # design variables
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C*********************************************************
```

Suit3valid.sam:

```
      0      0      0
      0      0      1
      0      1      0
      0      1      1
      1      0      0
      1      0      1
      1      1      0
      1      1      1
      0.5    0.5    0.5
      0.0333 0.8556 0.3769
      0.6143 0.4333 0.1167
      0.1276 0.0344 0.7252
      0.9925 0.4751 0.3961
      0.529  0.8567 0.9059
      0.6865 0.4227 0.41
      0.3008 1      0.2559
      0.7573 0      0.7573
      1      0.5    0.5
      0.0111 0.2663 0.3472
      0.5    0.5    1
      0.5    0.5    0
      0.3834 0.9532 0.6156
      0.9998 0.7204 0.1767
      0.0123 0.7001 0.785
      0.0015 0.2976 0.7563
      0.0    1.0     0.7927
      1.0    1.0     0.4309
      0.0    0.5     0.7007
```

Suit3valid.gau.fit:

```
      1      0.46716      11.95818    17.40336      -16.55119
```

Altcov.f:

```
**********************************************************************
*
      program altcov
*
*  This program calculates the alternated correlation matrix, given the
*       initial correlation matrix and predicted prediction errors at
*       possible new data points.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
*
**********************************************************************
*
```

```
* Input files:
* ------------
*  altcov.params.h - parameter file, specifying numdv, numsamp,
*                       errmax, lambda, fprefix, fprefix2, fprefixnew
*  fprefix.cov     - initial correlation matrix
*  fprefix2.out    - predicted prediction errors at possible new data points
*
* Output files:
* -------------
*  fprefixnew.cov  - alternated correlation matrix
*
* Variables:
* ----------
*  inicov     = the initial correlation matrix
*  newcov     = the alternated correlation matrix
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*  numsamp = number of data samples from which the correlation matrix
*              is calculated
*
* Local Variables:
* ----------------
*  DOUBLE PRECISION
*  ----------------
*  errpred = the predicted prediction errors associated with each data
*            and possible new data points
*
************************************************************************

      integer numsamp,numgoal,numdv
      double precision lambda,errmax1,errmax2,gamma
      double precision TargetH1,TargetL1,TargetS1,
     &       TargetH2,TargetL2,TargetS2,
     &       TargetH3,TargetL3,TargetS3
      double precision ymax1,ymin1,
     &       ymax2,ymin2,ymax3,ymin3,
     &       yconstant1,yconstant2,yconstant3
      character TargetType1,TargetType2,TargetType3
      character*20 fprefix,fprefix2,fprefixnew
      character*20 fprefix3,fprefix4,fprefix5,fprefix6
C
C  include parameter settings for numdv,numsamp,fprefix,fprefix2,fprefixnew,
C  errmax, lambda, e.g., in the one-variable problem, for the first step:
C  numdv=1,numsamp=8,fprefix='step1newp',fprefix2='errpred1',
C  fprefixnew='step1altnewp',errmax=0.50,lambda=2.0
C
      include 'altcov.params.h'

      double precision inicov(numsamp,numsamp),
     &       newcov(numsamp,numsamp),
     &       errpred(numresp,numsamp),
     &       goalachieve(numgoal,numsamp),
     &       responsey(numgoal,numsamp),
     &       alpha(2),eta(2),
     &       response,goalachievement,
     &       errmax(numresp),TargetH(numgoal),
     &       TargetL(numgoal),TargetS(numgoal),
     &       ymax(numgoal),ymin(numgoal),
     &       yconstant(numgoal)
      character TargetType(numgoal)
      integer i,j,k,lenstr
      character*20 ftitle
```

```
      character*20 deckfile,deckfile2,deckfile3,outfile
      character*20 deckfile4,deckfile5,deckfile6

      errmax(1)=errmax1
      errmax(2)=errmax2
      TargetH(1)=TargetH1
      TargetL(1)=TargetL1
      TargetS(1)=TargetS1
      TargetH(2)=TargetH2
      TargetL(2)=TargetL2
      TargetS(2)=TargetS2
      TargetH(3)=TargetH3
      TargetL(3)=TargetL3
      TargetS(3)=TargetS3
      ymax(1)=ymax1
      ymin(1)=ymin1
      ymax(2)=ymax2
      ymin(2)=ymin2
      ymax(3)=ymax3
      ymin(3)=ymin3
      yconstant(1)=yconstant1
      yconstant(2)=yconstant2
      yconstant(3)=yconstant3
      TargetType(1)=TargetType1
      TargetType(2)=TargetType2
      TargetType(3)=TargetType3

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefix2,lenstr)
       ftitle=fprefix2
       deckfile2=ftitle(1:lenstr) // '.out'

       call getlen(fprefix3,lenstr)
       ftitle=fprefix3
       deckfile3=ftitle(1:lenstr) // '.out'

       call getlen(fprefix4,lenstr)
       ftitle=fprefix4
       deckfile4=ftitle(1:lenstr) // '.out'

       call getlen(fprefix5,lenstr)
       ftitle=fprefix5
       deckfile5=ftitle(1:lenstr) // '.out'

       deckfile6=fprefix6

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.cov'


       open(21,file=deckfile,status='old')
       open(23,file=deckfile2,status='old')
       open(25,file=deckfile3,status='old')
       open(28,file=deckfile4,status='old')
       open(29,file=deckfile5,status='old')
```

```
       open(30,file=deckfile6,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,deckfile2,deckfile3,
     &       deckfile4,deckfile5,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (inicov(i,j),j=1,numsamp)
      close(21)

C
C  initialize errpred
C
      print *
      write(6,*) 'Reading in and calculating errpred...'
      do 15 j=1,numresp
            do 20 i=1,numsamp
                if (i.le.numold) then
                    errpred(j,i)=0.0
                else
                    if (j.eq.1) then
                       read(23,*) errpred(j,i)
                    else
                       read(25,*) errpred(j,i)
                    endif
                endif
                if (abs(errpred(j,i)).gt.(errmax(j))) then
                    errpred(j,i)=errmax(j)
                endif
 20   continue
 15   continue
      close(23)
      close(25)

      print *
      write(6,*)
     &  'Reading in responses and calculating goal.achievement...'
      do 55 j=1,numgoal
         do 60 i=1,numsamp
            if (j.eq.1) then
               read (28,*) responsey(j,i)
            elseif (j.eq.2) then
               read (29,*) responsey(j,i)
            else
               read (30,*) responsey(j,i)
            endif
            response=responsey(j,i)+yconstant(j)
            if (TargetType(j).eq.'H') then
               call Hgoalachievecal(goalachievement,TargetH(j),
     &              response,ymax(j),ymin(j),gamma)
               goalachieve(j,i)=goalachievement
            else if (TargetType(j).eq.'L') then
               call Lgoalachievecal(goalachievement,TargetL(j),
     &              response,ymax(j),ymin(j),gamma)
               goalachieve(j,i)=goalachievement
            else if (TargetType(j).eq.'S') then
               call Sgoalachievecal(goalachievement,TargetS(j),
```

```
     &              response,ymax(j),ymin(j),gamma)
                goalachieve(j,i)=goalachievement
             endif
 60   continue
 55   continue
      close(28)
      close(29)
      close(30)

C
C  calculate the alternated correlation matrix
C
       do 30 i=1,numsamp
          do 40 j=i,numsamp
          if (i.eq.j) then
             newcov(i,j)=1.0
          elseif (((i.le.numold).AND.(j.le.numold)).OR.
     &             ((i.gt.numold).AND.(j.gt.numold))) then
          newcov(i,j)=inicov(i,j)
          newcov(j,i)=newcov(i,j)
          elseif (((i.le.numold).AND.(j.gt.numold)).OR.
     &             ((i.gt.numold).AND.(j.le.numold))) then
          if (inicov(i,j).eq.1) then
          newcov(i,j)=inicov(i,j)
          newcov(j,i)=newcov(i,j)
          elseif (inicov(i,j).lt.1) then
             alpha(i)=0
             alpha(j)=0
             do 50 k=1,numresp
                alpha(i)=alpha(i)+abs(errpred(k,i))/lambda/
     &             errmax(k)/numresp
                alpha(j)=alpha(j)+abs(errpred(k,j))/lambda/
     &             errmax(k)/numresp
 50   continue
             alpha(i)=1-alpha(i)
             alpha(j)=1-alpha(j)
             eta(i)=0
             eta(j)=0
             do 65 k=1,numgoal
                eta(i)=eta(i)+goalachieve(k,i)/numgoal
                eta(j)=eta(j)+goalachieve(k,j)/numgoal
 65   continue
             eta(i)=1-eta(i)
             eta(j)=1-eta(j)
          newcov(i,j)=inicov(i,j)
     &   *alpha(i)*alpha(j)*eta(i)*eta(j)
          newcov(j,i)=newcov(i,j)
          endif
          endif
 40   continue
 30   continue

C
C  write alternated correlation matrix into specified .cov file
C
       do 80 i=1,numsamp
         write(27,79) (newcov(i,j),j=1,numsamp)
 79     format(30(f13.5,1x))
 80   continue
      close(27)

      print *
      write(6,*) 'Alternated correlation matrix written to .cov file'
```

```
      stop
      end

*****************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      character*1 blank
      character*20 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

*****************************************************************
*
      subroutine Hgoalachievecal(goalachievement,TargetH,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      double precision goalachievement,TargetH,response
      double precision ymax,ymin,gamma
```

```
      if (response.le.ymin) then
         goalachievement=0.00000000
      else if (response.ge.min(TargetH,ymax)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(response-ymin)/
     &    (min(TargetH,ymax)-ymin)/gamma
      endif

      return
      end


*****************************************************************
*
      subroutine Lgoalachievecal(goalachievement,TargetL,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      double precision goalachievement,TargetL,response
      double precision ymax,ymin,gamma

      if (response.ge.ymax) then
         goalachievement=0.0000000000
      else if (response.le.max(TargetL,ymin)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(ymax-response)/
     &    (ymax-max(ymin,TargetL))/gamma
      endif

      return
      end

*****************************************************************
*
      subroutine Sgoalachievecal(goalachievement,TargetS,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
```

```
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      double precision goalachievement,TargetS,response
      double precision ymax,ymin,gamma

      if (response.ge.ymax) then
         goalachievement=0.00000000
      else if (response.le.ymin) then
         goalachievement=0.00000000
      else if (response.eq.TargetS) then
         goalachievement=1.0/gamma
      else if (response<TargetS.AND.response>ymin) then
          goalachievement=(response-ymin)/(TargetS-ymin)/gamma
      else if (response>TargetS.AND.response<ymax) then
          goalachievement=(response-TargetS)/(ymax-TargetS)/gamma
      endif

      return
      end
```

Altcov.params.h:

```
C*********************************************************
C                                                        *
C  Parameter input file for 'altcov'                     *
C    Author: Yao Lin                                     *
C      Date: 3/26/2003                                   *
C                                                        *
C*********************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=3,numsamp=20,numold=18,
     &          numgoal=3,numresp=2,
     &          fprefix='suit3valid',
     &          fprefix2='Qit3st3err.gau',
     &          fprefix3='Jit3st3err.gau',
     &          fprefix4='Qit3val.gau',
     &          fprefix5='Jit3val.gau',
     &          fprefix6='repmoutput1.out',
     &          fprefixnew='suit3altvalid',
     &          errmax1=0.35,
     &          errmax2=0.00268,
     &          lambda=2.0,
     &          ymax1=-6.9,ymin1=-16.0,
     &          TargetL1=-20.0,
     &          TargetH1=-1.0,TargetS1=-1.0,
     &          ymax2=0.01164,ymin2=0.00056,
     &          TargetL2=0.0015,
     &          TargetH2=-1.0,TargetS2=-1.0,
     &          ymax3=0.00033,ymin3=0.00005,
     &          TargetL3=0.00025,
     &          TargetH3=-1.0,TargetS3=-1.0,
     &          TargetType1='L',
     &          TargetType2='L',
     &          TargetType3='L',
     &          yconstant1=0.0,
```

```
     &          yconstant2=0.0,
     &          yconstant3=0.0,
     &          gamma=1.25)

C
C  numdv = # design variables
C  numsamp = # samples in data set
C  numold = # old data points in the data set
C
C  fprefix = prefix of titles of file that stores the initial
C            correlation matrix for both old and possible new
C            data points
C
C  fprefix2 = prefix of titles of file that stores the
C             predicted prediction errors at possible new
C             data points
C
C  fprefix3 = prefix of titles of file that stores the
C             predicted response values at all points
C
C
C  fprefixnew = prefix of titles of file that stores the
C               alternated correlation matrix for both old and
C               possible new data points, with prediction errors
C               at these points considered
C
C  errmax = maximum value of the absolute predicted prediction error
C
C  lambda = coefficient used to gauge the adjustment to initial
C           correlation matrix
C*********************************************************************
```

Detcov.f:

```
C*********************************************************************
C
      program detcov
C
C  This program calculates the determinant given a matrix.  Particularly,
C       in SEED, it is used to calculate the determinant of the
C       correlation matrix.
C
C  Updated by: Yao Lin, March 26, 2003
C
C  Original code developed by:
C  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
C
C*********************************************************************
C
C Input files:
C ------------
C  detcov.params.h - parameter file, specifying numdv, numsamp,
C                       coedet, fprefix
C  .cov             - correlation matrix
C
C Output files:
C -------------
C  .det             - determinant of the correlation matrix
C
```

```
C Variables:
C ----------
C   cov     = the input correlation matrix for which we calculate
C             determinant
C
C Parameter Variables (to be specified by user in dace.params.h):
C ---------------------------------------------------
C   numsamp = number of data samples from which the correlation matrix
C               is calculated
C
C Local Variables:
C ----------------
C   DOUBLE PRECISION
C   ----------------
C   work     = vector of length 'numsamp' used as temporary storage
C   invmat   = inverse of the correlation matrix (numsamp x numsamp)
C
C   INTEGER
C   -------
C   ipvt    = vector of length 'numsamp' of pivot locations
C
C**********************************************************************

      integer numsamp
      double precision coedet
      character*16 fprefix
C
C  include parameter settings for numdv,numsamp,fprefix
C
      include 'detcov.params.h'

C**********************************************************************
C
C  include LINPACK routines used to find determinant of correlation matrix
C
C**********************************************************************

C      include 'dgefa.f'
C      include 'dgedi.f'

C**********************************************************************

      double precision cov(numsamp,numsamp),work(numsamp),
     &       dummy2,detR,det(2),rcond,z(numsamp)
      integer i,j,ipvt(numsamp),dummy,lenstr,info
      character*16 ftitle
      character*20 deckfile,outfile
      err=0.0000
C
C  open necessary .cov and .det files based on 'fprefix' name,
C  e.g., step1.cov, step1.det
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix

       deckfile=ftitle(1:lenstr) // '.cov'
       outfile=ftitle(1:lenstr) // '.det'

       open(21,file=deckfile,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,outfile
```

```
       print *, numsamp
C
C  initialize cov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (cov(i,j),j=1,numsamp)
      close(21)

C
C     Start to calculate the determinant of the correlation matrix;
C        initialization.
C
      do 307 i=1,numsamp
        work(i)=0.0d0
        ipvt(i)=0
 307  continue

C
C     If there is any error in the calculation in DGEFA (singular matrix),
C        this program will set the determinant to 0.
C
      call dgeco(cov,numsamp,numsamp,ipvt,rcond,z)
      if( rcond .eq. 0 ) then
          write(27,78) err
 78   format(10(f13.5,1x))
          close(27)
          go to 1000
      endif
C
C In DGEDI, last flag is: 1 (inverse only), 10 (Det only), 11 (both)
C
      call dgedi(cov, numsamp, numsamp, ipvt, det, work, 10)
      detR=det(1)*10.0d0**det(2)
      detR=coedet*detR

