The Design and Analysis of Computer Experiments

Robert John Buck

as fulfillment for a Ph. D. at
City University, London, U. K.

Department of Statistics

May 1993

Table of Contents

List of Tables

List of Figures

Acknowledgements

I have received support at many levels and from many people during my studies all of which has been greatly appreciated. I would like to thank my supervisor Professor Henry Wynn for encouraging me to turn my research into a Ph.D. thesis and for his support and advice. I would also like to thank Professor Jerry Sacks for getting me started in an interesting field of research and teaching me how research in statistics gets done. I would also like to thank Professor Will Welch for his help in keeping me organized, truly an undervalued skill, and his valued instruction in statistics and computing.

Financially I was supported by a number of organizations during my work and would like to acknowledge them as well. I have been supported in the U.S. by the AT&T Affiliates Program, by Intel Corp., by AFOSR and NSF through DMS 86-09819, by NSA MDA 904-89-H-2011 and by Cray Research, Inc. through the National Center for Supercomputing Applications, University of Illinois. I have also been supported in the U.K. by the Science and Engineering Research Council and Mentor Graphics.

On a personal level, I would like to again thank Professor Henry Wynn, Professor Jerry Sacks, and Professor Will Welch for many enlightening conversations about a wide variety of topics both in and out of statistics. I would like to thank the many graduate students of the Biochemistry Department at the University of Illinois for making me an unofficial member of the department during my stay at the University. I would also like to thank Ron Bates, my colleague at City University, for being my mate and explaining the game of cricket.

Finally, I would like to thank my family for their love and total support in everything I have done. I would like to specifically thank my parents for continuing to give me the best of themselves. Most importantly I want to thank my wife, Cheryl, for her love and support and who is my closest and most cherished friend.

Declaration of Power of Discretion

I grant powers of discretion to the University Librarian to allow this thesis to be copied in whole or in part without further reference to me. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

ABSTRACT

Computer simulation modeling is an important part of engineering design. The process of creating quality products under variable conditions is called robust engineering design. Frequently these simulation models are expensive to run and it can take many runs to find the appropriate parameter settings of the engineering design. To reduce the cost of robust engineering design, it has been proposed that statistical models be used to predict the results of the simulator at unobserved values of the inputs. This involves running experiments on the computer simulation models, or computer experiments. Computer experiments have no random error and frequently involve a large number of experimental factors; because of this, there are many reasons why standard prediction and design methods may not work well.

Methods from spatial statistics, frequently refered to as kriging, are used to predict new observations of the simulation model. A generalized linear model with unknown covariance parameters is used on several examples of high dimensions. The model requires estimation of the covariance function parameters and methods are described for parameter estimation and model building.

Latin hypercube sampling is used for the experimental design. Latin hypercube sampling is as easy to use as Monte Carlo sampling and has been shown to have better estimation properties. The space filling properties of Latin hypercube sampling are investigated here and shown to fill the design space more uniformly than Monte Carlo sampling.

These statistical methods are applied to two circuit simulation models. The results show that these methods work well on computer experiments and can form the basis for a methodology in robust engineering design. Although these methods have been applied to circuit design problems, the methods are applicable to a wide variety of computer simulation models.

Robust engineering design received a great deal of attention when Taguchi's ideas were introduced. An investigation into the methods that Taguchi introduced revealed some shortcomings from a statistical view. General conclusions are drawn about the efficacy of traditional Taguchi methods compared with the prefered model-based approach of the thesis.

Key to Symbols and Abbreviations

DACE - Design and Analysis of Computer Experiments. Used here as the name for a robust engineering design methodology.

FORWARD - Name of algorithm for computing maximum likelihood estimates

LHS - Latin Hypercube Sampling

LM - Loss Method

MC&B - McKay, Conover, and Beckman

ONETIME - Name of algorithm for computing maximum likelihood estimates

RED - Robust Engineering Design

RSM - Response Surface Methodology

RM - Response Method

SN - Signal - Noise

SRS - simple random sample

STW - Shoemaker, Tsui and Wu

SWMW - Sacks, Welch, Mitchell, and Wynn

$\mathbf{s}_i, S$ - i^{th} row of experimental design S

$\mathbf{x}, \mathbf{X}$ - input value for prediction, deterministic and random.

$y(\mathbf{x}), Y(\mathbf{x})$ - response variable, deterministic and random.

$\mathbf{y}_S, \mathbf{Y}_S$ - data from experiment with design S.

$\hat{y}(\mathbf{x})$ - estimate of $y(\mathbf{x})$.

n, n_D, n_N, n_r - sample size: general, inner array, outer array, random sample.

$V(\mathbf{x}, \mathbf{w}), R(\mathbf{x}, \mathbf{w})$ - Variance and Correlation function of stochastic process.

R_S - Correlation matrix for design S

$(\theta, \mathbf{p})$ - Parameters for correlation function $R(\mathbf{x}, \mathbf{w})$

β_i - i^{th} coefficient of linear model.

$E_\theta(X)$ - Expectation of X over distribution with parameter θ.

$Var_\theta(X)$ - Variance of X over distribution with parameter θ.

-Chapter 1 -

Robust Engineering Design and Computer Experiments

1.1 Introduction

The power of computers has grown exponentially over the last decade. With the increase in power, the uses for computers has expanded rapidly. This is true particularly in the area of simulation modeling, where everything from the weather to airplanes are now being modeled on computers. Many problems being modeled have features that are not well known in reality. This leads to tinkering with the code or experimentation to get a better understanding of how the system being modeled works. Although computer power has been increasing, the expense of running many of these simulation models has increased even more rapidly and running the simulators can still be an expensive exercise. Experimental design and statistical prediction models can be used to minimize the number of runs of the simulation model, just as they do in physical or "real" experiments.

Another field that has developed rapidly at the same time is research into methods for building quality products, one aspect of which is robust engineering design. Research into methods for robust engineering design delve into a wide range of statistical topics from experimental design to data exploration, estimation and prediction. This thesis investigates the application of statistics to research strategies for robust engineering design for computer simulation models. Robust engineering design is a natural application for statistical research on computer experiments because of the widespread use of computer simulation in engineering design and the importance of robust engineering design in industry today.

The thesis has been organized so that those whose main interests are in either robust engineering design or design and analysis of computer experiments but not necessarily both may look at relevant sections without missing much. The reader most interested in robust engineering design is directed to Sections 1.3-1.5 and Chapters 5-7. Readers interested more in the statistical aspects of

the design and analysis of computer experiments should look to Section 1.2 and Chapters 2-4, and 6.

Sections 1.3-1.5 give an introduction to the problem of robust engineering design and have a brief overview of the main strategies for robust engineering design. Chapter 6 gives a detailed description of a strategy for robust engineering design developed from work on computer experiments and applies it to several examples. Chapter 7 looks more closely at the underlying problems in robust engineering design and discusses how the main strategies attempt to approximate the system and find solutions.

Section 1.2 gives an introduction to computer experiments and why new methods of analysis are useful for this field of research. Chapter 2 gives an overview of an area of spatial statistics known as kriging and defines the statistical model used throughout this thesis. Chapter 3 describes some methods in parameter estimation and applies the statistical models described in Chapter 2 to some examples. Chapter 4 discusses computer experiments and experimental design, in particular the use of Latin hypercube sampling, and develops some new theory on the space filling properties of Latin hypercube sampling. Chapter 6 as mentioned applies the methods of design and analysis of computer experiments to the problem of robust engineering design.

1.2 Computer Experiments - An Introduction

An excellent discussion of statistics and computer experiments is given in Sacks, Welch, Mitchell, and Wynn[1] and a summary of their comments would be useful. Interest in computer experiments is growing and numerous examples have begun to reach the literature. Naturally, traditional statistical methods have typically been applied to these early examples, but it is useful to restate the objectives of computer experiments and examine how computer experiments may differ from physical experiments.

The single factor which differentiates computer experiments from physical experiments is random error. Computer experiments, unlike physical experiments, do not have a random error component unless it has been explicitly added to the code via a random number generator. Despite similar goals, the absence of random error creates some important differences with physical experiments.

• The classic ideas in experimental design on blocking, replication and randomization are not applicable.

- The full complexity of the computer model is measurable.
- The error of prediction is due solely to systematic bias.
- The distributional assumptions of least squares models are irrelevant.

There are a large number of objectives for computer experiments, however the following could be considered the primary ones:

1. Prediction at untried inputs.
2. Optimize the response or a function of the response.
3. Tune the code to physical data.

Prediction can be thought of as a primary objective, since if one is able to predict the simulation model accurately and inexpensively, the predictor can be used as a cheap substitute in further studies such as optimization.

If prediction is taken as the main objective, the primary statistical questions are:

1. For what values of the inputs should data be collected, and how many?
2. How should the data be used to most accurately estimate or predict new observations of the simulation model?

As mentioned classic statistical methods have been applied to computer experiments, most notably least squares fitting. Since there is no random error to mask the complexity of the simulation model and simulation models are unlikely to have simple low order polynomial response surfaces, it is not surprising that Iman and Helton[2] found many situations where the response surface was not estimated adequately by least squares models. The suggested alternative for prediction models and experimental design are discussed in Chapter 2 and 4 respectively.

1.3 Robust Engineering Design - An Overview

The problem of designing and building high quality products has been highly publicized in the last 10 to 15 years. A perceived dominance in quality products by Japanese manufacturers and the introduction of Taguchi methods during the late 1970's and early 1980's led to a flurry of activity by manufacturers to incorporate Taguchi's philosophy. It has also led to a critical review of Taguchi's methods and efforts to develop better optimization strategies.

Robust Engineering Design (RED) is the process of designing a product that will perform well under variable conditions. The variability may be due to any

of a large number of possible sources: manufacturing variability, environmental variability, or product degradation over time. The design process is increasingly carried out on computers, using computer-aided design/computer-aided engineering (CAD/CAE) tools. The ideas of RED remain the same, but many of the differences between physical and computer experiments are reflected in differences between computer design and physical design problems.

The goal of researchers studying the process of RED is to develop a strategy that will make RED simple and efficient. The rest of this chapter describes the work that has been done to achieve this goal and reviews several currently popular methods. A discussion of some of the difficulties in implementing RED and how these methods attempt to resolve them can be found in Chapter 7.

All strategies in RED share many of the same ideas and definitions. First, the designer needs to define an appropriate measure of performance and create a list of variables or factors they feel will influence the performance of the product. Engineers commonly refer to these factors as parameters, but this will be avoided here because of possible confusion with the use of the word parameter as used in statistics. After observing the product's response at preselected conditions the designer can use estimates of the product's performance to select factor values to improve the product's performance under variable conditions.

Let $\mathbf{X}=(X_1,...,X_d)$ denote the d-dimensional vector of input factors the designer wishes to vary in the computer simulation model. Once the factor list, represented by $\mathbf{X}$, is obtained they need to be split into two categories, design and noise factors. *Design factors* are used by the designer to develop the product. *Noise factors* are variables that the designer has control of only during the design stage, e.g. manufacturing variability, or environmental variability. Once past the design stage these factors are not controllable and should be considered random variables. Some design factors may have variations from noise factors superimposed. We write $X_i=c_i+U_i$ to differentiate between the design factors, c_i, and noise factors, U_i, that may make up X_i. If an input factor is not a design factor, then c_i has a fixed value (make it 0) and is ignored. Similarly, if there is no noise factor component of X_i, then $U_i=0$. The performance y is, therefore, a function of $\mathbf{X}=\mathbf{c}+\mathbf{U}$, where $\mathbf{c}=(c_1,...,c_d)$ and $\mathbf{U}=(U_1,...,U_d)$.

Generally, non-additive U's can be dealt with by treating them as factors with no designable adjustment. In specific cases other routes may be available.

For example, to accommodate multiplicative variation, write $X_i = c_i U_i$, or work on a logarithmic scale to produce additive variation.

After some investigation the designer may want to subdivide the design factors into two further categories: location factors and dispersion factors. *Location factors* are design factors that have little influence on the amount of variability produced by the noise factors but affect the mean response. *Dispersion factors* are design factors that do influence product variability. One of the crucial aspects of all strategies in RED is how to identitify and investigate the interaction between dispersion factors and noise factors.

Let us investigate the mathematical formulation of the RED problem in more detail. In a physical system let

$$\mathbf{Y} = f(\mathbf{X}) + e(\mathbf{c}), \tag{1.3.1}$$

represent the outputs of the product or process under study. The output or response, $\mathbf{Y}$, is a random vector and the variability comes from two sources: $e(\mathbf{c})$ which is the random or measurement error and $\mathbf{U}$, the noise factors. Also note that the random error could be a function of the design factors and modeling $e(\mathbf{c})$ could help to improve product robustness. When using simulation models to emulate the physical system there is no measurement error and $e(\mathbf{c})$ does not need to be included in (1.3.1) and all variability is due to the noise factors.

The goal of RED is to find input factor settings so the response attains a stated target. In reality, the response is not an individual item, but a population of manufactured items operating under a range of possible conditions. Let Ω_D be the sample space for the design factors and Ω_N be the sample space for the noise factors. Then $\Omega = \Omega_D \times \Omega_N$ is the sample space for $\mathbf{X}$. RED studies are concerned with trying to find values of $\mathbf{c}$ so that all events in Ω_N attain the stated target. This is an unrealistic goal, but we can try to minimize the variation of this population around the target.

The variability of the population around the target needs to be measured so the appropriate levels for the design factors can be chosen. This variability is measured and summarized by the loss function and risk function. A function that measures the performance of a product under specific conditions is usually referred to as a *loss function*. Many features of a product's performance may make up the loss function, e.g. customer satisfaction and cost. A common

example of a loss function, $l(y)$, is the quadratic loss function, $(y-t)^2$, where y is the product response and t is the target the designer is trying to meet. The designer typically is not interested in results under specific conditions but in the product's performance under varying conditions. A function that defines how well the product performs, as measured by the loss function, under varying conditions is usually known as a *risk function* or performance measure. The goal of the designer is to minimize the risk function by proper selection of **c**. Two common forms of risk function are expected loss, $E_U(l(y))$, and maximum loss, $\max_U l(y)$. Computing the risk function requires knowing both $f(\mathbf{X})$ in (1.3.1) and the distribution of **U**. Since one or both are usually unknown the risk function is estimated and referred to as *estimated risk* or a performance statistic. When **y** is a vector, loss functions and risk functions are needed for each component of **y** and minimizing risk becomes a multivariate decision making problem.

The quadratic loss function is a commonly used loss function. When expected loss is used as the risk function in conjunction with the quadratic loss function, there are two ways to try to minimize the risk function. The risk function is the mean squared error of Y, $MSE(Y)$, and can be written as:

$$MSE(Y) = E_U(Y-t)^2 = Var_U(Y) + (E_U(Y)-t)^2,$$

where t is the target and Y is a random variable because it is a function of U. One approach is to use the dispersion factors to minimize $Var_U(Y)$ and then use the location factors to adjust to target, i.e. eliminate bias. This approach in the form described is called the *dual response approach*[3] because it usually involves modeling both the mean and variance of Y. The Taguchi method could be considered a variation of the dual response approach because the approach to minimizing risk is the same: minimize variability using dispersion factors and then adjust to target with the design factors. The second approach is to minimize $MSE(Y)$ directly, in which case identification of location and dispersion factors is not essential. This approach can not be used unless the response Y is known or estimated. The implementation and benefits of these two optimization methods will be discussed briefly later in this chapter and in Chapter 7.

To summarize, the goal of the designer is to minimize the variability of the population around a target. A robust product design is found by choosing dispersion factor settings that minimize this variability and using the location

factors to keep the product response on target. All strategies in RED follow this general philosophy either explicitly or implicitly.

Strategies for RED can be classified in two general categories. Shoemaker, Tsui, and Wu (STW)[4] refer to these general approaches as the loss model (LM) approach and the response model (RM) approach. The LM approach uses an experimental plan that allows risk to be estimated directly from experimental observations. The "Taguchi" method is the classic example of the LM approach. The RM approach does not attempt to directly estimate the risk, but instead concentrates on accurate prediction of the response using statistical models, or predictors. The RM approach uses *estimates* of the response rather than observed response values at a specific product factor setting to estimate risk.

1.4 Loss Model Approach ("Taguchi" Method)

The "Taguchi" method [5] [6] is a commonly used tool in many areas where quality improvement is an issue. Kacker[7] gives a good overview of the method, while Phadke and Dehnad [8] and Box, *et al.* [9] give some good additional comments and criticisms and Kacker and Shoemaker [10] and Phadke [11] provide more examples. The "Taguchi" method is an easily implemented procedure, which helped the spread of RED ideas in industry.

The Taguchi method can be briefly summarized by a few basic ideas. An experimental plan is developed so product variability can be measured at each setting of the design factors in the experimental plan. The settings of the design factors are usually deviations from a nominal or "working" set of design factor values. The designer can determine which design factors influence product variability by identifying which design factors have a significant effect on the estimated variance over the range of input values. The settings of design factors that influence product variability are chosen to minimize the product variability. Those design factors that do not influence product variability are used to either adjust the mean product response to its performance target or to save on costs. Note that in the Taguchi method it is assumed that the location factors can be used to adjust to the performance target and this allows the performance target to be separated from the risk function.

To estimate risk directly from the experiment it is necessary to have multiple observations at each setting of the design factors. These observations are intended to mimic the variability that occurs in the real world. If noise factors

can be controlled by the designer an experimental plan can be used to determine specific settings for these noise factors. An experimental plan for noise factors increases the separation of noise factor input values allowing maximum dispersion of the noise factor settings, which implies maximum dispersion of the product response, in a minimal number of runs. If the noise factors can not be incorporated into an experimental design then replication is the only option. Putting the noise factors in an experimental plan helps to produce larger variability in the response than random sampling, but it does not lend itself to producing true estimates of product variability.

The experimental plan used to simulate "natural" variability is a product array formed of two subplans, the inner and outer arrays in Taguchi's terminology. The *inner array* is for the design factors and the *outer array* is for the noise factors. All interaction effects between design factors and noise factors are estimable when a product array is used for the experimental plan. Taguchi has published a series of orthogonal arrays for use in constructing product arrays and further work has been carried out by a large number of researchers to find more arrays. The two subplans form a product array by combining all outer array runs with each inner array run. The resulting experimental plan has a total of $n_D n_N$ runs, where n_D and n_N are the number of runs for the inner and outer arrays respectively.

Product arrays increase rapidly in size with an increase in the number of factors involved. To keep the number of runs as small as possible some design concessions are usually made. The most important concession is that the number of interaction effects between design factors or between noise factors that are estimable in the experimental design is kept to a minimum. Frequently the designs are Plackett-Burman[12] type designs, designs that are only able to estimate main effects. Transformations to produce additive models can be used to help eliminate interactions that may exist. Box [13] and Logothetis[14] discuss possible strategies.

Besides reducing the number of interactions that are estimable, replication and the number of factor levels are reduced to help keep the size of the experimental design small. This leads to the use of saturated or almost saturated designs which create special problems in uncovering important factors. There has been a considerable amount of work in this area because of the importance

of the Taguchi method and the desire to make the product array as small as possible. Berk and Picard[15] give a good overview to this area of research. The experimental plan is usually limited to 2-level orthogonal arrays since 3-level orthogonal arrays double the degrees of freedom per factor hence almost doubling the size of the experimental plan. The use of 2-level experimental plans limits the class of models that may be used to estimate the response.

Taguchi uses the Signal-Noise (SN) ratio to measure product variability. As the term implies the function is a ratio of the mean (the signal) and variance (the noise). The goal of the Taguchi method is to find settings of the dispersion factors that maximize the SN ratio, which is equivalent to minimizing the risk function. Once the experiment has been run, SN ratios are calculated for each run in the inner array. The designer can use these results to identify the dispersion factors and location factors. The dispersion factor settings are chosen to maximize the SN ratio. The location factor settings are chosen to adjust the product response to target. Usually the dispersion factor settings are selected from the finite set of experimental plan values. Discussion of the actual process of choosing location factor settings is curiously ignored in the literature. The new settings should give an immediate improvement in product robustness. Further experimentation, with the new settings as the new nominal points, can be done to make further improvements in product robustness.

Vining and Myers[3] state that Taguchi has developed over 60 variations of the Signal-Noise ratio. This is due to the need for different SN ratios for different combinations of objectives and model assumptions. León, Shoemaker, and Kacker[16] and Box[13] give a detailed review of Taguchi's SN ratio. León, Shoemaker, and Kacker[16] introduce PerMIAs, a generalized version of Taguchi's SN ratio. The conclusion that comes from this work is that one must be careful in considering which SN ratio to use otherwise the results will be of dubious value.

Vining and Myers[3] suggested the dual response approach, estimation of the mean and variance separately for each point of the inner array, to overcome many of the difficulties in using the SN ratio. This method also introduces more complicated models for estimating the response moving from ANOVA to regression techniques. They use standard linear regression models for estimating the mean and variance. Nelder and Lee[17] extend this idea by using generalized linear models to simultaneously model the mean and variance. The strategy

remains the same, identify location and dispersion factors, minimize variability and adjust to target.

1.5 Response Model Approach

Response model approaches use an estimate of the response to get an estimate of product variability. Two methods, Response Surface Methodology (RSM) and the strategy reviewed in Welch and Sacks[18] referred to here as DACE, are now currently in use.

The approaches are similar in that they both use estimates of the response to estimate product variability and they both use combined arrays for their experimental plans. Since the response rather than risk is being modeled, it is less important to have replication at the design factor settings and eliminates the need for the outer array used in the LM approach. Instead all factors, design and noise, can be put into a single experimental plan, the *combined array*. The use of combined arrays generates large savings in experimental runs.

The RM approach fits well with the CAD/CAE RED problem since there is no real product variability. Any product variability is controlled by the designer through the CAD/CAE software. Instead of using experimental runs to model variability directly, the RM approach strives to build a good predictor of the response surface then apply the variability to the predictor rather than the simulator. Since the predictor is cheaper to run than the simulator, more "replications" can be used to get an estimate of product variability when using the predictor.

The two methods differ most notably in the choice of model for estimating the response. RSM uses classic regression models while DACE uses a Gaussian stochastic process. This difference leads to different strategies in searching for optimal solutions. These differences will be discussed in detail in the overview of the two methods and in the discussion section. As described in the literature, they also differ in the approach to minimizing risk. RSM carries over the ideas of minimizing variance and adjusting to target as used in the LM approach while DACE tries to minimize $MSE(Y)$ directly. In practice either optimization approach could be used with either RSM or DACE.

1.5.1 Response Surface Methodology

Response Surface Methodology introduced by Box and Wilson[19] has a long statistical history. RSM predates the ideas of Taguchi and has thus not covered noise factors in Taguchi's sense. RSM is discussed in many texts.[20 21 22] The

extensive theory on RSM can readily be applied to RED. Myers[23] extends it to include noise factors and applies it to RED. An early application of RSM can be found in Alvarez, *et al.*[24]

RSM can be split into three stages: screening and identification, region seeking, and optimization. Small experiments are carried out to identify location and dispersion factors. The estimated linear models developed from these experiments are used to indicate in which direction the designer should search in the design factor space for design factor settings that produce a more robust product design. When a region is found which indicates no further searching is necessary, a final experiment is carried out to locate the optimal solution.

The design and analysis of the experiments are drawn from the work in linear regression models. The experimental plans typically suggested are two or three-level orthogonal arrays, frequently fractional factorials or central composite designs. The same ideas on transformations used in the LM approach are applicable here, especially to separate location and dispersion factors. The models used are classic linear regression models typically of first or second order. By the nature of the division of product factors into design and noise factors, it seems essential that three way interactions involving design and noise factors also need to be considered. STW[4] give a real example where three way interactions are significant.

If separate models are proposed for the mean response and response variance and a combined array is used for the experimental design, identifying dispersion factors can be difficult. Box and Meyer [25] and Nair and Pregibon[26] discuss methods for finding and estimating dispersion factors.

The model that is generated from screening experiments can be used to estimate product variability. This can be done easily from the the model if the noise factors are assumed to be independent random variables with mean 0 and variance σ_i^2. If the noise factors are not independently distributed it becomes much more difficult to model the variance from the regression model. Monte Carlo estimation can be used to estimate variance using the assumed noise factor distribution. The models for mean response and product variability can then be used to find improved product factor settings. This also can lead to new regions to investigate for further improvements.

1.5.2 DACE

The DACE method described in Bernardo, *et al.*[27] and outlined in Welch and Sacks[18] was developed from previous work on prediction and computer experiments which is reviewed in Sacks, Welch, Mitchell, and Wynn.[1] Examples can be found in the literature[27] and in Chapter 6.

The sequential experimentation plan can be summarized in six steps.

Step 1. Model the performance y and develop the loss function.

Step 2. Design an experiment and collect the data from the simulation runs.

Step 3. Use the data to estimate the parameters of the statistical model chosen in Step 1.

Step 4. Decompose $\hat{y}$ into effects due to individual factors and interaction effects.

Step 5. If the predictor is not accurate enough then select a smaller region where the optimal response is most likely to occur. Repeat Steps 2-5.

Step 6. When the predictor reaches a satisfactory level of accuracy optimize the chosen performance measure. Do a confirmatory run. Return to Step 5 if necessary.

The aim is to scan a large region of the input space for likely solutions and focus on these regions during subsequent experimentation to produce more accurate models of the response surface.

The experimental plan is generated using Latin Hypercube Sampling (LHS) which was introduced by McKay, Conover, and Beckman[28] for use in computer experiments. LHS is a readily implemented experimental plan with good space filling properties. Chapter 4 contains a discussion of some theoretical aspects of LHS.

For the statistical model we use a stochastic process of the form

$$Y(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x})\beta_i + Z(\mathbf{x}).$$

Where $Z(\mathbf{x})$ is a stochastic process with mean zero and correlation

$$Corr(Z(\mathbf{x}),Z(\mathbf{w})) = R(\mathbf{x},\mathbf{w})$$

between the responses at two inputs $\mathbf{x}$ and $\mathbf{w}$ and variance

$$Var(Z(\mathbf{x})) = \sigma^2.$$

A discussion of this model is given in Sacks, Welch, Mitchell, Wynn[1] (SWMW) and an overview is given in Chapter 2. There are many possible choices for R, some are discussed in Chapter 2. The correlation function used in the above work is

$$R(\mathbf{x},\mathbf{w}) = \prod_i exp(-\theta_i |x_i - w_i|^{p_i})$$

The stochastic process interpolates between design points and computes estimates of the response according to the correlation matrix R and the observations. No prior assumptions about interactions and nonlinearities need to be made.

The initial stages may not generate models that can predict the response accurately enough for the optimization procedure to pinpoint product factor settings. The predictor should be accurate enough to eliminate regions where the solution will not be found and subsequent regions will be selected to eliminate these areas from consideration. The plots in Step 3 listed above give visual information about the relationship between product factors and the response and can help guide the selection of a new subregion. A few reductions in the size of the region under study will produce an accurate predictor and optimization algorithms can then be used to locate the product factor settings that meet design specifications.

It is feasible to estimate product variability directly from the statistical model used in DACE by integrating over the noise factors using the relevant distribution. In most cases it will be more practical to use a Monte Carlo estimate of the variance or MSE to estimate product variability. An estimate of the distribution of the noise factors needs to be fully described to compute the estimate in either case. The cost of prediction is inexpensive, so an estimate of product variability can use hundreds of predictions.

1.6 References

1. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., (1989) "Design and Analysis of Computer Experiments," *Statistical Science*, 4, 409-435.
2. Iman, R. L. and Helton, J. C., (1988) "An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models," *Risk Analysis*, 8, 71-90.

3. Vining, G. G. and R. H. Myers, (1990) "Combining Taguchi and Response Surface Philosophies: A Dual Response Approach," *Journal of Quality Technology*, 22, 38-45.

4. Shoemaker, A. C., Tsui, K.-L., and Wu, C. F. J., (1991) "Economical Experimentation Methods for Robust Design," *Technometrics*, 33, 415-428.

5. Taguchi, G. and Y. I. Wu, (1979) *Introduction to Off-Line Quality Control,* Central Japan Quality Control Association.

6. Taguchi, G., (1986) *Introduction to Quality Engineering,* Asian Productivity Organization, Tokyo.

7. Kacker, R. N., (1985) "Off-line Quality Control, Parameter Design and the Taguchi Method (with discussion)," *Journal of Quality Technology*, 17, 176-209.

8. Phadke, M. S. and Dehnad, K., (1988) "Optimization of Product and Process Design for Quality and Cost," *Quality and Reliability Engineering International*, 4, 105-112.

9. Box, G. E. P., Bisgaard, S., and Fung, C., (1988) "An Explanation and Critique of Taguchi's Contributions to Quality Engineering," *Quality and Reliability Engineering International*, 4, 123-131.

10. Kacker, R. N. and Shoemaker, A. C., (1986) "Robust Design: A Cost-Effective Method for Improving Maunfacturing Processes," *AT&T Technical Journal*, 65, 39-50.

11. Phadke, M. S., (1986) "Design Optimization Case Studies," *AT&T Technical Journal*, 65, 51-68.

12. Plackett, R. L. and Burman, J. P., (1946) "The Design of Optimum Multifactorial Experiments," *Biometrika*, 33, 305-325.

13. Box, G. E. P., (1988) "Signal to Noise Ratios, Performance Criteria and Transformations (with discussion)," *Technometrics*, 30, 1-40.

14. Logothetis, N., (1990) "Box-Cox Transformations and the Taguchi Method," *Applied Statistics*, 39, 31-48.

15. Berk, K. N. and Picard, R. R. , (1991) "Significance Tests for Saturated Orthogonal Arrays," *Journal of Quality Technology*, 23, 79-89.

16. León, R. V., Shoemaker, A. C., and Kacker, R. N., (1987) "Performance Measures Independent of Adjustment," *Technometrics*, 29, 253-265.

17. Nelder, J. A. and Lee, Y., (1991) ''Generalized Linear Models for the Analysis of Taguchi-type Experiments,'' *Applied Stochastic Models and Data Analysis*, 7, 107-120.

18. Welch, W. J. and Sacks, J., (1991) ''A System for Quality Improvement Via Computer Experiments,'' *Communications in Statistics-Theory and Methods*, 20, 477-496.

19. Box, G. E. P. and Wilson, K. B., (1951) ''On the Experimental Attainment of Optimum Conditions (with discussion),'' *Journal of the Royal Statistical Society - Series B*, 13, 1-45.

20. Box, G. E. P. and Draper, N. R., (1987) *Empirical Model Building and Response Surfaces* , John Wiley & Sons , New York, N.Y. .

21. Khuri, A. I. and Cornell, J. A., (1987) *Response Surfaces: Design and Analysis*, Marcel Dekker, New York, N.Y..

22. Box, G. E. P., Hunter, W. G., and Hunter, J. S., (1978) *Statistics for Experimenters* , John Wiley , New York .

23. Myers, R. H., (1991) ''Response Surface Methodology in Quality Improvement,'' *Communications in Statistics-Theory and Methods*, 20, 457-476.

24. Alvarez, A. R., Abdi, B. L., Young, D. L., Weed, H. D., Teplik, J., and Herald, E. R., (1988) ''Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design,'' *IEEE Transactions on Computer-Aided Design*, 7, 272-288.

25. Box, G. E. P. and Meyer, R. D., (1986) ''Dispersion Effects from Fractional Designs,'' *Technometrics*, 28, 19-27.

26. Nair, V. N. and Pregibon, D., (1988) ''Analyzing Dispersion Effects from Replicated Factorial Experiments,'' *Technometrics*, 30, 247-257.

27. Bernardo, M. C., Buck, R., Liu, L., Nazaret, W. A., Sacks, J., and Welch, W. J., (1992) ''Integrated Circuit Design Optimization Using a Sequential Strategy,'' *IEEE Transactions in Computer-Aided Design*, 11, 361-372.

28. McKay, M. D., Conover, W. J., and Beckman, R. J., (1979) ''A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code,'' *Technometrics*, 21, 239-245.

- Chapter 2 -

Spatial Statistics

2.1 Introduction

In Section 1.2 the nature of computer simulation models and how they invite the same types of questions as experiments in the physical world was discussed. This similarity in research questions would certainly invite the use of the same techniques. There are some differences between computer experiments and physical experiments that make the use of classic statistical techniques for computer experiments, such as linear models, suspect. The most important of these differences is that the output of a computer experiment is not affected by random error, the same input will always give the same output. These differences have been discussed in Section 1.2.

Since computer simulation models are typically fairly complex, nonlinear systems of equations it is unlikely that polynomials, especially low order polynomials favored in regression, will estimate the response surface accurately. There are several examples in the literature that emphasize this point.[1] [2] [3] Also, the discrepency between the true response surface and the predicted surface using polynomial models is due to bias and an increase in sample size will not be helpful in reducing this error. Another drawback of linear models is that they are not interpolators, i.e. a model where $\hat{y}(\mathbf{s}) = y(\mathbf{s})$ when $\mathbf{s}$ is a point in the experimental design, so estimates at the design points do not necessarily equal the response, which is known to be the true value.

The response surface which the simulation model produces is not random, however we can assume that y is a realization of a stochastic process. Then measures of uncertainty can be made. What is needed is a statistical model that has the following properties:

1. An ability to model complex surfaces.
2. The model is an interpolating predictor.
3. Developed theory and applications for realizations of random functions, or stochastic processes are available.

Statistical models from the field of spatial statistics have these properties, although the models have typically been studied in only two or three dimensions.

In Section 2.2 the statistical model used for this thesis is described. Section 2.3 is a discussion of some correlation functions that could be used with the model described in Section 2.2. Section 2.4 contains a description of a version of ANOVA main effects used with stochastic process and Section 2.5 describes some robustness properties of the model.

2.2 Statistical Model used in Thesis

The model treats the deterministic response $y(\mathbf{x})$ as a realization of a stochastic process, $Y(\mathbf{x})$, and has the form

$$Y(\mathbf{x}) = \sum_{i=1}^{k} f_i(\mathbf{x})\beta_i + Z(\mathbf{x}). \tag{2.2.1}$$

The stochastic process $Z(\mathbf{x})$ is assumed to have mean zero and covariance

$$V(\mathbf{w},\mathbf{x}) = \sigma^2 R(\mathbf{w},\mathbf{x})$$

between $Z(\mathbf{w})$ and $Z(\mathbf{x})$, where σ^2 is the process variance and $R(\mathbf{w},\mathbf{x})$ is the correlation.

Before deriving estimates for the parameters and $y(\mathbf{x})$, more notation needs to be defined. Let S be the experimental design with elements s_{ij}, $i = 1, \ldots, n$ and $j = 1, \ldots, d$, where n is the number of runs in the design and d is the number of factors in the experiment. Let $\mathbf{s}_i$ be the i^{th} row of S. Let $\mathbf{y}_S$ be the vector of responses for the design S. Let

$$R_S = \{R(\mathbf{s}_i, \mathbf{s}_j)\},\ 1 \le i \le n, 1 \le j \le n$$

be the correlation matrix for the stochastic process Z at the design sites and

$$r(\mathbf{x}) = [R(\mathbf{s}_1, \mathbf{x}), \ldots, R(\mathbf{s}_n, \mathbf{x})]'$$

be the correlations between the Z's at the design points and an untried input $\mathbf{x}$.

Let $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_k]'$ be the $k \times 1$ vector of coefficients for the linear model. Let the k functions in the regression at an untried input $\mathbf{x}$ be written as

$$f(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})]'$$

and let the $n \times k$ regression design matrix be written as

$$F = \begin{bmatrix} f'(\mathbf{s}_1) \\ \vdots \\ f'(\mathbf{s}_n) \end{bmatrix}.$$

There are two methods that can be used to develop a predictor for this model, a frequentist approach or a Bayesian approach. If a Gaussian distribution is assumed, which is the case for the work in this thesis, the two approaches produce the same results given an improper prior distribution on the β's for the Bayesian approach.

2.2.1 Best Linear Unbiased Predictor

One method of analysis of this class of models is found in the field of spatial statistics and is known as kriging.[4] A current review of the subject can be found in Cressie.[5] Given a design S and the response $\mathbf{y}_S$ and assuming that the β's and σ^2 are unknown but $R(\mathbf{x},\mathbf{w})$ is known, consider the linear predictor

$$\hat{y}(\mathbf{x}) = c'(\mathbf{x})\,\mathbf{y}_S$$

of $y(\mathbf{x})$. If the frequentist view is held, one can replace $\mathbf{y}_S$ by the random vector $\mathbf{Y}_S$ and treat $\hat{y}(\mathbf{x})$ as random. The best linear unbiased predictor (BLUP) is obtained by finding $c(\mathbf{x})$, an $n\times 1$ vector, which minimizes

$$MSE[\hat{y}(\mathbf{x})] = E[c'(\mathbf{x})\mathbf{Y}_S - Y(\mathbf{x})]^2$$

subject to the unbiasedness condition

$$E[c'(\mathbf{x})\mathbf{Y}_S] = E[Y(\mathbf{x})].$$

which gives

$$c'(\mathbf{x})F - f(\mathbf{x}) = 0.$$

The MSE can be rewritten as

$$MSE[\hat{y}(\mathbf{x})] = Var(Y(\mathbf{x})) + Var(\hat{Y}(\mathbf{x})) - 2Cov(Y(\mathbf{x}), \hat{Y}(\mathbf{x}))$$

$$= \sigma^2[1 + c'(\mathbf{x})R_S c(\mathbf{x}) - 2c'r(\mathbf{x})].$$

Therefore the BLUP is found by minimizing

$$c'(\mathbf{x})R_S c(\mathbf{x}) - 2c'r(\mathbf{x})$$

subject to

$$F'c(\mathbf{x}) - f(\mathbf{x}) = 0.$$

Let $\lambda(\mathbf{x})$ a $k\times 1$ vector be the Lagrange multipliers needed for the constrained minimization of the MSE. The Lagrangian function is

$$L_M = c'(\mathbf{x})R_S c(\mathbf{x}) - 2c'r(\mathbf{x}) + \lambda'(\mathbf{x})(F'c(\mathbf{x}) - f'(\mathbf{x}))$$

and using matrix differentiation

$$\frac{\partial L_M}{\partial c(\mathbf{x})} = 2R_S c(\mathbf{x}) - 2r(\mathbf{x}) + F\lambda(\mathbf{x}) = 0.$$

Solving for $c(\mathbf{x})$ gives

(2.2.2) $$c(\mathbf{x}) = R_S^{-1}(r(\mathbf{x}) - 1/2F\lambda(\mathbf{x}))$$

and using $F'c(\mathbf{x}) = f(\mathbf{x})$ to solve for $\lambda(\mathbf{x})$ gives

$$\lambda(\mathbf{x}) = 2(F'R_S^{-1}F)^{-1}(F'R_S^{-1}r(\mathbf{x}) - f(\mathbf{x})).$$

Substituting for $\lambda(\mathbf{x})$ in (2.2.2)

$$c(\mathbf{x}) = R_S^{-1}(r(\mathbf{x}) - F(F'R_S^{-1}F)^{-1}(F'R_S^{-1}r(\mathbf{x}) - f(\mathbf{x}))$$

and the predictor is

(2.2.3) $$\hat{y}(\mathbf{x}) = f'(\mathbf{x})\hat{\beta} + r'(\mathbf{x})R_S^{-1}(\mathbf{y}_S - F\hat{\beta})$$

where $\hat{\beta} = (F'R_S^{-1}F)^{-1}F'R_S^{-1}\mathbf{y}_S$ is the generalized least squares estimate of β.

These results show that the predictor is made up of two parts: the generalized least squares predictor and an interpolator through the residuals from the generalized least squares regression model. The mean squared error for $\hat{y}(\mathbf{x})$ is given by

(2.2.4) $$MSE(\hat{y}(\mathbf{x})) = \sigma^2 \left(1 - [f'(\mathbf{x}), r'(\mathbf{x})] \begin{bmatrix} 0 & F' \\ F & R_S \end{bmatrix}^{-1} \begin{bmatrix} f(\mathbf{x}) \\ r(\mathbf{x}) \end{bmatrix} \right).$$

Typically, the correlation parameters, $(\theta, \mathbf{p})$, are not known either. Zimmerman and Cressie[6] show that (2.2.3) is an unbiased estimator of $E(Y(\mathbf{x}))$ if it is assumed that the distribution of $(Y_S, Y(\mathbf{x}))$ is symmetric about its mean and that $(\theta, \mathbf{p})$ is an even and translation-invariant function of Y. This is true when $Y(\mathbf{x})$ is assumed to have a Gaussian distribution and the estimates of $(\theta, \mathbf{p})$ are the maximum likelihood estimates. However, the estimates of $MSE(\hat{y}(\mathbf{x}))$ will be biased.

2.2.2 Bayes Predictor

To develop a Bayes predictor for the model (2.2.1) assume that $Z(\mathbf{x})$ has known covariance and that the prior distribution on β is Gaussian with mean μ and covariance $\sigma_M^2 I$, where σ_M^2 is known. For simplicity, also assume that the prior on β is independent of $Z(\mathbf{x})$. By standard theory the best Bayes predictor for $y(\mathbf{x})$ is

(2.2.5) $$\hat{y}_B(\mathbf{x}) = E[Y(\mathbf{x}) \mid \mathbf{y}_S],$$

where $E[Y(\mathbf{x})|\mathbf{y}_S]$ is the expectation of $Y(\mathbf{x})$ conditioned on the data $\mathbf{y}_S$. The multivariate distribution of $(Y(\mathbf{x}),\mathbf{Y}_S)$ is multivariate Normal with mean

$$E\begin{bmatrix}\mathbf{Y}_S\\ Y(\mathbf{x})\end{bmatrix} = \begin{bmatrix}F\mu\\ f'(\mathbf{x})\mu\end{bmatrix}$$

and covariance

$$Cov\begin{bmatrix}\mathbf{Y}_S\\ Y(\mathbf{x})\end{bmatrix} = \begin{bmatrix}C & D\\ D' & E\end{bmatrix}$$

where

$$C = Cov(\mathbf{Y}_S) = R_S + \sigma_M^2 FF',$$

$$D = Cov(Y(\mathbf{x}),\mathbf{Y}_S) = r(\mathbf{x}) + \sigma_M^2 Ff(\mathbf{x})$$

and

$$E = Var(Y(\mathbf{x})) = \sigma^2 + \sigma_M^2 f'(\mathbf{x})f(\mathbf{x}).$$

Then the conditional mean of $Y(\mathbf{x})$ given $\mathbf{Y}_S = \mathbf{y}_S$ is

(2.2.6) $$E[Y(\mathbf{x})|\mathbf{y}_S] = f'(\mathbf{x})\mu + (r'(\mathbf{x}) + \sigma_M^2 f'F')(R_S + \sigma_M^2 FF')^{-1}(\mathbf{y}_S - F\mu)$$

The purpose of this section is to show that when $\sigma_M^2 \to \infty$, the Bayes predictor is the same as (2.2.3). To reduce the notation, let $f(\mathbf{x}) = f$ and $r(\mathbf{x}) = r$.

The inverted matrix in the second term on the right hand side of (2.2.6) can be rewritten as

$$(R_S + \sigma_M^2 FF')^{-1} = R_S^{-1} - \sigma_M^2 R_S^{-1} F(I + \sigma_M^2 F'R_S^{-1}F)^{-1}F'R_S^{-1}.$$

The inverted matrix in this equation can be rewritten as

(2.2.7) $$(I + \sigma_M^2 A)^{-1} = \frac{1}{\sigma_M^2}(I + \frac{1}{\sigma_M^2}A^{-1})^{-1}A^{-1},$$

where $A = F'R_S^{-1}F$. Since A is a positive definite matrix, for $\alpha = 1/\sigma_M^2 \to 0$

$$(I + \alpha A^{-1})^{-1} = I - \alpha A^{-1} + o(\alpha).$$

Rewriting (2.2.7) using this result, (2.2.6) can be rewritten as

(2.2.8) $$f'\mu + (r' + \sigma_M^2 f'F)(R_S^{-1} - CA^{-1}C' + \alpha CA^{-1}A^{-1}C' + o(\alpha))E,$$

where $C = R_S^{-1}F$, and $E = y_S - F\mu$. Now (2.2.8) can be rewritten as

(2.2.9) $$f'\mu + r'(R_S^{-1} - CA^{-1}C')E + \alpha r'CA^{-1}C'E + f'FCA^{-1}A^{1}C'E + o(\alpha),$$

since $\sigma_M^2 f'F(R_S^{-1} - CA^{-1}C')E = 0$. Taking the terms in (2.2.9) individually: $r'(R_S^{-1} - CA^{-1}C')E = r'R_S^{-1}y_S - r'R_S^{-1}F\hat{\beta}$, where $\hat{\beta} = (F'R_S^{-1}F)^{-1}F'R_S^{-1}y_S$, $f'FCA^{-1}A^{-1}C'E = f'\hat{\beta} - f\mu$, and $\alpha r'CA^{-1}E + o(\alpha) \to 0$ as $\alpha \to 0$. So simplifying (2.2.8) and returning to original notation gives

$$E(Y(\mathbf{x}) \mid \mathbf{Y}_S) = r'R_S^{-1}y_S - r'R_S^{-1}F\hat{\beta} + f'\hat{\beta}$$

$$= f'\hat{\beta} + r'R_S^{-1}(y_S - F'\hat{\beta})$$

as $\sigma_M^2 \to \infty$, $(\alpha \to 0)$, which is the same as the best linear unbiased predictor in (2.2.3).

2.2.3 Maximum Likelihood Estimation

The predictors described all involve model parameters. Typically, some or all of the parameters are not known and need to be estimated. There are many ways to estimate these parameters. If R_S is known, then the parameters β and σ^2 could be estimated by least squares. Several methods of estimating unknown covariances are described in Cressie.[5] A common method of estimation, which is used for the work in this thesis, is maximum likelihood estimation.

If the stochastic process is assumed to have a Gaussian distribution, then the density function is

$$(2\pi\sigma^2)^{-n/2} |R_S|^{-1/2} \exp\{\frac{-1}{2\sigma^2}(\mathbf{y}_S - F\beta)R_S^{-1}(\mathbf{y}_S - F\beta)\}$$

and the ln likelihood is

$$\ln L^* = -\frac{1}{2}[n \ln \sigma^2 + \ln |R_S| + (\mathbf{y}_S - F\beta)R_S^{-1}(\mathbf{y}_S - F\beta)/\sigma^2]. \quad (2.2.10)$$

The parameter estimates $\hat{\beta}$ and $\hat{\sigma}^2$ depend on the value of R_S hence on $(\theta, \mathbf{p})$. When R_S is known the maximum likelihood estimates of $\hat{\beta}$ and $\hat{\sigma}^2$ are

$$\hat{\beta} = (F'R_S^{-1}F)^{-1}F'R_S^{-1}\mathbf{y}_S$$

and

$$\hat{\sigma}^2 = \frac{1}{n}(y_S - F'\hat{\beta})'R_S^{-1}(\mathbf{y}_S - F'\hat{\beta}).$$

The estimate for β is the generalized least-squares estimate and $\hat{\sigma}^2$ is the standard MLE of σ^2 for a Gaussian distribution with a linear model and known covariance.

When R_S, or equivalently for the case here, when $(\theta, \mathbf{p})$ is not known maximum likelihood estimates are computed using an iterative procedure. First, an initial estimate of R_S is used to compute estimates of $\hat{\beta}$ and $\hat{\sigma}^2$. These estimates are substituted into (2.2.10) and the log likelihood can be rewritten as

$$\ln L = -\frac{1}{2}[n \ln \hat{\sigma}^2 + \ln |R_S|]. \tag{2.2.11}$$

Then (2.2.11) is minimized with respect to $(\boldsymbol{\theta}, \mathbf{p})$. These estimates of R_S are used to get new estimates of β and σ^2 and the procedure is repeated until $\ln L$ in (2.2.11) has reached a maximum. Further discussion on methods of computing maximum likelihood estimates is in Chapter 3.

2.3 Correlation Function

To compute estimates for the model given in the previous section a correlation function, $R(\mathbf{w}, \mathbf{x})$, needs to be specified. There is a large number of choices for $R(\mathbf{w}, \mathbf{x})$. The correlation function used here is from the stationary family of correlation functions, $R(\mathbf{w}, \mathbf{x}) = R(\mathbf{w} - \mathbf{x})$ and assumes that any non-stationary behavior can be modeled by the linear model part of the stochastic process. Also, we have chosen to restrict our choice of correlations to those which are products of one dimensional correlations, $R(\mathbf{w}, \mathbf{x}) = \prod R_j(w_j - x_j)$. One benefit of using this class of correlation functions is the simplification of some mathematical and computational problems. This is still a highly flexible family of correlation functions and is found to be adequate for predicting the response in most situations.

The correlation function used in the examples and study for this thesis is

$$R(\mathbf{w}, \mathbf{x}) = \prod_{j=1}^{d} \exp(-\theta_j |w_j - x_j|)^{p_j}, \tag{2.3.1}$$

where $\theta_j \geq 0$ and $1 \leq p_j \leq 2$. Other possible correlation functions are

$$R_l(\mathbf{w}, \mathbf{x}) = \prod_{j=1}^{d} (1 - \theta_j |w_j - x_j|)_+ \tag{2.3.2}$$

which gives a linear spline for the predicted response and

$$R_c(\mathbf{w}, \mathbf{x}) = \prod_{j=1}^{d} [1 - a_j (w_j - x_j)^2 + b_j |w_j - x_j|^3], \tag{2.3.3}$$

which for certain choices of a_j and b_j produce cubic spline predictors. Currin, *et al.*[3] compared the predictive ability for these correlation functions as well as others on several small examples. The empirical RMSE for the correlation

function (2.3.1) is consistently one of the best predictors of the correlation functions examined. Stein in his comments on SWMW[7] proposed the correlation function

$$\prod_{j=1}^{d} \frac{1}{\Gamma(\nu)2^{\nu-1}} (\alpha_j \mid w_j - x_j \mid)^{\nu} K_{\nu}(\alpha_j \mid w_j - x_j \mid), \tag{2.3.4}$$

where K_{ν} is a modified Bessel function of order ν. A comparison of this correlation function with (2.3.1) in the rejoinder to Stein in SWMW showed the predictive accuracy of the two correlation functions for the example tested was essentially the same.

Understanding the Parameters

The parameters in the correlation function (2.3.1) drive the predictor, especially when the linear model part of the stochastic process is assumed to be a constant. The shape of the prediction surface is not obvious from the values of the correlation parameters. The θ's and p's have different effects on the prediction surface. If $p = 2$ then the covariance function is infinitely differentiable and the prediction surface will be smooth which should be the case for most analytic functions. If $p < 2$ the covariance function is only once differentiable and the surface becomes rougher as $p \to 1$. For $p = 1$ the correlation function is a product of Ornstein-Uhlenbeck processes, which are continuous but not very smooth. Smaller p also has the effect of "inflating the value" of θ.

The meaning of the value for θ is more difficult to read. For $\theta = 0$ the variable is not significant; if $\theta = \infty$ then the variable is uncorrelated. For values of θ between zero and infinity the effect is somewhat relative to other θ's and the data. For "small" θ's the main effects are linear. As θ increases the effect of x on the response becomes more nonlinear. "Large" θ's also can imply that the variable is part of an interaction term. It is difficult to determine whether a "large" θ is due to nonlinear main effects or interactions without plotting the main effects.

2.4 Estimating Main Effects and Interactions

Since the parameter values themselves give only limited insight into the shape of the response surface, plotting the main effects and interactions of the input factors is strongly advocated. These effects are the continuous version of the effects in classic ANOVA, but instead of averaging over the data the models are integrated over the design space. The overall mean, the average of $y(\mathbf{x})$

over the experimental region, is defined to be:

$$\mu_0 = \int y(\mathbf{x}) \prod_{k=1}^{d} dx_k . \tag{2.4.1}$$

The main effect for x_i is defined to be:

$$\mu_i(x_i) = \int y(\mathbf{x}) \prod_{k \neq i} dx_k - \mu_0, \tag{2.4.2}$$

Second order interactions of x_i and x_j are

$$\mu_{ij}(x_i, x_j) = \int y(\mathbf{x}) \prod_{k \neq i,j} dx_k - \mu_i(x_i) - \mu_j(x_j) - \mu_0. \tag{2.4.3}$$

In such a way, interactions can be computed to any q-order interaction desired and $y(\mathbf{x})$ can be rewritten as

$$y(\mathbf{x}) = \mu_0 + \sum_{i=1}^{d} \mu_i(x_i) + \sum_{i<j} \mu_{ij}(x_i, x_j) + \cdots + \mu_{1\ldots d}(x_1, \ldots, x_d).$$

Just as in the discrete case, the sums of squares of the above decomposition can be written as

$$\int y^2(\mathbf{x}) = \mu_0^2 + \sum_{i=1}^{d} \mu_i^2(x_i) + \sum_{i<j} \mu_{ij}^2(x_i, x_j) + \cdots + \mu_{1\ldots d}^2(x_1, \ldots, x_d).$$

Plotting is difficult for more than 2-dimensions, but the information may be used to determine whether any higher-order interactions exist. To obtain estimates of these integrals, $y(\mathbf{x})$ can be replaced with $\hat{y}(\mathbf{x})$. Since

$$E(\hat{\mu}_I(\mathbf{x})) = E\left(\int \hat{y}(\mathbf{x}) \prod_{k \in I} dx_k\right) = \int E(\hat{y}(\mathbf{x})) \prod_{k \in I} dx_k = \int y(\mathbf{x}) \prod_{k \in I} dx_k = E(\mu_I),$$

where I is the index set of variables over which to integrate, the estimates of the main effects and interactions are unbiased. For numerical quadrature problems the overall mean, $\hat{\mu}_0$, can be used as an estimate of $\int y(\mathbf{x})\, d\mathbf{x}$. To see the effect of the input variables the integrals can be computed for m evenly spaced points and the values plotted.

These integrals are not difficult to compute. The only place where $\mathbf{x}$ occurs in the predictor (2.2.3) is in $r(\mathbf{x})$. Since $r(\mathbf{x})$ is a product of one dimensional correlation functions,

$$\int r(\mathbf{x}) \prod_{k \in I} dx_k = \prod_{k \in I} \int r(\mathbf{x}) dx_k .$$

This property reduces the cost and increases the numerical accuracy in

computing the estimates of main effects and interactions.

For large simulation models it is cumbersome to plot all the possible main effects and interactions, a total of $d(d+1)/2$ plots. To reduce the number of plots, ANOVA - like tables are used to determine important effects. These tables are constructed as follows:

1. Variation around the overall mean, $\int (Y-\mu_0)^2 d\mathbf{x}$, can be estimated by taking a random sample of size n_r and computing

$$SS(\hat{Y}) = \frac{1}{n_r}\sum_{k=1}^{n_r}(\hat{y}(\mathbf{x}_k)-\mu_0)^2.$$

   We have found that a random sample of size $n_r = 1000$ is a good compromise between efficiency and accuracy in obtaining an estimate of variation. We are not estimating variability in the statistical sense, but are trying to estimate the amount of fluctuation of the response surface about μ_0 and in that sense it is similar to the total sums of squares in ANOVA.

2. Let m equal the number of points for which $\mu_i(\mathbf{x}_k)$ are computed. For all $i = 1, \ldots, d$, estimate the corresponding squared integral by

$$SS(\mu_i) = \frac{1}{m}\sum_{k=1}^{m}[\mu_i(\mathbf{x}_k)]^2.$$

3. Repeat Step 2. for as many q-order interactions as desired.

4. For all the sums of squares computed in Steps 2 and 3 compute the ratio of $SS(\mu_i)/SS(\hat{Y})$.

Like discrete ANOVA sums of squares, the sums of squares can be decomposed for the continuous case, but

$$SS(\hat{Y}) \neq \sum_{i=1}^{d} SS(\mu_i) + \sum_{i<j} SS(\mu_{ij}) + SS(\mu_{1,\ldots,d})$$

because $SS(\hat{Y})$ is only an estimate of $\int (Y-\mu_0)^2 d\mathbf{x}$, however the decomposition should be a reasonable estimate. The sums of squares for main effects and interactions will allow the variables to be ranked in importance for their effect on the response. The ratios $SS(\mu_I)/SS(\hat{Y})$ provide a measure of importance of the effect compared to the variation in the response surface. These ratios are not the equivalent of F-tests in ANOVA, since the decomposition here is comparing the effects to the variation of the response over the input space and not the error

variance around the response. The choice of the cut-off value is a subjective decision. Besides choosing which effects to plot, the ratios can be used to search for higher-order interactions if so desired.

