The automatic design of experiments : Some practical algorithms.

GREENFIELD, A. A.

The Automatic Design of Experiments

Some Practical Algorithms

by A. A. Greenfield B.Sc., F.I.S., F.S.S.

A thesis submitted to the Council for National
Academic Awards for the degree of Doctor of
Philosophy

April 1979 Sheffield City Polytechnic

The Automatic Design of Experiments

Some Practical Algorithms

ABSTRACT

The purpose of this study was to develop a methodology, represented as a set of programmable algorithms, for the design of experiments of the types that are generally likely to be useful in the physical sciences. This has been achieved by adding to the established theory and practice of designing factorial experiments for both qualitative and quantitative variables.

Algorithms were developed for designing fractional two-level factorial experiments according to a pre-specified model to be fitted, expressed in terms of required effects to be estimated. These algorithms are extended in two ways. One of these is to allow a fractional two-level factorial design to be augmented with extra points so that quadratic effects can be estimated. The second is to enable fractional asymmetric multi-level factorial experiments to be designed: balanced fractions first by applying the theory of cyclic groups; then further reduction in the size of the design by using the trace and determinant of the information matrix.

The application of the algorithms is illustrated with examples drawn from the physical sciences, particularly metallurgy. The algorithms developed in the study have been fully implemented using standard Fortran 4 with a few specified exceptions. These programs are listed in three appendices. The programs have been run on computers in research laboratories in Australia and the United States as well as in Britain. They will benefit research scientists who are planning experiments and have access to interactive computers.

The principles of algorithmic development are explained and the whole text is supported by references and by a glossary of more important terms.

The Automatic Design of Experiments

Some Practical Algorithms

CONTENTS

The Automatic Design of Experiments

Some Practical Algorithms

CHAPTER ONE

INTRODUCTION

1 Background

Gauss (1809) was the first person to allude to the design considerations of making physical observations. Most of his great work on the theory of the motions of heavenly bodies was devoted to the development of algorithms for computing orbits from precise observations. Then, in the third section of the second book of the work, he developed the normal, or Gaussian, density function and the method of maximum likelihood, and he presented the method of least squares (which he claimed to have been using since 1795) and the method of weighted least squares. In the midst of this he commented, as an aside and without proof, that if only a few observations were to be made to determine an orbit they should be as remote from each other as possible to minimise the effects of observational errors. When he later developed this statistical theory into a full treatise (1821) he discussed at some length the further design problem of the effect of an extra observation on already estimated coefficients and the conditions that must be imposed to ensure minimum variance of the estimates.

It was a century later when Smith (1918) suggested maximising the determinant of **X'X**, known as the information matrix or the cross-product of the design matrix (**X**), as a criterion for designing experiments. The determinant is inversely proportional to the generalised variance of the estimated coefficients. Smith applied her criterion to experiments for estimating polynomials of varying degrees. Given the degree of the polynomial to be fitted, the number of observations to be made, and the interval over which the polynomial should be fitted, she determined the spacings between observation points and the proportions of observations to be made at those points in order to minimise the generalised variance of the estimates of the coefficients of the polynomial. The method is neatly presented in three pages by Kendal and Stuart (1966) who had the advantage over Smith of modern matrix notation and algebra. However, despite the clumsy notation of her day, Smith went on to consider the effect of heteroscedasticity of errors on the optimum allocation of observations.

The concept of experimental design grew most rapidly in agricultural work. Fisher (1925) introduced the subject briefly in his first edition of 'Statistical Methods for Research Workers.' He illustrated that applied statisticians were mainly concerned with data examination, analysis, and statistical tests of contrasts. The method of experimental design seems to have been: think of an arrangement of trials, such as a Latin square, then see if the arrangement meets the experimental criteria. These criteria were: can the desired contrasts be estimated from the data, and will the arrangement lead to statistical tests of the estimated contrasts?

It seems as if Fisher realised the importance of experimental design at the time the book was published, for within a year (1926) he published a major paper on 'The arrangement of field experiments' and later wrote the first definitive text (1935) on 'The design of experiments'.