C
C  write predicted values to specified .det file
C
      write(27,79) detR
 79   format(10(f13.5,1x))
      close(27)

      print *
      write(6,*) detR
1000  write(6,*) 'Coefficient*Determinant written to .det file'

      stop
      end

*********************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
```

```
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
**********************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

      subroutine dgeco(a,lda,n,ipvt,rcond,z)
      integer lda,n,ipvt(1)
      double precision a(lda,1),z(1)
      double precision rcond
c
c     dgeco factors a double precision matrix by gaussian elimination
c     and estimates the condition of the matrix.
c
c     if  rcond  is not needed, dgefa is slightly faster.
c     to solve  a*x = b , follow dgeco by dgesl.
c     to compute  inverse(a)*c , follow dgeco by dgesl.
c     to compute  determinant(a) , follow dgeco by dgedi.
c     to compute  inverse(a) , follow dgeco by dgedi.
c
c     on entry
c
c        a       double precision(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        rcond   double precision
c                an estimate of the reciprocal condition of  a .
```

```
c                for the system  a*x = b , relative perturbations
c                in  a  and  b  of size  epsilon  may cause
c                relative perturbations in  x  of size  epsilon/rcond .
c                if  rcond  is so small that the logical expression
c                           1.0 + rcond .eq. 1.0
c                is true, then  a  may be singular to working
c                precision.  in particular,  rcond  is zero  if
c                exact singularity is detected or the estimate
c                underflows.
c
c        z       double precision(n)
c                a work vector whose contents are usually unimportant.
c                if  a  is close to a singular matrix, then  z  is
c                an approximate null vector in the sense that
c                norm(a*z) = rcond*norm(a)*norm(z) .
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     linpack dgefa
c     blas daxpy,ddot,dscal,dasum
c     fortran dabs,dmax1,dsign
c
c     internal variables
c
      double precision ddot,ek,t,wk,wkm
      double precision anorm,s,dasum,sm,ynorm
      integer info,j,k,kb,kp1,l
c
c
c     compute 1-norm of a
c
      anorm = 0.0d0
      do 10 j = 1, n
         anorm = dmax1(anorm,dasum(n,a(1,j),1))
   10 continue
c
c     factor
c
      call dgefa(a,lda,n,ipvt,info)
c
c     rcond = 1/(norm(a)*(estimate of norm(inverse(a)))) .
c     estimate = norm(z)/norm(y) where  a*z = y  and  trans(a)*y = e .
c     trans(a)  is the transpose of a .  the components of  e  are
c     chosen to cause maximum local growth in the elements of w  where
c     trans(u)*w = e .  the vectors are frequently rescaled to avoid
c     overflow.
c
c     solve trans(u)*w = e
c
      ek = 1.0d0
      do 20 j = 1, n
         z(j) = 0.0d0
   20 continue
      do 100 k = 1, n
         if (z(k) .ne. 0.0d0) ek = dsign(ek,-z(k))
         if (dabs(ek-z(k)) .le. dabs(a(k,k))) go to 30
            s = dabs(a(k,k))/dabs(ek-z(k))
            call dscal(n,s,z,1)
            ek = s*ek
   30    continue
```

```
         wk = ek - z(k)
         wkm = -ek - z(k)
         s = dabs(wk)
         sm = dabs(wkm)
         if (a(k,k) .eq. 0.0d0) go to 40
            wk = wk/a(k,k)
            wkm = wkm/a(k,k)
         go to 50
   40    continue
            wk = 1.0d0
            wkm = 1.0d0
   50    continue
         kp1 = k + 1
         if (kp1 .gt. n) go to 90
            do 60 j = kp1, n
               sm = sm + dabs(z(j)+wkm*a(k,j))
               z(j) = z(j) + wk*a(k,j)
               s = s + dabs(z(j))
   60       continue
            if (s .ge. sm) go to 80
               t = wkm - wk
               wk = wkm
               do 70 j = kp1, n
                  z(j) = z(j) + t*a(k,j)
   70          continue
   80       continue
   90    continue
         z(k) = wk
  100 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
c
c     solve trans(l)*y = w
c
      do 120 kb = 1, n
         k = n + 1 - kb
         if (k .lt. n) z(k) = z(k) + ddot(n-k,a(k+1,k),1,z(k+1),1)
         if (dabs(z(k)) .le. 1.0d0) go to 110
            s = 1.0d0/dabs(z(k))
            call dscal(n,s,z,1)
  110    continue
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
  120 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
c
      ynorm = 1.0d0
c
c     solve l*v = y
c
      do 140 k = 1, n
         l = ipvt(k)
         t = z(l)
         z(l) = z(k)
         z(k) = t
         if (k .lt. n) call daxpy(n-k,t,a(k+1,k),1,z(k+1),1)
         if (dabs(z(k)) .le. 1.0d0) go to 130
            s = 1.0d0/dabs(z(k))
            call dscal(n,s,z,1)
            ynorm = s*ynorm
```

```
  130    continue
  140 continue
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
      ynorm = s*ynorm
c
c     solve  u*z = v
c
      do 160 kb = 1, n
         k = n + 1 - kb
         if (dabs(z(k)) .le. dabs(a(k,k))) go to 150
            s = dabs(a(k,k))/dabs(z(k))
            call dscal(n,s,z,1)
            ynorm = s*ynorm
  150    continue
         if (a(k,k) .ne. 0.0d0) z(k) = z(k)/a(k,k)
         if (a(k,k) .eq. 0.0d0) z(k) = 1.0d0
         t = -z(k)
         call daxpy(k-1,t,a(1,k),1,z(1),1)
  160 continue
c     make znorm = 1.0
      s = 1.0d0/dasum(n,z,1)
      call dscal(n,s,z,1)
      ynorm = s*ynorm
c
      if (anorm .ne. 0.0d0) rcond = ynorm/anorm
      if (anorm .eq. 0.0d0) rcond = 0.0d0
      return
      end


      subroutine dgedi(a,lda,n,ipvt,det,work,job)
      integer lda,n,ipvt(1),job
      double precision a(lda,1),det(2),work(1)
C
C     dgedi computes the determinant and inverse of a matrix
C     using the factors computed by dgeco or dgefa.
C
C     on entry
C
C        a       double precision(lda, n)
C                the output from dgeco or dgefa.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
C
C        ipvt    integer(n)
C                the pivot vector from dgeco or dgefa.
C
C        work    double precision(n)
C                work vector.  contents destroyed.
C
C        job     integer
C                = 11   both determinant and inverse.
C                = 01   inverse only.
C                = 10   determinant only.
C
C     on return
C
C        a       inverse of original matrix if requested.
```

```
C                otherwise unchanged.
C
C        det     double precision(2)
C                determinant of original matrix if requested.
C                otherwise not referenced.
C                determinant = det(1) * 10.0**det(2)
C                with  1.0 .le. dabs(det(1)) .lt. 10.0
C                or  det(1) .eq. 0.0 .
C
C     error condition
C
C        a division by zero will occur if the input factor contains
C        a zero on the diagonal and the inverse is requested.
C        it will not occur if the subroutines are called correctly
C        and if dgeco has set rcond .gt. 0.0 or dgefa has set
C        info .eq. 0 .
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,dswap
C     fortran dabs,mod
C
C     internal variables
C
      double precision t
      double precision ten
      integer i,j,k,kb,kp1,l,nm1
C
C
C     compute determinant
C
      if (job/10 .eq. 0) go to 70
         det(1) = 1.0d0
         det(2) = 0.0d0
         ten = 10.0d0
         do 50 i = 1, n
            if (ipvt(i) .ne. i) det(1) = -det(1)
            det(1) = a(i,i)*det(1)
C           ...exit
            if (det(1) .eq. 0.0d0) go to 60
   10       if (dabs(det(1)) .ge. 1.0d0) go to 20
               det(1) = ten*det(1)
               det(2) = det(2) - 1.0d0
            go to 10
   20       continue
   30       if (dabs(det(1)) .lt. ten) go to 40
               det(1) = det(1)/ten
               det(2) = det(2) + 1.0d0
            go to 30
   40       continue
   50    continue
   60    continue
   70 continue
C
C     compute inverse(u)
C
      if (mod(job,10) .eq. 0) go to 150
         do 100 k = 1, n
            a(k,k) = 1.0d0/a(k,k)
            t = -a(k,k)
```

```
            call dscal(k-1,t,a(1,k),1)
            kp1 = k + 1
            if (n .lt. kp1) go to 90
            do 80 j = kp1, n
               t = a(k,j)
               a(k,j) = 0.0d0
               call daxpy(k,t,a(1,k),1,a(1,j),1)
   80       continue
   90       continue
  100    continue
C
C        form inverse(u)*inverse(l)
C
         nm1 = n - 1
         if (nm1 .lt. 1) go to 140
         do 130 kb = 1, nm1
            k = n - kb
            kp1 = k + 1
            do 110 i = kp1, n
               work(i) = a(i,k)
               a(i,k) = 0.0d0
  110       continue
            do 120 j = kp1, n
               t = work(j)
               call daxpy(n,t,a(1,j),1,a(1,k),1)
  120       continue
            l = ipvt(k)
            if (l .ne. k) call dswap(n,a(1,k),1,a(1,l),1)
  130    continue
  140    continue
  150 continue
      return
      end

      subroutine daxpy(n,da,dx,incx,dy,incy)
C
C     constant times a vector plus a vector.
C     uses unrolled loops for increments equal to one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),da
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if (da .eq. 0.0d0) return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C        code for unequal increments or equal increments
C          not equal to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
C
C        code for both increments equal to 1
```

```
C
C
C        clean-up loop
C
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end

      subroutine  dscal(n,da,dx,incx)
C
C     scales a vector by a constant.
C     uses unrolled loops for increment equal to one.
C     jack dongarra, linpack, 3/11/78.
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision da,dx(*)
      integer i,incx,m,mp1,n,nincx
C
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dx(i) = da*dx(i)
   10 continue
      return
C
C        code for increment equal to 1
C
C
C        clean-up loop
C
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end
```

```
      subroutine  dswap (n,dx,incx,dy,incy)
C
C     interchanges two vectors.
C     uses unrolled loops for increments equal one.
C     jack dongarra, linpack, 3/11/78.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
C
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
C
C       code for unequal increments or equal increments not equal
C         to 1
C
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dx(ix)
        dx(ix) = dy(iy)
        dy(iy) = dtemp
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
C
C       code for both increments equal to 1
C
C
C       clean-up loop
C
   20 m = mod(n,3)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
   30 continue
      if( n .lt. 3 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,3
        dtemp = dx(i)
        dx(i) = dy(i)
        dy(i) = dtemp
        dtemp = dx(i + 1)
        dx(i + 1) = dy(i + 1)
        dy(i + 1) = dtemp
        dtemp = dx(i + 2)
        dx(i + 2) = dy(i + 2)
        dy(i + 2) = dtemp
   50 continue
      return
      end


      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)
C
```

```
C     dgefa factors a double precision matrix by gaussian elimination.
C
C     dgefa is usually called by dgeco, but it can be called
C     directly with a saving in time if  rcond  is not needed.
C     (time for dgeco) = (1 + 9/n)*(time for dgefa) .
C
C     on entry
C
C        a       double precision(lda, n)
C                the matrix to be factored.
C
C        lda     integer
C                the leading dimension of the array  a .
C
C        n       integer
C                the order of the matrix  a .
C
C     on return
C
C        a       an upper triangular matrix and the multipliers
C                which were used to obtain it.
C                the factorization can be written  a = l*u  where
C                l  is a product of permutation and unit lower
C                triangular matrices and  u  is upper triangular.
C
C        ipvt    integer(n)
C                an integer vector of pivot indices.
C
C        info    integer
C                = 0  normal value.
C                = k  if  u(k,k) .eq. 0.0 .  this is not an error
C                     condition for this subroutine, but it does
C                     indicate that dgesl or dgedi will divide by zero
C                     if called.  use  rcond  in dgeco for a reliable
C                     indication of singularity.
C
C     linpack. this version dated 08/14/78 .
C     cleve moler, university of new mexico, argonne national lab.
C
C     subroutines and functions
C
C     blas daxpy,dscal,idamax
C
C     internal variables
C
      double precision t
      integer idamax,j,k,kp1,l,nm1
C
C
C     gaussian elimination with partial pivoting
C
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
C
C        find l = pivot index
C
         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
C
C        zero pivot implies this column already triangularized
```

```
C
         if (a(l,k) .eq. 0.0d0) go to 40
C
C           interchange if necessary
C
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
C
C           compute multipliers
C
            t = -1.0d0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
C
C           row elimination with column indexing
C
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d0) info = n
      return
      end

      integer function idamax(n,dx,incx)
C
C     finds the index of element having max. absolute value.
C     jack dongarra, linpack, 3/11/78.
C     modified 3/93 to return if incx .le. 0.
C     modified 12/3/93, array(1) declarations changed to array(*)
C
      double precision dx(*),dmax
      integer i,incx,ix,n
C
      idamax = 0
      if( n.lt.1 .or. incx.le.0 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
C
C        code for increment not equal to 1
C
      ix = 1
      dmax = dabs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(dabs(dx(ix)).le.dmax) go to 5
         idamax = i
         dmax = dabs(dx(ix))
    5    ix = ix + incx
```

```
   10 continue
      return
C
C        code for increment equal to 1
C
   20 dmax = dabs(dx(1))
      do 30 i = 2,n
         if(dabs(dx(i)).le.dmax) go to 30
         idamax = i
         dmax = dabs(dx(i))
   30 continue
      return
      end

      double precision function dasum(n,dx,incx)
c
c     takes the sum of the absolute values.
c     jack dongarra, linpack, 3/11/78.
c     modified 3/93 to return if incx .le. 0.
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dtemp
      integer i,incx,m,mp1,n,nincx
c
      dasum = 0.0d0
      dtemp = 0.0d0
      if( n.le.0 .or. incx.le.0 )return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dtemp = dtemp + dabs(dx(i))
   10 continue
      dasum = dtemp
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,6)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dabs(dx(i))
   30 continue
      if( n .lt. 6 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,6
        dtemp = dtemp + dabs(dx(i)) + dabs(dx(i + 1)) + dabs(dx(i + 2))
     &  + dabs(dx(i + 3)) + dabs(dx(i + 4)) + dabs(dx(i + 5))
   50 continue
   60 dasum = dtemp
      return
      end

      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
```

```
c     modified 12/3/93, array(1) declarations changed to array(*)
c
      double precision dx(*),dy(*),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     & dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
```

Detcov.params.h:

```
C*************************************************************
C                                                            *
C  Parameter input file for 'detcov'                         *
C    Author: Yao Lin                                         *
C      Date: 3/26/2003                                       *
C                                                            *
C*************************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=3,numsamp=27,fprefix='suit3altvalid',
     &          coedet=1e8 )
C
C  numdv = # design variables
```