2.5 Robustness

Since the model parameters are unknown and finding a good predictor is the goal, we would like the prediction error to be robust to misspecified model parameters. In general, the linear model part of the stochastic process is reduced to a constant, β_0. This in turn is estimated by the mean of the data, $\bar{y}$, the robustness properties of which are well studied. When the linear model term is β_0 the robustness of the covariance function parameters, $(\theta,\mathbf{p})$ is important. There are two small scale studies that provide insight into the robust properties of the covariance parameters.

Sacks, Schiller, and Welch[8] carried out a small robustness study where $\theta_1 = \cdots = \theta_d = \theta$ and $p_1 = \cdots = p_d = 2$ to find optimal experimental designs using integrated mean squared error (IMSE)

$$J_\theta(S,\hat{Y}) = \frac{1}{\sigma^2}\int E_\theta(\hat{Y}(\mathbf{x}) - Y(\mathbf{x}))^2\,dx.$$

as the optimization criterion. The study also contains evidence for the robustness of the correlation function parameters. Two studies are carried out, for $d=2$ and $d=7$. In both cases $Y(\mathbf{x})$ is the realization of a Gaussian stochastic process. Both studies show that the *IMSE* is reasonably robust to misspecified θ, especially if θ is underestimated.

A small empirical study in two dimensions is described in Welch, *et al.*[9] In this study a deterministic function was used as the example and the maximum likelihood estimates were computed. The MLE values of θ and $\mathbf{p}$ were perturbed separately to see how the changes affected the empirical root mean squared error (ERMSE). Changing $\mathbf{p}$ from its MLE value of 2.0 to 1.0 only increased the ERMSE from 5.5% to 8% of the range of Y and even at $\mathbf{p}=1.8$ the ERMSE increased only marginally. The changes to θ were similar to those from the previous study which showed that underestimation (even up to an order of magnitude) had limited effect on the ERMSE while overestimation of θ by an order of magnitude increased ERMSE from 5.5% to 12% of the range of Y.

2.6 References

1. Iman, R. L. and Helton, J. C., (1988) "An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models," *Risk Analysis*, 8, 71-90.

2. Yu, T. K., Kang, S. M., Sacks, J., and Welch, W. J., (1991) "Parametric Yield Optimization of CMOS Analogue Circuits by Quadratic Statistical Circuit Performance Models," *International Journal of Circuit Theory and Applications*, 19, 579-592.

3. Currin, C., Mitchell, T., Morris, M., and Ylvisaker, D., (1991) "Bayesian Prediction of Deterministic Functions, with Application to the Design and Analysis of Computer Experiments," *Journal of the American Statistical Association*, 86, 953-963.

4. Matheron, G., (1963) "Principles of Geostatistics," *Economic Geology*, 58, 1246-1266.

5. Cressie, N., (1991) *Statistics for Spatial Data*, John Wiley & Sons , New York, N.Y. .

6. Zimmerman, D. L. and Cressie, N., (1991) "Mean Squared Prediction Error in the Spatial Linear Model with Estimated Covariance Parameters," *Annals of the Institute of Statistical Mathematics*, 43, 27-43.

7. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., (1989) "Design and Analysis of Computer Experiments," *Statistical Science*, 4, 409-435.

8. Sacks, J., Schiller, S. B., and Welch, W. J., (1989) "Design for Computer Experiments," *Technometrics*, 31, 41-47.

9. 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, 15-25.

- Chapter 3 -

Parameter Estimation and Model Building

3.1 Introduction

In Section 2.2 we outline a model that treats the deterministic output of a computer code as the realization of a stochastic process, following SWMW.[1] A discussion of this statistical model can be found in Chapter 2. The model automatically adapts to nonlinear and interaction effects in the data. This reduces the problem of statistical model building to a problem of screening for important factors. The approaches discussed in this chapter are used to identify important variables (model building) and build a predictor (parameter estimation) without making any assumptions of linearity or additivity.

The model parameter estimates are typically computed by maximum likelihood methods with the assumption that the response is a realization of a Gaussian stochastic process. Given the correlation parameters $(\theta,\mathbf{p})$ of correlation function (2.3.1) the MLE of β is the generalized least squares estimate and the MLE of σ^2 is $\hat{\sigma}^2 = 1/n\,(\mathbf{y} - F\hat{\beta})^T R_S^{-1}(\mathbf{y} - F\hat{\beta})$. Substituting $\hat{\beta}$ and $\hat{\sigma}^2$ into the likelihood (2.2.4), the problem is to maximize

$$l(\theta,\mathbf{p}) = -\frac{1}{2}(n \ln \hat{\sigma}^2 + \ln \det R_S), \tag{3.1.1}$$

which is a function of only the correlation parameters, $(\theta,\mathbf{p})$, and the data. Full maximum likelihood estimation of all the correlation parameters, followed by plotting of estimated main effects and interactions, could be used to identify important effects. However, if the dimension d of $\mathbf{x}$ is large, there will be many correlation parameters, and maximum likelihood is intractable or at least numerically costly.

Initially, the maximum likelihood estimates are computed for unconstrained $(\theta,\mathbf{p})$; even for small problems this is too expensive to be feasible. For example, as mentioned in Welch, *et al.*,[2] to compute the MLE for a problem with 20 input variables and an experimental plan of 50 runs it took 2 CPU hours on a Cray X-MP. One option is to use a single parameter, p, by setting $p_i = p$

for $i = 1, \ldots, d$. This reduces the number of parameters to be estimated approximately by half.

Two strategies for using the likelihood equation to get parameter estimates are described in this chapter. One strategy, which will be referred to as FORWARD, limits the number of parameters that are optimized by initially assigning all the variables to a single (θ_0, p_0), i.e. $\theta_i = \theta_0$ and $p_i = p_0$ for $i = 1, \ldots, d$, and then during a number of stages allows significant variables to have their own (θ, p) when computing the MLE. The algorithm is essentially a forward selection technique for the correlation parameters, θ_i. This method reduces the number of parameters to be estimated, but requires several, possibly many, MLE optimizations of increasing costs. The method gives good results at a reasonable cost for sets of input variables with a low proportion of important factors. The second strategy, which will be referred to as ONETIME, continually updates the parameter estimates by computing MLE for (θ_i, p_i) given the rest of the parameters are fixed.

Before describing the two algorithms some notation needs to be developed. Let $D = \{1, \ldots, d\}$ be the set of indices for the input variables and let C be a subset of D and C' the complement of C. Let $\theta_C = \{\theta_i = \theta_0 \text{ for } i \in C\}$, i.e. those variables in C will have the same parameter estimate $\hat{\theta}_c$. Also, let $\theta_{C'}$ be the set of unconstrained parameters for those variables in C'. Both algorithms are sequential in nature; let $\hat{\theta}_{C_k}$ be the parameter estimate at the k^{th} stage of the algorithm. This notation is applied to the power parameters, **p**, as well.

The FORWARD algorithm is described in Section 3.3 and two examples are presented in Section 3.4 to show how the algorithm, and the statistical model in general, performs. Section 3.5 describes the second algorithm, ONETIME, and Section 3.6 compares the performance of the two algorithms. Section 3.7 provides some further remarks. First, in Section 3.2 a brief overview of the costs of computing the maximum likelihood estimates.

3.2 Computing Costs for the MLE

Optimization for large numbers of parameters can be expensive. Computer costs can be divided into two types, time and memory. The memory costs vary little in comparison to time costs for the different algorithms so when referring to costs, time costs will be implied. The cost of computing the maximum likelihood estimate is a function of the cost due to the number of calls to the

objective function, the likelihood equation, and the length of time required to compute the objective function. The total cost of the algorithm is approximately the product of these two costs.

The cost due to the number of function calls will not be considered here. There is an extensive literature on the subject of optimization algorithms. A recent text by Zhigjavsky[3] gives a broad survey of the field. The optimization algorithm used is the AMOEBA subroutine as given in Numerical Recipes.[4] This routine is a variation of the Nelder-Mead simplex algorithm.[5] There are some overhead costs when running the optimization algorithm, but these costs are greatly outweighed by the cost of computing the likelihood.

There are several specialized algorithms for computing MLE. Marshall and Mardia[6] and Kitanidis[7] propose methods for covariance functions which are linear in their parameters. Zimmerman[8] discusses methods for making computations easier for regularly spaced data. Dietrich and Osborne[9] develop algorithms for a restricted set of covariance functions with the assumption that the range parameters are known. Vecchia[10] shows that the likelihood can be approximated by the product of likelihoods from subsets of the data. This technique requires anisotropic parameters to be known or non-existant for the algorithm to be effective. None of these methods are applicable given our choice of design and covariance function, so we have continued to use the more general optimization approach already mentioned to compute maximum likelihood estimates.

We discovered that for the correlation function (2.3.1), the generation of the covariance matrix, R, is responsible for well over half the cost of computing an individual likelihood value which means that it is the cause of over half the cost of computing the MLE. The reason for this is twofold. First, the correlation matrix requires on $O(dn^2)$ arithmetic operations. Secondly, when computing unconstrained estimates of the parameters the correlation matrix needs to be computed at each step of the optimization. It is clear that the computation of MLE's for this model quickly becomes prohibitively expensive, especially since the sample size, n, increases as d increases. The methods described in the next section attempt to reduce computational costs by looking at near-optimal estimates of model parameters.

3.3 FORWARD Algorithm

The basic ideas of the FORWARD algorithm are as follows. Initially, in the correlation function (2.3.1) we set $\theta_1 = \cdots = \theta_d = \theta_0$ and $p_1 = \cdots = p_d = p_0$. so that numerical maximization of the likelihood is only over θ_0 and p_0. As the factors have the same scales, this reflects prior belief that all factors are on the same footing. (Alternatively, knowledge that particular factors are active could be used to shortcut the first few of the stages to be described.)

At each stage, let C denote the set of indices of factors constrained to share *common* values of θ_j and of p_j, while the remaining factors are allowed their own values of θ_j and p_j. Starting with $C = \{1, \ldots, d\}$, the algorithm iterates in the following way. For each j in C in turn, we remove the constraint $\theta_j = \theta_0$ and $p_j = p_0$ and maximize the log likelihood (3.1.1) subject to $\theta_i = \theta_0$ and $p_i = p_0$ for all i in $C - \{j\}$. The x_j that leads to the largest likelihood is removed from C. The procedure continues until none of the factors in C makes a large improvement in the likelihood relative to the previous stage. We now give a formal definition of the algorithm, then we discuss some variants to reduce computing time.

1. Maximize the log likelihood (3.1.1) subject to $\theta_1 = \cdots = \theta_d = \theta_0$ and $p_1 = \cdots = p_d = p_0$. and denote the maximum by l_0.
2. Set $C = \{1, \ldots, d\}$.
3. Repeat Steps 4, . . . , 7 until termination.
    4. For each j in C do:
        5. Maximize the log likelihood in equation (3.1.1) subject to $\theta_i = \theta_0$ and $p_i = p_0$ for all i in $C - \{j\}$, and denote the maximum by l_j.
    6. Let j^* denote the factor producing the largest increase, $l_j - l_0$, in the log likelihood at Step 5.
    7. IF: $l_{j^*} - l_0$ is sufficiently large then set $C = C - \{j^*\}$ and $l_0 = l_{j^*}$ (remove factor j^* from C)
       ELSE: stop, taking the estimates associated with l_0 from the previous stage.

The optimization algorithm used at Steps 1 and 5 is the Numerical Recipies routine AMOEBA; see Chapter 5 for details. This algorithm relies on choosing

several starting points to help avoid settling for only a local optimum. This is a characteristic of many optimization routines. We make several, usually five, tries from different starting points. Even when these tries drive to the same optimum it does not guarantee that a global optimum has been found, a common problem with maximum likelihood.

The algorithm is similar in spirit to forward selection of regression variables, but the relationship between the factor effects and the introduced parameters is more subtle. In experiments with factor sparsity, we have typically found that the few 'strong' factors are the first to demand their own values of θ_j and p_j. These strong factors usually have a large estimated θ_j, and removing them from C tends to drive down the estimated common θ_0 for factors in C. If there are relatively few runs, the estimated common θ_0 can eventually become zero, suggesting that factors in C are completely inactive. On the other hand, the estimated common θ_0 may be nonzero at termination, suggesting that factors in C have a minor effect. These minor effects need not be identical just because the factors have the same correlation parameters. We have also noticed in problems where factor sparsity is absent that factors can be removed from C because they demand a zero value of θ_j and are presumably inactive.

Thus, the algorithm tends to terminate early by identifying exceptions: factors that are either exceptionally active or exceptionally inactive. In all cases, the final maximum likelihood estimates can be used to construct a predictor of $y(\mathbf{x})$. Then, estimated effects can be plotted, as outlined in Section 2.4, to identify the important factors, interactions, etc.

Discussion about the stopping criterion has been deliberately vague. From one stage to the next, two correlation parameters are introduced. A standard asymptotic likelihood ratio test would suggest that twice the improvement in the log likelihood is distributed χ^2_2, with a critical value of about 6 for 5% significance. We have found a cutoff of 5--6 reasonable in a number of applications, although there are many reasons why this should not be regarded as a proper statistical test. The common θ_0 provides another indication of termination. If its estimate becomes zero, the factors sharing the common θ_0 are apparently inactive. If $\theta_0 \neq 0$ it is not necessarily true that all variables in C are active. Main effects plots or further testing using 2 ln L can be used to indicate which variables in C are truly active. This occurs because the difference between θ_0 and

θ_i for some i in C may be too small to be significant. Finally, when a stage produces only a modest change in the likelihood, not meeting the above criterion, it is prudent to run the algorithm for at least one further stage. We have sometimes found that several small changes in the likelihood can be followed by a larger change, presumably because the model does not fit well until several factors all receive their own correlation parameters.

The algorithm as described may still require excessive computer time. At each stage, a maximum likelihood computation is performed in Step 5 for each of the (many) factors in C. An adaptation dramatically reduces computational cost further.

Steps 4 and 5 find the factor that gives maximum increase in the likelihood when given its own values of θ_j and p_j. To get an inexpensive indication of the change in the likelihood, we replace Step 5 above by a maximization of the likelihood only over θ_j, keeping all other parameters (θ_i for $i \neq j$ and p_i for all i) fixed at the values that produced l_0 at the previous iteration:

5′. Maximize the log likelihood (3.1.1) over θ_j, keeping all other parameters fixed at the values estimated at the previous iteration, and denote the maximum by l'_j.

This one-dimensional line search, or some other adaptation, is forced by practical necessity. It is obviously much cheaper than the optimization over many parameters. In a number of test examples, including those reported in Section 3.4, it introduces the same factors as the full optimization. Even if it failed to find the factor giving the greatest change in the log likelihood, the factor could still emerge at a later stage. One could also try optimizing over both θ_j and p_j, but θ_j seems to be the more important parameter.

Letting j^* denote the factor index in C that produces the largest increase $l'_j - l_0$ in Step 5′, we now perform the stage's single maximum likelihood computation involving more than one parameter. Thus, Step 6 is replaced by:

6′a. Let j^* denote the factor producing the largest increase, $l'_j - l_0$, in the log likelihood at Step 5′.

6′b. Maximize the log likelihood (3.1.1) subject to $\theta_i = \theta_0$ and $p_i = p_0$ for all i in $C - \{j^*\}$, and denote the maximum by l_{j^*}.

Note that the stopping criterion is based on the likelihood calculated at Step 6′b and not on the line search approximation in Step 5′.

Other variants which were tried include giving several factors their own θ_j and p_j values simultaneously, if more than one factor is indicated by the line searches.

At each stage of this algorithm two optimizations take place. The first is the MLE optimization for $(\theta_C, p_C, \theta_{C'}, p_{C'})$. The number of parameters estimated at each stage is $2k+2$ for $k \leq d$, where k is the number of parameters in C', rather than the $2d+2$ parameters for the full parameterization. The second optimization is for finding the input variable which maximizes the reduction of the likelihood (step 5′). This optimization is extremely cheap. Not only is it a one dimensional optimization problem on θ_i which usually requires no more than 10-20 function calls, the calculation of R_S is reduced to $O(n^2)$.

3.4 Examples

It is important to test the algorithms on known functions of suitable complexity so that a true measure of the algorithms success can be taken. The first example therefore takes a completely known function. In the second example we embed a real code, involving six inputs whose effects are fairly well understood, in a function of 20 inputs, where the 14 further inputs are designed to be almost inactive.

3.4.1 A Known Function

For the first example, $y(\mathbf{x})$ is a known function defined on the 20-dimensional input space $[-1/2,+1/2]^{20}$. The most important part of $y(\mathbf{x})$ is

$$5x_{12}/(1+x_1) + 5(x_4-x_{20})^2 + x_5 + 40x_{19}^3 - 5x_{19},$$

and there are very small effects from most of the remaining factors:

$$0.05x_2 + 0.08x_3 - 0.03x_6 + 0.03x_7 - 0.09x_9 - 0.01x_{10} - 0.07x_{11}$$
$$+ 0.25x_{13}^2 - 0.04x_{14} + 0.06x_{15} - 0.01x_{17} - 0.03x_{18}.$$

This function is designed to be challenging, with strong nonlinear effects and two interactions.

We will describe analyses of data from Latin hypercube sampling designs[11] of 30, 40, and 50 runs. For a discussion of Latin hypercube designs refer to Chapter 4.

We first describe the analysis of the data from the 50-run design. Taking the predictor (2.2.3) with correlation function (2.3.1) and the linear model part just a constant β, we used the algorithm of Section 3.3 as modified for the cheap line searches (Step 5′). Table 3.1 shows that the initial maximum log likelihood subject to $\theta_1 = \cdots = \theta_{20} = \theta$ and $p_1 = \cdots = p_{20} = p$ is -33.5 (-2 ln L = 67.0). Line searches over θ_j for $j = 1, \ldots, 20$ in turn, lead to a best maximum log likelihood of -27.6 when θ_{12} is unconstrained. This is similar to the value of -26.0 (-2 ln L = 52.0) given in Table 3.1 for the full maximization over θ_{12}, p_{12}, and the other 19 factors' common θ_c and p (Step 6′b). At the next stage, the line searches identify θ_{19}, and so on.

Factors With Own θ_j and p_j	$\hat{\theta}$ for Factors in C	-2 Log Likelihood	Change
--	.15	67.0	--
12	.051	52.0	15.0
12, 19	.039	43.6	8.4
12, 19, 20	.035	33.0	10.6
12, 19, 20, 4	.00032	5.4	27.6
12, 19, 20, 4, 1	.000015	-20.6	26.0
12, 19, 20, 4, 1, 5	0	-42.4	21.8

Table 3.1 Known-Function Example

The first six factors to receive their own θ_j and p_j are x_{12}, x_{19}, x_{20}, x_4, x_1, and x_5. All of these stages lead to large changes in the likelihood, satisfying our benchmark criterion of about 5--6 for twice the increase in the log likelihood. With these six factors having their own θ_j's and p_j's, the estimated common θ_c is zero, and the line searches show approximately zero change in the log likelihood for the remaining factors. Thus, having correctly identified the six important factors, the algorithm terminates. Running time is about 5 minutes on a Cray X-MP.

Clearly, the cheap line searches (Step 5′) of the algorithm in Section 3.3 correctly identify the important factors. There is also a considerable saving in

computing time: running the algorithm with the fuller search in Step 5 takes a total of nearly 2 hours of Cray X-MP CPU time.

At termination, the estimated θ_j's and p_j's are:

j	1	4	5	12	19	20
$\hat{\theta}_j$	0.021	0.036	0.000085	0.011	0.0030	0.030
$\hat{p}_j$	2.00	2.00	2.00	2.00	1.70	2.00

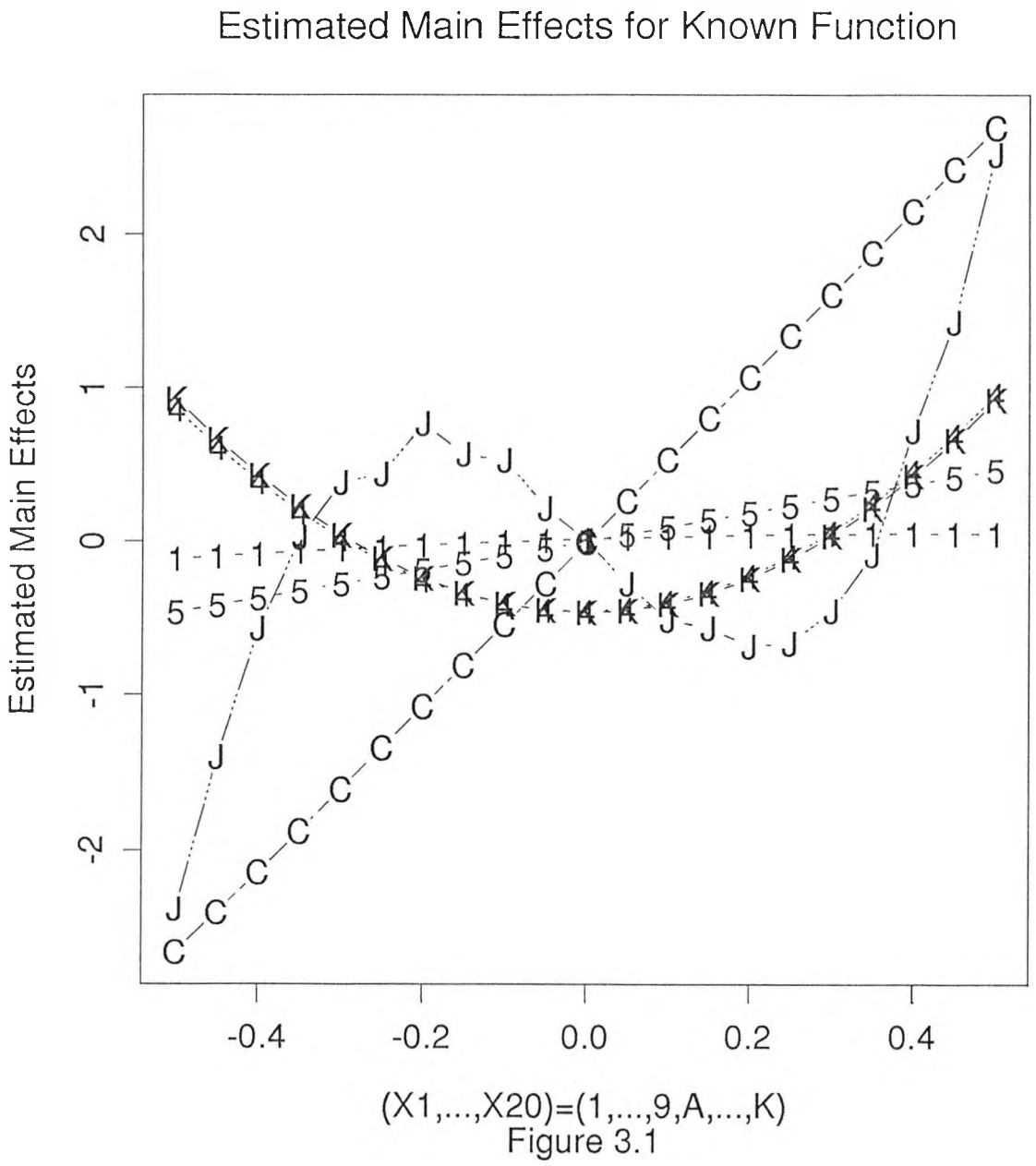
Rather than attempt to interpret these numbers it is usually more productive to plot the estimated main effects and interactions as outlined in Section 2.4.

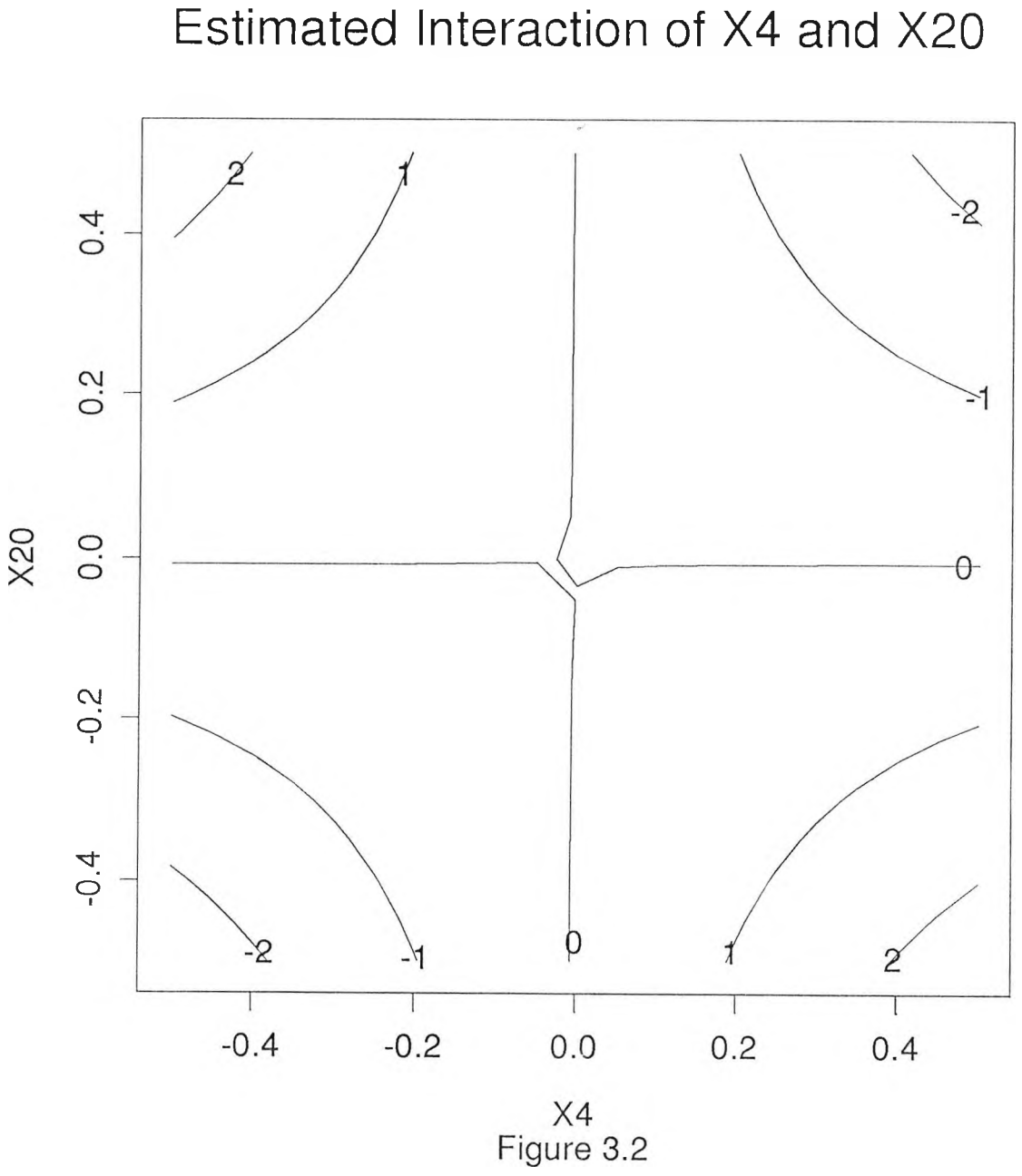
Figure 3.1 shows the estimated main effects of x_1, x_4, x_5, x_{12}, x_{19}, and x_{20}. [The remaining factors have an estimated common θ of zero, so the fitted predictor $\hat{y}(\mathbf{x})$ is constant with respect to these variables.] The quadratic effects of x_4 and x_{20} and the cubic effect of x_{19} are immediately apparent. Inspection of the estimated two-factor interaction effects for each pair of identified factors suggests large interactions between x_1 and x_{12} and between x_4 and x_{20}. For example, Figure 3.2 is the contour plot of the estimated interaction effect of x_4 and x_{20}, which agrees well with the true interaction, $-10x_4x_{20}$. The contour plot of the estimated interaction of x_1 and x_{12} has contours ranging from about -1 to 1, and is therefore also non-trivial relative to the estimated main effects in Figure 3.1. Thus, although x_1 has no main effect, it is picked up by the algorithm, and its purely interaction effect is revealed. The remaining estimated two-factor interactions are negligible: none of these plots has contours of magnitude much larger than about ± 0.1. Thus, only the two real interactions are identified.

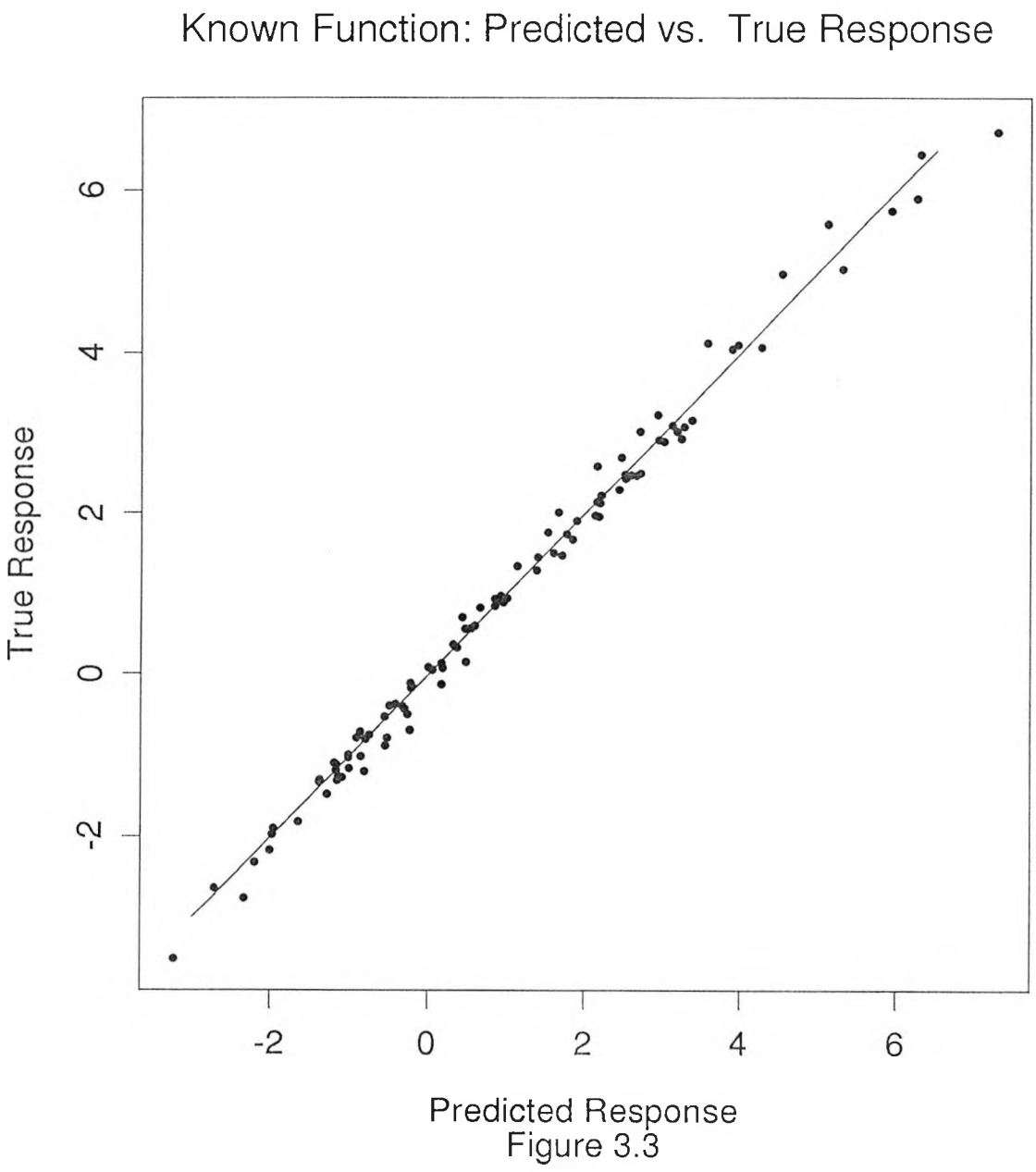
To assess the accuracy of a predictor, we now typically perform a cross validation. Let $\hat{y}_{-i}(\mathbf{x}_i)$ denote the best linear unbiased predictor (2.2.3) of $y(\mathbf{x}_i)$ based on all the data *except* the observation $y(\mathbf{x}_i)$. A cross-validation version of the empirical root mean square error (ERMSE) is then

$$\left\{\frac{1}{n}\sum [\hat{y}_{-i}(\mathbf{x}_i)-y(\mathbf{x}_i)]^2\right\}^{1/2}. \tag{3.4.1}$$

Here, the cross-validation ERMSE is 0.201, relative to a data range of about -4.4 to 8.1. To minimize computation, the MLEs of the correlation parameters are not re-computed for each prediction; they are still based on the complete data set. Nonetheless, the cross-validation ERMSE is a good measure of prediction uncertainty in this example. To show this, we generated y at 100 random points


Estimated Main Effects for Known Function
Estimated Main Effects
2
1
0
-1
-2
-0.4
-0.2
0.0
0.2
0.4
(X1,...,X20)=(1,...,9,A,...,K)
Figure 3.1


Estimated Interaction of X4 and X20
X20
0.4
0.2
0.0
-0.2
-0.4
2
1
-2
-1
0
-2
-1
0
1
2
-0.4
-0.2
0.0
0.2
0.4
X4
Figure 3.2


Known Function: Predicted vs. True Response
True Response
-2
0
2
4
6
-2
0
2
4
6
Predicted Response
Figure 3.3

in the 20-dimensional region. At these new points, the ERMSE is 0.198, very close to the cross-validation ERMSE. The plot in Figure 3.3 of $\hat{y}(\mathbf{x})$ against $y(\mathbf{x})$ at the 100 random points demonstrates the accuracy of the predictor.

To reduce computing time for the likelihood maximizations, we performed another analysis, fixing $p_1 = \cdots = p_{20} = p$ (regardless of the status of C). This roughly halves the number of correlation parameters to optimize, and reduces computing time by about 25%. The six important factors are still identified, but the accuracy of the final predictor is compromised. The ERMSE at the same 100 random points, is now .247 (versus .198 with different p_j's).

We now describe some standard screening methods applied to the data from the 50-run Latin hypercube. Fitting a first-order model in $x_1, \ldots, x_{20}$ to the ranks of y by least squares (a minor departure from the stepwise method advocated by Iman and Conover[12]), and arbitrarily taking $|t|>2$ to indicate significance, identifies only x_1 and x_{12}. The relatively unimportant x_{13} has $t=1.98$, so might also be included in practice. Repeating this with the raw y's rather than the ranks gives $|t|>2$ for the unimportant variables x_{17} and x_{18} as well. Residual analysis is not very revealing because there are several inadequacies masking each other; though, armed with foreknowledge, there is a suggestion of the cubic effect of x_{19}. Using residual analysis to cope with possible interactions would be tedious when there are many x_j's.

Without identifying the important factors, any predictor is likely to be inaccurate. Even if a quadratic model is fitted by least squares to the six important factors, the predictor remains relatively inaccurate. Fitting such a model, followed by backward elimination removing the term with the smallest $|t|$ value until $|t|>2$ for all terms, gives an ERMSE at the 100 random points of about 0.91, nearly five times as large as that for our predictor.

Alternative screening methods based on other designs might also be considered. For example, two-level, Plackett-Burman designs[13] are often used for screening, at least in physical experiments with random error. A 28-run, first-stage design (plus center point) would allow further runs to estimate interactions and quadratic effects for the important factors and still probably stay within a total of 50 runs. As with the above linear main-effects analyses, the Plackett-Burman design cannot be expected to do well, and it only finds x_{12} and x_{19}.

Applying our algorithm to the data from the n=30 or n=40 Latin hypercubes is less successful. With n=30 only x_1 and x_{12} are detected, and with n=40 only x_{12} is found. In both cases, several of the remaining important factors would be the next to enter, but they lead to small changes in the likelihood. Thus, it appears that our challenging example is too challenging without at least 50 runs. Repeating the n=40 and n=50 analyses with data from new Latin hypercubes confirms that 40 runs are inadequate but 50 runs is successful.

To summarize the first example, with 50 runs from a Latin hypercube our algorithm correctly identifies the six important factors, and the fitted model leads to a fairly accurate predictor. Fitting first-order models to the raw responses or their ranks by least squares fails to identify some of the important factors and may even find unimportant inputs to be significant. Similarly, a first-stage, Plackett-Burman design fails here. The complexity of the response relationship necessitates 50 runs for our algorithm to be successful.

3.4.2 Circuit Simulation

The second example is based on the circuit-simulation code analyzed by SWMW.[1] In their treatment, six inputs (transistor widths) are varied, and with 30 runs of the code a reasonably accurate predictor was found. Thus, the nature of the relationship between y and $x_1, \ldots, x_6$ is fairly well understood. To the output of this real code, a clock skew, a small contribution from $x_7, \ldots, x_{20}$ was added. In this way a function with a 20-dimensional input is created, where it is known which variables are important and their effects, yet the function is realistic.

A 50-run Latin-hypercube design was again tried for the 20 inputs. All inputs are normalized to [−1/2,+1/2]. To generate the data, factors $x_1, \ldots, x_6$ from the design are fed into the circuit-simulation code, and the 50 resulting clock skews are augmented with small, linear effects due to $x_7, \ldots, x_{20}$.

Table 3.2 shows that the algorithm gives factors x_5, x_3, x_6, x_2, and x_4, in that order, their own θ_j's and p_j's. Thereafter, the change in the log likelihood is rather smaller, and the algorithm terminates with these factors. Thus, all of the "real" inputs except x_1 are identified; the original SWMW[1] analysis also found x_1 to be irrelevant (to the surprise of the engineer), so the algorithm correctly identifies the five important factors.

Factors With Own θ_j and p_j	$\hat{\theta}$ for Factors in C	-2 Log Likelihood	Change
--	.014	-139.0	---
5	.0032	-147.0	8.0
5, 3	.0027	-153.8	6.8
5, 3, 6	.0067	-161.4	7.6
5, 3, 6, 2	.0018	-169.8	8.4
5, 3, 6, 2, 4	.00043	-196.6	26.8
5, 3, 6, 2, 4, 18	.00014	-201.0	4.4
5, 3, 6, 2, 4, 18, 9	.0000014	-204.6	3.6

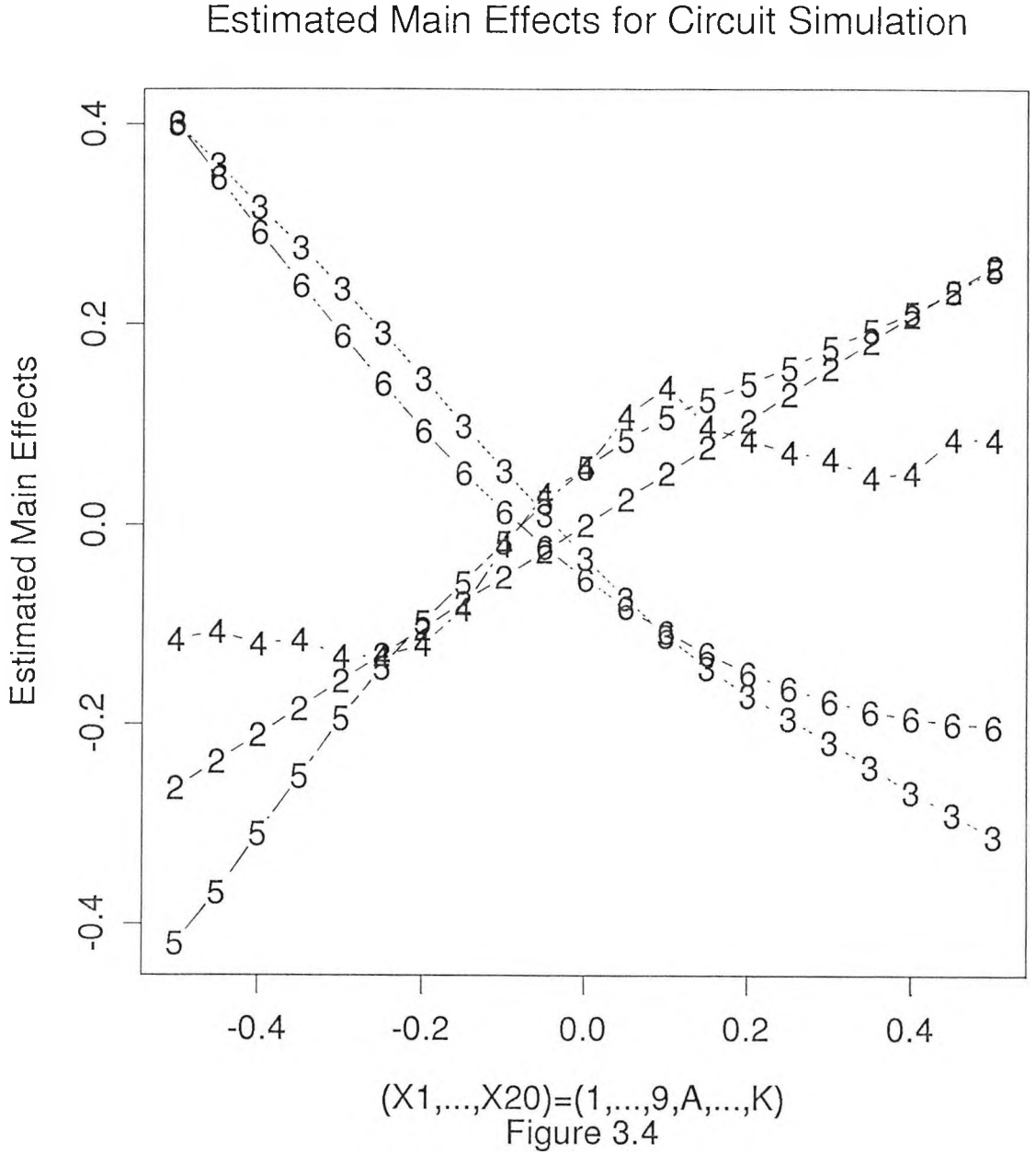
Table 3.2 Circuit-Simulation Example

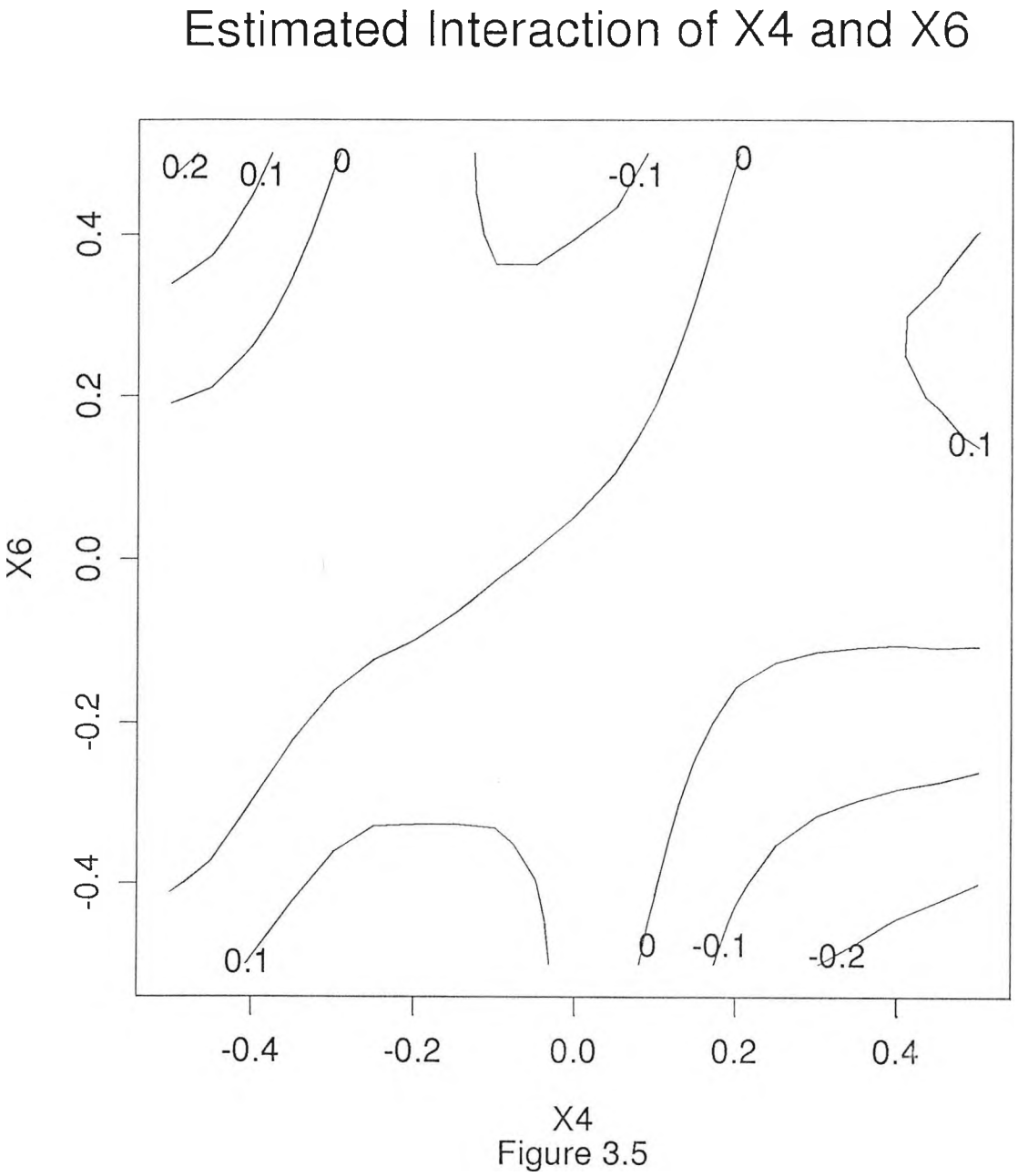
At termination, the estimated θ_j's and p_j's are:

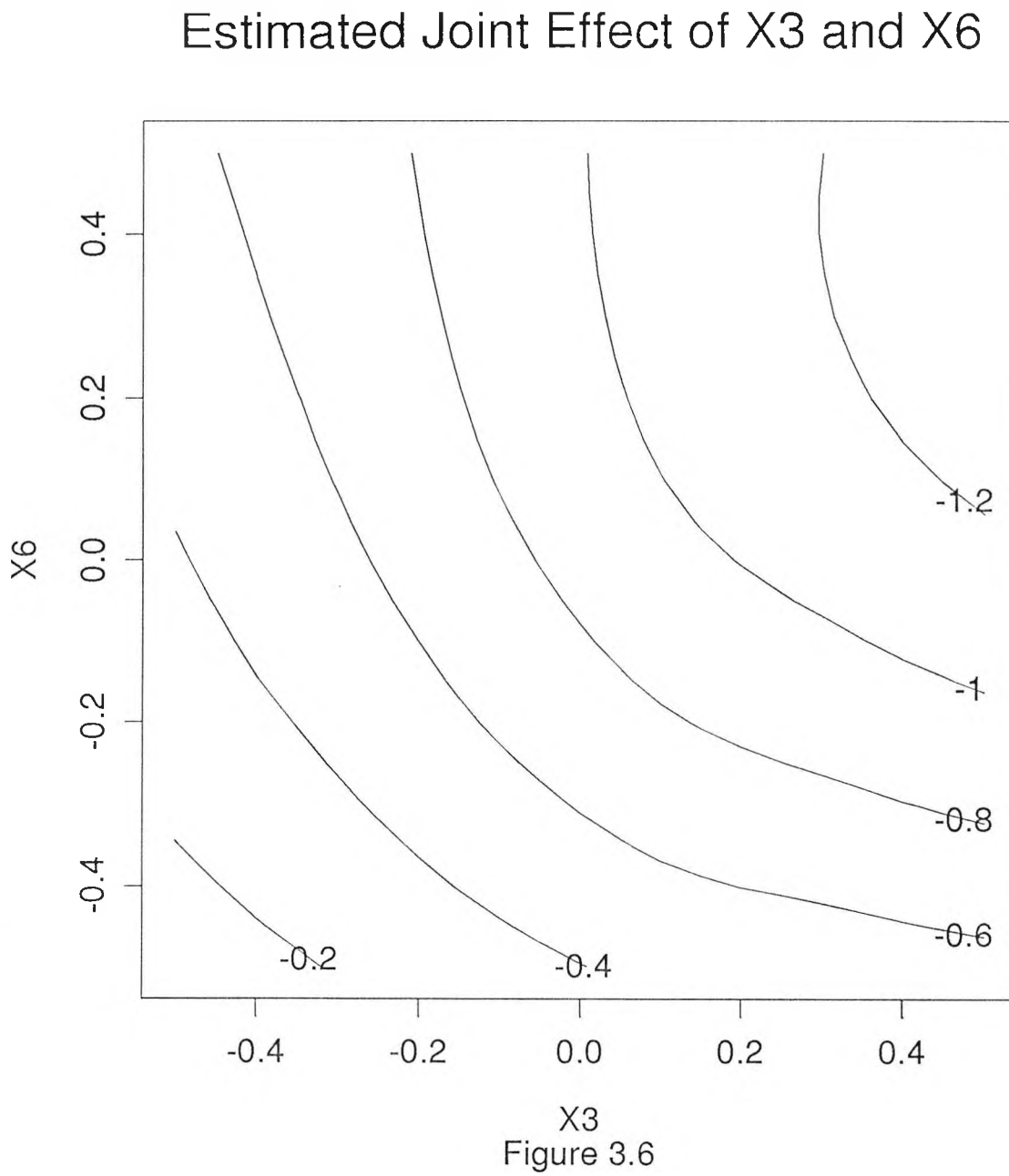
j	2	3	4	5	6
$\hat{\theta}_j$	0.035	0.058	0.15	0.065	0.97
$\hat{p}_j$	2.00	1.90	1.61	1.85	2.00

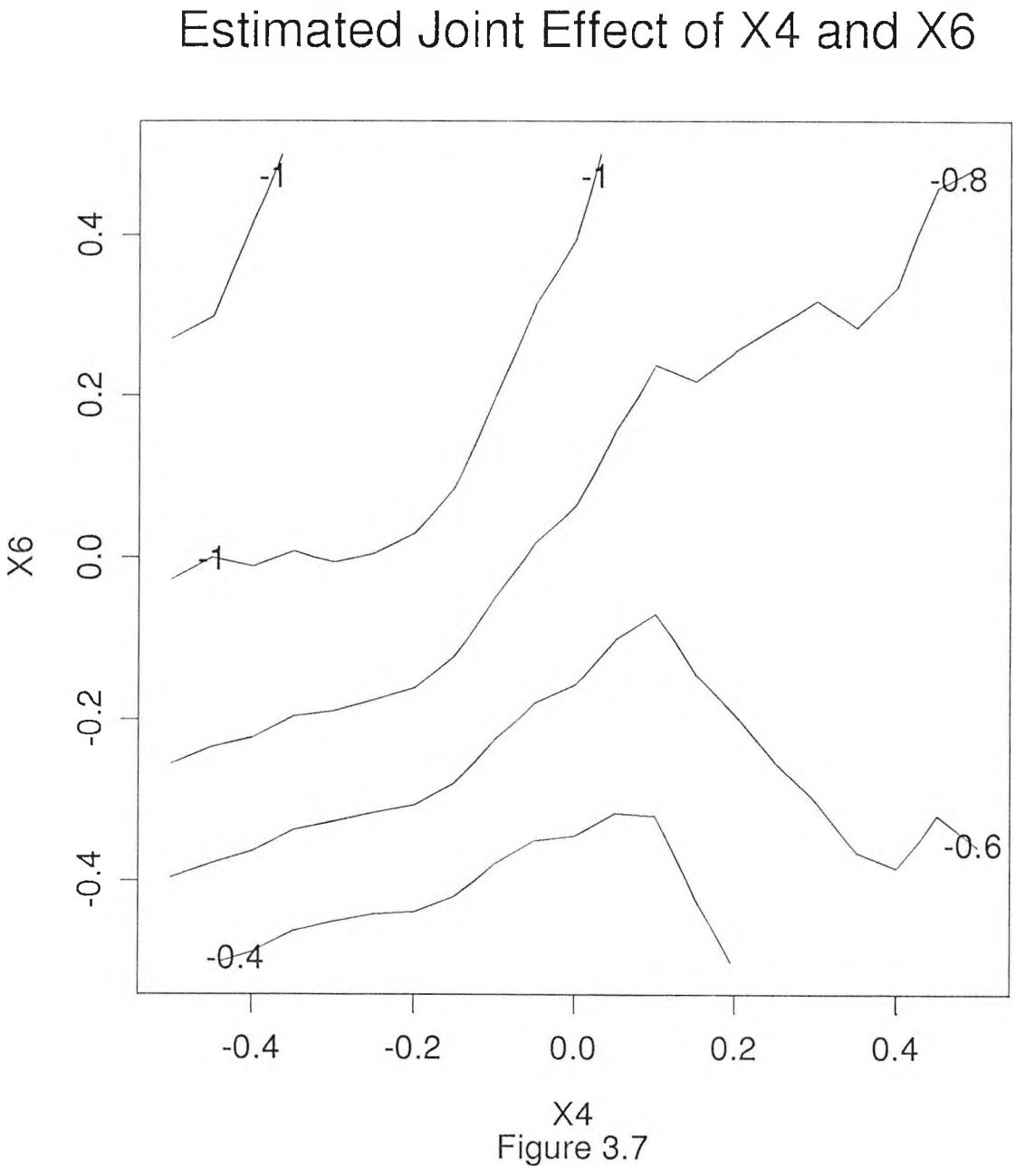
The estimated main effects are shown in Figure 3.4. Factors x_1 and $x_7, \ldots, x_{20}$, which share a common $\hat{\theta} = 0.00043$ and $\hat{p} = 1.96$, produce a blur of very small main effects in the figure.

Plots of the estimated interactions identify the interactions between x_3 and x_6 and between x_4 and x_6 as reasonably large relative to the main effects in Figure 3.4. The larger of these is the x_4—x_6 interaction, plotted in Figure 3.5, which was also found in the SWMW[1] analysis. As with standard factorial experiments, when two factors interact their joint effect on the response should be considered. Figures 3.6 and 3.7 show the estimated joint effect (overall mean plus main effects plus interaction effect) of x_3 and x_6 and of x_4 and x_6 on the clock skew. As skews close to zero are desirable, these joint effects call for small values of x_3 and x_6 with the value of x_4 less important. The main effects of factors x_2 and x_5 could also be used to bring the skew on target, and, in fact, there are many combinations of $x_2, \ldots, x_6$ that make the predicted skew close to zero.


Estimated Main Effects for Circuit Simulation
Estimated Main Effects
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
(X1,...,X20)=(1,...,9,A,...,K)
Figure 3.4


Estimated Interaction of X4 and X6
0.2
0.1
0
-0.1
0
0.1
0.1
0
-0.1
-0.2
X6
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
X4
Figure 3.5


Estimated Joint Effect of X3 and X6
X6
0.4
0.2
0.0
-0.2
-0.4
-1.2
-1
-0.8
-0.6
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
X3
Figure 3.6


Estimated Joint Effect of X4 and X6
-1
-1
-0.8
-0.6
-0.4
X6
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
X4
Figure 3.7

The cross-validation ERMSE in (3.4.1) is 0.104, relative to a data range of -1.71 to -0.10. Therefore the predictor captures the major part of the variability in the function. At 100 new, random points the ERMSE is 0.102, so the cross validation is again a good indicator of prediction accuracy.

The cheap line search in Step 5′ of the algorithm in Section 3.3 is successful in identifying the important factors. Running the algorithm with the fuller search in Step 5 finds the same factors, but increases running time from about 5 minutes to over 1 hour on a Cray X-MP.

Fitting a first-order model in $x_1, \ldots, x_{20}$ to the skews, y, identifies $x_2, \ldots, x_6$ as having $|t|>2$. Using the ranks of the skews instead is less conclusive: $t=1.64$ for x_4, whereas $t=1.79$ for one of the unimportant factors. A second-order model fitted to $x_2, \ldots, x_5$, followed by backward elimination removing the term with the smallest $|t|$ value until $|t|>2$ for all terms, gives an ERMSE of 0.131 at the 100 random points, about 30% greater than that from our predictor.

Running the algorithm on data from a 30-run Latin hypercube successfully finds $x_2, \ldots, x_5$, so a smaller experiment would be adequate for screening in this case. The resulting predictor is, of course, not so accurate; it has an ERMSE of 0.167.

3.5 ONETIME Algorithm

The FORWARD algorithm has two drawbacks. The algorithm relies heavily on the assumption that a small number of variables effect the response. If this is not the case the computation of large optimization problems is still required. Also, it assumes the number of important factors is a low percentage of the total number of input factors. If the percent of significant factors is large, say more than 50% of the factors, FORWARD will screen out the unimportant factors and leave the important factors in C.