Fisher had a profound effect on the development of experimental design because his work was based on agricultural experience in which the independent variables were usually qualitative factors or could be treated as such, and the method of analysis was 'analysis of variance'. He and his contemporaries applied considerable ingenuity to finding designs which were orthogonal in that the contrasts between levels of different factors could be estimated independently. A host of design types was developed: randomised blocks, balanced incomplete blocks, split plots, Latin squares, Youden squares, and lattices, among the better known. Many books were written about these agricultural designs, variations on them, applications, their analysis, and what to do when missing values upset the balance and made estimation and testing difficult. The most authoritative of the books covering the subject were probably Cochran and Cox (1950), Kempthorne (1952), Brownlee (1953), and Davies (1954). Uses were found for these designs in other than the agricultural sciences. They were applied with quantitative variables as well as qualitative: the levels of the quantitative factors were usually equally spaced for convenience. The criterion proposed by Smith in 1918 seemed to have passed unnoticed.

The theoretical unification of the methods of analysing these types of design was presented by Tocher (1952) when he at the same time suggested that computers be set to work to generate all possibly useful designs. There were several warnings in the discussion of that paper against the 'sausage machine' approach to experimental design.

In his 1926 paper, Fisher discussed the concept of factorial experiments. There is no trace of an individual originator of these; they seem to have grown out of general discussion at Rothamsted. Fisher argued that while these could become very large and complex experiments, they had the advantages that:

1. the plots are used several times over to determine the average effects of different factors;
2. only by factorial design can any information be obtained on how responses to one factor are affected by another (that is: they permit the estimation of interactions);
3. factorial experiments provide a wider inductive basis for conclusions on the effects of the factors;

At the same time he recognised the possibility of confounding: the deliberate sacrifice of some unimportant information so as to improve the precision of estimates of important effects. The methodology for dealing with confounding was developed by Yates (1933) who also designed an algorithm (1937) for easy analysis of two-level factorials. A further advantage of confounding was soon realised: it could be used to select a fraction of a factorial. The theory of fractional replication was developed by Finney (1945) and Kempthorne (1947).

The factorials which have had the widest impact are those in which each factor has only two levels. Their advantages are that they are easy to design and to analyse and they can just as easily represent quantitative variables as well as qualitative. They have further advantages which will be discussed in later chapters, since much of the work of this study is based on the design of fractional two-level factorials.

The development of agricultural type experiments by the Fisher group was represented in so much literature that for several decades the rest of the scientific world was largely misled into believing that the subject of experimental design comprised an understanding of only those agricultural designs. Also, because the mean effects or contrasts were so easy to compute, estimation was largely disregarded as an aspect of analysis and the emphasis was placed on tests of significance. In'The Design of Experiments' Fisher wrote: 'Every experiment may be said to exist only in order to give the facts a chance of disproving the null hypothesis.'

Experimentation in the physical sciences is often much more complicated than the traditional field trials, and estimation of effects is not so easy. Thus analysis tends to be by the regression method of least squares as developed by Gauss rather than by Fisher's analysis of variance. It was Tocher (1952a) who showed that regression analysis was applicable to designed experiments as well as to naturally occurring data. This observation, together with a growing literature on determining the relative efficiencies of designs (see Wald (1943) and Ehrenfeld (1953)), led Kieffer (1959) to resurrect the criterion suggested by Smith in 1918. This criterion, which is one of several alternatives described by Kieffer, has subsequently been the subject of many published papers. It is known as the criterion of D-optimality and, in simple terms, it expresses the objective of choosing a design which maximises the determinant of the $\underset{\sim}{X}'\underset{\sim}{X}$ matrix. It is unfortunate, however, that the many papers published by the Kieffer school (see Kieffer and Wolfovitz (1959), Kieffer (1959), Wynn (1970, 1972)) are long, intricate, mathematically involved, and literally obscure, so that they have had little influence on the originally applied subject of experimental design. Indeed the subject has gone two ways: at the applied level there have been some developments of agricultural type experiments, particularly in the augmentation of two-level factorials with extra points to permit the estimation of quadratic effects (which will be described in chapter four); and on the theoretical level it has become a branch of optimising mathematics remote from the original intended purpose. The mathematical developments are summarised by Fedorov (1972).

