Strategies for Sequential Design of Experiments

by

Rizwan R. Koita

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1994

Author .... [illegible]
Department of Electrical Engineering and Computer Science
August 2, 1994

Certified by .... [illegible] 8/2/94
David H. Staelin
Professor of Electrical Engineering
Thesis Supervisor

Accepted by .... [illegible]
F. R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Strategies for Sequential Design of Experiments

by

Rizwan R. Koita

Submitted to the Department of Electrical Engineering and Computer Science
on August 2, 1994, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

The growth of competitiveness in the manufacturing industry has resulted in an acute need to improve product quality. Efficient methods for Design of Experiments (DOE) have thus become more important than ever before. With increases in complexity of manufacturing processes and the cost of experimentation, it has become important to use all prior knowledge of the processes, derived first from theory and experience and later from experimentation, to determine subsequent experiments.

The objective of this work is to develop sequential DOE methods and software tools which can be used by manufacturers with no prior knowledge in design of experiments.

Techniques for designing blocks of fractional factorial experiments have been developed and implemented in computer software. In particular, given a block of factorial experiments and a list of probable interactions, different sets of experiments are designed to augment the original block of experiments and form new factorial designs with different confounding patterns. The set of experiments which produces a new design with minimal confounding between the variables and the probable interactions is the optimal set of experiments. Software has been implemented to generate the optimal Full-Block or Half-Block of experiments. The fold-over design technique, which is popularly used in DOE, is a special case of the Full-Block design strategy. The analysis routines have been extended to analyze multiple blocks of experiments with different confounding patterns and to determine the effects of the confounding interactions.

A One-at-a-Time design strategy has been proposed which uses the results of the previous experiments to design the optimal experiments one at a time. This strategy assumes that there is only one significant variable or interaction on each significant column of the design matrix. Therefore, this strategy is particularly useful when the number of significant effects is sparse. A hypothesis is made that certain interactions are significant on the basis of their variables. The optimal experiment for this hypothesis, i.e., the experiment which is expected to yield the maximum output quality, is conducted. The result of this experiment is compared with the predicted output quality of all possible plant models. The model which gives the least prediction error is selected as the next hypothesis and the procedure is repeated. The simulation results for this design methodology have been very promising.

Thesis Supervisor: David H. Staelin
Title: Professor of Electrical Engineering

Acknowledgments

First and foremost, I would like to express my sincere thanks to my advisor Professor David Staelin for his excellent guidance, constant support and understanding. He taught me many things about research and life. Working with him was a privilege.

Thanks to my fellow group members: Ambrose, Carlos, Mark, Michelle and Mike for their help and cooperation. I appreciate the discussions I had with them and enjoyed the group meetings.

I acknowledge the efforts of the EECS Department for making it possible for me to rejoin MIT, particularly the staff of the EECS Graduate Office who have been so patient and supportive through my difficult periods.

Special thanks to Asif Shakeel, Sankar Sunder, S. R. Venkatesh and T. A. Venkatesh. Spending time with you all has probably been the most memorable part of my stay at MIT. Thanks to Amit Kumar who has always been a source of immense help and encouragement and to Firdaus Bhathena for helping me on numerous occasions.

I am very grateful to God for helping to me complete my masters degree at MIT and making life such an interesting challenge.

Finally, I am deeply indebted to my grandmother, my parents, my sister Irfana, Zarina Aunty and Rekha whose love and sacrifices have made a dream come true.

The work in this thesis was carried out at the Research Laboratory of Electronics and was supported by the Leaders for Manufacturing program at MIT.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we outline the motivation driving the research in this area. A brief overview of the existing methodologies is given in Section 2 along with the outline of the thesis in Section 3. Section 4 gives the notation used in the report.

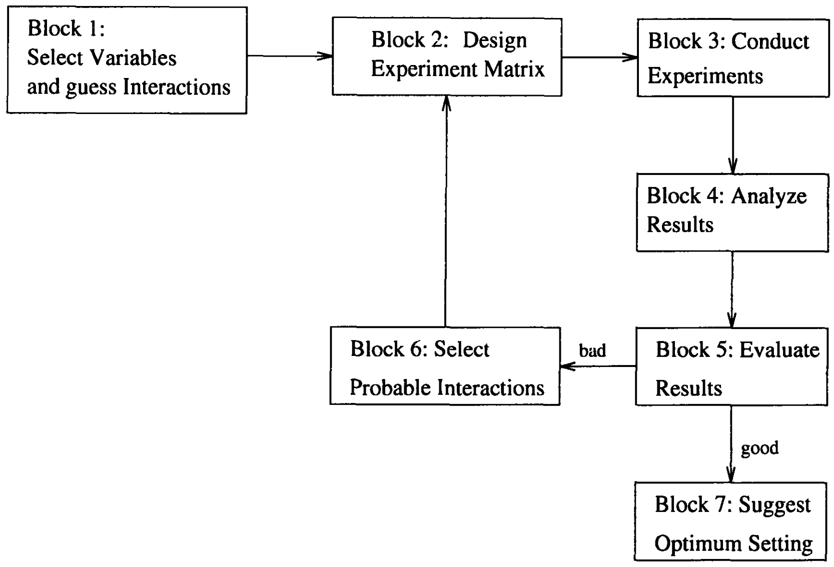
1.1 Reseach Motivation

With the increase in competitiveness in the manufacturing industry the methods of Experimental Design are becoming increasingly important. The need for efficient Design of Experiments (DOE) stems from the desire to improve quality and productivity quickly and economically. The concepts in Design of Experiments have been extensively researched since Fisher [11] introduced the basic theory in 1926.

Although the basic ideas in the DOE are relatively simple, their application involves use of comparatively advanced concepts in algebra and statistics. Hence, despite the extensive literature, DOE has remained largely outside the purview of manufacturing and is used very infrequently, if at all, in today's industry.

With the advent of high speed digital computers it is now possible to design and search for optimal experimental designs and to use statistics without the need of the experimenter to understand all the concepts. The goal of this thesis is to develop an experimental design package for a user who is familiar with the plant/process to be optimized, but has no prior experience with experimental design.

The basic proposed strategy for the DOE method developed here is shown in Figure 1-1. An experimenter inexperienced with DOE can be expected to only perform the following



Block 1: Select Variables and guess Interactions
Block 2: Design Experiment Matrix
Block 3: Conduct Experiments
Block 4: Analyze Results
Block 5: Evaluate Results
Block 6: Select Probable Interactions
Block 7: Suggest Optimum Setting
bad
good

Figure 1-1: Proposed Experimental Design Strategy

tasks:

Block 1: Decide which parameters/ variables affect the process, and the approximate range in which they can be varied for the normal operation of the plant. Also, based on the knowledge of the process, the experimenter can guess some of the probable interactions.

Block 3: Perform the set of experiments as determined by Block 2 of Figure 1-1.

Traditionally, an experimenter was expected to perform the tasks of all the blocks. But as discussed above, it is possible to develop software so that the experimenter can successfully use DOE without being required to understand all the concepts used in the implementation of Blocks 2, 4, 5, 6 and 7 of Figure 1-1. With such simplification, DOE can be used by a large number of manufacturers who are presently unable to do so.

This thesis is the continuation of the work done by Paul Fieguth et al. [10] based on the insights developed by Dr. Ashraf Alkairy [1]. The earlier work focused on developing software for Blocks 2 and 4. The contribution of this thesis is in the development of the theory and software for Blocks 5, 6 and 7 along with an improvement of Block 2.

The thrust of the thesis is on designing sequential experiments. The need for sequential designs arises due to:

- Lack of a priori knowledge of the process/plant.
- Constraints of available resources on the size of experiments.
- Desire to find a good operating point even if unable to determine the best operating point.
- Need to conduct the experiment in a systematic and efficient manner.

All these factors are outlined in greater detail in the next chapter.

1.2 Background

The concept of Design of Experiments (DOE) was first introduced by Fisher [11]. Plackett and Burman [19] proved that in order for the parameters of the design matrix to have certain optimal properties it was essential that the designs have an orthogonal structure. This resulted in the use of Factorial Designs. In these designs, as the number of variables controlling a given process grow, the number of experiments required to completely analyze the variables and their interactions grow exponentially and so become impractical to perform.

Ideas of fractional factorial designs were developed to reduce the total number of experiments and yet retain the beneficial properties of factorial designs. The reduction in the total number of experiments leads to the problem of confounding.

Two variables (or interactions) T_i and T_j are said to be confounded with each other if they vary in the same manner in all the experiments. That is, the value of the variables for the mth experiment is given by,

$$T_{i,m} = k \cdot T_{j,m} \qquad \forall m \tag{1.1}$$

for some fixed k. In two-level factorial designs each of the variables are set at -1 or +1 level in all the experiments. Therefore, T_i and T_j are confounded with each other if,

$$T_{i,m} = +T_{j,m} \qquad \forall m \tag{1.2}$$

or

$$T_{i,m} = -T_{j,m} \qquad \forall m \tag{1.3}$$

Given a process, there exists no well defined technique to select which variables/ interactions are to be confounded. The selection is done on an ad hoc basis and hence an element of subjectivity is introduced in the designs. After the experiments are carried out all the significant variables/interactions may not be determined as there may be overlapping between then.

In order to use DOE effectively it is very important that the exact dependence of quality on the variables and their interactions be established. Hence, whenever two or more variables or interactions are confounded some technique is required by which their separate effects can be determined.

Some of the DOE techniques available to determine the unknown plant models are described below. The discussion is intentionally concise and should only serve to familiarize the reader with the techniques.

1.2.1 Search Designs

Search Designs is a technique used for determining important interactions. Search designs were developed by J. N. Srivastava [21]. The basic strategy of search designs is to design a set of experiments which has the capacity to estimate the effects of a set of variables and interactions and a few unknown interactions of a specified order. Readers may consult Ghosh [13] or Srivastava [22] for a more detailed discussion on Search Designs.

Although the methodology is interesting there are shortcomings.

- The technique is relatively complicated and there exists no general technique to solve all kinds of problems.
- The technique does not use *a posteriori* information and hence there is no obvious extension to sequential experimentation.

1.2.2 Response Surface Methods

The Response Surface Methodology (RSM) is an alternative to factorial DOE. Like factorial DOE, experiments are designed on a process to determine the relationship between the input variables and the response. But in RSM the variables are not constrained to lie at any fixed levels. Each design technique has its own advantages and disadvantages. These issues are discussed in greater detail in [17].

RSM is a sequential design procedure and there are many software packages which use RSM for process optimization [14]. Some of the popular ones are:

- Simplex - Evolutionary Operations (Simplex - V) [1]
- ULTRAMAX [2]

The ideas used in these procedures are significantly different from Factorial DOE discussed in this report. Hence the readers interested in RSM techniques may find the references useful.

1.3 Thesis Overview

The thesis focuses on performing sequential experimental design using two different approaches:

1. Sequential Block Design Strategy
2. One-at-a-Time Design Strategy

Chapter 2 is based on these strategies and gives an overview. The need for sequential experimentation is discussed and the basic assumptions on the models of the manufacturing processes are enumerated.

In Chapter 3 the theory of Sequential Block Design is developed. The detail proofs are given in the Appendices. The algorithm used in the computer software is discussed along with the results of computer simulations.

Chapter 4 focuses on the Analysis of Sequential Block Designs. Classical techniques such as Daniel Plots and Anova are discussed briefly. A closed loop technique for determining significant interactions is described.

Chapter 5 deals with the One-at-a-Time Design Strategy. The basic concepts and advantages are presented in this chapter. Simulation results are included to justify the strategy.

[1]Simplex - V is a tradename of Statistical Program, Houston, Texas
[2]Ultramax is a tradename of Ultramax Corporation, Cincinnati, Ohio

Chapter 6 describes the Complete DOE Software Tool which being currently developed at MIT. It also lists the achievements of the thesis and suggests possible future work on sequential design of experiments.

1.4 Notation

In this report we will try to use consistent notation. Given a plant **P**, its output or quality is denoted by Y. The performance of that plant is governed by input variables. These variables are denoted by T_1, T_2,..., T_n. The model of the plant is assumed to be non-linear and the output Y is assumed to be a linear function of the variables and their interactions. The interaction between the variables T_i and T_j is denoted by I_{ij}. Denoting the coefficients of the variables and interactions by β we can represent the output as,

$$Y = \beta_0 + \beta_1 T_1 + \beta_2 T_2 + \ldots + \beta_n T_n + \ldots + \beta_{ij} I_{ij} + \ldots + \varepsilon \tag{1.4}$$

$$Y = \begin{pmatrix} 1 & T_1 & T_2 & \ldots & T_n & \ldots & I_{ij} & \ldots \end{pmatrix} \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \\ \vdots \\ \beta_{ij} \\ \vdots \end{pmatrix} + \varepsilon \tag{1.5}$$

If there is more than one experiment then there will be one equation like Equation 1.5 for each of the experiments. All such equations can be stacked together.

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix} = \begin{pmatrix} 1 & T_{1,1} & T_{2,1} & \dots & T_{n,1} & \dots & I_{ij,1} & \dots \\ 1 & T_{1,2} & T_{2,2} & \dots & T_{n,2} & \dots & I_{ij,2} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & T_{1,m} & T_{2,m} & \dots & T_{n,m} & \dots & I_{ij,m} & \dots \end{pmatrix} \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \\ \vdots \\ \beta_{ij} \\ \vdots \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{pmatrix} \tag{1.6}$$

This can be represented as,

$$\bar{Y} = \bar{X} \cdot \bar{\beta} + \bar{\varepsilon} \tag{1.7}$$

where $\bar{Y}$ is the **Output Vector**, $\bar{X}$ is the **Regression Matrix**, $\bar{\beta}$ is the **Coefficient Vector** and $\bar{\varepsilon}$ is the **Error Vector**.

1.4.1 Factorial Fractional Designs

For a process, **P**, dependent on n variables, if the experiments are designed is such that

- In each of the experiments, the variables are at either one of two levels, denoted by '+' and '-'.
- There are 2^n different experiments.

then the experimental design is called a **Full Factorial Design.**

If the experimental design has half the number of experiments of a Full Factorial Design then it is called a **One-Half Factorial Design.** Similarly, if it has a quarter of the experiments of a Full Factorial then it is called a **One-Quarter Factorial Design.** In general, a $1/2^k$ fraction of a Full Factorial Design is called a 2^{n-k} **Factorial Design.**

This report deals with DOE using Fractional Factorial Designs which are based on Binary Orthogonal Matrices. Appendix A discusses some of the relevant properties of Binary Orthogonal Matrices. Table 1.1 is an example of a 2^{5-2} Factorial Design based on the $2^3 \times 2^3$ Binary Orthogonal Matrix shown in Table 1.2. The first column of of the matrix

Variable:	T_1	T_2	T_3	T_4	T_5
Experiment	v_1	v_2	v_3	v_{12}	v_{23}
1	+	+	+	+	+
2	+	+	-	+	-
3	+	-	+	-	-
4	+	-	-	-	+
5	-	+	+	-	+
6	-	+	-	-	-
7	-	-	+	+	-
8	-	-	-	+	+

Table 1.1: 2^{5-2} Factorial Design Matrix

Expt.	v_{avg}	v_1	v_2	v_3	v_{12}	v_{23}	v_{31}	v_{123}
1	+	+	+	+	+	+	+	+
2	+	+	+	-	+	-	-	-
3	+	+	-	+	-	-	+	-
4	+	+	-	-	-	+	-	+
5	+	-	+	+	-	+	-	-
6	+	-	+	-	-	-	+	+
7	+	-	-	+	+	-	-	+
8	+	-	-	-	+	+	+	-

Table 1.2: Basic and Non-Basic Columns of a Binary Orthogonal Matrix

consists only of +1 and is called the Average Column (v_{avg}).

1.4.2 Resolution of Factorial Designs

Resolution is a useful concept associated with factorial designs. The resolution of a design is a measure of the ability of the design to resolve confounding between interactions.

A 2^{n-k} factorial design is said to be of resolution R if no Pth order interaction is confounded with another interaction of less than $(R-P)$th order. For example, a design has a resolution III if no variable is confounded with another variable but at least one second order interaction is confounded with a variable. Designs with higher resolution are preferred.

In this report we will usually begin the sequential DOE procedure using a design of resolution IV or higher. Such a design has the property that none of the variables are confounded with second order interactions.

1.4.3 Basic Columns and Variables of Factorial Designs

In this subsection we discuss the notation used to represent an factorial design. An factorial design is viewed as a mapping of variables and interactions to a set of columns of a binary orthogonal matrix.

As discussed in Appendix A, given a binary orthogonal matrix $\bar{X}$ of size 2^m, we can determine a set of m independent columns. These columns are called **basic columns**. The other $2^m - m$ columns are called **non-basic columns**. The basic columns are denoted by the symbols v_1, v_2, ..., v_m. Every non-basic column corresponds to a product of a unique set of basic columns. A non-basic column is denoted by v_{ijk} if it is generated by the product of the columns v_i, v_j and v_k. An example of a 2^3 binary orthogonal matrix is given in Table 1.2.

The set of experiments can be represented by assigning to every variable T_1, T_2, ..., T_n, a column of the orthogonal matrix. For example, comparing Table 1.1 and Table 1.2, we can represent the information of the experiments simply as shown in Design(I) of Table 1.3.

It is clear that if T_1 lies on column v_1 and T_2 lies on column v_2, then the interaction of these, represented by I_{12}, must lie on the column corresponding to the product of columns v_1 and v_2, namely column v_{12}. Therefore, all variables and their interactions must correspond to some basic or non-basic columns. In general, there are more variables and interactions

Expts.	v_{avg}	v_1	v_2	v_3	v_{12}	v_{23}	v_{31}	v_{123}
Design(I)		T_1	T_2	T_3	T_4	T_5		
Interactions		I_{24}	I_{14}	I_{25}	I_{12}	I_{23}	I_{13}	I_{15}
			I_{35}				I_{45}	I_{34}
Design(II)		T_1	T_2	T_3		T_4		T_5
Interactions		I_{45}	I_{34}	I_{24}	I_{12}	I_{23}	I_{13}	I_{14}
					I_{35}	I_{15}	I_{25}	
Design(III)	T_4	T_1	T_2	T_3				T_5
Interactions		I_{14}	I_{24}	I_{34}	I_{12}	I_{23}	I_{13}	I_{45}
					I_{35}	I_{15}	I_{25}	

Table 1.3: Changes in Confounding Patterns

than there are columns. Hence, each column of the matrix may be associated with more than one variable or interaction. All variables and interactions lying on one column are said to be **confounded** with each other. All variables and interactions which are confounded with each other are indistinguishable from each other, with respect to the given experiment.

The orthogonal matrix used to design a 2^{n-k} factorial experiment in n variables has $n-k$ basic columns and $2^{n-k}-(n-k)$ non-basic columns. The design of experiments corresponds to assigning $n-k$ variables to the basic columns and the remaining k variables to the non-basic columns. The assignment of these non-basic variables govern the overall confounding pattern of the design. A set of variables lying on a set of basic columns are called a set of **basic variables** and the other variables are called **non-basic variables.**

Example 1-1

- Suppose a process **P** depends on 5 variables T_1, ..., T_5. A 2^{5-2} Fractional Factorial experiment is to be designed to study the process. As explained above, this is equivalent to assigning 3 variables to a set of basic columns and the other 2 variables to any non-basic columns. If T_1, T_2 and T_3 are assigned to to v_1, v_2 and v_3 respectively, then T_4 and T_5 can be assigned to any of the non-basic columns. Different choices of non-basic columns will lead to different confounding patterns.

  In Design(I) of Table 1.3, T_4 and T_5 are assigned to columns v_{12} and v_{23} respectively. The corresponding positions of the second order interactions are shown in the Table. When the assignments of T_4 and T_5 are changed to columns v_{23} and v_{123} respectively, the confounding pattern of the design changes as shown in Design(II) of Table 1.3. Design(III) is an example in which T_4 and T_5 are assigned to v_{avg} and v_{123} respectively. Such a design is not very desirable as it confounds the average effect with the main effect of T_4. Hence, both these important effects cannot be determined.

Design(I) and Design(II) have resolution III because there are second order interactions which confound with the variables but there are no variables which confound with another variable or the average column. Design(III) has resolution I because T_4 confounds with the average column - which can be considered as a zero order interaction.

In this example only T_4 and T_5 have been shifted keeping the variables T_1, T_2 and T_3 fixed. If the basic variables are shifted from one column to another along with the non-basic variables there are problems caused due to rearrangement of rows. This issue is discussed in greater detail in Appendix B. ■

1.4.4 Shifting with respect to a Columns

This report deals with sequential experimentation. Using sequential experimentation it is possible to shift the variables and interactions to different columns of the design matrix and reduce confounding. In order to mathematically define shifting, we introduce the concept of *Shifting with respect to a Column.* A variable (or interaction) T_i is said to have shift with respect to column v_s if and only if the old column of T_i, v_x, and the new column of T_i, v_y, are such that $v_s = v_x \cdot v_y$.

1.4.5 Fold-Over Designs

Given a block of 2^{n-k} factorial experiments, if the signs of all the elements of this block are reversed, a fold-over design is obtained. This technique is originally due to Box and Wilson [6].

If the first block has resolution III, then it can be shown that the fold-over design together with the first block results in an factorial design of resolution IV, that is one in which none of the second-order interactions are confounded with the variables. Therefore, using such a design it is possible to determine the effect of all the variables independent of the effect of any second-order interactions.

Chapter 2

Sequential Experimental Design

In this chapter the concept of sequential design of experiments is introduced. Section 1 deals with the non-sequential approach to the DOE and the problems with it. In Section 2 the overall sequential approach is outlined. Section 3 lists the assumptions made on the manufacturing models.

2.1 Non-Sequential Approach to DOE

The concept of Sequential Experimental Design is not completely understood especially when the variables are at three and higher levels. Therefore, most of the commercial software available for factorial DOE support non-sequential experimental design[1]. As shown

[1]A few of the commercial software products surveyed (PC-QPI and RS/Discover) do have sequential design features but they do not generate optimal experiments based on previous results. Rather the user has to design the experiments and the software analyzes the results. Also, these commercial software products do not support sequential block analysis.

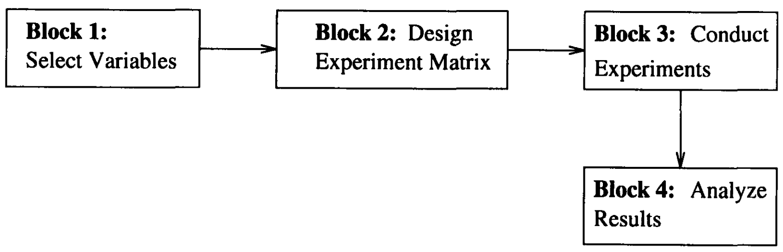

Block 1: Select Variables
Block 2: Design Experiment Matrix
Block 3: Conduct Experiments
Block 4: Analyze Results

Figure 2-1: Non-Sequential Approach to Experimental Design

in Figure 2-1 the non-sequential approach to DOE can be divided into the following steps:

1. Selecting variables and potential variable interactions which are perceived as being most important in determining the performance of a given process.

2. Given the number of variables, designing a set of experiments which is to be performed.

3. Conducting the experiments and obtaining the results, i.e. values of the output parameters to be optimized, such as strength, variance, etc.

4. Analyzing the results to obtain the coefficients of the variables and their interactions in the mathematical expression Equation 1.6 predicting the output parameters.

5. Using these coefficients to suggest new settings of the control variables, T, to improve the process.

In order to use non-sequential experimental design, the experimenter has to guess the probable interactions. The commercial DOE software can be used only after the list of probable interactions has been obtained. Some of the main features currently available in commercial DOE software include the following:

- Designing factorial experiments based on various criteria of optimality such as D-optimality [2] [2].

- Analyzing results using different techniques. Some of the commonly used include:

    1. Analysis of Variance (Anova)

    2. Normal Probability Plots

    3. Bayesian Estimation

Despite these advancements the commercial DOE software products are limited in their application. Some of the main problems are:

- It is not possible to suggest a general design methodology which can be applied to all processes. The concept of DOE depends a lot on the specific process to be optimized and therefore a design that may be good for one process might not be so for another.

[2] A design matrix is said to be D-optimal with respect to a set of matrices if it has the maximum determinant in the set.

- The experimenter has to guess the interactions. In order to be safe it is necessary to guess very conservatively. Thus, the effects of a much large set of interactions has to be determined than is actually necessary. This results in the use of more experiments than necessary.

- Irrespective of the sophistication of the design and analysis procedures, one cannot overcome the fundamental limitations of confounding. That is if one variable/ interaction is confounded with another, then the effects of each cannot be evaluated separately irrespective of the nature of the analysis.

- Typically, there is a limit to an experimenter's knowledge about the process he or she wishes to optimize. Hence it is often true that having performed the experiment some of the following situations might arise.

    1. Some of the columns of the test matrix which were assigned to variable/ interaction(s) which the experimenter thought were unimportant were found to have large coefficients.
    2. The results of the experiments suggests the possible presence of interaction(s) which are confounded with other variables/ interactions and hence could not be evaluated.
    3. Only a few variables and interactions are found to be important.

Hence the experimenter might be forced to ask: *WHAT NEXT???* This is exactly the question that we hope to address in the course of this thesis.

2.2 Sequential Design Philosophy

In this project we aim to address the issue of DOE in a systematic manner. Given a process which needs to be optimized the experimenter should start by studying its physics. Based on this, a group of variables and interactions should be selected. The software will then use this information and will design a matrix which has minimum possible confounding between these variables and interactions. After conducting the experiments and analyzing the results it may happen that the experimenter is not completely satisfied due to one or more of the reasons stated in Section 2.1. Thus in order to proceed the software needs to consider all the possible conditions for which an experimenter may not be satisfied with the results of the first set of experiments and suggest acceptable solutions for each of these conditions.

It is difficult to determine which set of experiments the experimenter should perform, in the most general sense. The choice of experiments depends not only on the specific questions which the experimenter wishes to answer but also on the nature of the process at hand. There are many factors which need to be considered.

- The level of noise in the experiments.
- The number of experiments that the experimenter wishes to perform.
- The accuracy required in estimating the parameters.

Fractional Factorial Designs have very interesting and important properties and are very popular in experimental design techniques.

- The experiments are easy to perform.
- The analysis of the results yield uncorrelated estimates of the effects of the variables and the interactions.
- Factorial designs support sequential experimentation.
- They can be applied to a large class of manufacturing processes.

Given these advantages, it is desirable to design a new fractional factorial experiment. So the question is how to design a new block of experiments which will supplement the old block of experiments and enable the experimenter to resolve many of the problems stated above? The Sequential Block Design and the One-at-a-Time Design are developed with this perspective. They are briefly described below.

2.2.1 Sequential Block Design

In this methodology first a fold-over block of experiments is designed. Since the design used is at least of resolution IV, the effects of the variables can be determined independent of the effects of second order interactions. Based on the assumptions on the models discussed in Section 2.3, a list of probable significant interactions is formed. There may be confounding between these interactions.

With the help of the Half-Block Design and the Full-Block Designs discussed in Chapter 3, an optimal set of experiments is obtained which 'best' disentangles the confounding interactions. Hence, the new set of experiments allows the determination of the effects of the confounding interactions. It may happen that there are still a few interactions which remain confounded, but the optimal set of experiments ensures that these are relatively less important. If desired, a second optimal design may be used to determine the effects of these interactions. Hence using this method we can sequentially sort out the confounding between interactions. Sequential block designs are discussed in detail in Chapter 3.

2.2.2 One-at-a-Time Design

In most manufacturing processes the number of significant variables and interactions affecting the quality is usually small. Suppose a fold-over block of experiments has been conducted. It is often reasonable to assume that at most one variable or interaction is significant in any column of the design matrix.

Therefore, once the effects of the columns have been determined, only the significant variable or interaction in each column needs to be determined. This is done by first making a hypothesis about the significant variable or interaction in each column. Based on this hypothesis an optimal operating point is computed. The actual output of the plant at that operating point is obtained. The actual output is compared with the predicted output and the hypothesis is verified. If the hypothesis fails, a new hypothesis is proposed.

There are several advantages of this approach. One-at-a-Time designs are discussed in detail in Chapter 5.

2.3 Assumptions on Manufacturing Models

In this report we assume that the manufacturing systems that we deal with satisfy the following assumptions. In case there are any deviations they will be noted in the report.

A1 - Sparsity-of-Effect Principle: Most of the systems are dominated by the effects of the variables and the low-order interactions. Most of the higher order interactions are negligible. This assumption will also be referred to as the *Sparsity Principle* in the report.

This assumption is widely used and is supported by experienced practitioners of DOE.

A2 - Minimal Complexity of Models: Most systems are governed by interactions that are important whenever the main effects of some of their variables are important too. This assumption will be called the *Simplicity Principle* in the report.

A3 - Uncorrelated Noise Distribution: The noises in the different experiments are uncorrelated and are approximately gaussian distributed with zero mean and constant variance. This justifies the use of least-square techniques in estimating the manufacturing models.

A4 - Sequential Experimentation: It is possible to combine the results of two or more fractional factorial experiments to assemble sequentially a larger design to better estimate the effects of the significant variables and interactions.

Chapter 3

Sequential Block Design

This chapter discusses the concept of Sequential Experiments using Block Designs. Section 1 discusses the need for such a methodology. Section 2 deals with the initial steps required to do sequential experimental design. Section 3 discusses Half-Block Designs. A detailed example is given to help the reader understand the technique. Section 4 discusses Full-Block Designs and Fold-over Designs. Section 5 illustrates a method for completing an incomplete block of experiments. The mathematical proofs of the procedures are given in the Appendices.

3.1 Need for Sequential Block Design

Consider a situation in which a manufacturer, inexperienced in DOE, desires to use experimental design to improve the quality of the product. Before starting experimentation, the manufacturer must do the following:

- Choose variables and their operating ranges.
- Guess the important interactions affecting the response.
- Choose a response or quality variable which really provides useful information about the process under study.

It is not desirable to do only one block of experiments or a large first block of experiments since:

1. The levels of the variables may be incorrectly chosen, making the effects of some of the variables dominate the results.

2. Since the significant variables and interactions are not known *a priori*, it is advantageous to use the information from the first block of experiments to design subsequent ones.

3. There are always constraints of time, funds etc. on the total number of experiments that can be performed. Therefore, it is desirable to conduct most experiments when there is more information available about the process.

It is usually recommended that the experimenter invest no more than 25 percent of the available resources in the first block of experiments [17]. Therefore, it is necessary to use an experimental design technique which supports sequential experimentation. Hence, fractional factorial designs are often used. A detailed description of factorial designs can be found in several books including [9], [17], [8]. In this report the reader is assumed to be relatively familiar with the basic concepts of fractional factorial designs.

3.2 Sequential Design Methodology

Given a process **P** which depends on n variables $T_1,\ldots,T_n$, a complete factorial design requires 2^n experiments to determine the effects of all the variables and interactions. Even for moderate values of n, the complete factorial design requires an infeasible number of experiments to be performed. For example, for n equal to 8, 9 and 10, the number of experiments required is 256, 512 and 1024 respectively.

Most manufacturing systems satisfy the assumptions given in Section 2.3 and therefore require far fewer experiments to completely determine the significant variables and interactions. The basic steps involved in sequential block design are described below.

Step 1: Designing First Block of Experiments

We advocate that the first block of experiments should be a fold-over design. To obtain a fractional factorial design which is also a fold-over design, first a 2^m factorial matrix is designed, where m is the smallest integer for which $2^m \geq n$. The n variables are then assigned to unique columns of this matrix. The 2^m factorial matrix is now folded i.e. a matrix is formed whose elements are obtained by changing the sign of the elements of the factorial matrix. The matrix is appended to the original factorial matrix and the new matrix of 2^{m+1} experiments is the desired fold-over design.

Variables & Interactions		T_1	T_2	T_3	T_4	I_{12} I_{34}	I_{23} I_{14}	I_{31} I_{24}
Expts.		v_1	v_2	v_3	v_{123}	v_{12}	v_{23}	v_{31}
1	B	+	+	+	+	+	+	+
2	L	+	+	-	-	+	-	-
3	K	+	-	+	-	-	-	+
4	1	+	-	-	+	+	+	-
1'	F	-	-	-	-	+	+	+
2'	O	-	-	+	+	+	-	-
3'	L	-	+	-	+	-	-	+
4'	D	-	+	+	-	-	+	-

Table 3.1: Fold-over Design Matrix for Example 3-1

Example 3-1

- Consider a process **P** which depends on 4 variables T_1, T_2, T_3 and T_4. To obtain a fold-over design, a 2^3 factorial experiment needs to be designed. First a 2^2 factorial matrix is designed as shown in Block 1 of Table 3.1. The four variables are assigned to the four columns of this matrix. Then the matrix is folded over and appended to Block 1. The overall matrix corresponds to the desired fold-over design. Note that none of the variables are confounded with any second order interaction. ■

Step 2: Analyzing the Results

After conducting the fold-over experiments, the coefficients of the columns of the orthogonal matrix are obtained. The fold-over design has resolution IV. Hence, if it is assumed that the third and higher order interaction are insignificant, the estimates of the main-effects of the variables are the coefficients of the columns on which they lie. Using the analysis techniques described in the next chapter, a list of significant variables is obtained.

Step 3: Determining Probable Interactions

On the basis of the *Minimal Complexity of Model* assumption, a list of all second order interactions containing at least one significant variable is made. From this list, the interactions which lie on columns with significant coefficients are selected. This group of interactions are called the *probable interactions.* By assumption, all the true interactions must lie in this group. It may happen that there may be many probable interactions which are confounded with each other and hence their effects cannot be determined from the results of the first block of experiments.

Step 4: Resolving Confounding between Interactions

In case there is confounding between the probable interactions it should be possible to conduct more experiments to unconfound them and determine their separate effects. If we view the variables and interactions as 'sitting' on the columns of the orthogonal matrix, the DOE techniques correspond to 'placing' the variables on the columns of the matrix. Once the variables have been assigned to the columns of the matrix, the positions of the interactions get fixed.

Hence, in order to unconfound the interactions, the variables need to be shifted from one column to another with the help of more experiments. In order to unconfound the interactions, we must exactly understand the mechanics by which the variables can be shifted from their original positions. Once the mechanics is understood it is possible to search through the design and determine the columns to which the variables can be shifted to achieve maximum unconfounding.

Shifting a variable from one column to another is equivalent to designing an experiment with a different confounding pattern. This idea is discussed in more detail in Appendix B. Given that the first block of 2^{n-k} factorial experiments has been conducted it is desired that the second block of experiments leads to a fractional factorial design with a different confounding pattern. There are essentially two options.

1. If possible, replace some of the rows of the first block of experiments by those of a second block, and create a distinct fractional factorial design.

2. If possible, append the rows of the first block of experiments to the rows of a second block of experiments to create a larger fractional factorial design having a different confounding pattern.

Both these options are possible. The technique based on option 1) is called the Half-Block design and the one based on option 2) is called the Full-Block design. These designs are described in the next two sections.

3.3 Half-Block Designs

The focus of this section is to study the problem of unconfounding probable interactions. It is assumed that a 2^{n-k} factorial experiment has been conducted on the process and the effects of the variables are known. On the basis of this, a list of probable interactions is obtained. The effects of all the probable interactions are known, except for those which are confounded with each other.

We need to determine the effects of the confounded probable interactions. Therefore, a fractional factorial experiment has to be designed which has a different confounding pattern. In order to save resources it is natural to inquire if it is possible to design a new factorial experiment in which many of the experiments of the first block are similar, rather than one in which all the 2^{n-k} experiments are different from those of the first block. This issue has been discussed in Appendix B. The main result is given below. The readers interested in the proof can refer to Appendix B.

Result:

Given a block of 2^{n-k} factorial experiments in n variables,

1. *At least 2^{n-k-1} experiments of the block need to be changed to obtain a new block of factorial experiments with a different confounding pattern.*

2. *New blocks of factorial experiments which can be obtained by doing an additional set of 2^{n-k-1} experiments, can be generated by shifting the variables with respect to every column of the design matrix.*

3.3.1 Algorithm to Determine the Optimal Half-Block Design

To begin with, we need to assign each of the probable interactions a weight based on the significance of its variables and the coefficient of its column. That is, an interaction of two important variables should be weighted more than one in which only one of the variables is significant. Also, an interaction lying on a column with a large coefficient should be weighted more.

In this report, the weight of an interaction is obtained by taking the absolute value of the products of the coefficients of its column and those of its variables. There are other acceptable means of assigning the weights. Further work could be done to examine them and determine if some are better than others.

The goodness of a design k is evaluated on the basis of four quantities:

1. Number of columns in which the variables confound with probable interactions, Nvi_k.

2. Sum of the weights of the interactions confounded in 1., Wvi_k.

3. Number of columns in which the probable interactions are confounded with each other, Nii_k.

4. Sum of the weights of the interactions confounded in 3., Wii_k.

For any two designs, design k and design l, design k is said to be better than design l if $Nvi_k < Nvi_l$. If $Nvi_k = Nvi_l$, then design k is better than design l if $Wvi_k < Wvi_l$. If terms 1. and 2. are equal for both the designs, then design k is better than design l if $Nii_k < Nii_l$. Eventually if terms 1., 2., and 3. are equal, then design k is better than design l if $Wii_k < Wii_l$.
This method of comparing different designs ensures that:

- Designs with less confounding between variables and interactions are preferred.

- Designs which have confounding between probable interactions with high weights are avoided.

- A design with no confounding is preferred over all other designs.

This method of comparison has been found to yield good results. Nevertheless, there are other measures which would be acceptable too. Further research should be done to determine the existence of an 'optimal' comparison method which performs best over the ensemble of manufacturing processes.