The new algorithm, ONETIME, is a numerical optimization algorithm rather than the model building algorithm of FORWARD. ONETIME reduces the calculation of the parameter estimates to a series of two dimensional optimizations over the pair (θ_i, p_i) in (θ_D, p_D). These small optimization problems are more likely to find global optima so a single starting point, the previous values for (θ_i, p_i), is sufficient. The steps in the algorithm are as follows:

Step 1.

Let $C_0 = D$ and $k = 0$ and compute $L_k^* = \max l(\theta_{C_0}, p_{C_0})$. Now let C be the empty set, so $C' = D$ and let $(\hat{\theta}_D, \hat{p}_D) = (\hat{\theta}_{C_0}, \hat{p}_{D_0})$.

Step 2.

For variable x_i, $i = 1, \ldots, d$ compute

$$L_i = \max_{(\theta_i, p_i)} l(\theta_i, p_i \mid \hat{\theta}_{D-\{i\}}, \hat{p}_{D-\{i\}}).$$

and let $(\hat{\theta}_i^*, \hat{p}_i^*)$ be the estimates that maximize L_i. Replace estimates for (θ_i, p_i) in $(\hat{\theta}_D, \hat{p}_D)$. with $(\hat{\theta}_i^*, \hat{p}_i^*)$ and let $L_{k+1}^* = L_d$.

Step 3.

Repeat Step 2 until $L_{k+1}^* - L_k^* < \varepsilon$.

This algorithm gives unconstrained estimates of $(\theta, \mathbf{p})$.

This algorithm is very similar to the FORWARD algorithm except that instead of computing the MLE over $(\beta, \sigma^2, \theta, \mathbf{p})$ we use the estimates of θ and $\mathbf{p}$ as a basis for improving the likelihood one variable at a time. The cost for Step 2 in this algorithm is only slightly larger than the cost of Step 5′ in FORWARD. The savings comes from never performing the costly step of estimating the MLE over all parameters through standard procedures. Of course this means that we can never be sure that the estimates we get are the maximum likelihood estimates of the parameters. The number of repetitions of Step 2 before convergence is dependent on the number of input variables. From the examples it appears that 15-20 iterations is adequate for most problems. Choosing the stopping rule is not straightforward; the use of the test statistic $2 \ln L$ is not advised since this is a numerical convergence algorithm. It appears that significant reduction in the likelihood value can still be made after $L_{k+1}^* - L_k^*$ approaches zero, so it is important not to stop too soon.

One can use these estimates for similar model building or screening practices as the FORWARD algorithm by testing the hypothesis $H_0 : \theta_i = 0$ for $i = 1, \ldots, d$. As in the FORWARD algorithm, $2 \ln L$ can be used as the test statistic under the assumption that it has a χ_2^2 distribution.

3.6 Comparison of FORWARD and ONETIME Algorithms

The FORWARD algorithm uses a fairly traditional approach to computing parameter estimates. The ONETIME algorithm is a numerical optimization

routine and why it should find the global optimum is an open question. Two possible explanations have to do with unimodality of the likelihood function and the starting optimization of the algorithm. If the likelihood function is unimodal then the algorithm should work quite well. It is unlikely that the likelihood function will be unimodal under all circumstances, but there is some (conflicting) evidence[14 15 16] about the prevalence of unimodality, none of which pertains to the covariance structure used here. Starting with a common $(\theta,\mathbf{p})$ and computing the MLE may also help to explain the success of the ONETIME algorithm. The MLE, $(\hat{\theta},\hat{\mathbf{p}})$, behaves somewhat like a weighted average of the "important" factors (θ large) and the "unimportant" factors (θ small). The θ's for these two classes quickly diverge with the θ's for the important factors initially increasing and the θ's for the unimportant factors quickly being driven to near zero. Then it is just a matter of the important factors sorting themselves out to their respective values. From this, just the first few stages of the ONETIME algorithm could be used as a screening method and then a more traditional maximum likelihood approach could be used with the θ's for the unimportant variables set to zero.

Since the theoretical evidence available to show that the likelihood maximization of the ONETIME algorithm are MLE is extremely limited, empirical evidence must be used to show the efficiency of ONETIME. Any algorithm that uses likelihood calculations to obtain parameter estimates should accomplish three goals.

1) Find parameter estimates that have a likelihood value near the maximum likelihood when each x_i has its own unconstrained parameters (θ_i, p_i).

2) The predictive ability of these estimates should be nearly as good as the full MLE.

3) It must be cheaper than computing the actual MLE.

The algorithm is of no interest if it does not achieve the last two goals and it is difficult to determine its usefulness if the first goal is not accomplished.

Thirteen different response variables are used to compare the performance of the two algorithms. The 13 response variables span 4 different computer simulation models and 8 different experimental designs. All the experimental designs used are Latin hypercube samples.[11]

3.6.1 Description of Examples

Two of the simulation models are described thoroughly in Section 3.4. The results for the known function example in Section 3.4.1 are identified as TOY30, TOY40, TOY50 depending on the sample size and the results for the circuit example in Section 3.4.2 are labeled TAT30, and TAT50 respectively. The other 2 simulation models are also computer circuit simulation models. One is a VLSI Voltage-Shifter circuit and is referred to as the ATT example. The other is a proprietary circuit from INTEL.

In the ATT example we model 3 different responses: Voltage (VOLT), Gain (GAIN), and Bandwidth (BW). Each response is modeled using 14 input factors and a sample size of 62. This example is discussed in detail in Section 6.3.

For the INTEL example we ran two experiments using the same input factors. One experiment covered the full experimental space of the input variables and the second experiment looked at a subset of this space. For the experiment on the full space 3 response variables: TDH, VCTDL, and ICC, are used to compare the two MLE algorithms. The sample size for this experiment varied for the different responses since the responses at some input settings were not valid. The sample sizes were 63, 67 and 59 respectively. From the second experiment study two response variables, VSTDH and TDL, were used for comparision. The sample size for the responses in this experiment is 75 runs. The decision about which response variables to test for which experiment in the INTEL example was arbitrarily made, there were 8 response variables in the actual problem and it did not seem necessary to make comparisons for all 8 responses from all stages of the primary study. A more detailed description of this example is discussed in Section 6.4

3.6.2 Comparison Results

For each example the two algorithms, FORWARD and ONETIME, are compared on three points: computation time, 2 ln likelihood value, and predictive efficiency. We do not include a comparison with a true MLE calculation for two reasons. The first and simplest is that it would be prohibitively expensive, but this does not excuse the need to make the comparison. The second reason addresses that need. The FORWARD algorithm computes MLEs on a constrained set of parameters. As long as FORWARD correctly places the variables

in the sets C and C' and the factors in C are insignificant or have small effects then the MLE from the FORWARD algorithm should be nearly the same as those from an unconstrained optimization. In the two toy simulation models the important variables are known and FORWARD successfully captured all the important variables, so estimates should be essentially the same as if the parameters had been unconstrained. For the ATT and INTEL examples the important variables are not known, but the success in searching for optimal parameter values, which was the primary purpose for investigating these circuits, is supporting evidence for a good statistical model. It seems reasonable then, that if ONETIME compares favorably to FORWARD then it will compare favorably to the unconstrained MLE.

As mentioned in the introduction an algorithm needs to be able to accomplish three goals to be considered as a substitute for unconstrained maximum likelihood estimates. Two of these three criteria are easy to measure, the time it takes to reach a solution and the computed maximum likelihood value. The third criterion, predictive ability of the resulting model, is measured by one or two statistics. The two toy examples were inexpensive to run and the code was available for running further tests. For these examples Latin hypercube samples were generated with n=100 to calculate the RMSE of prediction. Random samples were not available for the two circuit simulator examples so for all four examples cross-validation RMSE of prediction was computed.

When optimizing the likelihood function one parameter at a time, as in ONETIME, parameter order may be important. To check the effect of parameter order the experimental design columns were randomly permuted to generate nine different data sets and parameter estimates are computed for each data set. The timing results for ONETIME in Table 3.3 is for one data set not the combined time for all nine permutations. All other results for ONETIME gives the best result of the nine data sets.

The timing results are in Table 3.3 and Table 3.4 gives the -2 ln likelihood values for the 13 data sets. The timing results in Table 3.3 clearly show that the ONETIME algorithm is much faster than FORWARD even if one considered the sum of all nine trials. Table 3.4 shows that the two algorithms give roughly the same likelihood value, except for TOY40 and VOLT where ONETIME did considerably better. The %OPT column of Table 3.4 gives the percentage of the 9 permutations that reach the best likelihood value. For 9 of the 13 responses the

results were stable over the permutation of design columns. Two responses with a low %OPT, TOY40 and TOY30, had considerable variation in the computed maximum likelihood values for the different permuations which is reflected in the high Coefficient of Variation, which is shown in the CV column of Table 3.4. For CUBIC TOY40, the variability is due to a major improvement in the maximum likelihood value for one of the permuted data sets. The improvement is more modest for VOLT and is more likely due to the factors in C not being homogeneous. The results show that parameter order will occasionally affect the likelihood computation, but using 4 or 5 permuted designs should capture the best result. This still leads to a considerable savings in time. The results also show that the variability in likelihood solutions between permuted data sets may be used to distinguish between stable and unstable optimizations.

The prediction and cross validation results are in Table 3.5. From the results of the two toy examples we can see that the ONETIME algorithm does only slightly worse predicting new input values than the FORWARD algorithm. One can also see that for the circuit simulation models that the two algorithms give similar cross-validation results. In all examples, except the cubic toy problem for $n = 30$ and $n = 40$, both algorithms give a prediction error $\leq 10\%$ of the response range. These results combined with the likelihood results are strong evidence that the ONETIME and FORWARD algorithms compute parameter estimates effectively for the examples given.

Both algorithms use -2 ln L as a test statistic, but for somewhat different purposes. FORWARD uses -2 ln L to determine which variable to add next to the model, the equivalent of an F-test in regression. The ONETIME algorithm uses the test statistic more like a multiple comparison t-test in regression. For both of these tests, assuming a χ^2_2 distribution, the critical value is 6.00. See Tables 3.6-3.8 for listing of $\Delta(-2 \ln L)$ for the ONETIME algorithm. The equivalent results for the FORWARD algorithm give the same list with only a small number of exceptions, besides the failure of the FORWARD algorithm to identify important factors for TOY30 and TOY40. When the "significant" factors from the test statistic are compared with the size of their main effects plots there is strong agreement between the two measures of factor influence. The comparison does show that factors with test statistic values between 6.0 and 20.0 tend to be insignificant factors or have small linear effects. Multiple testing has some influence on overestimation of significant factors. For example,

$\chi^2_{.996} = 11.3$ would be the Bonferroni style critical value for 15 multiple comparisons and $\alpha = .95$. This brings the critical value much more into line with what the main effects show to be important factors.

3.7 Conclusion

The curse of dimensionality apparently requires a rapid increase in the number of observations as the dimension of the input grows. Yet, with factor sparsity, the problem is not nearly so bad, provided the few important factors can be identified. In this situation, the methods used here can find the important variables and can detect curvature and interactions, without explicitly modeling such effects. The simplicity of modeling the effects when using these methods carries over to the issue of experimental design as well. Since the methods do not explicitly model interactions, etc., the reasons for using orthogonal designs are diminished and with it the difficulties in choosing alias structures for experimental designs for unknown models. Fitting a useful predictor of the response is often possible without collecting further data. Cross validation seems to provide a good indication of accuracy.

Various methods based on fitting first-order models failed to find the important effects in the first example. This does not rule out the possibility that a determined application of residual analysis, etc., would improve these regression models to the point of being useful. On the other hand, searching for interactions would, at best, be fairly tedious with a large number of input factors. Moreover, the methods discussed here are data adaptive and can find interactions and other complexities in a fairly automatic way.

Computing time can be substantial for the FORWARD algorithm, but is reasonable for the ONETIME algorithm (30-45 minutes for problems given here on Sun SPARC2). Often, computer codes are themselves computationally expensive, so expensive data justify a careful analysis. Even if data are cheap to generate, it may be difficult to solve a high-dimensional problem simply by increasing the amount of data; a more intricate analysis may be necessary anyway.

EXAMPLE	FORWARD	ONETIME
CUBIC TOY50	283.6	16.9
CUBIC TOY40	181.7	9.8
CUBIC TOY30	108.5	6.1
TAT TOY50	373.9	13.9
TAT TOY30	70.5	5.6
ATT VOLT	561.4	22.1
ATT GAIN	504.8	20.7
ATT BW	505.1	13.8
INTEL TDH	349.6	15.7
INTEL VCTDL	279.4	16.4
INTEL ICC	201.9	11.6
INTEL VSTDH	350.2	16.9
INTEL TDL	604.3	24.6

Table 3.3 Computation Time (CPU seconds on Cray XMP)

EXAMPLE	FORWARD	ONETIME	%OPT	WORST	ICVI
CUBIC TOY50	-42.3	-42.9	100	-42.9	0.00
CUBIC TOY40	10.9	-11.8	11	19.2	1.06
CUBIC TOY30	7.6	7.6	33	29.4	0.54
TAT TOY50	-205.7	-209.6	78	-195.3	0.02
TAT TOY30	-115.5	-116.8	55	-114.9	0.01
ATT VOLT	-166.2	-179.6	100	-179.6	0.01
ATT GAIN	-61.9	-61.9	89	-54.0	0.04
ATT BW	-236.9	-237.2	33	-210.9	0.04
INTEL TDH	140.1	139.9	100	139.9	0.01
INTEL VCTDL	-247.7	-248.9	100	-248.9	0.00
INTEL ICC	342.8	342.6	100	342.6	0.00
INTEL VSTDH	-438.3	-440.5	100	-440.5	0.00
INTEL TDL	-344.7	-347.4	78	-335.4	0.01

Table 3.4 Maximum Likelihood Results (-2*ln Likelihood)

EXAMPLE	RANGE of Y	FORWARD	ONETIME
CUBIC TOY50	10.4	0.20 (0.20)	0.26 (0.20)
CUBIC TOY40	7.8	0.89 (1.6)	0.27 (0.31)
CUBIC TOY30	12.5	0.87 (1.10)	0.89 (0.81)
TAT TOY50	1.62	0.09 (0.10)	0.13 (0.14)
TAT TOY30	1.52	0.06 (0.16)	0.09 (0.15)
ATT VOLT	3.85	0.124	0.102
ATT GAIN	5.32	0.40	0.41
ATT BW	2.14	0.24	0.27
INTEL TDH	27.38	2.41	2.33
INTEL VCTDL	1.06	0.11	0.12
INTEL ICC	133.9	11.6	11.9
INTEL VSTDH	0.343	0.041	0.041
INTEL TDL	5.62	0.047	0.048

Table 3.5 Cross-Validation (Prediction) RMSE

INPUT	TDH	VCTDL	ICC	VSTDH	TDL
X_1	0.0	5.0	45.6	108.9	117.2
X_2	86.1	39.0	0.7	0.5	17.3
X_3	12.2	48.7	55.1	20.9	478.4
X_4	23.1	7.6	18.1	61.9	99.8
X_5	26.1	69.4	28.3	7.5	15.0
X_6	94.8	9.3	18.5	0.0	0.0
X_7	5.2	4.2	0.0	4.8	73.7
X_8	0.0	5.2	14.2	101.7	340.5
X_9	36.7	8.1	6.9	9.6	0.0
X_{10}	0.0	40.1	0.7	5.9	2.3
X_{11}	64.2	0.0	21.2	0.0	474.2

Table 3.6 Change in -2 ln L for Intel Example $\chi^2_{95,2} = 6.0$

INPUT	TOY30	TOY40	TOY50	TAT30	TAT50
X_1	17.1	54.9	57.9	-0.5	1.4
X_2	0.0	0.0	0.5	-0.5	1.8
X_3	0.0	0.0	1.0	-0.5	2.3
X_4	34.6	67.2	139.5	33.3	84.8
X_5	9.2	35.5	63.5	62.8	142.0
X_6	0.0	0.0	0.0	-0.5	1.4
X_7	0.0	0.0	0.0	0.7	10.9
X_8	0.0	0.0	0.4	8.7	1.4
X_9	0.0	0.0	0.0	-0.5	7.5
X_{10}	2.5	0.0	0.0	-0.5	1.5
X_{11}	0.0	0.0	0.0	-0.5	4.6
X_{12}	81.6	114.9	196.1	59.2	84.4
X_{13}	0.0	0.0	0.0	-0.5	1.4
X_{14}	0.0	0.0	0.1	0.4	1.4
X_{15}	0.0	0.0	0.0	-0.5	1.4
X_{16}	0.0	0.0	0.0	-0.5	1.4
X_{17}	0.0	0.0	0.0	15.4	4.4
X_{18}	0.0	0.0	0.0	1.8	16.0
X_{19}	34.5	80.5	191.6	52.1	125.2
X_{20}	27.4	64.7	138.5	59.2	112.0

Table 3.7 Change in -2 ln L for Cubic Toy and Tat Toy Examples $\chi^2_{.95,2} = 6.0$

INPUT	VOLT	GAIN	BW
X_1	140.6	105.2	0.0
X_2	3.6	0.7	191.2
X_3	-0.4	37.6	25.9
X_4	43.9	9.2	61.0
X_5	20.9	0.0	71.6
X_6	2.6	0.0	0.0
X_7	106.7	0.0	97.2
X_8	237.2	122.2	4.2
X_9	237.9	84.3	32.5
X_{10}	86.6	0.0	0.0
X_{11}	90.1	2.1	0.0
X_{12}	-0.4	6.4	0.0
X_{13}	31.3	0.0	0.0
X_{14}	-0.4	15.9	0.0

Table 3.8 Change in -2 ln L for ATT Example $\chi^2_{.95,2} = 6.0$

3.8 References

1. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., (1989) "Design and Analysis of Computer Experiments," *Statistical Science*, 4, 409-435.
2. 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, 15-25.
3. Zhigljavsky, A. A., (1991) *Theory of Global Random Search*. Kluwer Academic Publishers.
4. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., (1986) *Numerical Recipes*, Cambridge University Press, Cambridge.
5. Nelder, J. A. and Mead, R., (1965) "A Simplex Method for Function Minimization," *Computer Journal*, 7, 308-313.
6. Marshall, R. J. and Mardia, K. V., (1985) "Minimum Norm Quadratic Estimation of Components of Spatial Covariance," *Mathematical Geology*, 17, 517-525.
7. Kitanidis, P. K., (1985) "Minimum-Variance Unbiased Quadratic Estimation of Covariance of Regionalized Variables," *Mathematical Geology*, 17, 195-208.
8. Zimmerman, D. L., (1989) "Computationally Exploitable Structure of Covariance Matrices and Generalized Covariance Matrices in Spatial Models," *J. Statist. Comput. Simul.*, 32, 1-15.
9. Dietrich, C. R. and Osborne, M. R., (1991) "Estimation of Covariance Parameters in Kriging via Restricted Maximum Likelihood," *Mathematical Geology*, 23, 119-135.
10. Vecchia, A. V., (1988) "Estimation and Model Identification for Continuous Spatial Process," *Journal of the Royal Statistical Society - Series B*, 50, 297-312.
11. McKay, M. D., Conover, W. J., and Beckman, R. J., (1979) "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, 21, 239-245.
12. Iman, R. L. and Conover, W. J., (1980) "Small Sample Sensitivity Analysis Techniques for Computer Models, With Application to Risk Assessment (with discussion)," *Communications in Statistics - Theory and*

Methods, A9, 1749-1874.

13. Plackett, R. L. and Burman, J. P., (1946) ''The Design of Optimum Multifactorial Experiments,'' *Biometrika*, 33, 305-325.
14. Mardia, K. V. and Watkins, A. J., (1989) ''On Multimodality of the Likelihood in the Spatial Linear Model,'' *Biometrika*, 76, 289-295.
15. Dietrich, C. R., (1991) ''Modality of the Restricted Maximum Likelihood for Spatial Gaussian Random Fields,'' *Biometrika*, 78, 833-840.
16. Warnes, J. J. and Ripley, B. D., (1987) ''Problems with Likelihood Estimation of Covariance Functions of Spatial Gaussian Processes,'' *Biometrika*, 74, 640-642.

- Chapter 4 -

Latin Hypercube Sampling

4.1 Introduction

Most experimental designs have been developed for physical experiments, but there are several important differences between physical and computer experiments that make these experimental design less than ideal. Computer experiments tend to have large number of input variables with large regions of interest and nonlinear response variables. These two traits create problems in choosing a strategy for experimental design. Large numbers of input variables make spreading points throughout the input space a difficult task without also making the sample sizes too large. If the function is nonlinear, then it is necessary to take more values across the range of the inputs so the function can be mapped accurately.

The first part of this section is devoted to an overview of some standard experimental designs and their use with computer experiments. The second part gives an introduction to Latin hypercube sampling (LHS). In section 4.2 it is shown that LHS asymptotically fills the entire experimental space. Section 4.3 gives a brief introduction to discrepency functions and describes how the variability of random designs, such as LHS, can be used as a measure or discrepency. This section also compares results for variance and other discrepency functions for LHS and simple random sampling (SRS).

4.1.1 Experimental Design and Computer Experiments

Most research in experimental design has been based on the assumption that the response can be effectively modeled by a low order polynomial and that the models are estimated using least squares. Steinberg and Hunter[1] give the most recent overview of experimental design. Concentrating for the moment just on the properties of experimental designs there are several difficulties in using standard experimental design methods that have been developed for physical experiments.

Factorial designs, and more generally, orthogonal arrays[2 3] are the classic experimental designs for low order polynomials but do not adequately address

the problems of modeling nonlinear functions and large numbers of input factors. For factorial designs, sample size increases rapidly as the number of input factors increases. The design could be highly fractionated, but this leads to estimation and confounding problems when using linear models. The number of levels at which each variable is tested also needs to remain small to keep the sample size small. This can make estimation of nonlinear functions difficult. Finding fractional designs is difficult for larger problems and must either be looked up in published tables, which only addresses a finite number of possibilities, or computer algorithms could be used to determine the alias structure.[4 5] These algorithms do not necessarily find the best designs and can be computationally expensive.

Many orthogonal arrays have been developed that are smaller than regular fractional factorials. A notable example is Plackett-Burman type designs,[6] which are designed for estimating main effects. Wang and Wu[7 8] is some of the more recent work to make orthogonal arrays flexible in the selection of factor levels and keeping the number of runs small. These orthogonal arrays tend to be difficult to compute and still have the same problem with the tradeoff between sample size and number of factor levels. Results are scattered throughout the literature so determining what types of designs are available and best for any given situation is not a simple task.

Orthogonality is a useful property for experimental designs, but it is not an essential one, especially if the purpose of the experiment is to develop a prediction model and not estimate parameters. For computer experimentation orthogonal arrays have a major drawback. One of the properties of an orthogonal array is that if the design is projected onto a smaller space, as is the case if input variables are nonsignificant, the corresponding design is still orthogonal, but with more replicates at each design point. For computer experiments this replication is a waste of sampling points since there is no measurement error.

For other classes of experimental design, such as α-optimality and maximin, the choice of sample size is more flexible than for orthogonal arrays. However, these designs have difficulties of their own. Optimal designs usually require that model parameters be known to develop the experimental design. Of course, it is usually just these parameters that the investigator is trying to estimate. These designs also tend to be very expensive to compute when there is a moderate number of input factors.

Initially, experimental designs for computer experiments were developed for numerical integration problems.[9] [10] These attempts did not have much success beyond d=1 and the numerical analysis literature [11] also have had difficulties devising strategies for d>1. Monte Carlo simulation studies, which typically used simple random sampling, quickly became the sampling method of choice for computer experiments because they are quick and easy to implement for high dimension problems and many of the initial studies in computer experiments investigated the distribution of the response given "random" inputs. However, SRS can be improved upon as a design strategy.

4.1.2 Latin Hypercube Sampling - A Review

Latin hypercube sampling, introduced by McKay, Conover and Beckman (MC&B),[12] is an extension of lattice sampling described by Patterson.[13] Initially, LHS was used in sensitivity analysis as an improvement to SRS for estimating cumulative distribution functions.[14] [15] The length of the citation list for MC&B indicates that LHS is in common use today.

Constructing a Latin hypercube sample is a straightforward process. Assume the experimental space has been scaled to $[0,1]^d$, where d is the number of input variables. Let $\mathbf{z} = [0,1, \ldots, n-1]$, where n is the number of runs in the experimental plan. Then

$$s_{.j} = \frac{\pi_j(\mathbf{z})+1/2}{n}, j = 1, \ldots, d$$

is the j^{th} column of the experimental design S, where $\pi_1, \ldots, \pi_d$ are independent uniform random permutations of $\mathbf{z}$. This algorithm places the design points in the center of the randomly selected sections of the grid. MC&B suggest randomizing the location within these selected locations, i.e. let $s_{ij}^* = s_{ij} + \gamma_{ij}$, where γ_{ij} is a uniform random variable on $[-1/2n, 1/2n]$. This helps to simplify some theoretical calculations, but does not improve the structure of the design. This becomes apparent when n is a moderate size, causing the range of γ_{ij} to be trivially small. For the designs in this thesis the design values will not be randomized.

Several variations on MC&B initial LHS algorithm have been developed. Iman and Conover [16] discuss methods for inducing correlations between the input variables. Handcock[17] develops an algorithm called cascading Latin hypercube sampling. This type of Latin hypercube sampling generates clusters of

points at several levels, the number of levels chosen by the designer. At each level a standard LHS is carried out to locate the clusters, then at the lowest level one further LHS is created to locate the design points for the cascading Latin hypercube sampling. This design was developed to improve estimates of scale and smoothing parameters in the Matérn class of covariance functions.

MC&B show that for estimators of the form

$$T(Y_1, \ldots, Y_n) = \frac{1}{n}\sum_{i=1}^{n} g(Y_i),$$

where $Y_i = h(\mathbf{x}_i)$ and $h(\mathbf{x}_i)$ is a square integrable function, the variance is smaller using LHS than it is using random sampling or stratified sampling if certain conditions of monotonicity are met. Stein[18] and Owen[19 20 21] expand on the work by MC&B on variance estimation and asymptotic behavior of the estimates. Stein extends the work on comparing variances by showing that asymptotically $Cov(h(X_1), h(X_2)) \leq 0$ so that $Var(h(\mathbf{X}_{LHS})) \leq Var(h(\mathbf{X}_{SRS}))$ without the monotonicity restrictions given by MC&B. Stein also shows that the more additive the function $Y = h(\mathbf{x})$, the smaller the variance is when using LHS. Stein derives a central limit theorem for $E(h(X))$ and outlines another algorithm as an alternative to the one given in Iman and Conover.[16] Owen extends Stein's work on the central limit theorem and shows that the variance for integrals of $h(\mathbf{X})$ is less using LHS than SRS.

None of these papers address the physical, or space filling properties, of LHS. Two concepts can be used to investigate how well a design fills up the input space.

1. How completely does the design cover the input space?
2. How uniformly are the design points spread throughout the input space?

The answer to the first question tries to get a measure on how well the code has been exercised, i.e. how well the full range of all input variables is covered. The latter question is addressed by using discrepency functions to compare designs to a uniform distribution. For example, a 2^k full factorial design does not cover the input space very completely only having observations in the corners while SRS covers the full input space more completely. Conversely, the 2^k design spreads points more uniformly through the input space than SRS.

4.2 Asymptotic Space Filling Property

The purpose of this section is to show that for any neighborhood around an arbitrarily selected point $\mathbf{x}$, the probability of at least one design point being in the neighborhood tends to one as $n \to \infty$. This statement implies that for some n there will be no point in the input space that will not be within some prescribed distance of a design point, with probability 1. After the proof, an application of this property is given.

Proof:

Let a neighborhood of $\mathbf{x} = (x_1, \ldots, x_d)$ be defined as $N_\delta(\mathbf{x}) = [\mathbf{x} \pm \delta]$. Let $\mathbf{s}_i^* = (s_{i1}^*, \ldots, s_{id}^*) = \mathbf{s}_i$, if $s_{i1} \in N_\delta(x_1)$. If $n > 1/\delta$, then there exists a design point $\mathbf{s}_i^*$ in S. Since $n > 1/\delta$ and the values of the design points are equally spaced at intervals of $1/n$, $P(s_{ik}^* \in N_\delta(x_k)) = p_\delta$, where $p_\delta = \max_{j=1,\ldots,n} \{\frac{j}{n} : \frac{j}{n} < \delta\}$. The design points are randomized independently over the input variables so these probabilities are independent for $k = 2, \ldots, d$. Then

$$P(s_{ik} \notin N_\delta(\mathbf{x}), k=2,\ldots,d) = 1 - \prod_{k=2}^{d} P[s_{ik}^* \in N_\delta(x_k)] = 1 - p_\delta^{d-1}.$$

It is sufficient that there is at most one point in $N_\delta(\mathbf{x})$ and the probability of this is

$$1 - \prod_{s_{i1} \in N_\delta(x_1)} (1 - p_\delta^{d-1}) = 1 - (1 - p_\delta^{d-1})^{np_\delta} \to 1, \; n \to \infty \tag{4.2.1}$$

since p_δ is a constant.

For SRS the first dimension has the same probability distribution as any other dimension so

$$P(\mathbf{s} \in N_\delta(\mathbf{x})) = 1 - (1 - \delta^d)^n,$$

so SRS has this property as well.

4.2.1 Application

Computer simulation models typically do not have any measurement error, which implies that theoretically the $MSE(Y(\mathbf{x}))$ can be zero. This will happen if $h(\mathbf{x})$ is a simple linear model and regression is used to estimate the model. However, if the model is incorrect then bias will prevent $MSE(Y(\mathbf{x})) \to 0$ for regression no matter how large the sample size.

Assume that $y(\mathbf{x})$ is a realization of a stochastic process which has the form

$$Y(\mathbf{x}) = \beta + Z(\mathbf{x}).$$

Where $Z(\mathbf{x})$ is a stochastic process with mean zero and correlation

$$Corr(Z(\mathbf{x}), Z(\mathbf{w})) = R(\mathbf{x}, \mathbf{w})$$

between the responses at two inputs $\mathbf{x}$ and $\mathbf{w}$ and the variance

$$Var(Z(\mathbf{x})) = \sigma^2.$$

Let R_S be the correlation matrix for the experimental plan, S and $r(\mathbf{x}) = R(\mathbf{x}, \mathbf{s}_i)$ be the vector of correlations between $\mathbf{x}$ and each point in the experimental plan. If the best linear unbiased predictor (2.2.3) is used and it is assumed β is known, then $MSE(Y(\mathbf{x})) = \sigma^2(1 - r'(\mathbf{x})R_S^{-1}r(\mathbf{x}))$. Details of this model can be found in Chapter 2. The result given in this section can be used to show that $MSE(Y(\mathbf{x}))\to 0$ for any $\mathbf{x}$ as the design points get close to $\mathbf{x}$ $(n\to\infty)$ for an interpolating predictor.

Proof:

Let $\mathbf{x} = \mathbf{s}_i^* + \delta$, where

$$\mathbf{s}_i^* = \min_{i=1,\ldots,n} \|\mathbf{x} - \mathbf{s}_i\|, \text{ then } \mathbf{s}_i^* \in N_\delta(\mathbf{x}).$$

Then $MSE(Y(\mathbf{x}))$ can be rewritten as

$$\sigma^2(1 - r'(\mathbf{s}_i^* + \delta)R_S^{-1}r(\mathbf{s}_i^* + \delta)).$$

Let $r_\delta(\mathbf{s}_i) = r(\mathbf{s}_i + \delta)$ then by the Cauchy-Schwarz theorem:

$$|(r'_\delta(\mathbf{s}_i^*)R_S^{-1}r_\delta(\mathbf{s}_i^*))(r'(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*))|^{\frac{1}{2}} \geq |r'_\delta(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*)|.$$

Since $MSE(Y(\mathbf{x}))\geq 0$ and $MSE(Y(\mathbf{s}_i^*))=0$, this implies that $r'(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*) = 1$ and

$$1 \geq |r'_\delta(\mathbf{s}_i^*)R_S^{-1}r_\delta(\mathbf{s}_i^*)|^{\frac{1}{2}} \geq |r'_\delta(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*)|.$$

Now because $r(\mathbf{x})$ is a continuous function, as $\delta\to 0$

$$r'_\delta(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*)\to r'(\mathbf{s}_i^*)R_S^{-1}r(\mathbf{s}_i^*) = 1$$

and $MSE(Y(\mathbf{x}))\to 0$. Combined with the argument in Section 4.2 it is clear that $MSE(Y(\mathbf{x}))$ can be made arbitrarily small (in probability) by taking n sufficiently large: i.e. under LHS there exists n_0 such that for $n>n_0$ and η,ε

arbitrarily small, $P(MSE(Y(\mathbf{x})<\eta)>1-\varepsilon$. This is true even if $R(\mathbf{x},\mathbf{w})$ is misspecified since the predictor is an interpolator and $r(\mathbf{s})=0$ even if $R(\mathbf{x},\mathbf{w})$ is incorrect.

4.3 Discrepency Functions

Discrepency functions are used to measure how well points are uniformly spread throughout the design space. Let $\mu(A)$ be the volume of the region A. Then $\mathbf{x}=\{x_1,x_2,\ldots,x_N\}$ is a *uniform sequence* if for any A

$$\frac{\#\{x_i\in A\}}{N}\rightarrow\mu(A),\ N\rightarrow\infty.$$

Discrepency functions measure the deviation of finite sequences from $\mu(A)$. Two common discrepency functions are ordinary discrepency, or Kolmogorov deviation, and L_2 discrepency. Ordinary discrepency is defined as

$$D_N=\sup_A\left|\frac{\#\{x_i\in A\}}{N}-\mu(A)\right|$$

for $A=\{[0,b_1]\times\cdots\times[0,b_d]\}$. L_2 discrepency is defined as

$$D_N^{L_2}=\int_0^y\left[\frac{\#\{x_i\in A\}}{N}-\mu(A)\right]^2dy,$$

where $A=\{[0,y]\text{x}\cdots\text{x}[0,y]\}$. For random designs, such as SRS and LHS, L_2 discrepency can be written as

$$D_N^{L_2}=\int_0^y Var(X)dy,$$

where X is a random variable with a binomial distribution.

4.3.1 Variance as Discrepency Function

Most experimental plans, such as orthogonal arrays, D-optimal and Maxi-min designs can be considered deterministic, i.e. the design points are determined numerically and there is no random component involved. When a LHS or a SRS is generated the resulting experimental plan is a random event or random design. Deterministic designs typically are difficult to derive for large numbers of input variables or may be inflexible in the number of experimental runs in the design, such as for orthogonal arrays. Random designs are more readily computed than deterministic designs since they rely on the process of randomization to spread out the design points while deterministic designs rely on various, often complex,

mathematical structures to derive the design. This is one of the reasons why SRS has been so popular in large experimental settings.

An experimental plan has good space filling properties if the experimental region is covered fully and uniformly. The results in Section 4.2 show that the experimental region is completely covered asymptotically for LHS. One way to determine how uniformly a random design fills the experimental region is to measure the variability in the number of design points that fall in an arbitrary region of the experimental region. If the variability is small then the number of points in a randomly placed region will be close to the mean for that sized region. Since the region is randomly placed the mean number of design points in different regions must be approximately the same. This implies that the design points are dispersed over the experimental region in a more uniform manner than a experimental plan with higher variance.

Stein[18] has shown this to be true asymptotically by his proof that the variance of any square integrable function from a Latin hypercube sample is less than from a simple random sample. Let $h(\mathbf{x})$ be an indicator function where

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is in } V^d \\ 0 & \text{otherwise} \end{cases}, \tag{4.3.1}$$

where V^d is a region in $[0,1]^d$. The following result shows that this can be extended to all n for the function $h(\mathbf{x})$ given in (4.3.1).

One of benefits of using variance as a measure of discrepency for random designs is that no simulation studies are necessary for comparing different random designs and average squared discrepency, which for random designs is equivalent to $\int Var_p(X)dp$ can be used for comparison to deterministic designs. The mean and variance are now computed for SRS and LHS.

In general the problem can be set out as follows. Let the full experimental region be scaled to be on the unit cube $[0,1]^d$. A rectangular subregion, $V^d = V_1 \times, \ldots, \times V_d$, of the experimental space can be described as follows. Select $0 \le p_i \le 1$ and $0 \le v_i \le 1 - p_i$ for $i = 1,...,d$. Then $[v_1, v_1 + p_1] \times, \ldots, \times [v_d, v_d + p_d] = V^d$ is a subregion of the full experimental space, where p_i is the length of the i^{th} side. Let $p_1 = \cdots = p_d = p$. Let S be a design for d input variables with n runs and s_{ij} be the i^{th} observation for the j^{th} input variable.

Let $X = |V^d \cap S|$, be the number of design points in V^d. Now let

$$Q_{ij} = \begin{cases} 1 & \text{if } s_{ij} \text{ is in } V_j \\ 0 & \text{otherwise} \end{cases},$$

then

$$X = \sum_{i=1}^{n} \prod_{j=1}^{d} Q_{ij}.$$

Let X_{LHS} and X_{SRS} be the number of points in V^d for LHS and SRS respectively.

Note that for fixed i, Q_{ij} is independent of Q_{ik} for all j and k for both LHS and SRS since randomization on each input variable is taken separately in both sampling methods. However, Q_{ik} is not independent of Q_{jk}, $i \neq j$, for LHS but is for SRS. Since the probability of a point from a SRS being in V^d is p^d, the relative volume of V^d to $[0,1]^d$, $X_{SRS} \sim Bin(n, p^d)$ and $E(X_{SRS}) = np^d$ and $Var(X_{SRS}) = np^d(1-p^d)$.

Let $m = [np] + 1$, where $[np]$ is the integer part of np. Fix j and let $Q_j = \sum_{i=1}^{n} Q_{ij}$. Then Q_j is the number of design points for the j^{th} variable that are in V_j and $Q_j = m$ or $Q_j = m-1$ depending on the position of V^d in $[0,1]^d$. Then

$$P(Q_{ij} = 1 \mid Q_j = m) = m/n$$

and

$$P(Q_{ij} = 1 \mid Q_j = m-1) = (m-1)/n.$$

Define $P(Q_j = m) = p^*$ and assume that $p^* = np - (m-1)$. Then

$$E(X_{LHS}) = \sum_{i=1}^{n} \prod_{j=1}^{d} E_{Q_j}(E(Q_{ij} \mid Q_j)).$$

Since the columns of S and the marginal distribution of the rows of S are identically distributed

$$E(X_{LHS}) = \sum_{i=1}^{n} \prod_{j=1}^{d} (p^* \frac{m}{n} + (1-p^*)\frac{m-1}{n}) = n\,(\frac{p^*}{n} + \frac{m-1}{n})^d = np^d.$$

Assume Q_j is binomially distributed with probability $p^* = np - (m-1)$, then $E(X_{LHS}) = E(X_{SRS})$

To compute $Var(X_{LHS})$ it remains to compute $E(X_{LHS}^2)$.

$$E(X_{LHS}^2) = \sum_{i=1}^{n}\prod_{j=1}^{d} E_{Q_j} E(Q_{ij}^2 \mid Q_j) + 2\sum_{i<k}\prod_{j=1}^{d} E_{Q_j} E(Q_{ij}Q_{kj} \mid Q_j)$$

$$= E(X) + n(n-1)(p^* \frac{m(m-1)}{n(n-1)} + (1-p^*)\frac{(m-1)(m-2)}{n(n-1)})^d .$$

Then

$$Var(X_{LHS}) = np^d + n(n-1)(\frac{(m-1)}{n(n-1)})^d (2np - m)^d - (np^d)^2.$$

To prove that $Var(X_{LHS})$ is uniformly less than $Var(X_{SRS})$ it needs to be shown that $Var(X_{SRS}) \geq Var(X_{LHS})$ for all p which follows if

$$np^d(1-p^d) \geq np^d + n(n-1)(\frac{(m-1)}{n(n-1)})^d (2np - m)^d - (np^d)^2.$$

This can be reduced to

$$p^{2d} \geq (\frac{m-1}{n})^d (\frac{2np - m}{n-1})^d \tag{4.3.2}$$

and can be rewritten as

$$f(p) = n(n-1)p^2 - 2np(m-1) + m^2 - m \geq 0.$$

The first and second derivatives of $f(p)$ show that the extremum is a minimum and is found at $p = (m-1)/(n-1)$; the value of $f(p)$ at the minimum is $n-m$ which is greater than zero and the inequality is shown to be correct.

This result shows that $Var(X_{LHS}) \leq Var(X_{SRS})$ over all randomly placed cubic regions. The result is readily extended to any rectangular region by replacing p by p_j, p^* by p_j^* and m by m_j in the equations above. The equations remain fundamentally the same and equation (4.3.2) can be rewritten as

$$\prod_{j=1}^{d} p_j^2 \geq \prod_{j=1}^{d} (\frac{m_j - 1}{n}) \prod_{j=1}^{d} (\frac{2np_j - m_j}{n-1}).$$

Since the result holds for any p such that $(m-1)/n \leq p \leq m/n$ when $d = 1$, then it also holds for the product when the p_j's are different.

The result also holds for any given (nonrandomly placed) rectangular region by using the design S^t where $s_{ij}^t = s_{ij} + \gamma_j$ and γ_j is a random variable with a uniform distribution on $[-1/2n, 1/2n]$.

4.3.2 Behavior of Latin Hypercube Sampling

For the results in this section assume that V^d is cubic with sides of length p. Variance is a function of n, p, and d and p^d is the percent volume of $[0,1]^d$ which the randomly placed cubic region covers. For SRS, variance is only a function of n and $v = p^d$. This is not the case for LHS, but from a few empirical studies and examination of (4.1.1) it is apparent that changing d has only a minimal effect on the results if n and v are fixed. Figures 4.1 and 4.2 show the typical shape for the standard deviations of X for LHS and SRS in relation to n and v respectively. Figure 4.2 shows that $Var(X_{LHS})$ is nearly the same as $Var(X_{SRS})$ for small regions V^d where the chances of any points falling in V^d are small; then well before $v = 0.5$ the two variances diverge only to converge again at $v = 1.0$.

The relative efficiency shown in Figures 4.3 and 4.4 is the ratio of standard deviation of X_{SRS} to X_{LHS}. Figure 4.3 shows that as $n \to \infty$ the relative efficiency has a maximum limit which it approaches fairly quickly. This behavior is true for all values of v. Figure 4.4 is a plot of this limit of relative efficiency versus v. The figure shows that the amount of clustering of design points in LHS becomes relatively less than that for SRS as the scale of observation increases. The relative efficiency quickly drops to one for $0.98 < v \leq 1.0$.

The scatter of points in Figure 4.3 is due to oscillatory behavior of $Var(X_{LHS})$ as a function of n. The reason for this can be seen from the expansion

$$Var(X_{LHS}) = E_U \; Var(X_{LHS} \mid U) + Var_U(E(X_{LHS} \mid U)).$$

Since U is binomial with probability p^*, $Var_U(E(X_{LHS} \mid U))$ is a quadratic function for $(m-1)/n \leq p \leq m/n$ and

$$Var(X_{LHS}) = E_U \; Var(X_{LHS} \mid U) \text{ for } p^* = 0 \text{ or } 1.$$

The oscillations dampen to the limit as $n \to \infty$ since $np \approx [np]$ for all n.

The mean and variance of X_{SRS} is the same whether X_{SRS} is a combination of two samples of size n_1 and n_2 or one sample of size $n_1 + n_2$. This can readily be seen from the equations for the mean and variance. This is true for the mean of X_{LHS} but is not the case for the variance. However, Table 4.1 shows that the variance for two combined samples is only slightly larger than a single sample. The efficiency will vary somewhat given different n and v

Std. Dev. vs. Sample Size

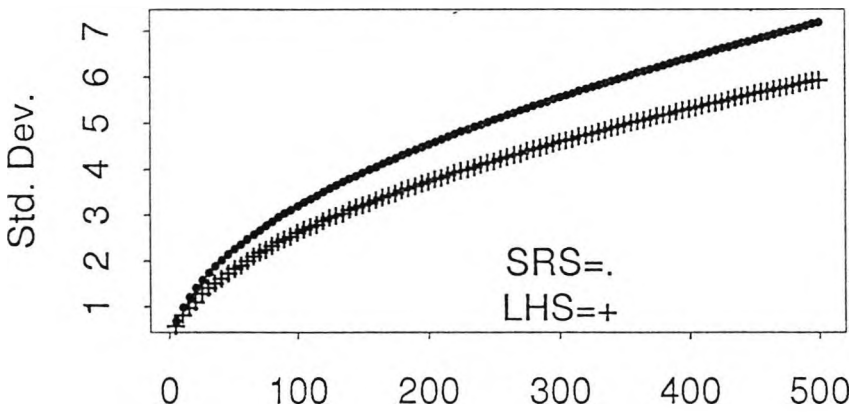


Fig. 4.1

Std. Dev. vs. % Volume

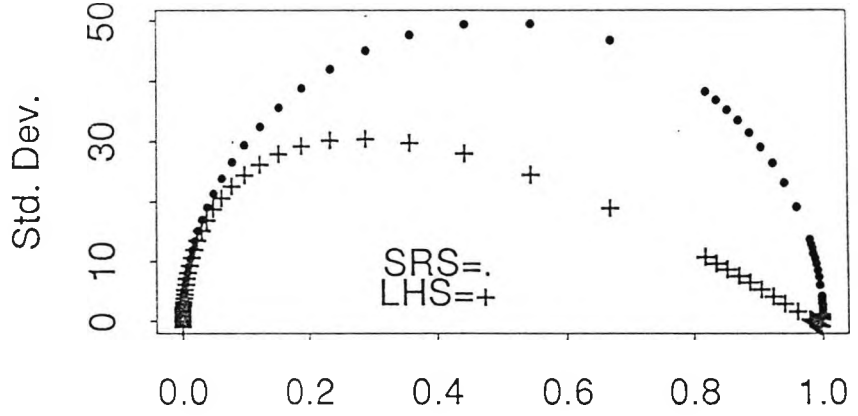


Fig. 4.2

Efficiency vs. Sample Size

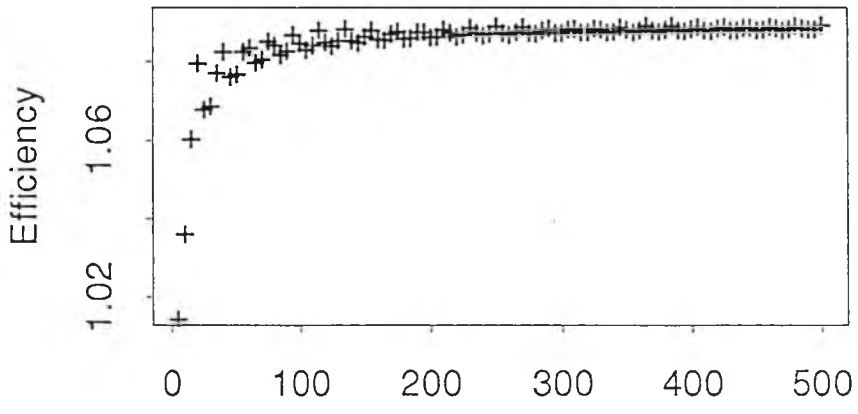


Fig. 4.3

Efficiency vs. % Volume

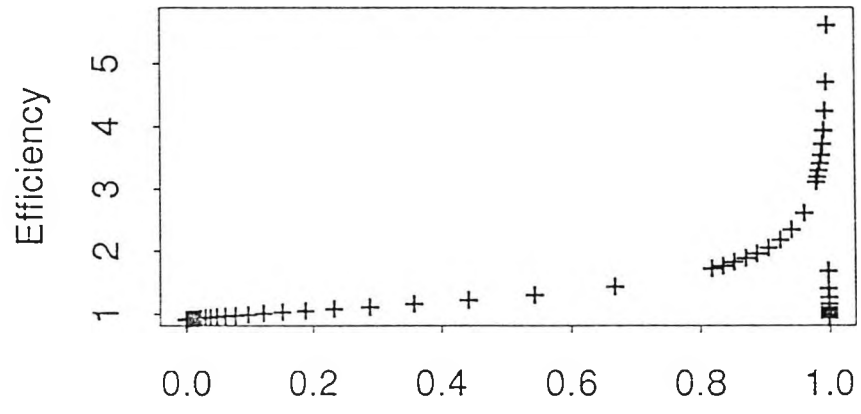


Fig. 4.4

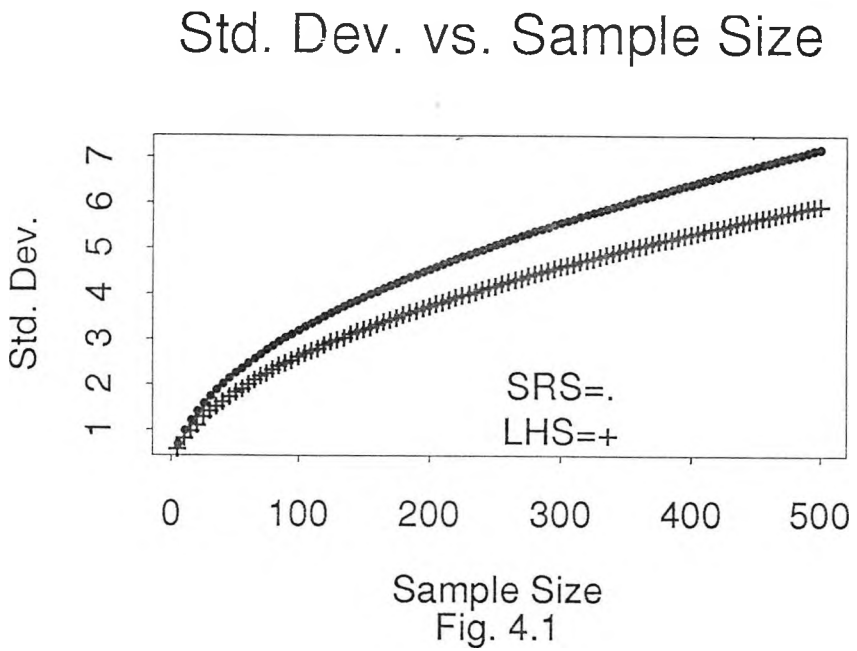


Fig. 4.1

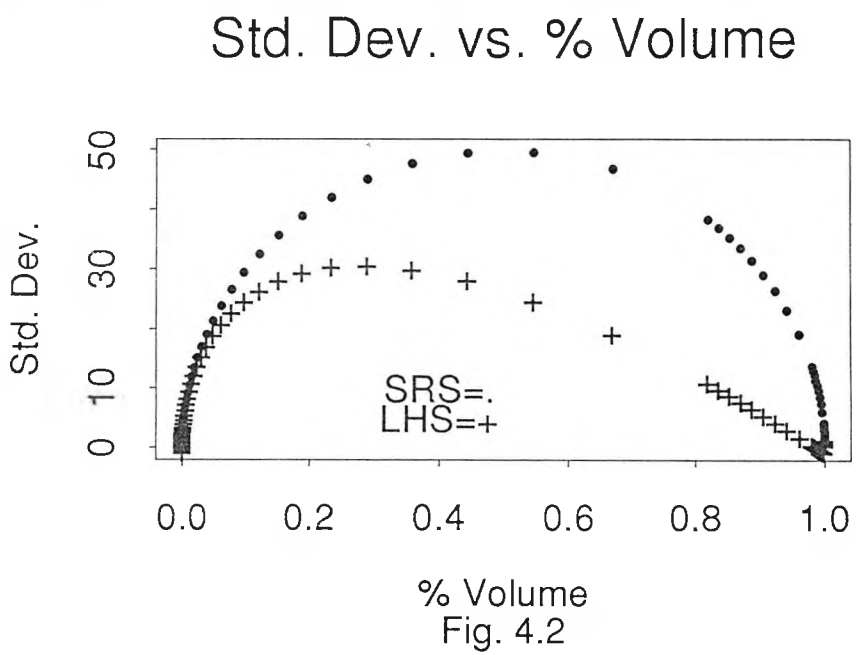


Fig. 4.2

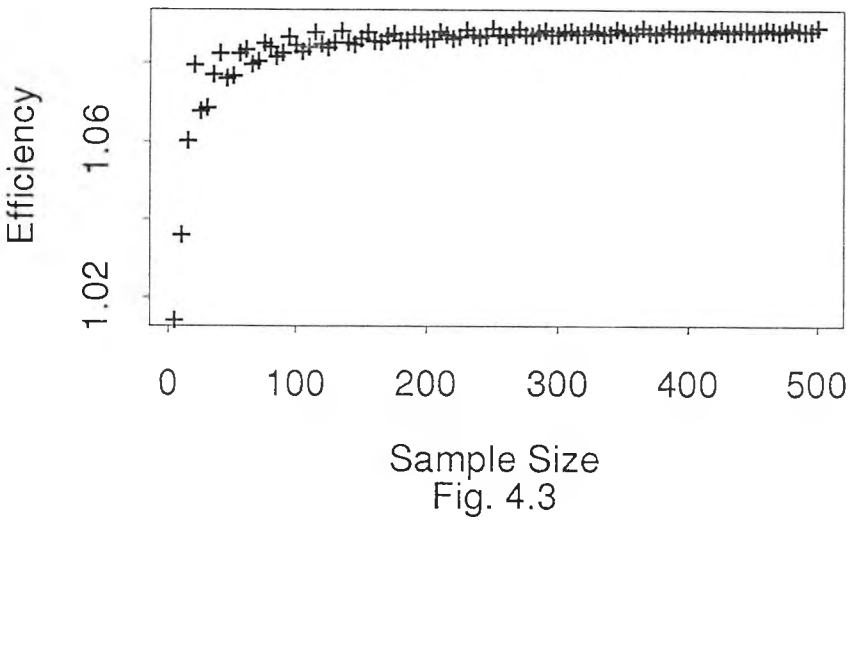


Fig. 4.3

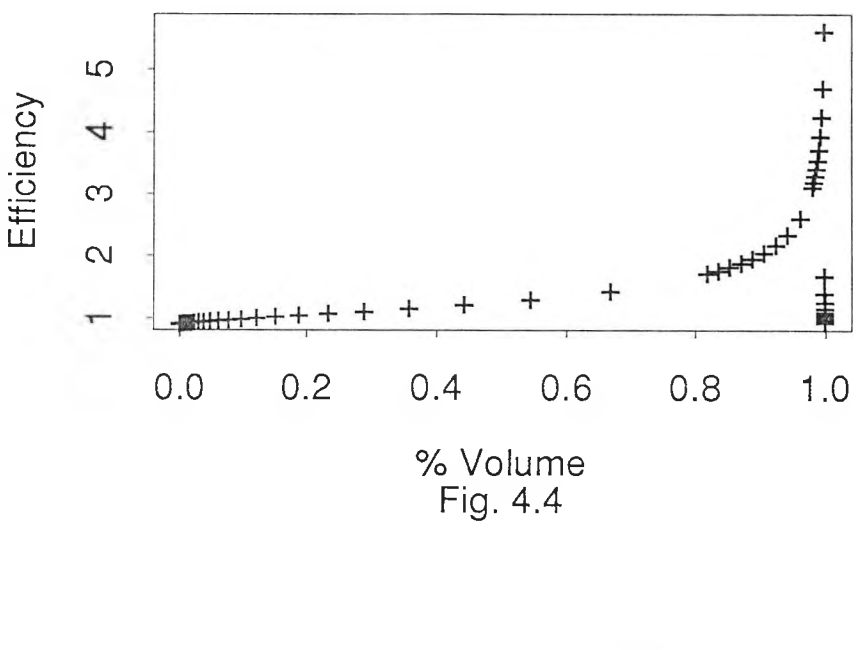


Fig. 4.4

because of the oscillatory behavior of $Var(X_{LHS})$.

	% Volume with d=10		
n	0.1	0.4	0.7
29	1.89	3.04	1.03
58	3.75	5.27	2.04
116	7.46	10.44	4.04
232	14.89	20.64	8.01

Table 4.1: Variance of Combined LHS

4.4 Conclusion

Latin hypercube sampling was developed to improve the variance of simulation study estimates. LHS has been shown to have better variances for these estimates than simple random sampling. LHS has also been used as the experimental design for model estimation for linear models and Gaussian stochastic processes. The benefits of using LHS for model estimation is that it tests each variable at many different input values. The variance of these model estimates have also been shown to be better than for SRS, especially if the models are nearly linear.

The work in this chapter uses the variance of the number of design points in an arbitrary region to measure the uniformity of a design and shows that LHS cover the experimental space more completely and more uniformly than SRS. It also shows that several small LHS, with a total number of n runs, have nearly the same space filling capacity as one large LHS with n runs.