2 Objectives

One of the objectives of the present research has been to make a practical contribution which will help non-statistical research scientists, particularly physical scientists such as chemists, physicists, metallurgists, and engineers, with a fairly closely defined sub-area of what has become a massive subject. Just as there has had to be some selection of material for the preceding historical introduction, with many aspects omitted and many contributors unmentioned, the choice of a sub-area that can reasonably be tackled in a single study must inevitably leave most of the subject untouched.

The choice will be discussed more fully in chapter two. At this stage let it suffice that the aim has been to meet most of the experimental design needs of physical research and to develop automatic methods of design that will obviate the need for the research worker to identify the type of design suitable for his work. The interactive nature of the algorithms that have been developed will lead naturally, through questions and answers about his research objectives, to the identification of a suitable type of design.

A classical research situation, encountered almost daily in any industrial laboratory for research and development, can be described as follows:

The objective is first declared as the optimisation of a product or a process; the characteristics of that product are identified; precisions of measures of those characteristics are stated; and the control variables are identified, usually the compositional and process variables, with ranges and precisions.

Sometimes the objectives of the experiment are represented as a mathematical model relating the measures of the product characteristics with the control variables, but this is rare. More usually the research worker has little idea of the pertaining relationships and can express them only in vague qualitative terms. Clarity usually follows questioning, however, so that it is possible to write down at least a simple linear model including expected first order interactions and perhaps also to include some quadratic terms.

On the basis of this information an experiment is designed so as to estimate the parameters of the model as precisely and as accurately as possible within the limitations of experimental costs. The objectives of an experiment must always be to answer a precisely stated question or set of questions. Almost always these questions can be stated in terms of a mathematical model whose parameters are to be estimated or perhaps compared with an alternative model. Sometimes an objective goes so far as to include optimisation, but even this is a particular case of estimation.

Experimental design can be laborious if done manually. Some research workers, familiar with fractional two-level factorial experiments, have spent days finding a suitable fraction. Even then the labour was probably worthwhile because a suitable fraction would achieve the experimental objectives with considerable cost and time savings. It was the original purpose of this study to assist the research worker in obtaining quickly and easily an experimental design suitable to his objectives. The aim was for the following dialogue to take place between the laboratory computer and the research metallurgist. The dialogue would be through a keyboard and typewriter terminal.

The user would first establish the date and research name, whereupon the program would open up a new data file under that name. It would then begin to ask the user questions about his variables. Which are the dependent variables and which are the independent? The answers may be given as names or as numbers. Also identified would be the intermediate variables which, to the physical metallurgist say, may be worth recording to extend his fundamental understanding of the subject, but from the viewpoint of a predictive statistical model may be ignored. An example of this type of variable is grain size. It is not an independent variable from the viewpoint of experimental design because it cannot be controlled directly. Strictly it is a dependent variable because grain size is determined as the response to control or independent variables such as composition and process treatment. On the other hand it cannot be claimed to be a commercial characteristic of steel, although there are said to be relationships between the grain size and the commercial characteristics. So I call it an intermediate variable.

Having established the names of the variables and their type, the computer program would probe deeper. What are the anticipated ranges of the variable values? What interactions exist between independent variables? (The meaning of "interaction" will be discussed later). And what curvatures might be expected for each of the dependent variables? What accuracies might be expected in meeting the specifications of independent variables? Are there any practical reasons for dividing the full set of observations into blocks? Which of the variables are quantitative, and of these which have continuous values and which have discrete values? Which of the variables are qualitative and how many levels of each quality are there? Are there any mutual constraints between the variables: such as in a metallurgical experiment when one element must have a low and narrow range when a second element is at a high level, and vice versa, so that their mutual region of variation is banana shaped?