Given a block of experiments, the variables are shifted to with respect to different columns and different designs are obtained. The search procedure follows the Tree Search Algorithm. Each time a better design is found it is called the current optimum design and is stored in memory. The search procedure stops when either all four quantities given above reduce to zero, or if no design is found which is better than the current optimum design. The Tree Search algorithm ensures that all acceptable sets of 2^{n-k-1} experiments are searched.

The results given in Appendix B greatly reduce the search. Shifting any one variable completely determines the permissible columns for the placement of all the other variables. That is, once any one of the variables has been shifted, the other $n-1$ variables can each be placed in only 2 of the 2^{n-k} columns. Whenever the variables are placed in the permissible

Experiment:	1	2	3	4	1'	2'	3'	4'
Result:	16.21	8.41	23.64	15.35	11.72	-6.22	16.57	0.84

Table 3.2: Results of Fold-over Design of Example 3-2

Column:	v_{avg}	v_1	v_2	v_3	v_{123}	v_{12}	v_{23}	v_{13}
Variable		T_1	T_2	T_3	T_4	I_{12}	I_{23}	I_{13}
& Interaction:						I_{34}	I_{14}	I_{24}
Coefficient:	10.84	5.11	-0.33	-2.22	-0.31	-3.31	0.24	6.19

Table 3.3: Analysis of Design of Example 3-2

columns, there exists a block of 2^{n-k-1} experiments which along with 2^{n-k-1} experiments of the first block gives a new 2^{n-k} factorial design.

Once the variables have been placed in the permissible columns, the procedure for obtaining the new experiments is simple. The new design is generated from the old one by shifting some of the variables with respect to a column of the old design as described in Appendix B. The new block of experiments are the 2^{n-k-1} rows of the new design matrix in which the elements of this column are -1. The rows for which the elements of the column are +1 are common to both the new and the old block of experiments and therefore these experiments need not be done again. Once the 2^{n-k-1} new experiments have been conducted, the new matrix can be formed and the effects of the probable interactions can be determined.

Example 3-2

- Consider the process given in Example 3-1. Suppose that the true expression for quality is given by,

$$y = 10.6 + 5.2T_1 - 2.5T_3 - 3.1I_{12} + 6.1I_{13} \tag{3.1}$$

The first block of experiments is the fold-over design of Table 3.1. The results obtained are shown in Table 3.2. The coefficients of the columns obtained from these results are shown in Table 3.3.

From Table 3.3 it is clear that only columns v_{avg}, v_1, v_3, v_{13} and v_{12} are significant.

Experiment	T_1	T_2	T_3	T_4	Result
1"	-	-	-	+	10.31
2"	-	-	+	-	-6.49
3"	-	+	-	-	17.33
4"	-	+	+	+	-0.36

Table 3.4: Half-Block Experiments and Results of Example 3-2

Column:	v_{avg}	v_1	v_2	v_3	v_{123}	v_{12}	v_{23}	v_{13}
Variable		T_1	T_2	T_3	I_{14}	I_{12}	T_4	I_{13}
& Interaction:			I_{34}	I_{24}		I_{23}		
Coefficient:	10.57	5.37	-0.17	-2.32	0.07	-3.46	-0.15	6.31

Table 3.5: Analysis of New Design of Example 3-2

Therefore, the significant variables are T_1 and T_3. The possible significant interactions are: I_{12}, I_{13}, I_{14}, I_{23} and I_{34}. Of these, only I_{12}, I_{34} and I_{13} lie on significant columns and constitute the set of probable interactions.

Since it is assumed that third and higher order interactions are negligible, the coefficients of the columns v_{avg}, v_1 and v_3 correspond to the effects of the constant, T_1 and T_3 respectively. Since I_{24} is assumed negligible, the coefficient of v_{13} is an estimate of the effect of I_{13}. The effects of probable interactions I_{12} and I_{34} cannot be determined separately as the interactions are confounded with each other. Therefore, we need to design a Half-Block experiment to unconfound the effects of I_{12} and I_{34}. The Half-Block algorithm suggests shifting T_4 from column v_{123} to v_{23}, that is, shifting T_4 with respect to column v_1. Thus, the experiments in which the elements of column v_1 are '-' need to be done. The sign of T_4 is reversed in each of these experiments. The Half-Block experiments are shown in Table 3.4. The results of theses experiments are given in the same table.

The experiments of Table 3.4 along with Experiments 1, 2, 3 and 4 of the the first block give a new orthogonal matrix which has a different confounding pattern. The confounding pattern and the result of the analysis of the new matrix is shown in Table 3.5.

From Table 3.5 it is observed that each column has only one significant variable or probable interaction. Thus, the effect of each of them can be determined. Since the column v_2 in Table 3.5 has a non-significant coefficient, it implies that I_{34} is not significant. The coefficients of columns of I_{13} and I_{12} have significant coefficients which agree well in the result of the analysis of the old and the new design matrix. Hence I_{13} and I_{12} are significant interactions.

The results of the analysis of the two designs can be improved by taking the average of the corresponding effects in Table 3.3 and Table 3.5. Dropping the non-significant effects, the model of the plant is given by:

$$y = 10.71 + 5.24T_1 - 2.27T_3 - 3.39I_{12} + 6.25I_{13} \tag{3.2}$$

Equation 3.1 and Equation 3.2 agree quite well. Hence the Half-Block methodology has been effectively used to estimate the model of this process.

In this example it is not possible to determine the optimal operating point of the process before obtaining the model of the process because the conditions cited in Section 5.5 are not satisfied.

The interactions I_{14} and I_{23} were dropped form the list of probable interactions because they were located on a non-significant column. If these interactions are significant the coefficients of the columns in Table 3.3 and Table 3.5 will not be consistent. For example, if I_{23} is significant, the coefficients of the column I_{12} would not be similar in both Table 3.3 and Table 3.5 since it includes the effect of I_{23} in Table 3.5. This procedure can be used to verify the process model. ■

3.3.2 Advantages and Disadvantages of Half-Block Designs

The Half-Block designs procedure described in this section is suitable for a large number of situations.

1. If the number of confounded probable interactions is small.

2. If the first block of experiments is large. The size of the design matrix increases exponentially with the number of variables, n. Even in the worst case the number of probable interactions (of second order) increase with n^2. Therefore, as the number of variables increase, the number of columns increases and it becomes easier to

disentangle the probable interactions using Half-Block designs.

3. Given a block of 2^{n-k} factorial experiments, a Half-Block design uses 2^{n-k-1} experiments of this block to forms a new block of 2^{n-k} factorial experiments. The analysis of the new block yields uncorrelated estimates of the effects of the unconfounded variables and interactions.

There are a few limitations of this design technique.

1. When a Half-Block design strategy is applied to a block of 2^{n-k} factorial experiments, 2^{n-k-1} experiments are determined which form a new block of 2^{n-k} factorial experiments with a group of 2^{n-k-1} experiments from the first block. The new block has a different confounding pattern. It is possible to estimate the effects of the variables and the interactions by analyzing all the $2^{n-k} + 2^{n-k-1}$ experiments which have been conducted. The estimates obtained from this analysis will be correlated with each other because the columns of the regression matrix are not orthogonal to each other.

   Instead, if only the experiments of the new block of 2^{n-k} factorial experiments are analyzed, the regression matrix is orthogonal and the estimates will be uncorrelated. These estimates will have a larger variance because the number of experiments used in the analysis is reduced. Thus there is a trade-off between smaller variance and uncorrelatedness of the estimates.

2. If the design is highly confounded, it may not be possible to unconfound all the probable interactions using Half-Block designs.

   This limitation is due to the fact that Half-Block designs can only restructure the confounding pattern and not decrease the confounding. The optimal restructuring ensures that the probable interactions get unconfounded at the expense of the non-significant interactions. Therefore, the only manner in which the overall confounding can be reduced is to design a larger matrix which is less confounded. This is the issue of the next section.

3.4 Full-Block Designs

This section deals with the theory and application of Full-Block Designs. The Half-Block designs described in the last section are very useful but there may be situations where, having conducted a 2^{n-k} factorial experiment it may happen that:

1. The number of confounded probable interactions is large.

2. Uncorrelated estimates of low variance are required.

3. Resources permit conducting more than a Half-Block Design.

In such cases it is more desirable to have a design technique which simultaneously uses all the experiments to determine the model of the process. The use of all the experiments is advantageous on two counts. One, the larger design matrix leads to less confounding and two, the estimates obtained from analysis have a smaller variance.

The basic problem can thus be stated as follows. Having done a block of 2^{n-k} factorial experiments on a process, which is the next block of experiments that need to be performed so that the analysis of both blocks 'best' unconfounds the probable interactions? This question is the focus of Appendix C. The main result is stated below. The proof and an illustrative example is given in Appendix C.

Result:
Given a block of 2^{n-k} factorial experiments in n variables,

1. *A new block of 2^{n-k} experiments forms a block of 2^{n-k+1} factorial experiments along with the original block only if both the new and the original block have the same confounding pattern.*

2. *There are 2^k-1 possible blocks of 2^{n-k} experiments which can form a block of 2^{n-k+1} factorial experiments with the original block. The new blocks can be generated by reversing the signs of the columns of the non-basic variables in the original block.*

3.4.1 Algorithm to Determine the Optimal Full-Block Design

In the Full-Block design algorithm, different blocks of 2^{n-k} experiments are generated by reversing the signs of all the possible subsets of the non-basic variables. Each block is unique. The confounding pattern of the overall design is determined for each of these

Experiment:	1	2	3	4	1'	2'	3'	4'
Result:	18.59	18.43	13.89	14.60	2.81	-7.18	18.05	8.58

Table 3.6: Results of Fold-over Design of Example 3-3

Column:	v_{avg}	v_1	v_2	v_3	v_{123}	v_{12}	v_{23}	v_{13}
Variable & Interaction:		T_1	T_2	T_3	T_4	I_{12} I_{34}	I_{23} I_{14}	I_{13} I_{24}
Coefficient:	10.97	5.41	4.94	-2.50	0.04	-2.81	0.17	2.36

Table 3.7: Analysis of Design of Example 3-3

blocks. The Tree Search algorithm is used to determine the optimal new design using the criterion described in the previous section.

The results of Appendix C lead to a considerable simplification in the search procedure. The Tree Search algorithm ensures that all 2^k-1 block designs, which can be used with the first block to give a 2^{n-k+1} Fractional Design, are searched. Hence the search gives the true optimum design.

The new block of experiments to be performed corresponds to the old design in which the signs of certain columns of the non-basic variables are reversed in accordance with the result of the search.

Once the new block of experiments have been carried out, the new design can be appended to the old one. The overall design corresponds to a 2^{n-k+1} factorial design. The algorithm used to generate the new block of experiments guarantees that the overall design matrix has optimal confounding between the probable interactions.

Example 3-3

- Suppose the process in Example 3-1 is given by,

$$y = 10.6 + 5.2T_1 + 4.9T_2 - 2.5T_3 - 3.1I_{12} + 6.1I_{13} - 3.7I_{24} \tag{3.3}$$

Let the first block of experiments be the fold-over design of Table 3.1. The results of these experiments are given in Table 3.6. The coefficients of the columns of the matrix are given in Table 3.7. It is observed that the variables T_1, T_2 and T_3 are significant. According to our assumptions the possible significant interactions are I_{12}, I_{13}, I_{14},

Experiment	T_1	T_2	T_3	T_4	Result
1	+	+	+	-	24.76
2	+	+	-	+	11.11
3	+	-	+	+	21.72
4	+	-	-	-	6.58
5	-	-	-	+	9.99
6	-	-	+	-	-14.78
7	-	+	-	-	25.29
8	-	+	+	+	1.14

Table 3.8: Full-Block Experiments for Example 3-3

Column:	v_{avg}	v_1	v_2	v_3	v_4	v_{12}	v_{13}	v_{14}
Var.& Int.:		T_1	T_2	T_3	T_4	I_{12}	I_{13}	I_{14}
Coefficient:	10.85	5.36	4.90	-2.51	0.15	-2.88	6.04	0.14

Column:	v_{23}	v_{24}	v_{34}	v_{123}	v_{124}	v_{134}	v_{234}	v_{1234}
Var.& Int.:	I_{23}	I_{24}	I_{34}	I_{123}	I_{124}	I_{134}	I_{234}	I_{1234}
Coefficient:	0.03	-3.67	-0.11	0.01	0.07	0.05	0.05	0.12

Table 3.9: Analysis of New Design of Example 3-3

I_{23}, I_{24} and I_{34}. But the column of v_{23} is not significant. This implies that I_{23} and I_{14} are not significant [1]. Thus there are 4 probable interactions I_{12}, I_{34}, I_{13} and I_{24}. The first two and the last two probable interactions are confounded with each other.

When a Half-Block design algorithm is applied to this problem, it is found that there exists no Half-Block design which can unconfound all the probable interactions. Thus we have to resort to a Full-Block design. The Full-Block design algorithm suggests reversing the sign of the column of T_4 keeping the other columns unchanged. Table 3.8 gives the Full-Block experiment. The results of the experiments are also given in the same table.

The results of the analysis of all the 2^4 factorial experiments are given in Table 3.9. Using Table 3.9 and selecting the significant terms, the estimate of the model is given by:

$$y = 10.85 + 5.36T_1 + 4.90T_2 - 2.51T_3 - 2.88I_{12} + 6.04I_{13} - 3.67I_{24} \tag{3.4}$$

This estimate compares very well with the true plant model,

$$y = 10.6 + 5.2T_1 + 4.9T_2 - 2.5T_3 - 3.1I_{12} + 6.1I_{13} - 3.7I_{24} \tag{3.5}$$

Notice that if our assumption that I_{23} and I_{14} are not significant were wrong, the coefficients of the column on which they lie would be significantly different in the results of Table 3.9 and Table 3.7. Comparing the two results would permit the estimation of the effects of these interactions. ■

3.5 Completing Block Designs

This section deals with the issue of completing an incomplete block of fractional factorial experiments. Suppose a block of 2^{n-k} factorial experiment has been conducted followed by a Half-Block of experiments. The Half-Block has the property that it forms a 2^{n-k} orthogonal matrix with 2^{n-k-1} experiments of the first block of experiments. There may be situations where, after conducting a Half-Block, it is desired that a set of experiments is found which

[1]There is a rare possibility that I_{23} and I_{14} have effects which are similar in magnitude but opposite in sign and therefore cancel each other. A check for this possibility is discussed later.

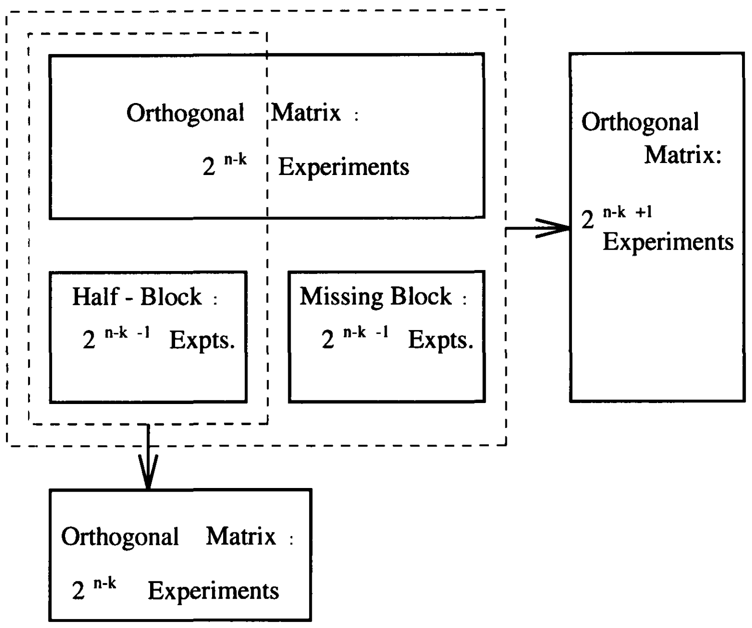


Figure 3-1: Pictorial Representation of Completing Block Design

together with all the experiments conducted previously, determine a orthogonal matrix of size 2^{n-k+1}. This idea is represented in Figure 3-1. The procedure for completing a block is equivalent to finding the missing block of 2^{n-k-1} experiments shown in Figure 3-1.

The concept of completing blocks is discussed in detail in Appendix D. The procedure is demonstrated with an illustrative example. The general procedure for completing blocks is given below.

Procedure for Determining the Missing Block:

Given an incomplete block of m ($2^{n-k} < m < 2^{n-k+1}$) experiments of a block of 2^{n-k+1} factorial experiments,

Step 1. Determine the basic variables and the confounding pattern of the incomplete block.

Step 2. Check which of the 2^{n-k+1} combinations of +1 and -1 are missing from the columns of the basic variables. There will be $2^{n-k+1}-m$ such combinations.

Step 3. For each of the combinations determined in Step 2, determine the entries of the non-basic variables of the matrix using the confounding pattern of the incomplete block.

Chapter 4

Analysis of Sequential Experiments

This chapter deals with the analysis techniques used to analyze sequential experiments. Section 1 deals with classical techniques of analysis. Section 2 discusses the issue of closed loop techniques for determining interactions.

4.1 Classical Analysis Techniques

In this section we deal with the classical analysis techniques that have been added to the MIT computer software to analyze sequential block designs. These techniques are well known and are discussed very briefly.

4.1.1 Analysis of Variance (Anova)

It is probably the most important technique used in statistical inference. The name is derived from a partitioning of the total variability of the experimental results into component parts. For a process **P**, dependent on n variables T_1, ..., T_n, if Y_i $(i = 1, \ldots, N)$ is the result of the ith experiment and $\bar{Y}$ is the mean of the results, then the total sum of squares

$$SS_T = \sum_{i=1}^{N} (Y_i - \bar{Y})^2 \tag{4.1}$$

is used as a measure of the overall variability of the results. Suppose the process is modeled as,

$$Y = \beta_0 + \beta_1 T_1 + \beta_2 T_2 + \ldots + \beta_n T_n + \beta_{12} I_{12} + \ldots + \beta_{ij} I_{ij} + \ldots + \varepsilon \tag{4.2}$$

where the effects β_x are determined using the least-square technique, then it can be shown that

$$SS_T = \sum_i {\beta_i}^2 + \sum_{i,j} {\beta_{ij}}^2 + \sum_i {\varepsilon_i}^2 \tag{4.3}$$

The last term in Equation 4.3 is called the sum of squares of error, SS_E, that is,

$$SS_E = \sum_i {\varepsilon_i}^2 \tag{4.4}$$

If there are a terms in the model of the plant shown in Equation 4.2, then SS_T and SS_E have $N-1$ and $N-a$ degrees of freedom, respectively. The F-ratio F_x, for a coefficient β_x, is given by

$$F_x = \frac{{\beta_x}^2}{SS_E/(N-a)} \tag{4.5}$$

and it represents the measure of significance of the coefficient. If it is assumed that the noise has a gaussian distribution then F_x has an F-distribution. If F_x is large then the variable/ interaction of the coefficient is significant.

A more detailed discussion on Anova can be found in many books including [17],[8].

4.1.2 Normal Probability Plots

This method of analysis attributed to Daniel [7] provides a simple way to determine significant variables and interactions. Given the result of a factorial experiments, the coefficients of the columns of the orthogonal matrix are calculated using the Least Squares technique.

In order to construct a normal probability plot[1], the N coefficients are arranged in increasing order and the kth of these ordered coefficients is plotted versus the cumulative probability point $P_k = (k - 1/2)/N$ on the normal probability paper.

The coefficients which contain only the noise effects tend to be normally distributed with a zero mean and a common variance. Thus they lie along a straight line on the plot. The slope of this line is the estimate of the common variance.

[1]A normal probability plot is a plot on a normal probability paper, a graph paper in which the ordinate has been scaled so that the cumulative normal distribution plots as a straight line.

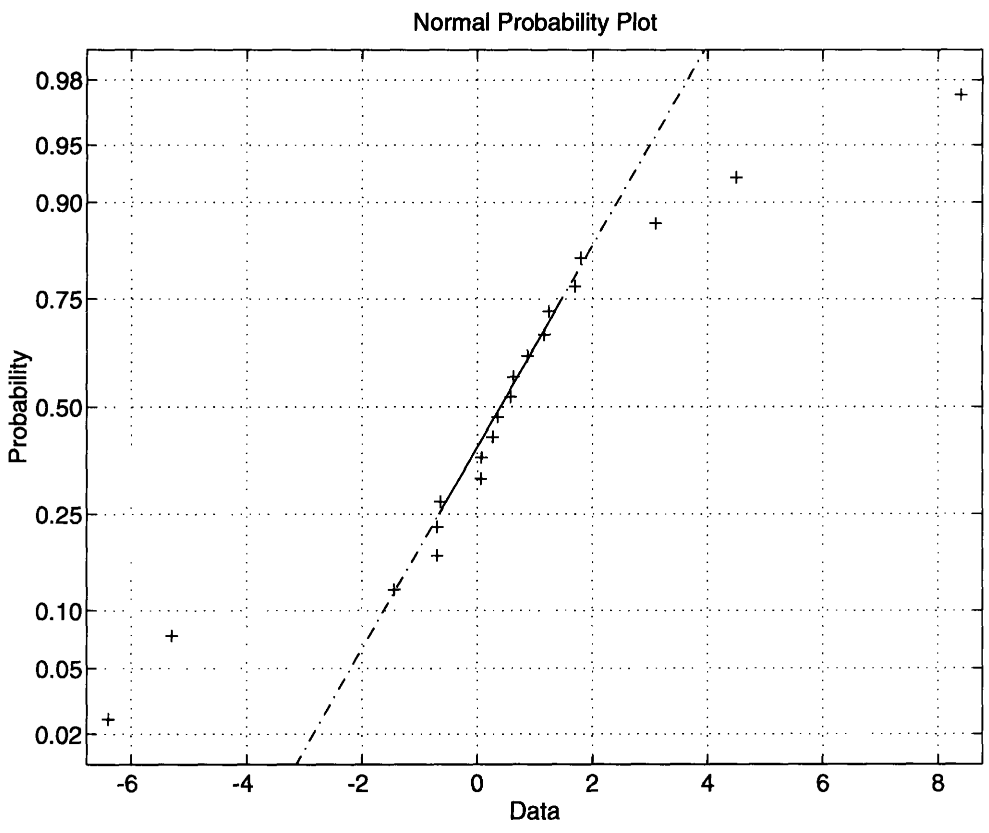


Figure 4-1: Normal Probability Plot

On the other hand, the coefficients which contain the effects of the dominant variables will typically have non-zero means and lie significantly away from the straight line. Thus the normal probability plot can be used to determine these dominant coefficients.

Example 4-1

- A group of 20 random numbers are generated on the computer. Of these 20 numbers, 15 are normally distributed with zero mean and unit variance. The remaining 5 numbers have unit variance but non-zero means. As seen from Figure 4-1 the numbers with zero means tend to lie on a straight line and can be distinguished from the other numbers with non-zero means. ■

4.1.3 Lenth's Algorithm for Determining Significant Coefficients

In this section we describe the Lenth's algorithm [15] which has been used to determine the significant coefficients in this report. The advantages of this algorithm are that it is quick and easy to incorporate in computer software, and it does not require the user to use any judgment in determining the significant coefficients. The algorithm is described below.

Consider a process on which a 2^{n-k} factorial design is conducted and the results of the experiments are analyzed. Let the coefficients of the design matrix be denoted by $\alpha_1, \ldots, \alpha_m$ ($m = 2^{n-k}$). If the noise in the process is assumed to be uncorrelated and gaussian distributed, then the coefficients will have a normal distribution. Let α_i be independently distributed with mean γ_i and variance σ^2. Let,

$$s_o = 1.5 \times \overset{median}{\scriptstyle i} \ |\alpha_i| \tag{4.6}$$

The *pseudo standard error (PSE)* is defined as,

$$PSE = 1.5 \times \underset{|\alpha_i| < 2.5 \cdot s_o}{median} |\alpha_i| \tag{4.7}$$

It has been shown in [15] that when the assumption of *Sparsity Principle* (Section 2.3) is valid, PSE is a fairly good estimate of σ. Once the estimate of σ is known, the coefficient α_i is significant if,

$$|\alpha_i| > t_{.975;m/3} \times PSE \tag{4.8}$$

where $t_{.975;m/3}$ is the .975th percentile of a t-distribution of $m/3$ degrees of freedom. The range of values of $t_{.975;m/3}$ varies from 3.76 for $m = 8$ to 1.99 for $m = 256$.

4.2 Closed Loop Technique for Determining Interactions

In most manufacturing processes the third and higher order interactions are negligible. Suppose a fold-over experiment has been conducted and analyzed. The significant columns can be determined by using any of the techniques discussed in the previous section. Since the fold-over design has resolution of IV, the coefficients of the columns with variables are the estimates of the effects of the variables. The group of variables lying on significant columns form the group of significant variables.

Thus, only the effects of the interactions remain to be determined. In case there is confounding, the assumption of *Simplicity Principle* discussed in Section sec-ass, helps reduce the total number of interactions that need to be analyzed. The group of all the interactions which satisfy the assumption and lie on columns with significant coefficients are called *probable interactions* and have been discussed in Chapter 3.

Thus further experimentation is needed to determine the effects of the confounded probable interactions and to determine which of them are significant. The Half-Block and the Full-Block designs of Chapter 3 were developed with this goal.

It is possible to view the problem of determining the significant probable interactions from a different perspective. Instead of determining the effects of all the probable interactions, we can hypothesize that a subset of the probable interactions are significant and then design experiments to check this hypothesis. This approach is developed in Chapter 5.

Chapter 5

One-at-a-Time Designs

The sequential block design strategy was discussed in Chapter 3. In this chapter we propose an alternative One-at-a-Time Design strategy to determine the unknown probable interactions. Section 1 deals with the need to have such a method and the kinds of applications where it will be useful. Section 2 outlines the algorithm and illustrates it with examples. Section 3 presents a mathematical justification of this approach. The advantages of the One-at-a-Time approach are discussed in Section 4. Section 5 deals with the concept of Partial Optimization.

5.1 Need for One-at-a-Time Design Strategy

Consider a manufacturing process for which the assumptions of *Sparsity Principle, Simplicity Principle* and *Uncorrelated Noise Distribution* discussed in Section 2.3 hold. Let this process depend on n variables $T_1, \ldots, T_n$. Suppose a 2^{n-k} factorial fold-over experiment is conducted on this process and the significant variables and probable interactions are determined. In many processes the number of significant variables and interactions are small and it is reasonable to assume that *each significant column has only one significant variable or interaction lying on it.*

Therefore, under these assumptions, once a 2^{n-k} factorial experiment is analyzed,

1. Significant variables and their effects are known.

2. Unconfounded probable interactions and their effects are known.

3. *The effects of the confounded interactions are known although the interactions need*

to be determined.

If the number of confounding probable interactions is small there should be no need to design either a Half-Block or a Full-Block experiment to determine the true interactions. Instead it should be possible to determine the interactions with only a few more experiments. In the next section we describe such a technique.

5.2 One-at-a-Time Design Strategy

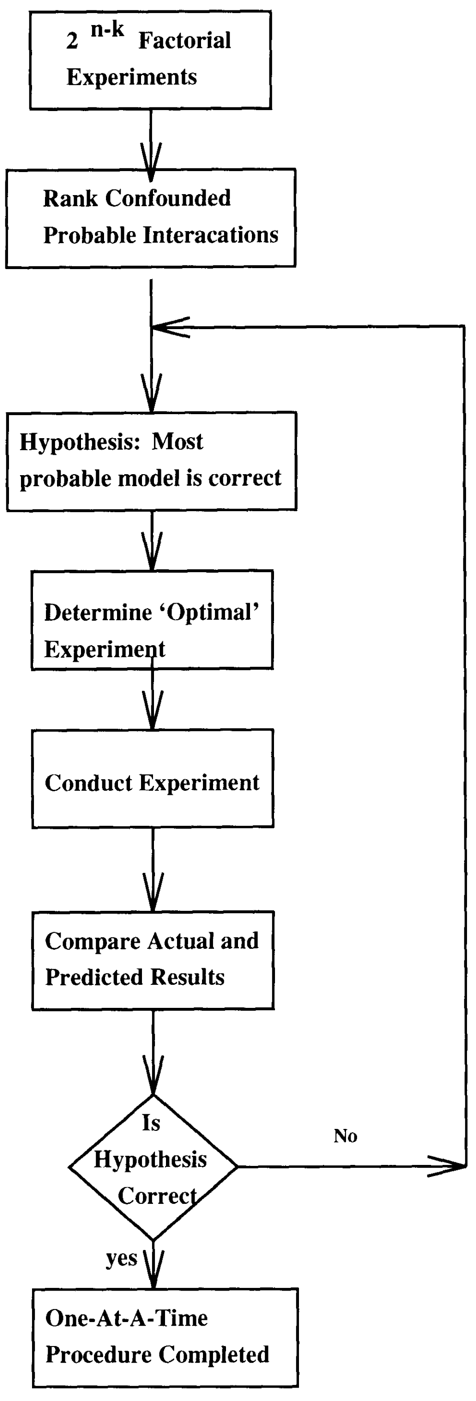
Consider the example of the previous section in which a 2^{n-k} factorial experiment is conducted and the significant variables and probable interactions are determined. Let m columns of the orthogonal matrix have more than one probable interaction and let the number of probable interactions on each of these columns be denoted by n_1, n_2, ..., n_m. If it is assumed that each significant column has only one significant variable or interaction then there are a total of N_T possible models of the process, where N_T is given by:

$$N_T = n_1 \cdot n_2 \ldots n_{m-1} \cdot n_m \tag{5.1}$$

That is, there are N_T models which are completely indistinguishable with respect to the first block of 2^{n-k} experiments. Of these N_T possible models there is only one model which corresponds to the true plant. Hence, under the above assumptions, the problem of determining the plant is equivalent to determining the correct model from a set of N_T possible models.

To begin with, the confounded probable interactions are rank ordered on the basis of the significance of their constituent variables and the significance of the coefficients of the columns on which they lie. In this report the absolute value of the product of the effects of the variables and coefficient of the column of the interaction is used as the weight. Therefore, for a given column, usually interactions in which both the variables are significant, are ranked above those which have only one significant variable.

The highest ranked probable interaction from every column is selected and a hypothesis is made that these interactions correspond to the true interactions. On the basis of this hypothesis an optimal experiment is designed which would give the highest quality if the hypothesis is correct. If this experiment has already been carried out then the next high-



2 n-k Factorial Experiments
Rank Confounded Probable Interacations
Hypothesis: Most probable model is correct
Determine 'Optimal' Experiment
Conduct Experiment
Compare Actual and Predicted Results
Is Hypothesis Correct
No
yes
One-At-A-Time Procedure Completed

Figure 5-1: One-at-a-Time Design Strategy

est ranked interactions are selected and an optimal experiment is designed. The optimal experiment is conducted on the actual plant.

Since the new experiment is different from those carried out earlier, not all the N_T possible models predict the same result. The models are ranked in increasing order of their prediction errors, that is the squared difference between the actual and predicted results. If two or more models give the same least prediction error, then the one which has higher ranked interactions is ranked higher. The highest ranked model, that is the model yielding the lowest prediction error, is selected as the next hypothesis. Once again an experiment is designed which would give the highest quality if the new hypothesis is correct. If this experiment has been conducted before, then an experiment is designed such that it is the optimal experiment for the model which gives the next least prediction error.

Having conducted the second experiment the prediction errors are computed for all the models and the next hypothesis is made. The same recursive algorithm is implemented until the hypothesis is verified by the results. Once the hypothesis is verified, ideally no further experimentation is required since any further experimentation will lead to the selection of the same hypothesis over and over again. In presence of noise it is advisable to do a few more experiments to ensure that the results are consistent. The outline of the above procedure is shown in Figure 5-1.

Example 5-1

- Consider a manufacturing process which depends on 7 variables T_1, …,T_7 and the true model of the process is given by

$$y = 20.5 + 3.8T_1 + 4.1T_2 + 9.0T_4 - 3.7I_{34} + 2.3I_{13} - 6.1I_{15} \qquad (5.2)$$

  A 2^{7-3} factorial fold-over experiment is conducted and the results are analyzed. The confounding pattern is shown in Table 5.1. From Table 5.2 (a) it is observed that variables T_1, T_2 and T_4 are significant. Also, the columns 9, 10 and 12 are significant. The probable interactions lying on the significant columns are:

  - Column 9: I_{12} and I_{34}
  - Column 10: I_{24} and I_{13}
  - Column 12: I_{15} and I_{26}

Column:	1	2	3	4	5	6	7	8
	v_{avg}	v_1	v_2	v_3	v_{123}	v_4	v_{124}	v_{134}
Variable:		T_1	T_2	T_3	T_4	T_5	T_6	T_7

Column:	9	10	11	12	13	14	15	16
	v_{12}	v_{13}	v_{23}	v_{14}	v_{24}	v_{34}	v_{1234}	v_{234}
Interaction:	I_{12} I_{34} I_{56}	I_{13} I_{24} I_{57}	I_{14} I_{67} I_{23}	I_{15} I_{26} I_{37}	I_{16} I_{47} I_{25}	I_{17} I_{46} I_{35}	I_{45} I_{27} I_{36}	

Table 5.1: Confounding pattern of the 2^{7-3} factorial experiments of Examples 5-1, 5-2 and 5-3

Thus, assuming that only one of the interactions in each of the columns is significant results in $N_T = 8$.

The initial guess of the process model is shown in Table 5.2(a). On the basis of this hypothesis, Experiment 1 is designed. The model shown in Table 5.2(b) gives the least prediction error and is the next hypothesis. The most significant interaction, I_{15}, is determined. Experiment 2 is designed to optimize the current hypothesis. Notice that it is not possible to discriminate between the pairs of interactions (I_{12}, I_{34}) and (I_{24}, I_{13}) using the first 2 experiments because both the interactions in the pairs take the same value in both the experiments.

Since the hypothesis has not changed after the Experiment 2, Experiment 3 is designed to optimize the next best hypothesis, that is, the model which gives the next least prediction error. Experiment 4 is designed in the same manner as Experiment 3. By now the true process model has been determined. Interestingly, from Table 5.2(f), we notice that the true optimal operating point of the process, Experiment 5, is the same as Experiment 3. This means that the true optimal point of the process was determined prior to the determination of the true process model. ■

Example 5-2

- Consider a process which is dependent on 7 variables T_1, ..., T_n. A 2^{7-2} factorial experiment is conducted on the process. The experiments are the same as those of Example 5-1. The confounding pattern of the block is shown in Table 5.1. The results of the One-at-a-Time strategy are given in Table 5.3.

```
--------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)         True Cf.  True(V/I)
--------------------------------------------------
 1      20.04   Avg.               20.50    Avg.
 2       3.69   T1                  3.80    T1
 3       3.87   T2                  4.10    T2
 4      -0.08   T3                          -----
 5       9.26   T4                  9.00    T4
 6      -0.05   T5                          -----
 7       1.09   T6                          -----
 8      -0.52   T7                          -----
 9      -4.09   I12                -3.70    I34
10       1.67   I24                 2.30    I13
11       0.05   -----                       -----
12      -5.77   I26                -6.10    I15
13       0.67   -----                       -----
14       0.71   -----                       -----
15       0.52   -----                       -----
16       0.34   -----                       -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Initial Guess of Process Model.



--------------------------------------------------
```

(a) Results after Experiment 0

```
--------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)         True Cf.  True(V/I)
--------------------------------------------------
 1      20.04   Avg.               20.50    Avg.
 2       3.69   T1                  3.80    T1
 3       3.87   T2                  4.10    T2
 4      -0.08   T3                          -----
 5       9.26   T4                  9.00    T4
 6      -0.05   T5                          -----
 7       1.09   T6                          -----
 8      -0.52   T7                          -----
 9      -4.09   I12                -3.70    I34
10       1.67   I24                 2.30    I13
11       0.05   -----                       -----
12      -5.77   I15                -6.10    I15
13       0.67   -----                       -----
14       0.71   -----                       -----
15       0.52   -----                       -----
16       0.34   -----                       -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.    Experiment       Prediction       Observation
--------------------------------------------------
 1.      - + - + - - -     29.04             31.54

RMS Pred. Error:            2.50

--------------------------------------------------
```

(b) Results after Experiment 1

```
--------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)         True Cf.  True(V/I)
--------------------------------------------------
 1      20.04   Avg.               20.50    Avg.
 2       3.69   T1                  3.80    T1
 3       3.87   T2                  4.10    T2
 4      -0.08   T3                          -----
 5       9.26   T4                  9.00    T4
 6      -0.05   T5                          -----
 7       1.09   T6                          -----
 8      -0.52   T7                          -----
 9      -4.09   I12                -3.70    I34
10       1.67   I24                 2.30    I13
11       0.05   -----                       -----
12      -5.77   I15                -6.10    I15
13       0.67   -----                       -----
14       0.71   -----                       -----
15       0.52   -----                       -----
16       0.34   -----                       -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.    Experiment       Prediction       Observation
--------------------------------------------------
 1.      - + - + - - -     29.04             31.54
 2.      - + - + + + -     42.68             42.51


RMS Pred. Error:            1.77

--------------------------------------------------
```

(c) Results after Experiment 2

```
--------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)         True Cf.  True(V/I)
--------------------------------------------------
 1      20.04   Avg.               20.50    Avg.
 2       3.69   T1                  3.80    T1
 3       3.87   T2                  4.10    T2
 4      -0.08   T3                          -----
 5       9.26   T4                  9.00    T4
 6      -0.05   T5                          -----
 7       1.09   T6                          -----
 8      -0.52   T7                          -----
 9      -4.09   I12                -3.70    I34
10       1.67   I24                 2.30    I13
11       0.05   -----                       -----
12      -5.77   I15                -6.10    I15
13       0.67   -----                       -----
14       0.71   -----                       -----
15       0.52   -----                       -----
16       0.34   -----                       -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.    Experiment       Prediction       Observation
--------------------------------------------------
 1.      - + - + - - -     29.04             31.54
 2.      - + - + + + -     42.68             42.51
 3.      + + - + - + -     41.96             43.86

RMS Pred. Error:           1.81

--------------------------------------------------
```

(d) Results after Experiment 3

Table 5.2: Results of One-at-a-Time Designs of Example 5-1

```
---------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)      True Cf. True(V/I)
---------------------------------------------------
 1     20.04    Avg.            20.50   Avg.
 2      3.69    T1               3.80   T1
 3      3.87    T2               4.10   T2
 4     -0.08    T3                      -----
 5      9.26    T4               9.00   T4
 6     -0.05    T5                      -----
 7      1.09    T6                      -----
 8     -0.52    T7                      -----
 9     -4.09    I34             -3.70   I34
10      1.67    I13              2.30   I13
11      0.05    -----                   -----
12     -5.77    I15             -6.10   I15
13      0.67    -----                   -----
14      0.71    -----                   -----
15      0.52    -----                   -----
16      0.34    -----                   -----

---------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment      Prediction      Observation
---------------------------------------------------
 1.     - + - + - - -    29.04             31.54
 2.     - + - + + + -    42.68             42.51
 3.     + + - + - + -    46.80             43.86
 4.     - + + + + + -    30.98             29.36


RMS Pred. Error:          2.09

---------------------------------------------------
                (e) Results after Experiment 4
```

```
---------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)      True Cf. True(V/I)
---------------------------------------------------
 1     20.04    Avg.            20.50   Avg.
 2      3.69    T1               3.80   T1
 3      3.87    T2               4.10   T2
 4     -0.08    T3                      -----
 5      9.26    T4               9.00   T4
 6     -0.05    T5                      -----
 7      1.09    T6                      -----
 8     -0.52    T7                      -----
 9     -4.09    I34             -3.70   I34
10      1.67    I13              2.30   I13
11      0.05    -----                   -----
12     -5.77    I15             -6.10   I15
13      0.67    -----                   -----
14      0.71    -----                   -----
15      0.52    -----                   -----
16      0.34    -----                   -----

---------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment      Prediction      Observation
---------------------------------------------------
 1.     - + - + - - -    29.04             31.54
 2.     - + - + + + -    42.68             42.51
 3.     + + - + - + -    46.80             43.86
 4.     - + + + + + -    30.98             29.36
 5.     + + - + - + -    46.80             43.64

RMS Pred. Error:          2.35

---------------------------------------------------
                (f) Results after Experiment 5
```

Table 5.2: Results of One-at-a-Time Designs of Example 5-1

The example indicates that the more significant interactions are determined before the others. Also, the example demonstrates that the procedure determines the optimal operating point of the process before it determines the true process model.■

Example 5-3

- This example is a variant of Example 5-2 in which the crucial assumption that there is only one significant variable or interaction per column of the design matrix, has been violated. In Example 5-2, Column 9 has one significant interaction I_{34}. In this example the true plant model has two significant interactions I_{56} and I_{34} on Column 9. The coefficients of the two interactions are chosen to be effectively the same as that of I_{34} in Example 5-2. Therefore, the results of the first block of experiments are identical in both these examples. When the One-at-a-Time strategy is applied to the plant the output behavior is very interesting. The results are shown in Table 5.4. In the first few experiments, neither of the two interactions in Column 9 are selected by the algorithm. But after Experiment 4, I_{34} is selected. Although the true model of the plant cannot be determined, the experiments tend to improve the output quality. Also, the interactions in the rest of the columns are determined correctly.