4.5 References

1. Steinberg, D. M. and Hunter, W. G., (1984) "Experimental Design: Review and Comment (with discussion)," *Technometrics*, 26, 71-130.
2. Box, G. E. P. and Draper, N. R., (1987) *Empirical Model Building and Response Surfaces* , John Wiley & Sons , New York, N.Y. .
3. Box, G. E. P., Hunter, W. G., and Hunter, J. S., (1978) *Statistics for Experimenters* , John Wiley , New York .

4. Franklin, M. F., (1985) "Selecting Defining Contrasts and Confounded Effects in p^{n-m} Factorial Experiments," *Technometrics*, 27, 165-172.

5. Wynn, H. P., Sivaloganathan, S., Buck, R. J., and Lewis, S. M., (1992) "Generate: An Algorithm for the Computer Generation of Orthogonal Arrays with Specific Alias Structure," Engineering Design and Quality Centre Technical Report #30.

6. Plackett, R. L. and Burman, J. P., (1946) "The Design of Optimum Multifactorial Experiments," *Biometrika*, 33, 305-325.

7. Wang, J. C. and Wu, C. F. J., (1991) "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *Journal of the American Statistical Association*, 86, 450-456.

8. Wang, J. C. and Wu, C. F. J., (1992) "Nearly Orthogonal Arrays with Mixed Levels and Small Runs," *Technometrics*, 34, 409-422.

9. Sacks, J. and Ylvisaker, D., (1970) "Statistical Designs and Integral Approximation," in *Proc. 12th Bien. Sem. Canad. Math. Congress*, ed. R. Pyke, pp. 115-136, Canadian Mathematical Congress, Montreal.

10. Ylvisaker, D., (1975) "Designs on Random Fields," in *A Survey of Statistical Design and Linear Models*, ed. J. N. Srivastava, pp. 593-607, North-Holland, Amsterdam.

11. Davis, P. J. and Rabinowitz, P., (1984) *Methods of Numerical Integration,* 2nd ed., Academic, Orlando, Fla..

12. McKay, M. D., Conover, W. J., and Beckman, R. J., (1979) "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, 21, 239-245.

13. Patterson, H. D., (1954) "The Errors of Lattice Sampling," *Journal of the Royal Statistical Society - Series B*, 16, 140-149.

14. Iman, R. L. and Conover, W. J., (1980) "Small Sample Sensitivity Analysis Techniques for Computer Models, With Application to Risk Assessment (with discussion)," *Communications in Statistics - Theory and Methods*, A9, 1749-1874.

15. Downing, D. J., Gardner, R. H., and Hoffman, F. O., (1985) "An Examination of Response Surface Methodologies for Uncertainty Analysis in Assessment Models," *Technometrics*, 27, 151-163.

16. Iman, R. L. and Conover, W. J., (1982) ''A Distribution-Free Approach to Inducing Rank Correlation Among Input Variables,'' *Communications in Statistics- Simulation and Computation*, 11, 311-334.

17. Handcock, M. S., (1991) ''On Cascading Latin Hypercube Designs and Additive Models for Experiments,'' *Communications in Statistics-Theory and Methods*, 20, 417-440.

18. Stein, M., (1987) ''Large Sample Properties of Simulations Using Latin Hypercube Sampling,'' *Technometrics*, 29, 143-151.

19. Owen, A. B., (1992) ''A Central Limit Theorem for Latin Hypercube Sampling,'' *Journal of the Royal Statistical Society - Series B*, 54, 541-552.

20. Owen, A. B., (1991) ''Orthogonal Arrays for Computer Experiments, Integration and Visualization,'' *Technical Report, University of Stanford.*

21. Owen, A. B., (1992) ''Lattice Sampling Revisited: Monte Carlo Variance of Means Over Randomized Orthogonal Arrays,'' Stanford University Technical Report.

- Chapter 5 -

Numerical Optimization Algorithms

5.1 Introduction

To avoid having the topic of optimization repeating itself at various points in other chapters a description and discussion of the numerical optimization algorithms used in this thesis has been left to this chapter. The robust engineering design method described in Chapter 3 uses numerical optimization algorithms at two separate times: for maximization of the likelihood equation and when searching for a robust engineering design with the statistical model as a cheap emulator of the simulation model. The MLE optimization problem is a univariate optimization problem but has a large number of inputs. The search for optimal engineering designs in the described examples were multivariate optimization problems with a moderate to large number of inputs.

Although these optimization problems play an integral part in developing the predictor and finding optimal engineering designs, we did not have the inclination or expertise to develop new optimization algorithms. Our goal was to find an optimization algorithm that was already available and achieved good results. During the investigation of the examples in Chapter 6, several different optimization algorithms were used. A brief outline and discussion of the algorithms are given in Section 5.2-5.4. The rest of this section gives a brief overview of loss functions, optimization algorithms and their relationship.

Numerical optimization algorithms search for the minimum (maximum) of the objective function given possible constraints on (functions of) the inputs. The objective function for model parameter estimation is the likelihood function as stated in (3.1.1). This objective function has a univariate response. The objective function for optimization of the engineering design is the loss function. As mentioned in Section 1.3, the loss function may be a multivariate response function.

Two ways to handle problems with a multivariate response objective function are: to create a univariate response from the multiple responses, or to treat one of the responses, or a function of several responses, as the target to be minimized and the rest held within stated boundaries or constraints. Frequently,

the organization of the response vector into targets and constraints follows readily from the specification description. Several possibilities for the loss function were considered from these two approaches, especially for the voltage-shifter circuit example in Chapter 6. However, a thorough study for the best loss function was not carried out because it was beyond the scope of the main focus of our research.

The optimization algorithm that can be used for finding the optimal solution depends on the objective function. For example, the Nelder-Mead simplex method does not allow constraints on the inputs, while the adaptive random search (ARS) algorithm and the non-linear programming method, NPSOL, do allow constraints. The Nelder-Mead simplex algorithm has been used for computing maximum likelihood estimates throughout the research. For multivariate objective functions a univariate response needs to be created from the multivariate response if the Nelder-Mead simplex is to be used. If constraints are added then ARS or NPSOL can be used.

There is some common notation for all algorithms. Let $\mathbf{x}$ be the d-dimensional vector of inputs over which the function $y(\mathbf{x})$ is to be optimized, where $y(\mathbf{x})$ may be a multivariate response.

5.2 Nelder-Mead Simplex

The Nelder-Mead simplex is an optimization algorithm which requires a univariate response objective function and no constraints on the inputs. This optimization algorithm is used to compute the maximum likelihood estimates for the model described in Chapter 2. Since the correlation parameters are constrained, $\theta \geq 0$ and $1 \leq p \leq 2$, and the Nelder-Mead simplex algorithm requires no constraints, the correlation parameters are translated to $\theta' = \ln \theta$ and $p' = \sin^{-1}(2*(p-1.5))$ during the search part of the optimization.

The basic idea of the Nelder-Mead simplex algorithm is to take $d+1$ points, $\mathbf{x}_1, \ldots, \mathbf{x}_{d+1}$, in the input space and replace the highest point, $\mathbf{x}_H$ with $\mathbf{x}_H = \alpha \mathbf{x}_H$, where $\alpha = -1, 0 < \alpha < 1$, or $\alpha > 1$ or a suitable combination of the three. The different α's are referred to as a reflection, contraction or expansion. A stepwise outline of the algorithm as loosely described in Numerical Recipes is:

1. Starting point $\mathbf{x}_0$ and

$$\mathbf{x}_i = \mathbf{x}_0 + \lambda \mathbf{e}_i, i = 1, \ldots, d.$$

2. Compute reflection. If there is an improvement then try expansion, if not try contraction.

3. If no movement in Step 2. try multiple contraction around the value of $\mathbf{x}$ which gives a minimum.

4. Repeat Steps 2 and 3 until stopping rule.

Two stopping rules: either the change in $\mathbf{x}$ from one step to the next is less than *tol* or the value of $y(\mathbf{x})$ from one step to the next is less than *ftol*, where *tol* and *ftol* are user defined tolerances.

This algorithm will always find a local minimum, but does not necessarily find the global minimum, a common problem with optimization algorithms. To improve the chances that the minimum found is a global minimum it is suggested that the algorithm be run several times, each time from a different starting point. From personal experience starting from 3 to 5 different points is usually sufficient.

Two different versions of this algorithm were used at various times for computing maximum likelihood estimates, the subroutine E04CCF from the NAG library and AMOEBA from Numerical Recipes.[1] From informal comparisons of the two algorithms we decided that the AMOEBA version of the algorithm was more efficient and regularly found better solutions than the NAG version.

5.3 Adaptive Random Search

The first algorithm used for the engineering design optimization problem with constraints was adaptive random search[2 3 4] as described by Pronzato, *et al.*[5] As the name implies this is a global random optimization algorithm. Most random search algorithms are an iteration of two steps: generate new observations from some distribution and select the best setting from the new observations according to a given rule and repeat with new observations.

The adaptive random search algorithm chooses its new observations from distributions with different variances to create ever decreasing search areas. The best of the search areas is then investigated more closely before the procedure repeats itself. Let $\mathbf{z} = \mathbf{x} + \mathbf{r}$ where $\mathbf{r}$ is a random vector, normally distributed with mean zero and variance

$$\Sigma(\sigma) = \text{diag}\ [\sigma_1, \sigma_2, \ldots, \sigma_d].$$

Let $\sigma^{(1)} = \mathbf{x}_{\max} - \mathbf{x}_{\min}$, where $\mathbf{x}_{\max}$ and $\mathbf{x}_{\min}$ are the upper and lower bounds of

the input space and $\sigma^{(i)}=0.1^{i-1}\sigma^{(1)}$, $i=2,\ldots,f_1$. An outline for the algorithm is:

1. Select $\mathbf{x}_{max}$ and $\mathbf{x}_{min}$ and set the starting point $\mathbf{x}_0=0.0$. Let $k=0$.
2. For $j=1,\ldots,f_1$:

   Generate $f_2(j)$ observations of $\mathbf{y}(\mathbf{x}_k+\mathbf{r})$ where $\mathbf{r}$ is normally distributed with variance $\Sigma(\sigma^{(j)})$. If $\mathbf{z}$ is outside the input space then $\mathbf{x}$ is ignored. Let $\mathbf{x}_k$ equal the best set of inputs, $\mathbf{z}$, from the new observations. Let $k=k+1$.
3. For the most successful $\sigma^{(j)}$, generate f_4 new observations of $\mathbf{y}(\mathbf{x}_k+\mathbf{r})$ and let $\mathbf{x}_k$ equal the best set of inputs, $\mathbf{z}$, from the new observations. Let $k=k+1$.
4. Repeat Steps 2 and 3 until a stopping rule has been reached. There is a series of three stopping rules, the order of implementation is: Stop if
   a. a maximum number of iterations is reached.
   b. $\sigma^{(f_1)}$ has been selected f_5 times.
   c. $\mathbf{y}(\mathbf{x})$ has reached some predetermined criterion.

The last rule typically is not implemented because of the difficulty in establishing criteria approximating the optimal solution.

There are a number of parameters in the algorithm that need to be assigned. We used the values suggested in Pronzato, *et al.*: $f_1=f_5=5$, $f_2(i)=f_3/i$, and $f_3=f_4=100$. Since the input variables are scaled to [−0.5,0.5] for our prediction models, $f_1=4$ was also used because $f_1=5$ did not generate a lot of movement relative to the required accuracy in $\mathbf{x}$.

The adaptive random search algorithm can handle any type of objective function and constraint problem, as long as appropriate decision making rules are coded, because the new observations are selected at random. It is only a matter of testing whether the new point fulfills the criterion better than previously selected points. If the new point is better then the previous point it is kept for reference and additional points are tested to try to improve on it.

5.4 NPSOL

A method for handling optimization problems with a smooth nonlinear objective function and both linear and nonlinear constraints is a nonlinear programming algorithm which has several acronyms, but will be referred to here as NPSOL. The implementation of the algorithm used is the NAG version of

NPSOL, subroutine E04UCF. The description that follows is an overview of the NAG manual description. See also Gill, *et al.*[6]

A formal construction of the problems which NPSOL is designed to solve can be stated as

$$\text{(5.4.1)} \qquad \text{Minimize } y(\mathbf{x}) \text{ subject to: } l \le \begin{Bmatrix} \mathbf{x} \\ A\mathbf{x} \\ c(\mathbf{x}) \end{Bmatrix} \le u,$$

where $y(\mathbf{x})$ is the objective function and $\mathbf{x}$, A, and $c(\mathbf{x})$ are the bound, linear and nonlinear constraint functions respectively.

An initial estimate of the solution is given and the routine first finds a solution that satisfies the bound and linear constraints. Once this occurs a series of major and minor iterations are carried out. The major iterations are used to find the optimal solutions, while the minor iterations are used to find solutions to quadratic programming subproblems needed for the major iteration. Derivatives for $y(\mathbf{x})$ and $c(\mathbf{x})$ are requested for the algorithm, but finite differences are used to compute estimates for those derivatives that are not furnished.

The major iterations create a sequence $\{\mathbf{x}_k\}$ that converge to $\mathbf{x}^*$, a first order Kuhn-Tucker point of (5.4.1). The sequence is of the form $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{p}$, where x_k is the current value, $\alpha > 0$ is the step length and $\mathbf{p}$ is the search direction. Given the search direction the major iteration calculates a steplength α that produces a sufficient decrease in an augmented Lagrangian merit function.

The search direction is determined from the solution of the quadratic programming subproblem and is the minor iteration of NPSOL. The quadratic programming subproblem can be specified as

$$\text{Minimize } g'\mathbf{p} + \frac{1}{2}\mathbf{p}'H\mathbf{p} \quad \text{subject to} \quad l^* \le \begin{Bmatrix} \mathbf{p} \\ A\mathbf{p} \\ A_N\mathbf{p} \end{Bmatrix} \le u^*,$$

where g is the gradient of $y(\mathbf{x})$ at $\mathbf{x}$, H is an approximation to the Hessian of the Lagrangian function, A_N is the Jacobian matrix of $c(\mathbf{x})$ and l^* and u^* are new bounds, which are a function of l and u and the constraints in (5.4.1).

5.5 Conclusion

Most deterministic algorithms only guarantee a local optimum and several starting points are used to help ensure a global optimum. This is not necessary for the ARS algorithm because the algorithm moves around the input space in a random manner. Eliminating multiple starting points reduces the total number of observations that may be used. From casual observations we felt that ARS did not consistently give as good results as AMOEBA or NPSOL.

Numerical optimization methods have not received much attention in the discussion about robust engineering design methods, but there are many benefits to their use. The use of existing optimization algorithms can lead to better solutions more quickly than the sequential improvement philosophy of Taguchi. The search for the best optimization algorithm was not exhaustive by any means and an improvement in the optimization algorithm would immediately improve the efficiency of the search for maximum likelihood estimates and optimal engineering designs.

5.6 References

1. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., (1986) *Numerical Recipes,* Cambridge University Press, Cambridge.

2. Andronikou, A. M., Bekey, G. A., and Masri, S. F., (1982) ''Identification of Nonlinear Hysteretic Systems using Random Search,'' *Proceedings 6th IFAC Symposium on Identification and System Parameter Estimation*, 1, 263-268.

3. Bekey, G. A. and Masri, S. F., (1983) ''Random Search Techniques for Optimization of Nonlinear Systems with Many Parameters,'' *Math. Comput. Simulation*, 25, 210-213.

4. Masri, S. F., Bekey, G. A., and Safford, F. B., (1976) ''An Adaptive Random Search Method for Identification of Large-Scale Nonlinear Systems,'' *Proceedings 4th IFAC Symposium on Identification and System Parameter Estimation*, 246-255.

5. Pronzato, L., Walter, E., Venot, A., and Lebruchec, J., (1984) ''A General-Purpose Global Optimizer: Implementation and Applications,'' *Mathematics and Computers in Simulation*, 26, 412-422.

6. Gill, P. E., Hammarling, S. J., Murray, W., Saunders, M. A., and Wright, M. H., (1986) "User's Guide to LSSOL (Version 1.0)," Dept. of Operations Research, Stanford University (Report SOL 86-1), Stanford University.

- Chapter 6 -

Case Studies in Robust Engineering Design

6.1 Introduction

A major problem for integrated circuit designers is how to design circuits so that performances, as predicted by a circuit simulator, are insensitive to uncontrollable variations in the manufacturing process and in the operating conditions. Statistical circuit design methods [1] attempt to optimize the performances of a circuit design in the presence of these uncontrollable variations.

Statistical modeling of circuit and process simulators to achieve consistently good performance has become increasingly prominent.[2 3 4] In some examples,[3 4] circuit performances are modeled as quadratic functions of all the input factors of interest: the designable factors, the uncontrollable statistical variations, and the operating conditions. The statistical model is fitted from relatively few runs of the simulator. A key characteristic of these approaches is that the statistical model can be used as a computationally cheap surrogate for maximizing yield, minimizing variations of performances around targets, or optimizing other measures of quality.

These and other existing tools for statistical circuit design work well when the number of circuit factors is small, no more than about ten, and the space of factor values is sufficiently restricted to admit simple modeling. With more factors, the number of circuit simulations required to fit quadratic models can exceed practical limits. Also, if the factors are allowed wide ranges the simple quadratic models may not be effective for approximating the performance functions.[3 4] The methods of Taguchi[5] for finding robust designs likewise appear inadequate for treating many nonlinear situations.[3] In these circumstances optimization is a formidable task. All standard routines have difficulties that inhibit or prohibit their use. Non-differentiability of performance functions stops some. Failure to converge and getting trapped at local optima are common occurrences. Many have trouble incorporating constraints on the performance functions. Most serious is that all require large numbers of function evaluations.

Unless function evaluations are cheap, direct application of an optimization routine to the simulated performances is not feasible. Optimizing via an inexpensive approximation to the simulator is more feasible, but the approximation itself must be accurate for reliable optimization. Unfortunately, these two requirements---few simulator runs and an accurate approximation---conflict with each other.

These difficulties are overcome by multi-stage experimentation and, during each stage, by building a statistical model (predictor) of the simulator. The advantages of sequential experimentation for optimization are well known, e.g. Box and Draper,[6] in which the term "evolutionary operation" is used. Such methods have typically relied on the adequacy of simple models over small factor ranges to indicate a local direction of improvement. Instead, a predictor is built on the region of interest and obtain sub-ranges of the factors where the optimum is predicted to lie. On the smaller region we build a more accurate model and continue the search. To cope with many factors, their large ranges, and, hence, complex performances, a class of approximating functions (predictors) that is highly data-adaptive is used. These predictors often have much less error than polynomial models.[7] [8] This blend of sequential experimentation and modeling allows the optimum to be found in a few stages (usually 2 or 3) and with comparatively few simulation runs.

In outline the approach is:

Step 1. Postulate a statistical model for each performance.

Step 2. Plan an experiment and run the simulator to collect the data.

Step 3. Use the data to fit the models.

Step 4. Check the accuracy of prediction and plot the factor effects.

Step 5. If the models are insufficiently accurate, choose a subregion for the next experiment and return to Step 1.

Step 6. When the models are sufficiently accurate, optimize the objective (loss, yield, etc.,) using the fitted model in place of the performance functions.

Section 6.2 formulates the problem and elaborates on these steps. In Section 6.3 and 6.4 these techniques are applied to multiple performances and criteria, and incorporate manufacturing variations. The example used in Section 6.3 is that of a GaAs voltage shifter circuit. In Section 6.4 the methods are applied

to a digital logic circuit with 20 performances of interest, but with no manufacturing variations. In Section 6.5 the methods and the results of their applications are discussed.

6.2 Modeling and Optimization

Circuit performances generally depend on design factors, operating conditions (environmental noise factors), and on statistical variations of device factors (uncontrollable manufacturing noise factors). Some designable factors may have uncontrollable variations superimposed. We focus initially on a single performance, denoted by y. Let $\mathbf{X}=(X_1,...,X_d)$ denote the d-dimensional vector of varying input factors to the circuit simulator, all the other inputs remaining fixed. We typically normalize each X_i to lie in [-0.5, 0.5]. We write $X_i=c_i+U_i$ to differentiate between the controllable and uncontrollable components of X_i. If an input factor has no designable adjustment, then c_i has a fixed value (make it 0) and is ignored. Similarly, if there is no uncontrollable variation, then $U_i=0$. The performance y is, therefore, a function of $\mathbf{X}=\mathbf{c}+\mathbf{U}$, where $\mathbf{c}=(c_1,...,c_d)$ and $\mathbf{U}=(U_1,...,U_d)$. Finally, let $\mathbf{x}$ be a realization of $\mathbf{X}$.

We adopt the Taguchi[5] objective of minimizing a "loss", for example a measure of variability around a target performance. For recent accounts of Taguchi methods see Dehnad[9] , Phadke[10] or Chapters 1.2-1.4, 7. The particular loss used invariably depends on the problem. For the example of Section 6.3, the target value of the output voltage is 5 V and it is the fluctuation around 5 V due to $\mathbf{U}$ that we want to minimize. This suggests the loss structure

$$L_{MAX}(\mathbf{c}) = \int |y(\mathbf{c}+\mathbf{U})-5|^2 \, d\Gamma(\mathbf{U}), \tag{6.2.1}$$

where $\Gamma(\mathbf{U})$ is the noise factor distribution and the ultimate objective to minimize this loss by choice of $\mathbf{c}$. A more complicated loss involving many more performance functions and criteria is exemplified in Section 6.4.

We now detail the six step scheme outlined in Section 1.

Step 1. Postulate a tentative approximating model for the performance.

Low order polynomial models are well known, but, unless the performance function is simple, which is likely to occur only when the input ranges are small, these models can be misleading. For this and other reasons [7 8] we have adopted a different class of models. A brief overview is given here, but see Chapter 2 for a thorough discussion. If $\mathbf{x}$ is the vector of input factors, let

(6.2.2) $$y(\mathbf{x}) = \beta + Z(\mathbf{x}).$$

Here, $Z(\mathbf{x})$ is a stochastic process having a correlation structure

$$Corr\ (Z(\mathbf{x}),Z(\mathbf{w})) = R(\mathbf{x},\mathbf{w})$$

between the responses at two vectors of inputs $\mathbf{x}$ and $\mathbf{w}$ and having the constant variance

$$Var\ Z(\mathbf{x}) = \sigma^2.$$

We can also include, for example, linear terms:

(6.2.3) $$y(\mathbf{x}) = \beta_0 + \sum_{i=1}^{d} \beta_i x_i + Z(\mathbf{x}).$$

The correlation structure of $Z(\mathbf{x})$ captures the systematic departures from a simple model, such as the constant in (6.2.2). Typically, $Z(\mathbf{x})$ and $Z(\mathbf{w})$ are highly correlated for $\mathbf{x}$ near $\mathbf{w}$. As y is deterministic and is often smooth, Z (through R) should have corresponding properties. The specific R we use is of the form

(6.2.4) $$R(\mathbf{x},\mathbf{w}) = \prod_i \exp(-\theta_i |x_i - w_i|^{p_i}).$$

The correlation constants θ and $\mathbf{p}$ are unknown, as are σ^2 and β or $\beta_1, \ldots, \beta_d$. The θ's are non-negative and the p's are between 1 and 2. These constants are estimated in Step 3.

Whether to include the linear terms in model (6.2.3) can be dealt with in each individual problem. In most cases we have found little advantage to using (6.2.3) so we typically start with (6.2.2) to allow more data to be used for estimating the correlation constants of R (higher order models such as (6.2.3) "use up" degrees of freedom).

Step 2. Plan an experiment and run the simulator to collect the data.

We use a Latin hypercube experimental plan to select the inputs. These plans have some attractive properties for computer experiments. They are simple to generate and cover the experimental region fairly uniformly. See Chapter 4 for a further discussion of Latin hypercube sampling.

The question of sample size is a difficult one. Earlier empirical evidence indicated that the estimation procedure used in Step 3 needs about 3 observations per constant. Since we may want to discard outlying data, some extra

observations should be added at initial stages of experimentation. Discarding data is suspect if the goal is to model performances over the entire region. Our purpose, however, is optimization, so deleting outlying data can be helpful by allowing more accurate fitting on more relevant parts of the region. Criteria for selecting sample sizes are the subject of ongoing research.

Step 3. Use the data to fit the model.

We estimate the correlation constants in R via maximum likelihood and obtain $\hat{y}$. The mathematical details and computational methods are described in Chapters 2 and 3.

Step 4. Check the accuracy of prediction and plot the factor effects.

To measure the accuracy of the predictor $\hat{y}$ from the fitted model, we compute a root mean squared error of prediction using (2.2.4) at a number of randomly selected points in the region. In the example of Section 6.3 we choose 20 points. The range of these errors is a good indicator of the accuracy of the predictor. In the example of Section 6.4 extra runs were not available, so cross-validation estimates of error (3.4.1) were used to estimate the accuracy of the predictor.

To visualize the fitted models, we decompose $\hat{y}$ into a mean value, main effects due to individual factors, and second-order interactions between them.[8] This is described in Section 2.4 as well. The estimated main effect due to x_i is the average of $\hat{y}$ over all factors except x_i minus the mean value. These effects are then plotted. The main effect can provide an excellent, yet simple, indication of how a factor affects the performance. Contour plots of the estimated interaction effects are useful for indicating the joint influence of pairs of factors.

At this point we reach a fork in the procedure. If prediction is sufficiently accurate for the particular problem go to Step 6; otherwise proceed to Step 5.

Step 5. Choose a subregion for the next experiment.

An optimization routine (see Step 6 below) can be used to find the center of the new subregion, while the plots in Step 4 are useful in choosing new limits for the factors. The new region has to be selected to take into account the fact that uncontrollable variations cannot be restricted. Then repeat Steps 1 to 4, with data drawn from the new subregion.

Step 6. Optimize the loss function.

The loss depends on the performance function. We replace y by $\hat{y}$ and seek to optimize the resulting predicted loss. For example, in Section 6.3 we minimize over $\mathbf{c}$, $\hat{L}_{MAX}(\mathbf{c}) = \int |\hat{y}(\mathbf{c}+\mathbf{U})-5|^2 \, d\Gamma(\mathbf{U})$ as an estimate of (6.2.1). Because we commonly meet non-differentiable functions subject to complicated constraints various numerical optimization algorithms have been tried. See Chapter 5 for more details. Inspection of the main effects plots can help choose a starting point. This algorithm is also used at Step 5. After finding an estimate of the optimum we do a confirmatory run. If the confirmatory run is unsatisfactory, we take steps to improve the models; see Section 6.3 for an example. A new stage with further data might be necessary if we can not improve the fit of the models.

When there are multiple performances we model the performances individually following Steps 1, 3 and 4. Only one experimental plan is carried out. Optimization is then performed on a single loss function which combines the multiple criteria.

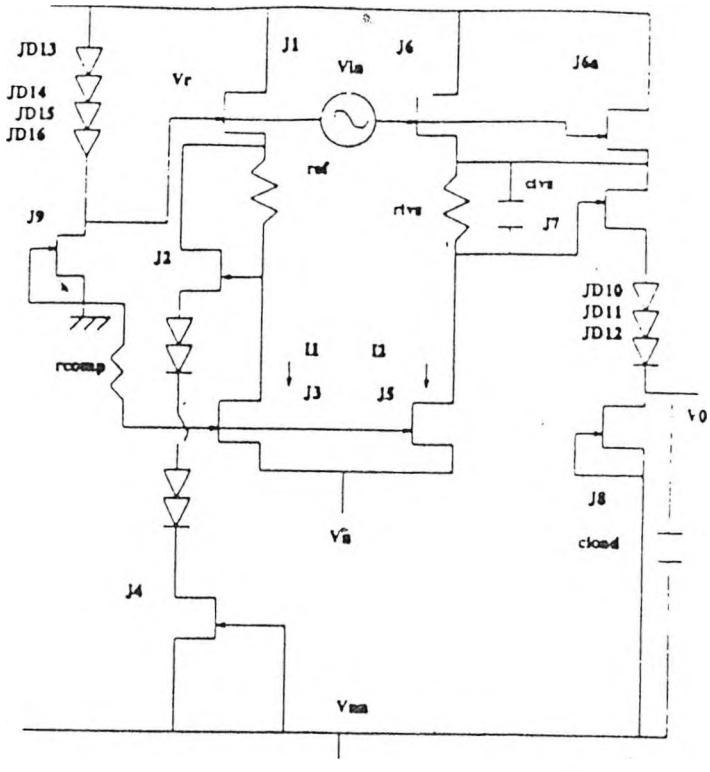
The six steps just described clearly can accommodate other classes of models in Step 1 and other optimizing algorithms in Step 6. We have found that our particular choices make the sequential process efficient.

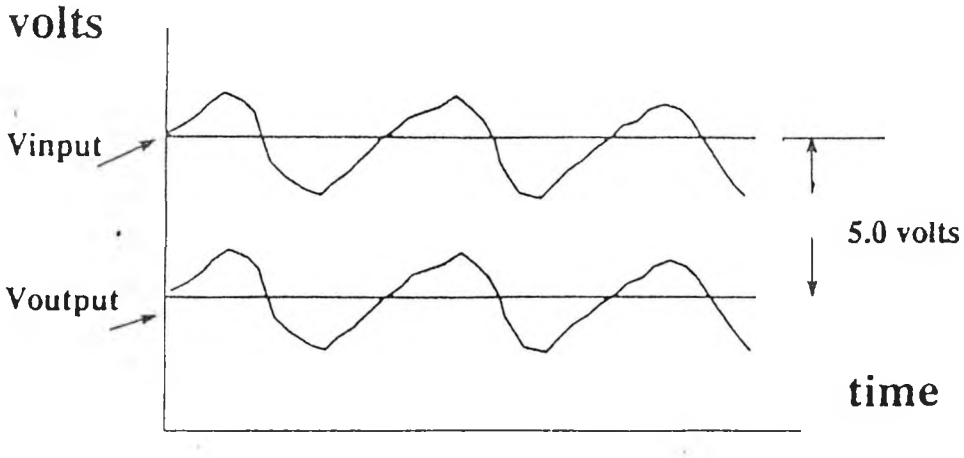
6.3 Voltage-Shifter Circuit Example

Figure 6.1(a) shows a GaAs voltage-shifter circuit. It ideally shifts the circuit input signal by 5 V as in Figure 6.1(b). Such a circuit can be applied to amplifiers or other larger circuits requiring a level-shifting function; in our case it is used for high-frequency transmission. This means that as well as achieving an accurate level shift of 5 V, the circuit should provide a large AC gain and a broad AC frequency bandwidth so that high-frequency systems can operate properly.

As the bandwidth becomes broader, the waveform of the gain may rise before falling. For stability of the circuit, this ripple effect should be minimized. The objective here is to maximize gain and bandwidth while keeping the voltage shift close to the target of 5 V and minimizing ripple.

Table 6.1 gives the ranges of the 14 varying inputs to the circuit simulator. We use logarithmic scales for some factors. Nine of the factors are designable; of these, log(ref), log(cload), log(clvs), and log(j1) ($x_1, \ldots, x_4$ respectively)


JD13
JD14
JD15
JD16
J1
J6
J6a
Vr
Vin
ref
J9
J2
J7
JD10
JD11
JD12
J3
J5
J8
closed
J4


volts
Vinput
Voutput
5.0 volts
time

have significant additive uncontrollable variations. For example, log(ref) has a total range of [log(0.9 KΩ), log(6.4 KΩ)], but a designable range of [log(1.2 KΩ), log(4.8 KΩ)]. The other designable factors, j7, j9, log(rcomp), log(rlvs/ref) and j6/j1 ($x_5, \ldots, x_9$ respectively), have negligible noise components which were set equal to zero. The noise factors, $x_{10}, \ldots, x_{14}$ are noises in the power supplies (vpp, vn, and vnn), the threshold voltage (jmodvto), and the ohmic resistance (jmodrs). The ranges of all variations correspond to ±3σ. All of the total ranges are later normalized to [-0.5, +0.5].

i	Factor	Range of c_i	Range of u_i
1	log(ref in K Ω)	[log(1.2), log(4.8)]	± log(1.33)
2	log(cload in pF)	[log(0.05), log(0.2)]	± log(2)
3	log(clvs in pF)	[log(0.01), log(1.0)]	± log(2)
4	log(j1 in mm)	[log(0.02), log(0.045)]	± log(1.1)
5	j7 in mm	[0.015, 0.045]	---
6	j9 in mm	[0.0075, 0.0225]	---
7	log(rcomp in K Ω)	[log(1.35), log(5.4)]	---
8	log(rlvs/ref)	[log(0.2475), log(1.0)]	---
9	j6/j1	[0.5, 2.0]	---
10	vpp in V	---	[4.5, 5.5]
11	vn in V	---	[-3.3, -2.7]
12	vnn in V	---	[-5.72, -4.68]
13	jmodvto in V	---	[-0.9375, -0.5625]
14	jmodrs in Ω	---	[1.05, 2.45]

Table 6.1 Input factors and their ranges: Voltage-shifter circuit

The performances we model are

y_1 = log(3dB bandwidth), bandwidth measured in GHz,

y_2 = voltage shift (V),

y_3 = gain in dB, the frequency response at 0.1 GHz.

To monitor ripple we also model $y_4, \ldots, y_9$, the gains at frequencies of 0.191, 0.363, 0.692, 1.318, 2.512, and 4.786 GHz. We use as a numerical measure of the ripple:

(6.3.1) $$RIP = \max_{j=3,\ldots,9} (y_j - y_3).$$

Thus RIP measures the size of the upward fluctuation of the frequency response curve.

The most desirable circuit has maximum bandwidth and gain, a voltage shift of 5 V, and zero ripple. These goals cannot be met because of the uncontrollable variations, and because some of these criteria may conflict. We therefore need a means to combine these performances in order to measure the quality of a particular design.

The route we follow is to form a loss statistic to balance the need for good nominal performance and for low variability over the uncontrollable variations. With the notation of Section 6.2, each y_j is a function of $\mathbf{x} = \mathbf{c} + \mathbf{U}$, where $\mathbf{x} = (x_1, \ldots, x_{14})$, and similarly for $\mathbf{c}$ and $\mathbf{U}$. So by good nominal ($\mathbf{U} = 0$) performance, we mean $y_1(\mathbf{c})$ and $y_3(\mathbf{c})$ large, $|y_2(\mathbf{c}) - 5|$ small, and RIP($\mathbf{c}$) small.

To measure the variabilities of bandwidth and gain around the nominal we use

$$Var(j,\mathbf{c}) = \int (y_j(\mathbf{c}+\mathbf{U}) - y_j(\mathbf{c}))^2 \, d\Gamma(\mathbf{U})$$

for j = 1 and 3. Here Γ is the distribution of $\mathbf{U}$, which we take to be independent normals with standard deviations given by 1/6 of the ranges of the u_i's. We also let

$$Var(2,\mathbf{c}) = \int |y_2(\mathbf{c}+\mathbf{U}) - 5|^2 \, d\Gamma(\mathbf{U})$$

measure the variability of the voltage shift around the target. We do not use a corresponding variability measure for ripple; the first stage experiment suggests it is sufficient to only look at nominal ripple.

Formally, the problem we pose is

(6.3.2) $$\max_{\mathbf{c}} \left[y_1(\mathbf{c}) + y_3(\mathbf{c}) - \sqrt{Var(1,\mathbf{c})} - \sqrt{Var(3,\mathbf{c})} \right]$$

subject to

$$\sqrt{Var(2,\mathbf{c})} \leq 0.1V$$

and

$$RIP(\mathbf{c}) \leq 0.01dB.$$

We now apply the six-step modeling and optimization strategy described in Section 6.2 to the loss function and constraints given above. Each performance y_j is modeled as

$$y_j(\mathbf{x}) = \beta_j + Z_j(\mathbf{x}), \tag{6.3.3}$$

but all the p_i's in (6.2.4) are taken to be equal. Since there are nine responses there are nine sets of correlation constants to be estimated.

At the first stage we select a Latin hypercube experimental design of 75 runs for the 14 input factors. Table 6.2 lists the values of y_1, y_2, y_3, and RIP for the first five runs. There are some very badly behaved circuits. For example, runs 2 and 3 have very low gains. Thirteen points in all have gain $< -7\ dB$, and these outlying data are deleted for the statistical modeling (as noted in Step 2).

Run	Volt	Gain	log(BW)	Ripple
1	5.31	-5.58	9.5992	1.09
2	6.53	-15.51	9.9065	0.00
3	6.60	-19.95	9.4026	0.00
4	4.48	-2.65	9.9210	0.00
5	5.44	-2.37	9.7036	0.54

Table 6.2 Performances for the first five experimental-design points: Voltage-shifter circuit.

Model (6.3.3) is fit separately to each of y_1 ,..., y_9 using the remaining 62 runs (Step 3). Typical root mean squared errors associated with the predictors $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$ are 0.05 log GHz, 0.14 V, and 0.30 dB (Step 4). They are sizeable, suggesting the need to reduce the experimental region to improve accuracy.

To guide the choice of the second-stage experimental region (Step 5) we optimize (6.3.2) with respect to the design factors $\mathbf{c}$ and with y_j replaced by $\hat{y}_j$. The integrals in (6.3.2) for $Var(j,\mathbf{c})$ are estimated from 100 Monte Carlo samples of $\mathbf{U}$ from $\Gamma(\mathbf{U})$. Around this tentative optimum we choose a sub-region for $c_1,...,c_9$ using main-effect plots and interaction plots.

Figures 6.2, 6.3, and 6.4 show the main effects of the 14 inputs on bandwidth, voltage, and gain. As the interaction plots contain no exploitable information, it is sufficient to consider these figures only. There are several notable features:

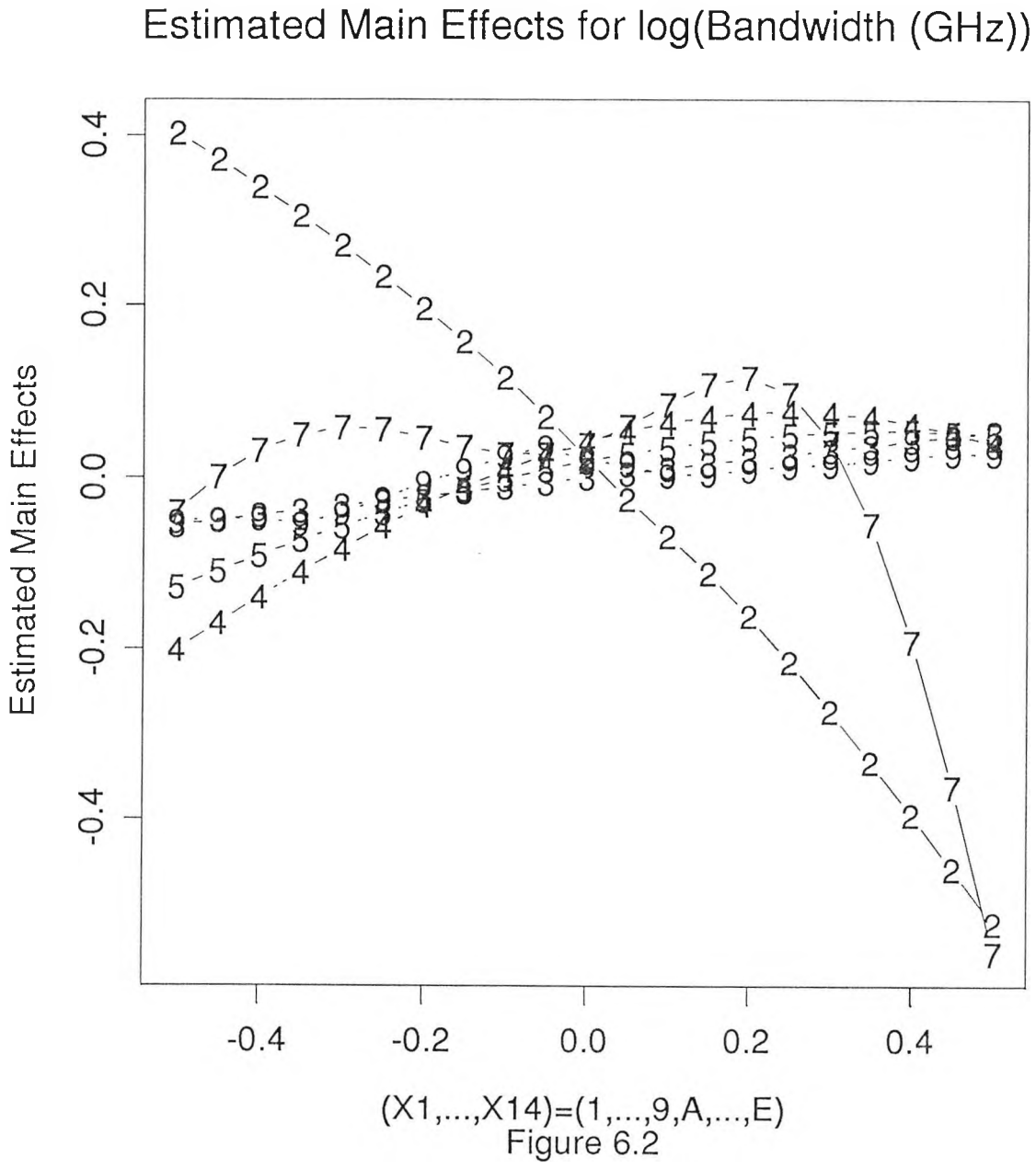
(a) The estimated bandwidth, $\hat{y}_1$, is strongly dependent on cload (x_2), with lower values giving higher values of bandwidth. There is some dependence on j1 (x_4), j7 (x_5), rcomp (x_7), and little effect from the uncontrollable factors $x_{10}, \ldots, x_{14}$.

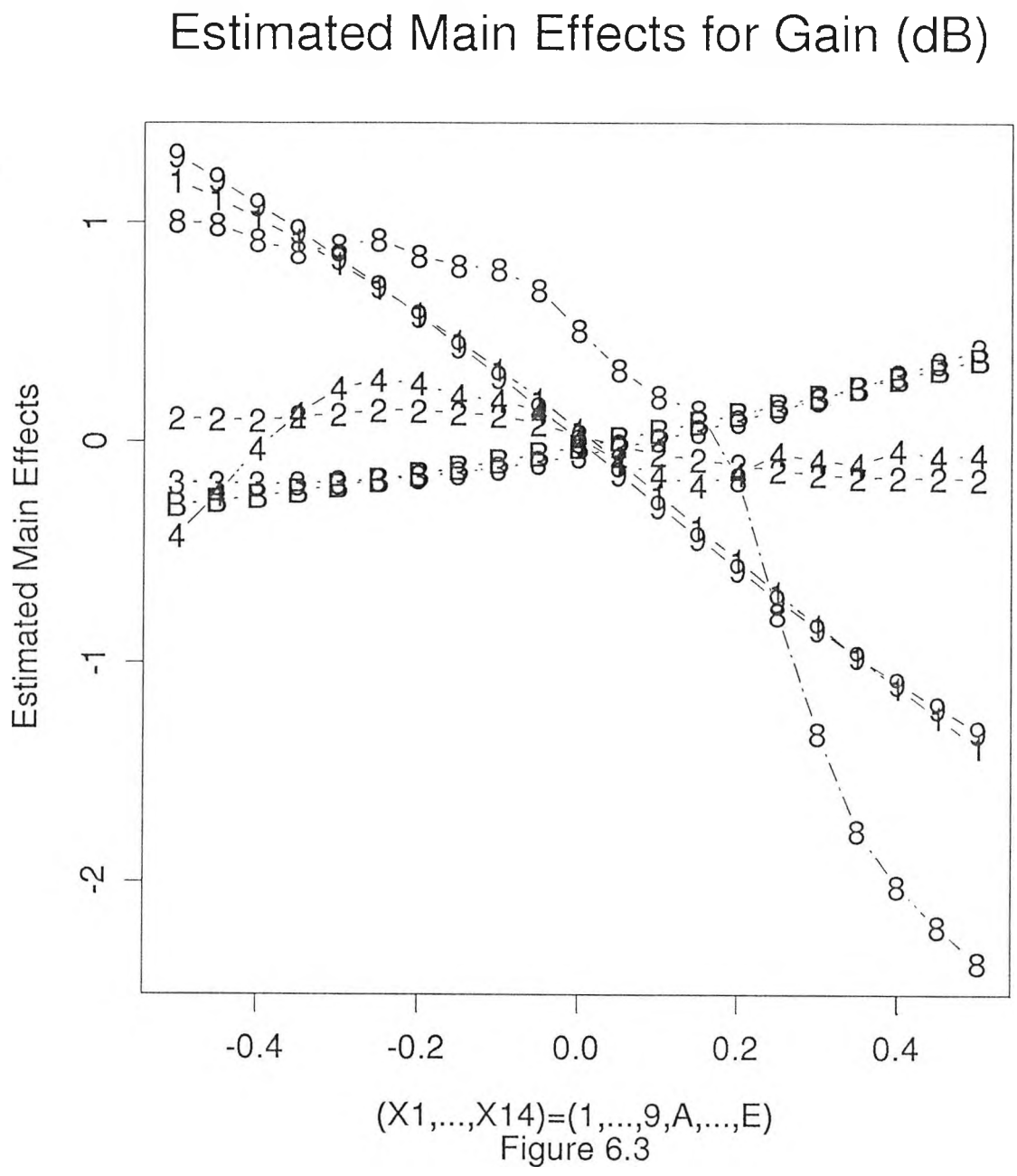
(b) The estimated voltage, $\hat{y}_2$, depends strongly on rlvs/ref (x_8) and j6/j1 (x_9). Also, ref (x_1), j1 (x_4) and rcomp (x_7) have effects of a practical magnitude. This means that there are many tradeoffs between c_8, c_9, etc., that can keep voltage on target. Of considerable importance is the fact that the uncontrollable factors vpp (x_{10}), vn (x_{11}), and jmodvto (x_{13}) have a significant effect on $\hat{y}_2$, indicating potential difficulty in controlling variability of the voltage shift.

(c) The estimated gain, $\hat{y}_3$, has large effects from ref (x_1), rlvs/ref (x_8), and j6/j1 (x_9), with smaller but practical effects from clvs (x_3), j1 (x_4) and from the uncontrollable factor jmodrs (x_{14}).

Based on the tentative optimization and these plots, we reduce the ranges for the controllable factors. For example, on the normalized, logarithmic scale, the "optimal" value of (c_1) is roughly at its lower bound of -0.35. Figures 6.3 and 6.4 show that the only way to get larger gain values while maintaining voltage shift near 5 is to take small values of x_1. Figure 6.2 shows that x_1 is unimportant for bandwidth. Thus, we choose a fairly tight sub-region for c_1: the interval [-0.353, -0.3]. After adding in ± 0.147, the $\pm 3\sigma$ range for u_1, which cannot be reduced, the total range for x_1 becomes [-0.50,-0.15] after some rounding. Similarly, promising sub-ranges are identified for $c_2, \ldots, c_9$.

We now repeat steps 2 - 4 on the new region. Fifty runs from a Latin hypercube design produce no poor circuits, a reflection of the move to an appropriate part of the space. The root mean squared error of prediction drops to about 0.004 log GHz, 0.035 V, and 0.01 dB for $\hat{y}_1$, $\hat{y}_2$, and $\hat{y}_3$, suggesting that we now have predictors reliable enough for optimization. However, when the "optimal" $\mathbf{c}$ is tested by a confirmatory run for the nominal circuit ($\mathbf{U} = 0$), the nominal voltage is 4.88 V rather than the predicted 4.95 V. Rather than taking more data we add first-order regression terms to the model for voltage, as in


Estimated Main Effects for log(Bandwidth (GHz))
Estimated Main Effects
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
(X1,...,X14)=(1,...,9,A,...,E)
Figure 6.2


Estimated Main Effects for Gain (dB)
Estimated Main Effects
1
0
-1
-2
-0.4
-0.2
0.0
0.2
0.4
(X1,...,X14)=(1,...,9,A,...,E)
Figure 6.3

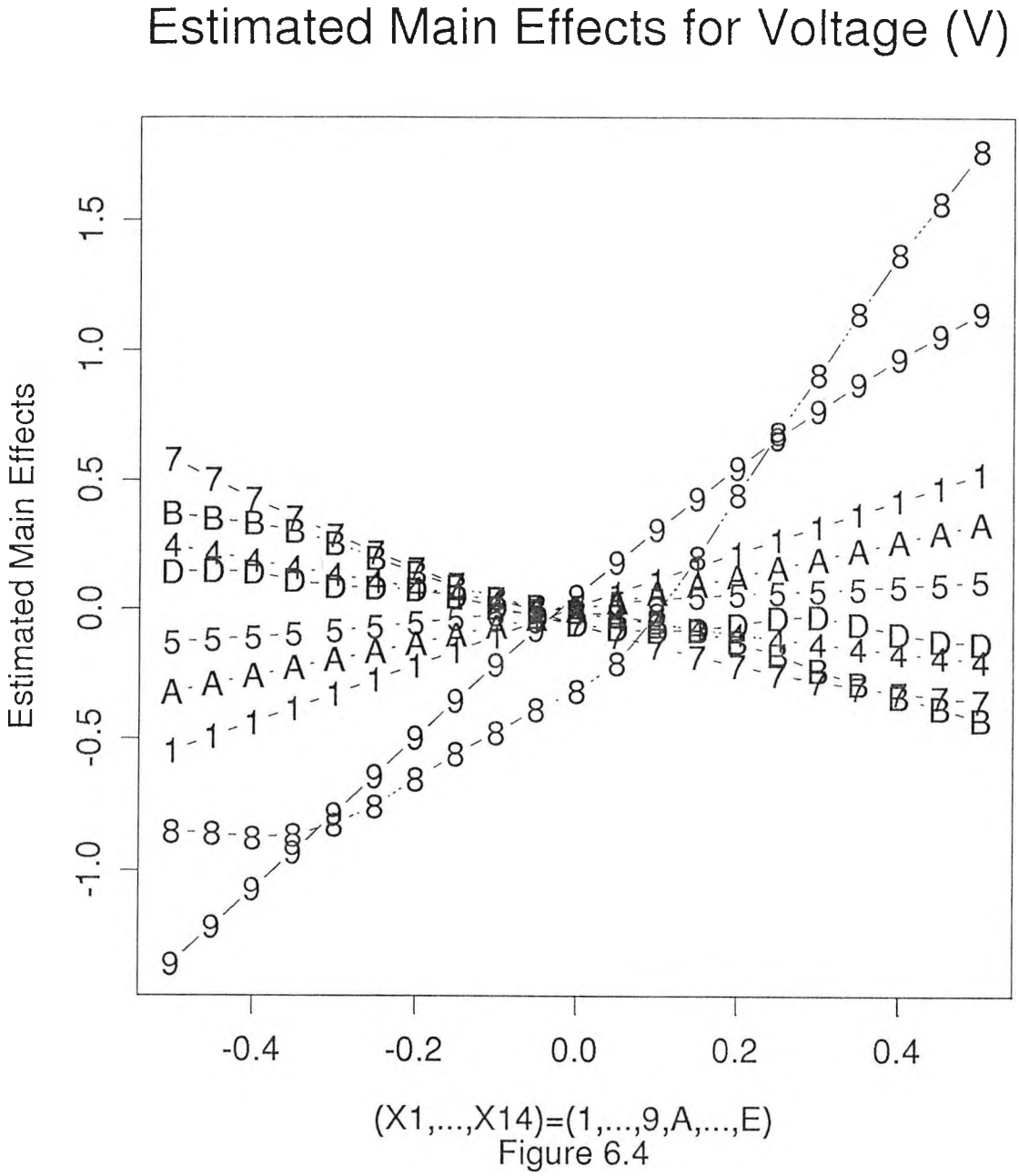

Estimated Main Effects for Voltage (V)
Estimated Main Effects
1.5
1.0
0.5
0.0
-0.5
-1.0
-0.4
-0.2
0.0
0.2
0.4
(X1,...,X14)=(1,...,9,A,...,E)

Figure 6.4

(6.2.3). This improves the accuracy of the voltage predictor enough so that the estimated optimum meets the required voltage constraints. The new "optimal" c_i's lead to accurate predictions at the nominal ($\mathbf{U} = 0$) and produce low variabilities as shown in Table 6.3. The improvement over the initial design is indicated in Table 6.3. The confirmations for $Var(j,\mathbf{c})$ required a Monte Carlo sample of 100 circuit simulation runs---we do this just for demonstration purposes. The Monte Carlo sample is obtained by picking 100 random vectors **U** according to their distribution.

	Initial	Nominal		$\sqrt{Var(j,\mathbf{c})}$	
		Predicted	True	Predicted	True
log(bandwidth (GHz))	log(3.33)	log(6.62)	log(6.41)	.044	.045
voltage (V)	4.99	4.975	4.929	.091	.111
gain (dB)	-2.786	-1.913	-1.928	.037	.038
ripple (dB)	0.000	0.000	0.000	---	---

Table 6.3 Nominal performances and variabilities at the second-stage optimal **c**: Voltage-shifter circuit.

6.4 Output Buffer Example

The example is an output buffer. Output buffers translate logic signals between integrated circuits and external connections. Of particular interest to the designer when considering components are:

1. The time between state transitions.
2. The control of voltage spikes due to changing currents.

These two performance criteria are in direct conflict, i.e. faster switching means more noise and so a trade-off is necessary.

The example is a proprietary circuit from INTEL so certain information has been left out or altered to mask the true results. The input ranges for the experimental plan are scaled to $[-0.5, 0.5]$ for analysis and this is the range in which the results will be reported. Also, the values used for targets in the constraints have been replaced with constants, K_ν, where ν is the variable associated with the constant.

There are 11 input variables used in experimentation, ten device sizes (T_i, P_j, N_j $i=1,2$ and $j=1,...,4$) and C_{load}. All the input variables were assumed for this exercise to not have uncontrollable variation, i.e. $\mathbf{U}=0$. The ranges for these input variables covered a large area to look for as many solutions as possible. No nominal settings were given as in the example in Section 6.3.

There are 16 response variables used to define the specifications for the circuit. There are two types of specifications, constraints and targets. The response variables are:

1. The delays in nanoseconds, T_{DH} and T_{DL}.
2. The four power supply noises in volts, V_{CTDH}, V_{CTDL}, V_{STDH}, and V_{STDL}.
3. The two drive strengths in volts, V_{UP} and V_{DN}.
4. The two DC Current drives in milliamps, I_{OL} and I_{OH}.
5. The two loaded output transition times in nanoseconds, T_R and T_F.
6. The four peak currents in milliamps, I_{STDL}, I_{STDH}, I_{CTDL}, and I_{CTDH}.

The goal is to find a circuit that minimizes the four power supply noises given a value of the input variable C_{load} and a set of constraints on the remaining 12 response variables. The three values of C_{load} that were investigated are -0.25, 0.0, and 0.25. The general requirements for the optimal circuit design are the following.

Given the following constraints:

1. $T_{DH} < K_{TDH}$ and $T_{DL} < K_{TDL}$.
2. $I_S = K_{IS} I_{STDL} - I_{STDH} > 0$ and $I_C = K_{IC} |I_{CTDH}| - |I_{CTDL}| > 0$.
3. $T_R < K_{TR}$ and $T_F < K_{TF}$.
4. $I_{OL} > K_{IOL}$ and $I_{OH} > K_{IOH}$.
5. $V_{UP} > K_{VUP}$ and $V_{DN} < K_{VDN}$.

Find the device sizes that minimize $V_{\max} = \max(V_{STDL}, V_{STDH}, V_{CTDH}, V_{CTDL})$.

6.4.1 Results

For this example it took two stages, each with an experimental plan of 75 runs, to produce statistical models accurate enough to locate an accurate estimate of the input factor values for the optimal circuit. We found that for $C_{load}=0.25$ there were no viable solutions. For $C_{load}=0.0$ we were able to find factor

values so that $V_{\max}=0.875V$. For $C_{load}=-0.25$ we found factor values so that $V_{\max}=0.595V$.

A question posed at the conclusion of the second stage led to a third stage experiment. The goal for this stage was to find the minimum $V_{\max}$ at $C_{load}=-0.25$ and $C_{load}=0.0$ with no constraints except for the delay constraints. For $C_{load}=0.0$ we were able to find factor values so that $V_{\max}=0.765V$. For $C_{load}=-0.25$ we found factor values so that $V_{\max}=0.500V$. The rest of this section gives a detailed account of how these solutions were located.

Stage 1

The primary goal of this stage was to reduce the size of the problem. This was accomplished by reducing the region of the input space where the search for the optimal factor values was conducted and by trying to reduce the number of response variables that needed tending. Since we were looking for results at three separate values of C_{load} there are two possible strategies for reducing the region, separate regions for each value of C_{load} or one region which is large enough to contain the solutions for all C_{load} values of interest. After analyzing the data from the first stage a single subregion was used to search for solutions for all values of C_{load}.