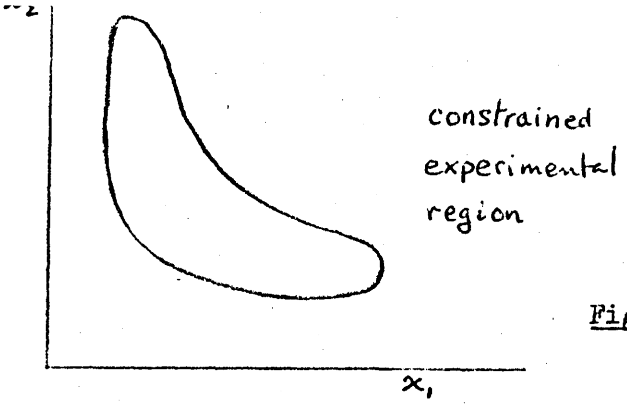


Figure 1

The computer program would allocate a disk area to the research project and would store the information so far obtained. It would then produce the most efficient design corresponding to this information.

The research worker would be expected to follow the computer-printed design and return to the computer later with his results. The analysis programs would take into account any missing, spoilt, or extra data. The computer would produce reports in the form of prints of the analysis, plots of contours, and sections of response surfaces. These would assist the user to determine whether to make further observations, in which case the computer would offer its advice on further observation points, or to produce a final report and clear the disk area for another user.

Process research increasingly calls for the real-time analysis of data as it is collected, rather than waiting for an experiment or a series of experiments to be completed before data analysis begins. This presents the further challenge of automatic sequential analysis of data and synchronous revision of experimental design. Thus the aim of this research included, originally, the prospect of extending to the on-line situation the automatic design and analysis of experiments already described. In these cases we should be logging data from and controlling processes whose properties may not be known in advance: the computer would establish mathematical models describing the processes and would improve these models as it acquired more data. Thus the computer would learn from experience, but rather more quickly than a human being, although admittedly with some limitations.

The system would be entirely flexible so that even if operations changed overnight from metallurgy to hydroponics it would still be useful. Within each laboratory there would be a wall-mounted termination board and keyboard connected directly to the central computer. Terminals would be labelled with types of signals that could be connected: analog input and output, digital input and output; and the permitted voltage ranges. Within the computer would be a suite of generalised data analysis, acquisition, and control programs. The user in the laboratory would connect leads from his experimental process to the termination board. Through his keyboard he would have a conversation with the computer similar to that described for the off-line automatic design and analysis of experiments. He would signal to the computer when the experiment was set up and ready to go. And it would go! The flexibility must be stressed: it would not matter to the system whether the experimental process under study was a miniature electroslag refining plant or a tomato plant, so long as the signal types, voltage ranges, and sampling frequencies were suitable to the computer.

In some ways the original aim of this research as described above was over-ambitious and unrealistic. Within limitations, however, it is still practical and achievable, certainly worth pursuing, and some of it is already within reach.

One of these limitations is dictated by the plethora of approaches to experimental design. The review paper by Herzberg and Cox (1969) listed nearly 900 references. It is notable that most were of a highly theoretical and non-applied nature and none was concerned with automatic design of experiments as an aid to the industrial research scientist. That paper nevertheless highlighted a point of considerable importance in this current study: that the class or classes of experiment studied should be sufficiently narrow to allow significantly noticeable and useful progress. This point was made by Tocher(1952b)who wrote: "It soon became clear that any such account, if treated in the detail commensurate with the importance of the subject, would be excessively long and that some curtailment of the programme would be necessary. Consequently, ... attention is concentrated almost entirely on those experiments normally referred to as block experiments."

The choice of experimental classes has accordingly been restricted in this present study and is discussed in the next chapter.

A further limitation is the extent to which a conversation between research worker and computer can be allowed to proceed without the guidance of a statistician. While it should be possible to develop programs to support question and answer routines with descriptive text and graphical illustrations to explain difficult points to the conversing scientist when he seeks clarification, it became apparent during the study that such a system would be far from easy to implement. Indeed, to be wholly satisfactory, it would need a much deeper study into the psychology and linguistics of program instruction. Hence, while the original aim of developing automatic design procedures was maintained, it has been restricted to providing an aid to the applied consulting statistician and to the initiated research scientist rather than providing a conversational system available to all-comers regardless of their knowledge, or lack of knowledge, of elementary mathematical modelling and experimental design and analysis.