  As expected, the RMS prediction error for this example is much larger than that for the previous examples. The large RMS prediction error suggests that there may be something wrong with the plant model that has been determined. ■

5.3 Mathematical Justification

In this section we give a justification of the One-at-a-Time strategy introduced in this chapter. We show that under certain assumptions, the model which gives the least prediction error is the Maximum Likelihood estimate of the the model of the plant.

Claim:
Suppose T One-at-a-Time experiments are conducted and the results are Y_1, ..., Y_T. If the noise in these experiments is independent and gaussian distributed with zero mean and constant variance λ, then the model which minimizes the squared prediction error is the Maximum Likelihood Estimate of the plant.

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)        True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.               22.50  Avg.
 2       5.45  T1                  5.30  T1
 3       6.24  T2                  7.30  T2
 4       2.95  T3                  3.10  T3
 5       0.72  T4                        -----
 6      -4.19  T5                 -4.90  T5
 7      -0.53  T6                        -----
 8       0.31  T7                        -----
 9      -3.54  I12                -3.70  I34
10       2.84  I13                 2.90  I57
11       0.47  -----                     -----
12       0.26  -----                     -----
13       0.26  -----                     -----
14      -4.10  I35                -4.30  I17
15      -0.35  -----                     -----
16      -0.46  -----                     -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Initial Guess of Process Model.



-------------------------------------------------
                (a) Results after Experiment 0
```

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)        True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.               22.50  Avg.
 2       5.45  T1                  5.30  T1
 3       6.24  T2                  7.30  T2
 4       2.95  T3                  3.10  T3
 5       0.72  T4                        -----
 6      -4.19  T5                 -4.90  T5
 7      -0.53  T6                        -----
 8       0.31  T7                        -----
 9      -3.54  I12                -3.70  I34
10       2.84  I57                 2.90  I57
11       0.47  -----                     -----
12       0.26  -----                     -----
13       0.26  -----                     -----
14      -4.10  I17                -4.30  I17
15      -0.35  -----                     -----
16      -0.46  -----                     -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.  Experiment      Prediction      Observation
-------------------------------------------------
 1.    + + + + - - +    32.41            29.42

RMS Pred. Error:         3.00

-------------------------------------------------
                (b) Results after Experiment 1
```

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
-------------------

Col.  Est. Cf.  Pred(V/I)        True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.               22.50  Avg.
 2       5.45  T1                  5.30  T1
 3       6.24  T2                  7.30  T2
 4       2.95  T3                  3.10  T3
 5       0.72  T4                        -----
 6      -4.19  T5                 -4.90  T5
 7      -0.53  T6                        -----
 8       0.31  T7                        -----
 9      -3.54  I12                -3.70  I34
10       2.84  I57                 2.90  I57
11       0.47  -----                     -----
12       0.26  -----                     -----
13       0.26  -----                     -----
14      -4.10  I17                -4.30  I17
15      -0.35  -----                     -----
16      -0.46  -----                     -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.  Experiment      Prediction      Observation
-------------------------------------------------
 1.    + + + + - - +    32.41            29.42
 2.    + + + + - - -    45.67            45.98

RMS Pred. Error:         2.13

-------------------------------------------------
                (c) Results after Experiment 2
```

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
------------------

Col.  Est. Cf.  Pred(V/I)        True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.               22.50  Avg.
 2       5.45  T1                  5.30  T1
 3       6.24  T2                  7.30  T2
 4       2.95  T3                  3.10  T3
 5       0.72  T4                        -----
 6      -4.19  T5                 -4.90  T5
 7      -0.53  T6                        -----
 8       0.31  T7                        -----
 9      -3.54  I56                -3.70  I34
10       2.84  I57                 2.90  I57
11       0.47  -----                     -----
12       0.26  -----                     -----
13       0.26  -----                     -----
14      -4.10  I17                -4.30  I17
15      -0.35  -----                     -----
16      -0.46  -----                     -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.  Experiment      Prediction      Observation
-------------------------------------------------
 1.    + + + + - - +    32.41            29.42
 2.    + + + + - - -    45.67            45.98
 3.    + + + + - + -    51.70            48.82

RMS Pred. Error:         2.41

-------------------------------------------------
                (d) Results after Experiment 3
```

Table 5.3: Results of One-at-a-Time Designs of Example 5-2

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)        True Cf.  True(V/I)
-------------------------------------------------
 1       22.51  Avg.                22.50  Avg.
 2        5.45  T1                   5.30  T1
 3        6.24  T2                   7.30  T2
 4        2.95  T3                   3.10  T3
 5        0.72  T4                         -----
 6       -4.19  T5                  -4.90  T5
 7       -0.53  T6                         -----
 8        0.31  T7                         -----
 9       -3.54  I34                 -3.70  I34
10        2.84  I57                  2.90  I57
11        0.47  -----                      -----
12        0.26  -----                      -----
13        0.26  -----                      -----
14       -4.10  I17                 -4.30  I17
15       -0.35  -----                      -----
16       -0.46  -----                      -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment       Prediction       Observation
-------------------------------------------------
 1.     + + + + - - +     32.41            29.42
 2.     + + + + - - -     45.67            45.98
 3.     + + + + - + -     44.61            48.82
 4.     + + + - - - -     51.32            53.66

RMS Pred. Error:            2.84

-------------------------------------------------
```
(e) Results after Experiment 4

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)        True Cf.  True(V/I)
-------------------------------------------------
 1       22.51  Avg.                22.50  Avg.
 2        5.45  T1                   5.30  T1
 3        6.24  T2                   7.30  T2
 4        2.95  T3                   3.10  T3
 5        0.72  T4                         -----
 6       -4.19  T5                  -4.90  T5
 7       -0.53  T6                         -----
 8        0.31  T7                         -----
 9       -3.54  I34                 -3.70  I34
10        2.84  I57                  2.90  I57
11        0.47  -----                      -----
12        0.26  -----                      -----
13        0.26  -----                      -----
14       -4.10  I17                 -4.30  I17
15       -0.35  -----                      -----
16       -0.46  -----                      -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment       Prediction       Observation
-------------------------------------------------
 1.     + + + + - - +     32.41            29.42
 2.     + + + + - - -     45.67            45.98
 3.     + + + + - + -     44.61            48.82
 4.     + + + - - - -     51.32            53.66
 5.     + + - + - - -     46.85            49.79

RMS Pred. Error:            2.86
-------------------------------------------------
```
(f) Results after Experiment 5

```
-------------------------------------------------
COLUMNWISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)        True Cf.  True(V/I)
-------------------------------------------------
 1       22.51  Avg.                22.50  Avg.
 2        5.45  T1                   5.30  T1
 3        6.24  T2                   7.30  T2
 4        2.95  T3                   3.10  T3
 5        0.72  T4                         -----
 6       -4.19  T5                  -4.90  T5
 7       -0.53  T6                         -----
 8        0.31  T7                         -----
 9       -3.54  I34                 -3.70  I34
10        2.84  I57                  2.90  I57
11        0.47  -----                      -----
12        0.26  -----                      -----
13        0.26  -----                      -----
14       -4.10  I17                 -4.30  I17
15       -0.35  -----                      -----
16       -0.46  -----                      -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment       Prediction       Observation
-------------------------------------------------
 1.     + + + + - - +     32.41            29.42
 2.     + + + + - - -     45.67            45.98
 3.     + + + + - + -     44.61            48.82
 4.     + + + - - - -     51.32            53.66
 5.     + + - + - - -     46.85            49.79
 6.     + + + - - - -     51.32            54.93

RMS Pred. Error:            3.00

-------------------------------------------------
```
(g) Results after Experiment 6

Table 5.3: Results of One-at-a-Time Designs of Example 5-2

```
-------------------------------------------------
COLUMN-WISE  SUMMARY:

-------------------
Col. Est. Cf.  Pred(V/I)      True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I12         4.70, -8.40  I56, I34
10       2.84  I13                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I35               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Initial Guess of Process Model.




-------------------------------------------------
```

(a) Results after Experiment 0

```
-------------------------------------------------
COLUMN-WISE  SUMMARY:
-------------------

Col. Est. Cf.  Pred(V/I)      True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I12         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.   Experiment      Prediction      Observation
-------------------------------------------------
 1.     + + + + - - +    32.41            34.43

RMS Pred. Error:         2.02

-------------------------------------------------
```

(b) Results after Experiment 1

```
-------------------------------------------------
COLUMN-WISE  SUMMARY:
-------------------

Col. Est. Cf.  Pred(V/I)      True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I12         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.   Experiment      Prediction      Observation
-------------------------------------------------
 1.     + + + + - - +    32.41            34.43
 2.     + + + + - - -    45.67            45.89


RMS Pred. Error:         1.44

-------------------------------------------------
```

(c) Results after Experiment 2

```
-------------------------------------------------
COLUMN-WISE  SUMMARY:
-------------------

Col. Est. Cf.  Pred(V/I)      True Cf. True(V/I)
-------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I12         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

-------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
-----------------------

Expt.   Experiment      Prediction      Observation
-------------------------------------------------
 1.     + + + + - - +    32.41            34.43
 2.     + + + + - - -    45.67            45.89
 3.     + + + + - + -    44.61            37.78

RMS Pred. Error:         4.12

-------------------------------------------------
```

(d) Results after Experiment 3

Table 5.4: Results of One-at-a-Time Designs of Example 5-3

```
--------------------------------------------------
COLUMN-WISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)      True Cf.  True(V/I)
--------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I34         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment      Prediction      Observation
--------------------------------------------------
 1.     + + + + - - +   32.41             34.43
 2.     + + + + - - -   45.67             45.89
 3.     + + + + - + -   44.61             37.78
 4.     + + + - - - -   51.32             66.29

RMS Pred. Error:          8.29

--------------------------------------------------
                (e) Results after Experiment 4
```

```
--------------------------------------------------
COLUMN-WISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)      True Cf.  True(V/I)
--------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I34         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment      Prediction      Observation
--------------------------------------------------
 1.     + + + + - - +   32.41             34.43
 2.     + + + + - - -   45.67             45.89
 3.     + + + + - + -   44.61             37.78
 4.     + + + - - - -   51.32             66.29
 5.     + + - + - - -   46.85             59.33

RMS Pred. Error:          9.28
--------------------------------------------------
                (f) Results after Experiment 5
```

```
--------------------------------------------------
COLUMN-WISE  SUMMARY:
--------------------

Col.  Est. Cf.  Pred(V/I)      True Cf.  True(V/I)
--------------------------------------------------
 1      22.51  Avg.              22.50  Avg.
 2       5.45  T1                 5.30  T1
 3       6.24  T2                 7.30  T2
 4       2.95  T3                 3.10  T3
 5       0.72  T4                       -----
 6      -4.19  T5                -4.90  T5
 7      -0.53  T6                       -----
 8       0.31  T7                       -----
 9      -3.54  I34         4.70, -8.40  I56, I34
10       2.84  I57                2.90  I57
11       0.47  -----                    -----
12       0.26  -----                    -----
13       0.26  -----                    -----
14      -4.10  I17               -4.30  I17
15      -0.35  -----                    -----
16      -0.46  -----                    -----

--------------------------------------------------
EXPERIMENT-WISE  SUMMARY:
------------------------

Expt.   Experiment      Prediction      Observation
--------------------------------------------------
 1.     + + + + - - +   32.41             34.43
 2.     + + + + - - -   45.67             45.89
 3.     + + + + - + -   44.61             37.78
 4.     + + + - - - -   51.32             66.29
 5.     + + - + - - -   46.85             59.33
 6.     + + + - - - -   51.32             63.59

RMS Pred. Error:          9.84

--------------------------------------------------
                (g) Results after Experiment 6
```

Table 5.4: Results of One-at-a-Time Designs of Example 5-3

Proof:

Consider the model discussed in the previous section. Let the N_T possible models be denoted by $M_1, \ldots, M_{N_T}$.

In the One-at-a-Time design methodology it is assumed that the coefficients of the variables and interactions have been determined and only the confounded interactions need to be determined.

Given the T observations, let $\bar{Y}_T = [Y_1, \ldots, Y_T]$. The Maximum Likelihood Estimate is defined by,

$$M_{ML}[\bar{Y}_T] = \arg\max_i \ \mathbf{P}(M_i; \bar{Y}_T) \tag{5.3}$$

$$\Longrightarrow M_{ML}[\bar{Y}_T] = \arg\max_i \ \mathbf{P}(M_i) \cdot \mathbf{P}(\bar{Y}_T / M_i) \tag{5.4}$$

In case the probabilities of the different models $\mathbf{P}(M_i)$, are known, Equation 5.4 can be simplified.. In absence of any such information, all models can be assumed to be equally likely. That is, $\mathbf{P}(M_i) = 1/N_T$, and Equation 5.4 reduces to,

$$M_{ML}[\bar{Y}_T] = \arg\max_i \ \mathbf{P}(\bar{Y}_T / M_i) \tag{5.5}$$

Let the predicted result of model M_i for the jth experiment be denoted by $\hat{Y}_{ij}$. Since the noise in the different experiments is assumed to be independent,

$$\mathbf{P}(\bar{Y}_T / M_i) = \prod_{j=1}^{T} f_e(Y_j - \hat{Y}_{ij}) \tag{5.6}$$

where $f_e(\cdot)$ is the probability distribution of the noise corrupting the results. If this distribution is assumed to be gaussian with zero mean and constant variance λ, then

$$\mathbf{P}(\bar{Y}_T / M_i) = \prod_{j=1}^{T} \frac{1}{\sqrt{2\pi\lambda}} \cdot \exp\left(-\frac{1}{2\lambda}(Y_j - \hat{Y}_{ij})^2\right) \tag{5.7}$$

$$= \frac{1}{(2\pi\lambda)^{T/2}} \cdot \exp\left(-\frac{1}{2\lambda}\sum_{j=1}^{T}(Y_j - \hat{Y}_{ij})^2\right) \tag{5.8}$$

Maximizing the likelihood function in Equation 5.5 is the same as maximizing its logarithm. Thus,

$$M_{ML}[\bar{Y}_T] = \arg\max_i \ \log \mathbf{P}(\bar{Y}_T / M_i) \tag{5.9}$$

Using Equation 5.8 in Equation 5.9 we get,

$$\begin{aligned}
M_{ML}[\bar{Y}_T] &= \arg\max_i \; \log\left[\frac{1}{(2\pi\lambda)^{T/2}}\cdot\exp\left(-\frac{1}{2\lambda}\sum_{j=1}^{T}(Y_j-\hat{Y}_{ij})^2\right)\right] && (5.10)\\
&= \arg\max_i \left[-\frac{T}{2}\log 2\pi\lambda-\frac{1}{2\lambda}\cdot\sum_{j=1}^{T}(Y_j-\hat{Y}_{ij})^2\right] && (5.11)\\
&= \arg\min_i \left[\frac{1}{2\lambda}\cdot\sum_{j=1}^{T}(Y_j-\hat{Y}_{ij})^2\right] && (5.12)\\
&= \arg\min_i \left[\sum_{j=1}^{T}(Y_j-\hat{Y}_{ij})^2\right] && (5.13)
\end{aligned}$$

The expression on the right hand side of the equation is the total squared prediction error of model M_i. Thus, the Maximum Likelihood Estimate of the plant, M_{ML}, is one which minimizes the total prediction error. This proves the claim.

The One-at-a-Time strategy selects models based on minimum prediction error. Hence, under the above assumptions, the results obtained using One-at-a-Time strategy converge to the maximum likelihood estimate of the plant. The most critical assumption is that there is only one unknown significant interaction per significant column of the orthogonal matrix.

5.4 Advantages of One-at-a-Time Design Strategy

The One-at-a-Time design strategy can be very useful particularly when the number of confounded probable interactions is small. The main advantages of the One-at-a-Time design strategy are:

1. **Maximum Likelihood Estimate:** The One-at-a-Time strategy uses the minimum prediction error criterion to hypothesize the plant model. This criterion is equivalent to the Maximum Likelihood estimate under the assumptions discussed in the previous section. These assumptions are justified in most situations.

2. **Designing Optimal Experiment:** Given the hypothesized plant model, the One-at-a-Time strategy designs the optimal experiment for this hypothesis, i.e. the experiment which would maximize the quality of the output if the hypothesis is correct. This leads to interesting behavior.

- In case the hypothesis is wrong and some very significant interactions have been guessed wrongly, the optimal experiment for the hypothesis tends to produce a very poor output. Although this experiment is far away from the true optimal, it can be very useful. The prediction errors for such an experiment tend to be very large for the wrong models. Hence, this experiment helps in discriminating between different models.
- If the hypothesis is correct, the optimal experiment will produce a very good quality output. The prediction error of the correct model will be much smaller compared to the wrong model and the hypothesis will be verified.

3. **Validation of Assumptions:** When a block of factorial experiments is analyzed some of the columns have large coefficients which correspond to the effects of significant variables and interactions of the plant. The non-significant coefficients can be used to determine the variance of the plant.

   Suppose that the One-at-a-Time design strategy is applied and the most probable plant model is determined. If it is found that the average prediction error is very large, it could be due to the the violations of one or more assumptions made above. In such a situation it is desirable that sequential block design strategy should be used to determine the correct plant model.

4. **Determination of the Most Significant Interactions:** Suppose that there are two columns, one of which has a large significant coefficient and the other has a small significant coefficient. Suppose that each of these columns contain two probable interactions. It is clear that models which contain the wrong probable interaction in the column with the larger coefficient will give larger prediction errors than models which have the wrong probable interaction in the column with smaller coefficient.

   In the simulations it is observed that after a few One-at-a-Time experiments have been carried out, all models having wrong probable interactions in the significant columns with large coefficients give prediction errors which are more than an order of magnitude greater than the error of the correct model. Hence, the probable interactions lying on such columns can be determined within a few One-at-a-Time experiments and the number of possible models is then considerably reduced.

Column:	v_{avg}	v_1	v_2	v_3	v_{123}	v_{12}	v_{23}	v_{13}
Variable		T_1	T_2	T_3	T_4	I_{12}	I_{23}	I_{13}
& Interaction:						I_{34}	I_{14}	I_{24}
Coefficient:	10.6	7.2	0.1	-6.5	-0.2	-3.1	-0.3	2.3

Table 5.5: Results of Design of Example 5-4

It is shown in the next section that not all the unknown probable interactions need to be determined to determine the optimal operating point of the true plant. Hence, the One-at-a-Time strategy needs to be implemented only until the very significant probable interactions have been determined.

5.5 Partial Optimization

The goal of the DOE procedure is to determine the plant model. But in many cases only the optimal operating point, that is the operating point of the plant which produces the best quality, needs to be determined. Hence, in such cases, the sole purpose of determining the plant model is to determine the optimal operating point. In this section we will demonstrate that it is not always necessary to determine the complete plant model to determine the optimal operating point. It may be possible to determine it only on the basis of the significant variables and the known significant interactions.

Example 5-4

- Consider a process dependent on 4 variables T_1, T_2, T_3 and T_4. Suppose a 2^{4-1} factorial experiment is conducted on this process and the results are as shown in Table 5.5. Using Table 5.5, we can express the model of the process as

$$Y = 10.6 + 7.2T_1 - 6.5T_3 - 3.1 \begin{pmatrix} I_{12} \\ I_{34} \end{pmatrix} + 2.3 \begin{pmatrix} I_{13} \\ I_{24} \end{pmatrix} \tag{5.14}$$

where only one of the interactions expressed in (.) is actually significant. If the terms involving T_1 are arranged together, the model can be expressed as,

$$Y = 10.6 + T_1[7.5 + (-3.1T_2) + (2.3T_3)] + \text{ terms not involving } T_1 \text{ ....} \tag{5.15}$$

The terms within (.) might not be present in the true model. Since, $|7.5| > |-3.1| + |2.3|$, irrespective of which combination of interactions is really important, the quality Y is maximized only if T_1 is set at +1. Therefore the optimal setting of T_1 is +1. Similarly,

$$Y = 10.6 + T_3[-6.5 + (-3.1T_4) + (2.3T_1)] + \text{ terms not involving } T_3 \;.... \tag{5.16}$$

Since, $|-6.5| > |-3.1| + |2.3|$, the optimal setting for T_3 is -1 irrespective of which interactions are really important. It is not possible to determine the optimal settings of T_2 and T_4 since their settings are dependent on the unknown interactions. ■

It is clear from this example that it may be possible to determine the optimal settings of some (if not all) of the variables without knowing the exact plant model. In the above example the optimal settings of the variables T_1 and T_3 could be determined because their coefficients were dominant. The general analysis for such cases is given below.

1. **Dominant Variable:**

   Consider a process in which a 2^{n-k} factorial fold-over experiment has been carried out. The results are analyzed and the model of the process is determined.

$$Y = \beta_0 + \beta_1 T_1 + \ldots + \beta_n T_n + \ldots + \beta_{ij} I_{ij} + \gamma_1 \begin{pmatrix} \hat{I}_{11} \\ \hat{I}_{1p_1} \end{pmatrix} + \ldots + \gamma_m \begin{pmatrix} \hat{I}_{m1} \\ \hat{I}_{mp_m} \end{pmatrix} \tag{5.17}$$

   where β_i is the known coefficient of the ith variable, β_{ij} is the known coefficient of the known probable interaction I_{ij}, and γ_k is the known coefficient of the kth significant column with an unknown probable interaction. The probable interactions lying on the kth column are $\hat{I}_{k1}, \ldots, \hat{I}_{kp_k}$.

   Let Di be the set of all the columns which have one unknown probable interaction of T_i. Namely, if $k \epsilon Di$, then the columns contain $\hat{I}_{i \cdot g(i,k)}$, the interaction between T_i and $T_{g(i,k)}$. The optimal setting of variable T_i can be determined if the expression

$$\beta_i + \sum_{j \neq i} \beta_{ij} T_j + \sum_{k \epsilon Di} \gamma_k T_{g(i,k)} \tag{5.18}$$

   has a known sign irrespective of the unknown setting of the other variables.

   If the setting of all the variables are unknown then the condition in Equation 5.18 can be restated as,

$$|\beta_i| \overset{?}{>} \sum_{j \neq i} |\beta_{ij}| + \sum_{k \epsilon Di} |\gamma_k| \tag{5.19}$$

If Equation 5.19 is true, the optimal setting of T_i is +1 if β_i is positive and is -1 if β_i is negative. Thus if a variable is dominant, its optimal setting can be determined without the complete knowledge of the process.

2. **Dominant Pair of Variables:**

In many situations Equation 5.19 cannot be applied because one or more known significant interactions are large. That is, one or more of the coefficients β_{ij} are large. In such situations it may be possible to determine the optimal settings of variables taken two at a time. This procedure is explained below.

Suppose that β_{ij} is large. In the expression for quality given in Equation 5.17, the terms containing T_i and T_j are separated as follows:

$$\begin{aligned} Y &= T_i[\beta_i + \text{ interactions involving } T_i] \\ &\quad + \beta_{ij} I_{ij} + T_j[\beta_j + \text{ interactions involving } T_j] \\ &\quad + \text{ terms not involving } T_i \text{ and } T_j \end{aligned} \tag{5.20}$$

Suppose that the terms corresponding to the coefficients of T_i and T_j in Equation 5.20 satisfy Equation 5.19. The range of these terms is,

$$R_i \;\epsilon\; [\theta_i^{min}, \theta_i^{max}] = \left[\left(|\beta_i| - \sum_{l \neq i,j} |\beta_{il}| - \sum_{k \epsilon Di} |\gamma_k| \right), \left(|\beta_i| + \sum_{l \neq i,j} |\beta_{il}| + \sum_{k \epsilon Di} |\gamma_k| \right) \right] \tag{5.21}$$

$$R_j \;\epsilon\; [\theta_j^{min}, \theta_j^{max}] = \left[\left(|\beta_j| - \sum_{l \neq j,i} |\beta_{jl}| - \sum_{k \epsilon Dj} |\gamma_k| \right), \left(|\beta_j| + \sum_{l \neq j,i} |\beta_{jl}| + \sum_{k \epsilon Dj} |\gamma_k| \right) \right] \tag{5.22}$$

It can be shown that the optimal settings of T_i and T_j can be determined only if the ranges R_i and R_j do not intersect and if β_{ij} lies in the intervals 1, 3 or 5 of Figure 5-2.

In case any of the above conditions are not satisfied, the optimal setting of T_i and T_j

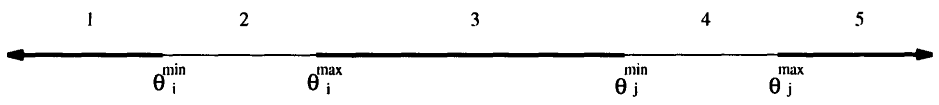


Figure 5-2: Permissible range of values of β_{ij}

cannot be determined. Further experiments are required to determine more unknown probable interactions before the settings of T_i and T_j can be determined.

3. **Dominant Group of Three or more Variables:**
   In principle it is possible to analyze groups of 3 or more variables to determine their optimal setting. But the procedure becomes very complicated. Also the conditions required to be able to determine the settings are fairly strong and will not be satisfied very often.

Thus it is seen that if the plant contains some dominant variables or pairs of variables, then their optimization may be possible without the complete determination of the interactions. The setting of the other variables can be obtained by further experimentation. Usually the variables that cannot be set easily using this technique are ones which are associated with weak interactions and hence they do not affect the quality very much.

Chapter 6

Conclusions and Future Work

In this chapter the basic ideas of the thesis are integrated. Section 1 discusses the Complete DOE Software Tool which is being currently developed at MIT. Section 2 outlines the contributions of the thesis. The possible areas for future research are discussed in Section 3.

6.1 The Complete DOE Software Tool

In this section we show how the different ideas of Block Design and One-at-a-Time Design are being integrated to develop a Complete DOE Software Tool. The underlying motivation for developing such a DOE software tool is that it should aid an inexperienced user in DOE to successfully improve the performance of his/her plant more rapidly than with existing software tools.

Consider the flow-chart shown in Figure 6-1. Given a plant, an experimenter has to select the variables affecting the performance and guess the important interactions. This permits the experimenter to use the past knowledge about the plant which may be very important. Even if some of the interactions are guessed incorrectly, a few more experiments will generally be required to estimate them, but nothing is lost. On the basis of this information, an optimal fold-over factorial block of experiment is designed. Next, the experiment results are analyzed and the effects of the variables and the guessed interactions are determined. The variables with significant effects are used to generate the list of probable interactions - in accordance with the *Simplicity Principle.*

Usually there is confounding within the set of probable interactions and guessed interactions. Therefore further experimentation is required to determine the effects of all these

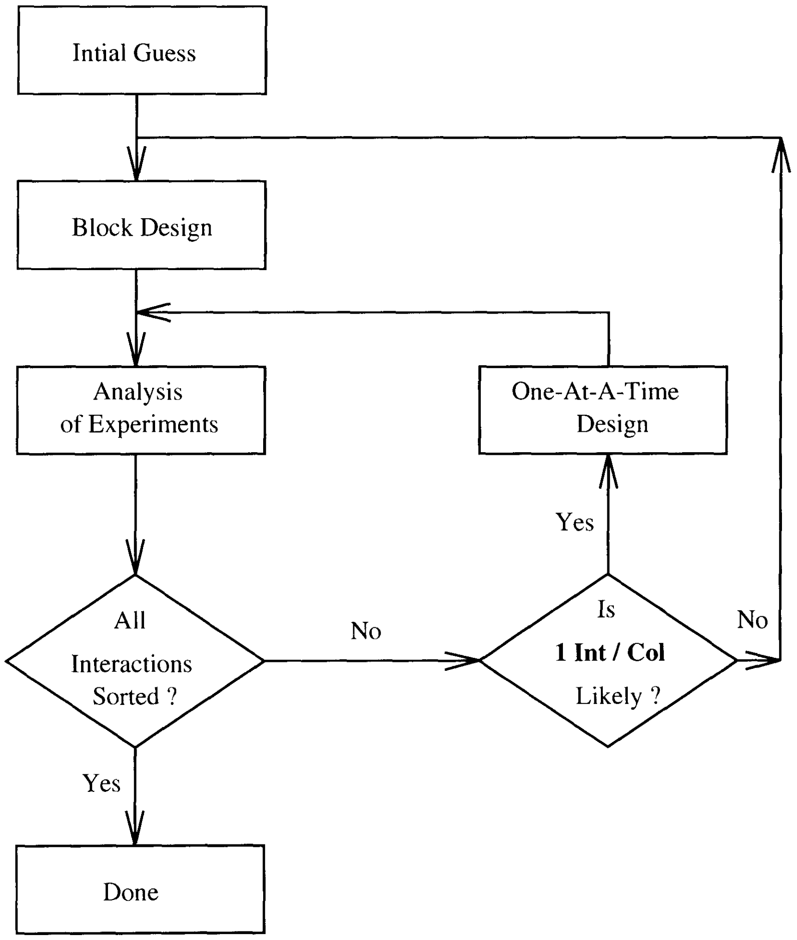


Figure 6-1: Overview of Complete DOE Software Tool

interactions. At this stage there are 2 possibilities.

Case 1: The number of significant columns is small compared to the number of unknown interactions and it is reasonable to assume that there is only one unknown significant interaction per significant column of the design matrix.

Case 2: The number of significant columns is not small and there may be more than one unknown significant interaction per significant column of the design matrix.

If Case 1 is valid for the given plant, the assumptions of the One-at-a-Time Design Strategy discussed in Chapter 5 are satisfied, and the One-at-a-Time Technique can be effectively used to determine the plant model and optimize its performance.

If Case 1 is not valid, the Block Design Technique described in Chapter 3 can be used. With the help of Half-Block or Full-Block Designs the confounding of the interactions can be sorted either completely or to an extent that the One-at-a-Time Design Strategy can be applied.

Presently, we are implementing the Complete DOE Software Tool. Much of the work has been completed. The basic concepts of Block Design and One-at-a-Time Design Strategies have been implemented and tested separately. Also, the analysis tools have been developed. Techniques required to simultaneously analyze block designs with different confounding patterns have been developed and implemented in computer software.

Some of the concepts that can be added to make the software more automated and less dependent on the user are discussed in Section 3.

6.2 Thesis Achievements

The goal of the DOE project at MIT is to automate the entire process of DOE. At the time this thesis began, the goal was well defined but the path to achieve it was a little unclear. During the course of the thesis, the process of sequential DOE has been formulated in a structured fashion. We have identified the conceptual problems that need to be solved before the goal of the Complete DOE Software Tool can be completely realized. Some of these problems have been resolved and implemented. As for the other problems, they have been formulated in a mathematically meaningful manner. Even if these problems are hard to tackle, we know exactly what questions need to be addressed in the overall project.

The contributions of the thesis are enumerated below.

Block Designs: The techniques of Half-Block and Full-Block Designs are implemented in computer software. They have been found to be very useful in resolving confounding interactions.

Completing Block Designs: Algorithm for completing incomplete blocks of experiments have been developed and added to the software.

Block Analysis: The analysis routines of the DOE software have been improved to enable it to simultaneously analyze confounding interactions in different blocks of experiments.

One-at-a-Time Designs: The concept of One-at-a-Time Designs have been developed and verified by simulation results. The results are very encouraging and we believe that the One-at-a-Time Design Strategy can be generalized as explained in the next section.

Complete DOE Software Tool: The basic ideas for the Complete DOE Software Tool are developed and their implementation is being carried out.

6.3 Future Work

The following represents some of the more significant issues that could be added to the computer software to make it more general and useful to various manufacturing processes.

- As seen in Figure 6-1 the Complete DOE Software Tool needs to make a decision whether only one unknown significant interaction per column is likely or not. Presently, this decision is made by the experimenter using the software. It may be possible to determine the probability of there being only one unknown significant interaction per column. Using this probability and the level of risk that the experimenter is willing to take, the computer can make that decision which minimizes the expected value of the overall cost of experimentation.

- Once the Block Design option is selected the experimenter has the following options:

  1. Design a Half-Block of experiments for any of the previous blocks.

2. Design a Full-Block of experiments for any of the previous blocks.
3. Complete any of the previous incomplete blocks.

Therefore as the number of blocks of experiments conducted increases there are more options. If the cost model for the experiments is known, the software could choose between the above options based on:

- Total number of confounded probable interactions.
- Relative unconfounding achieved with each of the options.

- The present software can be used with any plant in which the variables are at 2 levels. The two-level factorial designs have many strong advantages as outlined in Chapter 1. But there may be situations in which the plant intrinsically has one or more of its variables at 3 or more levels. Extension of the software is required to handle such situations.

- It should be possible to extend the ideas of One-at-a-Time Design Strategy to be able to design arbitrary numbers of optimal experiments based on the results of the experiments conducted earlier.

- A Graphical User Interface (GUI) can be added to the software to make its use easier. Also, adding various graphical features can help the experimenter get a better understanding of the results.

Appendix A

Binary Orthogonal Matrices

In this section we enumerate some of the properties of Binary Orthogonal Matrices which are required in the report.

The Binary Orthogonal matrices belong to a general class of matrices called Hadamard Matrices [19], matrices in which each of the elements are either +1 or -1. The most general Binary Orthogonal Matrices are the Plackett-Burman Designs. These designs are based on the Hadamard Matrices of size $m \times m$ where m is a multiple of 4. In the Plackett-Burman Designs all the columns of the matrices are balanced (equal number of +1 and -1 elements) and pairwise orthogonal. When m is not a power of two, these designs have a very complicated alias structure and should be used very carefully. A detailed discussion of these designs is given in [19].

Orthogonal Binary Matrices in which m is a power of two are widely used and have very useful properties. Hence only Binary Orthogonal Matrices in which $m = 2^n$ for some n, are studied in the report. For simplicity of terminology the term *Orthogonal Matrix* is used to denote such a Binary Orthogonal Matrix, as defined below.

A.1 Definitions

Orthogonal Matrix: An Orthogonal Matrix, M, of size $2^n \times 2^n$ is defined as a matrix whose elements are either +1 or -1 and which satisfies the condition $M^T M = 2^n \times$I, where I is an identity matrix of size $2^n \times 2^n$. Table A.1 is an example of an Orthogonal Matrix of size $2^3 \times 2^3$.