There are three ways in which the number of response variables can be reduced:

1. Several response variables can be combined into one function.
2. They are superfluous for the circuit specifications given.
3. They can be made superfluous by restricting the search area to a region where the response always meets the relevant constraint.

From viewing the scatter plots of input vs. response it was apparent that there were many observations, particularly on the edges of the input space, that had response values far from the performance specifications. This led to the use of three "sub" designs. The subdesign for T_R, T_F, T_{DL}, and T_{DH} contained 63 points. The subdesign for the response variables V_{STDL}, V_{STDH}, V_{CTDL}, and V_{CTDH} contained 67 points. The third subdesign used 59 points and was used to model the two constraints I_C and I_S.

It was also apparent that four of the response variables: V_{UP}, V_{DN}, I_{OL}, and I_{OH} are dependent on a single input variable, either T_1 or T_2, so there was no

need to model these responses. See Figure 6.5a-d. From the graphs of the experimental data it is clear that to meet the constraints on V_{UP}, V_{DN}, I_{OL}, and I_{OH} that T_1 and T_2 have to be greater than -0.25. Because of the clear relationship between these responses and the input factors the number of response variables which need to be considered can be reduced to twelve.

There were two opportunities to combine response variables into a single function. We started by modelling I_S and I_C and their corresponding constraints. By generating predictors for I_S and I_C as well as for the individual responses we found that the cross-validation ERMSE (3.4.1) for I_S and I_C were only slightly larger than the cross-validation ERMSE for I_{STDL} and I_{CTDH} respectively, while the CV RMSE for I_{STDH} and I_{CTDL} were an order of magnitude larger. From these results a decision was made to use I_S and I_C rather than the individual responses. We also considered a similar approach for $V_{\max}$; this is a much more complex function and did not give as accurate a predictor as $max(\hat{V}_{STDL}, \hat{V}_{STDH}, \hat{V}_{CTDL}, \hat{V}_{CTDH})$ so was abandoned. By considering these options we were able to reduce the number of "response variables" from 12 to 10.

The remaining 10 responses are modeled as (6.3.7); CV RMSE estimates and main effects plots were produced as well. The CV RMSE estimates showed that the statistical models do not predict the response variables accurately enough for making precise statements about an optimal solution. The main effects plots showed that most variables depend on C_{load} and that response variables could be divided into roughly two groups, those that depend on T_1, P_1 and N_3 and those that depend on T_2, N_2 and P_4. See Table 6.4 for a list of significant input variables for all responses and the CV ERMSE for the respective response variables. Some of the more important interactions are :

1. T_1, P_3 for I_S.
2. T_2, N_2 and T_2, C_{load} for T_{DH}.
3. T_2, N_3 for T_{DL}.
4. P_1, P_2 for V_{CTDL}.
5. N_2, C_{load} for V_{STDH}.

Since the predictors were not sufficiently accurate, the goal at this time is not to find a precise value of an optimal solution, but to narrow our search to a

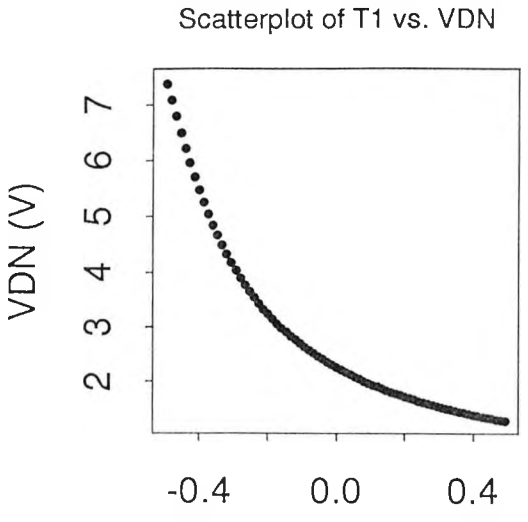


Figure 6.5(a)

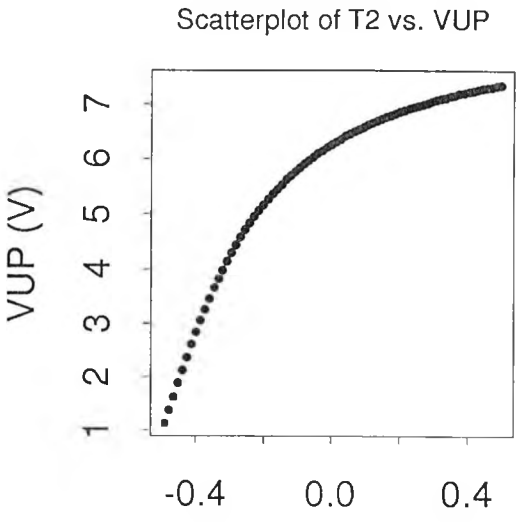


Figure 6.5(b)

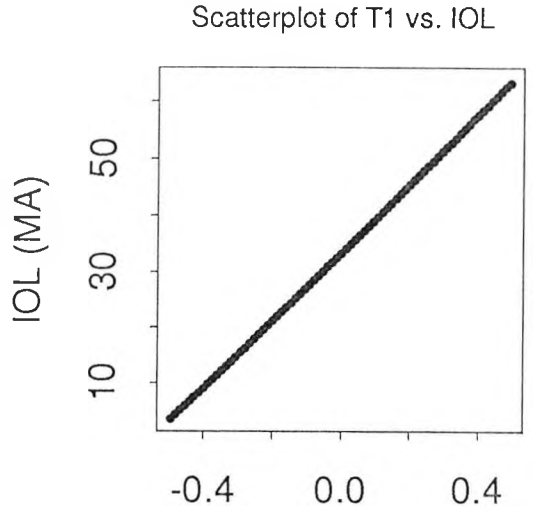


Figure 6.5(c)

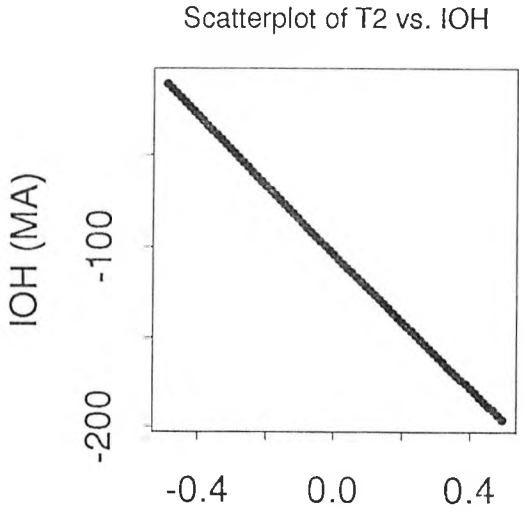


Figure 6.5(d)

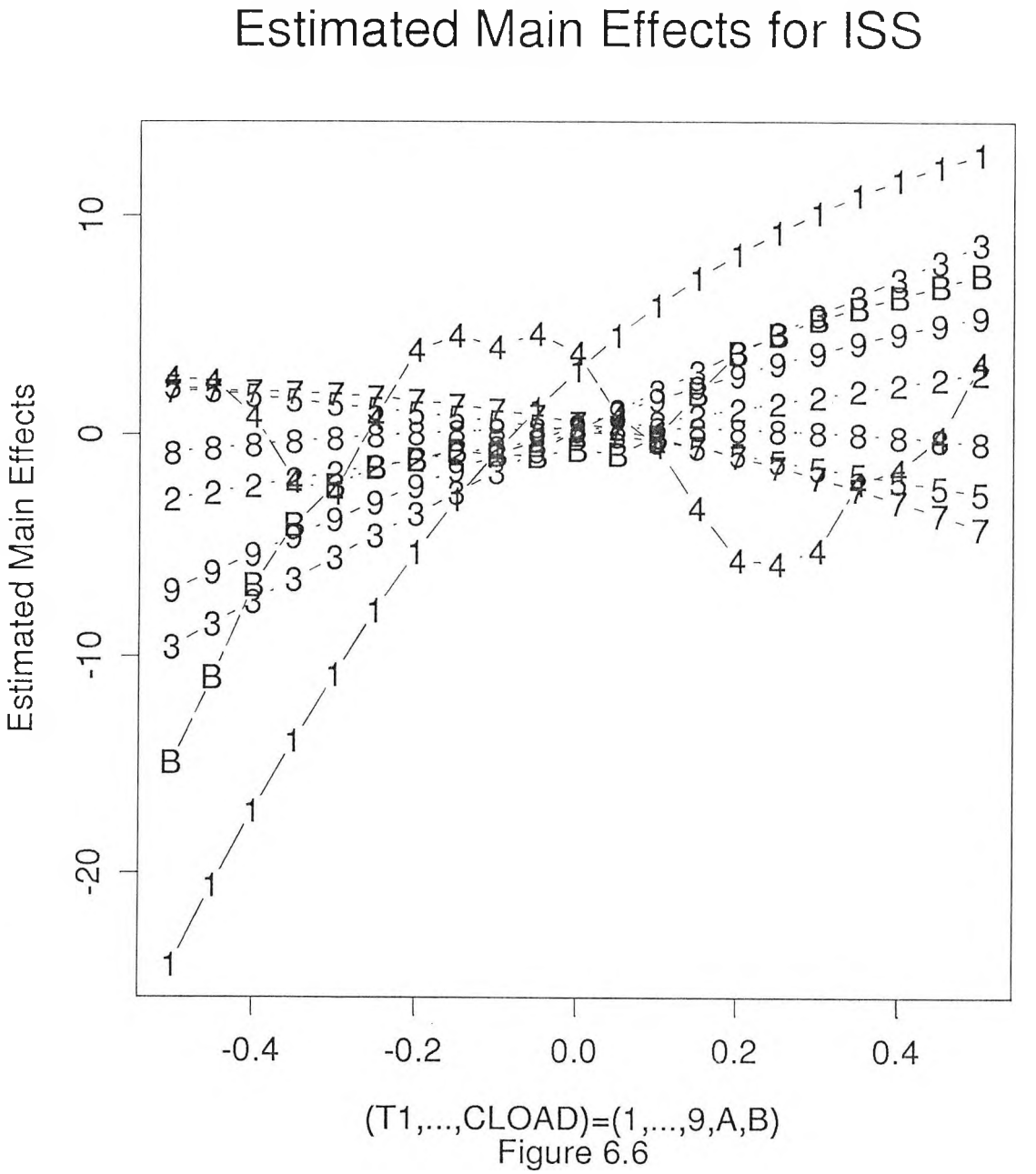
smaller region of the original input space. The next step (Step 5) was to use the statistical models of the response variables to determine the subregion of the input space where the optimal configuration of factor values was located.

Response Variable	Influential Input Variables	CV ERMSE	Data Range
T_F	T_1, C_{load}	0.82	20.1
T_R	T_2, C_{load}	0.80	22.3
I_S	$T_1, C_{load}, P_1, P_4, N_1, P_3$	4.86	82.75
I_C	$N_3, P_1, N_2, T_1, C_{load}, P_4, N_3, P_2$	13.8	133.75
T_{DH}	$T_2, C_{load}, P_4, N_2, P_2, P_3, P_1$	2.14	27.4
T_{DL}	$C_{load}, T_2, T_1, P_1, N_3, N_1$	1.27	13.9
V_{CTDH}	$N_2, T_2, P_4, C_{load}, N_4, P_2, T_1$	0.08	1.07
V_{CTDL}	$P_2, P_1, N_4, T_2, N_1, N_2, P_4$	0.10	1.07
V_{STDH}	$T_1, N_1, P_3, C_{load}, N_3, N_2, N_4, P_2$	0.10	0.82
V_{STDL}	$P_1, T_1, N_4, T_2, N_1, C_{load}, N_3$	0.13	1.48

Table 6.4 Influential Variables in Stage 1 in order of importance.

First, response variables that are only affected by two inputs were studied. The main effects plots, see Figs. 6.6 and 6.7, show that the I_S constraint is met when T_1 and $T_2 > -0.25$, so can be dropped along with V_{UP}, V_{DN}, I_{OL}, and I_{OH}. The models for T_R and T_F show that T_1 or T_2 and C_{load} are the only influential variables. Figure 6.8 shows that the equation $aT_1 + b*C_{load} < K_{TF}$, gives an accurate picture of the relationship between T_1 and T_F. The variables T_2 and T_R have a similar relationship as is shown in Figure 6.9. These functional estimates show that when $C_{load} < 0.0$ the constraints for T_R and T_F are met if T_1 and T_2 are > -0.25 and when $C_{load} > 0.0$ the constraints are met if T_1 and $T_2 \geq 0.0$. By dealing with just T_1 and T_2 we have reduced the number of response variables to seven.

The problem has now been reduced to finding the region(s) in the factor space that meet the constraints on T_{DH}, T_{DL}, and I_C and minimizes $V_{\max}$. The main effects plots, Figs. 6.10-6.16, indicate that the inputs and responses could


Estimated Main Effects for ISS
Estimated Main Effects
10
0
-10
-20
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.6

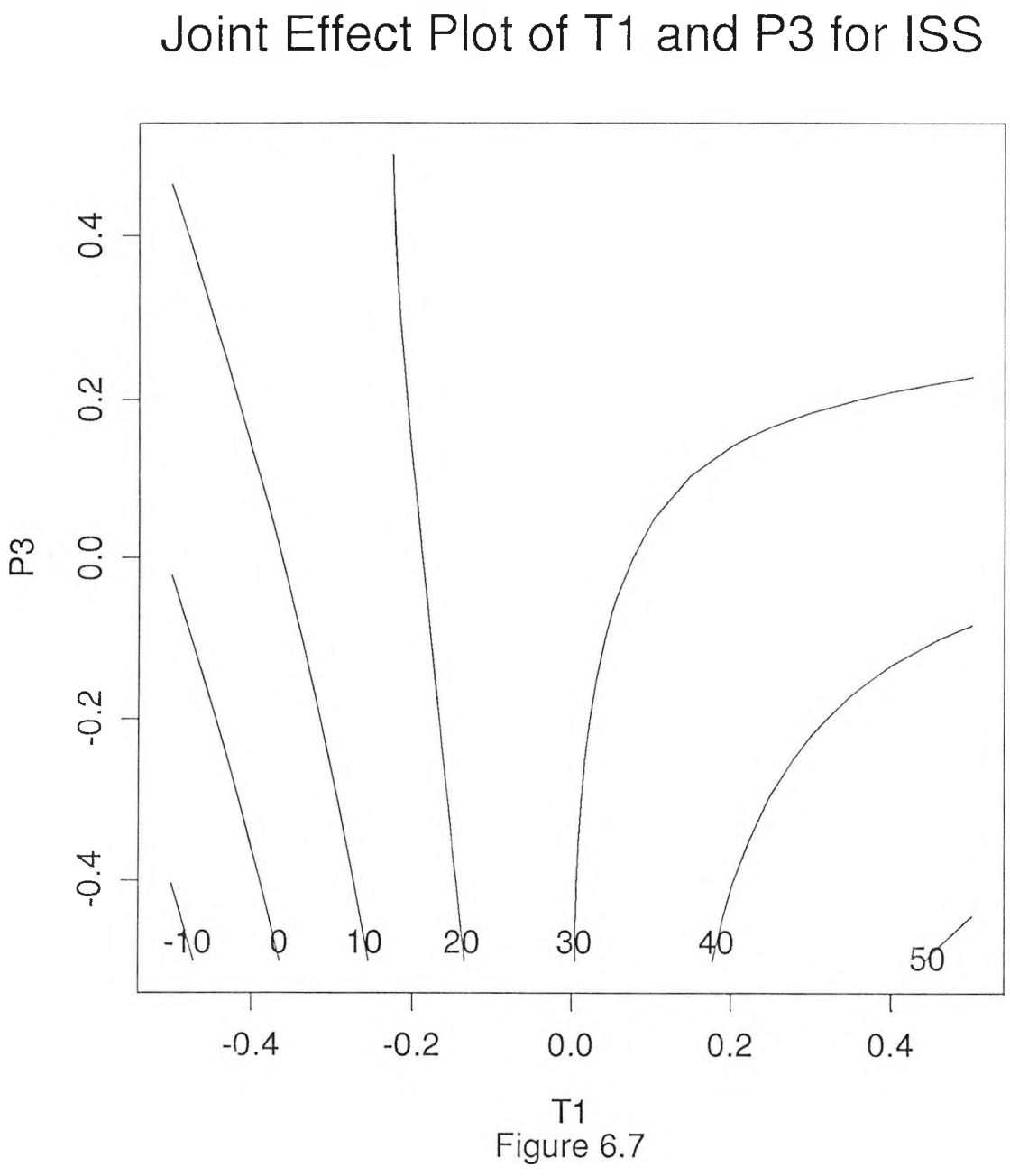
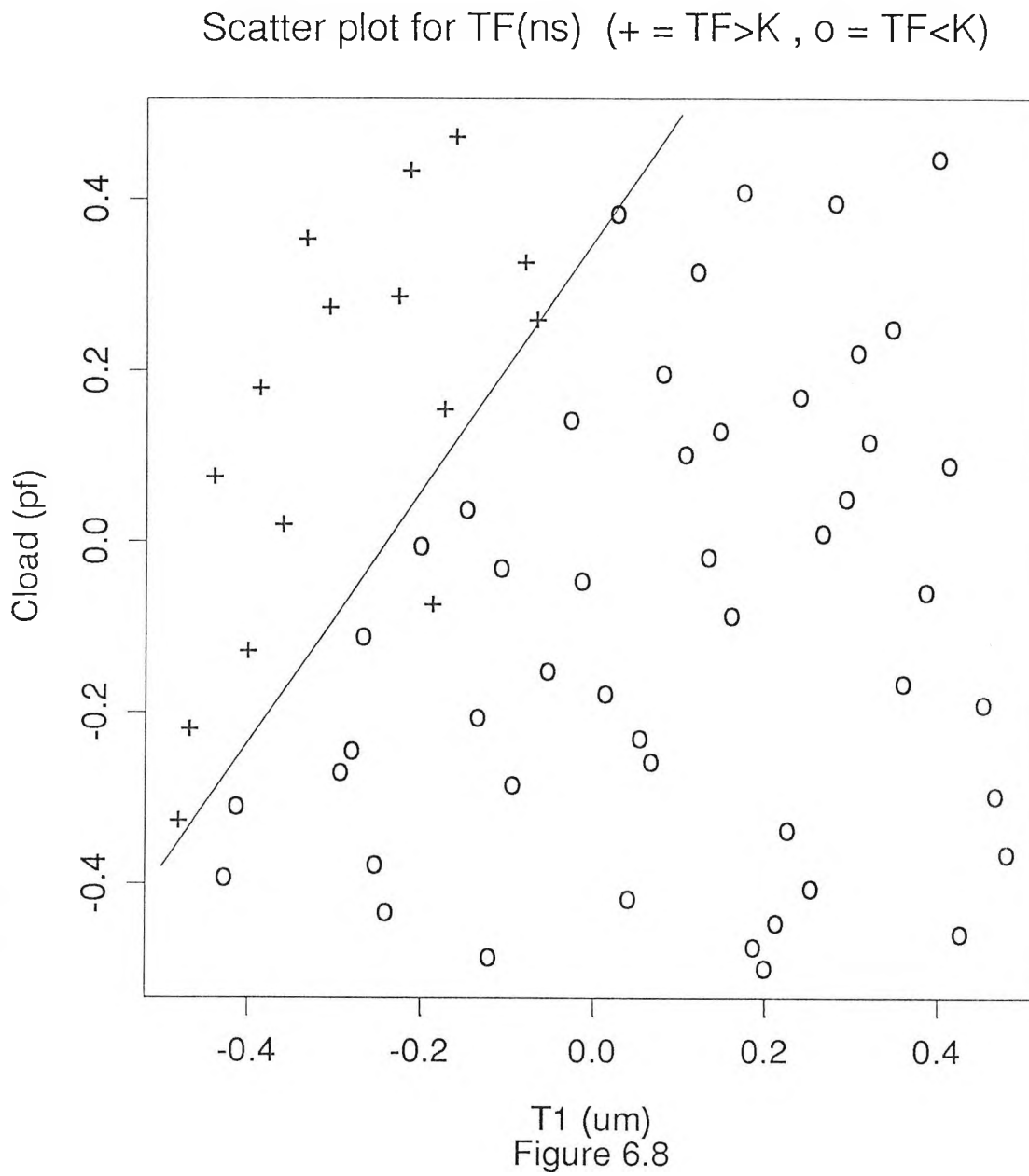
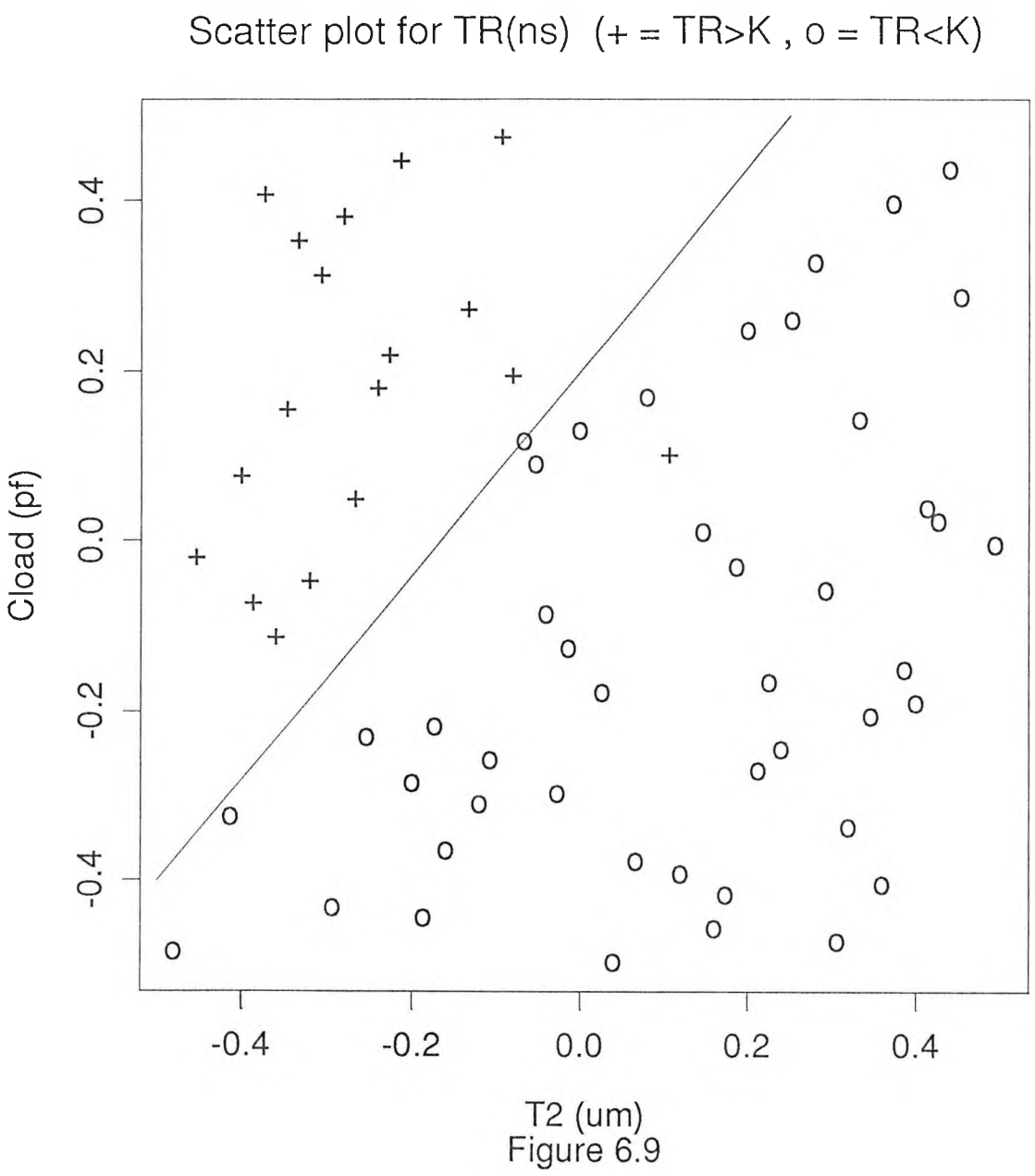
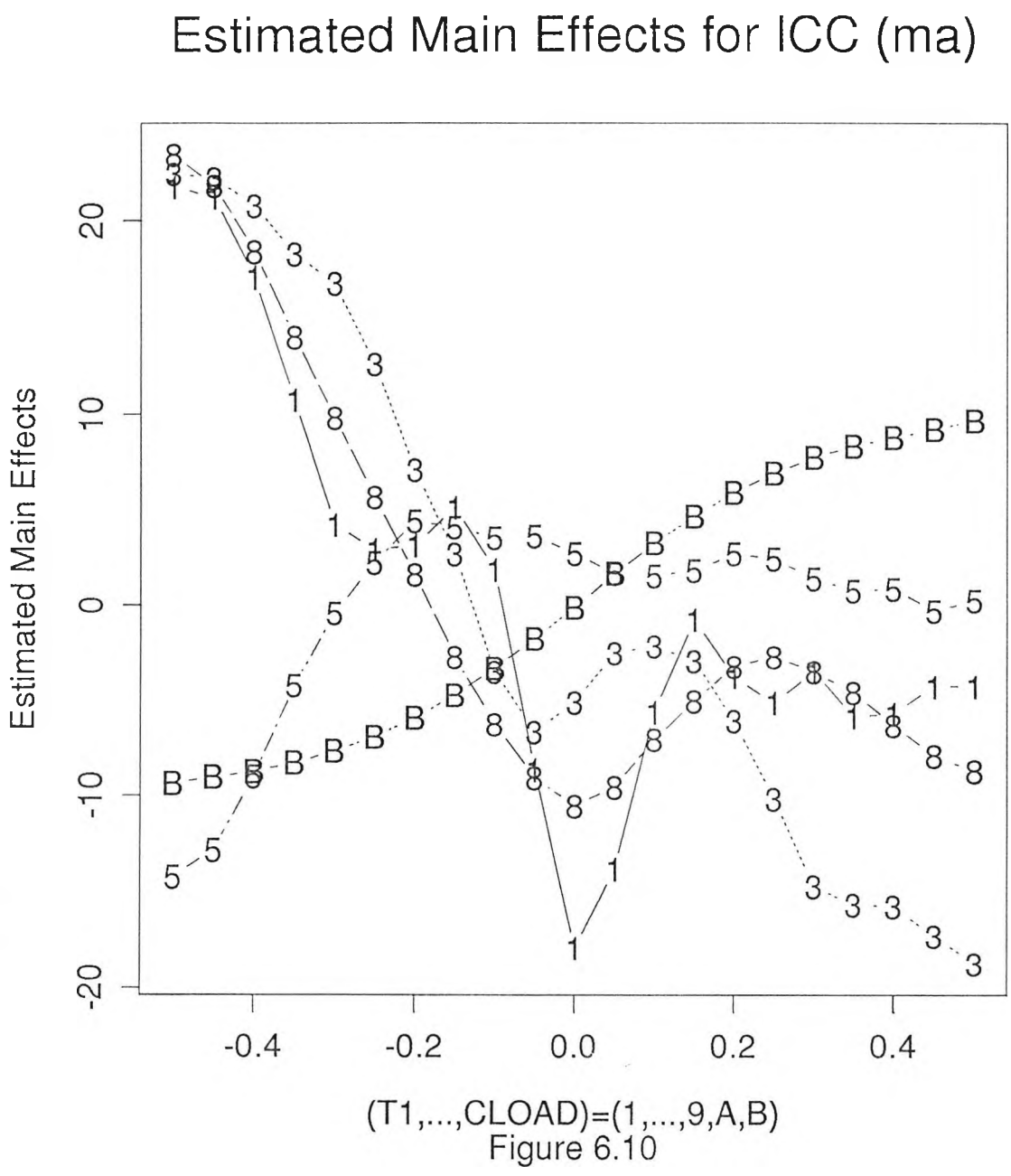

Joint Effect Plot of T1 and P3 for ISS
P3
0.4
0.2
0.0
-0.2
-0.4
-10
0
10
20
30
40
50
-0.4
-0.2
0.0
0.2
0.4
T1

Figure 6.7


Scatter plot for TF(ns) (+ = TF>K , o = TF<K)
Cload (pf)
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
T1 (um)
Figure 6.8


Scatter plot for TR(ns) (+ = TR>K , o = TR<K)
Cload (pf)
0.4
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
T2 (um)
Figure 6.9


Estimated Main Effects for ICC (ma)
Estimated Main Effects
20
10
0
-10
-20
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.10

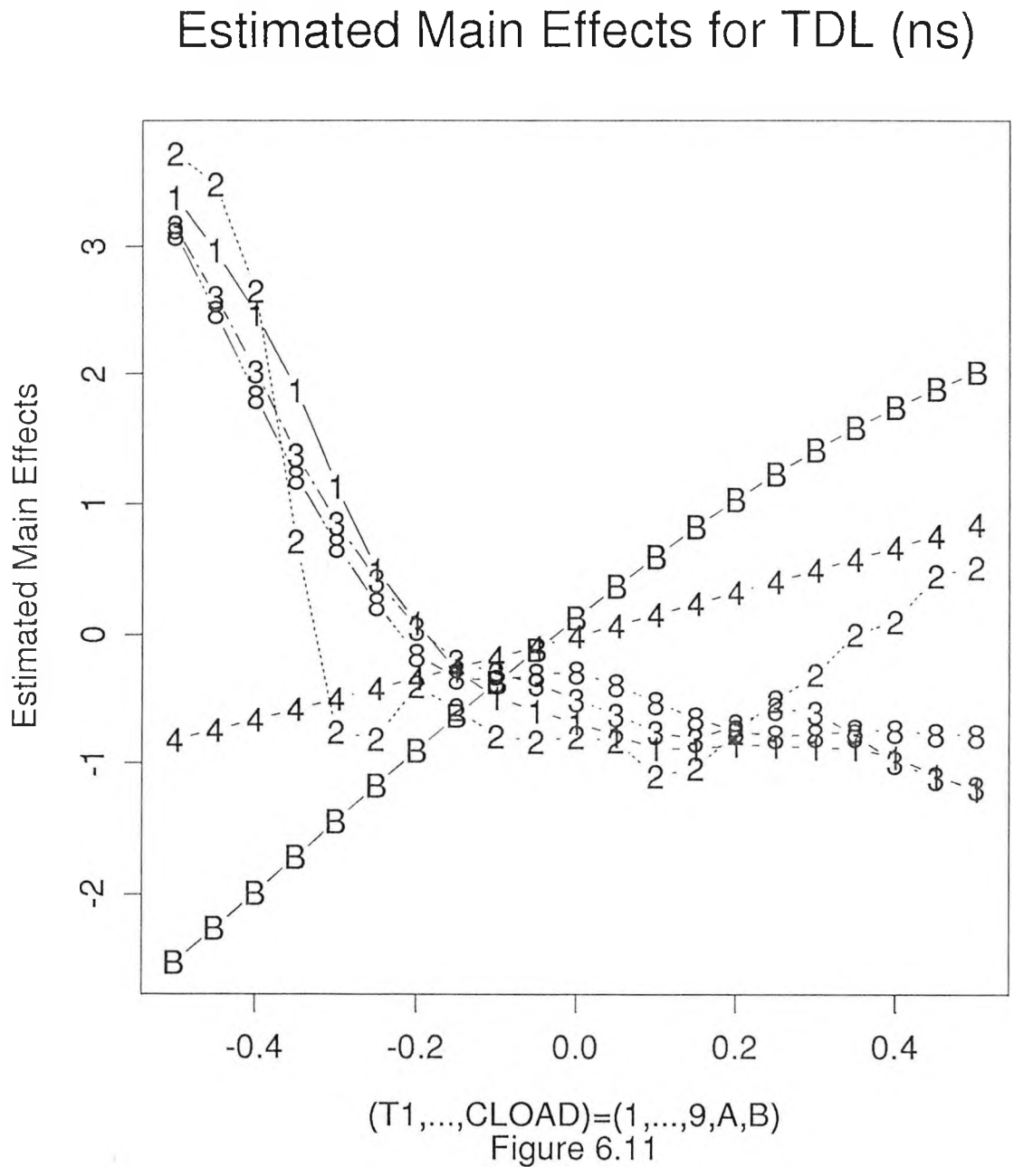
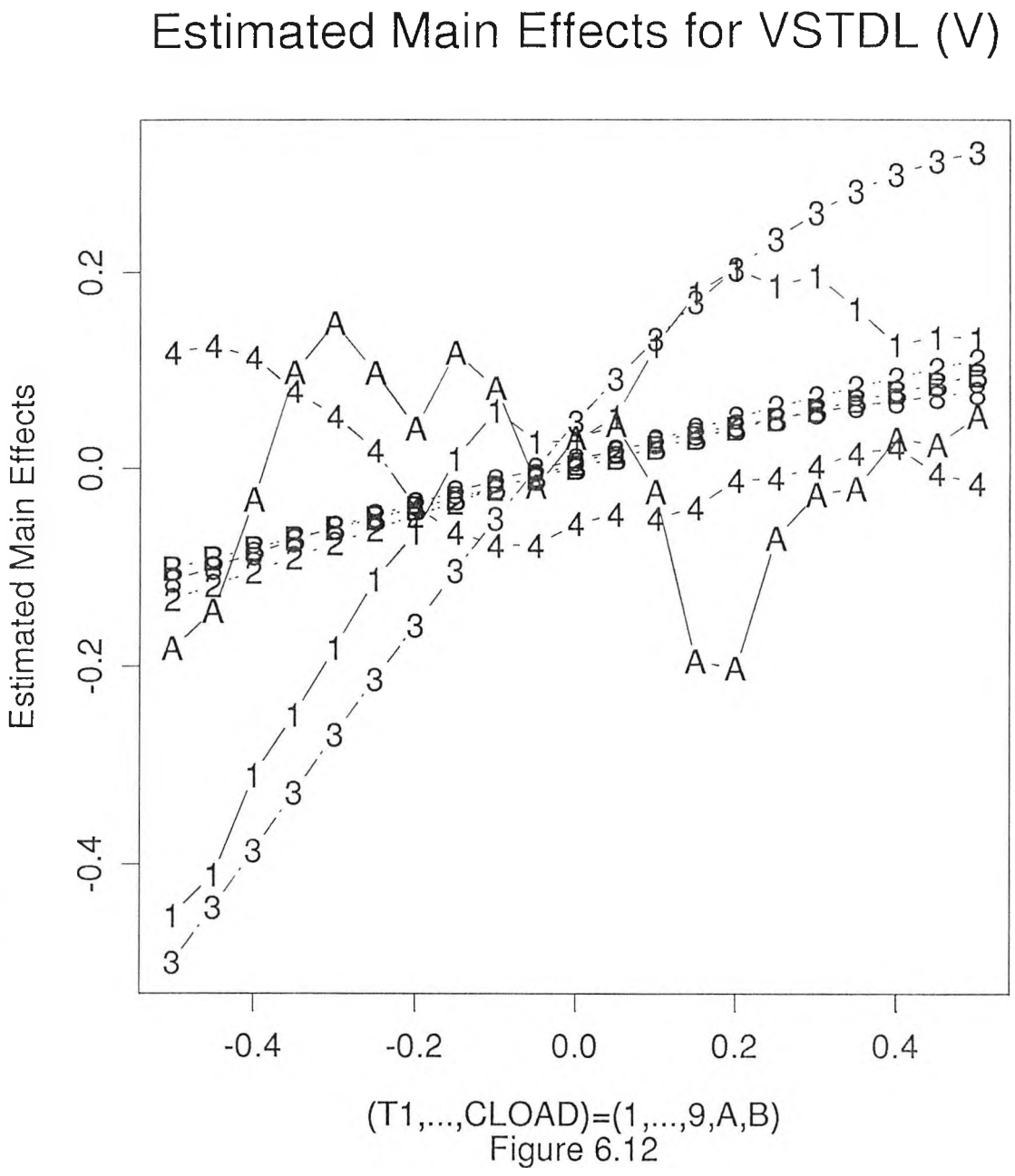
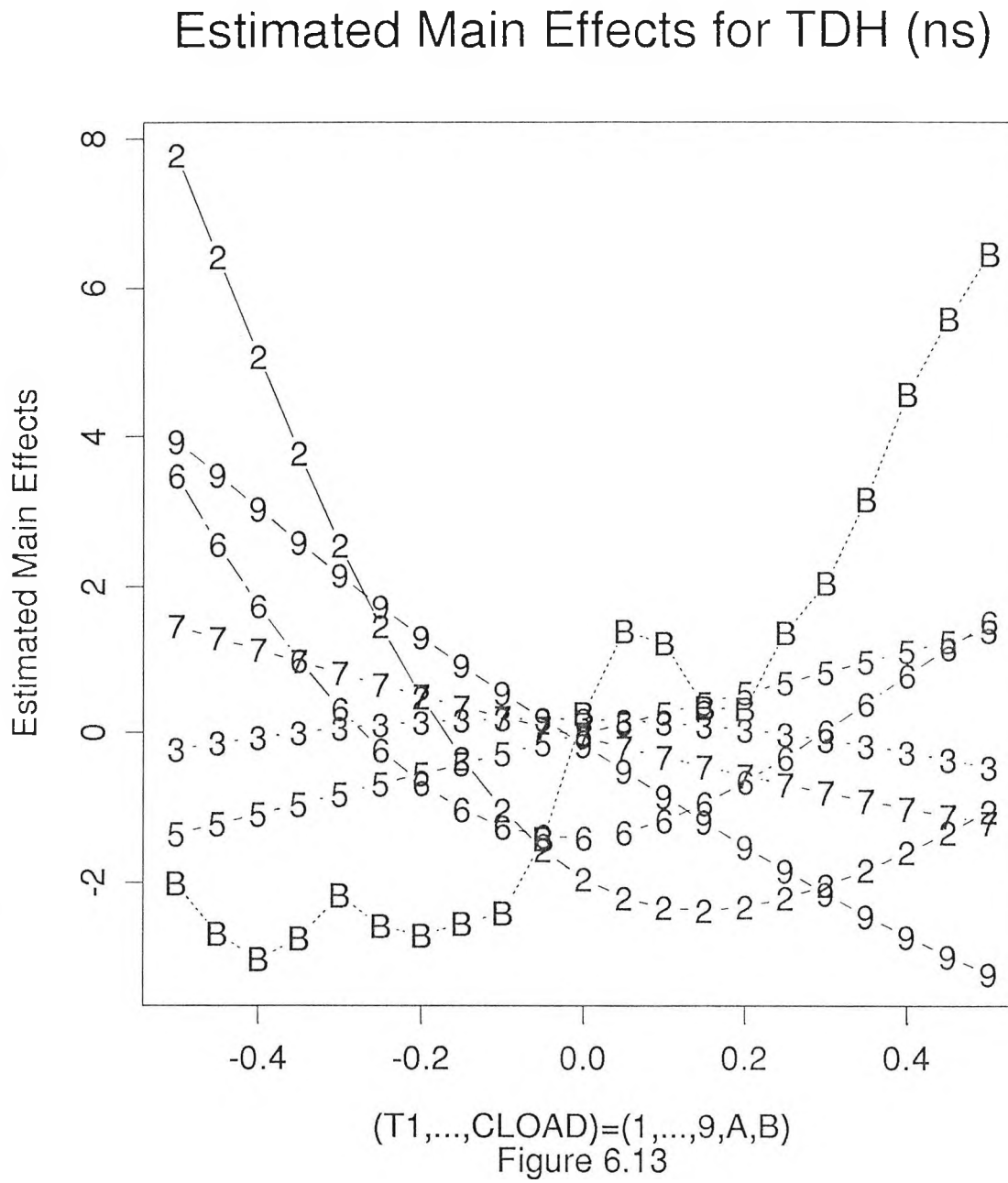

Estimated Main Effects for TDL (ns)
Estimated Main Effects
3
2
1
0
-1
-2
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)

Figure 6.11


Estimated Main Effects for VSTDL (V)
Estimated Main Effects
0.2
0.0
-0.2
-0.4
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.12


Estimated Main Effects for TDH (ns)
Estimated Main Effects
8
6
4
2
0
-2
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.13

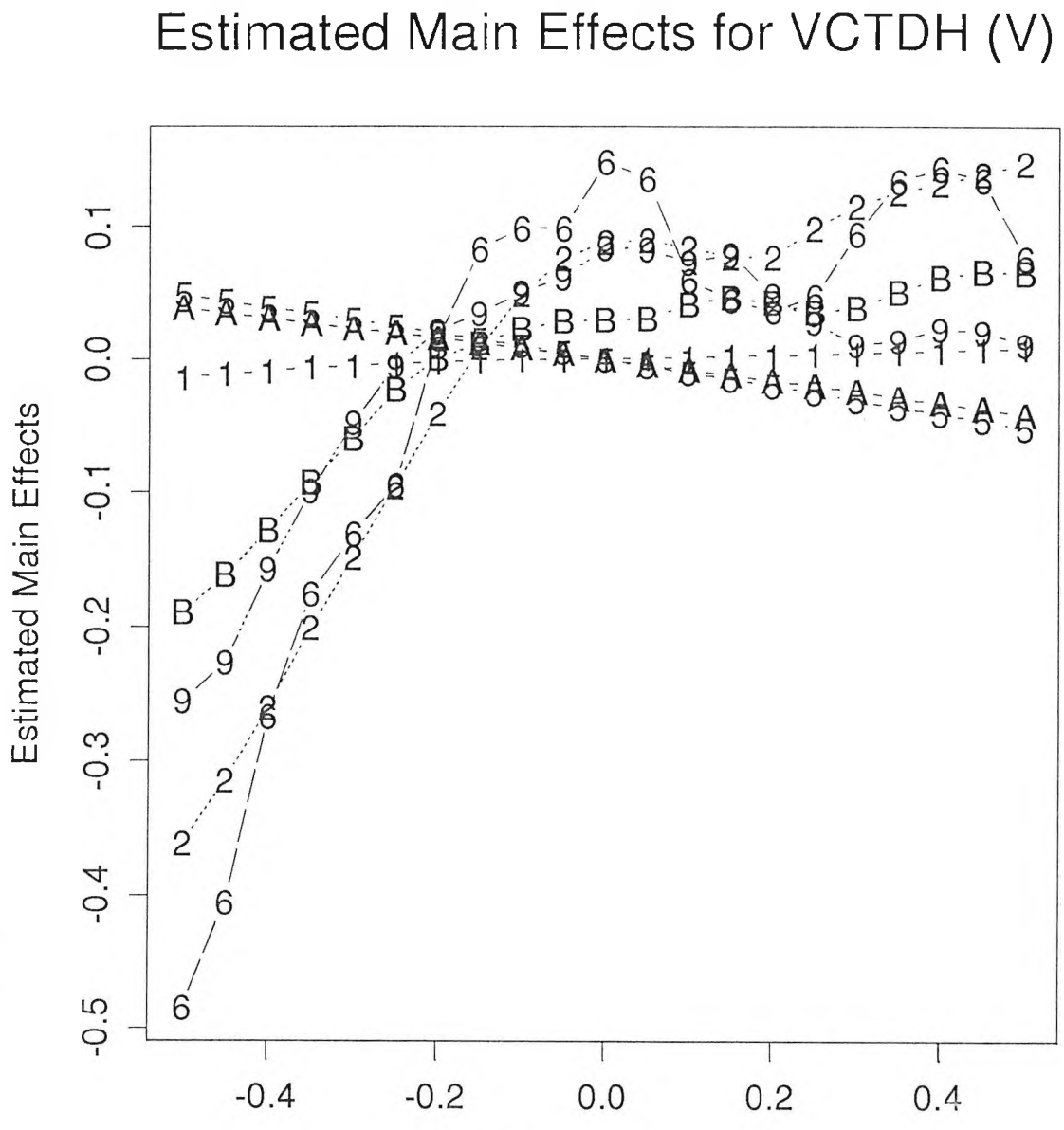
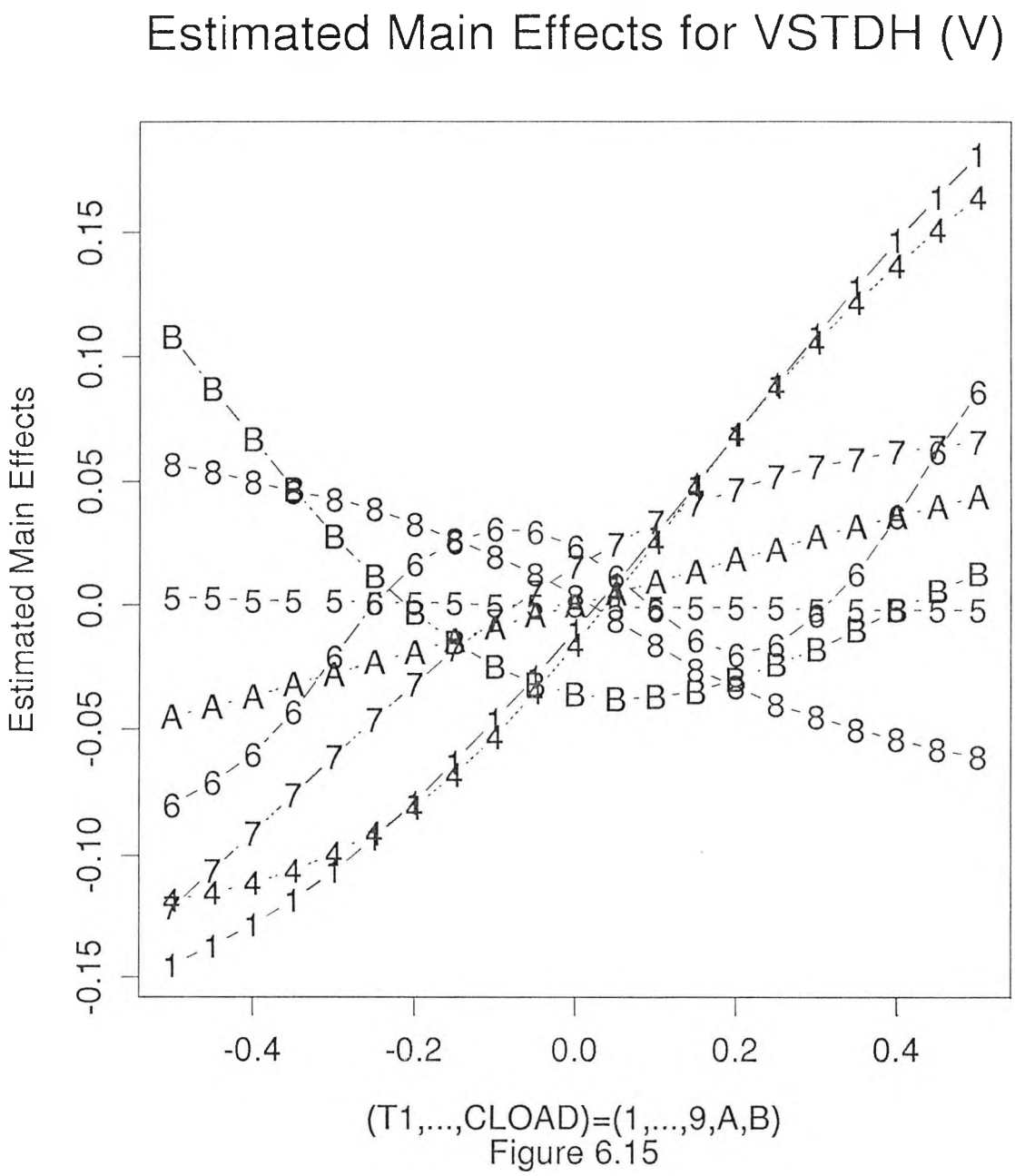
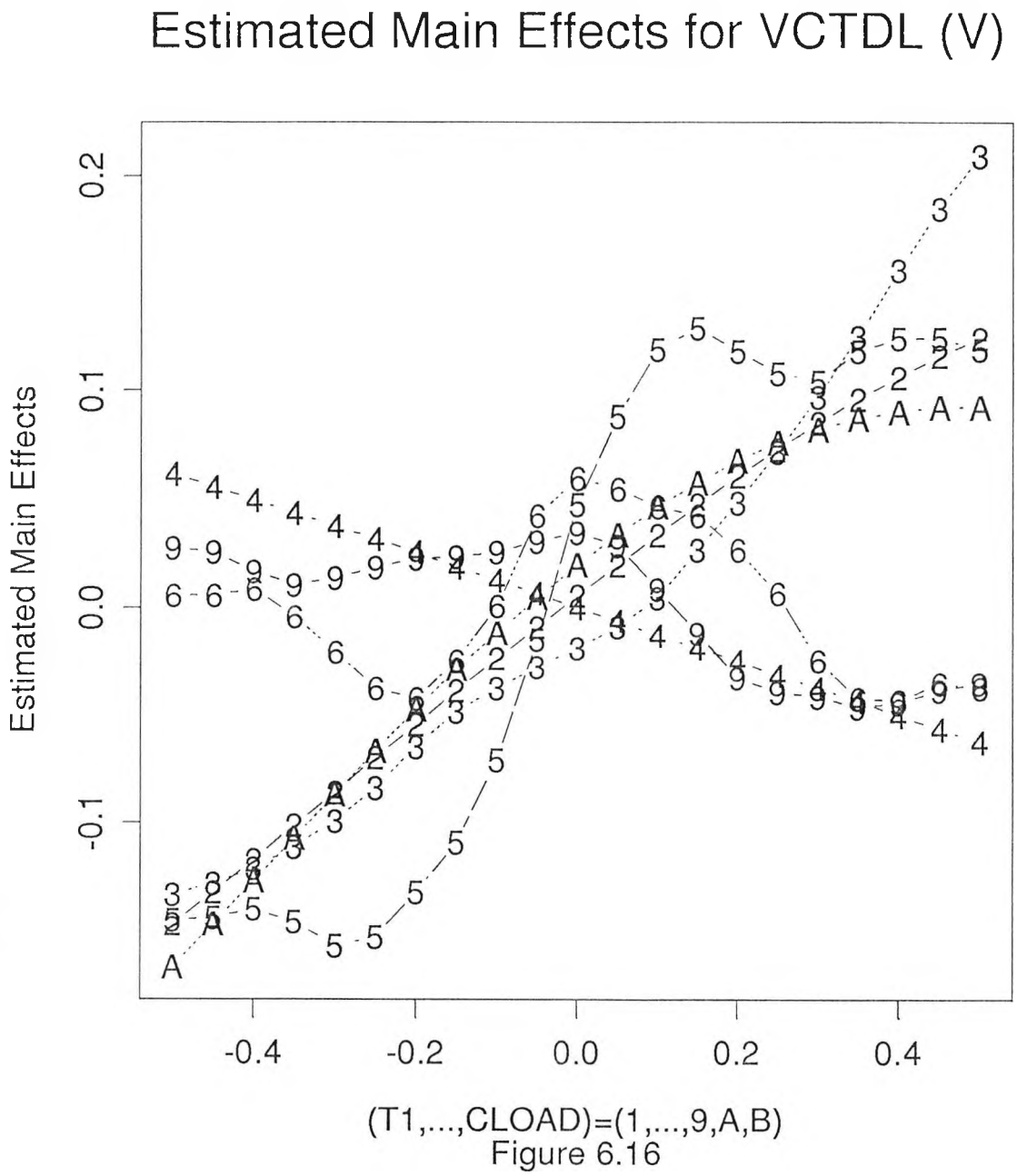

Estimated Main Effects for VCTDH (V)
Estimated Main Effects
0.1
0.0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)

Figure 6.14


Estimated Main Effects for VSTDH (V)
Estimated Main Effects
-0.15
-0.10
-0.05
0.0
0.05
0.10
0.15
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.15


Estimated Main Effects for VCTDL (V)
Estimated Main Effects
0.2
0.1
0.0
-0.1
-0.4
-0.2
0.0
0.2
0.4
(T1,...,CLOAD)=(1,...,9,A,B)
Figure 6.16

be roughly divided into two groups. T_1 and P_1 influence T_{DL}, I_C, and V_{STDL}, while T_{DL} and I_C are also strongly influenced by N_3. The response variables T_{DH} and V_{CTDH} are affected by T_2, N_2, and P_4. V_{STDH} and V_{CTDL} appear to be affected by many variables making them easier to adjust and are typically the smallest of the four noises over the region, allowing us to focus our attention elsewhere initially.

The question now is, how can the noise variables be minimized and still meet the delay and I_C constraints? First, the main effects plots show the region of T_2 that does the most to reduce V_{CTDH} is out of bounds due to other constraints. This leads us to consider keeping T_2 to the right and N_2 and P_4 as far left as the T_{DH} constraint allows. This also works well for the T_R constraint, because we now do not need to worry about $C_{load} > 0.0$. The main effects plots show that P_2 only influences I_C and V_{CTDL}. If we let P_2 be slightly negative we maximize the positive effect on I_C while reducing V_{CTDL}.

Input Variable	Stage1 Range	Stage2 Range	Stage3 Range
T_1	[-0.5,0.5]	[0.0,0.3]	[-0.4,0.1]
T_2	[-0.5,0.5]	[-0.1,0.3]	[-0.1,0.3]
P_1	[-0.5,0.5]	[-0.4,-0.1]	[-0.4,0.0]
N_1	[-0.5,0.5]	[0.0,0.25]	[-0.3,0.1]
P_2	[-0.5,0.5]	[-0.2,-0.1]	[-0.4,-0.1]
N_2	[-0.5,0.5]	[-0.5,0.0]	[-0.5,0.0]
P_3	[-0.5,0.5]	[0.0,0.2]	[0.0,0.2]
N_3	[-0.5,0.5]	[-0.4,0.0]	[0.0,0.4]
P_4	[-0.5,0.5]	[-0.4,0.25]	[-0.4,0.25]
N_4	[-0.5,0.5]	[0.25,0.5]	[0.25,0.5]
C_{load}	[-0.5,0.5]	[-0.4,0.3]	[-0.4,0.3]

Table 6.5 Regions of Inputs Modeled at 4 Different Stages

We then move on to dealing with T_{DL}, V_{STDL}, and I_C. The main components of these three responses are T_1, P_1 and N_3. For all three responses the

affect of these three inputs are nearly identical. For example, decreasing T_1, P_1 or N_3 from .1 to -.2 increases the value for T_{DL} by the same amount. We are now in somewhat of a quandary, decreasing T_1, P_1 or N_3 increases I_C and decreases V_{STDL} which is the goal. However, decreasing these inputs increases T_{DL} which we do not want. The problem is to find a favorable tradeoff for these three variables. After selecting the ranges for these variables we choose the remaining input ranges to minimize V_{STDH} and V_{CTDL}. The subregion used in the next stage is in Table 6.5.

Stage 2

The data from the second stage experiment show that the region selected meets the constraints for T_R, T_F, and I_S as expected. We also find that the constraint for I_C is fulfilled over the full region and that V_{CTDL} is always less than one of the other noise variables. This reduces the number of response variables that need to be modelled to five. We compute the parameter estimates for our statistical models of the responses. After looking at the CV ERMSE (Table 6.6) we decide our models are accurate enough to make a serious attempt at finding an optimal circuit design.

Response Variable	Influential Input Variables	CV ERMSE
T_{DL}	C_{load}, P_1, T_1, P_3, P_2	0.05
T_{DH}	T_2, C_{load}, N_1, T_1, N_2	0.11
V_{CTDH}	N_2, T_2, C_{load}, P_3, N_1, N_4,N_3, T_1	0.05
V_{STDH}	N_1, N_2, C_{load}, P_2, N_4, T_1, T_2, P_3, P_1	0.09
V_{STDL}	P_1, N_1, P_2, C_{load}, T_2, P_4, N_4, N_3	0.07

Table 6.6 Influential Variables in Stage 2 in order of importance.

The number of constraint variables that still may be an issue has been reduced to the two delays. The objective function therefore is now:

1. Given: $T_{DL}<K_{TDL}$ and $T_{DH}<K_{TDH}$.
2. Find V_{max}.

Since V_{CTDL} is always less than the other peak noises we feel it can be eliminated from the computation of $\hat{V}_{\max}$, to generate some savings in the cost of running the optimizer. Since the values of the delays that are used in the optimization are only estimates of the true values it is prudent to run the optimization for constraints $K_{TDL} \pm \varepsilon$ and $K_{TDH} \pm \varepsilon$, where ε can be taken to be approximately the $RMSE(T_{DH})$ and $RMSE(T_{DL})$ respectively. This helps to ensure that the delay constraints are met when the true values are actually computed. Six inputs generated using the optimization routine and different constraint values, three for $C_{load} = -0.25$ and three for $C_{load} = 0.0$, were run through the simulation model for confirmation. The results from the confirmation points give circuits that meet all the constraints and $V_{\max} = 0.585$ for $C_{load} = -0.25$ and $V_{\max} = 0.875$ for $C_{load} = 0.0$. We also were able to determine that there was no circuit that met all the constraints for $C_{load} = 0.25$.