```
C  numsamp = # samples in data set
C
C  fprefix = prefix of titles of files to opened/used
C
C  coedet = when the value of determinant is very small,
C           this coefficient is used to magnify the value.
C*********************************************************
```

D.2.3 Implementation of SEED in iSIGHT in Section 7.4

Figures presented in this section illustrate how the SEED method is implemented in iSIGHT. The organization of tasks in Iteration I – Step 3 is shown in Figure D.19. The organization of tasks in Iteration I – Step 7 is shown in Figure D.20.

In Iteration I – Step 3, since the covariance matrix is not adjusted, there are only two simulation codes used in iSIGHT, Covmat and Detcov. In Iteration I – Step 7, with information from metamodels of prediction errors, we use five simulation codes in iSIGHT, Covmat, Qerr, Jerr, Altcov, and Detcov. Covmat is used to formulate the covariance matrix, Qerr and Jerr are metamodels to predict prediction errors, Altcov is

used to adjust entries of the covariance matrix, and Detcov is used to calculate the determinant.

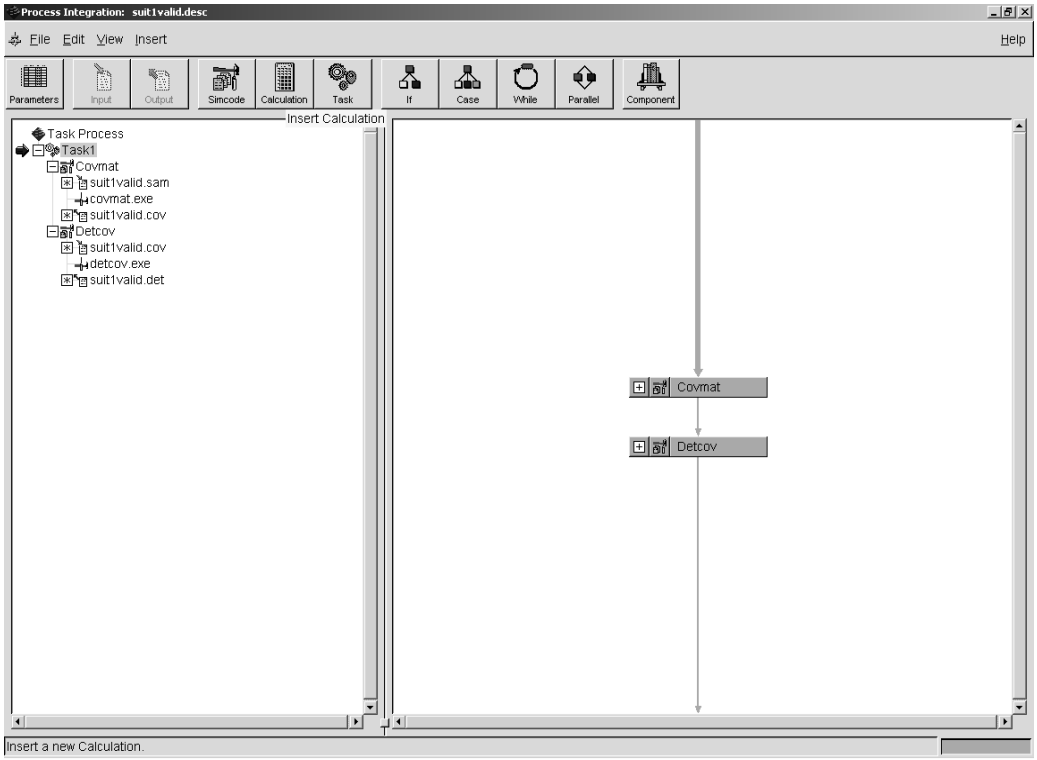


Figure D.19 Implementation of SEED in iSIGHT – Iteration I, Step 3

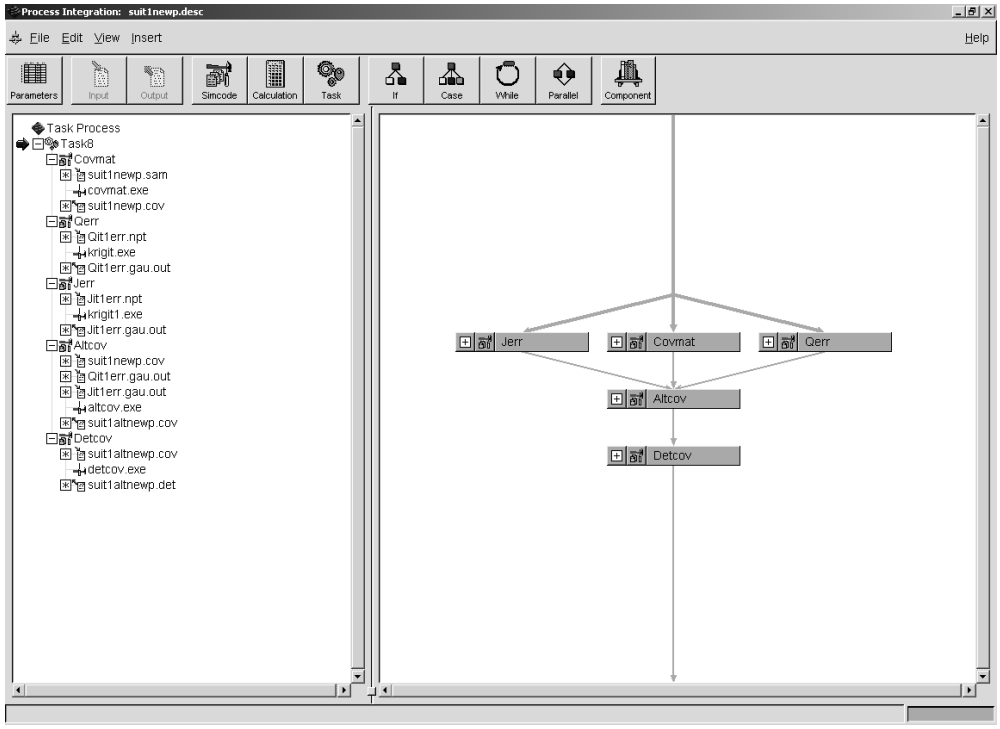


Figure D.20 Implementation of SEED in iSIGHT – Iteration I, Step 7

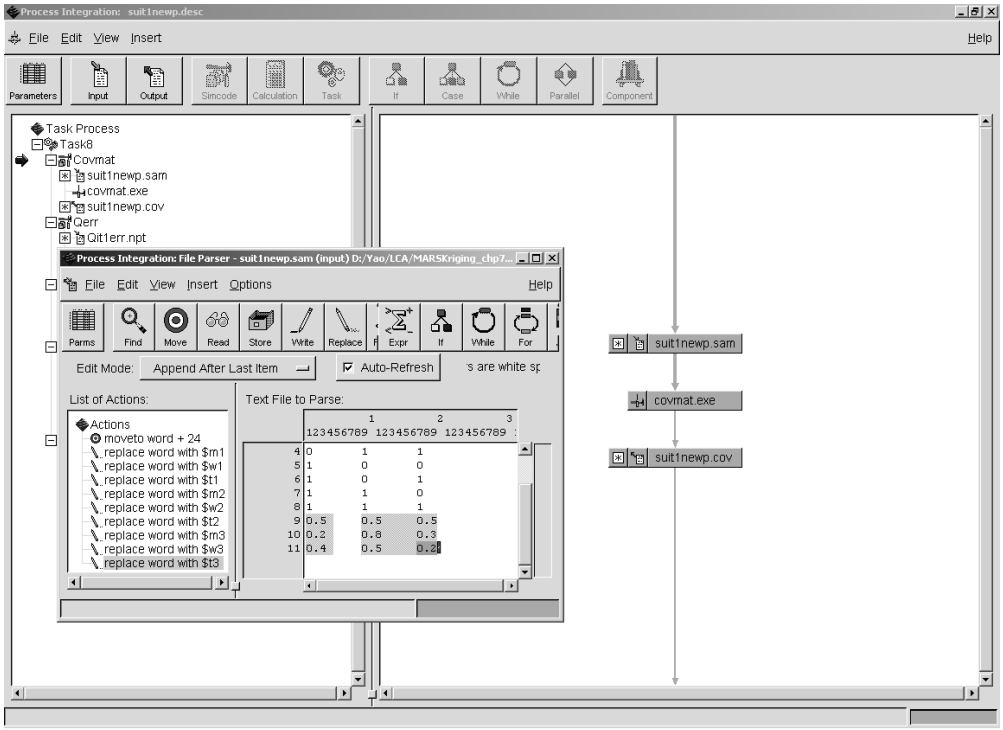


Figure D.21 File Parsing of Input in iSIGHT – Iteration I, Step 7

D.2.4 Twenty Eight Points Identified with SEED

Listed below are 28 points identified with SEED in Section 7.4.

Table D.3 Twenty Eight Points Identified with SEED

Mdot **(kg/s)**	***W*** **(m)**	***t*** **(m)**	***Mdot_n***	***W_n***	***t_n***	***Q*** **(W)**	***J*** **(m/N)**
0.0005	0.0150	0.0002	0	0	0	-11.01	0.00749
0.0005	0.0150	0.0008	0	0	1	-14.37	0.00022
0.0005	0.0350	0.0002	0	1	0	-6.65	0.01167
0.0005	0.0350	0.0008	0	1	1	-9.56	0.00027
0.003	0.0150	0.0002	1	0	0	-42.24	0.00749
0.003	0.0150	0.0008	1	0	1	-109.66	0.00022
0.003	0.0350	0.0002	1	1	0	-19.86	0.01167
0.003	0.0350	0.0008	1	1	1	-23.03	0.00027
0.00175	0.0250	0.0005	0.5	0.5	0.5	-17.49	0.00076
0.00058	0.0321	0.00043	0.0333	0.8556	0.3769	-9.58	0.00126
0.00204	0.0237	0.00027	0.6143	0.4333	0.1167	-16.69	0.00405
0.00082	0.0157	0.00064	0.1276	0.0344	0.7252	-14.65	0.00036
0.00298	0.0245	0.00044	0.9925	0.4751	0.3961	-18.58	0.00106
0.00182	0.0321	0.00074	0.529	0.8567	0.9059	-19.76	0.00031
0.00222	0.0235	0.00045	0.6865	0.4227	0.41	-17.64	0.00098
0.00125	0.0350	0.00035	0.3008	1	0.2559	-14.85	0.00232
0.00239	0.0150	0.00065	0.7573	0	0.7573	-70.69	0.00034
0.003	0.0250	0.0005	1	0.5	0.5	-18.94	0.00076
0.00053	0.0203	0.00041	0.0111	0.2663	0.3472	-11.79	0.00118
0.00175	0.0250	0.0008	0.5	0.5	1	-18.53	0.00025
0.00175	0.0250	0.0002	0.5	0.5	0	-15.92	0.0099
0.00146	0.0341	0.00057	0.3834	0.9532	0.6156	-17.53	0.00061
0.003	0.0294	0.00031	0.9998	0.7204	0.1767	-19.52	0.00314
0.00053	0.0290	0.00067	0.0123	0.7001	0.785	-11.22	0.00039
0.0005	0.0210	0.00065	0.0015	0.2976	0.7563	-12.91	0.00037
0.0005	0.035	0.00068	0	1	0.7935	-8.92	0.0004
0.003	0.035	0.00046	1	1	0.4309	-21.65	0.00109
0.0005	0.025	0.00062	0	0.5	0.7	-11.68	0.00044

D.3 EXPLORATION OF DESIGN SOLUTIONS WITH THE INTEGRATED DESIGN PROCESS IN E-RCEM

All supporting materials and documents for studies in Section 7.5 are presented here. Contours plots of metamodels of responses (initial metamodels, metamodels of responses in Iteration I – Step 8 and Iteration II – Step 3) are illustrated in Section D.3.1. FORTRAN codes of the integrated design process in E-RCEM are presented in Section D.3.2. The implementation of E-RCEM in iSIGHT is illustrated in Section D.3.3. Twenty points identified from E-RCEM and their corresponding response values are listed in Section D.3.4.

D.3.1 Contour Plots of Metamodels of Responses

Contour plots illustrated below are drawn with predicted values from the kriging metamodels of responses in Section 7.5. Contour plots of metamodels of prediction errors are not drawn. The contour plots are drawn using Minitab® with default parameters.

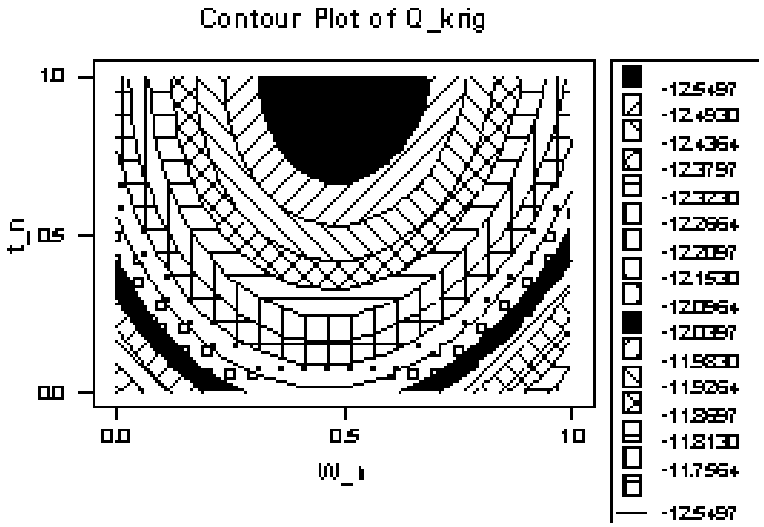


Figure D.22 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Device Width (Initial Kriging Metamodel with 6 Data Points)

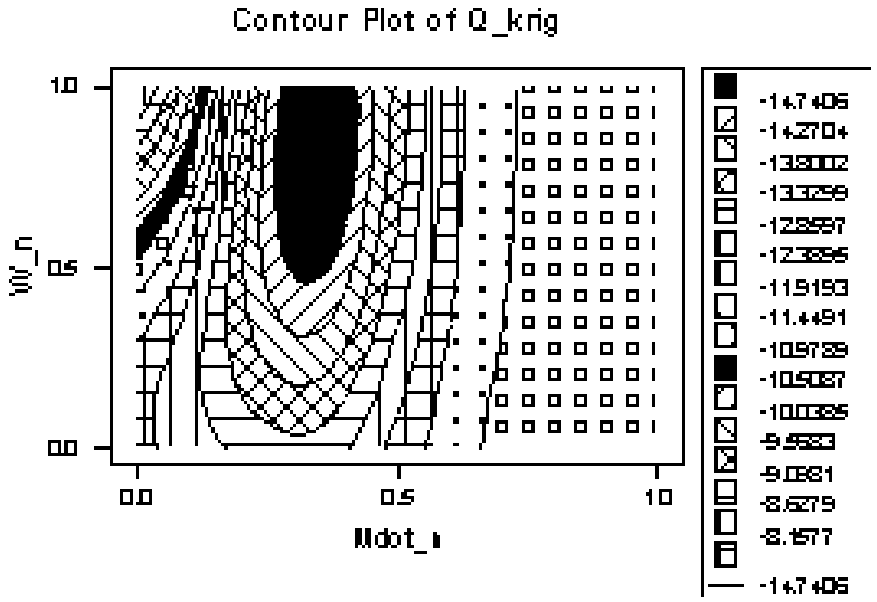


Figure D.23 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Initial Kriging Metamodel with 6 Data Points)

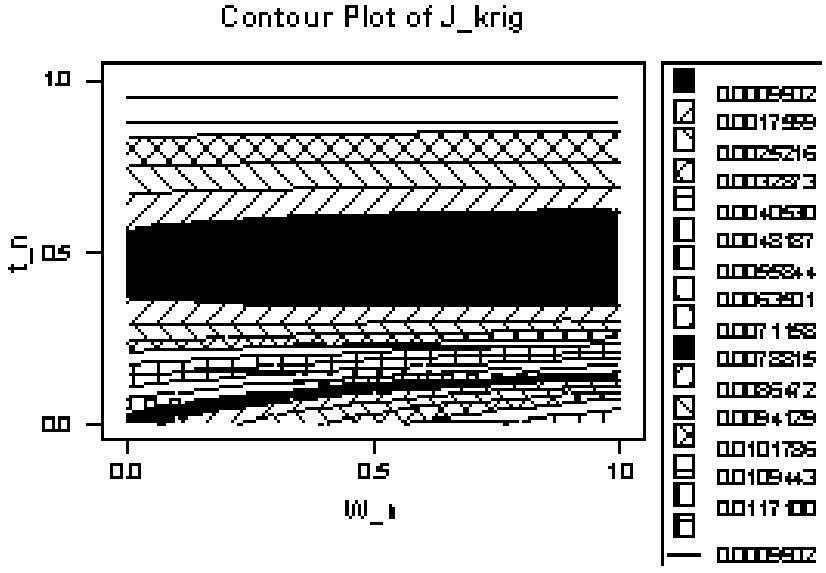


Figure D.24 Contour Plot of Compliance vs. Device Width and Wall Thickness (Initial Kriging Metamodel with 6 Data Points)

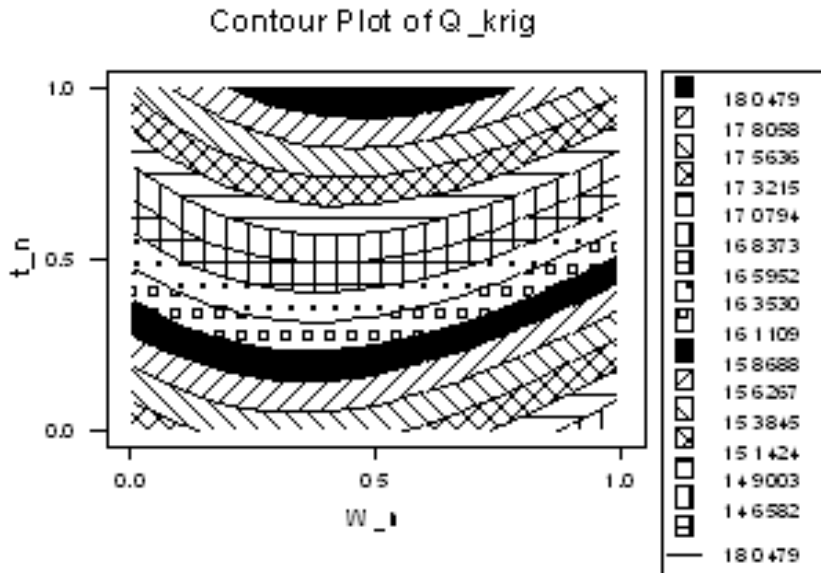


Figure D.25 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Device Width (Kriging Metamodel with 8 Data Points)

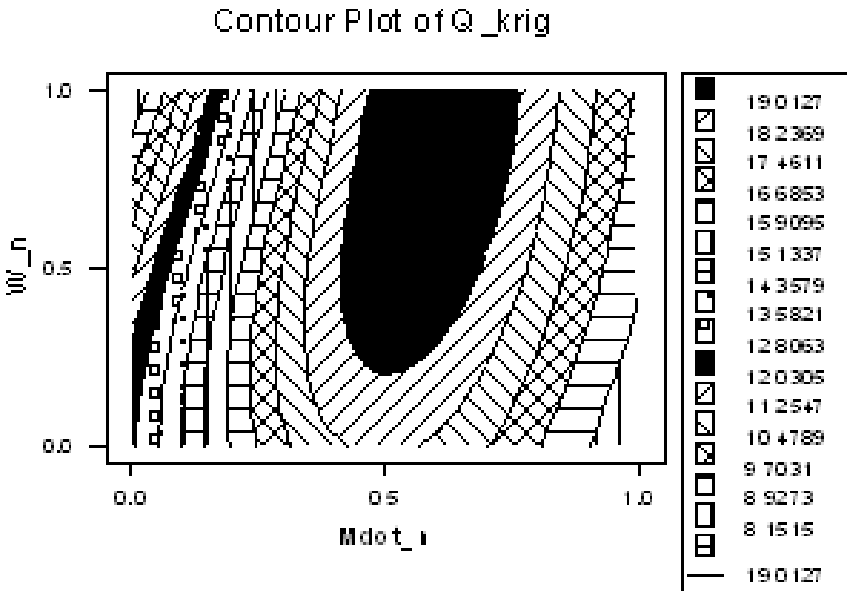


Figure D.26 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 8 Data Points)

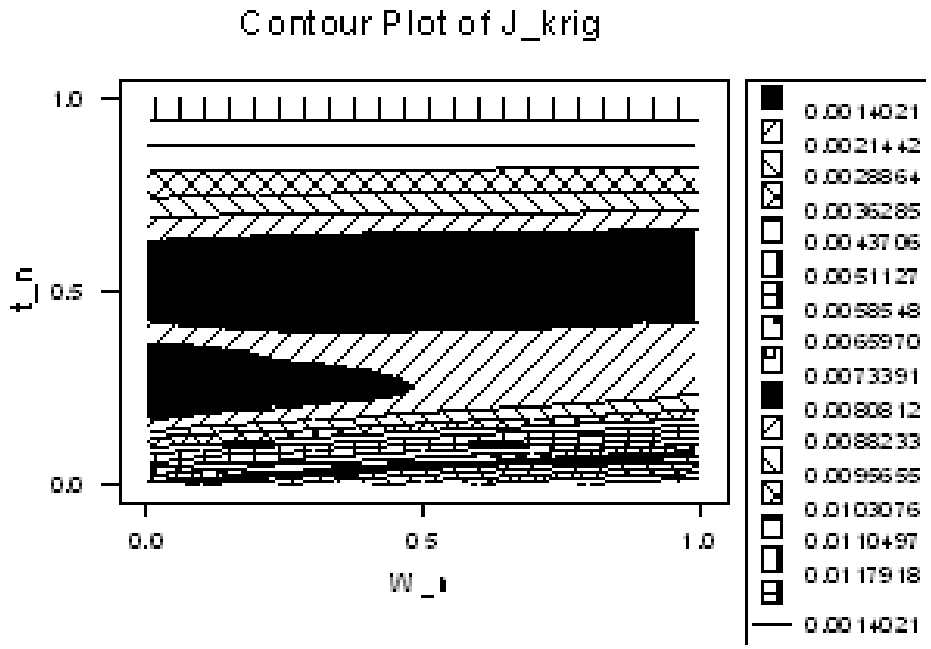


Figure D.27 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 8 Data Points)

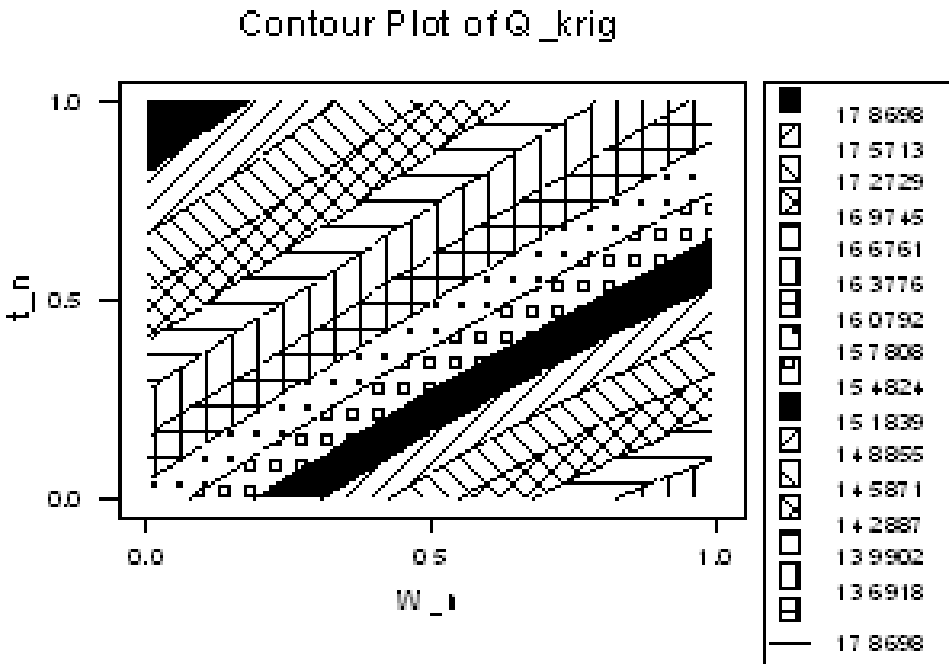


Figure D.28 Contour Plot of Heat Transfer Rate vs. Wall Thickness and Device Width (Kriging Metamodel with 8 Validation Points)

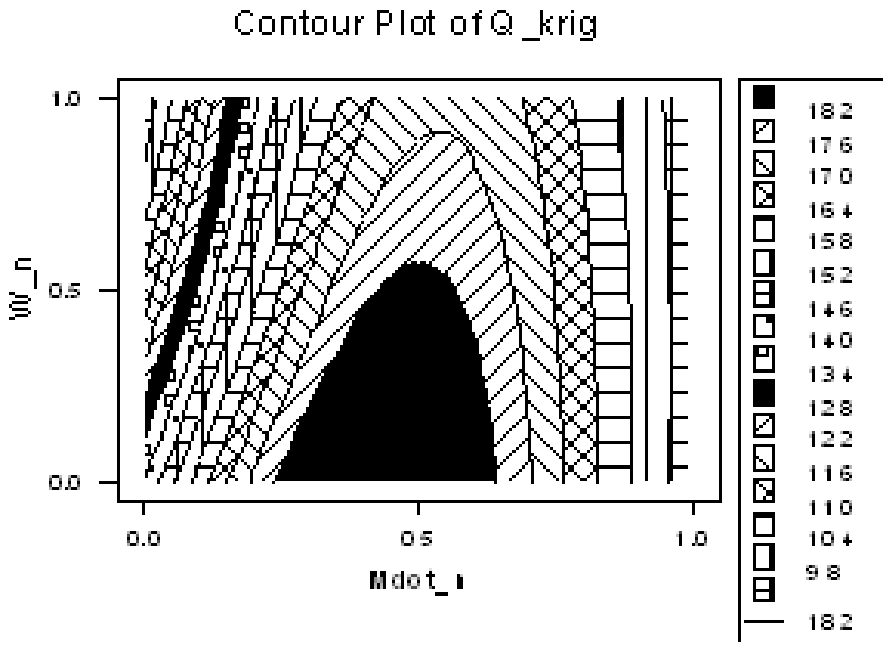


Figure D.29 Contour Plot of Heat Transfer Rate vs. Device Width and Mass Flow Rate (Kriging Metamodel with 8 Validation Points)

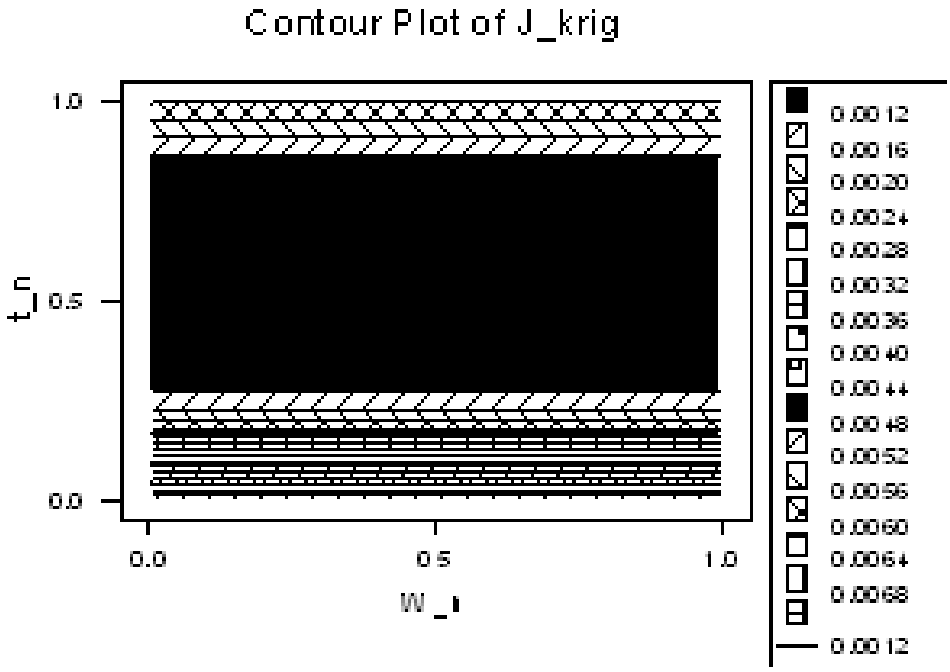


Figure D.30 Contour Plot of Compliance vs. Device Width and Wall Thickness (Kriging Metamodel with 8 Validation Points)

D.3.2 FORTRAN Programs Used in E-RCEM in Section 7.5

The FORTRAN program of altcov.f and altcov.params.h used in SEED, Iteration III – Step 4, in Section 7.5 are enclosed in this section. The programs of altcov.f and altcov.params.h are used to adjust entries of the covariance matrix. Other programs used in the integrated process in E-RCEM are the same as those presented in Appendix D.2.2.

Altcov.f:

```
************************************************************************
*
      program altcov
*
*  This program calculates the alternated correlation matrix, given the
*       initial correlation matrix and predicted prediction errors at
*       possible new data points.
*
*  Updated by: Yao Lin, March 26, 2003
*
*  Original code developed by:
*  Tim Simpson 25 February 1998 / Tony Giunta, 12 May 1997
*
************************************************************************
*
* Input files:
* ------------
*  altcov.params.h - parameter file, specifying numdv, numsamp,
*                       errmax, lambda, fprefix, fprefix2, fprefixnew
*  fprefix.cov     - initial correlation matrix
*  fprefix2.out    - predicted prediction errors at possible new data points
*
* Output files:
* -------------
*  fprefixnew.cov  - alternated correlation matrix
*
* Variables:
* ----------
*  inicov     = the initial correlation matrix
*  newcov     = the alternated correlation matrix
*
* Parameter Variables (to be specified by user in dace.params.h):
* ---------------------------------------------------
*  numsamp = number of data samples from which the correlation matrix
*              is calculated
*
* Local Variables:
* ----------------
*  DOUBLE PRECISION
*  ----------------
*  errpred = the predicted prediction errors associated with each data
*            and possible new data points
*
************************************************************************
```

```
      integer numsamp,numgoal,numdv
      double precision lambda,errmax1,errmax2,gamma
      double precision TargetH1,TargetL1,TargetS1,
     &       TargetH2,TargetL2,TargetS2,
     &       TargetH3,TargetL3,TargetS3
      double precision ymax1,ymin1,
     &       ymax2,ymin2,ymax3,ymin3,
     &       yconstant1,yconstant2,yconstant3
      character TargetType1,TargetType2,TargetType3
      character*20 fprefix,fprefix2,fprefixnew
      character*20 fprefix3,fprefix4,fprefix5,fprefix6
C
C  include parameter settings for numdv,numsamp,fprefix,fprefix2,fprefixnew,
C  errmax, lambda, e.g., in the one-variable problem, for the first step:
C  numdv=1,numsamp=8,fprefix='step1newp',fprefix2='errpred1',
C  fprefixnew='step1altnewp',errmax=0.50,lambda=2.0
C
      include 'altcov.params.h'

      double precision inicov(numsamp,numsamp),
     &       newcov(numsamp,numsamp),
     &       errpred(numresp,numsamp),
     &       goalachieve(numgoal,numsamp),
     &       responsey(numgoal,numsamp),
     &       alpha(2),eta(2),
     &       response,goalachievement,
     &       errmax(numresp),TargetH(numgoal),
     &       TargetL(numgoal),TargetS(numgoal),
     &       ymax(numgoal),ymin(numgoal),
     &       yconstant(numgoal)
      character TargetType(numgoal)
      integer i,j,k,lenstr
      character*20 ftitle
      character*20 deckfile,deckfile2,deckfile3,outfile
      character*20 deckfile4,deckfile5,deckfile6

      errmax(1)=errmax1
      errmax(2)=errmax2
      TargetH(1)=TargetH1
      TargetL(1)=TargetL1
      TargetS(1)=TargetS1
      TargetH(2)=TargetH2
      TargetL(2)=TargetL2
      TargetS(2)=TargetS2
      TargetH(3)=TargetH3
      TargetL(3)=TargetL3
      TargetS(3)=TargetS3
      ymax(1)=ymax1
      ymin(1)=ymin1
      ymax(2)=ymax2
      ymin(2)=ymin2
      ymax(3)=ymax3
      ymin(3)=ymin3
      yconstant(1)=yconstant1
      yconstant(2)=yconstant2
      yconstant(3)=yconstant3
      TargetType(1)=TargetType1
      TargetType(2)=TargetType2
      TargetType(3)=TargetType3

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
```

```
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.cov'

       call getlen(fprefix2,lenstr)
       ftitle=fprefix2
       deckfile2=ftitle(1:lenstr) // '.out'

       call getlen(fprefix3,lenstr)
       ftitle=fprefix3
       deckfile3=ftitle(1:lenstr) // '.out'

       call getlen(fprefix4,lenstr)
       ftitle=fprefix4
       deckfile4=ftitle(1:lenstr) // '.out'

       call getlen(fprefix5,lenstr)
       ftitle=fprefix5
       deckfile5=ftitle(1:lenstr) // '.out'

       deckfile6=fprefix6

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.cov'


       open(21,file=deckfile,status='old')
       open(23,file=deckfile2,status='old')
       open(25,file=deckfile3,status='old')
       open(28,file=deckfile4,status='old')
       open(29,file=deckfile5,status='old')
       open(30,file=deckfile6,status='old')
       open(27,file=outfile,status='unknown')

       print *
       print *, deckfile,deckfile2,deckfile3,
     &       deckfile4,deckfile5,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in sample data...'
      do 10 i=1,numsamp
 10     read (21,*) (inicov(i,j),j=1,numsamp)
      close(21)