Rows	c_{avg}	c_1	c_2	c_3	c_4	c_5	c_6	c_7
1	+	+	+	+	+	+	+	+
2	+	+	+	-	+	-	-	-
3	+	+	-	+	-	-	+	-
4	+	+	-	-	-	+	-	+
5	+	-	+	+	-	+	-	-
6	+	-	+	-	-	-	+	+
7	+	-	-	+	+	-	-	+
8	+	-	-	-	+	+	+	-

Table A.1: 3 × 3 Binary Orthogonal Matrix

Dependent Column: A column c_i is said to be dependent on a set of columns c_j, c_k, ..., c_p if and only if c_i is identically equal to the column obtained by taking an element-by-element product of the columns of c_j, c_k, ..., c_p.

Independent Column: A column c_i is said to be independent of a set of columns c_j, c_k, ..., c_p if and only if c_i is not dependent on any group of columns taken from the set of columns c_j, c_k, ..., c_p.

Independent Set of Columns: A set of columns c_j, c_k, ..., c_p are **independent** if none of the columns within the set is *dependent* on any group of other columns of the set.

Column c_4 in Table A.1 is dependent on columns c_1 and c_2, since column c_4 corresponds to the column formed by the product of the column c_1 and column c_2. Also, columns c_1, c_2 and c_3 form an independent set of columns.

A.2 Properties of Binary Orthogonal Matrices

Given below are some of the properties of an $2^n \times 2^n$ binary orthogonal matrix, M.

1. One of the columns of the orthogonal matrix consists of only +1 elements and is called the **Average Column**. This column is denoted as c_{avg} in Table A.1.

2. Every column other than the Average column has equal numbers of +1 and -1 elements.

3. A product of any two columns of the matrix equals a column of the matrix. That is, the product of column c_i and c_j must equal c_k for some k.

$$c_i \cdot c_j = c_k \tag{A.1}$$

Using the fact that,

$$c_i \cdot c_i = c_{avg} \qquad \forall i \tag{A.2}$$

it can be seen that,

$$\begin{aligned} & c_i \cdot c_j = c_k \\ \Longrightarrow \quad & c_i \cdot c_k = c_j \\ \Longrightarrow \quad & c_j \cdot c_k = c_i \end{aligned}$$

4. Any set of n independent columns can be used to generate the entire matrix. Hence, any set of n independent set of columns completely characterizes the entire matrix. Such a set of columns is defined as a **set of basic columns**. The other columns are called **non-basic** columns. The set of basic columns is not unique.

5. Every $2^n \times 2^n$ binary orthogonal matrix, M, has n basic column and $2^n - n$ non-basic columns.

6. Any set of n basic columns is such that every row of these columns corresponds to a unique combination of +1 and -1. There are two possible options for every element in a row. Therefore, there are a total of 2^n possible combinations, each corresponding to a different row of the orthogonal matrix.

Appendix B

Designing a Half-Block of Experiments

In this section we will try to illustrate the mechanism for shifting variables and interactions from one column to another when an additional block of experiments is carried out to augment the existing one. Let us formulate the problem in a general manner. Given a block of 2^{n-k} factorial experiments, is it possible to replace some of the experiments by other experiments, and obtain a new block of factorial experiments with a different confounding pattern?

Claim:

Given a block of 2^{n-k} factorial experiments in n variables,

1. *At least 2^{n-k-1} experiments of the block need to be changed to obtain a new block of factorial experiments with a different confounding pattern.*

2. *New blocks of factorial experiments which can be obtained by doing an additional set of 2^{n-k-1} new experiments, can be generated by shifting the variables with respect to every column of the design matrix.*

Proof:

Consider a process **P** which depends on n variables $T_1, T_2, \ldots, T_n$. Suppose a 2^{n-k} factorial experiment has been designed on it. Let the design be such that variables $T_1, T_2, \ldots, T_{n-k}$ form a set of basic variables. $T_{n-k+1}, \ldots, T_n$ are the corresponding non-basic variables.

It is desired that a set of experiments be designed which together with a set of experiments from the original block of experiments, determines a new block of 2^{n-k} factorial experiments with a different confounding pattern.

In order to change the confounding pattern, the variables must be shifted to different columns of the design matrix. In Part 1 we show that at least 2^{n-k-1} new experiments are required to produce a new factorial design. In Part 2 an algorithm is developed to generate the different new designs which can be obtained by doing 2^{n-k-1} new experiments.

Part 1:

In a factorial design the columns are balanced and orthogonal to each other. This implies that any two distinct columns of a 2^{n-k} factorial design will agree and disagree in exactly 2^{n-k-1} rows.

To change the confounding pattern at least one variable should be shifted from its existing column to a new column. This requires that an experiment be carried out for every row in which the elements of the existing column and new column differ with each other. There are exactly 2^{n-k-1} rows in which the elements of the two columns differ. Hence, at least 2^{n-k-1} experiments of the original design must be changed to obtain a new one with a different confounding pattern. This proves the first part of the claim.

Part 2:

In Part I it was demonstrated that 2^{n-k-1} experiments can be used to shift one variable from one column to another. Suppose that variable T_i is shifted from column v_i to column v_I by doing 2^{n-k-1} experiments, one for each row in which column v_i and v_I differ.

The question now arises whether it is possible to shift more than one variable using the same experiments. If another variable T_j is to be shifted from v_j to v_J along with T_i, then the columns v_j and v_J must differ in the same rows in which the columns v_i and v_I differ. That is, the product of the columns v_j and v_J must be the same as the product of the column v_i and v_I (the product of two columns is a column which has a +1 element for each row in which the elements of the two columns agree and a -1 element for each row in which the elements of the two columns disagree). That is,

$$v_s = v_i \cdot v_I = v_j \cdot v_J \tag{B.1}$$

Using the notation introduced in Section 1.4.4, we can restate the condition in Equation B.1 as - the variables T_i and T_j can be shifted by doing only 2^{n-k-1} new experiments if and only if they shift with respect to the same column v_s, of the design matrix.

Clearly, any number of variables can be shifted to different columns provided they all shift with respect to the same column of the design matrix. Moreover, the variables can be shifted with respect any column of the design matrix. Therefore, we have shown that by doing 2^{n-k-1} more experiments any of the variables can be shifted with respect to any column of the design matrix.

The new designs can be obtained by shifting all the possible combinations of the variables with respect to every column of the design matrix. This proves the second part of the claim.

Problems caused by rearrangement of rows

Although all the variables can be shifted with respect to any column of the design matrix, it is not necessary that the design obtained are all distinct. In particular, it may happen that given a column with respect to which the variables are being shifted, two different combinations of variables may produce equivalent designs, i.e. identical designs in which the rows are arranged differently. This point is best illustrated with the help of an example. Example B-1 addresses this issue.

A problem with the procedure of shifting of columns, as described above, is that the procedure is negated by changing the order of the rows of the designs. From the properties of binary orthogonal matrices discussed in Appendix A, it is known that the set of basic columns must have rows corresponding to all the permutations of $+$ and $-$. Thus the problem could be resolved if we ensure that any set of basic variables is not shifted. That is, we shift only subsets of non-basic variables.

In the Example B-1, T_1, T_2 and T_3 form a set of basic variables. Hence T_4 and/or T_5 can be shifted by doing 4 more experiments. Also, T_2, T_3 and T_4 form a set of basic variables, so T_1 and/or T_5 can be shifted keeping the other variables fixed. But, T_2, T_3 and T_5 do not form a basic set. Hence we cannot move T_1 and T_4 without encountering the problem of generating equivalent designs.

Therefore, we believe that an algorithm which generates different factorial designs by shifting all possible subsets of non-basic variables with respect to all the columns of the matrix will produce all possible designs where each will be distinct. Although this idea seems to be correct, the proof is not known to the author.

The MIT software currently generates new factorial designs by shifting all combinations of variables.

Example B-1

- Suppose a process **P** depends on variables T_1, T_2, T_3, T_4 and T_5. An initial design of 8 experiments is conducted as shown in Table B.1. The location of the variables and their second order interactions are shown in Table B.2.

  Given the Initial Design of Table B.2 four experiments (Expt. 3, 4, 7 and 8) are designed to shift the variables with respect to column v_2. Table B.1 gives the actual experiment matrix of the designs. In Design(I) T_4 is shifted from v_{13} to v_{123}. The other variables are kept unchanged. In Design(II) both T_1 and T_4 are shifted to columns v_{12} and v_{123} respectively. In Design(III) only T_1 is shifted to v_{12} and the other variables are kept unchanged.

  Examining Table B.2 we expect that the confounding patterns of the designs must be very different. But this is not so. From Table B.1 we observe that Design(I) and Design(III) are essentially equivalent to each other and only the order of the experiments are different. Hence, shifting T_1 to v_{12}, keeping the other variables unchanged, is exactly equivalent to shifting T_4 to v_{123}, keeping T_1, T_2, T_3 and T_5 unchanged.

  In the same way, from Table B.1 it is seen that Design(II) is equivalent to the Initial Design. That is, shifting two variables T_1 and T_4 has not changed anything, except the order of the rows.

  In the Initial Design and in Design(II) the variables T_1, T_2 and T_3 form a set of basic variables. The variables T_4 and T_5 correspond to I_{13} and I_{23} respectively, in both these design. Therefore the Initial Design and in Design(II) have the same confounding pattern and are equivalent. In the same way, in Design(I) and in Design(III) T_1, T_2 and T_3 form a set of basic variables. The variables T_4 and T_5 confound with I_{123} and I_{23} respectively, in both these design. Therefore, Design(I) and Design(III) are equivalent. ■

	Initial Design				
	T_1	T_2	T_3	T_4	T_5
Expt.	v_1	v_2	v_3	v_{13}	v_{23}
1	+	+	+	+	+
2	+	+	-	-	-
3	+	-	+	+	-
4	+	-	-	-	+
5	-	+	+	-	+
6	-	+	-	+	-
7	-	-	+	-	-
8	-	-	-	+	+

	Design(II)				
	T_1	T_2	T_3	T_4	T_5
Expt.	v_{12}	v_2	v_3	v_{123}	v_{23}
1	+	+	+	+	+
2	+	+	-	-	-
3	-	-	+	-	-
4	-	-	-	+	+
5	-	+	+	-	+
6	-	+	-	+	-
7	+	-	+	+	-
8	+	-	-	-	+

	Design(I)				
	T_1	T_2	T_3	T_4	T_5
Expt.	v_1	v_2	v_3	v_{123}	v_{23}
1	+	+	+	+	+
2	+	+	-	-	-
3	+	-	+	-	-
4	+	-	-	+	+
5	-	+	+	-	+
6	-	+	-	+	-
7	-	-	+	+	-
8	-	-	-	-	+

	Design(III)				
	T_1	T_2	T_3	T_4	T_5
Expt.	v_{12}	v_2	v_3	v_{13}	v_{23}
1	+	+	+	+	+
2	+	+	-	-	-
3	-	-	+	+	-
4	-	-	-	-	+
5	-	+	+	-	+
6	-	+	-	+	-
7	+	-	+	-	-
8	+	-	-	+	+

Table B.1: Equivalence of Different Designs

Expts.	v_{avg}	v_1	v_2	v_3	v_{12}	v_{23}	v_{13}	v_{123}
Initial Design		T_1	T_2	T_3		T_5	T_4	
Interactions		I_{34}	I_{35}	I_{25}	I_{12}	I_{23}	I_{13}	I_{15}
				I_{14}	I_{45}			I_{24}
Design(I)		T_1	T_2	T_3		T_5		T_4
Interactions		I_{45}	I_{35}	I_{25}	I_{12}	I_{23}	I_{13}	I_{15}
					I_{34}	I_{14}	I_{24}	
Design(II)			T_2	T_3	T_1	T_5		T_4
Interactions		I_{45}	I_{35}	I_{25}	I_{34}	I_{23}	I_{15}	I_{13}
		I_{12}		I_{14}			I_{24}	
Design(III)			T_2	T_3	T_1	T_5	T_4	
Interactions		I_{12}	I_{35}	I_{25}	I_{45}	I_{14}	I_{15}	I_{13}
		I_{34}				I_{23}		I_{24}

Table B.2: Changes in Confounding Pattern

Appendix C

Designing a Full-Block of Experiments

In Appendix B a technique was demonstrated by which, a block of 2^{n-k-1} experiments could be designed which along with 2^{n-k-1} experiments of the first block of 2^{n-k} factorial experiments, forms a new block of 2^{n-k} factorial experiments. In Section 1 we will demonstrate the technique of selecting a block of 2^{n-k} experiments which along with all the 2^{n-k} experiments of the first block, forms a block of 2^{n-k+1} factorial experiments. In Section 2 we show that the popular fold-over design technique is a special case of the Full-Block design strategy. The method of generating the fold-over design using the Full-Block design strategy is demonstrated.

C.1 Full-Block Design Strategy

Claim:

Given a block of 2^{n-k} factorial experiments in n variables,

1. *A new block of 2^{n-k} experiments forms a block of 2^{n-k+1} factorial experiments along with the original block only if both the new and the original block have the same confounding pattern.*

2. *There are 2^k-1 possible blocks of 2^{n-k} experiments which can form a block of 2^{n-k+1} factorial experiments with the original block. The new blocks can be generated by reversing the signs of the columns of the non-basic variables in the original block.*

Variable	T_1	T_2	$\dots$	T_{n-k}	T_{n-k+1}	$\dots$	T_n
Original Design	v_1	v_2	$\dots$	v_{n-k}	$v_{p(n-k+1)}$	$\dots$	$v_{p(n)}$
New Design	u_1	u_2	$\dots$	u_{n-k}	u_{n-k+1}	$\dots$	u_n

Table C.1: Columns of the variables in the original and the new design

Proof:

Factorial designs are of size 2^n. Hence it is clear that the number of experiments required to be added to a block of 2^{n-k} factorial experiments to form a larger factorial design must be at least 2^{n-k+1} - $2^{n-k} = 2^{n-k}$. In Part 1 of the proof we determine the constraints on the new block of 2^{n-k} experiments. In Part 2 we prove the second part of the above claim.

Consider a process which depends on n variables $T_1, T_2, \dots, T_n$. A block of 2^{n-k} factorial experiments is designed on this process. Let the design be such that variables T_1, T_2, ..., T_{n-k} form a set of basic variables. The variables T_{n-k+1}, ..., T_n are the corresponding non-basic variables. Let the columns of the basic variables T_1, T_2, ..., T_{n-k} be denoted by v_1, v_2, ..., v_{n-k} respectively. The column of a non-basic variable T_y corresponds to a unique product of the columns of the basic variables and is denoted by $v_{p(y)}$ as shown in Table C.1.

Part 1:

Suppose 2^{n-k} new experiments are to be designed such that these experiments and the 2^{n-k} experiments from the original design, form a 2^{n-k+1} factorial design. Let the columns of the variables T_1, T_2, ..., T_n in the new design be denoted by u_1, ..., u_n respectively as shown in in Table C.1. The requirement that the overall design be a factorial design puts severe constraints on the choice of the new design, that is on columns $u_1, \dots, u_n$, as shown below.

Constraint 1: Every column, u_i, must have equal numbers of +1 and -1 entries. This result follows from the the fact that all columns of the original design have equal numbers of +1 and -1 elements and so must the columns of the overall design.

Constraint 2: The columns of the overall design will be orthogonal only if the columns of the new design are orthogonal.

Proof: Consider any two distinct columns v'_x and v'_y of the overall design. They are orthogonal only if,

$$\begin{bmatrix} {v_x}^T & u_x^T \end{bmatrix} \cdot \begin{bmatrix} v_y \\ u_y \end{bmatrix} = 0 \qquad \text{(C.1)}$$

But, v_x and v_y are orthogonal to each other. That is, ${v_x}^T \cdot v_y = 0$. Therefore the necessary condition for Equation C.1 to hold is that,

$$u_x^T \cdot u_y = 0 \qquad \forall x \neq y \qquad \text{(C.2)}$$

Thus, the overall design will be orthogonal only if the new design is orthogonal.

Constraint 3: If a set of variables is independent in the original design, it is independent in the new design too.

Proof: Let us assume that the constraint is not true. Let variable T_x $(x \leq n-k)$ be dependent on variables T_i, T_j, ..., T_k $(i, j, \ldots, k \leq n-k)$ in the new design. That is, $u_x = u_i \cdot u_j \ldots u_k$ or $u_x = -u_i \cdot u_j \ldots u_k$. The arguments for both these cases are identical, so only the former case has been considered in the proof below.

The product of the columns of T_i, T_j, ..., T_k, and T_x in the overall design is given by,

$$I'_{ij\ldots kx} = \begin{bmatrix} v_i \\ u_i \end{bmatrix} \cdot \begin{bmatrix} v_j \\ u_j \end{bmatrix} \cdots \begin{bmatrix} v_k \\ u_k \end{bmatrix} \cdot \begin{bmatrix} v_x \\ u_x \end{bmatrix} = \begin{bmatrix} v_{ij\ldots kx} \\ u_{ij\ldots kx} \end{bmatrix} = \begin{bmatrix} v_{ij\ldots kx} \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = v'_{ij\ldots kx} \qquad \text{(C.3)}$$

In Equation C.3, $v_{ij\ldots kx}$ has equal numbers of +1 and -1 entries. Consequently, the column $v'_{ij\ldots kx}$ will have more +1 than -1 entries. Hence, the column $v'_{ij\ldots kx}$ cannot be part of an orthogonal matrix. Therefore the assumption that columns u_x is dependent on columns u_i, u_j, ..., u_k is wrong.

We can generalize the above result and claim that, the overall design will be orthogonal only if, every set of independent variables of the original design, is independent in the new design too. This implies that a set of basic variables of the original design must be independent in the new design.

Constraint 4: Every non-basic variable in the new design must correspond to the interaction of the same basic variables as it did in the original design. The sign of the interaction can differ.

Proof: Consider a non-basic variable T_y ($y > n - k$). Its column in the original design is denoted as $v_{p(y)}$. Let this variable be confounded with interaction $I_{ij...k}$ of basic variables T_i, $T_j, \ldots, T_k$ ($i, j, \ldots, k \leq n-k$) in the original design. That is, $v_{p(y)} = v_{ij...k}$ or $v_{p(y)} = -v_{ij...k}$. The arguments for both these cases are identical, so only the former case has been considered in the proof below. The product of the columns of T_i, T_j, ..., T_k, and T_y in the overall design is given by,

$$I'_{ij...ky} = \begin{bmatrix} v_i \\ u_i \end{bmatrix} \cdot \begin{bmatrix} v_j \\ u_j \end{bmatrix} \cdots \begin{bmatrix} v_k \\ u_k \end{bmatrix} \cdot \begin{bmatrix} v_{p(y)} \\ u_y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ u_{ij...ky} \end{bmatrix} \tag{C.4}$$

If the overall matrix is orthogonal then there are only two possible cases.

1. $I'_{ij...ky} = v'_{avg}$. That is all the elements of $u_{ij...ky}$ are +1.

2. The sum of the elements of $I'_{ij...ky}$ is zero. That is all the elements of $u_{ij...ky}$ are -1.

For any other choice of $u_{ij...ky}$, the columns $I'_{ij...ky}$ will not correspond to a column of a orthogonal matrix. Therefore, the only way in which the overall matrix will be orthogonal is if $u_{ij...ky} = +v_{avg}$ or if $u_{ij...ky} = -v_{avg}$. This implies that $u_y = v_{ij...k}$ or $u_y = -\ v_{ij...k}$ in the new design. That is, the non-basic variable in the new design must correspond to the interaction of the same basic variables as it did in the original design although the sign of the interaction may differ.

Combining the above constraints, we conclude that the new design and the original design must have the same basic variables, and the non-basic variables must correspond to the same interactions of the basic variables. That is, the confounding pattern of the new design must be the same as that of the original design. This proves the first part of the claim.

Variable	T_1	T_2	T_3	T_4	T_5
Expt. Block 1	v_1	v_2	v_3	v_{12}	v_{23}
Expt. Block 2	v_1	v_2	v_3	$\pm v_{12}$	$\pm v_{23}$

Table C.2: Table for Example C-1

Part 2:

In Part 1 we have established the following results:

- A set of basic variables of the original design must be a set of basic variables of the new design.
- Every non-basic variable of the new design must correspond to the same interaction of basic variables as in the original design, although the *sign* may differ.

Therefore, the only choice in selecting the new design is the signs of the interactions confounding with the non-basic variables. That is, if a non-basic variable T_y corresponds to interaction $v_{p(y)}$ in the first block of experiments, then it can correspond to either $+v_{p(y)}$ or to $-v_{p(y)}$ in the second.

There are k non-basic variables in the original design. Hence there are 2^k possible choices for selecting the signs of the columns. The choice corresponding to selecting all positive signs results in the same design as the original one. Hence the overall design will have two sets of identical rows and will not correspond to a 2^{n-k+1} factorial design. Each of the remaining 2^k-1 combinations corresponds to a distinct block design which can be used with the original design to form a larger factorial design. Thus, the second part of the claim is proved.

Using this result, we develop an algorithm which generates different block designs and selects the one which produces the minimum confounding between the variables and the probable interactions.

Example C-1

- Suppose a process depends on 5 variables $T_1, \ldots, T_5$ and 8 experiments have been performed as shown in Table C.2. T_1, T_2 and T_3 form a set of basic variables.

In the second block of 8 experiments T_4 and T_5 can be assigned to $\pm v_{12}$ and $\pm v_{23}$ respectively. Hence, there are essentially 4 choices. Choosing columns of T_4 and T_5 as $+v_{12}$ and $+v_{23}$ respectively in the second block, gives the same block of experiments as in the first block. Therefore, this choice is not valid.

If we choose $-v_{12}$ for T_4, then, T_4 along with T_1, T_2 and T_3 forms a set of basic variables of the overall design. Choosing the column of T_5 as $+v_{23}$ will make T_5 confound with the column I'_{23} in the overall design. If the $-v_{23}$ is selected then, T_5 will be confounded with I'_{134} in the overall design. In general, it is more desirable to confound T_5 with I'_{134} than with I'_{23}. ■

C.2 Fold-Over Designs

The procedure of folding factorial designs is commonly used to increase the resolution of the designs and enable them to determine important interactions. In this section we show that this procedure is a special case of the Full-Block design strategy. This result not only justifies the use of Full-Block designs, but also establishes its superiority over the fold-over designs in sorting out the important interactions.

Claim:

The fold-over design is a particular case of a Full-Block design. A fold-over design can be obtained by the following procedure:

1. *Determine the order of each non-basic variable, that is, order of the interaction of the basic variables with which it is confounded. (For example, if T_5 is confounded with the interaction of basic variables T_1, T_2 and T_3 then its order is 3.)*

2. *Design a Full-Block of experiments by changing the signs of the* **even-ordered** *non-basic variables only.*

Proof:

Consider a process dependent on n variables, T_1, T_2, ..., T_n, on which a 2^{n-k} factorial experiment has been designed. Let T_1, T_2, ..., T_{n-k} be a set of basic variables and T_{n-k+1}, T_2, ..., T_n be the corresponding set of non-basic variables. A non-basic variable corresponds

V/I	T_1	T_2	T_3	T_4	T_5
Expt.	v_1	v_2	v_3	v_{12}	v_{123}
1	+	+	+	+	+
2	+	+	-	+	-
3	+	-	+	-	-
4	+	-	-	-	+
5	-	+	+	-	-
6	-	+	-	-	+
7	-	-	+	+	+
8	-	-	-	+	-

Table C.3: Initial Design of Example C-2

to a distinct interaction of the basic variables. Hence, given the settings of the basic variables, the settings of the non-basic variables can be uniquely determined.

From the property of orthogonal matrices discussed in Appendix A, we know that the columns of the basic variables contain rows with all the possible combinations of +1 and -1 elements. Therefore, for every row of the design, there exists a second row in which the signs of all the elements of the basic columns are reversed. Clearly, in the second row the signs of the elements of the columns of the odd-ordered non-basic variables will be reversed too.

Comparing these two rows, we find that the signs of all the elements are reversed except those which lie on the even-ordered non-basic variables. Suppose a new design is obtained by reversing only the signs of the columns of the even-ordered non-basic variables. For every row of the original design, there will be a row in the new design which has the sign of all its elements reversed. Thus the new design is a fold-over design. Moreover, only the signs of the columns of the non-basic variables have been changed. Therefore the design is also a Full-Block design.

Thus we have determined a technique of designing the fold-over design using the Full-Block design strategy. This proves the claim.

Example C-2

- Consider the initial design shown in Table C.3. The fold-over design of the initial design is shown in Table C.4. Table C.4 also shows the Full-Block design obtained using the above procedure. Comparing the Full-Block design and the fold-over design

	Fold-over Design				
V/I	T_1	T_2	T_3	T_4	T_5
Expt.	$-v_1$	$-v_2$	$-v_3$	$-v_{12}$	$-v_{123}$
1	-	-	-	-	-
2	-	-	+	-	+
3	-	+	-	+	+
4	-	+	+	+	-
5	+	-	-	+	+
6	+	-	+	+	-
7	+	+	-	-	-
8	+	+	+	-	+

	Full-Block Design				
V/I	T_1	T_2	T_3	T_4	T_5
Expt.	v_1	v_2	v_3	$-v_{12}$	v_{123}
1	+	+	+	-	+
2	+	+	-	-	-
3	+	-	+	+	-
4	+	-	-	+	+
5	-	+	+	+	-
6	-	+	-	+	+
7	-	-	+	-	+
8	-	-	-	-	-

Table C.4: Fold-over and Full-Block Design of Example C-2

we find that they are identical, except that the rows are arranged differently. From this example, it can be seen that if the signs of the columns of the even-ordered non-basic variables are changed, the fold-over design is obtained. ■

Appendix D

Completing Fractional Factorial Blocks

In this section, we will study the techniques of completing a partially incomplete block of a factorial design.

Claim:

Given an incomplete block of m ($m > 2^{n-k}$) experiments of a complete block of a 2^{n-k+1} fractional factorial design,

1. *The incomplete block must have the same confounding pattern as the complete design block.*

2. *There exists a unique set of $2^{n-k+1}-m$ experiments which complete the block.*

Proof:

In order to prove the above claim, it is assumed that the given set of m experiments indeed corresponds to an incomplete factorial design. Later, using the properties of the above claim, a procedure is developed to check if the incomplete block is part of a factorial design or not. We shall prove the Claim 1 in two parts. In Part 1(a) we show that if any group of variables are confounded in the complete block then they must confound in the incomplete block too. In Part 1(b) we prove the converse result. The first part of the claim is proved by putting together the results of Part 1(a) and 1(b). Part 2 of the proof proves the second part of the claim.

Part 1(a):

Consider a block of 2^{n-k+1} factorial design. From this block any m $(m > 2^{n-k})$ rows are selected to form an incomplete block. The columns of the complete block are denoted by u_i. The columns of the incomplete block are indicated by prime and those of the missing block by double primes. Therefore,

$$u_i = \begin{bmatrix} u_i' \\ u_i'' \end{bmatrix} \qquad \forall i \tag{D.1}$$

For any column u_i of the complete block which is dependent on some other columns u_j, u_k, ..., u_p,

$$u_i = u_j \cdot u_k \cdot \ldots \cdot u_p \tag{D.2}$$

$$\begin{bmatrix} u_i' \\ u_i'' \end{bmatrix} = \begin{bmatrix} u_j' \\ u_j'' \end{bmatrix} \cdot \ldots \cdot \begin{bmatrix} u_p' \\ u_p'' \end{bmatrix} \tag{D.3}$$

Clearly, this relationship holds for the incomplete block too. Therefore, any group of variables which are confounded in the complete block, must be confounded in the incomplete block too.

Part 1(b):

Next we need to show that if that any column u_i' is dependent on columns u_j', u_k', ..., u_p' in the incomplete block, it must be so in the complete block too. In order to prove this, let us assume that the statement is not true. Suppose,

$$u_i' = u_j' \cdot u_k' \cdot \ldots \cdot u_p' \tag{D.4}$$

By assumption,

$$u_i'' \neq u_j'' \cdot u_k'' \cdot \ldots \cdot u_p'' \tag{D.5}$$

In any orthogonal matrix if a column is not equal to another, it must be orthogonal to it. Therefore, if the column u_i of the complete block is not dependent on the column corresponding to the product of columns u_j, u_k,..., u_p it must be orthogonal to it.

If the product of the columns u_j, u_k,..., u_p is denoted by $u_{jk...p}$, then

$$u_i^T \cdot u_{jk...p} = \left[\begin{array}{cc} u_i'^T & u_i''^T \end{array} \right] \cdot \left[\begin{array}{c} u'_{jk...p} \\ u''_{jk...p} \end{array} \right] = u_i'^T \cdot u'_{jk...p} + u_i''^T \cdot u''_{jk...p} = 0 \qquad \text{(D.6)}$$

But from Equation D.4 u_i' and $u'_{jk...p}$ are identical. Therefore,

$$u_i^T \cdot u_{jk...p} = m + u_i''^T \cdot u''_{jk...p} = 0 \qquad \text{(D.7)}$$

$$\Longrightarrow u_i''^T \cdot u''_{jk...p} = -m \qquad \text{(D.8)}$$

$$\Longrightarrow \left| u_i''^T \cdot u''_{jk...p} \right| = m > 2^{n-k} \qquad \text{(D.9)}$$

The dimension of u_i'' and $u''_{jk...p}$ is $2^{n-k+1}-m$ and their elements are +1 or -1. Hence,

$$\left| u_i''^T \cdot u''_{jk...p} \right| \leq 2^{n-k+1} - m < 2^{n-k} \qquad \text{(D.10)}$$

Comparing Equations D.9 and Equations D.10, there is a contradiction. Therefore the assumption that column u_i of the complete block is not dependent on the column corresponding to the product of columns u_j, u_k,..., u_p is incorrect. Hence the column u_i of the complete block is dependent on the columns u_j, u_k,..., u_p.

Therefore, if any column depends on a set of columns in the incomplete block, it must do so in the complete block too. Consequently, if a variable is confounded with a group of variables in the incomplete block, then it must be confounded with the same group of variables in the complete block too.

From the results of Part 1(a) and Part 1(b) we conclude that the confounding patterns of the complete and incomplete blocks must be identical.

Part 2:

In Part 1 it was shown that the confounding patterns of the complete and the incomplete blocks are the same. Therefore, any set of basic variables of the complete block must be a set of basic variables of the incomplete block and visa versa.

From the properties of Binary Orthogonal Matrices given in Appendix A, it is known that the rows of the basic variables must be such that each row corresponds to a unique

Expt.	T_1	T_2	T_3	T_4	T_5
1 (1)	+	+	+	+	+
2 (2)	+	+	-	-	-
3 (3)	+	-	+	+	-
4 (5)	-	+	+	-	+
5 (7)	-	-	+	-	-

Table D.1: Incomplete Block of Experiments

combination of +1 and -1. Also, the columns of the complete block contain all possible combinations of +1 and -1.

Using this property of Binary Orthogonal Matrices along with the result of Part 1, the incomplete block can be completed in the following steps:

Step 1. Determine the basic variables and the confounding pattern of the incomplete block.

Step 2. Check which of the 2^{n-k+1} combinations of +1 and -1 are missing from the columns of the basic variables. There will be $2^{n-k+1}-m$ such combinations.

Step 3. For each of the combinations determined in Step 2, determine the entries of the non-basic variables of the matrix using the confounding pattern of the incomplete block.

Since each of the steps of the above procedure is deterministic and unique, we prove Claim 2, i.e. there exists a unique set of $2^{n-k+1}-m$ experiments which completes the block.

Example D-1

- Consider a process **P** which is dependent on five variables T_1, T_2, T_3, T_4 and T_5. Say a 2^{5-2} fractional factorial experiment is to be carried out on this process, as shown in the Initial Design of Table B.1. Suppose that of these 8 experiments, 5 ($m = 5 > 4$) have already been carried out as shown in Table D.1. Which 3 experiments need to be carried out to complete such an incomplete block?

  Step 1: As per the procedure described above, the first step is to determine the 3 ($n - k + 1$) basic variables. From Table D.1 it is observed that the following combination of variables correspond to a basic set of variables:

	Step 2			Step 3	
Expt.	T_1	T_4	T_5	T_2=I_{145}	T_3=I_{14}
1 (6)	-	+	-	+	-
2 (4)	+	-	+	-	-
3 (8)	-	+	+	-	-

Table D.2: Missing Block of Experiments

(i) T_1, T_2, T_3 (ii) T_1, T_2, T_4 (iii) T_1, T_2, T_5 (iv) T_1, T_3, T_5

(v) T_1, T_4, T_5 (vi) T_2, T_3, T_4 (vii) T_2, T_4, T_5 (viii) T_3, T_4, T_5

Since only one set of basic variables is required, it is not necessary to enumerate all the sets of basic variables as done here. Let us select T_1, T_4 and T_5 as the set of basic variables.

Step 2: Scanning the columns of T_1, T_4 and T_5 the missing combinations of +1 and -1 are determined. They are shown in Table D.2.

Step 3: From Table D.1, it is observed that T_2 confounds with interaction I_{145} and T_3 confounds with I_{14}. Using this information the missing columns of T_2 and T_3 are determined in Table D.2.

Comparing Table D.2 with the Initial Design of Table B.1, we conclude that Expts. 1, 2 and 3 of Table D.2 correspond to the missing experiments 6, 4, and 8 of the Initial Design of Table B.1 respectively. Hence, the missing experiments of the incomplete block have been determined. ■

Determining Incomplete Factorial Blocks

All the results determined above assume that the the incomplete block of experiments indeed corresponds to a complete 2^{n-k+1} fractional factorial block. Therefore, if the set of experiments does not correspond to a complete block, the results do not hold.

A block of experiments which do not correspond to a complete block will have either more than $n-k+1$ or less than $n-k+1$ independent columns. Thus there will not be any set of $n-k+1$ basic variables. When the procedure for completing the block is applied to such a block, the method will fail in Step 1. Thus the procedure of determining the basic variables can be used to check if a block of experiments correspond to an incomplete block or not.