Stage 3

After Stage 2 a new question was posed, if all the constraints except the delays were eliminated how would the factor values for the optimal circuit from Stage 2 change? One of the reasons that the region for Stage 2 was chosen, given the similarity of the behavior of T_1, P_1, and N_3, was the constraints on the response variables forced us to stay away from large negative values of T_1. This led us to switch the range for T_1 and N_3 for the experimental region of this stage. Since T_1 could be set further to the left than the constraints previously allowed we could expect lower noise values at least for V_{STDL}. However T_2 increased T_{DH} much more than N_2 or P_4 so we could not try a similar swap with these variables. Also, from Stage 1 it appeared that when T_2 is very small it caused T_{DL} to increase as well. This additional source of increasing T_{DL} could not be countered effectively by other input variables. So only V_{STDL} is influenced by eliminating the constraints.

Since this region had also been under consideration for the search for an optimal circuit whether constraints were included or not, we ran the optimizer under both conditions. When searching for the optimal circuit with constraints the objective function included many of the constraints that had been eliminated by the choice of the region in Stage 2. The constraints and targets for this stage include: T_R, T_F, I_C, I_S as well as the delays and the three noises V_{STDL}, V_{STDH}, V_{CTDH}; again V_{CTDL} is not a factor in determining the maximum noise level.

Response Variable	Influential Input Variables	CV ERMSE
T_F	T_1, C_{load}	0.07
T_R	T_2, C_{load}, N_2	0.02
I_S	$T_1, P_1, C_{load}, N_1, P_2$	1.39
I_C	$T_1, P_2, T_2, P_1, C_{load}, N_2$	2.78
T_{DL}	T_1, C_{load}, P_1	0.09
T_{DH}	$P_4, C_{load}, N_2, T_2, P_2$	0.15
V_{CTDH}	$N_2, P_4, T_2, C_{load}, T_1$	0.02
V_{STDH}	$N_1, T_1, T_2, P_1, P_4, P_3, N_3, C_{load}$	0.05
V_{STDL}	$P_1, C_{load}, N_3, T_2, T_1$	0.06

Table 6.7 Influential Variables in Stage 3 in order of importance.

The results for this stage are summarized in Table 6.7. There is little difference in the estimates of the optimization with or without constraints. Upon confirmation we find that given $C_{load} = -0.25$ the circuit with constraints does slightly better, 0.5V, than does the circuit found in Stage 2. The circuit found for the optimization without constraints was not predicted well and did not give as good results as the circuit found with constraints. However, by looking at the main effects plots it is apparent that we could have expected only slight improvements since we were not able to influence the other noise variables by the change in the location of the search.

The circuit had previously been "optimized" using standard industry methods. Our solutions for $C_{load} = -0.25$ and $C_{load} = 0.0$ were as good or slightly better than those previously discovered. The strategy used by industry to determine their solutions required 3000 simulator runs compared to the 225 runs used by the sequential strategy in this example.

6.5 Discussion

6.5.1 Polynomial Models

As noted in Section 6.2, various methods of approximating the performance functions could be substituted in a modular way. To illustrate some of the disadvantages of using low order polynomials, a fairly standard choice for modeling,

we repeat parts of the voltage-shifter example in Section 6.3. At the first stage, a full quadratic model in 14 dimensions would require 120 points, considerably more than is necessary for our predictor. To avoid a comparison based on a different design and different data, a regression model is fit using the same data by selecting terms from a full quadratic model. For each performance function our selection process finds the same active variables as we found in Steps 3 and 4 and a quadratic model is fit to these variables.

The prediction errors are 1.5 to 3 times those from model (6.2.2). Moreover, the constrained optimization (6.4.2) leads to a different part of the space when using these less accurate quadratic regressions. A confirmation run at the regression optimum has large ripple and poor bandwidth-regression leads to a wrong part of the space.

At the second stage, regression models (fitted from the 50-point data set in the "correct" subregion) fare somewhat better. The prediction errors are now from 1.3 to 2 times those from model (6.2.2), and the optimization leads to essentially the same optimal point as model (6.2.2). This is not surprising. When factor ranges narrow low order polynomials become more competitive predictors.

Increasing the amount of data will not in itself guarantee substantially improved regression models. Reduction of systematic error from these models may also require including more model terms (e.g., third-order terms). As the models become larger, though, they will become computationally more demanding because of the need to do model selection.

6.5.2 Taguchi Approach

A traditional approach to resolving multiple performances is to combine them into a single objective function.[1] Yield is a common choice.[2 4] In contrast to yield, Taguchi[5] emphasizes reduction of variability in the performances of interest to minimize measures of economic loss. The resulting choice of designable factors is intended to be robust, that is, to make the circuit insensitive to manufacturing and other uncontrollable variabilities. Although yield is a convenient single criterion, one concern leading to Taguchi's view is the fact that the performance constraints defining yield are often arbitrary and create impractical differences between circuits that just pass and those that just fail.

In the voltage-shifter example we combined performances and criteria into a single objective and also followed Taguchi's route by incorporating reduction of variability as one of the criteria. However, by modeling performances, the user can substitute these models into tailor-made objective functions at the optimization stage. If desired, these models could also be used to predict, and hence optimize, yield.

Though our approach has the same overall objectives as the Taguchi approach, there are a number of differences between the strategy in Steps 1 to 6 of Section 6.2 and the *methods* advanced by Taguchi. Taguchi seeks to model criteria or losses. As noted,[3] this can not only lead to inefficient use of the data, but also to a poor engineering design. Furthermore, the Taguchi methods have difficulty reconciling multiple criteria in the optimization. An analysis of the voltage-shifter example based on Taguchi's signal-to-noise ratios, using an experimental plan of 729 simulator runs, had to be abandoned because of difficulty in reconciling the signal-to-noise ratios. For a more complete discussion of the different strategies for robust engineering design see Chapter 7.

6.5.3 Region Reduction

Region reduction can be approached in a sequential manner for the multiple criteria optimization problem. There are a few ad hoc rules to remember during region reduction.

1. Constraint response variables that are only influenced by one or two input variables should be handled first.
2. If $g_i(\mathbf{Y})$ is part of the objective function, $\hat{g}_i(\mathbf{Y})$ should be used if the prediction error for this model is not much worse than the error for $g_i(\hat{\mathbf{Y}})$.
3. The information gained from modeling the responses can be used to help cluster inputs and the responses they influence into fairly disjoint groups. Then region reduction is a series of "independent" region reduction problems rather than one large one. The variables that form these disjoint sets should be handled next.
4. Input variables that have a minimal effect on the target responses should be used to make sure the new region is as far inside the constraint region as possible.
5. The ranges for input variables that most influence target response variables should be adjusted last.

The decomposition of the optimization problem into clusters may also be useful to reduce the cost of numerical optimization. More work needs to be done to automate this procedure.

Another benefit of using statistical models with numerical optimization algorithms is brought out in the example. Often investigators will be interested in finding a solution for a fixed value of one or more of the input variables. They may also want to look at several different settings for these inputs. The use of statistical models gives more flexibility to the investigator in experimenting with the selection of input values.

This example showed several advantages in using a sequential strategy to find an optimal circuit over single stage methods. The sequential method allows the investigator to develop successively more accurate predictors using many fewer simulation runs than a single stage method would to get a predictor of equivalent accuracy. For example, if one reduces the range of each of 20 input variable by half, the area of the new range for the circuit described is 0.005% of the initial range. One would need to take a huge number of observations in a single stage method to produce a design with a similar density of simulation runs and hence similar accuracy of the predictor. Also, by reducing the experimental space there is a strong possibility of simplifying the performance function used during optimization. This was seen in our example by the elimination of various constraints from the performance function at the end of stage 1. In a one stage method this reduction would not be possible.

6.5.4 Latin Hypercube Sampling

We, and other teams, have found Latin hypercubes useful for simulation experiments. Instead of complete randomization of the Latin hypercube columns with respect to each other, we now use a method that controls the pairwise correlations between factors. For the examples of this article, near-orthogonality of the columns is probably desirable. Alternatively, if two uncontrollable variations are correlated, this could also be accommodated. For more details on the properties of Latin Hypercube Sampling see Chapter 4.

The use of LHS designs led to some unsuspected benefits in the digital logic circuit example. Because LHS designs use many levels for each input variable, drawing conclusions for simple response functions V_{DN}, V_{UP}, I_{OL}, and I_{OH} was straightforward. No modeling was required; plotting some simple graphs,

which should be plotted as a matter of course, gave enough information to come to the necessary decisions. Even the decisions for T_F and T_R could be reached by looking at some simple contour plots. This is because LHS when projected to lower dimensions is still a LHS. Even though the general relationships between inputs and the responses were known in advance and are relatively simple, without using LHS it would not have been possible to reach the appropriate conclusions as quickly, if at all. If the true relationships are not known even in these simple responses it could become difficult to model them with the two or three input levels typically used in most experimental plans.

6.5.5 Plots of Factor Effects

Our sequential strategy is assisted by graphical display of the effects of individual factors (Step 4). More automation in moving to the next subregion is desirable, and we are working on this problem, but engineers nonetheless often find these plots revealing. The information is easily displayed and little experience is required to interpret them.

6.5.6 Computation Time

When the stochastic-process model (6.2.2) is used, computing the maximum likelihood estimates of the correlation parameters is often the biggest computational expense. In the voltage-shifter example, the total time for Steps 3 and 4 (fit, check, and plot) for all nine performances is 6 hours on a SUN 3/80 at Stage 1. For comparison, generating the 75 simulator runs takes about 2 hours, and the optimization requires about 30 minutes.

In our experience these times are quite reasonable for moderately complicated problems like that of the voltage shifter, which was chosen because previous attempts at analysis had not been entirely successful. Fitting polynomials, usually a cheaper alternative, requires more data to achieve commensurate accuracy. More data obviously increases simulation time and, if used to fit larger polynomial models, model fitting and selection may become burdensome.

The model fitting strategies in Steps 1 and 3 have been used by other workers (A. Owen, J. Koehler, and S. Sharifazadeh at Stanford) for modeling device simulators (e.g., PISCES, SUPREM). Often, generating data is much more expensive for such simulators than for circuit simulation. With expensive data, the greater efficiency of the stochastic-process model (6.2.2) becomes even more advantageous.

6.6 Conclusion

We have given a strategy for finding optimum circuit designs with comparatively few circuit simulator runs. This strategy uses sequential experimentation and statistical modeling of performance functions to accurately approximate the simulator over successively smaller sub-regions on which to search for the optimum. We use models that treat a performance function as the realization of a stochastic process. These models are more data adaptive and flexible than polynomials, hence better accuracy typically follows. Many factors, performance functions and criteria can be treated using this strategy. The features of this strategy and its advantages are exemplified in two instances.

6.7 References

1. Brayton, R. K., Hachtel, and Sangiovanni-Vincentelli, A. L., (1981) "A Survey of Optimization Techniques for Integrated-Circuit Design," *Proceedings of the IEEE*, 69, 1334-1362.

2. Low, K. K. and Director, S. W., (1988) "An Efficient Macromodeling Approach for Statistical IC Process Design," in *IEEE Int. Conf. on Computer-Aided Design*, pp. 16-19, Santa Clara, CA.

3. Welch, W. J., Yu, Tat-Kuan, Kang, S. M., and Sacks, J., (1990) "Computer Experiments for Quality Control by Parameter Design," *Journal of Quality Technology*, 22, 15-22.

4. Yu, T. K., Kang, S. M., Sacks, J., and Welch, W. J., (1991) "Parametric Yield Optimization of CMOS Analogue Circuits by Quadratic Statistical Circuit Performance Models," *International Journal of Circuit Theory and Applications*, 19, 579-592.

5. Taguchi, G., (1986) *Introduction to Quality Engineering,* Asian Productivity Organization, Tokyo.

6. Box, G. E. P. and N. R. Draper, (1969) *Evolutionary Operation,* Wiley, New York.

7. Sacks, J., Schiller, S. B., and Welch, W. J., (1989) "Design for Computer Experiments," *Technometrics*, 31, 41-47.

8. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., (1989) "Design and Analysis of Computer Experiments," *Statistical Science*, 4, 409-435.

9. Dehnad, K., (1989) *Quality Control, Robust Design and the Taguchi Method*, Wadsworth.

10. Phadke, M. S., (1989) *Quality Engineering Using Robust Design*, Prentice Hall, New Jersey.

- Chapter 7 -

Discussion of Statistical Methods of RED

7.1 Introduction

It is useful to separate the general strategy and philosophy of RED, where there is widespread agreement, from the tools and methods that have evolved to implement this philosophy. All the methods are concerned with the same subjects: model fitting, interactions, loss functions and optimization, noise factor distribution and the efficiency with which RED can be carried out. It is useful to discuss these topics separately rather than within the different methods. By subdividing the RED problem into fairly independent components it may be easier to build a consensus strategy. This may eliminate some of the confusion that is inevitable when there are several methods available for those attempting to apply RED in industry. The rest of the sections in this chapter discusses these subjects and other RED related topics.

7.2 Estimation and Noise Parameter Distribution

When computing the mean and variance of Y the expectation is really over the noise factors, $\mathbf{U}$, not Y since Ω is the true sample space. To obtain estimates of the mean and variance of Y the distribution of $\mathbf{U}$ needs to be defined or approximated in some way. One of the fundamental differences between the LM approach and the RM approach is how assumptions are made about the distribution of $\mathbf{U}$. The other fundamental difference in the two approaches is how they estimate $E(Y)$ and predict estimates of $E(Y)$ at new settings of the design factors.

The methods for estimating the mean and variance of the response in the LM approach and the RM approach are almost the reverse of each other. The RM approach uses the predictor $\hat{y}(\mathbf{c},\mathbf{U}) = \hat{y}(\mathbf{X})$ to estimate Y and

$$E_U(\hat{y}(\mathbf{c},\mathbf{U})) = \int_{\Omega_N} \hat{y}(\mathbf{c},\mathbf{U}) d\mathbf{U}$$

is used to estimate $E_U(Y)$. $E_U(\hat{y}(\mathbf{X}))$ in turn can be estimated by

$$\hat{E}_{\mathbf{U}}(\hat{y}(\mathbf{c},\mathbf{U})) = \sum_{\omega_N} \hat{y}(\mathbf{c},\mathbf{U}) d\,\mathbf{U},$$

where ω_N is a computer generated random sample from the same distribution as **U**. The LM approach estimates $E(Y(s_D)) = \eta(\mathbf{s}_D)$ and $Var_U(Y(s_D)) = \sigma^2(\mathbf{s}_D)$ by

$$\hat{\eta}(\mathbf{s}_D) = \sum_{\mathbf{s}_i \in \mathbf{s}_N} y_i(\mathbf{s}_D)$$

and

$$\hat{\sigma}^2(\mathbf{s}_D) = \sum_{s_i \in S_N} (y_i(\mathbf{s}_D) - \hat{\eta}(\mathbf{s}_D))^2$$

respectively the mean and variance of Y over the outer array, S_N, at each design point, $\mathbf{s}_D$, of the inner array. The LM approach can then use the estimates $\hat{\eta}(\mathbf{s}_D)$ and $\hat{\sigma}^2(\mathbf{s}_D)$ to minimize variance and adjust to target or the estimates can be treated as response variables and modeled themselves, let the models be designated $\hat{\hat{\eta}}(\mathbf{c})$ and $\hat{\hat{\sigma}}^2(\mathbf{c})$ respectively.

There are two sets of assumptions needed for the estimation of $E_{\mathbf{U}}(Y)$ and $Var_{\mathbf{U}}(Y)$, the model fitting assumptions and the distribution assumptions of **U**. For the RM approach all the input factors are treated like design factors, i.e. none of the input factors are random variables, when modeling $\hat{y}(\mathbf{X})$. The distribution of **U** is only needed when estimating the risk function. The LM approach models the mean over **U**, so $\hat{\eta}(\mathbf{c})$ and $\hat{\sigma}(\mathbf{c})$ are functions of only the design factors. However, model *and* distribution assumptions about **U** are involved when estimating $\hat{\eta}(\mathbf{c})$ and $\hat{\sigma}(\mathbf{c})$ and so are involved in the estimation of $\hat{\hat{\eta}}(\mathbf{c})$ and $\hat{\hat{\sigma}}^2(\mathbf{c})$.

7.2.1 Distribution of Noise Parameter

Since the RM approach has no "replication" at the design factors it is impossible to get estimates of $E_{\mathbf{U}}(Y)$ and $Var_{\mathbf{U}}(Y)$ without explicitly stating the distribution of **U**. This allows for a lot of flexibility in the choice of distributions, but also requires a degree of knowledge (or confidence) by the designer to write down a distribution. Common distributional assumptions are that **U** has a symmetric distribution with mean zero and variance σ^2. It is also common to assume that **U** is normally distributed. One benefit of the RM approach is that changes in the assumed distribution can be made and new estimates of $E(Y)$ and $Var(Y)$ using a new distribution can be computed just by generating a random sample of $\hat{Y}$ using the new distribution. No new observations need to be taken

from the simulation model. These random samples should be inexpensive to generate since the predictor is inexpensive to run.

As described in Chapter 1, the LM approach estimates the mean and variance by using the outer array to get sample means and variances. The settings of the noise factors in the outer array are fixed at given experimental design values. In this situation, the *proxy* distribution of **U** is a discrete distribution with probability $1/n_N$ for each run in the outer array. The outer array could also be thought of as a set of representative points of the distribution of the noise factors. The nature of the distribution, assuming that the points in the outer array are truly representative, will depend on the experimental design of the outer array. Many of the assumptions that are stated explicitly in the RM approach, plus some that are not, are made implicitly when creating the outer array. Hence, the designer is not eliminating the decisions about these assumptions, they are just being made indirectly.

A common choice of experimental design for the outer array is a two-level orthogonal array which assumes that there are a minimal number of interactions between noise factors. If the points of this design are to be "representative" of the distribution of **U**, the noise factors must be independent and the distribution be symmetric with mean zero and variance a function of the range of the noise factors. The latter assumptions were made in the response model approach, while the assumption of independence was not. These assumptions can not be changed without changing the design of the outer array or changing to the RM approach style of analysis.

7.2.2 Model Fitting Assumptions

The design factors are the usual focal point of model fitting for both the RM and LM approach. However, the noise factors also have a role to play in fitting the response for *both* approaches. In the RM approach the design and noise factors are treated in the same manner when building the model.

Since the RM approach uses $\hat{y}(\mathbf{X})$ to get estimates of the mean and variance, only distribution assumptions enter into the computation of $\hat{E}_{\mathbf{U}}\hat{y}(\mathbf{c})$. This creates a nice division between modeling and distributional assumptions. Given the distribution of **U**, errors in the model will effect the estimation of the mean and variance. Model errors could occur due to bad assumptions for either the design factors and the noise factors, but are restricted to the estimation of $\hat{y}(\mathbf{X})$.

The LM approach assumptions about model building are split into two parts, the design factors are used to model $\hat{\eta}(\mathbf{c})$ and *model* assumptions for the noise factors are also implied when choosing the outer array for *estimating* $\hat{\eta}(\mathbf{c})$.

When a two level experimental design is used for the outer array the requirement that these points be "representative" of the distribution of $\mathbf{U}$ forces some implicit assumptions about the effect of $\mathbf{U}$ on Y. Since it is only a two-level design it is necessary to assume the effect of $\mathbf{U}$ on Y is linear. So even though the LM approach appears to more "data-driven" than the "model-driven" RM approach this is not the case. Unlike the LM approach, the RM approach does not need experimental designs with small numbers of levels which gives the RM approach greater flexibility in choosing designs and models.

It is also worthwhile to note that

$$E(f(\mathbf{c},\mathbf{U})) \neq E(f(\mathbf{c},\mu_U)),$$

where $\mu_U = E(\mathbf{U})$, unless $f(\mathbf{c},\mathbf{U})$ is a first order linear function in the noise factors. The designer may be particularly prone to this error when the RSM method is used. For example, if

$$Y = \mathbf{g}'_1(\mathbf{c})\beta + \mathbf{g}'_2(\mathbf{U})\alpha + \mathbf{g}'_3(\mathbf{c},\mathbf{U})\gamma$$

and let $E(\mathbf{U}) = 0$, then $E(Y) = \mathbf{g}'(\mathbf{c})\beta$ only if $\mathbf{g}(\mathbf{U})$ is a first order linear function for all the noise factors.

The assumptions for the design factors are typical model fitting assumptions, e.g. interactions, additivity, linearity. If the assumed model does not fit then errors in prediction will occur. The stochastic process used in DACE makes fewer direct assumptions about the shape of the function, such as interactions and linearity, than the LM approach and RSM, but model assumptions need to be checked for all the methods discussed. An important reason why so many different PerMIA's are needed is that they are dependent on which model assumptions are made. There are many model checking tools available and these should be used to determine the accuracy of the model.

The design-noise factor interactions are the driving force behind RED. Without these interactions reductions in product variability would not be possible. So it is important to identify and accurately estimate the important interactions. The LM approach avoids identification decisions by making all design-noise factor interactions estimable with the negative consequence of large sample

sizes. Most of the degrees of freedom in the LM approach are used to estimate all the $D \times N$ interactions. It is unlikely that all these terms are significant. It is more efficient to choose which interactions to estimate including $D \times D$ and $N \times N$ interactions. The RM approach is flexible in choosing which interaction terms to estimate, helping to reduce sample size, but opens up the possibility of missing an important interaction. This trade-off is a fundamental difference between the two approaches.

7.2.3 Further Topics

It is important to remember why the designer is interested in the variance of the noise factor. Variance is used to estimate the range of the noise factors. If the noise factor has a normal distribution with mean zero and variance $\sigma^2 = 1/9$ then 99% of input values will be between [-1,1]. If the noise factor has a uniform distribution and $\sigma=1/3$ only 91% of input values will be in the range [-1,1]. If the range [-1,1] represents 95% of the input values given a normal distribution it will only cover 70% of values if the noise factor has a uniform distribution. These examples show that even if an accurate estimate of the variance of the noise factors is known, if the distribution is not known there can be a significant misrepresentation of the actual product variation.

In both the LM approach and RSM it is suggested that some screening experiments be carried out. Screening of design factors is useful in reducing the sample size of future experimental plans. These screening experiments frequently have only 2-level experimental plans. Both the LM approach and RSM rely on investigating one region of the input space then moving to another, possibly completely new, region in the search for optimal product factor settings. Screening results may become invalid when moving from one region to another. If the response has a complicated surface, design factors that were unimportant in the original space may be important factors in later regions. This is true especially if there has been a considerable shift in location of the input range. If these new regions are not rescreened important factors may be lost.

DACE starts by looking at a large design factor space. DACE will have a tendency to miss factors with smaller effects because the power of the experimental plan will not be strong enough to find them. Small effect factors will start to appear as the subregions become smaller so it is important not to screen out factors too readily when using this approach as well.

7.3 Loss Functions and Optimization

Finding the product factor settings that best achieve the design specifications under manufacturing and environmental variability is the goal of RED. The product response can usually be described in mathematical terms. Finding optimal solutions of a mathematical function is the goal of a large body of research. If the mathematical functions are known, and computing solutions to these functions is inexpensive, it is straightforward to use a numerical optimization algorithm to search for a solution. In RED the function is usually unknown (physical experiments) or the function is too expensive to put through a numerical optimization algorithm (computer experiments). If an accurate and cheap predictor can be found it can be used in place of the unknown or expensive function. This is the goal of the RM approach. It seems wasteful to ignore the large volume of work on optimization and not aim for the best solution as directly as possible by using these optimization techniques.

The response surface over a large range of the design and noise factors is likely to be a complicated function. Simple first or second order polynomials will often not fit the true response surface very accurately. This has led RSM to focus on a small area of the input range. The response surface is more likely to behave as a low order polynomial locally. The search for optimal solutions is then a matter of moving from one region to another until an optimal region is found. We will refer to this as small region exploration. DACE is capable of modeling more complicated surfaces than second order polynomials so the designer can attempt to model much larger regions. DACE then looks for subregions where likely solutions appear to be located. We will refer to this as large region exploration.

There are several drawbacks to using small region exploration. When using small region exploration it is necessary to have nominal settings available before trying to minimize product variability, otherwise it may not be possible to adjust to target for the range of inputs chosen. Once the nominal settings have been chosen, reducing variability only guarantees an optimal solution for that starting point. It is possible that another choice of initial nominal settings may lead to a better optimal solution. Also, optimization algorithms are hampered when using small region exploration because the algorithms can not search for global optima over the full input space.

To get a reasonable picture of the response surface with large region exploration it is important to have a fairly large experiment at the first stage, possibly larger than the first stage of RSM but still considerably less then the LM approach. The goal of the first stage is to be able to find general trends and relationships between variables. It will give enough information to choose a subregion where a new, smaller experiment will give a more accurate picture of the response surface. It is true that looking over a large design factor range will initially include a lot of space that does not remotely meet the response requirements. Some of this is due to taking observations on a rectangular space and even though the information may be "useless" for the problem at hand it could be valuable later. It also allows the designer to locate *all* feasible regions. This gives more flexibility in choosing the best solution possible. The focusing aspect of large region exploration gives designer a solid endpoint to the search. Once an accurate predictor has been established there is no need for further experimentation.

The performance measure in engineering design problems may be very complicated. Frequently the performance measure will involve many responses, i.e. multi-objective criteria. Besides the usual task of reducing variability some responses may need to satisfy some constraint. Sometimes specifications require a function of several responses to be constrained in some way. Trying to develop portmanteau performance measures, such as SN ratios and PerMIAs, to handle all situations is a daunting task. A performance measure can handle a much larger range of design specification problems other than just reducing variability when using predictors of the response. The task of developing performance measures is not difficult with the RM approach. The performance measure typically will closely mirror the given engineering design specifications. Performance measures based on real engineering objectives can be used with confidence since the predictor emulates the true behavior of the process under study. Multi-objective optimization already familiar in engineering design studies to find nominal settings would require only a small adjustment to include robustness requirements.

The use of a numerical optimization algorithm is simplified if the terms in the performance measure that reduce variability are mean squared error rather than variance. This expectation can be written as the sum of two terms, $(E(\hat{y}(x)) - T)^2$, or bias, and $Var(\hat{y})$. Bias will dominate this sum when starting

the numerical optimization from a random set of values for the input factors. This could lead to finding solutions that are only local optima. However, the nature of region selection in DACE provides a check against this occurring. When selecting a new region for investigation the designer can use main effect and interaction plots to identify dispersion factors and regions where variability is reduced. This helps to ensure the new region contains a solution that gives a real reduction in variability.

7.4 Sample Size

Computer simulation models typically have large numbers of variables available for investigation. This is because the functions are complicated and all the factors that may affect the output are known and readily available for experimentation. For computer experiments the product array becomes very costly for even modestly sized problems by computer experiment standards. For example, a simulation code with 20 control factors and 10 noise factors will require over 800 runs of the simulation code. If each run takes one minute of CPU then the full experiment will take over 13 hours of CPU time. The combined array reduces the size of the experimental plan, it will need only a small fraction of this number of runs for an identically sized problem. Also, remember that any screening experiments that are used to help reduce the size of the product array should be included when discussing the costs of factor design.

A single experiment using DACE typically will not yield an immediate solution. Further experimentation will be necessary to explore the chosen subregions. The cost of this sequential experimentation is a linear function. The example given in the first paragraph of this section would allow 5 experiments of 150 runs each before being equal to the number of runs in one product array experiment. Examples from this thesis and Bernardo, *et al.*[1] show that for fewer runs than in one step of the LM approach DACE will find a globally optimal solution. Since the RSM approach uses combined arrays the size of experimental plans will be similar to DACE. The difference in sampling costs between DACE and RSM will depend more on the number of experiments carried out in the sequential search for a solution. DACE is likely to need fewer experiments than the RSM approach because DACE is able to accurately model much larger regions than RSM.

7.5 Ease of Use

An important reason why Taguchi's methods have become so widespread is the ease with which they can be learned and incorporated into real engineering design problems. Recent work on LM approaches have done much to improve on Taguchi's initial method. However, the improvements have created new complications or pointed to overlooked ones. The designer now must be concerned about transformations, interactions, design selection, and selection of the performance measure. When selecting the performance measure the designer must either select the correct SN ratio or PerMIA if using "traditional" methods or model both the mean and variance. For multi-variate response RED problems this can quickly become cumbersome as it doubles the number of models that need to be estimated. After factor settings minimizing product variability have been found, location factors need to be adjusted so the response is on target. Decisions then need to be made whether a new experiment should be conducted to make further improvements. Many of these concerns apply to RSM as well.

DACE helps to eliminate some of these choices and works toward finding an optimal solution. By looking at each stage individually it is apparent how the number of decisions that the designer needs to make can be reduced.

1. Choose Design. The use of LHS in computer experiments makes design selection easy, no concern about interactions and aliasing. Only a decision about sample size is necessary and general guidelines are already available.
2. Analyze Results. The use of the stochastic process eliminates the need to decide on the functional form to be estimated as well as expanding the class of functions that can be modeled.
3. Plot Analysis and Optimize. The plots help to identify important factors and the nature of their effect on the responses. The plots, combined with optimization results, help to locate solution regions. The optimization can take advantage of prepackaged optimization code so the designer only needs to develop the performance measure.
4. If the model is not accurate enough, choose the next subregion. Once a likely area to look in is selected, choosing the next subregion is not difficult. For example, reducing the ranges of 10 out of 20 factors by half reduces the size of the new region to .001 of the previous region. This is one reason why the stochastic process can produce accurate models after

only a few stages. This allows the designer to be conservative in selecting the subregion and reduces the chance of missing the "correct" region.

4a. If satisfied with the model, confirm solution from 3. Confirmation is important for any statistical approach to RED. Once the results of DACE have been confirmed the designer can then go on to search other promising regions that may have been discovered from the initial stages of experimentation. When these regions have been investigated the designer can be confident that the best possible factor settings for that engineering design have been found.

The first three steps can be easily automated. The designer only needs to be concerned with understanding the effects plots and writing routines to calculate the performance statistic for the optimization code. Step 4 is more difficult to automate because for large design problems there may be several subregions worth investigating and choosing the size of the next region is a fairly subjective process. At this point engineering principles as well as model information may be applied to help with selecting the next subregion. Choosing when to stop building new subregions is also a subjective decision depending on the model accuracy needed by the designer.

7.6 System and Tolerance Design

Robust engineering design has been split into three major phases, system design, parameter design and tolerance design. This categorization is useful but it has also led to a division in the development of methods in RED. The work on methods for parameter design is a good example. The LM approach and RSM typically assume that some nominal setting of product factors is available, i.e. that system design has been completed. Once factor settings have been chosen to increase product robustness the task of tolerance design is left uncompleted.

Separating system design from parameter design has led most research on methods for parameter design to assume that some nominal setting of the control factors is available as a starting point. However, a significant part of the cost in engineering design is the time it takes to find initial settings. It would be useful to eliminate the need of finding an initial nominal setting, but modeling the variance and the mean separately makes this difficult. When minimizing variability without any concern for the mean there is a strong chance that adjustment to

target will be impossible without affecting product variability. The LM approach and RSM have avoided this difficulty by using the nominal settings as a base from which to start.

DACE used in computer experiments can easily combine parameter design with the part of system design that involves finding the nominal settings. The designer typically knows the factor settings which meet engineering design specifications will fall within some broad region. Instead of spending time finding the nominal settings, the designer can perform an experiment using these expanded ranges and generate estimates of the responses. The designer can use these predictors to search for regions where engineering design specifications are met. This gives the designer several advantages. The first advantage is that the last stages of system design may be skipped. Second, it allows systematic search in a much larger region to locate all feasible designs. This could lead to a design selection that may have been overlooked. Third, it gives the designer an early glimpse at how the product factors behave in changing the response.

Tolerance design can be incorporated into either the RM or the LM approach. The available factor plots can be used to determine the spread of the response due to a specific noise factor. If the noise factor has a small slope this would be a candidate for loosening the specifications. Alternatively, the discovery that a noise factor has a large influence on the response could lead to a tightening of the tolerance for that noise factor. This information is not readily available when using the LM approach because it is bound up in the performance measure. The advantage of the RM approach is that the noise factor distribution can be changed and the effects tested without further experimentation by using the predictor to estimate what would occur under the new noise distribution.

7.7 Physical RED vs. Computer RED

The ideas in DACE discussed in this thesis were developed from work on computer-aided design problems. Many of the ideas are also applicable to physical RED problems. There are a few points where difficulties may arise. These areas need to be investigated to see what changes might be made for use in physical RED problems. One aspect of physical RED problems that is not of concern in CAD/CAE RED problems is experimental error. Experimental error is the result of uncontrolled noise factors, noise factors whose variability is

measured by replication and are not part of Taguchi's outer array, and measurement error. The ideas of combined arrays and modeling the response are still valid for controllable noise factors. When experimental error exists it needs to be measured and cannot be assumed to be constant over the range of inputs. Replication will be necessary and a measure of variability will need to be taken at each point in the combined array. This measure will need to be modeled and incorporated into the performance measure. These are difficulties that affect all three strategies discussed.

In computer experiments changing the values of the input factors is a straightforward task. In physical experiments this is frequently not the case. It is unlikely that LHS can be used in its present form for physical experiments. The number of different settings can be reduced depending on the ease with which the factor setting can be altered in the physical experiment. An arrangement of run order could be found to reduce the number of changes. Work of a similar nature has already been done on orthogonal arrays,[2] and may be applicable to these adjusted Latin Hypercube Samples.

There is no theoretical reason why the stochastic process models used in DACE cannot be used in physical experiments. One way to incorporate experimental error is to use a covariance structure of the form $\sigma^2(R + \gamma I)$, where I is the identity matrix and $\sigma^2\gamma$ is a measure of experimental error. Further work is being carried out to see how effective this model is in physical experiments.

7.8 Conclusion

The LM approach is a popular and successful strategy in RED but many statistical weaknesses have become apparent. The LM approach builds simple linear models and makes assumptions about the noise factor distribution which are hard to assimilate with reality. The RM approaches build more complicated models but the assumptions involved with these models also occur, sometimes implicitly, in the LM approach. Checking model assumptions is equally important for the LM and RM approaches, but the extra flexibility in choosing experimental design in the RM approach gives the RM approach an advantage in checking the assumptions.

The RM approaches have more flexibility in the choice of noise factor distribution. This allows the designer to choose noise factor distributions which mimic real variability and not one that is forced upon the designer by the choice

of design when using the outer array of LM approach. Separating the distribution assumptions of the noise factors from the experimental design allows experimentation with different distributions without the need for further observations from the simulation model.

The use of a stochastic process as a predictor helps to reduce model error by interpolation through the design points. This works well with experimentation on computer simulation models. DACE has no hidden costs or hidden assumptions so the statistical difficulties can be approached directly. When using the stochastic process of DACE many of the modeling difficulties that arise when using linear models can be avoided, such as confounding, nonlinearity and additivity. DACE gives the designer a straightforward and efficient way to solve engineering design problems. It also leads directly to optimal solutions so more time can be spent by the designer developing entirely new engineering designs rather than making adjustments to old designs.

7.9 References

1. Bernardo, M. C., Buck, R., Liu, L., Nazaret, W. A., Sacks, J., and Welch, W. J., (1992) "Integrated Circuit Design Optimization Using a Sequential Strategy," *IEEE Transactions in Computer-Aided Design*, 11, 361-372.

2. Cheng, C.-S., (1985) "Run Order of Factorial Designs," in *Proceedings of Berkley Conference in Honor of Jerzy Neyman and Jack Kiefer*, vol. 2, pp. 619-633, Wadsworth Advanced Books and Software, Hayward, CA.

- Chapter 8 -

Conclusions and Further Research

The investigation of methods for robust engineering design has led to inquiries into a number of statistical topics as well as others, such as optimization. Two areas that were investigated more thoroughly because of their connection to computer experiments was spatial modeling and experimental design. Also, to help the development of robust engineering design methods a thorough investigation of the underlying probability model was necessary to identify weak points in the different methods outlined. The conclusions from this thesis can be roughly broken into spatial models, Latin hypercube sampling and robust engineering design.

Spatial Models

The spatial models described in Chapter 2 were used to predict the response of several different circuit simulation models. The predictions were accurate enough so that the predictor could be used to find optimal robust engineering designs. These examples, as well as others mentioned in the literature, are evidence that the predictor, particularly with the covariance function (2.3.1), can be used successfully to estimate computer simulation models. The examples also extend the used of spatial statistical models to higher dimensions than usually found in the literature.

The work in this thesis resulted in a software package for the design and analysis of computer experiments. The software is still fairly crude, but complete. There are still improvements that can be made in the efficiency of parts of the code, most notably in computing cross-validation results. While the research described in Chapter 3 led to maximum likelihood estimation methods for the model parameters at a reasonable cost in high dimension problems, work still can be done to improve the cost effectiveness of estimating the parameters for the spatial models described.

The study of spatial statistics is a fast growing field with many opportunities for further research. The covariance function used in this thesis has not been investigated thoroughly on a theoretical basis. Further study of the robustness of the parameters in the covariance function from both the prediction and MLE

perspective would be useful. An investigation of a variety of known functions would be useful to determine the range of functions that may be well estimated, e.g. the limits to roughness of a function or the effect of nonstationarity. The spatial model described in Chapter 2 could also be applied to problems with measurement error.

The theoretical basis for the ONETIME algorithm used to compute maximum likelihood estimates is not well developed. There are many avenues to explore in understanding why and when this algorithm works successfully. An important topic that is currently receiving some attention is the modality of the likelihood function. This is an intriguing area where our models do not quite fall into the families described in previous work. Further understanding of the parameter values and what they mean could also be helpful to gain a better intuitive feel to why the ONETIME algorithm works as well as it does.

Latin Hypercube Sampling

Latin hypercube sampling was developed for designing experiments for high dimensional computer experiments. Latin hypercube sampling is fast and simple to implement and previous work has shown that Latin hypercube sampling is asymptotically more efficient than simple random sampling or stratified random sampling and estimates from Latin hypercube sampling are asymptotically normally distributed.

Latin hypercube sampling together with the spatial models used for the research in this thesis appear to form a good basis for the design and analysis of computer experiments. This conclusion is based mainly on the nature of computer experiments, but asymptotic results from Chapter 4 show that $MSE(\hat{Y}) \rightarrow 0$ as $n \rightarrow \infty$ for Latin hypercube sampling, but not for simple random sampling or many deterministic designs when an interpolator is used for the statistical predictor.

Variability in the spatial location of design points in random designs is used as a measure of discrepency and comparisons were made between Latin hypercube sampling and simple random sampling. This measure of discrepency showed that Latin hypercube sampling is better, on average, in evenly spacing points throughout the input space.

The investigation of the space filling properties of Latin hypercube sampling is still at the initial stages. Further comparisons of the new discrepency

function with more experimental designs, in particular stratified random sampling, need to be examined. Also, values for other discrepency functions should be computed for Latin hypercube sampling for comparison to nonrandom designs.

Robust Engineering Design

Taguchi developed a simple experimental method that helped to make product performance robust to many types of variability, thus improving quality. The methods he used to find the product settings, while solid in principle, are not particularly efficient and have many theoretical problems from a statistical point of view. Examination of the underlying model in the RED problem show that the statistical assumptions about the type of function estimated and noise factor distribution are used for both the LM and RM approach, but the RM approach allows for more flexibility in choosing what assumptions are made. In particular, the RM approach has more flexibility in the selection of designs and statistical models available for estimation and prediction and more flexibility and opportunity for experimentation with different distributions for the noise factors.

DACE predicts the complicated response surface of simulation models better than regression polynomials used in RSM. DACE also can use a different approach to optimization. DACE can model a larger input space and still produce a reasonable accurate predictor, hence DACE does not need an initial set of nominal values before starting the optimization search. Investigation into using DACE on physical systems and manufacturing processes needs to be carried out to check what adjustments may be necessary to the procedures used in DACE. Also, optimization algorithms play a more important role when using DACE and further investigation is needed on choosing algorithms for these multivariate optimization problems.

The general strategy for experimentation in RED has largely been settled, however there are many statistical questions still worthy of further investigation. One area of study in RED that has largely been ignored is the estimation of the noise factor distribution. This would involve applying much of the work on density estimation and further research into estimating covariance matrices and multivariate density functions. This is an important area to pursue from an application point of view and has many opportunities in statistical research.

- Appendix 1 -

DACE Code and User's Guide

A1.1 User's Guide

The DACE software performs several tasks:

1. Computes parameter estimates of stochastic process using likelihood function in two ways:
   a. Full MLE, i.e. all parameters are free to vary.
   b. ONETIME algorithm, see Section 3.5.
2. Computes likelihood values (3.1.1) for any number of given parameter estimates.
3. Predicts response given parameter estimates, via (2.2.3).
4. Computes cross-validation for parameter estimates, via (3.4.1).
5. Computes values for "main effects" plots, see Section 2.4.

A1.1.1 Source and Header Files

There are 10 source files and 10 header files necessary to compile the DACE software. The source files end with the suffix .c and the header files end with the suffix .h and the file names will be written in bold type. The prefixes for the 10 pairs of source and header files are: **dace**,**rb_alloc**, **rb_covm**, **rb_effects**,**rb_like**, **rb_math**, **rb_matman**, **rb_opt**, **rb_predict**, and **rb_print**. The main function is in **dace** and contains the structure for choosing the user requested tasks. The main tasks for the rest of the source files are as follows:

1. **rb_alloc** - memory allocation routines.
2. **rb_covm** - various covariance computation functions.
3. **rb_effects** - functions for computing main effects.
4. **rb_like** - functions for computing likelihoods.
5. **rb_math** - miscellaneous mathematical functions.
6. **rb_matman** - some matrix manipulation functions.
7. **rb_opt** - functions involved with optimization.

8. **rb_predict** - functions for computing predictions.
9. **rb_print** - various input/output routines.

A1.1.2 Variable Definitions

All variables will be capitalized in this document for clarity. In the source code only global variables are capitalized. There are 20 global variables used in the DACE software they are identified below as part of their definition. All variables except a few obvious character strings are either integer or double scalars, vectors or matrices. Variable descriptions follow C programming language notation.

1. N - Number of runs in experiment. Number of rows in **S. Input by user. Global Variable.
2. NX - Number of input variables. Number of columns in **S. Input by user. Global Variable.
3. P - Number of parameters in linear model part. Typically P=1, a constant linear model. Input by user. Global Variable.
4. SEED - Seed for random number generator used for starting MLE or generating random inputs for prediction. Input by user.
5. NFILES - Number of data files where input variables and responses are stored. At this time looks for either all data in one file or input variables in one file and response variables in another file. Input by user.
6. NY - Number of response variables in input files. Input by user.
7. SELY - Response variable that user wants to analyze. Input by user.
8. **S - Input variables (columns) experimental run (rows) values. Input by user.
9. **TEMPY - Matrix with all response variables (columns) for respective experimental runs (rows). Input by user.
10. *KIN1,*KIN2 - Index vectors containing subscripts of variables used to compute linear model terms from **S. Some examples,
    a. if KIN1[1]=1 and KIN2[1]=0 then $f_1(\mathbf{x}_i)=x_{i1}$.
    b. if KIN1[1]=1 and KIN2[1]=1 then $f_1(\mathbf{x}_i)=x_{i1}^2$.
    c. if KIN1[1]=1 and KIN2[1]=2 then $f_1(\mathbf{x}_i)=x_{i1}x_{i2}$.

11. *Y - Vector with response variable selected for analysis by user. Global Variable.
12. NDEC - Used repeatedly as indice for decisions on whether tasks described above should be carried out. Typically 0 or 1 but for decision on MLE can be 0, 1 or 2. No MLE = 0, FULL MLE = 1, ONETIME algorithm = 2. Input by user.
13. **SDIFF - N*(N-1)/2 by NX matrix which contains |S[i][j] - S[k][j]| for i<k, k<N and j<NX. Global Variable.
14. **F - Contains linear model input values. Global Variable.
15. *THETA - Vector of covariance function parameters. See (2.3.1).
16. *POWER - Vector of covariance function parameters. See (2.3.1).
17. *BETA - Vector of linear model parameters. See (2.3.1).
18. **V - Correlation matrix.
19. GAMMA - "Nugget" parameter in covariance function, $(**V' = **V + \gamma I)$, where I is the identity matrix.
20. SIGMAZ - "Scale" parameter in covariance function.
21. READ_PARAM - Indicator variable. If MLE has not been carried out program prepares to a user given file to read in model parameters. The order of parameters in file is: SIGMAZ, GAMMA, -2*LN Likelihood, *BETA,*THETA,*POWER.

    The following variables are used only in likelihood computations.

22. NLOOP - For FULL MLE it is the number of random starts for computing MLE, suggested value ≤ 5. For CHEAP MLE it is the number of rounds through all variables, suggested value is 15. Input by user.
23. NLIKE - The number of user given sets of model parameters which will be used to compute their likelihoods. Input by user.
24. FUNC_DECISION - Determines the type of constraints on the covariance function during the computation of the MLE:
    a. FUNC_DECISION=0: FULL MLE, all parameters except GAMMA=0 free to vary.
    b. FUNC_DECISION=1: COMMON (θ, p), $\theta_1 = , \ldots , = \theta_d$ and $p_1 = , \ldots , = p_d$.

c. FUNC_DECISION=2: SINGLE (θ, p), optimize over (θ_i, p_i) only.

d. FUNC_DECISION=3: GAMMA ONLY, Fix (θ, p) and optimize over GAMMA only.

This is a global Variable.

25. NGAM - Indicator variable that indicates whether GAMMA is fixed = 0. Global Variable.

26. OLD_THETA - Contains value of θ_i when optimizing one at a time on x_i. Global Variable.

27. OLD_POWER - Contains value of p_i when optimizing one at a time on x_i. Global Variable.

28. NPK - The variable which is currently being optimized during one at a time MLE. Global Variable.

29. LIKE - The likelihood value.

30. OLD_LIKE - The likelihood value from previous stage.

The following variables are used only in prediction and cross-validation computations.

31. NUMPRED - The number of points at which user wants to predict. Input by user.

32. RAND_IND - Indicator variable states whether user would like set of random input points for prediction, yes = 1. Input by user.

33. TRUEY_IND - If user gives a set of points for prediction, TRUEY_IND is indicator variable stating whether true responses are available, yes = 1. Input by user.

34. **PREDX - Set of points which will be used for prediction. Input by user if RAND_IND = 0.

35. *TRUEY - Response values for prediction sites. Input by user if TRUEY_IND = 1.

36. *YHAT - Prediction estimates for PREDX.

37. *EMSE - Expected RMSE estimates for PREDX.

38. **SUBV - Used in Cross-Validation. **SUBV=$**V_{-i}$, where $**V_{-i}$ is **V with the i^{th} row and column deleted.

39. **SUBS - Used in Cross-Validation. **SUBS=**S_{-i}, where **S_{-i} is **S with the i^{th} row deleted.

40. *SUBY - Used in Cross-Validation. *SUBY=*Y_{-i},where **Y_{-i} is **Y with the i^{th} row deleted.

41. **FPRED - Linear model terms for prediction sites.

42. *CVYHAT - Stores Cross-Validation estimates of response.

43. *CVMSE - Stores Cross-Validation estimates of Expected RMSE.

The following variables are used only in main effects computations.

44. INTLEV - The highest order of interaction levels user is interesting in computing. INTLEV=1 main effects only, INTLEV=2, Second order interactions. Input by user.

45. USESIG - Use "significance test" to determine which variables to include in main effects computations. The "significance test" is $H_0: \theta_i = 0$, significance level is $\Delta(-2ln\ L) > 6$. Input by user.

46. INT_IND - Indicator vector of length NX which states which variables are included for main effects computations, yes=1.

47. NUMPTS - The number of points at which to compute main effects. Currently hardwired into program at NUMPTS=21.

48. *MUIND - Vector which contains labels i of x_i the variables for which main effects are computerd

49. *X - Vector of points at which to compute main effects.

50. **MU - Results of main effects. For main effects columns are for different variables. For higher order interactions columns is results at X_{jk}.

51. MU0 - Overall mean.

52. *MU1 - Main effects.

53. NUMINTR - Number of interactions computed so far.

54 **INTR - Contains integration values for variables included for main effects computations.

55 *MULEXP - Product of integrals for variables not included for main effects computations.

56 *NULEXP - Product of integrals for variables included for main effects computations.

57 SSY - estimate of variability of estimated response surface from 1000 randomly selected points, $1/n_r \sum(\hat{y}_i - \mu_0)^2$.

58 *SS - Squared deviation of main effects from zero.

A1.1.3 Subroutines

Subroutines which can be found in the source files listed above except **rb_alloc.c** are briefly described below. For clarity they are written in this documentation as SUBROUTINE(). In the source code they are not capitalized.

1. void COVM() - Covariance function when computing full MLE. In **rb_covm.c**.
2. void COVM1() - Covariance function when computing MLE with common (θ,p) for all variables. In **rb_covm.c**.
3. void CHP_COVM() - Covariance function when computing MLE for ONE-TIME. Optimizing over one variable at a time, both θ and power. In **rb_covm.c**.
4. void SLC_COVM() - Covariance function when computing MLE for FORWARD. Optimizing over one variable at a time, but power fixed. In **rb_covm.c**.
5. double CALMU0() - Computes overall integrated mean. In **rb_effects.c**.
6. void CALMU1() - Computes main effects. In **rb_effects.c**.
7. void CALMU2() - Computes interaction effects. In **rb_effects.c**.
8. void INIT_ME() - Is the structure within which main effects, etc. are computed. In **rb_effects.c**.
9. double GETLIKE() - Computes likelihood for Gaussian stochastic process. In **rb_like.c**.
10. double COMPMLE() - Computes all parameter estimates for MLE. GETLIKE only computes estimates for θ, p, γ. In **rb_like.c**.
11. double CHPMLE() - Structure which computes CHEAP MLE. In **rb_like.c**.
12. void DET_SIG() - Determines "significance" of input variables. H_0: $\theta_i = 0$. Critical value is $\hat{\theta}_i \leq 6.0$. In **rb_like.c**.
13. void MULT_LIKE() - Structure which computes as many likelihoods as desired for user given parameter values. In **rb_like.c**.

14. void DES_DIST() - Computes $|s_{ik} - s_{jk}|$ for all $i < j \leq N$ and $1 \leq k \leq NX$. In **rb_matman.c**.
15. void VEC_V() - Takes **V, the covariance matrix, and translates it to a vector. In **rb_matman.c**.
16. void MAT_V() - Takes the vector version of **V and translates it back to a matrix. In **rb_matman.c**.
17. void CD() - Computes Cholesky decompostion. Output is lower triangular matrix. In **rb_math.c**.
18. void INVMAT() - Inverts matrix. Result is returned in lower triangular part of input matrix. In **rb_math.c**.
19. double RB_ABS() - Computes absolute value of x. In **rb_math.c**.
20. void QR() - Computes QR decompostion. In **rb_math.c**.
21. void XTOF() - Takes design matrix **S and computes **F the input matrix for the linear model part of stochastic process. In **rb_math.c**.
22. void GET_RINVY() - Computes $R_S^{-1}y$ for main effects. In **rb_math.c**.
23. double NORMIN() - Computes integral of $exp(-\theta|s_{ik}-s_{jk}|^{p_k})$. In **rb_math.c**.
24. double FUNK() - Function used in optimization routine for computing MLE's. In FUNK, FUNC_DECISION determines which covariance function to use. In **rb_opt.c**.
25. int AMOEBA() - The optimization routine used to compute MLE's It has been taken from Nelder & Mead *Numerical Recipes* for FORTRAN and translated to C. In **rb_opt.c**.
26. double RUNOPT() - Function from which optimization routine is run. In **rb_opt.c**.
27. void INITIAL() - Computes some initial calculations needed for getting predictions. In **rb_predict.c**.
28. void PREDICT() - Computes $\hat{y}$. In **rb_predict.c**.
29. void GET_EMSE() - Computes $MSE(\hat{y})$. In **rb_predict.c**.
30. void GET_PRED() - Function from which prediction and MSE estimates are computed. In **rb_predict.c**.

31. void GET_CV() - Function from which Cross-validation results are computed. In **rb_predict.c**.

32. double MEPRED() - Computes predictions for estimate of overall estimate of response surface variability. In **rb_predict.c**.

33. void PRNT_LIKE() - Routine prints results of likelihood calculations. In **rb_print.c**.

34. void PRED_OUT() - Routine prints results of prediction part of program. In **rb_print.c**.

A1.1.4 Example of Input To Dace

```
Title            /* title for job */
50 20 1 4658     /* n,nx,p,seed for random number*/
1 1 1            /* nfiles, ny,sely   */
toya.des50y      /* design file and response file  */
0                /* kin1 */
0                /* kin2 */
0                /* gamma */
/*  if any following decisions is no (=0) then associated
    inputs need not be entered                          */
1                /* decision on mle (1,full,2=cheap) */
5                /* # random starts for full, # loops thru cheap mle */
like.dump        /* dump file for mle */
mle.out          /* mle results file     */

1                /* decision on multiple likelihood calculations */
30               /* # of likelihoods to calculate          */
param.in         /* file with parameters for likelihood calculations */
like.out         /* file name for likelihood output        */

1                /* decision on prediction              */
param.in         /* file with parameters to read if necessary */
100 0            /* # points to predict at, from random inputs (yes=1)  */
toya.ran100y     /* file name with pts to predict at           */
1                /* Do you have true responses for prediction?  */
pred.out         /* file name for prediction output            */
```

```
1           /* decision for cv calcualations          */
cv.out      /* file name for cv output                */

1           /* decision for main effects calculations */
2 1         /* interaction level to compute, use sign level */
toya        /* prefix for output files: .mu0, .me, .int, .med added */
```

A1.1.5 Description of Output

The output files are reasonably self explanatory except for the main effects output and to a lesser extent the computation of likelihoods. Except for the main effects output, the output for each type of computation is their own file. As mentioned the output for the main effects is spread over four output files. There is also a dump file for maximum likelihood estimates which contains information about the optimization process, but the file **mle.out** contains all pertainent information.

All output contains the job title the parameter estimates used for the computation and the name of the data file used for the job. For the main effects output this information is in the file with suffix **.med**. The output files for the likelihood computations do not contain the parameter estimates. The results for prediction, cross-validation and maximum likelihood calculations are labeled and should be self explanatory. The parameter estimates from the MLE calculations are in the same order as needed for **param.in** file to make the input file easier to construct. The prediction and cross-validation output has the same format. A line after the parameter estimates gives the empirical RMSE, the standard deviation of the empirical RSME and the maximum deviation at the predicted values. A list of true and predicted values with $\widehat{RMSE}(Y)$ follows the summary statistics.