C
C  initialize errpred
C
      print *
      write(6,*) 'Reading in and calculating errpred...'
      do 15 j=1,numresp
            do 20 i=1,numsamp
                if (i.le.numold) then
                    errpred(j,i)=0.0
                else
                    if (j.eq.1) then
                       read(23,*) errpred(j,i)
                    else
                       read(25,*) errpred(j,i)
```

```
                    endif
                endif
                if (abs(errpred(j,i)).gt.(errmax(j))) then
                    errpred(j,i)=errmax(j)
                endif
 20   continue
 15   continue
      close(23)
      close(25)

      print *
      write(6,*)
     &  'Reading in responses and calculating goal.achievement...'
      do 55 j=1,numgoal
         do 60 i=1,numsamp
            if (j.eq.1) then
               read (28,*) responsey(j,i)
            elseif (j.eq.2) then
               read (29,*) responsey(j,i)
            else
               read (30,*) responsey(j,i)
            endif
            response=responsey(j,i)+yconstant(j)
            if (TargetType(j).eq.'H') then
               call Hgoalachievecal(goalachievement,TargetH(j),
     &              response,ymax(j),ymin(j),gamma)
               goalachieve(j,i)=goalachievement
            else if (TargetType(j).eq.'L') then
               call Lgoalachievecal(goalachievement,TargetL(j),
     &              response,ymax(j),ymin(j),gamma)
               goalachieve(j,i)=goalachievement
            else if (TargetType(j).eq.'S') then
               call Sgoalachievecal(goalachievement,TargetS(j),
     &              response,ymax(j),ymin(j),gamma)
               goalachieve(j,i)=goalachievement
            endif
 60   continue
 55   continue
      close(28)
      close(29)
      close(30)

C
C  calculate the alternated correlation matrix
C
      do 30 i=1,numsamp
         do 40 j=i,numsamp
         if (i.eq.j) then
            newcov(i,j)=1.0
         elseif (((i.le.numold).AND.(j.le.numold)).OR.
     &             ((i.gt.numold).AND.(j.gt.numold))) then
         newcov(i,j)=inicov(i,j)
         newcov(j,i)=newcov(i,j)
         elseif (((i.le.numold).AND.(j.gt.numold)).OR.
     &             ((i.gt.numold).AND.(j.le.numold))) then
         if (inicov(i,j).eq.1) then
         newcov(i,j)=inicov(i,j)
         newcov(j,i)=newcov(i,j)
         elseif (inicov(i,j).lt.1) then
            alpha(i)=0
            alpha(j)=0
            do 50 k=1,numresp
               alpha(i)=alpha(i)+abs(errpred(k,i))/lambda/
```

```
     &            errmax(k)/numresp
                alpha(j)=alpha(j)+abs(errpred(k,j))/lambda/
     &            errmax(k)/numresp
 50   continue
            alpha(i)=1-alpha(i)
            alpha(j)=1-alpha(j)
            eta(i)=0
            eta(j)=0
            do 65 k=1,numgoal
               eta(i)=eta(i)+goalachieve(k,i)/numgoal
               eta(j)=eta(j)+goalachieve(k,j)/numgoal
 65   continue
            eta(i)=1-eta(i)
            eta(j)=1-eta(j)
         newcov(i,j)=inicov(i,j)
     &   *alpha(i)*alpha(j)*eta(i)*eta(j)
         newcov(j,i)=newcov(i,j)
         endif
         endif
 40   continue
 30   continue

C
C  write alternated correlation matrix into specified .cov file
C
      do 80 i=1,numsamp
        write(27,79) (newcov(i,j),j=1,numsamp)
 79     format(30(f13.5,1x))
 80   continue
      close(27)

      print *
      write(6,*) 'Alternated correlation matrix written to .cov file'

      stop
      end

*************************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*************************************************************************
*
      character*1 blank
      character*20 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
```

```
         return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end

*****************************************************************
*
      subroutine Hgoalachievecal(goalachievement,TargetH,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*****************************************************************
*
      double precision goalachievement,TargetH,response
      double precision ymax,ymin,gamma

      if (response.le.ymin) then
         goalachievement=0.00000000
      else if (response.ge.min(TargetH,ymax)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(response-ymin)/
     &    (min(TargetH,ymax)-ymin)/gamma
      endif

      return
      end


*****************************************************************
*
      subroutine Lgoalachievecal(goalachievement,TargetL,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
```

```
*         New York, pp. 537-538.
*
*********************************************************************
*
      double precision goalachievement,TargetL,response
      double precision ymax,ymin,gamma

      if (response.ge.ymax) then
         goalachievement=0.0000000000
      else if (response.le.max(TargetL,ymin)) then
          goalachievement=1.0/gamma
      else
          goalachievement=(ymax-response)/
     &    (ymax-max(ymin,TargetL))/gamma
      endif

      return
      end

*********************************************************************
*
      subroutine Sgoalachievecal(goalachievement,TargetS,
     &      response,ymax,ymin,gamma)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*********************************************************************
*
      double precision goalachievement,TargetS,response
      double precision ymax,ymin,gamma

      if (response.ge.ymax) then
         goalachievement=0.00000000
      else if (response.le.ymin) then
         goalachievement=0.00000000
      else if (response.eq.TargetS) then
         goalachievement=1.0/gamma
      else if (response<TargetS.AND.response>ymin) then
          goalachievement=(response-ymin)/(TargetS-ymin)/gamma
      else if (response>TargetS.AND.response<ymax) then
          goalachievement=(response-TargetS)/(ymax-TargetS)/gamma
      endif

      return
      end
```

Altcov.params.h

```
C*********************************************************
C                                                        *
C  Parameter input file for 'altcov'                     *
```

```
C    Author: Yao Lin                                          *
C      Date: 3/26/2003                                        *
C                                                             *
C*************************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numdv=3,numsamp=20,numold=18,
     &          numgoal=3,numresp=2,
     &          fprefix='suit3valid',
     &          fprefix2='Qit3st3err.gau',
     &          fprefix3='Jit3st3err.gau',
     &          fprefix4='Qit3val.gau',
     &          fprefix5='Jit3val.gau',
     &          fprefix6='repmoutput1.out',
     &          fprefixnew='suit3altvalid',
     &          errmax1=0.35,
     &          errmax2=0.00268,
     &          lambda=2.0,
     &          ymax1=-6.9,ymin1=-16.0,
     &          TargetL1=-20.0,
     &          TargetH1=-1.0,TargetS1=-1.0,
     &          ymax2=0.01164,ymin2=0.00056,
     &          TargetL2=0.0015,
     &          TargetH2=-1.0,TargetS2=-1.0,
     &          ymax3=0.00033,ymin3=0.00005,
     &          TargetL3=0.00025,
     &          TargetH3=-1.0,TargetS3=-1.0,
     &          TargetType1='L',
     &          TargetType2='L',
     &          TargetType3='L',
     &          yconstant1=0.0,
     &          yconstant2=0.0,
     &          yconstant3=0.0,
     &          gamma=1.25)

C
C  numdv = # design variables
C  numsamp = # samples in data set
C  numold = # old data points in the data set
C
C  fprefix = prefix of titles of file that stores the initial
C            correlation matrix for both old and possible new
C            data points
C
C  fprefix2 = prefix of titles of file that stores the
C             predicted prediction errors at possible new
C             data points
C
C  fprefix3 = prefix of titles of file that stores the
C             predicted response values at all points
C
C
C  fprefixnew = prefix of titles of file that stores the
C               alternated correlation matrix for both old and
C               possible new data points, with prediction errors
C               at these points considered
C
C  errmax = maximum value of the absolute predicted prediction error
C
C  lambda = coefficient used to gauge the adjustment to initial
C           correlation matrix
```

Repmcal.f:

```
************************************************************************
*
      program repmcal
*

      integer numsamp,ncell
      double precision density,totalwidth,length

      character*20 fprefix,fprefixnew,fprefixnew1


      include 'repmcal.h'

      double precision variable(numsamp,3),reynolds(numsamp)
      double precision mdot(numsamp),pdrop(numsamp),vf(numsamp)
      double precision thickness,width,velocity,tin,a,as(numsamp)
      double precision dh,viscosity,af,friction,constraint(numsamp)
      integer i,j,lenstr
      character*16 ftitle
      character*20 deckfile,outfile,outfile1

C
C  open necessary fprefix.cov, fprefix2.out, and fprefixnew.cov files,
C  e.g., step1newp.cov, errpred1.out, step1altnewp.cov
C
       call getlen(fprefix,lenstr)
       ftitle=fprefix
       deckfile=ftitle(1:lenstr) // '.dat'

       call getlen(fprefixnew,lenstr)
       ftitle=fprefixnew
       outfile=ftitle(1:lenstr) // '.out'

       call getlen(fprefixnew1,lenstr)
       ftitle=fprefixnew1
       outfile1=ftitle(1:lenstr) // '.out'

       open(21,file=deckfile,status='old')
       open(27,file=outfile,status='unknown')
       open(30,file=outfile1,status='unknown')

       print *
       print *, deckfile,outfile
       print *, numsamp
C
C  initialize inicov
C
      print *
      write(6,*) 'Reading in points...'
      do 10 i=1,numsamp
 10     read (21,*) (variable(i,j),j=1,3)
      close(21)

C
C  initialize errpred
C

         tin=293

      do 20 i=1,numsamp
         mdot(i)=0.0005+(0.003-0.0005)*variable(i,1)
         width=0.015+(0.035-0.015)*variable(i,2)
```

```
         thickness=0.0002+(0.0008-0.0002)*variable(i,3)
         a=width*width
         dh=width-(ncell+1)*thickness
         af=dh*dh
         as(i)=a-af
         vf(i)=(a-af)/a
         velocity=mdot(i)/density/af
         viscosity=(0.4415*tin+51.638)*0.0000001
         reynolds(i)=velocity*density*dh/viscosity
         friction=64/reynolds(i)
         pdrop(i)=friction*length/dh*density
     &       *velocity*velocity/2
         constraint(i)=30-2663.35*mdot(i)-pdrop(i)
         write(27,76) reynolds(i),vf(i),constraint(i)
 76      format(3(f13.5,1x))
         write(30,79) as(i)
 79      format(f13.5,1x)
C 79      format(1(f13.5,1x))
 20   continue
      close(27)

      print *
      write(6,*) 'outputs written'

      stop
      end

*************************************************************************
*
      subroutine getlen(string,lenstr)
*
*
*  This subroutine is used to determine the actual length of the
*  filename prefix specified by the user in 'detcov.params.h'.
*
*  With this known, the .cov and .det suffixes are
*  concatenated onto the prefix, and the files are opened.
*
*  Author:  Tim Simpson, 2/15/98
*  Modified: Yao Lin,    3/26/2003
*
*  From:  Koffman and Friedman, Fortran (5th ed.), Addison-Wesley,
*         New York, pp. 537-538.
*
*************************************************************************
*
      character*1 blank
      character*16 string
      parameter (blank=' ')
      integer next
      do 10 next = LEN(string), 1, -1
        if (string(next:next).ne.blank) then
          lenstr=next
          return
        end if
 10   continue
      lenstr=0
      if (lenstr.eq.0) then
        write(6,*) 'You have not specified a file name prefix'
        stop
      end if
      return
      end
```

Repmcal.h

```
C*************************************************************
C                                                             *
C  Parameter input file for 'repmcal'                          *
C    Author: Yao Lin                                         *
C      Date: 3/26/2003                                       *
C                                                             *
C*************************************************************
C
C  specify parameter values for dace modeling software
C

      parameter ( numsamp=1,
     &          fprefix='repminput',fprefixnew='repmoutput',
     &          fprefixnew1='repmoutput1',
     &          density=1.205,length=0.075,ncell=8 )