Appendix E

Computer Software Listings

E.5 Header and Data Files

E.1.1 Startup.C

```
/************************************
*
*Program: Startup.c
*
*This program is the first program to be
*executed. It performs the following functions:
*1) Designs an optimal fold-over design.
*2) Reads the results of the experiments and
*   analyzes them.
*3) The results are used to form a list of
*   probable interactions.
*4) The databases required for the subsequent
*   programs are formed.
*
*Inputs: (a) No. of Variables (b) External
*   Interaction File (See sample file)
*Output: Results of the analysis of the first
*   block of experiments.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"
extern int fileline;

/* main variables */
int num_vars,  num_blocks, **expt_mat=NULL, **sel_row=NULL;
int *int_mat=NULL, *var_mat=NULL, *size_expt=NULL, *size_blk=NULL;
int **blk_mat=NULL, tot_ints, *type_blk=NULL, *blk_info=NULL;
char **var_label=NULL;
float **res_mat=NULL;

/* basic variables */
int *var_col=NULL, mat_rows, mat_cols, *mat=NULL, *conint=NULL;
int *sel_col=NULL, *overlap=NULL;
float *matres=NULL, *coeff=NULL, *covar=NULL, *prob_col=NULL;
char  file_name[100];

/* interaction variables */
int *sel_col_int=NULL, *list_int=NULL, *no_int_col=NULL, *sel_var=NULL;
/* eternal interaction variables */
int *cints=NULL, exint_count, *exint_wt=NULL;
void f_test(float, int, float, int, float *, float *);
void actual_plant_result(int, int, int *, float *);
void select_experiments(int, int *, int *, int *, int **);

main(){

void initialize_data(void);
void setup_matrix(void);
void select_imp_columns(void);
void free_up(void);
void make_list_int(void);
void make_data_base(void);

printf(file_divider);
initialize_data();
setup_matrix();
get_coefficients(mat_rows, mat_cols, mat, matres, coeff, covar, overlap );
select_imp_columns();
fflush(stdin); getch();
make_list_int();
make_data_base();

disp_var_info(num_vars, num_blocks, var_mat, type_blk, var_label);
disp_int_info(tot_ints, num_blocks, type_blk, int_mat, var_label);
disp_col_info(0, size_blk, blk_mat);
disp_blk_info(num_blocks, size_expt, size_blk, blk_mat, type_blk, blk_info);

printf("Save Database?  (y/n)");
fflush(stdin);
if(getch()=='y'){
   save_database(data_file, num_vars, tot_ints, num_blocks,
       size_expt, size_blk, type_blk, var_mat, int_mat,
       blk_mat, sel_row, expt_mat, res_mat, var_label, blk_info);
   }
free_up();
return;
}

/************************************
*
* Sets up the matrices mat and conint
*
*************************************/

static void setup_matrix(void){
```

```
int i, row, blk;

set_mat(num_vars, 0, expt_mat, sel_row, size_blk, &mat_rows,
                      &mat_cols, &var_col, &mat, &conint);
pmalloc( (void **) &matres,  mat_rows*sizeof(float));
pmalloc( (void **) &coeff,      mat_cols*sizeof(float));
pmalloc( (void **) &covar,      mat_cols*mat_cols*sizeof(float));
pmalloc( (void **) &overlap, mat_cols*sizeof(int));

/* assign results to matres */
for(i=0; i<mat_rows; i++){
    blk=sel_ptr(0, i, SBL);
    row=sel_ptr(0, i, SEP);
    matres[i] = res_ptr(blk, row);
    }
return;
}

/*************************************
*
* Initializes variables.
*
**************************************/

static void initialize_data(void){

int i,j,k, matsize;
char tempch[] = "T      ";
int set_exptmat(void);

printf("Enter number of variables:  ");
scanf("%d", &num_vars);

/* variable label */
pmalloc( (void **) &var_label, (num_vars+1)*sizeof(char *));
for(i=0; i<num_vars+1; i++){
    var_label[i] = NULL;
    pmalloc( (void **) &(var_label[i]), Label_Length*sizeof(char));
    }
for(i=1; i<num_vars+1; i++){
    itoa(i, tempch+1, 10);
    strcpy(var_label[i-1], tempch);
    }
strcpy(var_label[num_vars], "Avg.");
num_blocks = 1;
pmalloc( (void **) &size_expt, MAX_BLK*sizeof(int));
pmalloc( (void **) &size_blk,  MAX_BLK*sizeof(int));
pmalloc( (void **) &type_blk,  MAX_BLK*sizeof(int));
pmalloc( (void **) &blk_info,  MAX_BLK*sizeof(int));

pmalloc( (void **) &res_mat,   MAX_BLK*sizeof(float *));
pmalloc( (void **) &expt_mat,  MAX_BLK*sizeof(int *));
pmalloc( (void **) &sel_row,   MAX_BLK*sizeof(int *));

for(i=0; i<MAX_BLK; i++){
    size_expt[i]=0;
    size_blk[i]=0;
    type_blk[i]=0;
    blk_info[i]=0;
    res_mat[i]=NULL;
    res_mat[i]=NULL;
    expt_mat[i]=NULL;
    sel_row[i]=NULL;
    }

matsize = set_exptmat();
mat_rows      = matsize;
size_blk[0]  = matsize;
type_blk[0]=0;

pmalloc((void **)  &(sel_row[0]),     sof_selrow(0)*sizeof(int));
pmalloc( (void **) &(res_mat[0]),     sof_resmat(0)*sizeof(float));

/* set res_mat and get plant results */
printf("EXPT. NO.\t\tRESULT\n");
for(i=0; i<mat_rows; i++){
    actual_plant_result(1, num_vars, expt_ptr(0, i), &(res_ptr(0,i)) );
    printf("%3d\t\t\t%6.2f\n", i+1, res_ptr(0, i));
    }

/* set sel_row */
for(j=0; j<size_expt[0]; j++){
    sel_ptr(0, j, SBL) = 0;
    sel_ptr(0, j, SEP) = j;
    }
return;
}
```

```
/**************************************
*
* Reads the external interaction
* file and determines experiments in which
* none of the interactions confound with the
* variables.
*
**************************************/

static int set_exptmat(void){

int i, j, k, *tempmat=NULL, matsize;
int *endvar=NULL, th_ord;
void get_interactions(int *, int **);

get_interactions(&th_ord, &endvar);

if(th_ord==0){
   matsize=1;
   while(matsize<num_vars) matsize *= 2;
   pmalloc((void **) &tempmat, matsize*matsize*sizeof(int));
   create_matrix(matsize, tempmat);
   }
else select_experiments(th_ord, exint_wt, &matsize, endvar, &tempmat);

/* Creating a fold-over design..... */
size_expt[0] = 2*matsize;
pmalloc( (void **) &(expt_mat[0]),     sof_exptmat(0)*sizeof(int));

/* set expt_mat */
for(i=0; i<matsize; i++){
   for(j=0; j<num_vars; j++){
      *(expt_ptr(0, i) + j) = tempmat[i*matsize + j];
      *(expt_ptr(0, (i+matsize)) + j) = -tempmat[i*matsize + j];
      }
   }
pfree((void **) &tempmat);
return(2*matsize);
}

/**************************************
*
* Reads the interactions
* from the user defined input file.
* (Note: interactions upto 4th order
* can be entered)
*
**************************************/

static void get_interactions(int *pth_ord, int **pendvar){

int i, j, k, count, *tempvect=NULL;
char temp_str[100], f_name[]="c:\\rizwan\\data\\interaction.dat";
FILE *handle;

printf("Load External Interactions?  (y/n)\n");
fflush(stdin);
if(getch()!='y'){ *pth_ord=0; return;}

handle = fopen(f_name, "r");
if(handle==NULL) error( "Cant open output file", f_name, -1);
get_line( temp_str, handle);
if(temp_str[1]!='i') error("Invalid Interaction file:  ", f_name, fileline);

get_line( temp_str, handle); get_label(temp_str);
exint_count=atoi(temp_str);

pmalloc((void **) &tempvect, cint_len*sizeof(int));
pmalloc((void **) &cints,      cint_len*exint_count*sizeof(int));
pmalloc((void **) &exint_wt, exint_count*sizeof(int));
pmalloc((void **) pendvar,      exint_count*sizeof(int));

for(i=0; i<cint_len*exint_count; i++) cints[i]=0;
for(i=0; i<exint_count; i++) exint_wt[i]=0;

count=0;
while ((!get_line( temp_str, handle))&&(count<exint_count)){
   if (temp_str[0] == Field_Char){
      unget_line();
      break;
      }
   if(!get_label(temp_str)) error("error:", f_name, fileline);
   cints_ptr(count, COR) = atoi(temp_str);
   if((cints_ptr(count, COR)>4)||(cints_ptr(count, COR)<2))
      error("Invalid Interaction Order.  ", f_name, fileline);

   for(j=0; j<cints_ptr(count, COR); j++){
      if(!get_label(temp_str)) error("error:", f_name, fileline);
      cints_ptr(count, CV1+j) = atoi(temp_str)-1;
      }
```

```
    for(i=0; i<cints_ptr(count, COR); i++)
        for(j=i; j<cints_ptr(count, COR); j++)
            if(cints_ptr(count, CV1+i)>cints_ptr(count, CV1+j)){
            k = cints_ptr(count, CV1+i);
            cints_ptr(count, CV1+i) = cints_ptr(count, CV1+j);
            cints_ptr(count, CV1+j) = k;
            }

    if(!get_label(temp_str)) error("error:", f_name, fileline);
    exint_wt[count] = atoi(temp_str);
    count++;
    }

if(count != exint_count) error("error in interaction data", f_name, -1);

for(i=0, count=0; i<exint_count; i++)
    if(cints_ptr(i, COR)==3){
        memcpy(tempvect, &(cints_ptr(i, 0)), cint_len*sizeof(int));
        memcpy(&(cints_ptr(i,0)), &(cints_ptr(count, 0)), cint_len*sizeof(int));
        memcpy(&(cints_ptr(count, 0)),  tempvect, cint_len*sizeof(int));
        k = exint_wt[i];
        exint_wt[i] = exint_wt[count];
        exint_wt[count] = k;
        count++;
        }
*pth_ord = count;

for(count=0; count<exint_count; count++)
    (*pendvar)[count] = cints_ptr(count, CV1+cints_ptr(count, COR)-1);

fclose(handle);
return;
}

/*************************************
*
* Selects important effects using the
* Lenth's Algorithm(Technometrics, Nov '89)
*
*************************************/

static void select_imp_columns(void){

int i, j, temp=0, count;
float T_975=2.0,  *ord_eff=NULL, *abs_eff=NULL, temp1, temp2;
float So, ME, PSE, SSE, noise_var;

pmalloc((void **) &ord_eff, mat_cols*sizeof(float));
pmalloc((void **) &abs_eff, mat_cols*sizeof(float));
pmalloc((void **) &prob_col, mat_cols*sizeof(float));
pmalloc((void **) &sel_col, mat_cols*sizeof(int));
pmalloc((void **) &sel_var, mat_cols*sizeof(int));

for(i=0; i<mat_cols; i++) {sel_col[i]=1;  prob_col[i]=0.0;}

for(i=0; i<mat_cols; i++){
    ord_eff[i]=0.0;
    ord_eff[i]=abs_eff[i]= fabs(coeff[i]);
    }
/* ordering the values of abs_eff in accending order */
for(i=0; i<mat_cols; i++)
    for(j=i; j<mat_cols; j++)
        if(ord_eff[i]>ord_eff[j]){
            temp1=ord_eff[i]; ord_eff[i]=ord_eff[j]; ord_eff[j]=temp1;
            }
So = 1.5 * ord_eff[mat_cols/2-2];
for(i=0; i<mat_cols; i++) if(ord_eff[i] >= 1.5*So) { temp = i-1; break;}
PSE = 1.5 * ord_eff[temp/2];
ME = T_975 * PSE;

for(i=0; i<mat_cols; i++) if(abs_eff[i] <= ME) sel_col[i]=0;

for(i=0, count=0, SSE=0.0; i<mat_cols; i++)
    if(sel_col[i]==0){ count++; SSE += coeff[i]*coeff[i]; }

for(i=0; i<num_vars; i++)
    if(sel_col[var_col[i]]==1) sel_var[i]=1;
    else sel_var[i]=0;

if(count>1){
    noise_var = SSE/(count-1);
    for(i=0; i<mat_cols; i++){
        if(fabs(coeff[i]*coeff[i]) > (noise_var)){
            f_test(coeff[i]*coeff[i], 1, noise_var, count-1, &temp1, &temp2);
            prob_col[i]=1-temp2;
            }
        else prob_col[i]=0.0;
        }
    }
```

```
else error("Too few noise columns ...\n", NULL, -1);

printf("\nCut-off value:  %4.2f      Noise Var:  %4.2f\n\n", ME, noise_var);
pfree((void **) &abs_eff);
pfree((void **) &ord_eff);
return;
}

/*************************************
*
* frees arrays
*
*************************************/

static void free_up(void){
int i;

for(i=0; i<num_vars+1; i++)
    pfree((void **) &(var_label[i]));

pfree((void **) &mat);
pfree((void **) &var_col);
pfree((void **) &conint);
pfree((void **) &var_label);
pfree((void **) &expt_mat);
pfree((void **) &res_mat);
pfree((void **) &size_expt);
pfree((void **) &sel_row);
pfree((void **) &matres);
pfree((void **) &coeff);
pfree((void **) &covar);
pfree((void **) &overlap);
pfree((void **) &sel_col);
pfree((void **) &prob_col);
pfree((void **) &list_int);
pfree((void **) &no_int_col);
pfree((void **) &sel_col_int);
pfree((void **) &int_mat);
pfree((void **) &var_mat);
pfree((void **) &blk_mat);

return;
}

/*************************************
*
* Forms the list of probable interactions.
*
*************************************/

static void make_list_int(void){

int i, j, k, v1, v2, count, min_weight, weight_int;
int *tempvect=NULL, col;
float mcoeff1, mcoeff2, mwts1, wt_factor;

pmalloc((void **) &tempvect, int_len*sizeof(int));
pmalloc((void **) &no_int_col, mat_cols*sizeof(int));
pmalloc((void **) &sel_col_int, mat_cols*sizeof(int));
pmalloc((void **) &list_int, list_len*mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) no_int_col[i]=0;
for(i=0; i<mat_cols; i++) sel_col_int[i]=0;
for(i=0; i<list_len*mat_cols; i++) list_int[i]=0;

/* normalize the external interaction weights */
for(i=0, mcoeff1=0, mcoeff2=0; i<num_vars; i++){
    if(fabs(coeff[var_col[i]]) > mcoeff1){
        mcoeff2 = mcoeff1;
        mcoeff1 = fabs(coeff[var_col[i]]);
        }
    else if(fabs(coeff[var_col[i]])>mcoeff2) mcoeff2=fabs(coeff[var_col[i]]);
    }

for(i=0, mwts1=0; i<exint_count; i++)
    if(abs(exint_wt[i])>mwts1) mwts1 = abs(exint_wt[i]);
wt_factor = mcoeff1*mcoeff2/mwts1;

/* assign external interactions to list[] */
for(i=0; i<exint_count; i++){
    for(j=0, col=0; j<cints_ptr(i, COR); j++)
        col = conint[col*mat_cols + var_col[cints_ptr(i, CV1+j)]];

    no_int_col[col]++;
    list_ptr(col, (no_int_col[col]-1), LOR)=cints_ptr(i, COR);
    list_ptr(col, (no_int_col[col]-1), LWE)=(int)(exint_wt[i]*wt_factor*100);

    for(j=0; j<cints_ptr(i, COR); j++)
        list_ptr(col, (no_int_col[col]-1), LV1+j) = cints_ptr(i, CV1+j);
    }
```

```
/* determine 2nd order interactions */
for(v1=0; v1<num_vars; v1++){
   for(v2=v1+1; v2<num_vars; v2++) if((sel_var[v1]==1)||(sel_var[v2]==1)){
      k= conint[ var_col[v1]*mat_cols + var_col[v2] ];
      weight_int =
         (int) 100*fabs(coeff[k]*coeff[var_col[v1]]*coeff[var_col[v2]]);

      if(no_int_col[k]<max_int){
         no_int_col[k]++;
         list_ptr(k, (no_int_col[k]-1), LOR) = 2;
         list_ptr(k, (no_int_col[k]-1), LV1) = v1;
         list_ptr(k, (no_int_col[k]-1), LV2) = v2;
         list_ptr(k, (no_int_col[k]-1), LWE) = weight_int;
         }
      else{
         min_weight=weight_int;
         count=-1;
         for(i=0; i<no_int_col[k]; i++)
            if(min_weight> list_ptr(k, i, LWE)){
               min_weight=list_ptr(k, i, LWE);
               count=i;
               }
         if(count>=0){
            list_ptr(k, (count), LOR) = 2;
            list_ptr(k, (count), LV1) = v1;
            list_ptr(k, (count), LV2) = v2;
            list_ptr(k, (count), LWE) = weight_int;
            }
         }
      }
   }

for(i=0; i<mat_cols; i++)
   for(j=0; j<no_int_col[i]; j++)
      for(k=j+1; k<no_int_col[i]; k++)
         if(list_ptr(i, j, LWE) < list_ptr(i, k, LWE)){
            memcpy(tempvect, &(list_ptr(i, j, 0)), int_len*sizeof(int));
            memcpy(&(list_ptr(i, j, 0)), &(list_ptr(i, k, 0)),
                                 int_len*sizeof(int));
            memcpy(&(list_ptr(i, k, 0)), tempvect, int_len*sizeof(int));
            }

printf("Select all interactions?    (y/n)\n");
fflush(stdin);
if(getch()=='y')
   for(i=0; i<mat_cols; i++) sel_col_int[i]=1;
else
   for(i=0; i<mat_cols; i++) if(sel_col[i]==1) sel_col_int[i]=1;

for(i=0; i<num_vars; i++) if(sel_col[var_col[i]]==1)
      sel_col_int[var_col[i]]=0;
sel_col_int[0]=0; /* no interaction on avg. col */
return;
}

/*************************************
 *
 * Forms the database for the software.
 * The variables tot_ints, blk_mat,
 * var_mat, int_mat are initialized here.
 * The other variables were set in
 * initialize()
 *
 *************************************/

static void make_data_base(void){

int i, j, k, l, v1, sum, sign, count, *tempvect=NULL, col;

/* makes VARIABLE DATABASE */
pmalloc((void **) &var_mat, sof_varmat*sizeof(int));
for(i=0; i<sof_varmat; i++) var_mat[i]=0;
var_ptr(num_vars, VNO) = num_vars;
var_ptr(num_vars, VCO) = (int)(C_FACTOR*coeff[0]);

for(i=0; i<num_vars; i++){
   var_ptr(i, VNO) = i;
   var_ptr(i, VCO) = (int)(C_FACTOR*coeff[var_col[i]]);
   var_ptr(i, VB0) = var_col[i];
   }

/* makes INTERACTION DATABASE */
pmalloc((void **) &tempvect,     mat_cols*sizeof(int));

for(i=0, tot_ints=0; i<mat_cols; i++)
   if(sel_col_int[i]==1) tot_ints += no_int_col[i];

pmalloc((void **) &int_mat, sof_intmat*sizeof(int));
for(i=0; i<sof_intmat; i++) int_mat[i]=0;
for(i=0, count=0; i<mat_cols; i++){
    if(sel_col_int[i]==1){
        for(j=0; j<no_int_col[i]; j++){
            for(k=0; k<mat_cols; k++) tempvect[k]=1;

            int_ptr(count, INO) = count+1;
            int_ptr(count, IWE) = list_ptr(i, j, LWE);
            int_ptr(count, IOR) = list_ptr(i, j, LOR);

            for(k=0, col=0; k<int_ptr(count, IOR); k++){
                v1  = list_ptr(i, j, LV1+k);
                int_ptr(count, IV1+k) = v1;
                col = conint[col*mat_cols + var_col[v1]];
                for(l=0; l<mat_cols; l++)
                    tempvect[l] *= mat[l*mat_cols + var_col[v1]];
                }

            for(l=0; l<mat_cols; l++) tempvect[l] *= mat[l*mat_cols + col];
            for(l=0, sum=0; l<mat_cols; l++) sum +=tempvect[l];
            if(abs(sum)!=mat_cols)
                error("There is some mistake in interaction data", NULL, -1);
            sign = (sum>0)? 1:-1;
            int_ptr(count, IB0) = sign*col;

            if(no_int_col[i]==1){
                int_ptr(count, IUC) = 1;
                int_ptr(count, ICO) = (int)(C_FACTOR*coeff[k]);
                }
            count++;
            }
        }
    }

/* makes BLOCK DATABASE */
pmalloc((void **) &blk_mat, MAX_BLK*sizeof(int *));
for(i=0; i<MAX_BLK; i++) blk_mat[i]=NULL;

pmalloc((void **) &blk_mat[0], sof_blkmat(0)*sizeof(int));
for(i=0; i< sof_blkmat(0); i++) *(blk_mat[0]+i)=0;

for(i=0; i<mat_cols; i++){
    blk_ptr(0, i, BCO) = (int)(C_FACTOR*(coeff[i]));
    blk_ptr(0, i, BVA) = -1;
    for(j=0; j<num_vars; j++) if(var_col[j]==i) blk_ptr(0, i, BVA) = j;

    for(j=0, count=0; j<tot_ints; j++){
        if(abs(int_ptr(j, IB0))==i){
            sign = (int_ptr(j, IB0)>0)? 1: -1;
            blk_ptr(0, i, BI1+count) = sign*int_ptr(j, INO);
            count++;
            }
        }
    blk_ptr(0, i, BNI) = count;
    }
return;
}
```

E.1.2 Str_itr.C

```
/*************************************
*
*This routine is called by Startup.c if
*there are any third-order interactions.
*The routine selects the experiments so
*that none of the variables are confounded
*with the interactions.
*
*************************************/

#include "coninc.h"
#define enough_tries 100

int successes = 0;
int *optvar, minvar, minint, matsize;
float minvsum, minisum;
int *varcol=NULL, *int_pattern=NULL, *assign=NULL;
extern int num_vars, *cints;
static int *weights, int_count, *endvar;
static int rec_str(int *, stats *);

void select_experiments(int tint_count, int *tweights, int *pmatsize,
                              int *tendvar, int **ptempmat){

int i, j, k, l, prod, sum, *tempmat=NULL, *temp_vect=NULL;
stats *statvar;

weights  = tweights;
int_count= tint_count;
endvar = tendvar;
matsize=1;
while(matsize<num_vars) matsize *=2;
*pmatsize = matsize;
pmalloc((void **) &tempmat,     matsize*matsize*sizeof(int));
pmalloc((void **) &temp_vect,     matsize*sizeof(int));
pmalloc((void **) &int_pattern, matsize*matsize*sizeof(int));

create_matrix(matsize, tempmat);

/* set int_pattern (same as conint in other programs) */
for (i=0; i<matsize; i++) for (j=0; j<matsize; j++){
   for (k=0; k<matsize; k++)
      temp_vect[k] = tempmat[k*matsize+i]*tempmat[k*matsize+j];
   prod = 1;
   for (l=0; l<matsize; l++){
      sum = 0;
      for (k=0; k<matsize; k++)
         sum += (temp_vect[k]*tempmat[k*matsize+l]);
      if (abs(sum) == matsize) {int_pattern[i*matsize+j]=l; prod=0;}
      }
   if (prod != 0) printf( "Non orthogonal matrix found" );
   }
/* variables for recon */
pmalloc((void **) &optvar,     (num_vars)*sizeof(int));
pmalloc((void **) &varcol,       (num_vars)*sizeof(int));
pmalloc((void **) &assign,     2*matsize*sizeof(int));

for(i=0; i<2*matsize; i++) assign[i] = 0;
/* note that assign has (varno+1) and not (varno) */
varcol[0]=0;
varcol[1]=1;
assign[0]=1;
assign[1]=2;
minvar     = num_vars;
minvsum = minint = minisum = 0;
statvar->set=2;
statvar->curvar = 0;
statvar->curvsum= 0;
statvar->curint = 0;
statvar->curisum= 0;

rec_str(assign, statvar);
if(minvar>0) error("Higher order matrix required", NULL, -1);

pmalloc((void **) ptempmat,     matsize*matsize*sizeof(int));
for(i=0; i<matsize*matsize; i++) (*ptempmat)[i]=0;
for(i=0; i<num_vars; i++) for(j=0; j<matsize; j++)
   *(*ptempmat + j*matsize +i)=tempmat[j*matsize+optvar[i]];

pfree((void **) &optvar);
pfree((void **) &varcol);
pfree((void **) &int_pattern);
pfree((void **) &assign);
pfree((void **) &tempmat);
pfree((void **) &temp_vect);
return;
}
```

```
/*************************************/

int check_worse( stats *statvar ){

if (minvar < statvar->curvar) return( 1 );
else if (minvar == statvar->curvar){
    if (minvsum < statvar->curvsum) return( 1 );
    else if (minvsum == statvar->curvsum){
        if (minint < statvar->curint) return( 1 );
        else if((minint==statvar->curint)&&(minisum<=statvar->curisum))
                return( 1 );
        }
    }
return( 0 );
}

/********************************/

int  update_best( stats *statvar ){

int  update = 0;
if (minvar > statvar->curvar) update = 1;
else if (minvar == statvar->curvar){
    if (minvsum > statvar->curvsum) update = 1;
    else if (minvsum == statvar->curvsum){
        if (minint > statvar->curint) update = 1;
        else if((minint==statvar->curint)&&(minisum>statvar->curisum))
            update = 1;
        }
    }
if (update > 0){
    minvar = statvar->curvar;
    minvsum = statvar->curvsum;
    minint = statvar->curint;
    minisum = statvar->curisum;
    memcpy( optvar, varcol, (num_vars)*sizeof(int) );
    }
return( update );
}

/*************************************/

rec_str(int *assign, stats *statvar){

stats backstat;
int i, j, k, l, m;
int *assign2=NULL, pres_var, done, col;

/* check if successful arrangement */
if (statvar->set >= num_vars){
    /* new record, update limits */
    update_best( statvar );
    successes++;
    i = ((successes > enough_tries)+(statvar->curvar + statvar->curint == 0));
    return(i);
    }

/* save availability vector */
pmalloc( (void **)     &assign2,     2*matsize*sizeof( int ) );
memcpy( assign2,     assign,     2*matsize*sizeof( int ) );
memcpy( &backstat, statvar,      sizeof( stats ) );

/* loops over columns in which to put next variable */
done = 0;
pres_var = statvar->set;
for (col=0; (col<matsize) && (done != 1); col++){
    if (assign[col] == 0){
        assign[col] = pres_var+1;
        varcol[pres_var] = col;
        done = 0;

        /* find amount of interaction in this column */
        if (assign[matsize+col] != 0){
            /* indicate overlap */
            statvar->curvar++;
            statvar->curvsum += abs(assign[matsize+col]);

            /* was it marked as combination of interactions?, if so, remove */
            if (assign[matsize+col] < 0){
                statvar->curint--;
                statvar->curisum -= abs(assign[matsize+col]);
                }
            done = -check_worse( statvar );
            }
        /* updates interactions with other columns */
        for (i=0; i<int_count; i++) if (endvar[i] == pres_var){
            k = i*cint_len;    /* note: dim and mat_rows might not be equal */
            l = varcol[cints[k+1]];
            for (m=1; m<cints[k]; m++)
                  l = int_pattern[l*matsize+varcol[cints[k+1+m]]];
        /* interacting columns now in l, update */
        if (assign[matsize+l] == 0){
           /* new interaction here, check if occupied */
           if (assign[l] > 0){
              statvar->curvar++;
              statvar->curvsum += weights[i];
              }
           assign[matsize+l] = weights[i];
           }
        else if (assign[matsize+l] > 0){
           /* new overlapping interaction column */
           if (assign[l] == 0){
              statvar->curint++;
              statvar->curisum += assign[matsize+l] + weights[i];
              }
           else statvar->curvsum += weights[i];
           assign[matsize+l] = -assign[matsize+l]-weights[i];
           }
        else{
           /* already overlapping interaction */
           if (assign[l] == 0) statvar->curisum += weights[i];
           if (assign[l] != 0) statvar->curvsum += weights[i];
           assign[matsize+l] -= weights[i];
           }
        }

      /* tests this arrangement */
      done = -check_worse( statvar );

      if (done == 0){
         /* next variable */
         statvar->set++;
         done = rec_str( assign, statvar );
         }
      memcpy( assign, assign2, 2 * matsize * sizeof( int ) );
      memcpy( statvar, &backstat, sizeof( stats ) );

      }
   }
pfree( (void **) &assign2 );
return( (done == 1) );
}
```

E.2.1 Block.C

```
/************************************
*
*Program: Block.C
*
*This routine uses the database and
*designs the optimal Full-Block or
*Half-Block of experiments. The algorithm
*used here is outlined in Appendices B and
*C of the report. The experiments are then
*stored in the database.
*
*Input:  Block number.
*Output: Full-Block/Half-Block Design
*
************************************/

#include "coninc1.h"
#include "coninc2.h"

/* main variables */
int num_vars, num_blocks, **expt_mat=NULL, **sel_row=NULL;
int *int_mat=NULL, *var_mat=NULL, *size_expt=NULL, *size_blk=NULL;
int **blk_mat=NULL, tot_ints, *type_blk=NULL, *blk_info=NULL;
char **var_label=NULL;
float **res_mat=NULL;
int block_no=0;

/* variables for combine.c */
intstruct *int_root=NULL;
int *var_col=NULL, mat_rows, mat_cols, *mat=NULL, *conint=NULL;
int int_count, *endvar;
stats initstat;

/* variables for re_con() */
int *optvar=NULL, *new_assign=NULL, *old_assign=NULL, *cints=NULL;
int dim;
int *old_varcol=NULL, *new_varcol=NULL, *ord_ass=NULL, full_set;
int minvar, minint;
int minvsum, minisum, *weights=NULL;
int re_con(int *, stats *);

main(){
```

```
int main_col=-1;
FILE *out_file;
void setup_intdata(void);
void setup_variables( void );
void setup_fullset( void );
int  print_results( void );
void finish_up( void );
void save_comb_data(int);
void save_new_expts(int);
void get_new_assign(void);

out_file = fopen(data_file, "rb");
if(out_file==NULL) error("Cant open INPUT file", data_file, -1);
load_database(data_file, &num_vars, &tot_ints, &num_blocks,
       &size_expt, &size_blk, &type_blk, &var_mat, &int_mat,
       &blk_mat, &sel_row, &expt_mat, &res_mat, &var_label, &blk_info);

disp_int_info(tot_ints, num_blocks, type_blk, int_mat, var_label);
disp_blk_info(num_blocks, size_expt, size_blk, blk_mat, type_blk, blk_info);

printf("Please Enter Block No:");
scanf("%d", &block_no);
if((block_no<0)||(block_no>num_blocks-1)){
   printf("Invalid Block Number .......  Exiting Program");
   exit(1);
   }
set_mat(num_vars, block_no, expt_mat, sel_row, size_blk, &mat_rows,
       &mat_cols, &var_col, &mat, &conint);
setup_intdata();

num_blocks++;
setup_variables();
blk_info[num_blocks-1] = block_no;
printf("\nSelect  (1)Full-Block Design (2)Half-Block Design?  (1/2)");
scanf("%d", &full_set);
if(full_set==1){
  type_blk[num_blocks-1]  = TYPE_FBLK;
  size_expt[num_blocks-1] = mat_cols;
  size_blk[num_blocks-1]  = 2*mat_cols;
  dim = 2*mat_cols;
  setup_fullset();
  }
else{
  type_blk[num_blocks-1]  = TYPE_HBLK;
  size_expt[num_blocks-1] = mat_cols/2;
  size_blk[num_blocks-1]  = mat_cols;
  dim = mat_cols;
  }
get_new_assign();
re_con(new_assign, &initstat);
main_col = print_results();
save_new_expts(main_col);

printf("Save Experiments to Database?  (y/n)\n");
if((getch()=='y')&&(main_col>=0)) save_comb_data(main_col);
finish_up();
return(0);
}

/************************************
*
* Sets up the interaction data.
*
************************************/

static void setup_intdata(void){

int intno, i, j, k, l;
intstruct *cur_intptr=NULL;

/* set cints, int_root and weights */
pmalloc((void **) &int_root, sizeof(intstruct));
int_root->vars = NULL;
int_root->next = NULL;
cur_intptr = int_root;

for(i=0, int_count=0; i<tot_ints; i++) if(int_ptr(i, IUC)==0) int_count++;
pmalloc( (void **) &cints, int_count*cint_len*sizeof(int));
pmalloc( (void **) &weights, int_count*sizeof(int));
for(i=0; i<int_count*cint_len; i++) cints[i]=0;

for(i=0, intno=0; i<tot_ints; i++) if(int_ptr(i, IUC)==0){
   cints_ptr(intno, COR) = int_ptr(i, IOR);
   for(j=0; j<int_ptr(i, IOR); j++)
      cints_ptr(intno, CV1+j) = int_ptr(i, IV1+j)+1;
   weights[intno] = int_ptr(i, IWE);
   /* sort columns */
   for (j=0; j<cints_ptr(intno, COR)-1; j++)
      for (k=0; k<=j; k++)
      if (cints_ptr(intno, CV1+k) > cints_ptr(intno, CV1+1+k) ){
            l = cints_ptr(intno, CV1+k);
            cints_ptr(intno, CV1+k) = cints_ptr(intno, CV1+1+k);
            cints_ptr(intno, CV1+1+k) = l;
            warning("The Int.  Variables are not in right order", NULL, -1);
            }
    /* set int_root */
    cur_intptr->order = cints_ptr(intno, COR);
    pmalloc((void **) &(cur_intptr->vars), cur_intptr->order*sizeof(int));
    for (k=0; k<cur_intptr->order; k++)
        cur_intptr->vars[k] = cints_ptr(intno, CV1+k)-1;
    cur_intptr->weight = (float) weights[intno];
    cur_intptr->type = 1;

    pmalloc( (void **) &(cur_intptr->next), sizeof( intstruct ));
    cur_intptr = cur_intptr->next;
    cur_intptr->vars = NULL;
    cur_intptr->next = NULL;
    intno++;
    }
return;
}

/*************************************
*
* Sets up the variables for re_con()
*
*************************************/

static void setup_variables(void){

int i, j;
void get_ord_ass(void);

pmalloc( (void **) &optvar, (num_vars+1)*sizeof(int));

/***old_assign ***/
pmalloc( (void **) &old_assign, mat_rows*sizeof(int));
pmalloc( (void **) &old_varcol, (num_vars+1)*sizeof(int));
for(i=0; i<mat_rows; i++) old_assign[i]=0;
for(i=0; i<(num_vars+1); i++) old_varcol[i]=0;
for(i=0; i<num_vars; i++){
    j=var_col[i];
    old_assign[j]=i+1; /** check if correct **/
    old_varcol[i+1]=j;
    }
pmalloc( (void **) &ord_ass, (num_vars+1)*sizeof(int));
for(i=0; i<(num_vars+1); i++) ord_ass[i]=0;

get_ord_ass();
return;
}

/*************************************
*
* Determines the basic variables and
* sets up ord_ass[]
*
*************************************/

static void get_ord_ass(void){

int i, j, varno, temp, power;
int *covered=NULL, selected, no_cov;

pmalloc( (void **) &covered, mat_rows*sizeof(int));
for(i=0; i<mat_rows; i++) covered[i]=0;
power=0;
temp=mat_rows;
while(temp>1) {power +=1; temp /=2;}
ord_ass[1] = 1;
ord_ass[2] = 2;
covered[1] = covered[2] = 1;
covered[conint[1*mat_rows+2]] = covered[conint[1*mat_rows+1]]=1;
no_cov=4;

/* assign the basic variables to ord_ass[] */
for(i=3, varno=3; i<power+1; i++) {
    selected=0;
    while(selected==0){
        if(varno > num_vars) error("Not enough Basic Variables", NULL, -1);
        for( j=0; j<mat_rows; j++)
            if((covered[j]==1)&&(covered[conint[j*mat_rows+old_varcol[varno]]]==0))
                selected +=1;
        if(selected==no_cov){
            ord_ass[i] = varno;
            covered[old_varcol[varno]]=1;
            for( j=0; j<mat_rows; j++)
                if (covered[j]==1)
                    covered[conint[j*mat_rows+old_varcol[varno]]]=1;
            no_cov = no_cov*2;
        }
    else selected=0;
    varno ++;
    }
}

/* assign the non basic variables to ord_ass[] */
for(i=power+1, varno=1; i<num_vars+1; i++){
  do {
    selected=1;
    for(j=1; j<i; j++) if(ord_ass[j]==varno) selected=0;
    if(selected==1) ord_ass[i]=varno;
    varno++;
    if(varno>num_vars+1) error("Something is not correct!!!", NULL, -1);
    } while(selected==0);
  }
pfree( (void **) &covered);
return;
}

/*************************************
*
* Sets new_assign[] and new_varcol[]
*
*************************************/

static void get_new_assign(void){

int i, j, k, l, m, lastvar, found_nb, power, temp, varno;

pmalloc( (void **) &new_assign, 2*dim*sizeof(int));
pmalloc( (void **) &new_varcol, (num_vars+1)*sizeof(int));
for(i=0; i<2*dim; i++)         new_assign[i]=0;
for(i=0; i<num_vars+1; i++) new_varcol[i]=0;

power=0;
temp=dim;
while(temp>1) {power +=1; temp /=2;}
if(full_set==1) power--;
else power=0;
new_assign[0]=9999;
for(i=1; i<power+1; i++){
      varno = ord_ass[i];
      new_assign[old_varcol[varno]] = varno;
      new_varcol[varno] = old_varcol[varno];
    }

/* set endvar and basic var interaction weights .... */
pmalloc( (void **) &endvar, int_count*sizeof(int));

for(i=0; i<int_count; i++){
  lastvar=cints[i*cint_len];
  while(lastvar>0){
    found_nb=1;
    endvar[i]=cints[i*cint_len+lastvar];
    for(k=1; k<power+1; k++) if(endvar[i] == ord_ass[k]){
      lastvar -=1;
      found_nb=-1;
      }
    if(found_nb>0) break;
    }
  if(lastvar == 0){
    k=i*cint_len;
    l=new_varcol[cints[k+1]];
    for(m=1; m<cints[k]; m++) l=conint[l*dim+ new_varcol[cints[k+1+m]]];
    new_assign[dim+l] = weights[i];
    }
  }

minvar = mat_rows;
initstat.set=power;
initstat.curvar = initstat.curisum = 0;
initstat.curint = initstat.curisum = 0;
return;
}

/*************************************
*
* frees arrays
*
*************************************/

static void finish_up(void){

int i;

for(i=0; i<num_vars; i++) pfree((void **) &(var_label[i]));
pfree( (void **) &mat);
pfree( (void **) &var_col);
pfree( (void **) &conint);
pfree( (void **) &var_label);
pfree( (void **) &expt_mat);
pfree( (void **) &res_mat);
pfree( (void **) &size_expt);
pfree( (void **) &blk_mat);
pfree( (void **) &sel_row);
pfree( (void **) &cints);
pfree( (void **) &weights);
pfree( (void **) &endvar);
pfree( (void **) &old_assign);
pfree( (void **) &new_assign);
pfree( (void **) &old_varcol);
pfree( (void **) &new_varcol);
pfree( (void **) &ord_ass);
pfree( (void **) &optvar);

}

/*************************************
*
*  Prints Final Results ...
*
*************************************/

static int print_results( void){

int i, j, k, *var_list=NULL, max_label=0, main_col = -1;
char f_name[100];
stats mat_stats;
FILE *out_handle;

out_handle = stdout;

printf("Do you want to save the screen output?  (y/n)  ");
if(getch()=='y'){
   printf("\nEnter output file name:  ");
   fflush(stdin);
   gets(f_name);
   out_handle = fopen(f_name, "w");
   if(out_handle==NULL){
      warning( "Cant open output file", NULL, -1);
      out_handle = stdout;
      }
   }
fprintf( out_handle, file_divider );
fprintf( out_handle, "OPTIMAL COMBINATION IS: \n");
for(i=1; i<num_vars+1; i++){
   fprintf(out_handle, "Var:  %s \t  Column:  %d \t", var_label[i-1], optvar[i]);
   if(old_varcol[i]==optvar[i]) fprintf( out_handle, "Same col\n");
   else fprintf( out_handle, "Shifted w.r.t.  Col:  %d\n",
            (main_col = conint[dim*old_varcol[i]+optvar[i]]));
   }
fprintf( out_handle, file_divider );
if(out_handle == stdout) getch();
fprintf( out_handle, file_divider );

/* set_up new experiment. main_col=-1 implies no expts. required */
if(main_col!=-1){
   pmalloc( (void **) &var_list, (num_vars+1)*sizeof(int));
   for(i=0; i<num_vars; i++) var_list[i]=i;
   var_list[num_vars]=-1;
   for(i=0; i<num_vars; i++) max_label=max( max_label, strlen(var_label[i]));
   max_label++;

   fprintf( out_handle, "Old Overlap Information:\n");
   fprintf( out_handle, "------------------------\n");
   get_matrix_stats( dim, mat, num_vars, var_list, old_varcol+1,
         int_root, &mat_stats, 1, out_handle, var_label, max_label );

   fprintf( out_handle, "New Overlap Information:\n");
   fprintf( out_handle, "------------------------\n");
   get_matrix_stats( dim, mat, num_vars, var_list, optvar+1, int_root,
            &mat_stats, 1, out_handle, var_label, max_label );
   fprintf( out_handle, file_divider );
   if(out_handle == stdout) getch();

   pfree( (void **) &var_list);
   }
if(out_handle != stdout) fclose(out_handle);
return(main_col);
}

/*************************************
*
* Stores the new experiments in expt_mat[]
*
*************************************/

static void save_new_expts(int main_col){
```

```
int i, j, line_no, max_label=0;
char format_str[100], temp_str[100];
FILE *out_handle;

out_handle=stdout;
if(main_col<0){
   printf("No New Experiments Required.  \n");
   printf(file_divider);
   size_expt[num_blocks−1] = 0;
   size_blk[num_blocks−1]  = 0;
   type_blk[num_blocks −1] = 0;
   num_blocks−−;
   return;
   }

for(i=0; i<num_vars; i++) max_label = max( max_label, strlen(var_label[i]));
max_label++;

expt_mat[num_blocks−1]=NULL;
pmalloc( (void **) &(expt_mat[num_blocks−1]),
              sof_exptmat(num_blocks−1)*sizeof(int));
sprintf( format_str, "%%%ds ", max_label );
fprintf( out_handle, "\nNew set of %d Experiments:\n\t\t", dim/2 );
for (i=0; i<num_vars; i++)
  fprintf( out_handle, format_str, var_label[i] );
fprintf( out_handle, "\n" );
sprintf( temp_str, "%%%ds%%%%s%%%ds", max_label/2, max_label−max_label/2 );
sprintf( format_str, temp_str, " ", " " );

if(full_set!=1){
   for (i=0, line_no=0; i<dim; i++)
      if(mat[i*dim + main_col] == −1){
         fprintf( out_handle, "Expt #:  %2d\t", line_no+1);
         for (j=0; j<num_vars; j++){
            fprintf( out_handle, format_str,
               (mat[i*dim + optvar[j+1]] == 1) ? "+" : "-" );
            *(expt_ptr(num_blocks−1, line_no)+j)=mat[i*dim + optvar[j+1]];
            }
         line_no++;
         fprintf( out_handle, "\n" );
         }
   }
else{
   for (i=dim/2, line_no=0; i<dim; i++){
         fprintf( out_handle, "Expt #:  %2d\t", line_no+1);
         for (j=0; j<num_vars; j++){
            fprintf( out_handle, format_str,
               (mat[i*dim + optvar[j+1]] == 1) ? "+" : "-" );
            *(expt_ptr(num_blocks−1, line_no)+j)=mat[i*dim + optvar[j+1]];
            }
         line_no++;
         fprintf( out_handle, "\n" );
         }
   }
fprintf( out_handle, file_divider );
return;
}

/*************************************
 *
 * Increases the sizes of the variables for
 * the Full_Block of experiments.
 *
 *************************************/

static void setup_fullset( void ){

int *t_conint=NULL, *t_mat=NULL, i, j;

pmalloc( (void **) &t_conint, mat_rows*mat_rows*sizeof(int));
pmalloc( (void **) &t_mat, mat_rows*mat_rows*sizeof(int));
memcpy(t_conint, conint, mat_rows*mat_rows*sizeof(int));
memcpy(t_mat, mat, mat_rows*mat_rows*sizeof(int));
pfree( (void **) &conint);
pfree( (void **) &mat);
pmalloc( (void **) &conint, dim*dim*sizeof(int));
pmalloc( (void **) &mat, dim*dim*sizeof(int));
for(i=0; i<dim*dim; i++){ conint[i]=0;     mat[i]=0; }

/* setting new conint */
for(i=0; i<dim/2; i++){
   for(j=0; j<dim/2; j++) conint[i*dim+j] = t_conint[i*(dim/2)+j];
   for(j=dim/2; j<dim; j++)
      conint[i*dim+j] = t_conint[i*(dim/2)+j−dim/2] + dim/2;
   }
for(i=dim/2; i<dim; i++){
   for(j=0; j<dim/2; j++)
      conint[i*dim+j] = t_conint[(i−dim/2)*(dim/2)+j] + dim/2;
   for(j=dim/2; j<dim; j++)
```

```
        conint[i*dim+j] = t_conint[(i-dim/2)*(dim/2)+j-dim/2];
    }

/* setting new mat */
for(i=0; i<dim/2; i++)
    for(j=0; j<dim/2; j++)
        mat[i*dim+j] = t_mat[i*(dim/2)+j];
for(i=dim/2; i<dim; i++)
    for(j=0; j<dim/2; j++)
        mat[i*dim+j] = t_mat[(i-dim/2)*(dim/2)+j];
for(i=0; i<dim/2; i++) mat[i*dim + dim/2] = 1;
for(i=dim/2; i<dim; i++) mat[i*dim + dim/2] = -1;
for(j=dim/2+1; j<dim; j++) for(i=0; i<dim; i++)
    mat[i*dim+j] = mat[i*dim+dim/2]*mat[i*dim+j-dim/2];

pfree( (void **) &t_mat);
pfree( (void **) &t_conint);
return;
}

/**************************************
*
* Updates information and store it in the
* database.
*
**************************************/

static void save_comb_data(int main_col){

int i, count;

/* set sel_row */
sel_row[num_blocks-1]=NULL;
pmalloc((void **) &(sel_row[num_blocks-1]),
            sof_selrow(num_blocks-1)*sizeof(int));

if(type_blk[num_blocks-1]==TYPE_HBLK){
    for(i=0, count=0; i<dim; i++) if(mat[i*dim+main_col]==1){
        sel_ptr(num_blocks-1, count, SBL) = sel_ptr(block_no, i, SBL);
        sel_ptr(num_blocks-1, count, SEP) = sel_ptr(block_no, i, SEP);
        count++;
        }
    if(count!=dim/2) error("Some problem with data!!!", NULL, -1);

    for(i=0; i<dim/2; i++){
        sel_ptr(num_blocks-1, count, SBL) = num_blocks-1;
        sel_ptr(num_blocks-1, count, SEP) = i;
        count++;
        }
    }
if(type_blk[num_blocks-1]==TYPE_FBLK){
    for(i=0; i<dim/2; i++){
        sel_ptr(num_blocks-1, i, SBL) = sel_ptr(block_no, i, SBL);
        sel_ptr(num_blocks-1, i, SEP) = sel_ptr(block_no, i, SEP);
        }
    for(i=dim/2; i<dim; i++){
        sel_ptr(num_blocks-1, i, SBL) = num_blocks-1;
        sel_ptr(num_blocks-1, i, SEP) = i-dim/2;
        }
    }
save_database(data_file, num_vars, tot_ints, num_blocks,
            size_expt, size_blk, type_blk, var_mat, int_mat,
            blk_mat, sel_row, expt_mat, res_mat, var_label, blk_info);
return;
}
```