The remainder of this thesis comprises chapters on the choice of experimental classes, approaches adapted to their automatic design, some computer programming points, data analysis, and some applied examples. New contributions to the subject and outstanding problems are identified throughout the thesis.

3 Algorithms

In publications related to the earlier stages of this research (Greenfield (1972,1974)) the procedures that were developed were illustrated in program segments written in an extension of standard Fortran 4. The full programs were published as appendices. These programs were freely available and research laboratories in several countries tried to implement them. Some were successful and some were not. The problem was that programs are in general not easily portable from one machine to another even if the machines are claimed to support the same high level language because differences between machines lead to a unique dialect of a high level language for each. If a program were written precisely in Fortran 4 it would be portable. However, dialects are sufficiently different that an implementer may not see how to convert a program. Some dialects have extensions that are oriented towards the class of applications for which the computer has been designed. This applies particularly to scientific computers. It may be argued that programmers should stick rigidly to the standard, but if they do not use all the available extensions they are underusing the facilities.

In a later publication (Greenfield (1976)) the method of selecting defining contrasts in two-level experiments was described in a sequence of simple steps expressed in English. Subsequent correspondence proved that the procedure was immediately more comprehensible to potential users than if it had been published as a detailed Fortran program. A list of program statements is not the clearest way to describe a complicated procedure. On the other hand, a sequence of steps in English is not adequate to describe other than the simplest of mathematical procedures. In this thesis, therefore, an algorithmic style, which has recently become conventional in the computing world, will be used. This has advantages that will be described below.

An algorithm is a sequence of rules for solving a problem, usually, but not always, mathematical. The word is not new. It has been used with this meaning in English, German, and Latin (algorismus) for some centuries. Much of Gauss's astronomical and statistical work was couched in algorithmic terms and he used the word in the modern sense. However, in recent years it has become clear in computing circles that communication would be greatly improved if a universal convention for stating algorithms were adopted.

The primary purpose of an algorithm is to specify the correct sequence of rules for transforming an initial value, or a set of initial values, into a final value, or a set of final values. For example, an algorithm to design an experiment will be a sequence of rules that will transform an initial statement of 'model, variables, and allowable values' into the design of an experiment which would yield data suitable for the estimation and testing of the model coefficients.

A secondary purpose of an algorithm is to supply a sequence of rules that will minimise the time and effort needed to reach the correct solution to the problem for any arbitrary initial values.

Another purpose of an algorithm, and one that has become increasingly important, is to provide a sequence of rules that are easy to understand, simple to prove correct, and easy to change if the specifications of the problem and the way to solve it change. As algorithms become more and more complex there is increasing difficulty in understanding how they work, how to find and correct errors, and how to make development changes. It has been claimed that more than half a programmer's time is spent dealing with program correction, maintenance, and modification. Leading programmers, most notably Dijkstra (1973), Knuth (1973), andGoodman and Hedetniemi (1977), have developed a convention and a set of techniques that have already become widely adopted. These comprise a notation for flowcharts, a notation for the stepwise description of an algorithm, and an approach to programming known as 'top-down structured programming'. These will be explained here because they are used throughout the rest of this thesis to describe the experimental design algorithms. If these are properly understood, then any programmer, using any programming language, on any machine, should be able to transcribe the algorithms into working programs. Furthermore, for simple experimental designs, the designer should be able to follow the algorithms using pencil and paper to design his experiment manually.

A flowchart is a directed network having three kinds of box:

A function box, illustrated here, is used to represent a function $f: X \rightarrow Y$

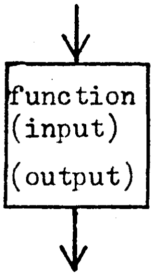


Figure 2

A predicate box, illustrated here, is used to represent a logical function

$p: X \rightarrow \{T, F\}$

which passes control along one of two paths.