The main effects output is divided into four output files with the suffices described above. They will be refered to here as **output.med**, **output.me**, **output.int**, and **output.sf**. The output file **output.med** contains the parameter estimates, etc. and the ANOVA-like sums of squares for main effects and higher order interactions. The output file **output.mu0** contains the value of $\int \hat{y}(\mathbf{x}) d\mathbf{x}$, β, KIN1 and KIN2. The main effects computations assume that a constant model has been used. The value of μ_0 in (2.4.1) is obtained by adding $\int \hat{y}(\mathbf{x}) d\mathbf{x}$ and β. The main effects in (2.4.2) are found in the output file **output.me**. The first row of the output file contains the number of factors which the main effects

were computed for and the indices for those factors. The main effects values start in the second row of the output file and the columns are the results for the different factors. The interaction effects in (2.4.3) and joint effects are constructed the same way and are in the files **output.int** and **output.sf** respectively. The joint effects are

$$\eta_{ij}(x_i, x_j) = \int y(\mathbf{x}) \prod_{k \neq i,j} dx_k .$$

The interaction effects are for all pairs of variables i, j, $i < j$, found in the output file **output.me**. The data in the respective output files are the interaction or joint effects values over a grid of points with the second dimension changing fastest and the data being read row by row. The matrices for the effects are also ordered with the second variable increasing fastest.

```
/***************This is dace.h*****************************/
int N;
int NX;
int P;
double **S;
double *Y;
double **V;
double **F;
double **SDIFF;

int NOPT;
int NGAM;
int NPK;
int FUNC_DECISION;
int NFCALLS;
double OLD_THETA;
double OLD_POWER;
double OLD_GAMMA;
double **TV;

char DUMP_FILE[40];
char MLE_FILE[40];
char PRED_FILE[40];
/*********************************************************************/
/*              This is dace.c                                       */
/*This is the main function for dace and calls all other routines*/
/*-------------------------------------------------------------------*/
#include<stdio.h>
#include<string.h>
#include<time.h>
#include "dace.h"
#include "rb_alloc.h"
#include "rb_covm.h"
#include "rb_effects.h"
#include "rb_like.h"
#include "rb_matman.h"
#include "rb_predict.h"
#include "rb_dace_prnt.h"
#define RAND_MAX (pow(2,31)-1)
/*-------------------------------------------------------------------*/
main()
{
   int i,j,k,nfiles,ny,sely,*kin1,*kin2,ndec,seed,read_param=1;
   int numpred,truey_ind,rand_ind;
   int nloop,itmax,nlike;
   int intlev,usesig,*int_ind;
   double **s;
```

```
double *theta,*power,gamma,*beta,sigmaz,like;
double **predx,*truey,ran_num,*yhat,*emse;
double tol,**tempy;
long temp_ran;
char temp_name[100],*fname,*oname,*data_file,*data_file2,*title;
char temp_title[100],datetime[30];
long clock;

FILE *in1;
FILE *in2;
FILE *out;
FILE *out1;

/* Read in job title and dimensions of the problem          */

 clock=time(0);
 strcpy(datetime,ctime(&clock));

 title=gets(temp_title,100);

scanf("%d %d %d %d",&N,&NX,&P,&seed);
scanf("%d %d %d",&nfiles,&ny,&sely);

 scanf("%s",temp_name);
 data_file=AllocChar(strlen(temp_name));
 strcpy(data_file,temp_name);

 if(nfiles==2)
 {
   scanf("%s",temp_name);
   data_file2=AllocChar(strlen(temp_name));
   strcpy(data_file2,temp_name);
   in2=fopen(data_file2,"r");
 }

/*  Allocate space to standard variables                    */

s=AllocDouble2(N,NX);
SDIFF=AllocDouble2(N*(N-1)/2,NX);
Y=AllocDouble(N);
theta=AllocDouble(NX);
power=AllocDouble(NX);
beta=AllocDouble(P);
kin1=AllocInt(P);
kin2=AllocInt(P);
F=AllocDouble2(N,P);
V=AllocDouble2(N,N);
```

```
  int_ind=AllocInt(NX);
  tempy=AllocDouble2(N,ny);

/*  Read in data                                */

  in1=fopen(data_file,"r");

  for(j=0;j<N;j++)
  {
    for(i=0;i<NX;i++) fscanf(in1,"%lf",&s[j][i]);
    for(k=0;k<ny;k++)
    {
      if(nfiles==1)fscanf(in1,"%lf",&tempy[j][k]);
      if(nfiles==2)fscanf(in2,"%lf",&tempy[j][k]);
    }
    Y[j]=tempy[j][sely-1];
  }

  FreeDouble2(N,ny,tempy);
  fclose(in1);
  if(nfiles==2)fclose(in2);

  for(i=0;i<P;i++) scanf("%d",&kin1[i]);
  for(i=0;i<P;i++) scanf("%d",&kin2[i]);
  scanf("%lf",&gamma);

  NGAM= (gamma) ? 1 : gamma ;

  srandom(seed);

  scanf("%d",&ndec);

/*  This section computes MLEs using one at a time likelihood */
/*  estimation techniques              */

  if(ndec)
  {
    read_param=0;
    itmax=100;
    tol=0.0001;

    scanf("%d",&nloop);

    scanf("%s",temp_name);
    strcpy(DUMP_FILE,temp_name);

    scanf("%s",temp_name);
```

```
strcpy(MLE_FILE,temp_name);

out=fopen(MLE_FILE,"w");
out1=fopen(DUMP_FILE,"w");

xtof(s,N,P,kin1,kin2,F);
des_dist(N,NX,s,SDIFF);

fputs(title,out);
fputs(title,out1);
fprintf(out,"\n%s\n",datetime);
fprintf(out1,"\n%s\n",datetime);

if(ndec==1)
{
  fprintf(out,"\n");
  fprintf(out1,"\n");
  fprintf(out,"RESULTS FROM FULL MLE\n");
  fprintf(out1,"RESULTS FROM FULL MLE\n");
  fprintf(out,"\n");
  fprintf(out1,"\n");
  itmax=itmax*50;
 like=mle(out,out1,N,NX,P,SDIFF,F,Y,theta,power,&gamma,beta,&sigmaz,tol,\e
 nloop,itmax);
}

if(ndec==2)
{
  fprintf(out,"\n");
  fprintf(out1,"\n");
  fprintf(out,"RESULTS FROM ONE AT A TIME MLE\n");
  fprintf(out1,"RESULTS FROM ONE AT A TIME MLE\n");
  fprintf(out,"\n");
  fprintf(out1,"\n");

 like=chpmle(out,out1,N,NX,P,SDIFF,F,Y,theta,power,&gamma,beta,&sigmaz,tol,\e
 nloop,itmax);
}

fprintf(out,"\n");
fprintf(out,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
if(nfiles==2)fprintf(out," AND %s",data_file2);
fprintf(out,"\n");
fprintf(out1,"\n");
fprintf(out1,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
if(nfiles==2)fprintf(out1," AND %s",data_file2);
fprintf(out1,"\n");
```

```
    fclose(out);
    fclose(out1);


  }

  scanf("%d",&ndec);

/*  This section computes nlike likelihood values for     */
/*    nlike sets of theta and power  */

  if(ndec)
  {
    read_param=0;
    scanf("%d",&nlike);
    scanf("%s",temp_name);
    fname=AllocChar(strlen(temp_name));
    strcpy(fname,temp_name);
    in1=fopen(fname,"r");
    FreeChar(strlen(temp_name),fname);

    scanf("%s",temp_name);
    fname=AllocChar(strlen(temp_name));
    strcpy(fname,temp_name);
    out=fopen(fname,"w");
    FreeChar(strlen(temp_name),fname);

    fputs(title,out);
    fprintf(out,"\n%s\n",datetime);
    fprintf(out,"\n");
    fprintf(out,"LIKELIHOOD RESULTS\n");
    fprintf(out,"\n");

    xtof(s,N,P,kin1,kin2,F);
    des_dist(N,NX,s,SDIFF);

    for(j=0;j<nlike;j++)
    {
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&theta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&power[i]);

      covm(N,NX,SDIFF,theta,power,gamma,V);
      like=compmle(N,P,Y,F,V,&sigmaz,beta);

      fprintf(out,"RESULTS FOR PARAMETER SET %d\n",j+1);
      fprintf(out,"-2*LN LIKELIHOOD= %lf\n",like);
      fprintf(out,"SIGMAZ= %lf\n",sigmaz);
```

```
      for(i=0;i<P;i++) fprintf(out,"BETA= %lf\n",beta[i]);
    }

    fprintf(out,"\n");
    fprintf(out,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
    if(nfiles==2)fprintf(out," AND %s",data_file2);
    fprintf(out,"\n");

      fclose(in1);
      fclose(out);

  }

  scanf("%d",&ndec);

/*  This section computes predicted values for given parameters and */
/*    given or randomly selected prediction points   */

  if(ndec)
  {

    if(read_param)
    {
      scanf("%s",temp_name);
      fname=AllocChar(strlen(temp_name));
      strcpy(fname,temp_name);
      in1=fopen(fname,"r");
      FreeChar(strlen(temp_name),fname);

      fscanf(in1,"%lf %lf %lf",&sigmaz,&gamma,&like);
      for(i=0;i<P;i++) fscanf(in1,"%lf",&beta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&theta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&power[i]);
      read_param=0;
    }

    scanf("%d %d",&numpred,&rand_ind);
    scanf("%d %d %d",&nfiles,&ny,&sely);
    predx=AllocDouble2(numpred,NX);

/*   computing random inputs for prediction if required   */

    if(rand_ind)
    {
     for(i=0;i<numpred;i++)
       for(j=0;j<NX;j++)
       {
```

```
        temp_ran=random();
        ran_num=temp_ran;
        predx[i][j]=ran_num/RAND_MAX-0.5;
      }
    }
    else
    {

/*    or reading inputs in from file                    */

      scanf("%s",temp_name);
      data_file=AllocChar(strlen(temp_name));
      strcpy(data_file,temp_name);

      if(nfiles==2)
      {
        scanf("%s",temp_name);
        data_file2=AllocChar(strlen(temp_name));
        strcpy(data_file2,temp_name);
        in2=fopen(data_file2,"r");
      }

      in1=fopen(data_file,"r");
      tempy=AllocDouble2(numpred,ny);
      truey=AllocDouble(numpred);

      for(j=0;j<numpred;j++)
      {
        for(i=0;i<NX;i++) fscanf(in1,"%lf",&predx[j][i]);
        for(k=0;k<ny;k++)
        {
          if(nfiles==1)fscanf(in1,"%lf",&tempy[j][k]);
          if(nfiles==2)fscanf(in2,"%lf",&tempy[j][k]);
        }
        truey[j]=tempy[j][sely-1];
      }

      FreeDouble2(numpred,ny,tempy);
      fclose(in1);
      if(nfiles==2)fclose(in2);
    }
    scanf("%s",temp_name);
    fname=AllocChar(strlen(temp_name));
    strcpy(fname,temp_name);

    out=fopen(fname,"w");
    FreeChar(strlen(temp_name),fname);
```

```
    yhat=AllocDouble(numpred);
    emse=AllocDouble(numpred);

    fputs(title,out);
    fprintf(out,"\n%s\n",datetime);
    fprintf(out,"\n");
    fprintf(out,"PREDICTION RESULTS\n");
    prnt_like(out,N,NX,P,theta,power,gamma,beta,sigmaz,like);
    fprintf(out,"\n");

    get_pred(N,NX,P,kin1,kin2,s,Y,theta,power,gamma,sigmaz,numpred,predx,yhat,emse);
    pred_out(out,numpred,truey_ind,truey,predx,yhat,emse);

    fprintf(out,"\n");
    fprintf(out,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
    if(nfiles==2)fprintf(out," AND %s",data_file2);
    fprintf(out,"\n");

    fclose(out);

  }

/*  This section computes cross-validation results for given parameters */

  scanf("%d",&ndec);

  if(ndec)
  {

    if(read_param)
    {
      scanf("%s",temp_name);
      fname=AllocChar(strlen(temp_name));
      strcpy(fname,temp_name);
      in1=fopen(fname,"r");
      FreeChar(strlen(temp_name),fname);

      fscanf(in1,"%lf %lf %lf",&sigmaz,&gamma,&like);
      for(i=0;i<P;i++) fscanf(in1,"%lf",&beta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&theta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&power[i]);
      read_param=0;
    }

    scanf("%s",temp_name);
    fname=AllocChar(strlen(temp_name));
    strcpy(fname,temp_name);
```

```
    out=fopen(fname,"w");
    FreeChar(strlen(temp_name),fname);

    fputs(title,out);
    fprintf(out,"\n%s\n",datetime);
    fprintf(out,"\n");
    fprintf(out,"CROSS-VALIDATION RESULTS\n");
    prnt_like(out,N,NX,P,theta,power,gamma,beta,sigmaz,like);
    fprintf(out,"\n");

    get_cv(out,N,NX,P,s,Y,theta,power,gamma,sigmaz,kin1,kin2);

    fprintf(out,"\n");
    fprintf(out,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
    if(nfiles==2)fprintf(out," AND %s",data_file2);
    fprintf(out,"\n");

    fclose(out);

  }

/*  This section computes main effects results for selected variables */

  scanf("%d",&ndec);

  if(ndec)
  {

    if(read_param)
    {
      scanf("%s",temp_name);
      fname=AllocChar(strlen(temp_name));
      strcpy(fname,temp_name);
      in1=fopen(fname,"r");
      FreeChar(strlen(temp_name),fname);

      fscanf(in1,"%lf %lf %lf",&sigmaz,&gamma,&like);
      for(i=0;i<P;i++) fscanf(in1,"%lf",&beta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&theta[i]);
      for(i=0;i<NX;i++) fscanf(in1,"%lf",&power[i]);
      read_param=0;
    }

    scanf("%d %d",&intlev,&usesig);

/*   either use significance levels to choose factors to compute */
/*   main effects for or input factors manually                  */
```

```
    if(usesig)
      det_sig(N,NX,P,s,Y,theta,power,gamma,kin1,kin2,int_ind);
    else
      for(i=0;i<NX;i++)scanf("%d",&int_ind[i]);

/*  preparation of output file names                    */

    scanf("%s",temp_name);
    oname=AllocChar(strlen(temp_name));
    strcpy(oname,temp_name);

    fname=AllocChar(strlen(oname)+4);
    strcpy(fname,oname);
    fname=strcat(fname,".med");
    out=fopen(fname,"w");
    FreeChar(strlen(oname)+4,fname);

    fputs(title,out);
    fprintf(out,"\n%s\n",datetime);
    fprintf(out,"\n");
    fprintf(out,"MAIN EFFECTS RESULTS\n\n");

    prnt_like(out,N,NX,P,theta,power,gamma,beta,sigmaz,like);
    fprintf(out,"\n");

    init_me(out,oname,N,NX,P,s,Y,theta,power,gamma,kin1,kin2,int_ind,intlev);

    fprintf(out,"\n");
    fprintf(out,"THE DATA FOR THIS RUN IS IN THE FILE %s",data_file);
    if(nfiles==2)fprintf(out," AND %s",data_file2);
    fprintf(out,"\n");

    fclose(out);

  }

  FreeDouble2(N,N,V);
  FreeDouble2(N,NX,s);
  FreeDouble2(N,P,F);
  FreeDouble(N,Y);
  FreeDouble(NX,theta);
  FreeDouble(NX,power);
  FreeDouble(P,beta);
  FreeDouble(numpred,yhat);
  FreeDouble(numpred,emse);
  FreeInt(P,kin1);
  FreeInt(P,kin2);
```

```
    FreeInt(NX,int_ind);

    return;
}
/*-------------------------------------------------------------*/
```

```
/***************This is rb_like.h*****************************/
extern double getlike();
extern double compmle();
extern double mle();
extern double chpmle();
extern void det_sig();
extern void mult_like();
/*****************************************************************/
/*              This is rb_like.c                                */
/*  These routines compute likelihood values                     */
/*---------------------------------------------------------------*/
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include "rb_alloc.h"
#include "rb_covm.h"
#include "rb_math.h"
#include "rb_opt.h"
#include "rb_dace_prnt.h"
#include "dace.h"
/*---------------------------------------------------------------*/
double getlike(n,p,y,f,v,sigmaz,r,c)
  int n,p;
  double *y,**f,**v,**r,*c,*sigmaz;
{
  int i,j,k;
  double fcn,det,**ftilda,tempf,tempy,*z,*ytilda,ss,*resid,**q;
  ftilda=AllocDouble2(n,p+1);
  ytilda=AllocDouble(n);
  resid=AllocDouble(n);
  q=AllocDouble2(n,p);
  z=AllocDouble(n);

  cd(n,v,z);
  det=0.;
  for(i=0;i<n;i++) det=det-log(z[i]);
  det=2*det;

  for(k=0;k<n;k++)
  {
    for(i=0;i<p;i++)
    {
      tempf=f[k][i];
      for(j=0;j<k;j++) tempf=tempf-v[k][j]*ftilda[j][i];
      ftilda[k][i]=tempf*z[k];
    }
  }
```

```
  for(k=0;k<n;k++)
  {
    tempy=y[k];
    for(j=0;j<k;j++) tempy=tempy-v[k][j]*ytilda[j];
    ytilda[k]=tempy*z[k];
  }

  if(p==0)
  {
    ss=0.;
    for(i=0;i<n;i++) ss=ss+ytilda[i]*ytilda[i];
    *sigmaz=ss/n;

    fcn=n*log(*sigmaz) + det ;
    fcn=fcn/100.;
    return(fcn);
  }

  for(i=0;i<n;i++) ftilda[i][p]=ytilda[i];
  qr(ftilda,n,p,p+1,q,r,ytilda,c);

  for(i=0;i<n;i++)
  {
    resid[i]=0.;
    for(j=0;j<p;j++) resid[i]=resid[i]+q[i][j]*c[j];
    resid[i]=ytilda[i]-resid[i];
  }

  ss=0.;
  for(i=0;i<n;i++) ss=ss+resid[i]*resid[i];
  *sigmaz=ss/n;

  fcn=n*log(*sigmaz) + det;
  fcn=fcn/100.;

  FreeDouble2(n,p+1,ftilda);
  FreeDouble2(n,p,q);
  FreeDouble(n,ytilda);
  FreeDouble(n,resid);
  FreeDouble(n,z);

  return(fcn);
}
/*-----------------------------------------------------------------*/
double compmle(n,p,y,f,v,sigmaz,beta)
  int n,p;
  double *y,**f,**v,*sigmaz,*beta;
```

```
{
  int i,j;
  double **r,*c,fcn;
  c=AllocDouble(n);
  r=AllocDouble2(p,p);

  fcn=getlike(n,p,y,f,v,sigmaz,r,c);
  fcn=fcn*100.;
  for(i=0;i<p;i++)
  {
    beta[i]=c[i];
    for(j=p-1;j>i;j--) beta[i]=beta[i]-r[i][j]*beta[j];
    beta[i]=beta[i]/r[i][i];
  }

  FreeDouble(n,c);
  FreeDouble2(p,p,r);

  return(fcn);
}
/*-------------------------------------------------------------*/
double mle(out,out1,n,nx,p,sdiff,f,y,theta,power,gamma,beta,sigmaz,tol,nloop,itmax)
  int n,nx,p,nloop,itmax;
  double **sdiff,**f,*y,*theta,*power,*gamma,*beta,*sigmaz,tol;
  FILE *out,*out1;
{
  int i,j,k,iter,nopt;
  double like,t_like,*dx,ran_num,tempgam,RAND_MAX,*gtheta,*gpower,ggamma;
  double old_like=10000000;
  long temp_num;

  nopt= (NGAM) ? 2*nx+1 : 2*nx;
  dx=AllocDouble(nopt);
  gtheta=AllocDouble(nx);
  gpower=AllocDouble(nx);

  RAND_MAX=2147483647;
  for(j=0;j<nloop;j++)
  {
    for(i=0;i<nx;i++)
    {
      temp_num=random();
      ran_num=temp_num;
      dx[i]=log(ran_num/RAND_MAX+0.000000001);
      ran_num=random();
      dx[nx+i]=asin(2*(ran_num/RAND_MAX-0.5));
    }
```

```
  ran_num=rand();
  if (NGAM) dx[2*nx]=log(ran_num/RAND_MAX+0.0000000001);

  NFCALLS=0;
  FUNC_DECISION=0;
  like=runopt(dx,nopt,tol,&iter,itmax);

  fprintf(out1,"NFCALLS= %d # ITERS= %d MAX ITERS ALLOWED= %d\n",\
  NFCALLS,iter,itmax);

  for(i=0;i<nx;i++)
  {
    theta[i]=exp(dx[i]);
    power[i]=sin(dx[nx+i])/2+1.5;
  }
  *gamma= (NGAM) ? exp(dx[2*nx]) : 0;

  covm(n,nx,sdiff,theta,power,*gamma,V);
  t_like=compmle(n,p,y,f,V,sigmaz,beta);

  if(like<old_like)
  {
    for(i=0;i<nx;i++)
    {
      gtheta[i]=exp(dx[i]);
      gpower[i]=sin(dx[nx+i])/2+1.5;
    }
    ggamma= (NGAM) ? exp(dx[2*nx]) : 0;
    old_like=t_like;
  }

  prnt_like(out1,n,nx,p,theta,power,*gamma,beta,*sigmaz,t_like);
}

for(i=0;i<nx;i++)
{
  theta[i]=gtheta[i];
  power[i]=gpower[i];
}
*gamma=ggamma;
covm(n,nx,sdiff,theta,power,*gamma,V);
t_like=compmle(n,p,y,f,V,sigmaz,beta);

prnt_like(out,n,nx,p,theta,power,*gamma,beta,*sigmaz,t_like);

FreeDouble(nopt,dx);
FreeDouble(nx,gtheta);
```

```
  FreeDouble(nx,gpower);

  return(old_like);
}
/*-------------------------------------------------------------*/
double chpmle(out,out1,n,nx,p,sdiff,f,y,theta,power,gamma,beta,sigmaz,tol,nloop,itmax)
  int n,nx,p,nloop,itmax;
  double **sdiff,**f,*y,*theta,*power,*gamma,*beta,*sigmaz,tol;
  FILE *out,*out1;
{
  int i,j,k,iter,nopt;
  double like,old_like,t_like,*dx,ran_num,tempgam,RAND_MAX;
  long temp_num;

/*   FILE *out;
  FILE *out1;
  out=fopen(MLE_FILE,"w");
  out1=fopen(DUMP_FILE,"w"); */

  nopt= (NGAM) ? 3 : 2;
  dx=AllocDouble(nopt);

  RAND_MAX=2147483647;
  temp_num=random();
  ran_num=temp_num;
  dx[0]=log(ran_num/RAND_MAX+0.000000001);
  ran_num=random();
  dx[1]=asin(2*(ran_num/RAND_MAX-0.5));
  ran_num=rand();
  if (NGAM) dx[2]=log(ran_num/RAND_MAX+0.0000000001);

  NFCALLS=0;
  FUNC_DECISION=1;
  old_like=runopt(dx,nopt,tol,&iter,itmax);

  for(i=0;i<nx;i++)
  {
    theta[i]=exp(dx[0]);
    power[i]=sin(dx[1])/2+1.5;
  }
  *gamma= (NGAM) ? exp(dx[2]) : 0;

        covm(n,nx,sdiff,theta,power,*gamma,V);
        t_like=compmle(n,p,y,f,V,sigmaz,beta);

  prnt_like(out1,n,nx,p,theta,power,*gamma,beta,*sigmaz,old_like);
  prnt_like(out,n,nx,p,theta,power,*gamma,beta,*sigmaz,old_like);
```

```
    covm1(n,nx,sdiff,theta,power,*gamma,V);

    FUNC_DECISION=2;

    for(j=0;j<nloop;j++)
    {
      for(i=0;i<nx;i++)
      {
        OLD_THETA=theta[i];
        OLD_POWER=power[i];
        dx[0]=log(theta[i]+0.000000001);
        dx[1]=asin(2*(power[i]-1.5));
        if(NGAM) dx[2]=log(*gamma+0.0000000001);
        NPK=i;
        NFCALLS=0;

        like=runopt(dx,nopt,tol,&iter,itmax);

      fprintf(out1,"NFCALLS= %d # ITERS= %d MAX ITERS ALLOWED= %d\n",\
      NFCALLS,iter,itmax);

        if(like<old_like)
        {
          theta[i]=exp(dx[0]);
          power[i]=sin(dx[1])/2+1.5;
          *gamma= (NGAM) ? exp(dx[2]) : 0;
          covm(n,nx,sdiff,theta,power,*gamma,V);
          t_like=compmle(n,p,y,f,V,sigmaz,beta);
          old_like=t_like;
        }
      }

      fprintf(out,"FOR LOOP %d -2*LN LIKELIHOOD= %10.4le\n",j+1,old_like);
      prnt_like(out1,n,nx,p,theta,power,*gamma,beta,*sigmaz,old_like);
    }

    prnt_like(out,n,nx,p,theta,power,*gamma,beta,*sigmaz,old_like);
/*  fclose(out);
    fclose(out1); */

    FreeDouble(nopt,dx);

    return(old_like);
}
/*-------------------------------------------------------------*/
void det_sig(n,nx,p,s,y,theta,power,gamma,kin1,kin2,int_ind)
  int n,nx,p,*kin1,*kin2,*int_ind;
```

```
  double **s,*y,*theta,*power,gamma;
{
  int i,j;
  double tlike,like,sigmaz;
  double **sdiff,**v,*ntheta,**r,*c,fcn,**f;

  sdiff=AllocDouble2(n*(n-1)/2,nx);
  v=AllocDouble2(n,n);
  f=AllocDouble2(n,p);
  r=AllocDouble2(p,p);
  c=AllocDouble(n);
  ntheta=AllocDouble(nx);

  des_dist(n,nx,s,sdiff);
  xtof(s,n,p,kin1,kin2,f);

  covm(n,nx,sdiff,theta,power,gamma,v);
  tlike=getlike(n,p,y,f,v,&sigmaz,r,c);
  tlike=tlike*100.;

  for(i=0;i<nx;i++)
  {
    int_ind[i]=0;
    for(j=0;j<nx;j++) ntheta[j]=theta[j];
    ntheta[i]=0;

    covm(n,nx,sdiff,ntheta,power,gamma,v);
    like=getlike(n,p,y,f,v,&sigmaz,r,c);
    like=like*100.;

    if(like-tlike>6)int_ind[i]=1;
  }

  FreeDouble2(n*(n-1)/2,nx,sdiff);
  FreeDouble2(n,n,v);
  FreeDouble2(n,p,f);
  FreeDouble2(p,p,r);
  FreeDouble(n,c);
  FreeDouble(nx,ntheta);

  return;
}
/*-----------------------------------------------------------------
void mult_like(n,nx,p,kin1,kin2,s,y,theta,power,gamma,nlike)
   int n,nx,p,*kin1,*kin2,nlike;
   double **s,*y,*theta,*power,gamma;
{
```

```
  int i,j;
  double **sdiff,**v,**f,*beta,like,sigmaz;
  sdiff=AllocDouble2(n*(n-1)/2,nx);
  f=AllocDouble2(n,p);
  v=AllocDouble2(n,n);

    xtof(s,n,p,kin1,kin2,f);
    des_dist(n,nx,s,sdiff);

    for(j=0;j<nlike;j++)
    {
      for(i=0;i<nx;i++) fscanf(in,"%lf",&theta[i]);
      for(i=0;i<nx;i++) fscanf(in,"%lf",&power[i]);
      fclose(in);

      covm(n,nx,sdiff,theta,power,gamma,v);
      like=compmle(n,p,y,f,v,&sigmaz,beta);

      fprintf(out,"RESULTS FOR PARAMETER SET %d:\n",j);
      fprintf(out,"-2*LN LIKELIHOOD= %lf\n",like);
      fprintf(out,"SIGMAZ= %lf\n",sigmaz);
      for(i=0;i<P;i++) fprintf(out,"BETA= %lf\n",beta[i]);
    }
  return;
}
/*-----------------------------------------------------------------*/
```

```
/***************This is rb_predict.h*****************************/
extern void initial();
extern void predict();
extern void get_emse();
extern void get_pred();
extern void get_cv();
extern double mepred();
/*************************************************************************/
/*                This is rb_predict.c                                   */
/*  These routines are used for prediction and cross-validation    */
/*-----------------------------------------------------------------*/
#include<math.h>
#include<stdio.h>
#include "rb_math.h"
#include "rb_alloc.h"
#include "rb_matman.h"
#include "rb_covm.h"
/*-----------------------------------------------------------------*/
void initial(n,p,f,v,vec1,vec2,fvf)
  int n,p;
  double **f,**v,**vec1,**vec2,**fvf;
{
  int i,j,k;
  double **a;
  a=AllocDouble2(p,n);

  invmat(n,v);

  for(i=0;i<n;i++)
    for(j=i+1;j<n;j++) v[i][j]=v[j][i];

  for(i=0;i<p;i++)
    for(j=0;j<n;j++)
      for(k=0;k<n;k++)  a[i][j]=a[i][j]+f[k][i]*v[k][j];

  for(i=0;i<p;i++)
    for(j=0;j<p;j++)
      for(k=0;k<n;k++)  fvf[i][j]=fvf[i][j]+a[i][k]*f[k][j];

  if(p==1) fvf[0][0]=1./fvf[0][0];

  if(p!=1) invmat(p,fvf);

  for(i=0;i<p;i++)
    for(j=i+1;j<p;j++)  fvf[i][j]=fvf[j][i];

  for(i=0;i<p;i++)
```

```
    for(j=0;j<n;j++)
    {
      vec1[i][j]=0;
     for(k=0;k<p;k++)
       vec1[i][j]=vec1[i][j]+fvf[i][k]*a[k][j];
    }

  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
    {
     vec2[i][j]=-1.0*v[i][j];
     for(k=0;k<p;k++)  vec2[i][j]=vec2[i][j]+a[k][i]*vec1[k][j];
    }

  FreeDouble2(p,n,a);

  return;
}
/*-------------------------------------------------------------*/
void predict(n,nx,p,s,y,theta,power,numpred,x,fpred,vec1,vec2,yhat)
  int n,nx,p,numpred;
  double **s,*y,*theta,*power,**x,**fpred,**vec1,**vec2,*yhat;
{
  int i,j,k;
  double *r,*v1,*v2;
  r=AllocDouble(n);
  v1=AllocDouble(p);
  v2=AllocDouble(n);

  for(i=0;i<p;i++)
    for(j=0;j<n;j++)  v1[i]=v1[i]+vec1[i][j]*y[j];

  for(i=0;i<n;i++)
    for(j=0;j<n;j++)  v2[i]=v2[i]-vec2[i][j]*y[j];

  for(i=0;i<numpred;i++)
  {
    for(j=0;j<n;j++)
    {
      r[j]=0;
      for(k=0;k<nx;k++)  r[j]=r[j]-theta[k]*pow(rb_abs(s[j][k]-x[i][k]),power[k]);
      r[j]=exp(r[j]);
    }

    yhat[i]=0;
    for(j=0;j<n;j++) yhat[i]=yhat[i]+r[j]*v2[j];
```

```
    for(j=0;j<p;j++) yhat[i]=yhat[i]+fpred[i][j]*v1[j];
  }

  FreeDouble(n,r);
  FreeDouble(p,v1);
  FreeDouble(n,v2);

  return;
}
/*-----------------------------------------------------------------*/
void get_emse(n,nx,p,s,theta,power,sigmaz,numpred,x,fpred,vec1,vec2,fvf,emse)
  int n,nx,p,numpred;
  double **s,*theta,*power,sigmaz,**x,**fpred,**vec1,**vec2,**fvf,*emse;
{
  int i,j,k;
  double *r,d1;
  r=AllocDouble(n);

  for(i=0;i<numpred;i++)
  {
    for(j=0;j<n;j++)
    {
      r[j]=0;
      for(k=0;k<nx;k++)  r[j]=r[j]-theta[k]*pow(rb_abs(s[j][k]-x[i][k]),power[k]);
      r[j]=exp(r[j]);
    }

    emse[i]=1;

    for(j=0;j<p;j++)
    {
      d1=0;
      for(k=0;k<p;k++)  d1=d1+fvf[j][k]*fpred[i][k];
      emse[i]=emse[i]+fpred[i][j]*d1;
    }

    for(j=0;j<p;j++)
    {
      d1=0;
      for(k=0;k<n;k++)  d1=d1+vec1[j][k]*r[k];
      emse[i]=emse[i]-2*fpred[i][j]*d1;
    }

    for(j=0;j<n;j++)
    {
      d1=0;
      for(k=0;k<n;k++)  d1=d1+vec2[j][k]*r[k];
```

```
      emse[i]=emse[i]+r[j]*d1;
    }

    emse[i]=sigmaz*emse[i];
  }

  FreeDouble(n,r);

  return;
}
/*--------------------------------------------------------------------------*/
void get_pred(n,nx,p,kin1,kin2,s,y,theta,power,gamma,sigmaz,numpred,predx,yhat,emse)
  int n,nx,p,*kin1,*kin2,numpred;
  double **s,*y,*theta,*power,gamma,sigmaz,**predx,*yhat,*emse;
{
  int i,j;
  double **f,**sdiff,**v,**fpred,**vec1,**vec2,**fvf;

  f=AllocDouble2(n,p);
  v=AllocDouble2(n,n);
  sdiff=AllocDouble2(n*(n-1)/2,nx);
  fpred=AllocDouble2(numpred,p);
  vec1=AllocDouble2(p,n);
  vec2=AllocDouble2(n,n);
  fvf=AllocDouble2(p,p);

  des_dist(n,nx,s,sdiff);
  xtof(s,n,p,kin1,kin2,f);
  covm(n,nx,sdiff,theta,power,gamma,v);
  xtof(predx,numpred,p,kin1,kin2,fpred);
  initial(n,p,f,v,vec1,vec2,fvf);
  predict(n,nx,p,s,y,theta,power,numpred,predx,fpred,vec1,vec2,yhat);
  get_emse(n,nx,p,s,theta,power,sigmaz,numpred,predx,fpred,vec1,vec2,fvf,emse);

  FreeDouble2(n,p,f);
  FreeDouble2(n,n,v);
  FreeDouble2(n*(n-1)/2,nx,sdiff);
  FreeDouble2(numpred,p,fpred);
  FreeDouble2(p,n,vec1);
  FreeDouble2(n,n,vec2);
  FreeDouble2(p,p,fvf);

  return;
}
/*--------------------------------------------------------------------------*/
void get_cv(out,n,nx,p,s,y,theta,power,gamma,sigmaz,kin1,kin2)
  int n,nx,p,*kin1,*kin2;
```

```
  double **s,*y,*theta,*power,gamma,sigmaz;
  FILE *out;
{
  int i,j,k,m;
  double **sdiff,**v,**subv,**subs,*suby,**f,**predx,*yhat,*emse;
  double **vec1,**vec2,**fvf,**fpred,*cvyhat,*cvmse;

  v=AllocDouble2(n,n);
  subv=AllocDouble2(n-1,n-1);
  sdiff=AllocDouble2(n*(n-1)/2,nx);
  subs=AllocDouble2(n-1,nx);
  suby=AllocDouble(n-1);
  f=AllocDouble2(n-1,p);
  fpred=AllocDouble2(1,p);
  predx=AllocDouble2(1,nx);
  vec1=AllocDouble2(p,n);
  vec2=AllocDouble2(n,n);
  fvf=AllocDouble2(p,p);
  yhat=AllocDouble(1);
  emse=AllocDouble(1);
  cvyhat=AllocDouble(n);
  cvmse=AllocDouble(n);

  des_dist(n,nx,s,sdiff);
  covm(n,nx,sdiff,theta,power,gamma,v);

  for(i=0;i<n;i++)
  {
    m=0;
    for(j=0;j<n;j++)
    {
      if(i==j)
      {
        for(k=0;k<nx;k++) predx[0][k]=s[j][k];
      }
      else
      {
        suby[m]=y[j];
        for(k=0;k<nx;k++) subs[m][k]=s[j][k];
        for(k=0;k<n;k++)
        {
          if(k<i)subv[m][k]=v[j][k];
          if(k>i)subv[m][k-1]=v[j][k];
        }
        m++;
      }
    }
```

```
    xtof(subs,n-1,p,kin1,kin2,f);
    xtof(predx,1,p,kin1,kin2,fpred);
    initial(n-1,p,f,subv,vec1,vec2,fvf);
    predict(n-1,nx,p,subs,suby,theta,power,1,predx,fpred,vec1,vec2,yhat);
    get_emse(n-1,nx,p,subs,theta,power,sigmaz,1,predx,fpred,vec1,vec2,fvf,emse);

    cvyhat[i]=yhat[0];
    cvmse[i]=emse[0];
  }

  pred_out(out,n,1,y,s,cvyhat,cvmse);

  FreeDouble2(n,n,v);
  FreeDouble2(n-1,n-1,subv);
  FreeDouble2(n*(n-1)/2,nx,sdiff);
  FreeDouble2(1,p,fpred);
  FreeDouble2(n-1,nx,subs);
  FreeDouble(n-1,suby);
  FreeDouble2(n-1,p,f);
  FreeDouble2(1,nx,predx);
  FreeDouble2(p,n,vec1);
  FreeDouble2(n,n,vec2);
  FreeDouble2(p,p,fvf);
  FreeDouble(1,yhat);
  FreeDouble(1,emse);
  FreeDouble(n,cvyhat);
  FreeDouble(n,cvmse);

  return;
}
/*-------------------------------------------------------------*/
double mepred(n,nx,p,s,y,theta,power,f,v,kin1,kin2)
  int n,nx,p,*kin1,*kin2;
  double **s,*y,*theta,*power,**f,**v;
{
  int i,j,numpred=1000;
  double **vec1,**vec2,**fvf,**fpred,**predx,*yhat;
  double yavg=0,ssy=0,RAND_MAX=2147483647;

  fpred=AllocDouble2(numpred,p);
  predx=AllocDouble2(numpred,nx);
  vec1=AllocDouble2(p,n);
  vec2=AllocDouble2(n,n);
  fvf=AllocDouble2(p,p);
  yhat=AllocDouble(numpred);

  for(i=0;i<numpred;i++)
```

```
    for(j=0;j<nx;j++) predx[i][j]=(random()/RAND_MAX)-0.5;

  xtof(predx,numpred,p,kin1,kin2,fpred);
  initial(n,p,f,v,vec1,vec2,fvf);
  predict(n,nx,p,s,y,theta,power,numpred,predx,fpred,vec1,vec2,yhat);

  for(i=0;i<numpred;i++) yavg=yavg+yhat[i];
  yavg=yavg/numpred;

  for(i=0;i<numpred;i++) ssy=ssy+(yhat[i]-yavg)*(yhat[i]-yavg);
  ssy=ssy/numpred;

  FreeDouble2(numpred,p,fpred);
  FreeDouble2(p,n,vec1);
  FreeDouble2(n,n,vec2);
  FreeDouble2(p,p,fvf);
  FreeDouble(numpred,yhat);

  return(ssy);
}
/*-------------------------------------------------------------------*/
```

```
/***************This is rb_effects.h*************************/
extern double calmu0();
extern void calmu1();
extern void calmu2();
extern void init_me();
/*********************************************************************/
/*              This is rb_effects.c                                 */
/*  These routines compute main effects and interactions             */
/*-------------------------------------------------------------------*/
#include<stdio.h>
#include<string.h>
#include<math.h>
#include "rb_alloc.h"
#include "rb_covm.h"
#include "rb_like.h"
#include "rb_math.h"
#include "rb_matman.h"
#include "rb_predict.h"
/*-------------------------------------------------------------------*/
double calmu0(n,rinvy,mulexp,numintr,intr,nulexp)
  int n,numintr;
  double *rinvy,*mulexp,**intr,*nulexp;
{
  int i,j;
  double result=0;

  for(i=0;i<n;i++) nulexp[i]=mulexp[i];

  for(i=0;i<numintr;i++)
    for(j=0;j<n;j++) nulexp[j]=nulexp[j]*intr[j][i];

  for(i=0;i<n;i++) result=result+nulexp[i]*rinvy[i];

  return(result);
}
/*-------------------------------------------------------------------*/
void calmu1(out1,fname,n,nx,numpts,int_ind,s,theta,power,ssy,rinvy,numintr,\
intr,nulexp,mu0,mu)
  int n,nx,numpts,*int_ind,numintr;
  double ssy,*nulexp,**intr,**s,*theta,*power,*rinvy,mu0,**mu;
  char *fname;
  FILE *out1;
{
  int i,j,k,*muind;
  int cnt=0;
  double *x,*bulexp,etox,*ss;
```

```
FILE *out;

bulexp=AllocDouble(n);
x=AllocDouble(numpts);
muind=AllocInt(numintr);
ss=AllocDouble(numintr);

x[0]=-0.5;
for(i=1;i<numpts;i++) x[i]=x[i-1]+1./(numpts-1);

for(i=0;i<nx;i++) if(int_ind[i]) muind[cnt++]=i;

for(i=0;i<numintr;i++)
  for(k=0;k<numpts;k++)
  {
    mu[k][i]=0;
    for(j=0;j<n;j++)
    {
      etox=pow(rb_abs(s[j][muind[i]]-x[k]),power[muind[i]]);
      etox=exp(-1.*theta[muind[i]]*etox);
      bulexp[j]=nulexp[j]*etox/intr[j][i];
      mu[k][i]=mu[k][i]+bulexp[j]*rinvy[j];
    }
  }

for(k=0;k<numpts;k++)
  for(i=0;i<numintr;i++) mu[k][i]=mu[k][i]-mu0;

out=fopen(fname,"w");
fprintf(out," %d",numintr);
for(i=0;i<numintr;i++) fprintf(out," %d",muind[i]+1);
  fprintf(out,"\n");
for(k=0;k<numpts;k++)
{
  for(i=0;i<numintr;i++)
  {
    fprintf(out,"%8.3lf",mu[k][i]);
    ss[i]=ss[i]+mu[k][i]*mu[k][i];
  }
  fprintf(out,"\n");
}

for(i=0;i<numintr;i++)
  fprintf(out1,"        Var %2d MSE= %12.4le    MSE/VAR(Y)= %8.4lf\n",muind[i]+1,\
  ss[i]/numpts,(ss[i]/numpts)/ssy);

fclose(out);
```

```
  FreeDouble(n,bulexp);
  FreeDouble(numpts,x);
  FreeInt(numintr,muind);
  FreeDouble(numintr,ss);

  return;
}
/*-----------------------------------------------------------------*/
void calmu2(out2,oname,n,nx,numpts,int_ind,s,theta,power,beta,ssy,rinvy,numintr,\
intr,nulexp,mu0,mu1)
  int n,nx,numpts,*int_ind,numintr;
  double ssy,*nulexp,**intr,**s,*theta,*power,*beta,*rinvy,mu0,**mu1;
  char *oname;
  FILE *out2;
{
  int i,j,k,l,m,*muind;
  int cnt=0,cntint=0;
  double *x,*bulexp,temp,etox,etoy,**mu,*ss;
  char *fname,*fname1;

  FILE *out;
  FILE *out1;

  fname=AllocChar(strlen(oname)+4);
  strcpy(fname,oname);
  fname=strcat(fname,".int");

  fname1=AllocChar(strlen(oname)+3);
  strcpy(fname1,oname);
  fname1=strcat(fname1,".sf");

  out=fopen(fname,"w");
  out1=fopen(fname1,"w");

  bulexp=AllocDouble(n);
  x=AllocDouble(numpts);
  mu=AllocDouble2(numpts,numpts);
  muind=AllocInt(numintr);

  x[0]=-0.5;
  for(i=1;i<numpts;i++) x[i]=x[i-1]+1./(numpts-1);

  for(i=0;i<nx;i++) if(int_ind[i]) muind[cnt++]=i;

  ss=AllocDouble(numintr*(numintr-1)/2);

  for(i=0;i<numintr;i++)
```

```
for(m=i+1;m<numintr;m++)
{
 for(k=0;k<numpts;k++)
  for(l=0;l<numpts;l++)
  {
    mu[k][l]=0;
    for(j=0;j<n;j++)
    {
      temp=pow(rb_abs(s[j][muind[i]]-x[k]),power[muind[i]]);
      etox=exp(-1.*theta[muind[i]]*temp);
      temp=pow(rb_abs(s[j][muind[m]]-x[l]),power[muind[m]]);
      etoy=exp(-1.*theta[muind[m]]*temp);
      bulexp[j]=nulexp[j]*etox*etoy/(intr[j][i]*intr[j][m]);
      mu[k][l]=mu[k][l]+bulexp[j]*rinvy[j];
    }
  }

 for(k=0;k<numpts;k++)
 {
   for(j=0;j<numpts;j++) fprintf(out1," %8.3lf",mu[k][j]+beta[0]);
   fprintf(out1,"\n");
 }

 for(k=0;k<numpts;k++)
   for(j=0;j<numpts;j++)
   {
      mu[k][j]=mu[k][j]-mu1[k][i]-mu1[j][m]-mu0;
      ss[cntint]=ss[cntint]+mu[k][j]*mu[k][j];
   }

 ss[cntint]=ss[cntint]/(numpts*numpts);
 fprintf(out2,"Var %2d Var %2d MSE= %12.4le     MSE/VAR(Y)= %8.4lf\n",\
 muind[i]+1,muind[m]+1,ss[cntint],ss[cntint]/ssy);
 cntint++;

 for(k=0;k<numpts;k++)
 {
   for(j=0;j<numpts;j++) fprintf(out," %8.3lf",mu[k][j]);
   fprintf(out,"\n");
 }

}

fclose(out);
fclose(out1);

FreeChar(strlen(oname)+3,fname1);
```

```
  FreeChar(strlen(oname)+4,fname);
  FreeDouble(n,bulexp);
  FreeDouble(numpts,x);
  FreeDouble(numintr*(numintr-1)/2,ss);
  FreeDouble2(numpts,numpts,mu);
  FreeInt(numintr,muind);

  return;
}
/*-----------------------------------------------------------------*/
void init_me(out2,oname,n,nx,p,s,y,theta,power,gamma,kin1,kin2,int_ind,intlev)
  int n,nx,p,*kin1,*kin2,intlev,*int_ind;
  double **s,*y,*theta,*power,gamma;
  char *oname;
  FILE *out2;
{
  int i,j,k;
  int numintr=0,numpts=21;
  double **f,**sdiff,**v,*beta,like,sigmaz;
  double *rinvy,*mulexp,*nulexp,**intr,mu0,**mu1,ssy;
  char *fname,*fname1;

  FILE *out;

  beta=AllocDouble(p);
  rinvy=AllocDouble(n);
  mulexp=AllocDouble(n);
  nulexp=AllocDouble(n);
  intr=AllocDouble2(n,nx);
  sdiff=AllocDouble2(n*(n-1)/2,nx);
  v=AllocDouble2(n,n);
  f=AllocDouble2(n,p);

  des_dist(n,nx,s,sdiff);
  xtof(s,n,p,kin1,kin2,f);
  covm(n,nx,sdiff,theta,power,gamma,v);
  like=compmle(n,p,y,f,v,&sigmaz,beta);

  covm(n,nx,sdiff,theta,power,gamma,v);

  ssy=mepred(n,nx,p,s,y,theta,power,f,v,kin1,kin2);
  fprintf(out2,"  Variance of Y= %12.5le\n\n",ssy);

  get_rinvy(n,p,f,v,y,beta,rinvy);

  for(i=0;i<n;i++) mulexp[i]=1;
```

```
for(i=0;i<nx;i++)
{
  if(int_ind[i])
  {
    for(j=0;j<n;j++) intr[j][numintr]= normin(theta[i],power[i],s[j][i]);
    numintr++;
  }
  else
  {
    for(j=0;j<n;j++) mulexp[j]=mulexp[j]*normin(theta[i],power[i],s[j][i]);
  }
}

mu0=calmu0(n,rinvy,mulexp,numintr,intr,nulexp);

mu1=AllocDouble2(numpts,numintr);

fname=AllocChar(strlen(oname)+4);
strcpy(fname,oname);
fname=strcat(fname,".mu0");
out=fopen(fname,"w");

fprintf(out,"%12.4le\n",mu0);
for(i=0;i<p;i++) fprintf(out," %12.4le",beta[i]);
fprintf(out,"\n");
for(i=0;i<p;i++) fprintf(out," %d",kin1[i]);
fprintf(out,"\n");
for(i=0;i<p;i++) fprintf(out," %d",kin2[i]);
fprintf(out,"\n");
fclose(out);

FreeChar(strlen(oname)+4,fname);

if(intlev<=2)
{
  fname=AllocChar(strlen(oname)+3);
  strcpy(fname,oname);
  fname=strcat(fname,".me");

  calmu1(out2,fname,n,nx,numpts,int_ind,s,theta,power,ssy,rinvy,numintr,\
  intr,nulexp,mu0,mu1);

  FreeChar(strlen(oname)+3,fname);
}

if(intlev==2)
  calmu2(out2,oname,n,nx,numpts,int_ind,s,theta,power,beta,ssy,rinvy,numintr,\
```

```
    intr,nulexp,mu0,mu1);

  FreeChar(strlen(oname)+3,fname1);
  FreeDouble(n,rinvy);
  FreeDouble(n,mulexp);
  FreeDouble(n,nulexp);
  FreeDouble2(n,nx,intr);
  FreeDouble2(n*(n-1)/2,nx,sdiff);
  FreeDouble2(n,n,v);
  FreeDouble2(n,p,f);
  FreeDouble2(numpts,numintr,mu1);

  return;
}
/*-------------------------------------------------------------------------*/
```

```
/***************This is rb_covm.h*****************************/
extern void covm();
extern void covm1();
extern void chp_covm();
extern void slc_covm();
/*************************************************************/
/*              This is rb_covm.c                            */
/* These routines compute covariance matrices                */
/* from the n*(n-1)/2 by nx matrix **sdiff                   */
/*-----------------------------------------------------------*/
#include<math.h>
#include<stdio.h>
#include "rb_math.h"
#include "rb_alloc.h"
#include "rb_matman.h"
/*-----------------------------------------------------------*/
void covm(n,nx,sdiff,theta,power,gamma,v)
  int n,nx;
  double **sdiff,*theta,*power,gamma,**v;
{
  int i,j,k;
  double *vdiff;
  vdiff=AllocDouble(n*(n-1)/2);

  for(i=0;i<n*(n-1)/2;i++)
  {
    for(k=0;k<nx;k++)
      vdiff[i]=vdiff[i]-theta[k]*pow(sdiff[i][k],power[k]);
    vdiff[i]=exp(vdiff[i]);
  }

  mat_v(n,vdiff,v);
  for(i=0;i<n;i++) v[i][i]=1.0+gamma;

  FreeDouble(n*(n-1)/2,vdiff);

  return;
}
/*-----------------------------------------------------------*/
void covm1(n,nx,sdiff,theta,power,gamma,v)
  int n,nx;
  double **sdiff,*theta,*power,gamma,**v;
{
  int i,j,k;
  double *vdiff;
  vdiff=AllocDouble(n*(n-1)/2);
```

```
  for(i=0;i<n*(n-1)/2;i++)
  {
    for(k=0;k<nx;k++)
      vdiff[i]=vdiff[i]-pow(sdiff[i][k],power[0]);
    vdiff[i]=exp(theta[0]*vdiff[i]);
  }

  mat_v(n,vdiff,v);
  for(i=0;i<n;i++) v[i][i]=1.0+gamma;

  FreeDouble(n*(n-1)/2,vdiff);

  return;
}
/*-----------------------------------------------------------------*/
void chp_covm(npk,n,sdiff,theta,power,gamma,ngam,old_theta,old_power,v)
  int n,npk,ngam;
  double **sdiff,*theta,*power,gamma,old_theta,old_power,**v;
{
  int i,j,k;
  double *vdiff;
  vdiff=AllocDouble(n*(n-1)/2);

  vec_v(n,v,vdiff);

  for(i=0;i<n*(n-1)/2;i++)
  vdiff[i]=vdiff[i]*exp(old_theta*pow(sdiff[i][npk],old_power)-\
  theta[0]*pow(sdiff[i][npk],power[0]));

  mat_v(n,vdiff,v);
  if(ngam=1)
    for(i=0;i<n;i++) v[i][i]=v[i][i]+gamma;

  FreeDouble(n*(n-1)/2,vdiff);

  return;
}
/*-----------------------------------------------------------------*/
void slc_covm(npk,n,sdiff,theta,old_theta,old_power,v)
  int n,npk;
  double **sdiff,*theta,old_power,old_theta,**v;
{
  int i,j,k;
  double *vdiff;
  vdiff=AllocDouble(n*(n-1)/2);

  vec_v(n,v,vdiff);
```

```
    for(i=0;i<n*(n-1)/2;i++)
      vdiff[i]=vdiff[i]*exp((old_theta-theta[0])*pow(sdiff[i][npk],old_power));

    mat_v(n,vdiff,v);

    FreeDouble(n*(n-1)/2,vdiff);

    return;
}
/*-------------------------------------------------------------*/
```

```
/***************This is rb_opt.h*****************************/
extern double funk();
extern int amoeba();
extern double runopt();
/*********************************************************************/
/*                  This is rb_opt.c                                 */
/*These routines are used for the optimization algorithms AMOEBA*/
/*-------------------------------------------------------------*/
#include<math.h>
#include "rb_alloc.h"
#include "rb_math.h"
#include "rb_like.h"
#include "dace.h"
/*-------------------------------------------------------------*/
double funk(dx)
  double *dx;
{
  int i,j,k;
  double **r,*c,*theta,*power,gamma,fcn,sigmaz;
  double **tv;

  tv=AllocDouble2(N,N);
  r=AllocDouble2(P,P);
  c=AllocDouble(N);

  NFCALLS++;

  if(FUNC_DECISION==0)
  {
/*   MLE - OPT 1 THETA,POWER FOR EACH VARIABLE  */

    theta=AllocDouble(NX);
    power=AllocDouble(NX);

    for(i=0;i<NX;i++)
    {
      theta[i]=exp(dx[i]);
      power[i]=sin(dx[NX+i])/2+1.5;
    }
      gamma= (NGAM) ? exp(dx[2*NX]) : 0;
      covm(N,NX,SDIFF,theta,power,gamma,tv);

    FreeDouble(NX,theta);
    FreeDouble(NX,power);
  }
  else if(FUNC_DECISION==1)
  {
```

```
/*   CHPMLE - OPT 1 THETA,POWER FOR ALL VARIABLES  */

    theta=AllocDouble(1);
    power=AllocDouble(1);

    theta[0]=exp(dx[0]);
    power[0]=sin(dx[1])/2+1.5;
    gamma= (NGAM) ? exp(dx[2]) : 0;
    covm1(N,NX,SDIFF,theta,power,gamma,tv);

    FreeDouble(1,theta);
    FreeDouble(1,power);

  }
  else
  {
    for(i=0;i<N;i++)
      for(j=i;j<N;j++)
      {
        tv[i][j]=V[i][j];
        tv[j][i]=tv[i][j];
      }
    switch(FUNC_DECISION)
    {
    case 2 :
    {
/*    CHPMLE - OPT 1 THETA,POWER AT TIME FOR EACH VARIABLE */

      theta=AllocDouble(1);
      power=AllocDouble(1);

      theta[0]=exp(dx[0]);
      power[0]=sin(dx[1])/2+1.5;
      gamma= (NGAM) ? exp(dx[2]) : 0;
      chp_covm(NPK,N,SDIFF,theta,power,gamma,NGAM,OLD_THETA,OLD_POWER,tv);

      FreeDouble(1,theta);
      FreeDouble(1,power);
      break;
    }
    case 3 :
    {
/*    OPT OVER GAMMA ONLY  */

      gamma=exp(dx[0]);
      tv[i][i]=tv[i][i]-OLD_GAMMA+gamma;
      break;
```

```
      }
      case 4 :
      {
/*      SLICE - OPT 1 THETA AT TIME POWER FIXED */

        theta=AllocDouble(1);

        theta[0]=exp(dx[0]);
        slc_covm(NPK,N,SDIFF,theta,OLD_THETA,OLD_POWER,tv);

        FreeDouble(1,theta);
        break;
      }
      }
    }

    fcn=getlike(N,P,Y,F,tv,&sigmaz,r,c);
    fcn=fcn*100;

    FreeDouble2(N,N,tv);
    FreeDouble2(P,P,r);
    FreeDouble(N,c);

    return(fcn);
}
/*-----------------------------------------------------------------*/
int amoeba(xi,fxi,nopt,ftol,itmax)
    int nopt,itmax;
    double *fxi,**xi,ftol;
{
    int i,j,k,ihi,inhi;
    int iter=0,ilo=1;
    double *pr,*prr,*pbar,rtol,ypr,yprr;
    double alpha=1.0,beta=0.5,gamma=2.0;

    pr=AllocDouble(nopt);
    prr=AllocDouble(nopt);
    pbar=AllocDouble(nopt);

    ihi= (fxi[0]>fxi[1]) ? 1 : 2;
    inhi= (fxi[0]<fxi[1]) ? 1 : 2;

    for(i=0;i<nopt+1;i++)
    {
      if(fxi[i]<fxi[ilo])ilo=i;
      if(fxi[i]>fxi[ihi])
      {
```

```
    inhi=ihi;
    ihi=i;
  }
  else if(fxi[i]>fxi[inhi]) if(i!=ihi)inhi=i;
}
rtol=2*rb_abs(fxi[ihi]-fxi[ilo])/(rb_abs(fxi[ihi])+rb_abs(fxi[ilo]));

while(rtol>ftol && iter<itmax)
{
 iter++;

 for(j=0;j<nopt;j++) pbar[j]=0;
 for(i=0;i<nopt+1;i++)
   if(i!=ihi) for(j=0;j<nopt;j++) pbar[j]=pbar[j]+xi[i][j];

 for(j=0;j<nopt;j++)
 {
   pbar[j]=pbar[j]/nopt;
   pr[j]=(1+alpha)*pbar[j]-alpha*xi[ihi][j];
 }
 ypr=funk(pr);

 if(ypr<=fxi[ilo])
 {
   for(j=0;j<nopt;j++)
     prr[j]=gamma*pr[j]+(1-gamma)*pbar[j];
   yprr=funk(prr);

   if(yprr<fxi[ilo])
   {
     for(j=0;j<nopt;j++) xi[ihi][j]=prr[j];
     fxi[ihi]=yprr;
   }
   else
   {
     for(j=0;j<nopt;j++) xi[ihi][j]=pr[j];
     fxi[ihi]=ypr;
   }
 }
 else if(ypr>=fxi[inhi])
 {

   if(ypr<fxi[ihi])
   {
     for(j=0;j<nopt;j++) xi[ihi][j]=pr[j];
     fxi[ihi]=ypr;
   }
```

```
  for(j=0;j<nopt;j++) prr[j]=beta*xi[ihi][j]+(1-beta)*pbar[j];
  yprr=funk(prr);
  if(yprr<fxi[ihi])
  {
    for(j=0;j<nopt;j++) xi[ihi][j]=prr[j];
    fxi[ihi]=yprr;
  }

  else
  {
    for(i=0;i<nopt+1;i++)
    {
      if(i!=ilo)
      {
        for(j=0;j<nopt;j++)
        {
          pr[j]=0.5*(xi[i][j]+xi[ilo][j]);
          xi[i][j]=pr[j];
        }
        fxi[i]=funk(pr);
      }
    }
  }
}
else
{
  for(j=0;j<nopt;j++) xi[ihi][j]=pr[j];
  fxi[ihi]=ypr;
}

ilo=1;

ihi= (fxi[0]>fxi[1]) ? 1 : 2;
inhi= (fxi[0]<fxi[1]) ? 1 : 2;

for(i=0;i<nopt+1;i++)
{
  if(fxi[i]<fxi[ilo])ilo=i;
  if(fxi[i]>fxi[ihi])
  {
    inhi=ihi;
    ihi=i;
  }
  else if(fxi[i]>fxi[inhi])
    if(i!=ihi)inhi=i;
}
rtol=2*rb_abs(fxi[ihi]-fxi[ilo])/(rb_abs(fxi[ihi])+rb_abs(fxi[ilo]));
```

```
  }

  FreeDouble(nopt,pr);
  FreeDouble(nopt,prr);
  FreeDouble(nopt,pbar);

  return(iter);
}
/*-----------------------------------------------------------------*/
double runopt(dx,nopt,tol,iter,itmax)
  int nopt,*iter,itmax;
  double *dx,tol;
{
  int i,j,k,nbest=0;
  double **xi,*ptxi,*fxi,fcn;

  xi=AllocDouble2(nopt+1,nopt);
  ptxi=AllocDouble(nopt);
  fxi=AllocDouble(nopt+1);

  for(j=0;j<nopt;j++) xi[0][j]=dx[j];
  for(k=0;k<nopt;k++)
    for(j=0;j<nopt;j++)  xi[k+1][j]= (k==j) ? dx[j]+1 : dx[j];

  for(k=0;k<nopt+1;k++)
  {
    for(j=0;j<nopt;j++) ptxi[j]=xi[k][j];
    fxi[k]=funk(ptxi);
  }

  *iter=amoeba(xi,fxi,nopt,tol,itmax);

  /*  write output results were here */

  fcn=fxi[0];
  for(k=1;k<nopt+1;k++)
    if(fcn>fxi[k])
    {
      nbest=k;
      fcn=fxi[k];
    }

  for(i=0;i<nopt;i++) dx[i]=xi[nbest][i];

  FreeDouble2(nopt+1,nopt,xi);
  FreeDouble(nopt,ptxi);
  FreeDouble(nopt+1,fxi);
```

```
    return(fcn);
}
/*-------------------------------------------------------------------*/
```

```
/**********************This is rb_print.h**********************/
/*                                                             */
extern void prnt_like();
extern void pred_out();
/*************************************************************/
/*                    This is rb_print.c                       */
/* These routine print output for the different tasks       */
/*************************************************************/
#include<stdio.h>
#include<math.h>
#include "rb_alloc.h"
/*-------------------------------------------------------------*/
/*  prnt_like prints parameter values and likelihood values  */
/*-------------------------------------------------------------*/
void prnt_like(out,n,nx,p,theta,power,gamma,beta,sigmaz,like)
  int n,nx,p;
  double *theta,*power,gamma,*beta,sigmaz,like;
/*  char file_name[40];*/
  FILE *out;
{
  int i;

/*  FILE *out;
  out=fopen(file_name,"a");*/

  fprintf(out,"\n");
  fprintf(out,"N= %d NX= %d\n",n,nx);
  fprintf(out,"SIGMAZ= %10.4le GAMMA= %8.4lf -2*LN LIKELIHOOD= %10.4le\n",\
  sigmaz,gamma,like);
  fprintf(out,"BETA=");
  for(i=0;i<p;i++)
  {
    fprintf(out," %10.4le",beta[i]);
    if((i+1)%5==0)fprintf(out,"\n");
  }
  if(p%5) fprintf(out,"\nTHETA= ");
  if(p%5==0) fprintf(out,"THETA= ");
  for(i=0;i<nx;i++)
  {
    fprintf(out," %10.4le",theta[i]);
    if((i+1)%5==0 && i+1<nx)fprintf(out,"\nTHETA= ");
  }
  if(p%5) fprintf(out,"\nPOWER= ");
  if(p%5==0) fprintf(out,"POWER= ");
  for(i=0;i<nx;i++)
  {
    fprintf(out," %10.4le",power[i]);
```

```
    if((i+1)%5==0 && i+1<nx)fprintf(out,"\nPOWER= ");
  }
  fprintf(out,"\n");
/*  fclose(out); */

}
/*-------------------------------------------------------------------------*/
/*   pred_out calculates summary statistics and prints prediction results   */
/*-------------------------------------------------------------------------*/
void pred_out(out,numpred,truey_ind,truey,predx,yhat,emse)
   int numpred,truey_ind;
   double *truey,**predx,*yhat,*emse;
   FILE *out;
{
   int i,j;
   double *diff,err=0,ss=0,diffmax=-1;
   diff=AllocDouble(numpred);

   if(truey_ind)
   {
     for(i=0;i<numpred;i++)
     {
       diff[i]=(truey[i]-yhat[i])*(truey[i]-yhat[i]);
       if(diff[i]>diffmax)  diffmax=diff[i];
       err=err+diff[i];
     }

     err=err/numpred;

     for(i=0;i<numpred;i++)  ss=ss+(diff[i]-err)*(diff[i]-err);

     ss=ss/numpred;

     fprintf(out,"  ERMSE      STD.ERR(ERMSE)     MAX|ERR|\n");
     fprintf(out,"%12.4le %12.4le    %12.4le\n\n",sqrt(err),sqrt(ss),\
     sqrt(diffmax));

     fprintf(out,"CASE          Y          YHAT        RMSE(YHAT)\n");
     for(i=0;i<numpred;i++)
     fprintf(out,"%4d %12.4le %12.4le %12.4le\n",i+1,truey[i],yhat[i],\
     sqrt(emse[i]));
   }
   else
   {
     fprintf(out,"CASE          YHAT        RMSE(YHAT)\n");
     for(i=0;i<numpred;i++)
     fprintf(out,"%4d  %12.4le %12.4le\n",i,yhat[i],sqrt(emse[i]));
```

```
    }

    FreeDouble(numpred,diff);

    return;
}
/*-------------------------------------------------------------*/
```

```
/***************This is rb_math.h*****************************/
extern void cd();
extern void invmat();
extern double rb_abs();
extern void qr();
extern double normin();
extern void xtof();
extern void get_rinvy();
/*****************************************************************/
/*              This is rb_math.c                                */
/*  these are various math routines                              */
/*---------------------------------------------------------------*/
#include<math.h>
#include "rb_alloc.h"
/*---------------------------------------------------------------*/
void cd(n,v,z)
   int n;
   double **v,*z;
{
   int i,j,k;
   double *vd;
   vd=AllocDouble(n);

   for(i=0;i<n;i++) vd[i]=v[i][i];

   for(i=0;i<n;i++)
   {
     z[i]=1./sqrt(v[i][i]);
     for(j=i+1;j<n;j++) v[j][i]=v[j][i]*z[i];

     for(j=i+1;j<n;j++)
        for(k=j;k<n;k++) v[k][j]=v[k][j]-v[j][i]*v[k][i];
   }

   for(i=0;i<n;i++) v[i][i]=vd[i];
   FreeDouble(n,vd);
   return;
}
/*---------------------------------------------------------------*/
void invmat(n,v)
    int n;
    double **v;
{
    int i,j,k,temp;
    double *z,*w,sum;
    w=AllocDouble(n);
    z=AllocDouble(n);
```

```
    cd(n,v,z);

    for(i=0;i<n;i++) v[i][i]=1./z[i];

    for(i=n-1;i>=0;i--)
    {
      v[i][i]=z[i];
      for(k=i;k>=0;k--)
      {
        sum=0;
        for(j=k+1;j<=i;j++)
        {
          sum=sum+v[j][i]*v[j][k];
          v[k][i]=(-sum*z[k]);
        }
      }
    }

    for(i=0;i<n;i++)
    {
      w[i]=0;
      for(j=i;j<n;j++) w[i]=w[i]+v[i][j]*v[i][j];
      for(j=i+1;j<n;j++)
      {
        v[j][i]=0;
        for(k=j;k<n;k++) v[j][i]=v[j][i]+v[i][k]*v[j][k];
      }
    }

    for(i=0;i<n;i++) v[i][i]=w[i];

    FreeDouble(n,z);
    FreeDouble(n,w);
    return;
}
/*-------------------------------------------------------------*/
double rb_abs(x)
  double x;
{
  if(x<0) x=-1*x;
  return(x);
}
/*-------------------------------------------------------*/
void qr(x,n,p,ncol,q,r,y,c)
  int n,p,ncol;
  double **x,*y,**q,**r,*c;
{
```

```
    int i,j,k;

    for(j=0;j<p;j++)
    {
      r[j][j]=0.;
      for(i=0;i<n;i++) r[j][j]=r[j][j]+x[i][j]*x[i][j];
      r[j][j]=sqrt(r[j][j]);

      for(i=0;i<n;i++) q[i][j]=x[i][j]/r[j][j];

      for(k=j+1;k<p;k++) r[j][k]=0.;
      for(i=0;i<n;i++)
        for(k=j+1;k<p;k++) r[j][k]=r[j][k]+x[i][k]*q[i][j];

      c[j]=0;
      for(i=0;i<n;i++) c[j]=c[j]+y[i]*q[i][j];

      for(i=0;i<n;i++)
        for(k=j+1;k<ncol;k++) x[i][k]=x[i][k]-q[i][j]*r[j][k];
    }

    return;
}
/*-----------------------------------------------------------------*/
double normin(theta,power,x)
  double theta,power,x;
{
  int i;
  double result;
  double u=-0.5,du=0.0001,sum=0;

  for(i=0;i<=10000;i++)
  {
    sum=sum+exp(-1.*theta*pow(rb_abs(x-u),power));
    u=u+du;
  }

  result=sum/10001;

  return(result);
}
/*-----------------------------------------------------------------*/
void xtof(s,n,p,kin1,kin2,f)
    int n,p,*kin1,*kin2;
    double **s,**f;
{
    int i,j;
```

```
  double ss;


  for(i=0;i<p;i++)
  {
      if(kin2[i]==0)

        if(kin1[i]==0)
         for(j=0;j<n;j++) f[j][i]=1;
        else
         for(j=0;j<n;j++) f[j][i]=s[j][kin1[i]-1];

      else
      {
       if(kin2[i]==kin1[i])
       {
        ss=0;
        for(j=0;j<n;j++) ss=ss+s[j][kin2[i]-1]*s[j][kin1[i]-1];
        for(j=0;j<n;j++) f[j][i]=1-n*s[j][kin1[i]-1]*s[j][kin2[i]-1]/ss;
       }
       else
         for(j=0;j<n;j++) f[j][i]=s[j][kin1[i]-1]*s[j][kin2[i]-1];
      }
  }

  return;
}
/*-------------------------------------------------------------*/
void get_rinvy(n,p,f,v,y,beta,rinvy)
 int n,p;
 double **f,**v,*y,*beta,*rinvy;
{
 int i,j;
 double yhat,*resid;
 resid=AllocDouble(n);

 for(i=0;i<n;i++)
 {
  yhat=0;
  for(j=0;j<p;j++) yhat=yhat+f[i][j]*beta[j];
  resid[i]=y[i]-yhat;
 }

 for(i=0;i<n;i++)
 {
  rinvy[i]=0;
  for(j=0;j<n;j++)  rinvy[i]=rinvy[i]+v[i][j]*resid[j];
```

```
  }

  FreeDouble(n,resid);

  return;
}
/*-------------------------------------------------------------------*/
```

```
/***************This is rb_matman.h************************/
extern void des_dist();
extern void vec_v();
extern void mat_v();
/*************************************************************/
/*              This is rb_matman.c                          */
/*  These routines perform various matrix manipulations    */
/*-----------------------------------------------------------*/
#include "rb_math.h"
/*-----------------------------------------------------------*/
void des_dist(n,nx,s,sdiff)
  int n,nx;
  double **s,**sdiff;
{
  int i,j,k,m=0;

  for(i=0;i<nx;i++)
  {
    m=0;
    for(j=0;j<n;j++)
      for(k=j+1;k<n;k++)
        sdiff[m++][i]=rb_abs(s[j][i]-s[k][i]);
  }

  return;
}
/*-----------------------------------------------------------*/
void vec_v(n,mat,vec)
  int n;
  double **mat,*vec;
{
  int i,j,m=0;

  for(i=0;i<n;i++)
    for(j=i+1;j<n;j++)
      vec[m++]=mat[i][j];

  return;
}
/*-----------------------------------------------------------*/
void mat_v(n,vec,mat)
  int n;
  double **mat,*vec;
{
  int i,j,m=0;

  for(i=0;i<n;i++)
```

```
    for(j=i+1;j<n;j++)
    {
      mat[i][j]=vec[m++];
      mat[j][i]=mat[i][j];
    }

  return;
}
/*-------------------------------------------------------------*/
```

```
/****************** This is rb_alloc.h ************************/
extern char **AllocChar2();
extern char *AllocChar();
extern void FreeChar();
extern long *AllocLong();
extern void FreeLong();
extern int *AllocInt();
extern int **AllocInt2();
extern void FreeInt();
extern void FreeInt2();
extern float *AllocFloat();
extern float **AllocFloat2();
extern void FreeFloat();
extern void FreeFloat2();
extern double *AllocDouble();
extern double **AllocDouble2();
extern void FreeDouble();
extern void FreeDouble2();
/*********************************************************************/
/*              This is rb_alloc.c                                   */
/* Routines used for allocating and freeing memory in DACE code*/
/*-------------------------------------------------------------*/
#include<malloc.h>
char **AllocChar2(n)
    int n;
{
    char **B;
    B = ( char **) calloc(n,sizeof(char *));
    return B;
}
/*-------------------------------------------------------------*/
char *AllocChar(n)
    int n;
{
    char *B;
    B = ( char *) calloc(n,sizeof(char));
    return B;
}
/*-------------------------------------------------------------*/
void FreeChar(n,B)
    int n;
    char *B;
{
    cfree(B, n, sizeof(char));
    return;
}
/*-------------------------------------------------------------*/
```

```
long *AllocLong(n)
    int n;
{
    long *B;
    B = ( long *) calloc(n,sizeof(long));
    return B;
}
/*-----------------------------------------------------------------*/
void FreeLong(n,A)
     int n;
     long *A;
{
     cfree(A, n, sizeof(long));
}
/*-----------------------------------------------------------------*/
int *AllocInt(n)
    int n;
{
    int *B;
    B = ( int *) calloc(n,sizeof(int));
    return B;
}
/*-----------------------------------------------------------------*/
int **AllocInt2( n, p)
    int n,p;
{
    int i;
    int **A;
    A = (int **) calloc(n , sizeof(int *));
    for(i=0;i<n;i++) A[i]=AllocInt(p);
    return A;
}
/*-----------------------------------------------------------------*/
void FreeInt(n,A)
     int n;
     int *A;
{
     cfree(A, n, sizeof(int));
}
/*-----------------------------------------------------------------*/
void FreeInt2( n, p , A)
     int n,p;
     int **A;
{
     int i;
     for(i=(n-1);i>=0;i--) cfree(A[i], p, sizeof(int) );
     cfree(A, n , sizeof(int *));
```

```
}
/*-------------------------------------------------------------*/
float *AllocFloat(n)
    int n;
{
    float *B;
    B = ( float *) calloc(n,sizeof(float));
    return B;
}
/*-------------------------------------------------------------*/
float **AllocFloat2( n, p)
    int n,p;
{
    int i;
    float **A;
    A = (float **) calloc(n , sizeof(float *));
    for(i=0;i<n;i++) A[i]=AllocFloat(p);
    return A;
}
/*-------------------------------------------------------------*/
void FreeFloat(n,A)
     int n;
     float *A;
{
     cfree(A, n, sizeof(float));
}
/*-------------------------------------------------------------*/
void FreeFloat2( n, p , A)
     int n,p;
     float **A;
{
     int i;
     for(i=(n-1);i>=0;i--) cfree(A[i], p, sizeof(float) );
     cfree(A, n , sizeof(float *));
}
/*-------------------------------------------------------------*/
double *AllocDouble(n)
    int n;
{
    double *B;
    B = ( double *) calloc(n,sizeof(double));
    return B;
}
/*-------------------------------------------------------------*/
double **AllocDouble2( n, p)
    int n,p;
{
```

```
    int i;
    double **A;
    A = (double **) calloc(n , sizeof(double *));
    for(i=0;i<n;i++) A[i]=AllocDouble(p);
    return A;
}
/*---------------------------------------------------------------*/
void FreeDouble(n,A)
     int n;
     double *A;
{
     cfree(A, n, sizeof(double));
}
/*---------------------------------------------------------------*/
void FreeDouble2( n, p , A)
     int n,p;
     double **A;
{
     int i;
     for(i=(n-1);i>=0;i--) cfree(A[i], p, sizeof(double) );
     cfree(A, n , sizeof(double *));
}
/*---------------------------------------------------------------*/
```