C
C  numsamp = # samples in data set
```

D.3.3 Implementation of E-RCEM in iSIGHT in Section 7.5

Figures presented in this section illustrate how the SEED method is implemented in iSIGHT. The organization of tasks in Iteration I – Step 5 is shown in .

In Iteration I – Step 5, with information from metamodels of prediction errors, we use eight simulation codes in iSIGHT, Covmat, Qerr, Jerr, Q, J, AsConstraint, Altcov, and Detcov. Covmat is used to formulate the covariance matrix, Qerr and Jerr are metamodels to predict prediction errors, Q and J are used to predict responses for total heat transfer rate and compliance, AsConstraint is used to calculate the response of cross-section area, A_s, and test design constraints. Altcov is used to adjust entries of the covariance matrix, and Detcov is used to calculate the determinant.

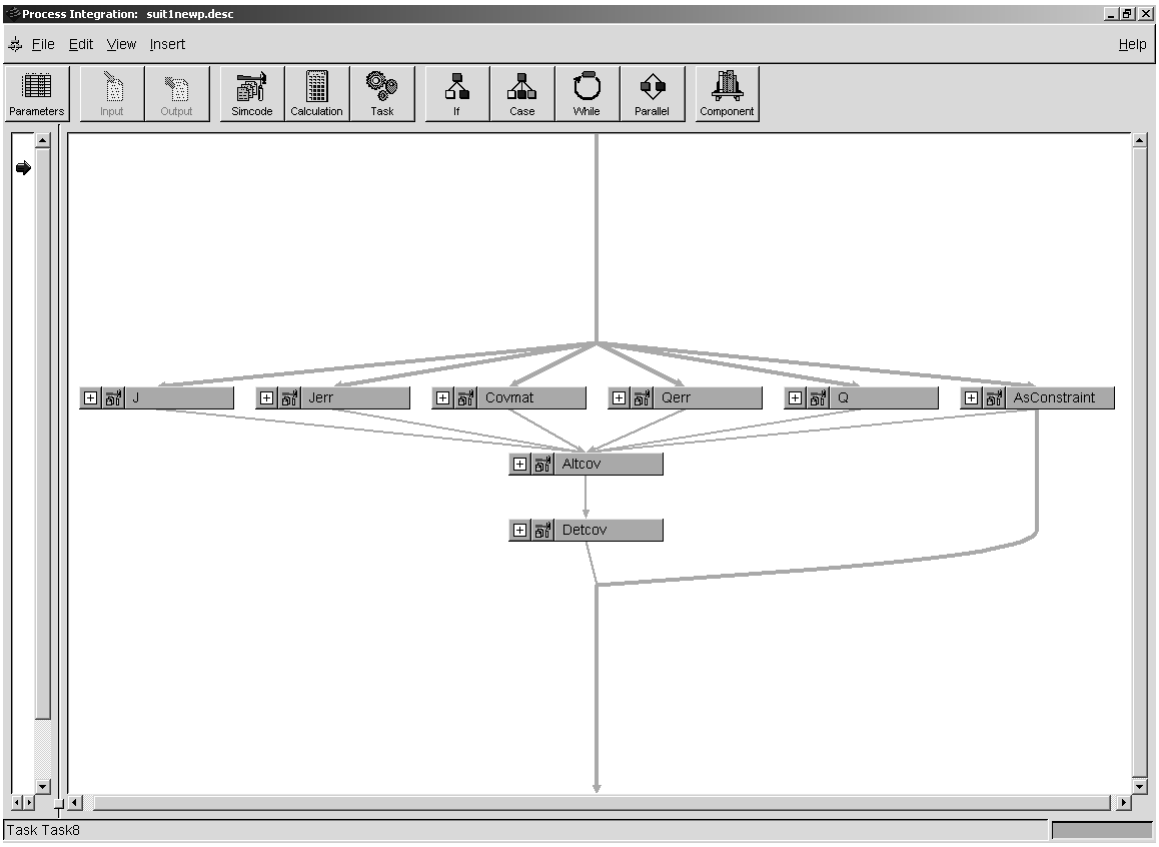


Figure D.31 Implementation of E-RCEM in iSIGHT – Iteration I, Step 5

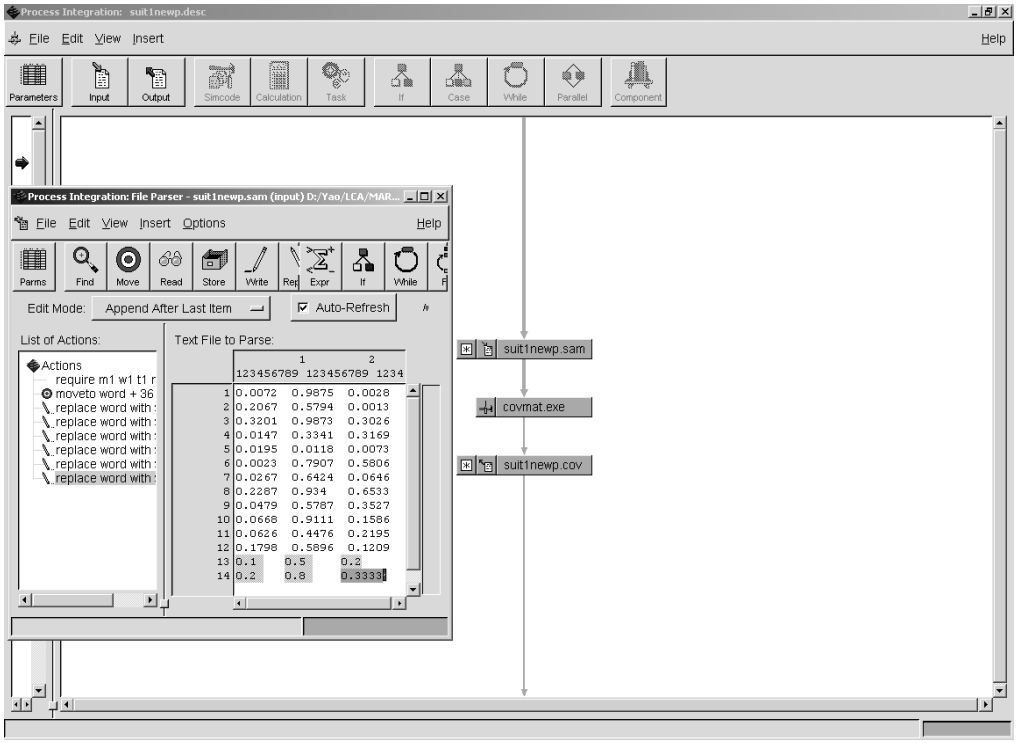


Figure D.32 Input Mapping for Covmat.f in E-RCEM – Iteration I, Step 5

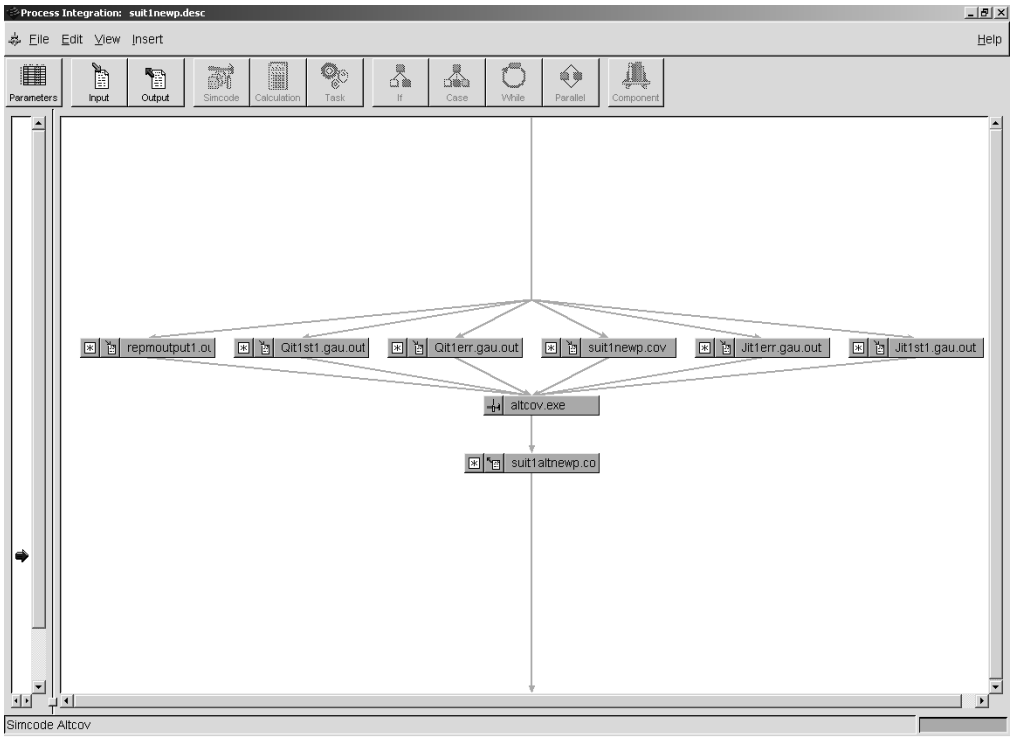


Figure D.33 Organization of Input and Output for Altcov.f in E-RCEM – Iteration I, Step 5

REFERENCES

Alexandrov, N. Dennis, J.E. Jr., Lewis, R.M. and Torczon, V., 1998, "A Trust Region Framework for Managing the Use of Approximation Models in Optimization," *Structural Optimization*, 15(1), 16-23.

Anstreicher, K. M., Famps, M., Lee, J. and Williams, J., 1996, "Continuous Relaxations for Constrained Maximum-Entropy Sampling", *Integer Programming and Combinatorial Optimization*, W. H. Cunningham, S. T. McCormick, and M. Queyranne, eds., No. 1084, Springer-Verlag, New York, 1996, pp.234-248.

Atkinson, A. C. and Haines, L. M., 1996, "Designs for Nonlinear and Generalized Linear Models," *Handbook of Statistics (Ghosh, S. and Rao, C. R., eds.)*, Elsevier Science, New York, pp.437-475.

Balabanov, V.O., Giunta, A.A., Golovidov, O., Grossman, B., Mason, W.H. and Watson, L.T., 1999, "Reasonable Design Space Approach to Response Surface Approximation," *Journal of Aircraft*, 36(1), 308-315.

Barton, R. R., 1992, December 13-16, "Metamodels for Simulation Input-Output Relations," *Proceedings of the 1992 Winter Simulation Conference (Swain, J. J., Goldsman, D., et al., eds.),* Arlington, VA, IEEE, pp. 289-299.

Barton, R. R., 1994, "Metamodeling: A State of the Art Review," *Proceedings of the 1994 Winter Simulation Conference*, Lake Beuna Vista, FL, IEEE.

Bernardo, J. M., 1979, "Expected Information as Expected Utility," *Ann. Statist.*, Vol. 7, pp.686-690.

Bernardo, J. M. and Smith, A. F. M., 1994, *Bayesian Theory*, New York: Wiley.

Berry, D. A. and Fristedt, B., 1985, *Bandit Problems: Sequential Allocation of Experiments*, Chapman and Hall, London.

Biles, W. E., 1984, "Design of Simulation Experiments," *Proceedings of the 1984 Winter Simulation Conference (WSC),* Dallas, TX, IEEE, pp. 99-104.

Booker, A. J., 1996, "Case Studies in Design and Analysis of Computer Experiments," *Proceedings of the Section on Physical and Engineering Sciences,* American Statistical Association.

Booker, A. J., Conn, A. R., Dennis, J. E., Frank, P. D., Serafini, D., Torczon, V. and Trosset, M., 1996, "Multi-Level Design Optimization: A Boeing/IBM/Rice Collaborative Project," *1996 Final Report, ISSTECH-96-031,* The Boeing Company, Seattle, WA.

Booker, A. J., Conn, A. R., Dennis, J. E., Frank, P. D., Trosset, M. and Torczon, V., 1995, "Global Modeling for Optimization: Boeing/IBM/Rice Collaborative Project," *1995 Final Report, ISSTECH-95-032,* The Boeing Company, Seattle, WA.

Box, G., 1988, "Signal-to-Noise Ratios, Performance Criteria, and Transformations," *Technometrics*, Vol. 30, No. 1, pp. 1-18.

Box, G. E. P. and Draper, N. R., 1987, *Empirical Model Building and Response Surfaces*, John Wiley & Sons, New York.

Box, G.E.P. and Draper, N.R., 1969, *Evolutionary Operation: A Statistical Method for Process Management*, John Wiley & Sons, New York.

Box, G. E. P., Hunter, W. G. and Hunter, J. S., 1978, *Statistics for Experimenters*, John Wiley &Sons, Inc., New York.

Box, G. E. P. and Wilson, K. B., 1951, "On the Experimental Attainment of Optimum Conditions," *Journal of the Royal Statistical Society*, Series B, 13, pp. 1-45.

Bras, B. A. and Mistree, F., 1991, "Designing Design Processes in Decision-Based Concurrent Engineering," *SAE Transactions, Journal of Materials & Manufacturing ,* SAE International, Warrendale, PA, pp. 451-458.

Brassard, M. and Ritter, D., 1994, "The Memory Jogger™ II – A Pocket Guide of Tolls for Continuous Improvement & Effective Planning," Methuen, MA, GOAL/QPC.

Breiman, L., Friedman, J. H., Olshen, R. and Stone, C. J., 1984, *Classification and Regression Trees*, Wadsworth, Belmont, California.

Byrne, D.M. and Taguchi, S., 1987, "The Taguchi Approach to Parameter Design," *40th Annual Quality Congress Transactions*, Milwaukee, WI, American Society for Quality Control, pp. 19-26.

Chaloner, K. and Verdinelli, I., 1995, "Bayesian Experimental Design: A Review," *Statistical Science,* Vol. 10, No. 3, pp. 273-304.

Chen, V. C. P., 1993, *Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming*, Ph.D. Dissertation, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

Chen, V. C. P., 1999, "Application of Orthogonal Arrays and MARS to Inventory Forecasting Stochastic Dynamic Programs," *Computational Statistics and Data Analysis*, 30, pp. 317-341.

Chen, V. C. P., Ruppert, D., and Shoemaker, C. A., 1999, "Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming," *Operations Research*, Vol 47, pp. 38-53.

Chen, W., 1995, *A Robust Concept Exploration Method for Configuring Complex Systems,* Ph.D. Dissertation, The George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA.

Chen, W., Allen, J. K., Mistree, F. and Tsui, K. L., 1995, "Integration of Response Surface Methods with the Compromise Decision Support Problem in Developing a General Robust Design Procedure," *ASME Design Automation Conference, Boston, Massachusetts, ASME*, New York, pp. 485-492.

Chen, W., Allen, J. K., Mavris, D. and Mistree, F., 1996a, "A Concept Exploration Method for Determining Robust Top-Level Specifications," *Engineering Optimization,* Vol. 26, No. 2, pp. 137-158.

Chen, W., Allen, J.K., Tsui, K. L., and Mistree, F., 1996b, "A Procedure for Robust Design: Minimizing Variations Caused by Noise Factors and Control Factors," *ASME Journal of Mechanical Design*, Vol. 118, No. 4, pp. 478-485.

Chen, W., Allen, J.K., Schrage, D.P. and Mistree, F., 1997, "Statistical Experimentation Methods for Achieving Affordable Concurrent Systems Design," *AIAA Journal*, 35(5), 893-900.

Cheng, B. and Titterington, D. M., 1994, "Neural Networks: A Review from a Statistical Perspective," *Statistical Science*, Vol. 9, No. 1, pp. 2-54.

Choi, H. and Fernandez, M. G., 2003, *Towards Finite Element-Based Thermal Topological Design of Unit Cells for Linear Cellular Alloys*, Semester Project Report, ME 6124, Spring, 2003.

Church, B. C., Dempsey, B. M., Clark, J. L., Sanders, T. H. and Cochran, J. K., 2001, "Copper Alloys from Oxide Reduction for High Conductivity Applications," *Proceedings of IMECE 2001, International Mechanical Engineering Congress and Exposition*, New York, 2001.

Clausing, D., 1994, *Total Quality Development – A Step by Step Guide to World-Class Concurrent Engineering,* ASME, New York.

Clausius, R., 1865, "Ueber Verschiedene fur die Anwendung Bequeme Formen der Hauptgleichungen der Mechanischen Warmetheorie," *Annalen der Physik und Chemie,* Vol.125, pp.353-400.Ericsson, K. A. and Simon, H. A., 1980, "Verbal Reports as Data," Vol. 87, No. 3, pp. 215-251.

Clyde, M. A., 1994, "A System for Bayesian Optimal Design Using XLISP-STAT," *ISDS Discussion Paper*, May 26, 1994.

Cochran, J. K., Lee, K. J., McDowell, D. L. and Sanders, T. H., 2000, "Low Density Monolithic Honeycombs by Thermal Chemical Processing," *Proceedings of the 4th*

Conference on Aerospace Materials, Processes, and Environmental Technology, Huntsville, AL, 2000.

Congdon, P., 2001, *Bayesian Statistical Modeling*, Chichester, England; New York: Wiley; 2001.

Cox, D. D. and John, S., 1995, March 13-16, "SDO: A Statistical Method for Global Optimization," *Proceedings of the ICASE/NASA Langley Workshop on Multidisciplinary Optimization (Alexandrov, N. M. and Hussaini, M. Y., eds.),* Hampton, VA, SIAM, pp. 315-329.

Cozzolino, J. M. and Zahner, M. J., 1973, "The Maximum Entropy Distribution of the Future Market Price of a Stock," *Operations Research*, Vol.21, pp.1200-1211.

Cressie, N. A. C., 1993, *Statistics for Spatial Data*, John Wiley & Sons, New York.

Currin, C., Mitchell, M., Morris, M., and Ylvisaker, D., 1991, "Bayesian Prediction of Deterministic Functions, with Applications to the Design and Analysis of Computer Experiments," *Journal of the American Statistical Association*, Volume 86, pp.953-963.

Dennis, J. E. and Torczon, V., 1995, March 13-16, "Managing Approximation Models in Optimization," *Proceedings of the ICASE/NASA Langley Workshop on Multidisciplinary Design Optimization (Alexandrov, N. M. and Hussaini, M. Y., eds.),* Hampton, VA, SIAM, pp. 330-347.

Dixon, L. C. W. and Szego, G. P., 1978, *The Global Optimisation Problem: An Introduction, Towards Global Optimisation 2*, North-Holland Publishing Company, New York.

Du, X. and Chen, W., 2001, "A Most Probable Point Based Method for Uncertainty Analysis," *Journal of Design and Manufacturing Automation*, Vol. 4, No. 1, pp. 47-66, 2001.

Du, X. and Chen, W., 2000, "Towards a Better Understanding of Modeling Feasibility Robustness in Engineering," *ASME Journal of Mechanical Design*, Vol. 122, No. 4, pp. 357-583, 2000.

DuMouchel, W. and Jones, B., 1994, "A Simple Bayesian Modification of *D*-optimal Designs to Reduce Dependence on an Assumed Model," *Technometrics*, Vol. 36, pp.37-47.

Ericsson, K. A. and Simon, H. A., 1980, "Verbal Reports as Data," Vol. 87, No. 3, pp. 215-251.

Eschenauer, H. A. and Olhoff, N., 2001, "Topology Optimization of Continuum Structures: A Review," *Applied Mechanics Reviews,* Vol. 54, No. 4, pp. 331-389. 3.

Evans, A. G., Hutchinson, J. W., Fleck, N. A., Ashby, M. F. and Wadley, H. N. G., 2001, "The Topological Design of Multifunctional Cellular Materials," *Progress in Materials Science,* Vol. 46, No. 3-4, 2001, pp. 309-327.

Fang, S.-C. and Tsao, H.-S.J., 1993, "An Unconstrained Convex Programming Approach to Solving Convex Quadratic Programming Problems," *Optimization*, Vol. 27, pp.235-243.

Fang, S.-C. and Tsao, H.-S.J., 1995, "Linear-Constrained Entropy Maximization Problem with Quadratic Cost and its Application to Transportation Planning Problems," *Transportation Science*, Vol. 29, pp.353-365.

Fang, S.-C., Rajasekera, J. R. and Tsao, H.-S.J., 1997, *Entropy Optimization and Mathematical Programming*, Kluwer Academic Publishers, Boston/London/Dordrecht.

Farhang-Mehr, A. and Azarm, S., 2002, "A Sequential Information-Theoretic Approach to Design of Computer Experiments," *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, Georgia, 2002.

Finger, S. and Dixon, J. R., 1989a, "A Review of Research in Mechanical Engineering Design. Part 1: Descriptive, Prescriptive, and Computer-Based Models of Design Processes," *Research in Engineering Design,* Vol. 1, pp. 51-67.

Finger, S. and Dixon, J. R., 1989b, "A Review of Research in Mechanical Engineering Design. Part 2: Representations, Analysis, and Design for the Life Cycle," *Research in Engineering Design,* Vol. 1, pp. 121-137.

Freeman, P. R., 1970, “Optimal Bayesian Sequential Estimation of the Median Effective Dose,” *Biometrika*, Vol. 57, pp.79-89.

Friedman, J. H., 1991, “Multivariate Adaptive Regression Splines (with discussion),” *Annals of Statistics*, Vol 19, pp.1-141.

Giunta, A. A., 1997, "Aircraft Multidisciplinary Design Optimization Using Design of Experiments Theory and Response Surface Modeling," *Ph.D. Dissertation and MAD Center Report No. 97-05-01,* Department of Aerospace and Ocean Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Giunta, A.A., Balabanov, V., Kaufmann, M., Burgee, S., Grossman, B., Haftka, R.T., Mason, W.H. and Watson, L.T., 1996, “Variable-Complexity Response Surface Design of An HSCT Configuration,” *Multidisciplinary Design Optimization: State of the Art – Proceedings of the ICASE/NASA Langley Workshop on Multidisciplinary Design Optimization*, SIAM, Hampton, VA, pp. 348-367.

Giunta, A. A., Balabanov, V., Haim, D., Grossman, B., Mason, W. H. and Watson, L. T., 1996, “Wing Design for a High-Speed Civil Transport Using a Design of Experiments Methodology,” *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Bellevue, WA, pp. 168-183.

Giunta, A. A., Dudley, J. M., Narducci, R., Grossman, B., Haftka, R. T., Mason, W. H. and Watson, L. T., 1994, September 7-9, "Noisy Aerodynamic Response and Smooth Approximations in HSCT Design," *5th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Panama City, FL, AIAA, Vol. 2, pp. 1117-1128. AIAA-94-4376-CP.