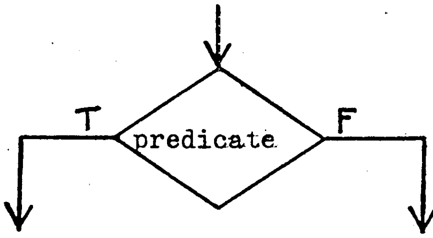


Figure 3

A collecting box, illustrated here, represents the passage of control from one of two incoming paths to one outgoing path.

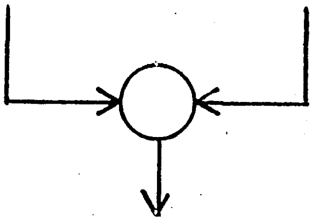


Figure 4

A structured program is one that can be expressed as a composition of the following four primitive flowcharts:

A functional composition, illustrated here, which is simply a sequence of function boxes.

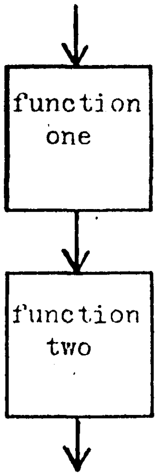


Figure 5

A selection, illustrated here, which uses a logical test, whose outcome is either true or false, to determine which of two alternative functions should be done. In practice, a test with more than two outcomes may be used but this is equivalent to a sequence of two-outcome tests.

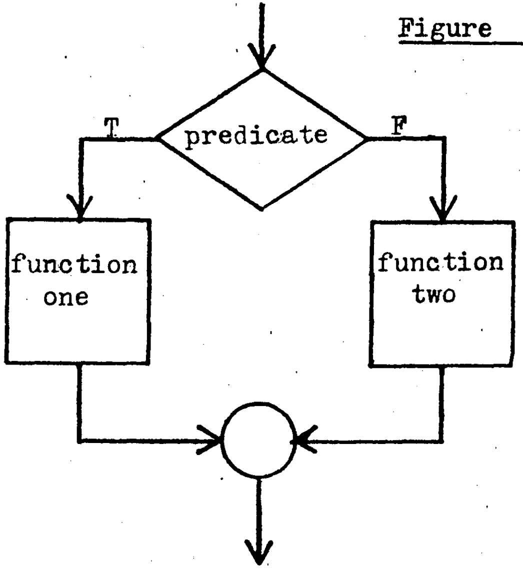


Figure 6

Two forms of iteration in which a logical test is used to decide whether or not a function should be repeated. The distinction between the two forms is that in one the first time the test is met is before the first time the function is met, and in the other the order of the first meeting is reversed.

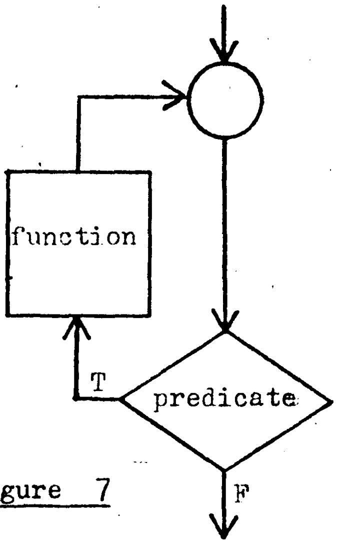


Figure 7

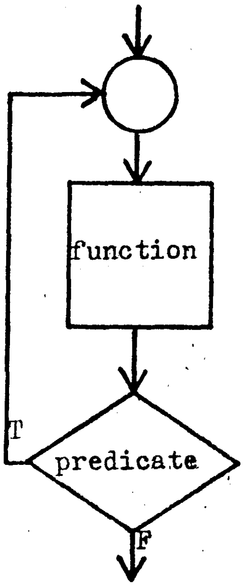


Figure 8

Structured programming is the process of designing algorithms in terms of structured flowcharts. Top-down structured programming means starting with a general statement of a function and then analysing it a step at a time into levels of greater detail until the stage is reached when code can be written easily in a high level language to implement the developed algorithm. This final stage is best done in certain languages like Algol and Coral which have been designed with an algorithmic nature. It is much more difficult, although still possible, with Fortran which has a different structure. I shall however use Fortran to illustrate programming features because it is by far the most widely used scientific programming language.