Bibliography

1. Adams, B. M. and Woodall, W. H., (1989) "An Analysis of Taguchi's On-Line Process Control Procedure Under a Random Walk Model," *Technometrics*, 31, 401-414.
2. Aitkin, M., (1987) "Modeling Variance Heterogeneity in Normal Regression Using GLIM," *Applied Statistics*, 36, 332-339.
3. Alvarez, A. R., Abdi, B. L., Young, D. L., Weed, H. D., Teplik, J., and Herald, E. R., (1988) "Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design," *IEEE Transactions on Computer-Aided Design*, 7, 272-288.
4. Andronikou, A. M., Bekey, G. A., and Masri, S. F., (1982) "Identification of Nonlinear Hysteretic Systems using Random Search," *Proceedings 6th IFAC Symposium on Identification and System Parameter Estimation*, 1, 263-268.
5. Atkinson, A. C., (1982) "Developments in the Design of Experiments," *International Statistical Review*, 50, 161-177.
6. Bailey, R. A., (1977) "Patterns of Confounding in Factorial Designs," *Biometrika*, 64, 597-603.
7. Bailey, R. A., (1982) "The Decomposition of Treatment Degress of Freedom in Quantitative Factorial Experiments," *Journal of the Royal Statistical Society - Series B*, 44, 63-70.
8. Bartlett, M. S. and Kendall, D. G., (1946) "The Statistical Analysis of Variance Heterogeneity and the Logarithmic Transformation," *Journal of the Royal Statistical Society - Series B*, 8, 128-150.
9. Beckman, R. J. and McKay, M. D., (1987) "Monte Carlo Estimation Under Different Distributions Using the Same Simulation," *Technometrics*, 29, 153-160.
10. Bekey, G. A. and Masri, S. F., (1983) "Random Search Techniques for Optimization of Nonlinear Systems with Many Parameters," *Math. Comput. Simulation*, 25, 210-213.
11. Berk, K. N. and Picard, R. R. , (1991) "Significance Tests for Saturated Orthogonal Arrays," *Journal of Quality Technology*, 23, 79-89.

12. Bernardo, M. C., Buck, R., Liu, L., Nazaret, W. A., Sacks, J., and Welch, W. J., (1992) "Integrated Circuit Design Optimization Using a Sequential Strategy," *IEEE Transactions in Computer-Aided Design*, 11, 361-372.

13. Bissel, A. F., (1989) "Interpreting Mean Squares in Saturated Fractional Designs," *Journal of Applied Statistics*, 16, 7-18.

14. Bissell, A. F., (1992) "Mean Squares in Saturated Fractional Designs Revisited," *Journal of Applied Statistics*, 19, 351-366.

15. Bjornstad, J. F., (1990) "Predictive Likelihood: A Review," *Statistical Science*, 5, 242-265.

16. Box, G. E. P. and Wilson, K. B., (1951) "On the Experimental Attainment of Optimum Conditions (with discussion)," *Journal of the Royal Statistical Society - Series B*, 13, 1-45.

17. Box, G. E. P. and Youle, P. V., (1955) "The Exploration and Exploitation of Response Surfaces: An Example of the Link Between the Fitted Surface and the Basic Mechanism of the System," *Biometrics*, 11, 287-323.

18. Box, G. E. P. and Hunter, J. S., (1957) "Multifactor Experimental Designs for Exploring Response Surfaces," *The Annals of Mathematical Statistics*, 28, 195-241.

19. Box, G. E. P. and Draper, N. R., (1959) "A Basis for the Selection of a Response Surface Design," *Journal of the American Statistical Association*, 54, 622-654.

20. Box, G. E. P. and Behnken, D. W., (1960) "Some New Three Level Designs for the Study of Quantitative Variables," *Technometrics*, 2, 455-475.

21. Box, G. E. P., (1963) "The Effects of Errors in the Factor Levels and Experimental Design," *Technometrics*, 5, 247-262.

22. Box, G. E. P. and Draper, N. R., (1963) "The Choice of a Second Order Rotatable Design," *Biometrika*, 50, 335-352.

23. Box, G. E. P. and N. R. Draper, (1969) *Evolutionary Operation*, Wiley, New York.

24. Box, G. E. P. and Draper, N. R., (1975) "Robust Designs," *Biometrika*, 62, 347-352.

25. Box, G. E. P., Hunter, W. G., and Hunter, J. S., (1978) *Statistics for Experimenters* , John Wiley , New York .

26. Box, G. E. P., (1982) ''Choice of Response Surface Design and Alphabetic Optimality,'' *Utilitas Mathematica*, 21B, 11-55.

27. Box, G. E. P. and Draper, N. R., (1982) ''Measures of Lack of Fit for Response Surface Designs and Predictor Variable Transformations,'' *Technometrics*, 23, 1-8.

28. Box, G. E. P. and Meyer, R. D., (1986) ''Dispersion Effects from Fractional Designs,'' *Technometrics*, 28, 19-27.

29. Box, G. E. P. and Meyer, R. D., (1986) ''An Analysis of Unreplicated Fractional Factorials,'' *Technometrics*, 28, 11-18.

30. Box, G. E. P. and Draper, N. R., (1987) *Empirical Model Building and Response Surfaces* , John Wiley & Sons , New York, N.Y. .

31. Box, G. E. P., (1988) ''Signal to Noise Ratios, Performance Criteria and Transformations (with discussion),'' *Technometrics*, 30, 1-40.

32. Box, G. E. P., Bisgaard, S., and Fung, C., (1988) ''An Explanation and Critique of Taguchi's Contributions to Quality Engineering,'' *Quality and Reliability Engineering International*, 4, 123-131.

33. Box, G. E. P. and Jones, G., (1992) ''Split-plot Designs for Robust Product Experimentation,'' *Journal of Applied Statistics*, 19, 3-26.

34. Boyles, R. A., (1991) ''The Taguchi Capability Index,'' *Journal of Quality Technology*, 23, 17-26.

35. Brayton, R. K., Hachtel, and Sangiovanni-Vincentelli, A. L., (1981) ''A Survey of Optimization Techniques for Integrated-Circuit Design,'' *Proceedings of the IEEE*, 69, 1334-1362.

36. Bullington, K. E., Hool, J. N., and Maghsoodloo, S., (1990) ''A Simple Method for Obtaining Resolution IV Designs for Use with Taguchi's Orthogonal Arrays,'' *Journal of Quality Technology*, 22, 260-264.

37. Buxton, J. R., (1992) ''Some Comments on the Application of Diagnostic Techiques in Empirical Modelling,'' *Journal of Applied Statistics*, 19, 211-222.

38. Chen, J. and Wu, C. F. J., (1991) ''Some Results on s^{n-k} Fractional Factorial Designs with Minimum Aberration or Optimal Moments,'' *Annals of*

Statistics, 19, 1028-1041.

39. Cheng, C.-S., (1985) "Run Order of Factorial Designs," in *Proceedings of Berkley Conference in Honor of Jerzy Neyman and Jack Kiefer*, vol. 2, pp. 619-633, Wadsworth Advanced Books and Software, Hayward, CA.

40. Cheng, C.-S., (1989) "Some Orthogonal Main Effects Plans for Asymmetrical Factorials," *Technometrics*, 31, 475-478.

41. Christensen, R., Johnson, W., and Pearson, L. M., (1992) "Prediction Diagnostics for Spatial Linear Models," *Biometrika*, 79, 583-592.

42. Cook, R. D. and Nachtsheim, C. J., (1980) "A Comparison of Algorithms for Constructing Exact D-Optimal Designs," *Technometrics*, 22, 315-324.

43. Cook, R. D. and Nachtsheim, C. J., (1989) "Computer-Aided Blocking of Factorial and Response Surface Designs," *Technometrics*, 31, 339-346.

44. Cox, D. R., (1990) "Role of Models in Statistical Analysis," *Statistical Science*, 5, 169-174.

45. Cressie, N., (1991) *Statistics for Spatial Data*, John Wiley & Sons , New York, N.Y. .

46. Currin, C., Mitchell, T., Morris, M., and Ylvisaker, D., (1991) "Bayesian Prediction of Deterministic Functions, with Application to the Design and Analysis of Computer Experiments," *Journal of the American Statistical Association*, 86, 953-963.

47. D'Errico, J. R. and Zaino, N. A., (1988) "Statistical Tolerancing Using a Modification of Taguchi's Method," *Technometrics*, 30, 397-405.

48. Daniel, C., (1959) "Use of Half-Normal Plots in Interpreting Factorial Two-Level Experiements," *Technometrics*, 1, 311-341.

49. Davidian, M. and Carroll, R. J., (1987) "Variance Function Estimation," *Journal of the American Statistical Association*, 82, 1079-1091.

50. Davis, P. J. and Rabinowitz, P., (1984) *Methods of Numerical Integration*, 2nd ed., Academic, Orlando, Fla..

51. Defeo, P. and Myers, R. H., (1992) "A New Look at Experimental Design Robustness," *Biometrika*, 79, 375-381.

52. Dehnad, K., (1989) *Quality Control, Robust Design and the Taguchi Method*, Wadsworth.

53. Deming, S. N. and Morgan, S. L., (1983) "Teaching the Fundamentals of Experimental Design," *Analytica Chimica Acta*, 150, 183-198.

54. Derringer, G. and Suich, R., (1980) "Simultaneous Optimization of Several Response Variables," *Journal of Quality Technology*, 12, 214-219.

55. Dietrich, C. R., (1991) "Modality of the Restricted Maximum Likelihood for Spatial Gaussian Random Fields," *Biometrika*, 78, 833-840.

56. Dietrich, C. R. and Osborne, M. R., (1991) "Estimation of Covariance Parameters in Kriging via Restricted Maximum Likelihood," *Mathematical Geology*, 23, 119-135.

57. Downing, D. J., Gardner, R. H., and Hoffman, F. O., (1985) "An Examination of Response Surface Methodologies for Uncertainty Analysis in Assessment Models," *Technometrics*, 27, 151-163.

58. Dozenaksay, N., Falten, F. W., and Tucker, W. T., (1991) "Identification of Out of Control Quality Characteristics in a Multivariate Manufacturing Environment," *Communications in Statistics - Theory and Methods*, 20, 2775-2790.

59. Draper, N. R., (1985) "Small Composite Designs," *Technometrics*, 27, 173-180.

60. Draper, N. R. and Lin, D. K. J., (1990) "Small Response Surface Designs," *Technometrics*, 32, 187-194.

61. Draper, N. R. and Lin, D. K. J., (1990) "Connections Between 2-Level Designs of Resolution III^* and V," *Technometrics*, 32, 283-288.

62. Draper, N. R., (1992) "Applied Regression Analysis Bibliography Update 1990-1991," *Communication in Statistics: Theory and Methods*, 21, 2415-2438.

63. Easterling, R. G., Johnson, M. E., Bement, T. R., and Nachtsheim, Christopher J., (1991) "Statistical Tolerancing Based on Consumer's Risk Considerations," *Journal of Quality Technology*, 23, 1-11.

64. Engel, J., (1992) "Modelling Variation in Industrial Experiments," *Applied Statistics*, 41, 579-594.

65. Fearn, T., (1992) "Box-Cox Transformations and the Taguchi Method: an Alternative Analysis of a Taguchi Case Study," *Applied Statistics*, 41, 553-562.

66. Fienberg, S. E., (1992) "A Brief History of Statistics in Three and One-Half Chapters: A Review Essay," *Statistical Science*, 7, 208-225.

67. Ford, I., Kitsos, C. P., and Titterington D. M., (1989) "Recent Advances in Nonlinear Experimental Design," *Technometrics*, 31, 49-60.

68. Franklin, M. F. and Bailey, R. A., (1977) "Selection of Defining Contrasts and Confounded Effects in Two-Level Experiments," *Journal of the Royal Statistical Society-C*, 26, 321-326.

69. Franklin, M. F., (1985) "Selecting Defining Contrasts and Confounded Effects in p^{n-m} Factorial Experiments," *Technometrics*, 27, 165-172.

70. Freeny, A. E. and Nair, V. N., (1992) "Robust Parameter Design with Uncontrolled Noise Variables," *Statistics Sinica*, July.

71. Fries, A. and Hunter, W. G., (1980) "Minimum Aberration 2^{k-p} Designs," *Technometrics*, 22, 601-608.

72. Gelfand, S. B. and Mitter, S. K., (1991) "Recursive Stochastic Algorithms for Global Optimization in R^d," *SIAM Journal on Control and Optimization*, 29, 999-1018.

73. Gill, P. E., Hammarling, S. J., Murray, W., Saunders, M. A., and Wright, M. H., (1986) "User's Guide to LSSOL (Version 1.0)," Dept. of Operations Research, Stanford University (Report SOL 86-1), Stanford University.

74. Giovannitti-Jensen, A. and Myers, R. H., (1989) "Graphical Assessment of the Prediction Capability of Response Surface Designs," *Technometrics*, 31, 159-171.

75. Grondona, M. O. and Cressie, N., (1991) "Using Spatial Considerations in the Analysis of Experiments," *Technometrics*, 33, 381-392.

76. Grove, D. M. and Davis, T. P., "Taguchi's Idle Column Method," *Technometrics*, 33, 349-354.

77. Hackl, P. and Ledolter, J., (1992) "A New Nonparametric Quality Control Technique," *Communications in Statistics - Simulations and Computation*, 21, 423-444.

78. Hahn, G. J. and Dershowitz, A. F., (1974) "Evolutionary Operation Today-Some Survey Results and Observations," *Applied Statistics*, 23, 214-218.

79. Hamada, M. and Wu, C. F. J., (1990) "A Critical Look at Accumulation Analysis and Related Methods (with discussion)," *Technometrics*, 32, 119-130.

80. Hamada, M., (1992) "An Explanation and Criticism of Minute Accumulating Analysis: Taguchi's Method for Life Test Data," *Journal of Quality Technology*, 24, 70-77.

81. Hamada, M. and Wu, C. F. J., (1992) "Analysis of Designed Experiments with Complex Aliasing," *Journal of Quality Technology*, 24, 130-137.

82. Handcock, M. S., (1991) "On Cascading Latin Hypercube Designs and Additive Models for Experiments," *Communications in Statistics-Theory and Methods*, 20, 417-440.

83. Hardin, J. M. and Kurkjian, B. M., (1989) "The Calculus for Factorial Arrangements: A Review and Bibliography," *Communications in Statistics: Theory and Methods*, 18, 1251-1277.

84. Harville, D. A., (1977) "Maximum Likelihood Approaches to Variance Component Estimation and to Related Problems," *Journal of the American Statistical Association*, 72, 320-340.

85. Harville, D. A. and Jeske, D. R., (1992) "Mean Squared Error of Estimation or Prediction under a General Linear Model," *Journal of the American Statistical Association*, 87, 724-731.

86. Haslett, J., (1992) "Spatial Data Analysis - Challenges," *The Statistician*, 41, 271-284.

87. Hill, W. J. and Hunter, W. G., (1966) "A Review of Response Surface Methodology: A Literature Review," *Technometrics*, 8, 571-590.

88. Hill, W. J. and Wiles, R. A., (1975) "Plant Experimentation (PLEX)," *Journal of Quality Technology*, 7, 115-122.

89. Hunter, J. S., (1985) "Statistical Design Applied to Product Design," *Journal of Quality Technology*, 17, 210-221.

90. Hunter, W. G. and Kittrell, J. R., (1966) "Evolutionary Operation: A Review," *Technometrics*, 8, 389-397.

91. Hunter, W. G., (1981) "The Practice of Statistics: The Real World is an Idea Whose Time Has Come," *The American Statistician*, 35, 72-76.

92. Ilumoka, A. and Spence, R., (1988) "Parameter Tolerance Design for Electrical Circuits," *Quality and Reliability Engineering International*, 4, 87-94.

93. Iman, R. L. and Conover, W. J., (1980) "Small Sample Sensitivity Analysis Techniques for Computer Models, With Application to Risk Assessment (with discussion)," *Communications in Statistics - Theory and Methods*, A9, 1749-1874.

94. Iman, R. L. and Conover, W. J., (1982) "A Distribution-Free Approach to Inducing Rank Correlation Among Input Variables," *Communications in Statistics- Simulation and Computation*, 11, 311-334.

95. Iman, R. L. and Helton, J. C., (1988) "An Investigation of Uncertainty and Sensitivity Analysis Techniques for Computer Models," *Risk Analysis*, 8, 71-90.

96. Jebb, A. and Wynn, H. P., (1989) "Robust Engineering Design Post-Taguchi," *Phil. Trans. R. Soc. London A*, 327, 605-616.

97. Jensen, S. T., Johansen, S., and Lauritzen, S. L., (1991) "Globally Convergent Algorithms for Maximizing a Likelihood," *Biometrika*, 78, 867-878.

98. Juan, J. and Pena, D., (1992) "A Simple Method to Identify Significant Effects in Unreplicated Two-Level Factorial Designs," *Communications in Statistics - Theory and Methods*, 21, 1383-1404.

99. Juran, J. M., *Quality Control Handbook,* Third Edition, McGraw Hill, New York.

100. Juran, J. M. and Gryna, F. M., *Quality Planning and Analysis.*

101. Kacker, R. N., (1985) "Off-line Quality Control, Parameter Design and the Taguchi Method (with discussion)," *Journal of Quality Technology*, 17, 176-209.

102. Kacker, R. N. and Shoemaker, A. C., (1986) "Robust Design: A Cost-Effective Method for Improving Maunfacturing Processes," *AT&T Technical Journal*, 65, 39-50.

103. Kacker, R. N. and Tsui, K.-L., (1990) "Interaction Graphs: Graphical Aids for Planning Experiments," *Journal of Quality Technology*, 22, 1-14.

104. Kacker, R. N., Lagergren, E. S., and Filliben, J. J., (1991) "Taguchi's Fixed-Element Arrays are Fractional Factorials," *Journal of Quality*

Technology, 23, 107-116.

105. Khuri, A. I. and Cornell, J. A., (1987) *Response Surfaces: Design and Analysis*, Marcel Dekker, New York, N.Y..

106. Kiefer, J., (1959) "Optimum Experimental Designs (with discussion)," *Journal of the Royal Statistical Society - Series B*, 21, 272-319.

107. Kitanidis, P. K., (1985) "Minimum-Variance Unbiased Quadratic Estimation of Covariance of Regionalized Variables," *Mathematical Geology*, 17, 195-208.

108. Kusaba, I., (1988) "Statistical Methods in Japanese Quality Control," *Societas Qualitatis*, 2, 1-3.

109. Lehmann, E. L., (1990) "Model Specification: The Views of Fisher and Neyman, and Later Developments," *Statistical Science*, 5, 160-168.

110. Lenth, R. V., (1989) "Quick and Easy Analysis of Unreplicated Factorials," *Technometrics*, 31, 469-474.

111. León, R. V., Shoemaker, A. C., and Kacker, R. N., (1987) "Performance Measures Independent of Adjustment," *Technometrics*, 29, 253-265.

112. Lewis, S., (1982) "Generators for Factorial Experiments," *Journal of Statistical Planning and Inference*, 6, 59-64.

113. Lin, D. K. J. and Draper, N. R. , (1992) "Projection Properties of Placett and Burman Designs," *Technometrics*, 34, 423-428.

114. Liu, W., (1992) "Predictive Screening," *Communications in Statistics-Theory & Methods*, 21, 2349-2366.

115. Logothetis, N., (1987) "Off-line Quality Control with Initial Exploration of Data," *GEC Journal of Research*, 5, 40-48.

116. Logothetis, N., (1990) "Box-Cox Transformations and the Taguchi Method," *Applied Statistics*, 39, 31-48.

117. Lotwick, H. W. and Silverman, B. W., (1982) "Methods for Analysing Spatial Processes of Several Types of Points," *Journal of the Royal Statistical Society - Series B*, 44, 406-413.

118. Low, K. K. and Director, S. W., (1988) "An Efficient Macromodeling Approach for Statistical IC Process Design," in *IEEE Int. Conf. on Computer-Aided Design*, pp. 16-19, Santa Clara, CA.

119. Lowe, C. W., (1974) "Evolutionary Operation in Action," *Applied Statistics*, 23, 218-226.

120. Mager, P. P., (1982) "Multivariate Response Surface Optimization," *Pharmizie*, 37, 658-660.

121. Mak, T. K., (1992) "Estimation of Parameters in Heteroscedastic Linear Models," *Journal of the Royal Statistical Society - Series B*, 54, 649-655.

122. Mardia, K. V., (1980) "Some Statistical Inference Problems in Kriging II," in *Proceedings of the 26th International Geological Congress, Sciences de la Terre, Serie "Informatique Ge´ologique*, vol. 15, pp. 113-131.

123. Mardia, K. V. and Marshall, R. J. , (1984) "Maximum Likelihood Estimation of Models for Residual Covariance in Spatial Regression," *Biometrika*, 71, 135-146.

124. Mardia, K. V. and Watkins, A. J., (1989) "On Multimodality of the Likelihood in the Spatial Linear Model," *Biometrika*, 76, 289-295.

125. Mardia, K. V. and Dryden, I. L., (1989) "The Statistical Analysis of Shape Data," *Biometrika*, 76, 271-281.

126. Marshall, R. J. and Mardia, K. V., (1985) "Minimum Norm Quadratic Estimation of Components of Spatial Covariance," *Mathematical Geology*, 17, 517-525.

127. Masri, S. F., Bekey, G. A., and Safford, F. B., (1976) "An Adaptive Random Search Method for Identification of Large-Scale Nonlinear Systems," *Proceedings 4th IFAC Symposium on Identification and System Parameter Estimation*, 246-255.

128. Matheron, G., (1963) "Principles of Geostatistics," *Economic Geology*, 58, 1246-1266.

129. McKay, M. D., Conover, W. J., and Beckman, R. J., (1979) "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, 21, 239-245.

130. McPherson, G., (1989) "The Scientists View of Statistics - A Neglected Area (with comments)," *Journal of the Royal Statistical Society-A*, 152, 221-240.

131. Mead, R., (1990) "The Non-Orthongonal Design of Experiments (with discussion)," *Journal of the Royal Statistical Society-A*, 153, 151-202.

132. Mee, R. W., (1990) "An Improved Procedure for Screening Based on a Correlated, Normally Distributed Variable," *Technometrics*, 32, 331-338.

133. Mitchell, T. J., (1974) "An Algorithm for the Construction of D-optimal Experimental Designs," *Technometrics*, 16, 203-210.

134. Mitchell, T. J., (1974) "Computer Construction of 'D-optimal' First-order Designs," *Technometrics*, 16, 211-220.

135. Mitchell, T. J. and Bayne, C. K., (1978) "D-optimal Fractions of Three-level Factorial Designs," *Technometrics*, 20, 369-380.

136. Monrod, H. and Bailey, R. A., (1992) "Pseudofactors Normal Use to Improve Design and Facilitate Analysis," *Applied Statistics*, 41, 317-336.

137. Montgomery, D. C., (1991) "Using Fractional Factorial Designs for Robust Design Process Development," *Quality Engineering*, 3, 193-205.

138. Morris, M. D. and Mitchell, T. J., (1983) "Two-level Multifactor Designs for Detecting the Presences of Interaction," *Technometrics*, 25, 345-355.

139. Morris, M. D., (1987) "Two-Stage Factor Screening Procedures Using Multiple Grouping Assignments," *Communications in Statistics - Theory and Methods*, A16, 3051-3067.

140. Morris, M. D., (1991) "Factorial Sampling Plans for Preliminary Computational Experiments," *Technometrics*, 33, 161-174.

141. Morris, M. D., Mitchell, T. J., and Ylvisaker, D., (1991) "Bayesian Design and Analysis of Computer Experiments: Use of Derivatives in Surface Prediction," ORNL Technical Report (ORNL/TM-11699).

142. Morrison, S. J., (1957) "The Study of Variability in Engineering Design," *Applied Statistics*, 6, 133-138.

143. Myers, R. H. and Carter, W. H. Jr., (1973) "Response Surface Techniques for Dual Response Systems," *Technometrics*, 15, 301-317.

144. Myers, R. H., Khuri, A. I., and Carter, W. H., (1989) "Response Surface Methodology: 1966-1988," *Technometrics*, 31, 137-157.

145. Myers, R. H., (1991) "Response Surface Methodology in Quality Improvement," *Communications in Statistics-Theory and Methods*, 20, 457-476.

146. Myers, R. H., Vining, G. G., Giovannitti-Jensen, A., and Myers, Sharon L., (1992) "Variance Dispersion Properties of Second-Order Response Surface Designs," *Journal of Quality Technology*, 24, 1-11.

147. Myers, R. H., Khuri, A. I., and Vining, G., (1992) "Response Surface Alternatives to the Taguchi Robust Parameter Design Approach," *The American Statistician*, 46, 131-139.

148. Nair, V. N. and Pregibon, D., (1986) "A Data Analysis Strategy for Quality Engineering Experiments," *AT&T Technical Journal*, 65, 73-84.

149. Nair, V. N. and Pregibon, D., (1988) "Analyzing Dispersion Effects from Replicated Factorial Experiments," *Technometrics*, 30, 247-257.

150. Nair, V. N., (1992) "Taguchi's Parameter Design: A Panel Discussion," *Technometrics*, 34, 127-161.

151. Nazaret, W. and Liu, L., (1990) "Computer Aided Design for Quality(CADQ)," *AT&T Technical Journal*, 69, 46-60.

152. Nelder, J. A. and Mead, R., (1965) "A Simplex Method for Function Minimization," *Computer Journal*, 7, 308-313.

153. Nelder, J. A. and Pregibon, D., (1987) "An Extended Quasi-likelihood Function," *Biometrika*, 74, 221-232.

154. Nelder, J. A. and Lee, Y., (1991) "Generalized Linear Models for the Analysis of Taguchi-type Experiments," *Applied Stochastic Models and Data Analysis*, 7, 107-120.

155. Nelder, J. A. and Lee, Y., (1992) "Likelihood, Quasi-likelihood and Psuedo-likelihood: Some Comparisons," *Journal of the Royal Statistical Society - Series B*, 54, 273-284.

156. O'Hagan, A., (1978) "Curve Fitting and Optimal Design for Prediction (with discussion)," *Journal of the Royal Statistical Society - Series B*, 40, 1-41.

157. Owen, A. B., (1991) "Orthogonal Arrays for Computer Experiments, Integration and Visualization," *Technical Report, University of Stanford.*

158. Owen, A. B., (1992) "Lattice Sampling Revisited: Monte Carlo Variance of Means Over Randomized Orthogonal Arrays," Stanford University Technical Report.

159. Owen, A. B., (1992) "A Central Limit Theorem for Latin Hypercube Sampling," *Journal of the Royal Statistical Society - Series B*, 54, 541-552.

160. Patterson, H. D., (1954) "The Errors of Lattice Sampling," *Journal of the Royal Statistical Society - Series B*, 16, 140-149.

161. Patterson, H. D., (1976) "Generation of Factorial Designs," *Journal of the Royal Statistical Society - Series B*, 38, 175-179.

162. Phadke, M. S., (1982) "Quality Engineering Using Design of Experiments," *Proceedings of the section on Statistical Education, ASA*, 11-20.

163. Phadke, M. S., (1986) "Design Optimization Case Studies," *AT&T Technical Journal*, 65, 51-68.

164. Phadke, M. S. and Dehnad, K., (1988) "Optimization of Product and Process Design for Quality and Cost," *Quality and Reliability Engineering International*, 4, 105-112.

165. Phadke, M. S., (1989) *Quality Engineering Using Robust Design*, Prentice Hall, New Jersey.

166. Plackett, R. L. and Burman, J. P., (1946) "The Design of Optimum Multifactorial Experiments," *Biometrika*, 33, 305-325.

167. Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T., (1986) *Numerical Recipes,* Cambridge University Press, Cambridge.

168. Pronzato, L., Walter, E., Venot, A., and Lebruchec, J., (1984) "A General-Purpose Global Optimizer: Implementation and Applications," *Mathematics and Computers in Simulation*, 26, 412-422.

169. Ripley, B. D., (1977) "Modelling Spatial Pattern (with discussion)," *Journal of the Royal Statistical Society - Series B*, 39, 172-212.

170. Rosenblatt, A. and Watson, G. F., (1991) "Concurrent Engineering," *IEEE Spectrum*, July, 22-37.

171. Sacks, J. and Schiller, S., "Spatial Designs," in *Statistical Decision Theory and Related Topics IV*, ed. Berger, J. O., vol. 2, pp. 385-399, Springer, New York.

172. Sacks, J. and Ylvisaker, D., (1970) "Statistical Designs and Integral Approximation," in *Proc. 12th Bien. Sem. Canad. Math. Congress*, ed. R. Pyke, pp. 115-136, Canadian Mathematical Congress, Montreal.

173. Sacks, J. and Ylvisaker, D., (1978) "Linear Estimation for Approximately Linear Models," *Annals of Statistics*, 6, 1122-1137.

174. Sacks, J., Schiller, S. B., and Welch, W. J., (1989) "Design for Computer Experiments," *Technometrics*, 31, 41-47.

175. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., (1989) "Design and Analysis of Computer Experiments," *Statistical Science*, 4, 409-435.

176. Saunders, I. W. and Eccleston, J. A., (1992) "Experimental Design for Continuous Processes," *Australian Journal of Statistics*, 34, 77-90.

177. Schneider, H. and Tang, K., (1990) "Adaptive Procedures for the 2-Stage Group Testing Problem Based on Prior Distributions and Costs," *Technometrics*, 32, 397-406.

178. Shilling, M. F., (1992) "Spatial Design When the Observations are Correlated," *Communications in Statistics - Simulation and Computation*, 21, 243-268.

179. Shoemaker, A. C. and Kackar, R. N., (1988) "A Methodology for Planning Experiments in Robust Product and Process Design," *Quality and Reliability Engineering International*, 4, 95-103.

180. Shoemaker, A. C., Tsui, K.-L., and Wu, C. F. J., (1991) "Economical Experimentation Methods for Robust Design," *Technometrics*, 33, 415-428.

181. Smith, D. M., (1991) "AS268-All Possible Subset Regressions Using the QR Decomposition," *Applied Statistics*, 40, 502-513.

182. Smyth, G. K., (1989) "Generalized Linear Models with Varying Dispersion," *Journal of the Royal Statistical Society - Series B*, 51, 47-60.

183. Stein, A. and Corsten, L. C. A., (1991) "Universal Kriging and Cokriging as a Regression Procedure," *Biometrics*, 47, 575-588.

184. Stein, M., (1987) "Large Sample Properties of Simulations Using Latin Hypercube Sampling," *Technometrics*, 29, 143-151.

185. Stein, M., (1989) "Asymptotic Distribution of Minimum Norm Quadratic Estimators of the Covariance Function of a Gaussian Random Field," *Annals of Statistics*, 17, 980-1000.

186. Stein, M., (1990) "Uniform Asymptotic Optimality of Linear Predictions of a Random Field using an Incorrect Second-order Structure," *Annals of Statistics*, 18, 850-872.

187. Stein, M., (1990) "Bounds on the Efficiency of Linear Predictions using an Incorrect Covariance Function," *Annals of Statistics*, 18, 1116-1138.

188. Stein, M., (1990) "A Comparison of Generalized Cross Validation and Modified Maximum Likelihood for Estimating the Parameters of a

Stochastic Process," *Annals of Statistics*, 18, 1139-1157.

189. Stein, M., (1991) "A Kernel Approximation to the Kriging Predictor of a Spatial Process," *Annals of the Institute of Statistical Mathematics*, 43, 61-75.

190. Steinberg, D. M. and Hunter, W. G., (1984) "Experimental Design: Review and Comment (with discussion)," *Technometrics*, 26, 71-130.

191. Steinberg, D. M., (1985) "Model Robust Response Surface Designs: Scaling Two-level Factorials," *Biometrika*, 72, 513-526.

192. Taguchi, G. and Y. I. Wu, (1979) *Introduction to Off-Line Quality Control*, Central Japan Quality Control Association.

193. Taguchi, G., (1986) *Introduction to Quality Engineering*, Asian Productivity Organization, Tokyo.

194. Taguchi, G., (1989) "Experimental Design for Dynamic Characteristics," *ASI Journal*, 2, 7-24.

195. Tjur, T., (1991) "Block Designs and Electrical Networks," *Annals of Statistics*, 19, 1010-1027.

196. Tribus, M. and Szonyi, G., (1989) "An Alternative View of the Taguchi Approach," *Quality Progress*, 22, 46-52.

197. Tsui, K.-L., (1988) "Strategies for Planning Experiments Using Orthogonal Array and Confounding Tables," *Quality and Reliability Engineering International*, 4, 113-122.

198. Turiel, T. P., (1988) "A FORTRAN Program to Generate Fractional Factorial Experiments," *Journal of Quality Technology*, 20, 63-72.

199. Vecchia, A. V., (1988) "Estimation and Model Identification for Continuous Spatial Process," *Journal of the Royal Statistical Society - Series B*, 50, 297-312.

200. Vecchia, A. V., (1992) "A New Method of Prediction for Spatial Regression Models," *Journal of the Royal Statistical Society - Series B*, 54, 813-830.

201. Vining, G. G. and R. H. Myers, (1990) "Combining Taguchi and Response Surface Philosophies: A Dual Response Approach," *Journal of Quality Technology*, 22, 38-45.

202. Vining, G. G. and Myers, R. H., (1991) "A Graphical Approach for Evaluating Response Surface Designs in Terms of the Mean Squared Error of Prediction," *Technometrics*, 33, 315-326.

203. Voss, D. T. and Dean, A. M., (1987) "Equivalence of Methods of Confounding in Asymmetrical Single Replicate Factorial Designs," *Annals of Statistics*, 15, 376-384.

204. Wang, J. C. and Wu, C. F. J., (1991) "An Approach to the Construction of Asymmetrical Orthogonal Arrays," *Journal of the American Statistical Association*, 86, 450-456.

205. Wang, J. C. and Wu, C. F. J., (1992) "Nearly Orthogonal Arrays with Mixed Levels and Small Runs," *Technometrics*, 34, 409-422.

206. Wang, W. and Lawson, J., (1988) "Estimating the Error of an Effect in Unreplicated 2-Level Fractional Factorial Designs," *Quality and Reliability Engineering International*, 4, 189-192.

207. Warnes, J. J. and Ripley, B. D., (1987) "Problems with Likelihood Estimation of Covariance Functions of Spatial Gaussian Processes," *Biometrika*, 74, 640-642.

208. Watkins, A. J. and Mardia, K. V., (1992) "Maximum Likelihood Estimation and Prediction Mean Square Error in the Spatial Linear Model," *Journal of Applied Statistics*, 19, 49-60.

209. Welch, W. J., (1982) "Branch-and-bound Search for Experimental Designs Based on D Optimality and Other Criteria," *Technometrics*, 24, 41-48.

210. Welch, W. J., (1983) "A Mean Squared Error Criterion for the Design of Experiments," *Biometrika*, 70, 205-213.

211. Welch, W. J., Yu, Tat-Kuan, Kang, S. M., and Sacks, J., (1990) "Computer Experiments for Quality Control by Parameter Design," *Journal of Quality Technology*, 22, 15-22.

212. Welch, W. J. and Sacks, J., (1991) "A System for Quality Improvement Via Computer Experiments," *Communications in Statistics-Theory and Methods*, 20, 477-496.

213. 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, 15-25.

214. Wong, W.-K., (1992) "A Unified Approach to the Construction of Minimax Designs," *Biometrika*, 79, 611-620.

215. Wu, C. F. J., Mao, S. S., and Ma, F. S., (1979) "SEL: A Search Method Based on Orthogonal Arrays," in *Statistical Design and Analysis of Industrial Experiments*, ed. S. Ghosh, pp. 279-310, Marcel Dekker.

216. Wu, C. F. J. and Chen, Youyi, (1992) "A Graph-Aided Method for Planning Two-Level Experiments when Certain Interactions are Important.," *Technometrics*, 34, 162-175.

217. Wynn, H. P., Sivaloganathan, S., Buck, R. J., and Lewis, S. M., (1992) "Generate: An Algorithm for the Computer Generation of Orthogonal Arrays with Specific Alias Structure," Engineering Design and Quality Centre Technical Report #30.

218. Yakowitz, S. J. and Szidarovsky, F., (1985) "A Comparison of Kriging and Nonparametric Regression Methods," *Journal of Multivariate Analysis*, 16, 21-53.

219. Yang, J. and Kushner, (1991) "A Monte Carlo Method for Sensitivity Analysis and Parameter Optimization of Nonlinear Stochastic Systems," *SIAM Journal on Control and Optimization*, 29, 1216-1249.

220. Ylvisaker, D., (1975) "Designs on Random Fields," in *A Survey of Statistical Design and Linear Models*, ed. J. N. Srivastava, pp. 593-607, North-Holland, Amsterdam.

221. Ylvisaker, D., (1987) "Prediction and Design," *Annals of Statistics*, 15, 1-19.

222. Young, D. L., Teplik, J., Weed, H. D., Tracht, N. T., and Alvarez, A. R., (1991) "Application of Statistical Design and Response Surface Methods to Computer-Aided VLSI Device Design II: Desirability Functions and Taguchi Methods," *IEEE Transactions on Computer-aided Design*, 10, 103-115.

223. Yu, T. K., Kang, S. M., Sacks, J., and Welch, W. J., (1991) "Parametric Yield Optimization of CMOS Analogue Circuits by Quadratic Statistical Circuit Performance Models," *International Journal of Circuit Theory and Applications*, 19, 579-592.

224. Zacks, S., (1991) "Inference Based on Taguchi's Saturated Designs," *Communications in Statistics-Theory and Methods*, 20, 497-510.

225. Zahn, D. A., (1975) ''Modifications of and Revised Critical Values for the Half-Normal Plot,'' *Technometrics*, 17, 189-200.

226. Zahn, D. A., (1975) ''An Empirical Study of the Half-Normal Plot,'' *Technometrics*, 17, 201-211.

227. Zahn, D. A. and Isenberg, D. J., (1983) ''Nonstatistical Aspects of Statistical Consulting,'' *The American Statistician*, 37, 297-302.

228. Zhigljavsky, A. A., (1991) *Theory of Global Random Search*. Kluwer Academic Publishers.

229. Zimmerman, D. L., (1989) ''Computationally Exploitable Structure of Covariance Matrices and Generalized Covariance Matrices in Spatial Models,'' *J. Statist. Comput. Simul.*, 32, 1-15.

230. Zimmerman, D. L. and Cressie, N., (1991) ''Mean Squared Prediction Error in the Spatial Linear Model with Estimated Covariance Parameters,'' *Annals of the Institute of Statistical Mathematics*, 43, 27-43.

231. Zimmerman, D. L. and Zimmerman, M. B., (1991) ''A Comparison of Spatial Semivariogram Estimators and Corresponding Ordinary Kriging Predictors,'' *Technometrics*, 33, 77-92.

232. Zimmerman, D. L. and Harville, D. A., (1991) ''A Random Field Approach to the Analysis of Field-Plot Experiments and Other Spatial Experiments,'' *Biometrics*, 47, 223-240.