E.2.2 Block_itr.C

```
/************************************
*
*This routine is called by Block.C. It
*searches over all the possible combinations
*of new designs and selects the optimal one.
*The procedure is described in Appendix B
*and Appendix C.
*
************************************/

#include "coninc.h"
#define enough_tries 100

/* variables for re_con() */
extern int *optvar, *old_assign, *cints, *old_varcol, full_set, dim;
extern int num_vars, *ord_ass, *new_varcol, *endvar, *conint, int_count;
extern int minvar, minint;
extern int *weights, minvsum, minisum;
int shifted_wrt_col, shifted_col=0, successes = 0;
long pass = 0;

/************************************/

int check_worse( stats *statvar ){

if (minvar < statvar->curvar) return( 1 );
else if (minvar == statvar->curvar){
   if (minvsum < statvar->curvsum) return( 1 );
   else if (minvsum == statvar->curvsum){
      if (minint < statvar->curint) return( 1 );
      else if((minint == statvar->curint)&&(minisum<= statvar->curisum))
         return( 1 );
      }
   }
return( 0 );
}

/************************************/

int   update_best( stats *statvar ){

int   update = 0;
if (minvar > statvar->curvar) update = 1;
else if (minvar == statvar->curvar){
   if (minvsum > statvar->curvsum) update = 1;
   else if (minvsum == statvar->curvsum){
      if (minint > statvar->curint) update = 1;
      else if((minint == statvar->curint)&&(minisum>statvar->curisum))
         update = 1;
      }
   }
if (update > 0){
   minvar = statvar->curvar;
   minvsum = (int) statvar->curvsum;
   minint = statvar->curint;
   minisum = (int) statvar->curisum;
   printf("minvar:  %2d  minvsum:  %4d ", minvar, minvsum);
   printf("minint:  %2d  minisum:  %3d \n", minint, minisum);
   memcpy( optvar, new_varcol, (num_vars+1)*sizeof(int) );
   }
return( update );
}

/************************************/

re_con( int *assign, stats *statvar ){
stats backstat;
int i, j, k, l, m;
int *assign2=NULL, pres_var, done, col, col_ok, just_shifted;

/* check if successful arrangement */
if (statvar->set >= num_vars){
   update_best( statvar );
   successes++;
   i=((successes>enough_tries)+(statvar->curvar+statvar->curint==0));
   return(i);
   }
/* save availability vector */
pmalloc( (void **) &assign2, 2 * dim * sizeof( int ) );
memcpy( assign2, assign, 2 * dim * sizeof( int ) );
memcpy( &backstat, statvar, sizeof( stats ) );

/* loop over columns in which to put next variable */
done = 0;
pres_var = ord_ass[statvar->set+1];
for (col=0; (col<dim) && (done != 1); col++){
   col_ok=0; just_shifted=0;
   if(full_set == 1){
        if(col == old_varcol[pres_var]) col_ok=1;
        if(col == old_varcol[pres_var] + dim/2) col_ok=1;
        shifted_wrt_col = dim/2;
        }
    else{
        if((shifted_col == 1)&&(assign[col] == 0)){
            if(col == old_varcol[pres_var]) col_ok=1;
            if(col == conint[old_varcol[pres_var]*dim + shifted_wrt_col])
                col_ok=1;
            }
        else if((shifted_col==0)&&(assign[col]==0)){
            col_ok=1;
            just_shifted=1;
            shifted_col=1;
            shifted_wrt_col = conint[old_varcol[pres_var]*dim+col];
            }
        }

    if ((assign[col] == 0)&&(col_ok == 1)){
        assign[col] = pres_var;
        new_varcol[pres_var] = col;
        done = 0;

        /* find amount of interaction in this column */
        if (assign[dim+col] != 0){
            /* indicate overlap */
            statvar->curvar++;
            statvar->curvsum += abs(assign[dim+col]);

            /* was it marked as combination of ints?, if so, remove */
            if (assign[dim+col] < 0){
                statvar->curint--;
                statvar->curisum -= abs(assign[dim+col]);
                }
            done = -check_worse( statvar );
            }

        /* update interactions with other columns */
        for (i=0; i<int_count; i++) if (endvar[i] == pres_var){
            k = i*cint_len;
            l = new_varcol[cints[k+1]];
            for (m=1; m<cints[k]; m++)
                l=conint[l*dim+new_varcol[cints[k+1+m]]];

            /* interacting columns now in l, update */
            if (assign[dim+l] == 0){
                /* new interaction here, check if occupied */
                if (assign[l] > 0){
                    statvar->curvar++;
                    statvar->curvsum += (float) weights[i];
                    }
                assign[dim+l] = weights[i];
                }
            else if (assign[dim+l] > 0){
                /* new overlapping interaction column */
                if (assign[l] == 0){
                    statvar->curint++;
                    statvar->curisum += (float) (assign[dim+l] + weights[i]);
                    }
                else statvar->curvsum += (int) weights[i];
                assign[dim+l] = -assign[dim+l] - weights[i];
                }
            else{
                /* already overlapping interaction */
                if (assign[l] == 0) statvar->curisum += (float) weights[i];
                if (assign[l] != 0) statvar->curvsum += (float) weights[i];
                assign[dim+l] -= weights[i];
                }
            }

        /* test this arrangement */
        done = -check_worse( statvar );
        if (done == 0){
            /* next variable */
            statvar->set++;
            done = re_con( assign, statvar );
            }
        memcpy( assign, assign2, 2 * dim * sizeof( int ) );
        memcpy( statvar, &backstat, sizeof( stats ) );
        if(just_shifted) shifted_col=0;
        }
    }
/* finish up */
pfree( (void **) &assign2 );
return( (done == 1) );
}
```

E.2.3 Comp_Blk.C

```
/*************************************
*
*This program implements the Complete
*Block Design Algorithm described in
*Appendix D. Given a Half-Block Design,
*it designs the missing block of
*experiments.
*
*The program performs the following tasks:
*1) Asks the user for a Half-Block Design.
*2) Determines the basic variables and the
*   confounding pattern of the design.
*3) Determines the missing block of expts.
*4) Updates the database with these expts.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"

/* main variables */
int num_vars,  num_blocks, **expt_mat=NULL, **sel_row=NULL;
int *int_mat=NULL, *var_mat=NULL, *size_expt=NULL, *size_blk=NULL;
int **blk_mat=NULL, tot_ints, *type_blk=NULL, *blk_info=NULL;
char **var_label=NULL;
float **res_mat=NULL;

/* variables for comp_blk.c */
int mat_rows, mat_cols, dim, *mat=NULL, *conint=NULL, *var_col=NULL;
int *comp_mat=NULL, block_no, expt_no, save_data;
FILE *out_file;

main(){

void setup_matrix(void);
void save_new_expts(void);
void finish_up(void);
void save_comp_data(int);
void complete_block(int **);

out_file = fopen(data_file, "rb");
if(out_file==NULL) error("Cant open INPUT file", data_file, -1);
printf(file_divider);
load_database(filen, &num_vars, &tot_ints, &num_blocks,
        &size_expt, &size_blk, &type_blk, &var_mat, &int_mat,
        &blk_mat, &sel_row, &expt_mat, &res_mat, &var_label, &blk_info);
printf("Enter Half Block No:");
scanf("%d", &expt_no);
if(type_blk[expt_no] != TYPE_HBLK)
   error("Block is not a Half-Block design", NULL, -1);
block_no = blk_info[expt_no];
setup_matrix();
num_blocks++;
blk_info[num_blocks-1] = expt_no;
type_blk[num_blocks-1] = TYPE_CBLK;
size_expt[num_blocks-1] = mat_cols - mat_rows;
size_blk[num_blocks-1]     = mat_cols;
complete_block(&comp_mat);
printf("Do you want to save database?  (y/n)");
if(gettch()=='y') save_data=1;
else save_data=0;

save_comp_data(save_data);
finish_up();
return;
}

/*************************************
*
* Determines the basic variables and the
*  missing block of expts.
*
*************************************/

static void complete_block(int **pcomp_mat){

int i, j, k, l, *basic_var=NULL, *temp_mat=NULL, multiplier, row;
int power=0, *covered=NULL, *temp_vect=NULL, no_cov, varno, selected;

pmalloc((void **) &covered, mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) covered[i]=0;
pmalloc((void **) &temp_vect, (mat_cols-mat_rows)*sizeof(int));

pmalloc((void **) &temp_mat, (mat_cols-mat_rows)*mat_cols*sizeof(int)),
for(i=0; i<(mat_cols-mat_rows); i++)
   for(j=0; j<mat_cols; j++) temp_mat[i*mat_cols +j]=0;
```

```
*pcomp_mat = temp_mat;
l=mat_cols;
while(l>1) {l/=2; power+=1;}
covered[0]=1;    /* avg. col */
no_cov=1;
varno=0;
pmalloc((void **) &basic_var, power*sizeof(int));

/* determine the basic variables */
for(i=0; i<power; i++) {
  selected=0;
  while(selected==0){
    if(varno > num_vars) error("Not enough Basic Variables", NULL, -1);
    for( j=0; j<mat_cols; j++)
      if ((covered[j]==1)&&
          (covered[conint[j*mat_cols+var_col[varno]]]==0)) selected +=1;
    if(selected==no_cov){
      covered[var_col[varno]]=1;
      for( j=0; j<mat_rows; j++)
        if (covered[j]==1)
          covered[conint[j*mat_cols+var_col[varno]]]=1;
      basic_var[i] = varno;
      no_cov = no_cov*2;
      }
    else selected=0;
    varno++;
    }
  }

/* determine which combinations of the basic variables are missing */
for(i=0; i<mat_cols; i++) covered[i]=0;
for(i=0; i<mat_rows; i++){
  for(j=0, no_cov=0, multiplier=1; j<power; j++){
    if(mat[i*mat_cols+var_col[basic_var[j]]]==1) no_cov += multiplier;
    multiplier *=2;
    }
  covered[no_cov]=1;
  }

/* add the missing combinations to temp_mat */
for(i=0, row=0; i<mat_cols; i++){
  if(covered[i]==0){
    no_cov=i;
    for(j=0; j<power; j++){
      if(no_cov & 1) temp_mat[row*mat_cols + var_col[basic_var[j]]] =1;
      else temp_mat[row*mat_cols + var_col[basic_var[j]]] =-1;
      no_cov>>=1;
      }
    row++;
    }
  }
/* completes temp_mat */
for(i=0; i<mat_cols-mat_rows; i++) temp_mat[i*mat_cols]=1;
for(i=0; i<mat_cols; i++) covered[i]=0;
covered[0]=1;
for(i=0; i<power; i++) covered[var_col[basic_var[i]]] = 1;
for(i=0; i<mat_cols; i++){
  for(j=i+1; j<mat_cols; j++){
    l=conint[i*mat_cols+j];
    if((covered[i]==1)&&(covered[j]==1)&&(covered[l]==0)){
      for(k=0; k<mat_cols-mat_rows; k++)
        temp_mat[k*mat_cols+l] =
          temp_mat[k*mat_cols+i]*temp_mat[k*mat_cols +j];
      covered[l]=1;
      }
    }
  }
pfree((void **) &basic_var);
pfree((void **) &temp_mat);
pfree((void **) &temp_vect);
pfree((void **) &covered);
return;
}

/*************************************
*
* Prints missing block of experiments.
*
*************************************/

static void print_block(char **var_label, int num_vars,
                        int blk_rows, int *print_mat){

char format_str[100], temp_str[100];
int i, j, max_label=0;
FILE *out_handle;

out_handle=stdout;
for(i=0; i<num_vars; i++) max_label = max( max_label, strlen(var_label[i]));
max_label++;
```

```
 fprintf( out_handle, file_divider);
 sprintf( format_str, "%%%ds ", max_label );
 fprintf( out_handle, "\nNew set of %d Experiments:\n\n", blk_rows );
 for (i=0; i<num_vars; i++)
   fprintf( out_handle, format_str, var_label[i] );
 fprintf( out_handle, "\n" );
 sprintf( temp_str, "%%%ds%%%%s%%%ds", max_label/2, max_label−max_label/2 );
 sprintf( format_str, temp_str, " ", " " );

 for (i=0; i<blk_rows; i++){
     for (j=0; j<num_vars; j++)
         fprintf(out_handle, format_str, (print_mat[i*num_vars+j]==1)?"+" : "-");
     fprintf( out_handle, "\n" );
     }
 fprintf( out_handle, file_divider);
 if( out_handle == stdout) getch();
 return;
 }

/*************************************
*
* Sets up the matrices mat and conint
*
*************************************/

static void setup_matrix(void){

int i, j, k, l, eq, sum, prod, varno, colno, *temp_vect=NULL;
int row, blk, power;

mat_rows=size_blk[block_no]+size_expt[expt_no];
mat_cols=1;
while(mat_cols < mat_rows){ mat_cols *=2; power +=1;}
pmalloc( (void **) &mat, mat_rows*mat_cols*sizeof(int));
pmalloc( (void **) &conint, mat_cols*mat_cols*sizeof(int));
pmalloc( (void **) &var_col, num_vars*sizeof(int));
pmalloc( (void **) &temp_vect, mat_rows*sizeof(int));
for(i=0; i<mat_rows; i++) for(j=0; j<mat_cols; j++) mat[i*mat_cols+j]=0;
for(i=0; i<mat_rows; i++) mat[i*mat_cols]=1;

/*** assign variables to mat ***/
for(i=0; i<size_blk[block_no]; i++){
    blk=sel_ptr(block_no, i, SBL);
    row=sel_ptr(block_no, i, SEP);
    memcpy( (mat + i*mat_cols+1), expt_ptr(blk, row), num_vars*sizeof(int));
    }
for(i=size_blk[block_no]; i<mat_rows; i++) memcpy((mat+i*mat_cols+1),
        expt_ptr(expt_no, i−size_blk[block_no]), num_vars*sizeof(int));

for(i=0; i<num_vars; i++) var_col[i]=i+1;

/* complete matrix mat */
colno=num_vars+1;
i = 0;  j = 1;
while (colno < mat_cols){
    /* take product of matrix columns */
    for (k=0; k<mat_rows; k++)
        temp_vect[k]=mat[k*mat_cols+i]*mat[k*mat_cols+j];
    /* look for vector in matrix */
    k = eq = 0;
    while ((k<colno) && (abs(eq)<mat_rows)){
        for (l=0, eq=0; l<mat_rows; l++) eq+=mat[l*mat_cols+k] * temp_vect[l];
        if (abs(eq) < mat_rows) k++;
        }
    if (abs(eq) < mat_rows){
        /* found new column, add it */
        for (l=0; l<mat_rows; l++) mat[l*mat_cols+colno] = temp_vect[l];
        colno++;
        }

    /* advance to next column pair */
    j++;
    if (j == colno){
        i++;  j = i+1;
        if (j == colno)
            error( "Enough ind.  columns not found in matrix.", NULL, −1 );
        }
    }
/* determine conint **/
for (i=0; i<mat_cols; i++) for (j=0; j<mat_cols; j++){
    for (k=0; k<mat_rows; k++)
        temp_vect[k]=mat[k*mat_cols+i]*mat[k*mat_cols+j];
    for (l=0, prod=1; l<mat_cols; l++){
        for (k=0, sum=0; k<mat_rows; k++)
            sum+=(temp_vect[k]*mat[k*mat_cols+l]);
        if (abs(sum) == mat_rows) { conint[i*mat_cols+j] = l; prod = 0; }
        }
    if (prod != 0) error( "Non orthogonal matrix found", NULL, −1 );
    }
```

```
pfree((void **) &temp_vect);
return;
}

/*************************************
*
* Updates the  database
*
*************************************/

static void save_comp_data(int save_data){

int i, j, count;
void print_block(char **, int, int, int *);

/* set expt_mat */
expt_mat[num_blocks-1]=NULL;
pmalloc((void **) &(expt_mat[num_blocks-1]),
          sof_exptmat(num_blocks-1)*sizeof(int));

for(i=0; i<mat_cols-mat_rows; i++)
    for(j=0; j<num_vars; j++)
        *(expt_ptr((num_blocks-1), i) +j) = comp_mat[i*mat_cols+var_col[j]];
printf("\n");
print_block(var_label, num_vars, mat_cols-mat_rows, expt_mat[num_blocks-1]);

if(save_data==1){
    /* set sel_row */
    sel_row[num_blocks-1]=NULL;
    pmalloc((void **) &(sel_row[num_blocks-1]),
              sof_selrow(num_blocks-1)*sizeof(int));
    for(i=0, count=0; i<size_blk[block_no]; i++){
        sel_ptr(num_blocks-1, count, SBL) = sel_ptr(block_no, i, SBL);
        sel_ptr(num_blocks-1, count, SEP) = sel_ptr(block_no, i, SEP);
        count++;
        }
    for(i=0; i<size_expt[expt_no]; i++){
        sel_ptr(num_blocks-1, count, SBL) = expt_no;
        sel_ptr(num_blocks-1, count, SEP) = i;
        count++;
        }
    for(i=0; i<size_expt[num_blocks-1]; i++){
        sel_ptr(num_blocks-1, count, SBL) = num_blocks-1;
        sel_ptr(num_blocks-1, count, SEP) = i;
        count++;
```

```
        }
    save_database(data_file, num_vars, tot_ints, num_blocks,
                  size_expt, size_blk, type_blk, var_mat, int_mat,
                  blk_mat, sel_row, expt_mat, res_mat, var_label, blk_info);
    }
return;
}

/*************************************
*
* Free pointers and arrays
*
*************************************/

static void finish_up(void){

int i;
for(i=0; i<num_vars; i++) pfree((void **) &(var_label[i]));
pfree( (void **) &mat);
pfree( (void **) &var_col);
pfree( (void **) &conint);
pfree( (void **) &var_label);
pfree( (void **) &expt_mat);
pfree( (void **) &res_mat);
pfree( (void **) &size_expt);
return;
}
```

E.2.4 Blk_anl.C

```
*************************************
*
*Program: Blk_anl.C
*
*This program analyzes the results of the
*block of experiments. It updates the
*interaction coefficients and sorts
*confounding interactions.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"

/* main variables */
int num_vars, num_blocks, **expt_mat=NULL, **sel_row=NULL;
int *int_mat=NULL, *var_mat=NULL, *size_expt=NULL, *size_blk=NULL;
int **blk_mat=NULL, tot_ints, *type_blk=NULL, *blk_info=NULL;
char **var_label=NULL;
float **res_mat=NULL;
int block_no=0;
int *conint=NULL, *var_col=NULL;

main(){
void load_res_data(char *);
void finish_up(void);
void analyse_data();

load_res_data(data_file);
printf("Enter Block NO:");
scanf("%d", &block_no);
if((block_no>num_blocks-1)||(block_no<0)){
   printf("Invalid Block Selected for Analysis .....  exiting Analysis\n");
   exit(1);
   }
if(blk_mat[block_no]!=NULL){
   printf("Block Analyzed before .....  exiting Analysis\n");
   exit(1);
   }
analyse_data();
finish_up();
return;
}

/*************************************
*
* Reads the information from the database.
*
*************************************/

static void load_res_data(char *filen){

int blk, row;
FILE *out_file;
void get_plant_output(void);

out_file = fopen(filen, "rb");
if(out_file==NULL) error("Cant open INPUT file", filen, -1);
printf(file_divider);
load_database(data_file, &num_vars, &tot_ints, &num_blocks,
        &size_expt, &size_blk, &type_blk, &var_mat, &int_mat,
        &blk_mat, &sel_row, &expt_mat, &res_mat, &var_label, &blk_info);

/** gets the output of the new experiments **/
for(blk=0; blk<num_blocks; blk++)
   if(res_mat[blk]==NULL){
      pmalloc((void **) &(res_mat[blk]), sof_resmat(blk)*sizeof(float));
      printf("Getting Results of Block %d\n", blk);
      for(row=0; row<size_expt[blk]; row++){
         actual_plant_result(1, num_vars, expt_ptr(blk, row),
            &(res_ptr(blk, row)));
         printf("Expt:  %d \tResult:  %7.2f\n", row, res_ptr(blk, row));
         }
      }
return;
}

/*************************************
*
* Analyzes the block of experiments
*
*************************************/

static void analyse_data(){

int i, j, k, row, blk, tot_rows, tot_cols, *big_mat=NULL, *overlap=NULL;
float *big_res=NULL, *big_covar=NULL, *big_coeff=NULL;
void update_database(int, int, int *, float *);
```

```
set_mat(num_vars, block_no, expt_mat, sel_row, size_blk, &tot_rows,
        &tot_cols, &var_col, &big_mat, &conint);

pmalloc((void **) &big_covar,     tot_cols*tot_cols*sizeof(float));
pmalloc((void **) &big_res,     tot_rows*sizeof(float));
pmalloc((void **) &big_coeff,     tot_cols*sizeof(float));
pmalloc((void **) &overlap,     tot_cols*sizeof(int));

/* assign variables to big_mat */
for(i=0; i<tot_rows; i++){
    blk=sel_ptr(block_no, i, SBL);
    row=sel_ptr(block_no, i, SEP);
    big_res[i] = res_ptr(blk, row);
    }
get_coefficients(tot_rows, tot_cols, big_mat, big_res, big_coeff,
                    big_covar, overlap);
update_database(tot_rows, tot_cols, big_mat, big_coeff);

pfree( (void **) &big_mat);
pfree( (void **) &big_res);
pfree( (void **) &big_coeff);
pfree( (void **) &big_covar);
pfree( (void **) &overlap);
return;
}


/*************************************
 *
 * Updates the database and sorts the
 * interactions.
 *
 *************************************/

static void update_database(int mat_rows, int mat_cols, int *mat, float *coeff){

int i, j, k, l, sum,  *temp_vect=NULL;
int sign, *no_int_col=NULL, *tot_int_col=NULL, col, varno, intno, intname;
int prod, count, *update_int=NULL;
float *net_coeff=NULL;

pmalloc((void **) &(blk_mat[block_no]), sof_blkmat(block_no)*sizeof(int));
for(i=0; i<sof_blkmat(block_no); i++) *(blk_mat[block_no] + i) = 0;
pmalloc((void **) &temp_vect,     mat_rows*sizeof(int));
pmalloc((void **) &net_coeff,     mat_cols*sizeof(float));
pmalloc((void **) &no_int_col,    mat_cols*sizeof(int));
pmalloc((void **) &tot_int_col,   mat_cols*sizeof(int));
pmalloc((void **) &update_int,    tot_ints*sizeof(int));
for(i=0; i<mat_cols; i++) net_coeff[i]  = coeff[i];
for(i=0; i<mat_cols; i++) no_int_col[i] = 0;
for(i=0; i<mat_cols; i++) tot_int_col[i] = 0;
for(i=0; i<tot_ints; i++) update_int[i] = 0;

for(i=0; i<mat_cols; i++) blk_ptr(block_no, i, BCO) = (int)(coeff[i]*C_FACTOR);
for(i=0; i<mat_cols; i++) blk_ptr(block_no, i, BVA) = -1;
for(i=0; i<num_vars; i++) blk_ptr(block_no, var_col[i], BVA) = i;


/* determines net effect on the columns */
for(i=0; i<num_vars; i++) net_coeff[var_col[i]] -= var_ptr(i, VCO)/C_FACTOR;

/* determines cols and sign of the interactions */
for(i=0; i<tot_ints; i++){
    col=0;
    for(k=0; k<mat_rows; k++) temp_vect[k]=1;
    for(j=0; j<int_ptr(i, IOR); j++){
        varno=int_ptr(i, IV1+j);
        for(k=0; k<mat_rows; k++) temp_vect[k]*=mat[k*mat_cols+var_col[varno]];
        col = conint[col*mat_cols + var_col[varno]];
        }
    for(k=0, sum=0; k<mat_rows; k++) sum += temp_vect[k]*mat[k*mat_cols+col]);
    if(abs(sum)!= mat_rows) error("Check Block !!", NULL, -1);
    sign = (sum>0) ? 1: -1;
    int_ptr(i, IB0+block_no) = sign*col;

    blk_ptr(block_no, col, BNI)+=1;
    intno =    blk_ptr(block_no, col, BNI);
    blk_ptr(block_no, col, BI1+intno-1) = sign*int_ptr(i, INO);

    if(int_ptr(i, IUC)==1) net_coeff[col] -= sign*coeff[col];
    else no_int_col[col]++;
    }

for(i=0; i<mat_cols; i++) if(no_int_col[i]==1){
    intname = abs(blk_ptr(block_no, i, BI1));
    for(j=0, intno=-1; j<tot_ints; j++) if(intname==int_ptr(j, INO)) intno=j;
    if(intno<0) error("Interaction Not found", NULL, -1);

    int_ptr(intno, IUC) = 1;
    int_ptr(intno, ICO) = (int) (net_coeff[i]*C_FACTOR);
    if(int_ptr(intno, IB0+block_no)<0) int_ptr(intno, ICO) *= -1;
    net_coeff[i]=0.0;
    update_int[intno] = 1;

    printf("Int ", intname);
    for(j=0; j<int_ptr(intno, IOR); j++)
        printf("%s ", var_label[int_ptr(intno, IV1+j)]);
    printf("\tCoeff:  %7.2f\n", int_ptr(intno, ICO)/C_FACTOR);
    no_int_col[i]--;
    }

printf("\n# Unknown Ints \tNet Coeff\n");
for(i=0; i<mat_cols; i++)
    printf("%d                  \t%7.2f\n", no_int_col[i], net_coeff[i]);
fflush(stdin); getch();

for(i=0, count=0; i<tot_ints; i++) if(update_int[i]==1) count++;
if(count>0) sort_interactions(block_no, update_int, num_vars, tot_ints,
            num_blocks, size_blk, type_blk, var_mat, int_mat, blk_mat);

printf("Do you want to save the database?  (y/n)");
fflush(stdin);
if(getch()=='y') save_database(data_file, num_vars, tot_ints, num_blocks,
                size_expt, size_blk, type_blk, var_mat, int_mat,
                blk_mat, sel_row, expt_mat, res_mat, var_label, blk_info);

pfree((void **) &temp_vect);
return;
}

/*************************************
*
* Frees the arrays
*
*************************************/

static void finish_up(void){

pfree( (void **) &expt_mat);
pfree( (void **) &res_mat);
pfree( (void **) &blk_mat);
pfree( (void **) &sel_row);
pfree( (void **) &size_expt);
pfree( (void **) &size_blk);
pfree( (void **) &type_blk);
pfree( (void **) &int_mat);
pfree( (void **) &var_mat);
pfree( (void **) &var_label);
pfree( (void **) &var_col);
pfree( (void **) &conint);
return;
}
```