Giunta, A., Watson, L. T. and Koehler, J., 1998, September 2-4, "A Comparison of Approximation Modeling Techniques: Polynomial Versus Interpolating Models," *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis & Optimization,* St. Louis, MI, AIAA, AIAA-98-4758.

Goldman, R.W., 2001, *Development of a Rollover-Warning Device for Road Vehicles*, Ph.D. Dissertation, Department of Mechanical and Nuclear Engineering, The Pennsylvania State University, December, 2001.

Gu, L., 2001, “A Comparison of Polynomial Based Regression Models in Vehicle Safety Analsysis,” *ASME Design Engineering Technical Conferences – Design Automation Conference,* Pittsburgh, PA, Paper No. DETC2001/DAC-21063.

Gu, S., Lu, T. J. and Evans, A. G., 2001, “On the Design of Two-Dimensional Cellular Metals for Combined Heat Dissipation and Structural Load Capacity,” *International Journal of Heat and Mass Transfer,* Vol. 44, No. 11, 2001, pp. 2163-2175.

Gu, X., Renaud, J.E., Batill, S.M., Brach, R.M., and Budhiraja, A.S., 2000, "Worst Case Propagated Uncertainty of Multidisciplinary Systems in Robust Design Optimization," *Structural and Multidisciplinary Optimization* , Vol.20, No.3, pp.190-213Guiasu, S., 1977, *Information Theory with Applications*, McGraw-Hill, New York.

Guiasu, S., 1986, “Maximum Entropy Condition in Queueing Theory,” *Journal of Operational Research Society*, Vol. 37, ppp.293-301.

Hastie, T., Tibshirani, R., and Friedman, J., 2001, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer Series in Statistics.

Hayes, A. M., Wang, A., Dempsey, B. M., and McDowell, D. L., 2001, “Mechanics of Linear Cellular Alloys,” *Proceedings of IMECE, International Mechanical Engineering Congress and Exposition*, New York, NY.

Healy, M. J., Kowalik, J. S. and Ramsay, J. W., 1975, "Airplane Engine Selection by Optimization of Surface Fit Approximations," *Journal of Aircraft,* Vol. 12, No. 7, pp. 593-599.

Holnicki-Szulc, J., Pawlowski, P. and Wiklo, M., 2003, “High-Performance Impact Absorbing Materials – The Concept, Design Tools and Applications,” *Institute of Physics Publishing, Smart Materials and Structures*, **12** (2003) 461-467.

Hubka, V., 1982, *Principles of Engineering Design,* Butterworth & Co. (Publishers) Ltd., London.

Ignizio, J. P., 1985, *Introduction to Linear Goal Programming*, Sage University Papers, Beverly Hills, CA.

Incropera, F. P. and DeWitt, D. P., 1996, *Fundamentals of Heat and Mass Transfer,* John Wiley & Sons, New York, 1996.

iSIGHT, Engineous Software, Inc., Cary, NC, Version 7.0, 2003.

Jaynes, E. T., 1957, "Information Theory and Statistical Mechanics II," *Physics Review*, Vol.108, pp.171-190.

Jin, R., Chen, W., and Sudjianto, A., 2002, "On Sequential Sampling for Global Metamodeling in Engineering Design," *ASME 2002 Design Engineering Technical Conferences and Computer and Information in Engineering Conference*, Montreal, Canada, September 29-October 2, 2002. Paper No. DETC2002/DAC-34092.

John, R.C. St. and Draper, N. R., 1975, "D-Optimality for Regression Designs: A Review," *Technometrics*, Vol. 17, No.1, February 1975.

Johnson, M. E., Moore, L. M. and Ylvisaker, D., 1990, "Minimax and Maximin Distance Designs," *Journal of Statistical Planning and Inference,* Vol. 26, No. 2, pp. 131-148.

Jones, D. R., Schonlau, M., and Welch, W. J., 1998, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, 13:455-492.

Journel, A. G. and Huijbregts, C. J., 1978, *Mining Geostatistics*, Academic Press, New York.

Kapur, J. N. and Kesavan, H. K., 1992, *Entropy Optimization Principles with Applications*, Academic Press, Boston.

Kiefer, J., 1958, "On the Nonrandomized Optimality and Randomized Non-optimality of Symmetrical Designs," *Ann. Math. Stat.* Vol. 29, p675-699.

Kiefer, J., 1959, "Optimum Experimental Designs," *Journal of the Royal Statistical Society B*, Vol. 21, pp. 298-325.

Kiefer, J., 1961, "Optimum Designs in Regression Problems," *Annals of Mathematical Statistics*, Vol. 21, pp. 272-304.

Kiefer, J., 1985, *Jack Carl Kiefer Collected Papers III*, Springer, New York.

Kiefer, J. and Wolfowitz, J., 1959, "Optimum Designs in Regression Problems," *Annals of Mathematical Statistics*, Vol. 30, pp. 271-294.

Kiefer, J. and Wolfowitz, J., 1960, "The Equivalence of Two Extremum Problems," *Canad. J. Math.*, Vol 12, 363.

Kleijnen, J.P.C., 1987, *Statistics: Models and Tools for Simulation Practitioners*, Marcel Dekker, NY.

Koch, P. N., 1998, *Hierarchical Modeling and Robust Synthesis for the Preliminary Design of Large Scale Complex Systems,* Ph.D. Dissertation, The G. W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Koch, P. N., Barlow, A., Mistree, F. and Allen, J. K., 1996, "Configuring Turbine Propulsion Systems Using Robust Concept Exploration," *ASME Design Engineering Technical Conferences,* Irvine, CA, Paper No. 96-DETC/DAC-1472.

Koch, P. N., Allen, J. K., Mistree, F. and Mavris, D., 1997, September 14-17, "The Problem of Size in Robust Design," *Advances in Design Automation,* Sacramento, CA, ASME, Paper No. DETC97/DAC-3983.

Koehler, J. R. and Owen, A. B., 1996, "Computer Experiments," *Handbook of Statistics (Ghosh, S. and Rao, C. R., eds.)*, Elsevier Science, New York, pp.261-308.

Kumar, V., Hoshino, K. and Kumar, U., 1989, "An Application of the Entropy Maximization Approach in Shopping Area Planning," *International Journal of General Systems*, Vol. 16, pp.25-42.

Laird, J. E., Newell, A. and Rosenbloom, P. S., 1987, "SOAR: An Architecture for General Intelligence," Vol. 33, No. 1, pp. 1-64.

Lee, D., 2001, "Maximum Entropy Sampling," In A.H. El-Shaarawi and W.W. Piegorsch, editors, "Encyclopedia of Environmetrics". Wiley, 2001.

Lee, J. and Williams J., 1999, *Generalized Maximum-Entropy Sampling*, University of Kentucky, Department of Mathematics, Technical report No. 99-10, July 1999.

Lewis, K., Lucas, T. and Mistree, F., 1994, September 7-9, "A Decision Based Approach to Developing Ranged Top-Level Aircraft Specifications: A Conceptual Exposition," *5th AIAA/USAF/NASA/ISSMOSymposium on Multidisciplinary Analysis and Optimization,* Panama City, FL, Vol. 1, pp. 465-481.

Li, H.-L. and Chou, C.-T., 1994, "A Global Approach for Nonlinear Mixed Discrete Programming in Design Optimization," *Engineering Optimization*, Vol. 22, pp. 109-122.

Lin, Y., 2000, *Robust Design Goal Formulations and Metamodeling Techniques*, MS Thesis, the George. W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, Georgia.

Lin, Y., Krishnapur, K., Allen, J. K. and Mistree, F., 1999, "Robust Design: Goal Formulations and A Comparison of Metamodeling Methods," *1999 ASME Design Automation Conference*, Las Vegas, Neveda, ASME DETC99/DAC-8608.

Lindley, D. V., 1956, "On a Measure of Information Provided by an Experiment," *The Annals of Mathematical Statistics*, Volume 27, pp. 986-1005.

Lu, T. J., 1999, "Heat Transfer Efficiency of Metal Honeycombs," *International Journal of Heat and Mass Transfer,* Vol. 42, No. 11, 1999, pp. 2031-2040.

Lucas, J.M., 1994, "Using Response Surface Methodology to Achieve a Robust Process," *Journal of Quality Technology*, Vol. 26, No. 4, pp. 248-260.

Mallet, C. G., 1998, *A Wavelet Tour of Signal Processing*, Academic Press, Boston.

Matheron, G., 1963, "Principles of Geostatistics," *Economic Geology*, 58, pp. 1246-1266.

Mavris, D.N., Bandte, O., DeLaurentis, D.A., 1999, "Robust Design Simulation: A Probabilistic Approach to Multidisciplinary Design," *AIAA Journal of Aircraft*, Vol. 36, No. 1, pp. 298-307.

McKay, M. D., Beckman, R. J. and Conover, W. J., 1979, "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics,* Vol. 21, No. 2, pp. 239-245.

Mistree, F., Smith, W. F., Bras, B., Allen, J. K. and Muster, D., 1990a, "Decision-Based Design: A Contemporary Paradigm for Ship Design," *Transactions, Society of Naval Architects and Marine Engineers,* Jersey City, New Jersey, pp. 565-597.

Mistree, F., Muster, D., Srinivasan, S. and Mudali, S., 1990b, "Design of Linkages: A Conceptual Exercise in Designing for Concept," *Mechanism and Machine Theory*, Vol. 25, No. 3, pp. 273-286.

Mistree, F., Smith, W. F. and Bras, B. A., 1993a, "A Decision-Based Approach to Concurrent Engineering," *Handbook of Concurrent Engineering*, Paresai, H. R. and Sullivan, W., ed., Chapman & Hall, New York, pp. 127-158.

Mistree, F., Hughes, O.F. and Bras, B.A., 1993b, "The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm," *Structural Optimization*, Vol. 5, No. 3, pp. 141-144.

Mistree, F., Lewis, K. and Stonis, L., 1994, "Selection in the Conceptual Design of Aircraft," *AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, Florida, September 7-9, 1994, 1153-1166, Paper No. AIAA-94-4382-CP.

Mitchell, T. J. and Morris, M. D., 1992a, "Bayesian Design and Analysis of Computer Experiments: Two Examples," *Statistica Sinica,* Vol. 2, pp. 359-379.

Mitchell, T. J. and Morris, M. D., 1992b, December 13-16, "The Spatial Correlation Function Approach to Response Surface Estimation," *Proceedings of the 1992 Winter Simulation Conference (Swain, J. J., Goldsman, D., et al., eds.),* Arlington, VA, IEEE, pp. 565-571.

Montgomery, D. C., 1991, *Design and Analysis of Experiments*, Third Edition, John Wiley & Sons, New York.

Montgomery, D. C. and Evans, D. M., Jr., 1975, "Second-Order Response Surface Designs in Computer Simulation," *Simulation,* Vol. 29, No. 6, pp. 169-178.

Muster, D. and Mistree, F., 1988, "The Decision Support Problem Techniques in Engineering Design," *The International Journal of Applied Engineering Education*, Vol. 4, No. 1, pp. 23-33.

Myers, R. H. and Montgomery, D. C., 1995, *Response Surface Methodology: Process and Product Optimization Using Designed Experiments*, John Wiley & Sons, New York.

Myers, R. H., Khuri, A. I. and Carter, W. H., 1989, "Response Surface Methodology: 1966-1988," *Technometrics,* Vol. 31, No. 2, May, pp. 137-157.

Nair, V.N., 1992, "Taguchi's Parameter Design: A Panel Discussion," *Technometrics*, Vol. 34, No. 2, pp. 127-161.

Nevill, G. E., 1989, "Computational Models of Design Processes," *Design Theory '88: Proceedings of the 1988 NSF Grantee Workshop on Design Theory and Methodology,* Springer-Verlag, New York.

Osio, I. G. and Amon, C. H., 1996, "An Engineering Design Methodology with Multistage Bayesian Surrogates and Optimal Sampling," *Research in Engineering Design,* Vol. 8, No. 4, pp. 189-206.

Otto, K. N. and Antonsson, E. K., 1993, "Extensions to the Taguchi Method of Product Design," *Journal of Mechanical Design,* Vol. 115, No. 1, pp. 5-13.

Pahl, G. and Beitz, W., 1984, *Engineering Design,* The Design Council/Springer-Verlag, London/Berlin.

Pahl, G. and Beitz, W., 1986, *Konstruktionslehre – Handbuch fuer Studium und Praxis,* Springer – Verlag, Berlin.

Parkinson, A., Sorensen, C. and Pourhassan, N., 1993, "A General Approach for Robust Optimal Design," *Transactions of the ASME*, Vol. 115, pp. 74-80.

Pedersen, K., Emblemsvag, J., Allen, J. K., and Mistree, F., 2000, "Validating Design Methods and Research – The Validation Square," *ASME Design Theory and Methodology Conference,* Baltimore, MD, ASME, DETC00/DTM-14579.

Phadke, M.S., 1989, *Quality Engineering using Robust Design*, Prentice Hall, Englewood Cliffs, NJ.

Pignatiello, J. J. and Ramberg, J. S., 1991, "Top Ten Triumphs and Tragedies of Genichi Taguchi," Vol. 4, pp. 211-225.

Pilz, J., 1991, *Bayesian Estimation and Experimental Design in Linear Regression Models*, New York: Wiley.

Pukelsheim, F., 1993, *Optimal Design of Experiments*, New York: Wiley.

Rajasekera, J. R. and Fang, S.-C., 1992, "Deriving an Unconstrained Convex Program for Linear Programming," *Journal of Optimization Theory and Applications*, Vol. 75, pp.603-612.

Ramakrishnan, B. and Rao, S.S., 1991, "A Robust Optimization Approach using Taguchi's Loss Function for Solving Nonlinear Optimization Problems," *Advances in Design Automation – Design Automation and Design Optimization*, Miami, FL, ASME, pp. 241-248.

Ramberg, J.S., Sanchez, S.M., Sanchez, P.J. and Hollick, L.J., 1991, "Designing Simulation Experiments: Taguchi Methods and Response Surface Metamodels," *Proceedings, 1991 Winter Simulation Conference*, Phoenix, AZ, IEEE, pp. 167-176.

Rangarajan, B., 1998, *Robust Concurrent Design of Automobile Engine Lubricated Components*, The George W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA.

Reddy, S. Y., 1996, August 18-22, "HIDER: A Methodology for Early-Stage Exploration of Design Space," *Advances in Design Automation (Dutta, D., ed.),* Irvine, CA, ASME, Paper No. 96-DETC/DAC-1089.

Renaud, J. E., 1992, August, *Sequential Approximation in Non-Hierarchic System Decomposition and Optimization: A Multidisciplinary Design Tool,* Doctoral Dissertation, Rensselear Polytechnic Institute, Troy, NY.

Renaud, J. E. and Gabrielle, G. A., 1991, September 22-25, "Sequential Global Approximation in Non-Hierarchic System Decomposition and Optimization," *Advances in Design Automation - Design Automation and Design Optimization (Gabriele, G., ed.),* Miami, FL, ASME, Vol. 32-1, pp. 191-200.