E.3.1 OAAT.C

```
/*************************************
*
*Program: OAAT.C
*
*This program performs the One-at-a-Time
*algorithm outlined in the report.
*The following tasks are performed:
*1) Read the database
*2) Determine number of unknown interactions
*   on each column of the matrix.
*3) Generate optimum designs based on the
*   current hypothesized process model.
*4) Update the database and save it.
*
*The program prompts the user for various
*inputs required.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"
extern int fileline;

/* main variables */
int num_vars,  num_blocks, **expt_mat=NULL, **sel_row=NULL;
int *int_mat=NULL, *var_mat=NULL, *size_expt=NULL, *size_blk=NULL;
int **blk_mat=NULL, tot_ints, *type_blk=NULL, *blk_info=NULL;
char **var_label=NULL;
float **res_mat=NULL;
int block_no=0;

/* basic variables */
int *var_col=NULL, mat_rows, mat_cols, opt_exptnum, *sel_col=NULL;
int *opt_exptmat=NULL;
float *coeff=NULL, *opt_res=NULL, noise_var, *net_coeff;

/* interaction variables */
int int_count, *sel_col_int=NULL, *list_int=NULL, *no_int_col=NULL;
int *cur_int_col=NULL, *intcol_ord=NULL, num_calls=0, *b_int_data=NULL;
int *chk_zero_int=NULL, *chk_col_int=NULL, *known_int=NULL, num_knownint;
int num_best_sel;
int act_int_count, *act_list_int=NULL, act_list_len=5, *conint=NULL;
float act_variance, est_variance, *act_var_coeff=NULL, *act_int_coeff=NULL;
float act_avg_coeff;

void f_test(float, int, float, int, float *, float *);
void actual_plant_result(int, int, int *, float *);

main(){
void load_oaat_data(char *);
void free_up(void);
void make_list_int(void);
void select_imp_interactions(void);
void update_database(void);
void get_act_plant(char *);

load_oaat_data(data_file);
make_list_int();
get_act_plant(plant_file);
select_imp_interactions();
update_database();
free_up();
return;
}

/*************************************
*
* Searches the different combinations
* of interactions.
*
*************************************/

static void select_imp_interactions(void){

int i, *opt_assign=NULL, done=0, data_num=0;
int *temp_int_data=NULL, opt_good=0;
char file_name[100];
void get_optimum(int *, int *, int);
void sort_interaction(int *);
void print_curr_summary(FILE *);
FILE *out_handle;

pmalloc((void **) &opt_assign, num_vars*sizeof(int));
pmalloc((void **) &temp_int_data, mat_cols*sizeof(int));
pmalloc((void **) &b_int_data, num_best*mat_cols*sizeof(int));

printf("Do you want to save the output?\n");
if(getch()=='y'){
        fflush(stdin);
        printf("Enter filename:  ");
        gets(file_name);
        out_handle=fopen(file_name, "w");
        }
    else out_handle=stdout;

    get_optimum(opt_assign, cur_int_col, 1);
    print_curr_summary(out_handle);
    if(out_handle!=stdout) print_curr_summary(stdout);

    while((opt_exptnum < lim_expts) && (done==0)){
        opt_good=0;
        data_num=0;
        while((opt_good==0)&&(opt_exptnum>0)){
            memcpy(temp_int_data, b_int_data+data_num*mat_cols,
                        mat_cols*sizeof(int));
            get_optimum(opt_assign, temp_int_data,0);
            opt_good=1;
            for(i=0; i<opt_exptnum; i++)
                if( memcmp(opt_assign, (opt_exptmat+num_vars*i),
                            num_vars*sizeof(int))==0) opt_good=0;
            if(opt_good==0){
                data_num++;
                printf("Optimal Selected before....  selecting next...\n");
                }
            if(data_num==num_best_sel){
                memcpy(temp_int_data, b_int_data, mat_cols*sizeof(int));
                get_optimum(opt_assign, temp_int_data,0);
                printf("All expts.  selected previously.  Sort Ints?  (y/n)\n");
                fflush(stdin);
                if(getch()=='y') sort_interaction(opt_assign);
                opt_good=1;
                break;
                }
            }
        fflush(stdin); getch();
        memcpy((opt_exptmat+num_vars*opt_exptnum),
                                opt_assign, num_vars*sizeof(int));
        actual_plant_result(1, num_vars, opt_assign, (opt_res+opt_exptnum));
        opt_exptnum++;
        for(i=0; i<mat_cols*num_best; i++) b_int_data[i]=0;

        get_best_interaction();
        memcpy(cur_int_col, b_int_data, mat_cols*sizeof(int));
        num_calls++;
        print_curr_summary(out_handle);
        if(out_handle!=stdout) print_curr_summary(stdout);
        printf("Do you wish to continue?  (y/n):  ");
        if(getch()=='n') done =1;
        }
    if(out_handle!=stdout) fclose(out_handle);
    pfree((void **) &opt_assign);
    return;
    }

/***********************************
*
* Sorts the confounded Interactions
*
***********************************/

void sort_interaction(int *opt_assign){

int i, j, k, col, opt_valid, opt_good, int1, int2, intvar[8], sign1, sign2;
int count, *rem_int_col=NULL;
float diff[8], tempf;
float model_plant_result(int *);

pmalloc( (void **) &rem_int_col, mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) rem_int_col[i]=0;

for(i=0, sign2=0; i<mat_cols; i++){
    sign1=0;
    if(sel_col_int[i]==1) for(j=0; j<no_int_col[i]; j++)
            if(chk_col_int[i*max_int + j]==1) sign1++;

    rem_int_col[i] = sign1;
    if((rem_int_col[i]+chk_zero_int[i])<=1) sign2++;
    }
if(sign2==mat_cols){
    printf("All interactions are sorted out...\n");
    return;
    }
for(i=0, col=0, tempf=-10000; i<mat_cols; i++)
    if((rem_int_col[i]+chk_zero_int[i]>1)&&(net_coeff[i]>tempf)){
        col=i;
        tempf = net_coeff[i];
        }
if(chk_zero_int[col]==1){
    printf("No Int.  case of Col:%2d being sorted ...  \n", col);
    return;
    }
  printf(file_divider);
  printf("No.\t Interaction \t Value");
  printf(file_divider);
  for(i=0; i<no_int_col[col]; i++)
    if(chk_col_int[col*max_int + i] == 1){
      for(j=0, sign1=1; j<list_ptr(col, i, LOR); j++)
        sign1 *= opt_assign[list_ptr(col, i, LV1+j)];
      int1=i;
      break;
      }
  for(i=int1+1, opt_valid=0; i<no_int_col[col]; i++)
    if(chk_col_int[col*max_int + i] == 1){

      for(j=0, sign2=1; j<list_ptr(col, i, LOR); j++)
                        sign2 *= opt_assign[list_ptr(col, i, LV1+j)];
      int2 = i;
      if(sign1!=sign2){ opt_valid=1; break; }
      }
  if(opt_valid==0){
    for(i=0, count=0; i<list_ptr(col, int1, LOR); i++, count++)
      intvar[count] = list_ptr(col, int1, LV1+i);
    for(i=0; i<list_ptr(col, int1, LOR); i++, count++)
      intvar[count] = list_ptr(col, int2, LV1+i);

    sign1 = cur_int_col[col];
    cur_int_col[col] = -1;
    tempf = model_plant_result(opt_assign);
    for(i=0; i<count; i++){
      opt_assign[intvar[i]] *= -1;
      diff[i] = fabs(model_plant_result(opt_assign) - tempf);
      opt_assign[intvar[i]] *= -1;
      }
    cur_int_col[col] = sign1;
    printf("Old Optimal assignment:\t ");
    for(j=0; j<num_vars; j++) printf( "%c ", (opt_assign[j]>0) ? '+':'-' );
    printf("\n");

    for(k=0; k<count; k++){
      for(i=1, sign1=0; i<count; i++) if(diff[sign1] > diff[i]) sign1 = i;
      opt_assign[intvar[sign1]] *= -1;
      for(i=0, opt_good=1; i<opt_exptnum; i++)
        if(memcmp(opt_assign, (opt_exptmat+num_vars*i),
                        num_vars*sizeof(int))==0) opt_good=0;
      if(opt_good==1) break;
      diff[sign1]=0;
      opt_assign[intvar[sign1]] *= -1;
      }
    printf("New Optimal assignment:\t ");
    for(j=0; j<num_vars; j++) printf( "%c ", (opt_assign[j]>0) ? '+':'-' );
    printf("\n");
    }
  else printf("Optimum Good for Sorting Ints in Col:%d\n", col);
  pfree((void **) &rem_int_col);
  return;
  }

/************************************
*
* Loads the variables from the  database
*
*************************************/

static void load_oaat_data(char *filen){

int i,j,k;

printf(file_divider);
load_database(filen, &num_vars, &tot_ints, &num_blocks,
        &size_expt, &size_blk, &type_blk, &var_mat, &int_mat,
        &blk_mat, &sel_row, &expt_mat, &res_mat, &var_label, &blk_info);

disp_blk_info(num_blocks, size_expt, size_blk, blk_mat, type_blk, blk_info);

/* make space for oaat experiments  */
opt_exptnum=0;
pmalloc( (void **) &opt_res, lim_expts*sizeof(float));
pmalloc( (void **) &opt_exptmat, num_vars*lim_expts*sizeof(int));

printf("Enter Block No:");
scanf("%d", &block_no);
if((block_no<0)||(block_no>num_blocks-1)||(blk_mat[block_no]==NULL)){
    printf("Invaild Block Number .....  Exiting Program");
    exit(1);
    }
disp_col_info(block_no, size_blk, blk_mat);

/* set var_col and coeff */
```

```
mat_rows=size_blk[block_no];
mat_cols=size_blk[block_no];

pmalloc( (void **) &var_col, num_vars*sizeof(int));
pmalloc( (void **) &coeff, mat_cols*sizeof(float));
pmalloc( (void **) &net_coeff, mat_cols*sizeof(float));
for(i=0; i<num_vars; i++) var_col[i]=var_ptr(i, VB0+block_no);
for(i=0; i<mat_cols; i++) coeff[i] = blk_ptr(block_no, i, BCO)/C_FACTOR;
for(i=0; i<mat_cols; i++) net_coeff[i] = blk_ptr(block_no, i, BCO)/C_FACTOR;
return;
}

/***************************************
*
* Selects important effects using the
* Lenth's Algorithm (Technometrics, Nov '89)
*
***************************************/

static void select_imp_columns(void){

int i, j, temp=0, safety_margin=3;
float T_975=2.0, So, ME, PSE, *ord_eff=NULL, *abs_eff=NULL;
float noise_var, temp1, temp2;

pmalloc((void **)    &ord_eff,    mat_cols*sizeof(float));
pmalloc((void **)    &abs_eff,    mat_cols*sizeof(float));
pmalloc( (void **)   &sel_col,    mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++){
   sel_col[i]=1;
   ord_eff[i]=0.0;
   ord_eff[i]=abs_eff[i]= fabs(coeff[i]);
   }

/* ordering the values of abs_eff in accending order */
for(i=0; i<mat_cols; i++)
   for(j=i; j<mat_cols; j++)
      if(ord_eff[i]>ord_eff[j]){
         temp1=ord_eff[i];
         ord_eff[i]=ord_eff[j];
         ord_eff[j]=temp1;
         }

So = 1.5 * ord_eff[mat_cols/2-2];
for(i=0; i<mat_cols; i++) if(ord_eff[i] >= 2.5*So) { temp = i-1; break;}
PSE = 1.5 * ord_eff[temp/2];
ME = T_975 * PSE;
for(i=0; i<mat_cols; i++) if(abs_eff[i] <= safety_margin*ME) sel_col[i]=0;
for(i=0, temp1=0.0, j=0; i<mat_cols; i++)
   if((abs_eff[i]>0.0001)&&(abs_eff[i]<ME)){
      temp1 += abs_eff[i]*abs_eff[i];
      j++;
      }
if(j>0) est_variance = temp1/j;
else est_variance = 0.0;

pfree((void **) &abs_eff);
pfree((void **) &ord_eff);
return;
}

/***************************************
*
* Free arrays
*
***************************************/

static void free_up(void){

int i;
for(i=0; i<num_vars; i++)
   pfree((void **) &(var_label[i]));
pfree((void **) &var_col);
pfree((void **) &var_label);
pfree((void **) &expt_mat);
pfree((void **) &res_mat);
pfree((void **) &size_expt);
pfree((void **) &sel_row);
pfree((void **) &coeff);
pfree((void **) &net_coeff);
pfree((void **) &sel_col);
pfree((void **) &list_int);
pfree((void **) &no_int_col);
pfree((void **) &sel_col_int);
pfree((void **) &cur_int_col);
pfree((void **) &intcol_ord);
pfree((void **) &b_int_data);
pfree((void **) &chk_zero_int);
pfree((void **) &chk_col_int);
pfree((void **) &opt_exptmat);
pfree((void **) &opt_res);
pfree((void **) &known_int);
return;
}

/*************************************
*
* Forms list_int from interaction data
*
*************************************/

static void make_list_int(void){

int i, j, k, l, v1, v2, count, min_weight, col;
int *tempvect=NULL;
void select_imp_columns(void);

pmalloc((void **) &tempvect,     int_len*sizeof(int));
pmalloc((void **) &no_int_col,    mat_cols*sizeof(int));
pmalloc((void **) &list_int,    list_len*mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) no_int_col[i]=0;
for(i=0; i<list_len*mat_cols; i++) list_int[i]=0;

/* Remove Known effects of the avg. effect and variables from net_coeff[] */
col = var_ptr(num_vars, VB0+block_no);
net_coeff[col] -= var_ptr(num_vars, VCO)/C_FACTOR;
for(i=0; i<num_vars; i++){
   col = var_ptr(i, VB0+block_no);
   net_coeff[col] -= var_ptr(i, VCO)/C_FACTOR;
   }

/* Determine Known Interactions and remove them from list_int */
for(i=0, num_knownint=0; i<tot_ints; i++)
   if(int_ptr(i, IUC)==1) num_knownint++;
pmalloc((void **) &known_int, num_knownint*sizeof(int));

for(i=0, count=0; i<tot_ints; i++){
    k=abs(int_ptr(i, IB0+block_no));
    if(int_ptr(i, IUC)==1){
        known_int[count]=i;
        count++;
        v1 = (int_ptr(i, IB0+block_no)>0)? 1:-1;
        net_coeff[k] -= v1*int_ptr(i, ICO)/C_FACTOR;
        }
   else{
        list_ptr(k, no_int_col[k], LOR) = int_ptr(i, IOR);
        list_ptr(k, no_int_col[k], LWE) = int_ptr(i, IWE);
        list_ptr(k, no_int_col[k], LSG) =
          (int_ptr(i, IB0+block_no)>0) ? 1:-1;

        for(l=0; l<list_ptr(k, no_int_col[k], LOR); l++)
          list_ptr(k, no_int_col[k], LV1+l) = int_ptr(i, IV1+l);
        no_int_col[k]++;
        if(no_int_col[k]>max_int)
          error("Too many Interactions in Column", NULL, -1);
        }
   }

/*sorts interactions in decreasing order of weight */
for(i=0; i<mat_cols; i++)
    for(j=0; j<no_int_col[i]; j++)
        for(k=j+1; k<no_int_col[i]; k++)
            if(list_ptr(i, j, LWE) < list_ptr(i, k, LWE)){
                memcpy(tempvect, &(list_ptr(i, k, 0)), int_len*sizeof(int));
                memcpy(&(list_ptr(i, k, 0)), &(list_ptr(i, j, 0)),
                                    int_len*sizeof(int));
                memcpy(&(list_ptr(i, j, 0)), tempvect, int_len*sizeof(int));
                }
select_imp_columns();

pmalloc((void **) &sel_col_int, mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) sel_col_int[i]=0;
for(i=0; i<mat_cols; i++) if(no_int_col[i]>0) sel_col_int[i] = 1;
int_count=0;
for(i=0; i<mat_cols; i++) if(sel_col_int[i]==1) int_count++;

pmalloc((void **) &cur_int_col, mat_cols*sizeof(int));
for(i=0; i<mat_cols; i++) cur_int_col[i]=0;

/* sort col in decending order and assign to intcol_ord ***/
pmalloc((void **) &intcol_ord, int_count*sizeof(int));
count=0;
for(i=0; i<mat_cols; i++) if(sel_col_int[i]==1) intcol_ord[count++] = i;
for(i=0; i<int_count; i++)
    for(j=i+1; j<int_count; j++){
        v1=intcol_ord[i]; v2=intcol_ord[j];
        if(abs(net_coeff[v1])<abs(net_coeff[v2])){
            intcol_ord[i]=v2;
            intcol_ord[j]=v1;
            }
        }

pmalloc((void **) &chk_zero_int, mat_cols*sizeof(int));
pmalloc((void **) &chk_col_int, max_int*mat_cols*sizeof(int));

/* chk_zero_int determines if the zero_int case in col is checked or not.  */
for(i=0; i<mat_cols; i++) chk_zero_int[i]=0;
for(i=0; i<mat_cols; i++) if((sel_col_int[i]==1)&&(sel_col[i]==1))
    chk_zero_int[i]=0;
for(i=0; i<max_int*mat_cols; i++) chk_col_int[i]=0;
for(i=0; i<mat_cols; i++)
    for(j=0; j<no_int_col[i]; j++) chk_col_int[i*max_int +j]=1;
return;
}

/************************************
*
* Sets up variables for Optimize.C
*
*************************************/

static void get_optimum(int *solved_var, int *opt_int_col, int prt){

int i, j, *int_data=NULL, intno, sign;
int *int_sign=NULL, count, opt_int_count;
float quality, *var_coeff=NULL, *optint_coeff=NULL, avg_coeff;
char temp_str[100];

opt_int_count = int_count + num_knownint;
for(i=0; i<mat_cols; i++)
    if((sel_col_int[i]==1)&&(opt_int_col[i]<0)) opt_int_count--;

pmalloc((void **) &int_data,       opt_int_count*num_vars*sizeof(int));
pmalloc((void **) &int_sign,       opt_int_count*sizeof(int));
pmalloc((void **) &var_coeff,      num_vars*sizeof(float));
pmalloc((void **) &optint_coeff,   opt_int_count*sizeof(float));

for(i=0; i<num_vars; i++) solved_var[i]=0;
for(i=0; i<opt_int_count; i++) int_sign[i]=1;
for(i=0; i<num_vars*opt_int_count; i++) int_data[i]=0;
for(i=0; i<num_vars; i++) var_coeff[i]= -var_ptr(i, VCO)/C_FACTOR;
avg_coeff = -blk_ptr(block_no, 0, BCO)/C_FACTOR;

for(i=0, count=0; i<mat_cols; i++)
    if((sel_col_int[i]==1)&&(opt_int_col[i]>=0)){
        sign = list_ptr(i. opt_int_col[i]. LSG);
        optint_coeff[count]            = -net_coeff[i]*sign;
        int_data[num_vars*count]    = list_ptr(i, opt_int_col[i], LOR);

        for(j=0; j< list_ptr(i, opt_int_col[i], LOR); j++)
            int_data[num_vars*count+1+j] = list_ptr(i, opt_int_col[i], LV1+j);
        count++;
        }
for(i=0; i<num_knownint; i++){
        intno = known_int[i];
        int_data[num_vars*count] = int_ptr(intno, IOR);
        for(j=0; j<int_ptr(intno, IOR); j++)
            int_data[num_vars*count+1+j] = int_ptr(intno, IV1+j);
        optint_coeff[count] = -int_ptr(intno, ICO)/C_FACTOR;
        count++;
        }

quality = -oaat_opt(solved_var, num_vars, opt_int_count, int_data,
                int_sign, avg_coeff, var_coeff, optint_coeff);

/* output optimal assignment */
if(prt!=0){
    printf("\nOpt.  Assignments:\t " );
    for (i=0; i<num_vars; i++) printf("%s ", var_label[i] );
    printf("\n" );
    printf( "Opt Qty:  %5.3f \t ", quality );
    for (i=0; i<num_vars; i++){
       sprintf( temp_str, "%%%dc ", (int) strlen( var_label[i]) );
       printf( temp_str, (solved_var[i] > 0) ? '+' : '-' );
        }
    printf(file_divider);
    }

if(opt_int_count>0){
    pfree( (void **) &int_data    );
    pfree( (void **) &int_sign    );
    pfree( (void **) &var_coeff   );
    pfree( (void **) &optint_coeff );
    }
return;
}

/************************************
*
* Calculates current model process output
```

```
*
**************************************/

static float model_plant_result(int *var_value){

int i, j, temp, intno, sign;
float res=0.0;

res = var_ptr(num_vars, VCO)/C_FACTOR;
for(i=0; i<num_vars; i++) res += var_ptr(i, VCO)/C_FACTOR*var_value[i];

for(i=1; i<mat_cols; i++)
    if((sel_col_int[i]==1)&&(cur_int_col[i]>=0)){
        for(j=0, temp=1; j<list_ptr(i, cur_int_col[i], LOR); j++)
            temp *= var_value[list_ptr(i, cur_int_col[i], LV1+j)];
        sign = list_ptr(i, cur_int_col[i], LSG);
        res += sign*net_coeff[i]*temp;
        }
for(i=0; i<num_knownint; i++){
    intno = known_int[i];
    for(j=0, temp=1; j<int_ptr(intno, IOR); j++)
        temp *= var_value[int_ptr(intno, IV1+j)];
    res  += int_ptr(intno, ICO)/C_FACTOR*temp;
    }
return(res);
}

/**************************************
*
* Reads the True process parameters from
* file plant_file.
*
**************************************/

static void get_act_plant(char *f_name){

int i, j, varno, tmat_rows, tmat_cols, *tvar_col=NULL, *tmat=NULL;
char temp_str[200];
FILE *handle;

handle = fopen(f_name, "r");
if(handle==NULL) error( "Cant open output file", f_name, -1);
get_line( temp_str, handle);
if(temp_str[1]!='n') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);     /* read num_vars */
get_line( temp_str, handle);
if(temp_str[1]!='a') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);     /* read avg. coeff */
act_avg_coeff=atof(temp_str);
get_line( temp_str, handle);
if(temp_str[1]!='v') error("Invalid actual model file:  ", f_name, fileline);
pmalloc((void **) &act_var_coeff, num_vars*sizeof(float));
for(i=0; i<num_vars; i++) act_var_coeff[i]=0.0;

while (!get_line( temp_str, handle )){
    if (temp_str[0] == Field_Char){
        unget_line();
        break;
        }
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    varno = atoi(temp_str)-1;
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_var_coeff[varno] = atof(temp_str);
    }

get_line( temp_str, handle);
if(temp_str[1]!='i') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);
act_int_count=atoi(temp_str);

pmalloc((void **) &act_list_int, act_list_len*act_int_count*sizeof(int));
pmalloc((void **) &act_int_coeff, act_int_count*sizeof(float));

i=0;
while (!get_line( temp_str, handle )){
    if (temp_str[0] == Field_Char){
        unget_line();
        break;
        }
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_list_int[i*act_list_len] = atoi(temp_str);

    for(j=0; j<act_list_int[i*act_list_len]; j++){
        if(!get_label(temp_str)) error("error:", f_name, fileline);
        act_list_int[i*act_list_len + 1 +j] = atoi(temp_str)-1;
        }

    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_int_coeff[i] = atof(temp_str);
    i++;
    }
if(i!=(act_int_count)) error("error in interaction data", f_name, -1);

get_line( temp_str, handle);
if(temp_str[1]!='1') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle);
if(!get_label(temp_str)) error("error:", f_name, fileline);
act_variance=atof(temp_str);

set_mat(num_vars, block_no, expt_mat, sel_row, size_blk, &tmat_rows,
        &tmat_cols, &tvar_col, &tmat, &conint);

pfree((void **) &tvar_col);
pfree((void **) &tmat);
return;
}

/*************************************
*
* Prints summary of current optimal process
* model and the true process model and the
* OAAT experiment results.
*
**************************************/

static void print_curr_summary(FILE *out_handle){

int i, j, col, col1, pred_prt, act_prt;
float variance, tempf;
float model_plant_result(int *);

fprintf( out_handle, file_divider);
fprintf( out_handle, "COLUMNWISE  SUMMARY:");
fprintf( out_handle, file_divider);
fprintf( out_handle,
   "Col.  Est.  Cf.  \tPred(V/I)       True Cf.\tTrue(V/I)");
fprintf( out_handle, file_divider);
fprintf( out_handle, "%2d  %6.2f\t", 1, coeff[0]);
fprintf( out_handle, "Avg.   \t\t%6.2f\tAvg.\n", act_avg_coeff);

for(col=1; col<mat_cols; col++){
   pred_prt=0;
   act_prt=0;
   fprintf( out_handle, "%2d  %6.2f\t", col+1, coeff[col]);
   for(i=0; (pred_prt==0)&&(i<num_vars); i++) if(var_col[i]==col){
      fprintf( out_handle, "%s\t\t", var_label[i]);
      pred_prt=1;
      }
   if((pred_prt==0)&&(sel_col_int[col]==1)&&(cur_int_col[col]>=0)){
      fprintf( out_handle, "I%d", list_ptr(col, cur_int_col[col], LV1)+1);
      for(i=1; i<list_ptr(col, cur_int_col[col], LOR); i++)
         fprintf(out_handle, "%d", list_ptr(col,cur_int_col[col],LV1+i)+1);
      fprintf( out_handle, "\t\t");
      pred_prt=1;
      }
   if(pred_prt==0) fprintf( out_handle, "-----\t\t");

   for(i=0; (act_prt==0)&&(i<num_vars); i++)
      if( (var_col[i]==col) && (fabs(act_var_coeff[i])>0.00001)){
         fprintf( out_handle, "%6.2f\t%s", act_var_coeff[i], var_label[i]);
         act_prt=1;
         }
   if(act_prt==0){
      for(i=0; i<act_int_count; i++){
         for(j=0, col1=0; j<act_list_int[i*act_list_len]; j++)
            col1 = conint[mat_cols*col1 +
                var_col[act_list_int[i*act_list_len+j+1]]];
         if(col==col1){
            fprintf( out_handle, "%6.2f\t", act_int_coeff[i]);
            fprintf( out_handle, "I%d", act_list_int[i*act_list_len+1]+1);
            for(j=1; j<act_list_int[i*act_list_len]; j++)
               fprintf( out_handle, "%d",
                   act_list_int[i*act_list_len+1+j]+1);
            fprintf( out_handle, "\t");
            act_prt=1;
            }
         }
      }
   if(act_prt==0) fprintf( out_handle, "\t-----\t");
   fprintf( out_handle, "\n");
   }
fprintf( out_handle, file_divider);
fflush(stdin);
if(out_handle==stdout) getch();

fprintf( out_handle, "EXPERIMENT-WISE  SUMMARY:");
fprintf( out_handle, file_divider );
if(opt_exptnum>0){
   fprintf( out_handle, "Expt.\tExperiment\tPrediction\tObservation");
```

```
    fprintf( out_handle, file_divider);
    for(i=0, variance=0; i<opt_exptnum; i++){
        fprintf( out_handle, "%2d.\t", i+1);
        tempf = model_plant_result(opt_exptmat+num_vars*i);
        for(j=0; j<num_vars; j++)
            fprintf(out_handle,"%c ", (*(opt_exptmat+i*num_vars+j)>0)?'+':'-');
        fprintf( out_handle, "\t%6.2f\t\t %6.2f  \t", tempf, opt_res[i] );
        variance += (opt_res[i]- tempf)*(opt_res[i]-tempf);
        fprintf( out_handle, "\n");
        }
    fprintf( out_handle, "\nRMS Pred.  Error:\t%6.2f \n",
          (float)sqrt((double) variance/opt_exptnum));
    }
else fprintf( out_handle, "Initial Guess of Process Model.  \n");

fprintf( out_handle, file_divider);
fprintf( out_handle, "\t\t(%c) Results after Experiment %d\n",
      ('a'+opt_exptnum), opt_exptnum);
return;
}

/**************************************
*
* Updates the database
*
**************************************/

static void update_database(void){

int i, j, k, l, sign, found, num_selint, selint_no;

size_expt[num_blocks]=opt_exptnum;
size_blk[num_blocks]=0;
type_blk[num_blocks]=TYPE_OAAT;
blk_info[num_blocks]=block_no;

pmalloc((void **) &(res_mat[num_blocks]),
    sof_resmat(num_blocks)*sizeof(float));
pmalloc((void **) &(expt_mat[num_blocks]),
    sof_exptmat(num_blocks)*sizeof(int));
sel_row[num_blocks] = NULL;
blk_mat[num_blocks] = NULL;

/* Update INT_MAT */
for(i=0; i<mat_cols; i++) if(sel_col_int[i]==1){
```

```
    /* Delete non-significant interactions */
    for(j=0, num_selint=0; j<no_int_col[i]; j++){
        if(chk_col_int[i*max_int+j]==0){
            for(k=0; k<tot_ints; k++){
                for(l=0, found=1; l<list_ptr(i, j, LOR); l++)
                    if(int_ptr(k, IV1+j)!=list_ptr(i, j, LV1+j)) found=0;
                if(found==1){
                    int_ptr(k, IUC) = 1;
                    int_ptr(k, ICO) = 0;
                    break;
                    }
                }
            if(found==0) error("Error in database",NULL,-1);
            }
        else{ num_selint++; selint_no = j; }
        }

    /* Store Selected Interactions */
    if((chk_zero_int[i]==0)&&(num_selint==1)){
        for(k=0; k<tot_ints; k++){
            for(l=0, found=1; l<list_ptr(i, selint_no, LOR); l++)
                if(int_ptr(k, IV1+j)!=list_ptr(i, selint_no, LV1+j)) found=0;
            if(found==1){
                sign = (int_ptr(k, IB0+block_no)>0)? 1:-1;
                int_ptr(k, IUC) = 1;
                int_ptr(k, ICO) = (int)(sign*net_coeff[i]*C_FACTOR);
                break;
                }
            }
        if(found==0) error("Error in database", NULL, -1);
        }
    }
num_blocks++;
printf("\nDo you want to save updated database?  (y/n)");
if(getch()=='y'){
    save_database(data_file, num_vars, tot_ints, num_blocks,
            size_expt, size_blk, type_blk, var_mat, int_mat,
            blk_mat, sel_row, expt_mat, res_mat, var_label, blk_info);
    }
return;
}
```

E.3.2 OAAT_itr.C

```
/************************************
*
*This routine is called by OAAT.C. It
*determines the prediction errors for
*all the possible combinations of
*interactions. Then it selects the models
*with the least prediciton errors and
*passes them to the main program.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"

int *opt_int_col=NULL,  num_opt_int, pass;
float  *opt_error=NULL, *err_per_int=NULL, err_no_int;

extern int num_vars,  block_no, num_calls, mat_cols, num_best_sel;
extern int int_count, *intcol_ord, *list_int, *b_int_data, *no_int_col;
extern int *chk_zero_int, *chk_col_int, *opt_exptmat, opt_exptnum;
extern int tot_ints, *known_int, num_knownint, *int_mat, *var_mat;
extern float *opt_res, *net_coeff;
extern char **var_label;

/************************************/

void get_best_interaction(void){

int i, j, k, count, temp, col_count, intno, *temp_int_col=NULL, col;
float  *err_vect=NULL, tempf;
float recur( int num_assign, int *temp_int_col, float *err_vect);

pass=0;
num_opt_int = 3*num_best;

pmalloc((void **) &temp_int_col, int_count*sizeof(int));
pmalloc((void **) &opt_int_col, int_count*num_opt_int*sizeof(int));
pmalloc((void **) &err_vect, opt_exptnum*sizeof(float));
pmalloc((void **) &opt_error, num_opt_int*sizeof(float));
pmalloc((void **) &err_per_int, max_int*sizeof(float));

for(i=0; i<int_count*num_opt_int; i++) opt_int_col[i]=−1;
for(i=0; i<int_count; i++) temp_int_col[i]=−1;
for(i=0; i<opt_exptnum; i++) err_vect[i]=opt_res[i];
for(i=0; i<num_opt_int; i++) opt_error[i]=BIG_VALUE;
for(i=0; i<max_int; i++) err_per_int[i]=0.0;
err_no_int =0.0;

/* subtract avg, variable component  and known ints. from the error */
for(i=0; i<opt_exptnum; i++)
   err_vect[i] −= var_ptr(num_vars, VCO)/C_FACTOR;
for(i=0; i<num_vars; i++)
   for(j=0; j<opt_exptnum; j++) err_vect[j] −=
      opt_exptmat[j*num_vars+i]*var_ptr(i, VCO)/C_FACTOR;

for(i=0; i<num_knownint; i++){
   intno = known_int[i];
   for(j=0; j<opt_exptnum; j++){
      for(k=0, temp=1; k<int_ptr(intno, IOR); k++)
         temp *= opt_exptmat[j*num_vars + int_ptr(intno, IV1+k)];
      err_vect[j] −= int_ptr(intno, ICO)/C_FACTOR*temp;
      }
   }

/* rotates intcol_ord */
col_count=0;
count=0;
if(num_calls>2){
   while((col_count<int_count)&&(count<2)){

      count=0;
      tempf =intcol_ord[0];
      for(i=0; i<int_count−1; i++) intcol_ord[i] = intcol_ord[i+1];
      intcol_ord[int_count−1]=tempf;
      col = intcol_ord[0];

      if(chk_zero_int[col]==1) count++;
      for(i=0; i<no_int_col[col]; i++)
         if(chk_col_int[col*max_int +i]==1) count++;
      col_count++;
      }
   }

/* calls the recursion routine, recur() */
recur(0, temp_int_col, err_vect);

/* set b_int_data */
num_best_sel=0;
for(j=0; j<num_opt_int; j++){
    count=0;
    if(opt_error[j]<BIG_VALUE-1){
        printf("Num_best_sel:  %d ", num_best_sel);
        for(i=0; i<int_count; i++){
            col = intcol_ord[i];
            intno = opt_int_col[j*int_count + i];
            if((intno ==-1)&& (chk_zero_int[col]==0)) count=1;
            if((intno>-1) && (chk_col_int[col*max_int + intno ]==0)) count=1;
            b_int_data[num_best_sel*mat_cols + col] = intno;
            printf("%2d ", intno);
            }
        if(count==0) num_best_sel++;
        printf(" OPT_ERROR: %6.3f \n", opt_error[j]);
        }
    if(num_best_sel >= num_best) break;
    }
fflush(stdin); getch();

pfree((void **) &temp_int_col);
pfree((void **) &opt_int_col);
pfree((void **) &err_vect);
pfree((void **) &opt_error);
pfree((void **) &err_per_int);
return;
}

/*************************************
*
* Main Recursion routine
*
*************************************/

float recur(int num_assign, int *inter_col, float *error_v){

int i, j, int_no;
int *temp_inter_col=NULL, temp_num_assign;
float err, *temp_error_v=NULL, iter_err = BIG_VALUE, temp_iter_err;

float update_min_error(int *, float *);
void update_error(int, int , float *);
void delete_interaction(void);

pmalloc((void **) &temp_inter_col, int_count*sizeof(int));
pmalloc((void **) &temp_error_v, opt_exptnum*sizeof(float));
memcpy(temp_inter_col, inter_col, int_count*sizeof(int));
memcpy(temp_error_v, error_v, opt_exptnum*sizeof(float));
temp_num_assign = num_assign;

if(num_assign>=int_count){
    err = update_min_error(inter_col, error_v);
    pass++;
    return(err);
    }

for(int_no=-1; int_no<no_int_col[intcol_ord[num_assign]]; int_no++){
    if((int_no==-1)&&(chk_zero_int[intcol_ord[num_assign]]==0)) continue;
    if((int_no >-1) &&
        (chk_col_int[intcol_ord[num_assign]*max_int + int_no] ==0)) continue;

    if(num_assign == 0) iter_err = BIG_VALUE;
    if(int_no >= 0) update_error(num_assign, int_no, error_v);
    inter_col[num_assign] = int_no;
    num_assign++;

    temp_iter_err = recur(num_assign, inter_col, error_v);
    if(temp_iter_err < iter_err) iter_err = temp_iter_err;
    memcpy(inter_col, temp_inter_col, int_count*sizeof(int));
    memcpy(error_v, temp_error_v, opt_exptnum*sizeof(float));
    num_assign = temp_num_assign;
    if((num_calls>1)&&(num_assign==0)){
        if(int_no == -1) err_no_int = iter_err;
        else err_per_int[int_no] = iter_err;
        }
    }

if((num_calls>1)&&(num_assign==0)) delete_interaction();

pfree((void **) &temp_inter_col);
pfree((void **) &temp_error_v);
return(iter_err);
}

/*************************************
*
* Prints the least prediction
* error for each of the interactions in
* the column. The interactions which have
* very large predicition errors can be
* deleted. These interactions will not be
* searched in the subsequent experiments.
*
*************************************/

static void delete_interaction( void ){

int i, j, col, v1, inp;
char chinp[10];

col = intcol_ord[0];
printf(file_divider);
printf("SUMMARY OF ERRORS WITH INTERACTIONS:\n");
printf("No.\t Interaction \t Error");
printf(file_divider);
printf(" 1 \tNo Int\t\t");
if(chk_zero_int[col]==1) printf("%7.2f\n", err_no_int);
else printf("----\n");
for(i=0; i<no_int_col[col]; i++){
   printf("%2d\t%s", i+2, var_label[list_ptr(col, i, LV1)]);
   for(j=1; j<list_ptr(col, i, LOR); j++)
      printf("-%s", var_label[list_ptr(col, i, LV1+j)]);
   printf("\t\t");

   if(chk_col_int[col*max_int + i] == 1) printf("%7.2f \n", err_per_int[i]);
   else printf("----\n");
   }
inp = -1;
while((inp>=-1)&&(inp<no_int_col[col])){
   printf("Enter No.  for deletion:");
   fflush(stdin);
   scanf("%[^\n]", &chinp);
   inp = atoi(chinp) -2;
   if(inp==-1){
      if(chk_zero_int[col]==0) printf("INVALID: Int.  deleted before!\n");
      else{
         printf("Searching No interaction STOPPED......  \n");
         chk_zero_int[col]=0;
         }
      }
   if((inp>=0)&&(inp<no_int_col[col])){
      printf("Int.  %s", var_label[list_ptr(col, inp, LV1)]);
      for(j=1; j<list_ptr(col, inp, LOR); j++)
         printf("-%s", var_label[list_ptr(col, inp, LV1+j)]);
         if(chk_col_int[col*max_int | inp]==0)
            printf("- Deleted before from Col.  %2d:  INVALID \n", col);
         else{
            printf("- Deleting from Column:%2d \n", col);
            chk_col_int[ col*max_int + inp ]=0;
            }
         }
      }
   return;
   }

/*************************************
*
* Checks if combination of interactions
* is better than current optimum.
*
*************************************/

float update_min_error(int *inter_col, float *error_v){

int count, i;
float temperr=0.0;

for(i=0; i<opt_exptnum; i++) temperr += error_v[i]*error_v[i];
count=num_opt_int;
if(temperr < opt_error[count-1]) {
   while((temperr < opt_error[count-1])&&(count>0)) count--;

   for(i=num_opt_int-1; i>count; i--){
      memcpy(opt_int_col+int_count*i, opt_int_col+ int_count*(i-1),
                 int_count*sizeof(int));
      opt_error[i] = opt_error[i-1];
      }
   memcpy(opt_int_col+int_count*count, inter_col, int_count*sizeof(int));
   opt_error[count]=temperr;
   }
return(temperr);
}

/*************************************
*
* Updates the prediciton error by subtracting
* the effect of the selected interaction
*
*************************************/
```

```
void update_error(int num_assign, int int_no, float *error_v){

int i, j, v1, sign, temp;
float i_coeff;

sign = list_ptr(intcol_ord[num_assign], int_no, LSG);
i_coeff = net_coeff[intcol_ord[num_assign]];
for(i=0; i<opt_exptnum; i++){
    for(j=0, temp=1; j<list_ptr(intcol_ord[num_assign], int_no, LOR); j++){
        v1      = list_ptr(intcol_ord[num_assign], int_no, LV1+j);
        temp *=opt_exptmat[i*num_vars + v1];
        }
    error_v[i] -= sign*i_coeff*temp;
    }
return;
}
```

E.3.3 OAAT_opt.C

```
/************************************
*
*This routine is a modification of
*optimize() written by P. Fieguth, M. Spina
*and D. DeCaprio. It is called by OAAT.C.
*Given the variables and the interactions,
*the function determines the optimal expt.
*i.e. the experiment which is expected to
*yeild the highest output quality.
*
************************************/

#include "coninc1.h"

float oaat_opt(int *solved_var, int num_vars, int num_ints, int *int_data,
int *int_sign, float avg_coeff, float *var_coeff, float *int_coeff){

/* optimality calculation variables */
int   *grouped_var=NULL, *valid_int=NULL, *tested_int=NULL;
int   int_used, bitlim, count, mask, i, j, k, l, done;
int   *temp_vect=NULL, *vect_transl=NULL;
float quality, min_quality, sum_quality, temp_float=0.0;

/* initialize - include all interations, no variables yet solved */
pmalloc( (void **) &grouped_var, num_vars * sizeof( int ) );
pmalloc( (void **) &vect_transl, num_vars * sizeof( int ) );
if( num_ints > 0 ){
    pmalloc( (void **) &valid_int, num_ints * sizeof( int ) );
    pmalloc( (void **) &tested_int, num_ints * sizeof( int ) );
    }
pmalloc( (void **) &temp_vect, num_vars * sizeof( int ) );
for (i=0; i<num_vars; i++) solved_var[i] = 0;

/* loop over variables to solve */
sum_quality = avg_coeff;
for (i=0; i<num_vars; i++) if (solved_var[i] == 0){
    /* mark all non-overlapping or dominant interactions as valid */
    for (j=0; j<num_ints; j++)  valid_int[j] = 1;

    done = 0;
    while (!done){
        /* reset list of variables in this group & tested interactions */
        /* only test non-overlapping or dominant interactions (from earlier) */
        for (j=0; j<num_vars; j++) grouped_var[j] = 0;
        for (j=0; j<num_ints; j++) tested_int[j] =
                (valid_int[j] > 0) ? 0:1;
        grouped_var[i] = 1;
        /* loop and build group of included variables */
        do{
            count = 0;
            for (j=0; j<num_ints; j++) if (tested_int[j] == 0){
                /* any variables of interest in this interaction? */
                k = 0;
                for (l=0; l<int_data[j*num_vars]; l++)
                    k += grouped_var[int_data[j*num_vars+l+1]];
                if (k > 0){
                    /* yes, include other variables in group */
                    for (l=0; l<int_data[j*num_vars]; l++)
                        grouped_var[int_data[j*num_vars+l+1]] = 1;
                    tested_int[j] = -1;
                    count++;
                    }
                }
            } while (count > 0);
        /* check number of variables in this group */
        count = 0;
        for (j=0; j<num_vars; j++){
            vect_transl[count] = j;
            count += grouped_var[j];
            }
        /* compare number of variables against threshold */
        if (count < Group_Threshold){
            /* ok, search for optimum */
            min_quality = MAXFLOAT/2;
            bitlim = 1 << count;
            for (j=0; j<bitlim; j++){
                /* try these settings - based on bits of j */
                quality = 0;
                mask = 1;
                for (k=0; k<count; k++){
                    temp_vect[vect_transl[k]] = ((j & mask) > 0) ? 1:-1;
                    mask <<= 1;
                    }
                /* add variable impacts to quality */
                for (k=0; k<num_vars; k++) if (grouped_var[k])
                    quality += var_coeff[k]*temp_vect[k];
                /* add interaction impacts to quality */
                for (k=0; k<num_ints; k++) if (tested_int[k] < 0){
                    temp_float = int_coeff[k]*int_sign[k];
                    for (l=0; l<int_data[k*num_vars]; l++)
                        temp_float *= temp_vect[int_data[k*num_vars+l+1]];
                    quality += temp_float;
                    }
                /* compare against current optimal */
                if (quality < min_quality){
                    min_quality = quality;
                    for (k=0; k<num_vars; k++) if (grouped_var[k])
                        solved_var[k] = temp_vect[k];
                    }
                }
            sum_quality += min_quality;
            done = 1;
            }
        else{
            /* try again with fewer ints - eliminate least important one */
            k = 0;
            temp_float = (MAXFLOAT / 2);
            for (j=0; j<num_ints; j++) if (valid_int[j])
                if (fabs(int_coeff[j]) < temp_float){
                    temp_float = fabs(int_coeff[j]);
                    k = j;
                    }
            /* remove smallest interaction */
            valid_int[k] = 0;
            }
        }
    }
pfree( (void **) &temp_vect );
pfree( (void **) &grouped_var );
if( num_ints > 0 ){
    pfree( (void **) &valid_int );
    pfree( (void **) &tested_int );
    }
pfree( (void **) &vect_transl );
return sum_quality;
}
```

E.4.1 Plant.C

```
/*************************************
*
*This routine is the simulated plant or
*process. The true plant model is stored
*in the file "plant_file". The routine is
*called by the other programs.
*It returns the value of the output quality
*of the plant at the desired operating point.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"
extern int fileline;

static inter_len=5;
static float *act_var_coeff=NULL, *act_int_coeff=NULL, lambda, act_avg_coeff;
static int act_int_count, *act_list_int=NULL;
static int read_plant_data=0;

void actual_plant_result(int num_pts, int num_vars,
                         int *var_data, float *plt_opt){

int i, j, k, *var_value=NULL;
float res, temp;
float get_rv(void);
void get_act_plant(char *, int);

pmalloc((void **) &var_value, num_vars*sizeof(int));
if(read_plant_data == 0){
   get_act_plant(plant_file, num_vars);
   read_plant_data=1;
   }
for(j=0; j<num_pts; j++){
   memcpy(var_value, var_data+j*num_vars, num_vars*sizeof(int));
   res = act_avg_coeff;
   for(i=0; i<num_vars; i++) res += act_var_coeff[i]*var_value[i];
   for(i=0; i<act_int_count; i++){
      for(k=0, temp=1; k<act_list_int[i*inter_len]; k++)
         temp *= var_value[act_list_int[i*inter_len +1+k]];
      res += act_int_coeff[i]*temp;
      }
   temp = get_rv()/4;
   res += lambda*temp;
   *(plt_opt+j) = res;
   }
pfree((void **) &var_value);
return;
}

/*************************************/
float get_rv(void){

float rv;
short i,j;

for(j=0, rv=0.0; j<47; j++)
   rv + = (rand()%1000 - 500)/1000;
return(rv);
}

/*************************************
*
* Loads the plant model from the user
* specified file.
*
*************************************/

static void get_act_plant(char *f_name, int num_vars){

int i, j, varno, tnum_vars;
char temp_str[200];
FILE *handle;

handle = fopen(f_name, "r");
if(handle==NULL) error( "Cant open output file", f_name, -1);
get_line( temp_str, handle);
if(temp_str[1]!='n') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);    /* read num_vars */
tnum_vars = atoi(temp_str);
if(tnum_vars!=num_vars) error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle);
if(temp_str[1]!='a') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);    /* read avg. coeff */
act_avg_coeff=atof(temp_str);
get_line( temp_str, handle);
if(temp_str[1]!='v') error("Invalid actual model file:  ", f_name, fileline);
pmalloc((void **) &act_var_coeff, tnum_vars*sizeof(float));
for(i=0; i<tnum_vars; i++) act_var_coeff[i]=0.0;

while (!get_line( temp_str, handle )){
    if (temp_str[0] == Field_Char){
        unget_line();
        break;
        }
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    varno = atoi(temp_str)-1;
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_var_coeff[varno] = atof(temp_str);
    }
get_line( temp_str, handle);
if(temp_str[1]!='i') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle); get_label(temp_str);
act_int_count=atoi(temp_str);

pmalloc((void **) &act_list_int, inter_len*act_int_count*sizeof(int));
pmalloc((void **) &act_int_coeff, act_int_count*sizeof(float));
i=0;
while (!get_line( temp_str, handle )){
    if (temp_str[0] == Field_Char){
        unget_line();
        break;
        }
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_list_int[i*inter_len] = atoi(temp_str);
    for(j=0; j<act_list_int[i*inter_len]; j++){
        if(!get_label(temp_str)) error("error:", f_name, fileline);
        act_list_int[i*inter_len + 1 +j] = atoi(temp_str)-1;
        }
    if(!get_label(temp_str)) error("error:", f_name, fileline);
    act_int_coeff[i] = atof(temp_str);
    i++;
    }
if(i!=(act_int_count)) error("error in interaction data", f_name, -1);
get_line( temp_str, handle);
if(temp_str[1]!='l') error("Invalid actual model file:  ", f_name, fileline);
get_line( temp_str, handle);
if(!get_label(temp_str)) error("error:", f_name, fileline);
lambda=atof(temp_str);
return;
}
```

E.4.2 Sort_int.C

```
/*************************************
*
*This routine is called by Blk_anl.C.
*It checks if there are any interactions
*which can be determined in the previous
*blocks of experiments. Such a situation
*arises when the effects of some of the
*confounding ints. become known in subsequent
*blocks of experiments.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"

void sort_interactions(int block_no, int *update_int, int num_vars,
    int tot_ints, int num_blocks, int *size_blk, int *type_blk,
    int *var_mat, int *int_mat, int **blk_mat){

int  j, k, l, var, blk, col, inter, unknown_int, int1;
int  int_name, intno, newint;
static int count=1;
float col_coeff;

printf("Update_int(%2d):  ", count++);
for(j=0; j<tot_ints; j++) printf("%d ", update_int[j]);
printf("\nHit any key to Continue.....\n");
fflush(stdin); getch();

for(inter=0; inter<tot_ints; inter++) if(update_int[inter]==1){
    intno=inter;
    update_int[inter]=0;
    if(int_ptr(intno, IUC)==0)
        error("Incorrect Interaction Specified", NULL, -1);

    for(blk=num_blocks-1; blk>=0; blk--) if(type_blk[blk]!=TYPE_OAAT){
        col = abs(int_ptr(intno, IB0+blk));
        col_coeff = blk_ptr(blk, col, BCO)/C_FACTOR;
        var = blk_ptr(blk, col, BVA);
        if((var>=0)&&(var<num_vars))
            col_coeff -= var_ptr(var, VCO)/C_FACTOR;
        unknown_int = blk_ptr(blk, col, BNI);
        for(j=0, newint=-1; j<blk_ptr(blk, col, BNI); j++){
```

```
        int_name = abs(blk_ptr(blk, col, BI1+j));

        for(k=0, int1=-1; k<tot_ints; k++)
           if(int_ptr(k, INO)==int_name) int1 = k;
        if(int1<0) error("Interaction not found!", NULL, -1);
        if(int_ptr(int1, IUC) == 1){
           if(blk_ptr(blk, col, BI1+j) > 0)
              col_coeff -= int_ptr(int1, ICO)/C_FACTOR;
           else col_coeff += int_ptr(int1, ICO)/C_FACTOR;
           unknown_int--;
           }
        else newint = int1;
        }

      if(unknown_int == 1){
        printf("Found Unconfounded Interaction:  %d\n", newint);
        int_ptr(newint, IUC)=1;
        int_ptr(newint, ICO)= (int)(col_coeff*C_FACTOR);
        if(int_ptr(newint, IB0+blk)<0) int_ptr(newint, ICO) *= -1;
        update_int[newint] = 1;
        }
      }
   }
for(inter=0, k=0; inter<tot_ints; inter++) if(update_int[inter]==1) k++;

if(k>0) sort_interactions(block_no, update_int, num_vars, tot_ints,
          num_blocks, size_blk, type_blk, var_mat, int_mat, blk_mat);
return;
}
```

E.4.3 Disp_info.C

```
/************************************
*
*This program contains routines which
*display information about the database.
*
*************************************/

#include "coninc1.h"
#include "coninc2.h"

/************************************/

void disp_var_info(int num_vars, int num_blocks, int *var_mat,
              int *type_blk, char **var_label){
int i, j;

printf("\nVARIABLE INFORMATION:\n");
printf("----------------------\n\n");
printf("Var.  \t  Coeff");
for(i=0; i<num_blocks; i++) printf("\tBlk %d", i);
printf("\n\n");
for(i=0; i<num_vars+1; i++){
   printf("%s \t%7.2f", var_label[i], var_ptr(i, VCO)/C_FACTOR);
   for(j=0; j<num_blocks; j++)
      if(type_blk[j]!=TYPE_OAAT) printf("\t%2d", var_ptr(i, VB0+j));
      else printf("\t---");
   printf("\n");
   }
printf("\nHit key to continue .....  \n\n");
fflush(stdin); getch();
return;
}

/************************************/

void disp_int_info(int tot_ints, int num_blocks,
              int *type_blk, int *int_mat, char **var_label){
int i, j;

printf("\nINTERACTION INFORMATION:\n");
printf("-------------------------\n");
printf("Int.  \tC/UC \t  Coeff\tWeight");
for(i=0; i<num_blocks; i++) printf("  Bk%d", i);
printf("\tVars.\n\n");

for(i=0; i<tot_ints; i++){
   printf("%2d.  \t(%d) \t%6.1f\t%6d", i,
      int_ptr(i, IUC), int_ptr(i, ICO)/C_FACTOR, int_ptr(i, IWE));
   for(j=0; j<num_blocks; j++)
      if(type_blk[j]!=TYPE_OAAT) printf("  %2d ", int_ptr(i, IB0+j));
      else printf("  --- ");
   printf("\t%s", var_label[int_ptr(i, IV1)]);
   for(j=1; j<int_ptr(i, IOR); j++)
      printf("-%s", var_label[int_ptr(i, IV1+j)]);
   printf("\n");
   }
printf("\nHit key to continue .....  \n\n");
fflush(stdin); getch();
return;
}

/*************************************/

void disp_blk_info(int num_blocks, int *size_expt, int *size_blk,
          int **blk_mat, int *type_blk, int *blk_info){

int i, j;

printf("\nBLOCK INFORMATION:\n");
printf("------------------\n");
printf("Blk.    Type_Blk  Size_Expt  Size_Blk  Analyzed \tBlk.  Info\n\n");
for(i=0; i<num_blocks; i++){
   printf(" %d.       ", i);
   switch(type_blk[i]){
      case 0: printf("FIR_BLK    "); break;
      case 1: printf("OAAT       "); break;
      case 2: printf("HALF_BLK   "); break;
      case 3: printf("FULL_BLK   "); break;
      case 4: printf("COMP_BLK   "); break;
      default: error("Block type incorrect", NULL, -1);
      }
   printf("%3d         %3d        ", size_expt[i], size_blk[i]);
   printf("   %s \t   %d\n", (blk_mat[i]==NULL)? "No":"Yes", blk_info[i]);
   }
printf("\nHit key to continue .....  \n\n");
fflush(stdin); getch();
return;
}
```

```
/*************************************/

void disp_col_info(int block_no, int *size_blk, int **blk_mat){

int i, j;

printf("\nCOLUMN INFORMATION:\n");
printf("-------------------\n");
printf("Col.\tCoeff\tVar.  #Int.  Int1 Int2 Int3 Int4\n\n");
for(i=0; i<size_blk[block_no]; i++){
   printf("%2d\t%6.2f\t", i, blk_ptr(block_no, i, BCO)/C_FACTOR);
   if(blk_ptr(block_no, i, BVA)==-1) printf(" --    ");
   else printf(" %2d    ", blk_ptr(block_no, i, BVA));
   printf("%2d    ", blk_ptr(block_no, i, BNI));
   for(j=0; j<blk_ptr(block_no, i, BNI); j++)
      printf("%2d    ", blk_ptr(block_no, i, BI1+j));
   printf("\n");
   }
printf("\nHit any key to continue ......  \n\n");
fflush(stdin); getch();
return;
}
```

E.4.4 Database.C

```
/*************************************
*
*This routine  saves the
*database i.e.the information about the
*1) blocks 2) variables 3) interactions
*and other information that the
*subsequent programs need.
*
**************************************/

#include "coninc1.h"
#include "coninc2.h"
extern int fileline;

void save_database(char *filen, int num_vars, int tot_ints, int num_blocks,
    int *size_expt, int *size_blk, int *type_blk, int *var_mat, int *int_mat,
    int **blk_mat, int **sel_row, int **expt_mat, float **res_mat,
    char **var_label, int *blk_info){

int i, *wr_blkmat=NULL, *wr_selrow=NULL, *wr_exptmat=NULL;
int *wr_resmat=NULL;
FILE *out_file;

out_file = fopen(filen, "wb");
if(out_file==NULL) error("Cant open INPUT file", filen, -1);

pmalloc((void **)      &wr_selrow,      num_blocks*sizeof(int));
pmalloc((void **)      &wr_exptmat,     num_blocks*sizeof(int));
pmalloc((void **)      &wr_resmat,      num_blocks*sizeof(int));
pmalloc((void **)      &wr_blkmat,      num_blocks*sizeof(int));

for(i=0; i<num_blocks; i++)
    wr_selrow[i] = wr_exptmat[i] = wr_blkmat[i] = wr_resmat[i] = 0;
for(i=0; i<num_blocks; i++){
    if(blk_mat[i]!=NULL)     wr_blkmat[i] =1;
    if(expt_mat[i]!=NULL)    wr_exptmat[i]=1;
    if(res_mat[i]!=NULL)     wr_resmat[i] =1;
    if(sel_row[i]!=NULL)     wr_selrow[i] =1;
    }
fwrite(&num_vars,      sizeof(int), 1, out_file);
fwrite(&tot_ints,      sizeof(int), 1, out_file);
fwrite(&num_blocks,    sizeof(int), 1, out_file);
fwrite(size_expt,      MAX_BLK*sizeof(int), 1, out_file);
fwrite(size_blk,       MAX_BLK*sizeof(int), 1, out_file);
fwrite(type_blk,       MAX_BLK*sizeof(int), 1, out_file);
fwrite(blk_info,       MAX_BLK*sizeof(int), 1, out_file);
fwrite(var_mat,        sof_varmat*sizeof(int), 1, out_file);
fwrite(int_mat,        sof_intmat*sizeof(int), 1, out_file);
fwrite(wr_blkmat,      num_blocks*sizeof(int), 1, out_file);
fwrite(wr_selrow,      num_blocks*sizeof(int), 1, out_file);
fwrite(wr_exptmat,     num_blocks*sizeof(int), 1, out_file);
fwrite(wr_resmat,      num_blocks*sizeof(int), 1, out_file);

for(i=0; i<num_blocks; i++){
    if(wr_blkmat[i]==1) fwrite(blk_mat[i], sof_blkmat(i)*sizeof(int), 1, out_file);
    if(wr_selrow[i]==1) fwrite(sel_row[i], sof_selrow(i)*sizeof(int), 1, out_file);
    if(wr_exptmat[i]==1) fwrite(expt_mat[i], sof_exptmat(i)*sizeof(int), 1, out_file);
    if(wr_resmat[i]==1) fwrite(res_mat[i], sof_resmat(i)*sizeof(float), 1, out_file);
    }
for(i=0; i<num_vars+1; i++)
    fwrite(var_label[i], Label_Length*sizeof(char), 1, out_file);

pfree((void **) &wr_selrow);
pfree((void **) &wr_exptmat);
pfree((void **) &wr_resmat);
pfree((void **) &wr_blkmat);

fclose(out_file);
return;
}

/*************************************
*
* This routine  loads the database i.e.
* the information about the 1) blocks
* 2) variables 3) interactions etc.
*
**************************************/

void load_database(char *filen, int *pnum_vars, int *ptot_ints,
    int *pnum_blocks, int **psize_expt, int **psize_blk, int **ptype_blk,
    int **pvar_mat, int **pint_mat, int ***pblk_mat, int ***psel_row,
    int ***pexpt_mat, float ***pres_mat, char ***pvar_label, int **pblk_info){

int i, *rd_blkmat=NULL, *rd_selrow=NULL, *rd_exptmat=NULL;
int tot_ints, num_vars, num_blocks, *size_expt, *size_blk, *rd_resmat=NULL;
FILE *out_file;
```

```
out_file = fopen(filen, "rb");
if(out_file==NULL) error("Cant open INPUT file", filen, -1);

fread(pnum_vars, sizeof(int), 1, out_file);
fread(ptot_ints, sizeof(int), 1, out_file);
fread(pnum_blocks, sizeof(int), 1, out_file);
num_vars = *pnum_vars;
tot_ints = *ptot_ints;
num_blocks = *pnum_blocks;

pmalloc((void **) psize_expt, MAX_BLK*sizeof(int));
pmalloc((void **) psize_blk,  MAX_BLK*sizeof(int));
pmalloc((void **) ptype_blk,  MAX_BLK*sizeof(int));
pmalloc((void **) pblk_info,  MAX_BLK*sizeof(int));

fread(*psize_expt, MAX_BLK*sizeof(int), 1, out_file);
fread(*psize_blk,  MAX_BLK*sizeof(int), 1, out_file);
fread(*ptype_blk,  MAX_BLK*sizeof(int), 1, out_file);
fread(*pblk_info,  MAX_BLK*sizeof(int), 1, out_file);

size_expt = *psize_expt;
size_blk  = *psize_blk;

pmalloc((void **) pvar_mat,  sof_varmat*sizeof(int));
pmalloc((void **) pint_mat,  sof_intmat*sizeof(int));

fread(*pvar_mat,  sof_varmat*sizeof(int), 1, out_file);
fread(*pint_mat,  sof_intmat*sizeof(int), 1, out_file);

pmalloc((void **)    &rd_selrow,     num_blocks*sizeof(int));
pmalloc((void **)    &rd_exptmat,    num_blocks*sizeof(int));
pmalloc((void **)    &rd_resmat,     num_blocks*sizeof(int));
pmalloc((void **)    &rd_blkmat,     num_blocks*sizeof(int));

fread(rd_blkmat,     num_blocks*sizeof(int), 1, out_file);
fread(rd_selrow,     num_blocks*sizeof(int), 1, out_file);
fread(rd_exptmat,    num_blocks*sizeof(int), 1, out_file);
fread(rd_resmat,     num_blocks*sizeof(int), 1, out_file);

pmalloc( (void **) pres_mat,    MAX_BLK*sizeof(float *));
pmalloc( (void **) pexpt_mat,   MAX_BLK*sizeof(int *));
pmalloc( (void **) pblk_mat,    MAX_BLK*sizeof(int *));
pmalloc( (void **) psel_row,    MAX_BLK*sizeof(int *));

for(i=0; i<MAX_BLK; i++){
    (*pres_mat)[i]=NULL;
    (*pexpt_mat)[i]=NULL;
    (*pblk_mat)[i]=NULL;
    (*psel_row)[i]=NULL;
    }

for(i=0; i<*pnum_blocks; i++){
    if(rd_blkmat[i]==1){
        pmalloc( (void **) &((*pblk_mat)[i]), sof_blkmat(i)*sizeof(int));
        fread((*pblk_mat)[i], sof_blkmat(i)*sizeof(int), 1, out_file);
        }
    if(rd_selrow[i]==1){
        pmalloc( (void **) &((*psel_row)[i]), sof_selrow(i)*sizeof(int));
        fread((*psel_row)[i], sof_selrow(i)*sizeof(int), 1, out_file);
        }
    if(rd_exptmat[i]==1){
        pmalloc( (void **) &((*pexpt_mat)[i]), sof_exptmat(i)*sizeof(int));
        fread((*pexpt_mat)[i],  sof_exptmat(i)*sizeof(int), 1, out_file);
        }
    if(rd_resmat[i]==1){
        pmalloc( (void **) &((*pres_mat)[i]), sof_resmat(i)*sizeof(float));
        fread((*pres_mat)[i], sof_resmat(i)*sizeof(float), 1, out_file);
        }
    }
pmalloc((void **) pvar_label, (num_vars+1)*sizeof(char *));
for(i=0; i<num_vars+1; i++) (*pvar_label)[i]=NULL;
for(i=0; i<num_vars+1; i++)
    pmalloc((void **) &((*pvar_label)[i]), Label_Length*sizeof(char));
for(i=0; i<num_vars+1; i++)
    fread((*pvar_label)[i], Label_Length*sizeof(char), 1, out_file);

pfree((void **) &rd_selrow);
pfree((void **) &rd_exptmat);
pfree((void **) &rd_resmat);
pfree((void **) &rd_blkmat);

fclose(out_file);
return;
}
```

E.4.5 Set_mat.C

```
/************************************
*
*Given a block of experiments, this routine
*forms the design matrix and confounding table.
*
************************************/

#include "coninc1.h"
#include "coninc2.h"

void  set_mat(int num_vars, int block_no, int **expt_mat,
      int **sel_row, int *size_blk, int *pmat_rows, int *pmat_cols,
      int **pvar_col, int **pmat, int **pconint){

int i, j, k, l, eq, sum, prod, varno, colno, *temp_vect=NULL;
int row, blk, power, mat_rows, mat_cols;

mat_rows=size_blk[block_no];
mat_cols=1;
while(mat_cols < mat_rows){ mat_cols *=2; power +=1;}
*pmat_rows = mat_rows;
*pmat_cols = mat_cols;

pmalloc( (void **) pmat, mat_rows*mat_cols*sizeof(int));
pmalloc( (void **) pconint, mat_cols*mat_cols*sizeof(int));
pmalloc( (void **) pvar_col, num_vars*sizeof(int));
pmalloc( (void **) &temp_vect, mat_rows*sizeof(int));
for(i=0; i<mat_rows; i++) for(j=0; j<mat_cols; j++) (*pmat)[i*mat_cols+j]=0;
for(i=0; i<mat_rows; i++) (*pmat)[i*mat_cols]=1;

/*** assign variables to mat ***/
for(i=0; i<mat_rows; i++){
   blk=sel_ptr(block_no, i, SBL);
   row=sel_ptr(block_no, i, SEP);
   memcpy( (*pmat + i*mat_cols+1), expt_ptr(blk, row), num_vars*sizeof(int));
   }
for(i=0; i<num_vars; i++) (*pvar_col)[i]=i+1;

/* complete the mat */
colno=num_vars+1;
i = 0;  j = 1;
while (colno < mat_cols){
   /* take product of matrix columns */
   for (k=0; k<mat_rows; k++)
      temp_vect[k] = (*pmat)[k*mat_cols+i]*(*pmat)[k*mat_cols+j];

   /* look for vector in matrix */
   k = eq = 0;
   while ((k<colno) && (abs(eq)<mat_rows)){
      eq = 0;
      for (l=0; l<mat_rows; l++) eq += (*pmat)[l*mat_cols+k] * temp_vect[l];
      if (abs(eq) < mat_rows) k++;
      }
   if (abs(eq) < mat_rows){
      /* found new column, add it */
      for (l=0; l<mat_rows; l++)
         (*pmat)[l*mat_cols+colno] = temp_vect[l];
      colno++;
      }

   /* advance to next column pair */
   j++;
   if (j == colno){
      i++;  j = i+1;
      if (j == colno)
         error( "Not enough independent columns in matrix.", NULL, -1 );
      }
   }

/* form conint */
for (i=0; i<mat_cols; i++) for (j=0; j<mat_cols; j++){
   for (k=0; k<mat_rows; k++) temp_vect[k] =
      (*pmat)[k*mat_cols+i]*(*pmat)[k*mat_cols+j];
   for (l=0, prod=1; l<mat_cols; l++){
      sum = 0;
      for (k=0; k<mat_rows; k++) sum += (temp_vect[k]*(*pmat)[k*mat_cols+l]);
      if (abs(sum) == mat_rows) { (*pconint)[i*mat_cols+j] = l; prod = 0; }
      }
   if (prod != 0) error("Non orthogonal matrix found.", NULL, -1);
   }

pfree((void **) &temp_vect);
return;
}
```

E.5.1 Coninc1.H

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <values.h>
#include <alloc.h>
#include <memory.h>
#include <math.h>
#include <ctype.h>

#define   DOS_EOF   26
#define   Line_Length      80
#define   Label_Length   16
#define   Field_Char      '*'
#define   Comment_Char    '%'
#define   Group_Threshold   6      /* must be > 0, must be < 16 */

#define   file_header
  "Experiment Design Program Data File - Do Not Rename or Delete!\n"
#define   file_divider
  "\n---------------------------------------------------------------\n"

/* matrix optimality structure */
typedef struct{
  int    set;
  int    curvar;
  float  curvsum;
  int    curint;
  float  curisum;
  } stats;

/* interaction storage structure */
typedef struct{
  void   *next;
  int    order;
  int    *vars;
  float  weight;
  float  total;
  float  coeff;
  int    type;
  } intstruct;

typedef struct{
  void   *next;
  int    num;
  char   name[Label_Length];
  float  coeff;
  } var;

typedef struct{
  void   *next;
  float  mat_weight;
  int    mat_size;
  int    num_vars;
  int    num_ints;
  int    avg_col;
  int    *int_type;
  int    *int_col;
  int    *int_data;
  int    *int_sign;
  int    *var_col;
  int    *var_label;
  float  *reg_coeff;
  } matstruct;

/* prototypes */
void   print_int_mat(int, int, int *);
void   print_flo_mat(int, int, float *);
void    print_int_vect(int, int *);
void   print_flo_vect(int, float *);

void   warning( char *message, char *filename, int linenum );
void   error( char *message, char *filename, int linenum );
void   unget_line( void );
void   pfree( void **block );
void   pmalloc( void **block, size_t length );
void   get_coefficients(int, int, int *, float *, float *, float *, int *);
void   get_row( int, int, int * );
void   create_matrix( int size, int *conmat );

int    get_line( char *out_str, FILE *handle );
int    get_label( char *out_str );
int    convert_label( int num_vars, char **var_label, char *label );
int    check_worse( stats *statvar );
int    update_best( stats *statvar );
int    con( int *assign, stats *statvar );
int    intcmp(var *var_root, char *label, intstruct *cur_int, int *data );
int    get_matrix_stats( int dim, int *matrix, int num_vars, int *vars,
        int *cols, intstruct *introot, stats *matstats, int print,
        FILE *handle, char **var_label, int max_label );
float   optimize(int *solved_var, int avg_col, int num_vars, int num_ints,
      int *int_type, int *int_col, int *int_data, int *int_sign,
      int *var_col, float *reg_coeff );
```

E.5.2 Coninc2.H

```
#include <time.h>

#define max_int     5
#define C_FACTOR    1000.00

/* Interaction constants for Combine.C */
#define cint_len      (num_vars+1)
#define cints_ptr(intno, data)      cints[(intno)*cint_len + data]
#define COR  0
#define CV1  1

/* Constants for OAAT.C */
#define lim_expts    16
#define list_len      max_int*int_len
#define int_len       7  /* list_len = max_int*int_len */
#define num_best     10
#define BIG_VALUE    10000

#define list_ptr(col, intno, data)
                list_int[(col)*list_len+(intno)*int_len+(data)]
#define LOR 0
#define LV1 1
#define LV2 2
#define LWE 5
#define LSG 6
void get_best_interaction(void);

/* Constants for databases */
#define MAX_BLK 5
#define intmat_row (9+MAX_BLK)
#define varmat_row (2+MAX_BLK)
#define blkmat_row (3+max_int)

/* Sizes of database matrices */
#define sof_intmat          (tot_ints*(9+MAX_BLK))
#define sof_varmat          ((num_vars+1)*(2+MAX_BLK))
#define sof_blkmat(bno)      (size_blk[bno]*(3+max_int))
#define sof_exptmat(bno)     (size_expt[bno]*num_vars)
#define sof_resmat(bno)      (size_expt[bno])
#define sof_selrow(bno)      (2*size_blk[bno])

/* Definitions of pointers to database matrices */
#define var_ptr(vno, data)   (var_mat[(vno)*varmat_row + data])
```

```
#define int_ptr(ino, data)   (int_mat[(ino)*intmat_row + data])
#define blk_ptr(bno, col, data) *(blk_mat[bno] + (col)*blkmat_row + data)

/* Constants for var_mat */
#define VNO 0       /*variable number */
#define VCO 1       /*variable coefficient */
#define VB0 2       /*col in which the var is present in blk 0 */

/* Constants for int_mat */
#define INO 0       /*interaction number */
#define IOR 1       /*interaction order  */
#define IV1 2       /*1st var of interaction */
#define IV2 3       /*2nd var of interaction */
#define IV3 4       /*3rd var of interaction */
#define IV4 5       /*4th var of interaction */
#define IWE 6       /*weight of the interaction */
#define IUC 7       /*unconfounded? 1/0 */
#define ICO 8       /*coefficient of interaction */
#define IB0 9       /*col in which the int is present in blk 0 */

/* Constants for blk_mat */
#define BCO 0       /*coefficient of the column in the block */
#define BVA 1       /*variable sitting on the column */
#define BNI 2       /*number of interactions present on the column */
#define BI1 3       /*location of the first interaction */

/* Storage of Databases */
#define data_file "c:\\rizwan\\data\\$d_base$.dat"
#define plant_file "c:\\rizwan\\data\\true_pr.dat"

/* Constants for sel_expt */
#define SBL 0       /*position of the block info */
#define SEP 1       /*position of the expt. info */
#define sel_ptr(bno, expt, data) *(sel_row[bno]+2*(expt)+data)

/* Constants for expt_mat */
#define expt_ptr(bno, expt_no) (expt_mat[bno] + num_vars*(expt_no))
#define res_ptr(bno, expt_no) *(res_mat[bno]+expt_no)

/* Types of blocks */
#define TYPE_FIR    0
#define TYPE_OAAT   1
#define TYPE_HBLK   2
#define TYPE_FBLK   3
#define TYPE_CBLK   4
```

```
/*routines for saving and loading the database */
void save_database(char *filen, int num_vars, int tot_ints, int num_blocks,
    int *size_expt, int *size_blk, int *type_blk, int *var_mat, int *int_mat,
    int **blk_mat, int **sel_row, int **expt_mat, float  **res_mat,
    char **var_label, int *blk_info);

void load_database(char *filen, int *pnum_vars, int *ptot_ints,
    int *pnum_blocks, int **psize_expt, int **psize_blk, int **ptype_blk,
    int **pvar_mat, int **pint_mat, int ***pblk_mat, int ***psel_row,
    int ***pexpt_mat, float ***pres_mat, char ***pvar_label, int **pblk_info);

/* misc. routines */
float oaat_opt(int *solved_var, int num_vars, int num_ints, int *int_data,
    int *int_sign, float avg_coeff, float *var_coeff, float *int_coeff);

void sort_interactions(int block_no, int *update_int, int num_vars,
    int tot_ints, int num_blocks, int *size_blk, int *type_blk,
    int *var_mat, int *int_mat, int **blk_mat);

void set_mat(int num_vars, int block_no, int **expt_mat, int **sel_row,
    int *size_blk, int *pmat_rows, int *pmat_cols, int **pvar_col, int **pmat,
    int **pconint);

/*routines for displaying information */
void disp_var_info(int num_vars, int num_blocks, int *var_mat, int *type_blk,
    char **var_label);

void disp_int_info(int tot_ints, int num_blocks, int *type_blk, int *int_mat,
    char **var_label);

void disp_blk_info(int num_blocks, int *size_expt, int *size_blk,
    int **blk_mat, int *type_blk, int *blk_info);

void disp_col_info(int block_no, int *size_blk, int **blk_mat);
```

E.5.3 True_Process.Dat

```
*number of var
7
*avg
22.5
*vars
1 5.3
2 7.3
5 -4.9
3 3.1

*interaction
3
2 3 4  -8.4
2 1 7  -4.3
2 5 6  4.7
*lambda
4

*end
```

E.5.4 Interaction.Dat

```
*interactions

5
4 1 2 3 6 20
2 1 5    10
3 6 3 2  5
2 4 1     8
3 4 3 1  12

*end
```

Bibliography

[1] Ashraf Alkhairy. Optimal Product and Manufacturing Process Selection - Issues of Formulation and Methods for Parameter Design. *RLE Technical Report No. 572*, 1992.

[2] A. C. Atkinson and A. N. Donev. *Optimum Experimental Designs.* Oxford Science Publication, Oxford, 1992.

[3] Soren Bisgaard. A Method for Identifying Defining Contrasts for 2^{k-p} Experiments. *Journal of Quality Technology, 25, p28-35*, 1993.

[4] G. E. P. Box, W. G. Hunter, and J. S. Hunter. *Statistics for Experimenters.* John Wiley, New York, 1978.

[5] G. E. P. Box and R. D. Meyer. An Analysis for Unreplicated Fractional Factorials. *Technometrics, 28, p11-18*, 1986.

[6] G. E. P. Box and K. B. Wilson. On the Experimental Attainment of Optimal Conditions. *Journal Roy. Stat. Soc. Ser. B., 13, p1-45*, 1946.

[7] C Daniel. Use of Half-Normal Plots in Interpreting Factorial Two-Level Experiments. *Technometrics, 1, p311-341*, 1959.

[8] M. N. Das and N. C. Giri. *Design and Analysis of Experiments.* Wiley Eastern Limited, New Delhi, 1991.

[9] Aloke Dey. *Orthogonal Fractional Factorial Designs.* Wiley Eastern Limited, New Delhi, 1985.

[10] Paul Fieguth, M. S. Spina, and D. H. Staelin. Conformal Design of Experiments: An Automated Tool for Supporting Experimental Design and Analysis for Efficient

Improvement of Products and Processes. *MIT Leaders for Manufacturing Program Publication*, 1992.

[11] R. A. Fisher. The Arrangement of Field Experiments. *Journal Min. Agri. 33, p503-513*, 1926.

[12] R. A. Fisher. The Theory of Confounding in Factorial Experiments in Relation to the Theory of Groups. *Biometrika, Ann. Eugen. XI, p341*, 1942.

[13] S. Ghosh. On some new Search Designs for 2^m Factorial Experiments. *Journal of Statistical Planning and Inference, 5, p381-389*, 1981.

[14] Arnon Hurwitz. Sequential Process Optimization with a Commercial Package. *Sixth National Symposium on Statistics and Design in Automated Manufacturing*, 1993.

[15] Russell Lenth. Quick and Easy Analysis of Unreplicated Factorials. *Technometrics, 31, p469-473*, 1989.

[16] T. J. Mitchell. An Algorithm for Constructing D-Optimal Design. *Technometrics, 16, p203-210*, 1974.

[17] D. C. Montgomery. *Design and Analysis of Experiments.* John Wiley and Sons, New York, 1991.

[18] Srivastava J. N. On the Inadequacy of Orthogonal Arrays in Quality, Control and General Scientific Experimentation and the need for Probing Designs. *Comm. Statis. 16, p2901-2941*, 1987.

[19] R. L. Plackett and J. P. Burman. Design of Optimal Multi-Factorial Experiments". *Biometrika, 23, p305-325*, 1946.

[20] E. G. Schilling. The Relation of Analysis of Variance to Regression. *Journal of Quality Technology, 6, p74-83*, 1974.

[21] J. N. Srivastava. Design for Searching Non-negligible Effects. *A Survey of Statistical Designs and Models, Amsterdam, p507-509*, 1975.

[22] J. N. Srivastava and S. Ghosh. Balanced 2^m Factorial Designs of Resolution V which allows Search and Experimentation of one Extra Unknown Effect, $4 \leq m \leq 8$. *Comm. Statis. - Theory Methods A6, p141-166*, 1977.

[23] G. Taguchi. *Introduction to Quality Engineering.* Asian Productivity Organization, Tokyo, 1986.

[24] G. Taguchi. *System of Experimental Design.* UNIPUB/Kraus International Publication, White Plains, New York, 1987.

[25] C. F. J. Wu and Chen Youyi. A Graph-Aided Method for Planning Two-Level Experiments when Certain Interactions are Important. *Technometrics, 34, p162-174*, 1992.

[26] W. J. Youden. Partial Confounding in Fractional Replications. *Technometrics, 3, p353-358*, 1961.