Renaud, J. E. and Gabriele, G. A., 1994, "Approximation in Nonhierarchic System Optimization," *AIAA Journal,* Vol. 32, No. 1, pp. 198-205.

Renyi, A., 1961, "On Measures of Entropy and Information," In *Proc. 4th Berkeley Symp. Mathematical Statistics and Probability*, Volume 1, pp. 547-561. Berkeley: University of California Press.

Renyi, A., 1970, *Probability Theory*, Amsterdam, North-Holland.

Rodriguez, J. F., Renaud, J. E. and Watson, L. T., 1997, September 14-17, "Trust Region Augmented Lagrangian Methods for Sequential Response Surface Approximation and Optimization," *Advances in Design Automation (Dutta, D., ed.),* Sacramento, CA, ASME, Paper No. DETC97/DAC-3773.

Ross, P.J., 1988, *Taguchi Techniques for Quality Engineering*, McGraw-Hill, New York, NY.

Roth, K., 1982, *Konstruieren mit Konstruktionskatalogen,* Springer-Verlag, Berlin.

Rozvany, G. I. N., 2001, "Aims, Scope, Methods, History, and Unified Terminology of Computer-Aided Topology Optimization in Structural Mechanics," *Structural and Multidisciplinary Optimization,* Vol. 21, pp. 90-108.

Rumelhart, D. E., Widrow, B. and Lehr, M. A., 1994, "The Basic Ideas in Neural Networks," *Communications of the ACM,* Vol. 37, No. 3 (March), pp. 87-92.

Sacks, J. and Schiller, S., 1988, "Spatial Designs," *Statistical Decision Theory and Related Topics (Gupta, S. S. and Berger, J. O., eds.),* Springer-Verlag, New York, pp. 385-399.

Sacks, J., Welch, W. J., Mitchell, T. J. and Wynn, H. P., 1989a, "Design and Analysis of Computer Experiments," *Statistical Science,* Vol. 4, No. 4, pp. 409-435.

Sacks, J., Schiller, S. B. and Welch, W. J., 1989b, "Designs for Computer Experiments," *Technometrics,* Vol. 31, No. 1, February, pp. 41-47.

Sandgren, E., 1990, "Nonlinear Integer and Discrete Programming in Mechanical Design Optimization," *Journal of Mechanical Design*, Vol. 112, No. 2, pp. 223-229.

Sasena, M. J., 1998, *Optimization of Computer Simulations via Smoothing Splines and Kriging Metamodels*, M.S. Thesis, Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI.

Sasena, M J, Papalambros, P.Y., and Goovaerts, P., 2002, "Exploration of Metamodeling Sampling Criteria for Constrained Global Optimization," *Engineering Optimization*, 34(3):263–278, 2002.

Sayers, M.W. and Karamihas, S.M., 1998, *The Little Book of Profiling: Basic Information about Measuring and Interpreting Road Profiles*, The Regent of the University of Michigan.

Schonlau, M., 1997, *Computer Experiments and Global Optimization*, Doctoral thesis, University of Waterloo, Department of Statistics, Ontario, Canada.

Schonlau, M., Welch, W. J. and Jones, D. R., 1997, "Global Versus Local Search in Constrained Optimization of Computer Models," *Technical Report RR-97-11,* to appear in *New Developments and Applications in Experimental Design (Fluornoy, N., et al., Eds.),* Institute for Mathematical Statistics, Institute for Improvement in Quality and Productivity, University of Waterloo, Waterloo, Ontario, Canada.

Scott, C. H. and Jefferson, T. R., 1977, "Entropy Maximizing Models of Residential Location via Geometric Programming," *Geographical Analysis*, Vol. 9, pp.181-187.

Sebastiani P. and Wynn H.P., 1997, "Bayesian Experimental Design and Shannon Information," *Statistical Research Report*, No.17, October 1997.

Sebastiani, P. and Wynn, H. P., 2000, "Maximum Entropy Sampling and Optimal Bayesian Experimental Design," *J. R. Statist. Soc. B (2000)*, Vol. 62, Part 1, pp.145-157.

Sebastiani, P. and Wynn, H. P., 2001, "Experimental Design to Maximize Information," *Twentieth International Workshop on Bayesian Inference and Maximum Entropy in Science and Engineering, AIP Conference Proceedings*, pp.192-203.

Seepersad, C. C., B. M. Dempsey, J. K. Allen, F. Mistree and D. L. McDowell, 2003, "Design of Multifunctional Honeycomb Materials," *AIAA Journal*.

Seepersad, C.C., Dempsey, B.M., Allen, J.K., Mistree, F. and McDowell, D.L., 2002, "Design of Multifunctional Honeycomb Materials," *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, AIAA, Paper Number AIAA-2002-5626.

Shannon, C. E., 1948, "A Mathematical Theory of Communication," *Bell System Tech. J.*, 27:379–423, 623–659.

Shannon, C. E. and Weaver, W., 1962, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, Illinois.

Shewry, M. C. and Wynn, H. P., 1987, "Maximum Entropy Sampling," *Journal of Applied Statistics*, Vol.14, No.2, pp.165-170.

Shewry, M. C. and Wynn, H. P., 1988, "Maximum Entropy Sampling with Application to Simulation Codes," *Proceedings of the 12th World Congress on Scientific Computation*, IMAC88, Vol.2, pp.517-519.

Shoemaker, A. C., Tsui, K. L. and Wu, J., 1991, "Economical Experimentation Methods for Robust Design," *Technometrics,* Vol. 33, No. 4, pp. 415-427.

Shore, J. E., "Minimum Cross-entropy Spectral Analysis," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-29, pp.230-237.

Shupe, J. A., Muster, D., Allen, J. K. and Mistree, F., 1988, "Decision-Based Design: Some Concepts and Research Issues," *Expert Systems, Strategies and Solutions in Manufacturing Design and Planning*, Kusiak, A., ed., Society of Manufacturing Engineeris, Dearborn, Michigan, pp. 3-37 (Chapter 1).

Sigmund, O., 2001, "A 99 Line Topology Optimization Code Written in Matlab," *Structural Multidisciplinary Optimization,* Vol. 21, pp. 120-127.

Simon, H. A., 1982, "Models of bounded rationality," Cambridge, Massachusetts: MIT Press, Vol. 2, 1982.

Simpson, T. W., 1995, *Development of a Design Process for Realizing Open Engineering Systems*, MS Thesis, The G. W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA.

Simpson, T. W., 1998, *A Concept Exploration Method for Product Family Design*, Ph.D. Dissertation, The Georgia W. Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA.

Simpson, T. W., Chen, W., Allen, J. K. and Mistree, F., 1996, September 4-6, "Conceptual Design of a Family of Products Through the Use of the Robust Concept Exploration Method," *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Bellevue, WA, AIAA, Vol. 2, pp. 1535-1545. AIAA-96-4161-CP.

Simpson, T. W., Lautenschlager, U., and Mistree, F., 1997a, "Mass Customization in the Age of Information: The Case for Open Engineering Systems," *The Information Revolution: Present and Future (Read, W. H., and Porter, A. L., eds.)*, Ablex Publishing, Greenwich, CT, pp. 49-71.

Simpson, T.W., Peplinski, J., Koch, P.N. and Allen, J.K., 1997b, "On the Use of Statistics in Design and the Implications for Deterministic Computer Experiments," *ASME Design Engineering Technical Conferences*, Sacramento, CA Paper No. DETC97/DTM3881.

Simpson, T. W., Chen, W., Allen, J. K. and Mistree, F., 1997c, October 13-16, "Designing Ranged Sets of Top-Level Design Specifications for a Family of Aircraft: An Application of Design Capability Indices," *SAE World Aviation Congress and Exposition,* Anaheim, CA, AIAA-97-5513.

Smith, W. F. and Mistree, F., 1994, May 24-27, "The Development of Top-Level Ship Specifications: A Decision-Based Approach," *5th International Conference on Marine Design,* Delft, The Netherlands, pp. 59-76.

Steinberg, D. M., 1985, "Model Robust Response Surface Designs: Scaling Two-Level Factorials," *Biometrika*, Vol. 72, pp.513-26.

Stone, M., 1959, "Application of a Measure of Information to the Design and Comparison of Regression Experiments," *Ann. Math. Statist.*, Vol. 30, pp.55-70.

Su, J. and Renaud, J. E., 1996, September 4-6, "Automatic Differentiation in Robust Optimization," *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Bellevue, WA, AIAA, Vol. 1, pp. 201-215. AIAA-96-4005-CP.

Suh, N. P., 1990, *Principles of Design*, Oxford University Press, Oxford, U.K.

Sundaresan, S., Isshii, K. and Houser, D.R., 1993, "A Robust Optimization Procedure with Variations on Design Variables and Constraints," *Advances in Design Automation*, ASME DE-Vol. 69-1, pp. 379-386.

Taguchi, G., 1978, "Off-Line and On-Line Quality Control Systems," *Proceedings of International Conference on Quality Control*, Tokyo, Japan.

Taguchi, G., 1987, *System of Experimental Design: Engineering Methods to Optimize Quality and Minimize Costs*, UNIPUB/Kraus International Publications.

Taguchi, G., Elsayad, E. A. and Hsiang, T., 1989, *Quality Engineering in Production Systems*, McGraw-Hill, New York.

Toropov, V., van Keulen, F., Markine, V. and de Doer, H., 1996, September 4-6, "Refinements in the Multi-Point Approximation Method to Reduce the Effects of Noisy Structural Responses," *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Bellevue, WA, AIAA, Vol. 2, pp. 941-951. AIAA-96-4087-CP.

Torquato, S., Gibiansky, L. V., Silva, M. J. and Gibson, L. J., 1998, "Effective Mechanical and Transport Properties of Cellular Solids," *International Journal of Mechanical Sciences,* Vol. 40, No. 1, 1998, pp. 71-82.

Tribus, M., 1969, *Rational Descriptions, Decisions, and Designs*, Pergamon Press, New York.

Tribus, M. and Szonyi, G., 1989, "An Alternative View of the Taguchi Approach," *Quality Progress,* Vol. 22, No. 5, pp. 46-52.

Trosset, M. W. and Torczon, V., 1997, "Numerical Optimization Using Computer Experiments," *Report No. TR97-02,* Department of Computational and Applied Mathematics, Rice University, Houston, TX.

Tsai, J. C., 2002, *Statistical Modeling of the Value Function in High-Dimensional , Continuous-State SDP*, Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology.

Tsui, K–L., 1992, "An Overview of Taguchi Method and Newly Developed Statistical Methods for Robust Design," *IIE Transaction*, Vol. 24, No. 5, pp. 44-57.

Unal, R., Stanley, D.O., Engelund, W. and Lepsch, R., 1994, "Design for Quality using Response Surface Methods: An Alternative to Taguchi's Parameter Design Approach," *Engineering Management Journal*, Vol. 6 No. 3, pp. 40-48.

Vandenberghe, L., Boyd, S. and Wu, S. P., 1998, "Determinant Maximization with Linear Matrix Inequality Constraints," *SIAM Journal on Matrix Analysis and Application*, Vol. 19 (1998).

Venter, G., Haftka, R. T. and Starnes, J. H., Jr., 1996, "Construction of Response Sufaces for Design Optimization Applications," *6th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization,* Bellevue, WA, AIAA, Inc., pp. 548-564.

Wald, A., 1943, "On the Efficient Design of Statistical Investigations," *Ann. Math. Statist.*, Vol. 14, p134-140.

Wang, G., 2001, "Improvement on the Adaptive Response Surface Method for High-Dimensional Computation-Intensive Design Problems," *ASME Design Engineering Technical Conferences – Design Automation Conference*, Pittsburgh, PA, Paper No. DETC2001/DAC-21141.

Wang, G., 2003, "Adaptive Response Surface Method Using Inherited Latin Hypercube Designs," *ASME Journal of Mechanical Design*, 125(2), 210-220.

Wang, Y. and Lu, W., 1992, "Multicriterion Maximum Entropy Image Reconstruction from Projections," *European Journal of Operational Research*, Vol. 59, pp.324-329.

Wang, G. and Simpson, T.W., 2004, "Fuzzy Clustering Based Hierarchical Metamodeling for Design Space Reduction and Optimization," *Engineering Optimization*, Vol.36, No.3, June 2004, 313-335.

Welch, W.J., Buck, R.J., Sacks, J., Wynn, H.P., Mitchell, T.J. and Morris, M.D., 1992, "Screening, Predicting and Computer Experiments," *Technometrics*, 34(1), 15-25.

Welch, W. J., Yu, T.-K., Kang, S. M. and Sacks, J., 1990, "Computer Experiments for Quality Control by Parameter Design," *Journal of Quality Technology,* Vol. 22, No. 1, pp. 15-22.

Wijsman, R. A., 1973, "On the Attainment of the Cramer-Rao Lower Bound," *Ann. Statist.*, Vol. 1, pp.538-542.

Wilson, B., Cappelleri, D. J., Frecker, M. I. and Simpson, T. W., 2001, "Efficient Pareto Frontier Exploration Using Surrogate Approximations," Optimization and Engineering, 2:1 (31-50).

Wujek, B.A. and Renaud, J.E., 1998a, “New Adaptive Move-Limit Management Strategy for Approximate Optimization, Part 1,” *AIAA Journal*, 36(10), 1911-1921.

Wujek, B.A. and Renaud, J.E., 1998b, “New Adaptive Move-Limit Management Strategy for Approximate Optimization, Part 2,” *AIAA Journal*, 36(10), 1922-1934.

Wujek, B. A., Renaud, J. E., Batill, S. M. and Brockman, J. B., 1995, September 17-21, "Concurrent Subspace Optimization Using Design Variable Sharing in a Distributed Computing Environment," *Advances in Design Automation (Azarm, S., Dutta, D., et al., eds.),* Boston, MA, ASME, Vol. 82, pp. 181-188.

Yamada, M. and Rajasekera, J. R., 1993, “Portfolio Re-balancing with the Entropy Criteria,” *Report No.310, QUICK Research Institute Corp.*, Tokyo.

Ye, Q., 1997, *Orthogonal Latin Hypercubes and their Application in Computer Experiments*, Technical Report #305, University of Michigan.

Yu, J.-C. and Ishii, K., 1998, "Design Optimization for Robustness Using Quadrature Factorial Models," *Engineering Optimization,* Vol. 30, No. 3-4, pp. 203-225.

Zacks, S., 1996, “Adaptive Designs for Parametric Models,” *Handbook of Statistics (Ghosh, S. and Rao, C. R., eds.)*, Elsevier Science, New York, pp.151-180.