The top-down structured programming procedure will be illustrated with reference to Euclid's algorithm for determining the highest common factor (hcf) of two integers. This is also known in America as the greatest common divisor (gcd). I have chosen Euclid's algorithm as an illustration for three reasons: it is needed as a subroutine in the experimental design algorithms developed in chapter six; it is complex enough to illustrate development at several levels of detail; it is short enough and simple enough to serve as an illustration.

At the same time as using the example to illustrate top down programming in terms of flow chart representation, I shall use the occasion to illustrate the conventional linguistic representation. This bears a striking resemblance to the programming language Algol, an apt neologue from 'algorithmic language'. The conventions used for describing algorithms are however much more flexible than those of a programming language which has strict rules rather than useful conventions. Thus an algorithmic step may be described in the broadest functional terms using English or mathematical notation, rather than in explicit computational expressions, assignments and tests, although at the final stage of developing an algorithm these latter will appear.

One of the conventions is to exploit different typefaces to clarify meaning. Commonly used words are set in lower case boldface, indicated in typed copy by a wavy underscore. Examples are: **algorithm**, **and**, **do**, **else**, **fi**, **for**, **goto**, **if**, **od**, **set**, **then**, **through**, **to**, **while**. An algorithm name is set in boldface capitals, such as **HCF**. The derivation of an algorithm name may be italicised in parentheses. In typed copy italics are indicated by a straight underscore. For example:

Algorithm **HCF** (*Highest Common Factor*)

The word 'step' followed by a number, is used to label a step in the algorithm and is also set in italics: *Step 5* **This label may be indented to indicate the level of logic.** Immediately after the label *Step i*, a brief phrase in roman medium typeface in square brackets to describe the purpose of the step. Further comments, also in roman medium type, may appear within the step and are usually separated by semi-colons.

Mathematical, logical, and computational expressions are also put in medium type. The reverse arrow is used for assignment. For example:

$$K \leftarrow 5$$

means that the variable K is assigned the value 5.

The two words **fi** and **od** have been introduced so that the ends of conditional statements and sequential statements, initiated by the words if and do respectively, can be identified unequivocally.

In top down structured programming we repeatedly ask if the function being considered can be expressed as a primitive flowchart. Top down programming is illustrated in the following example which starts with a single function box. In practice, I do not always use strictly structured programming because it sometimes seems clumsy.

Algorithm HCF (Highest Common Factor)

Given two positive integers, j and k, find their highest common factor which is the largest positive integer, h, which divides both j and k.

Step 1 read j, k
Step 2 h ← hcf(j,k)
Step 3 write h

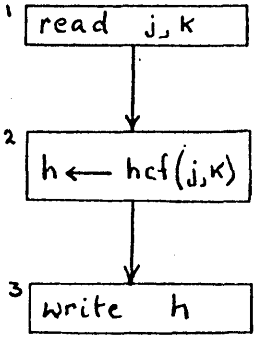


Figure 9

At this stage the method of evaluating the hcf has not been described, but the function has been expressed formally; that is, the function and variables have been indicated. The next stage is to analyse the function in terms of one of the primitive flowcharts. Statements that can be made immediately are:

a) If $j = k$ then $hcf(j,k) = j$
b) If $j = 1$ or $k = 1$ then $hcf(j,k) = 1$
c) If $j = 0$ then $hcf(j,k) = k$
d) If $k = 0$ then $hcf(j,k) = j$

However, the application of this function in experimental design is to do with factors with more than one level (see chapter six) and the initial values of j and k will always be greater than one. Thus only question (a) need be asked and the function meay be replaced by a primitive selection flowchart:

```
Algorithm HCF ( "  "   " " ")
     step 1 read  j,k
     step 2 if  j = k then step 3 h←j
               else step 4 h←hcf(j,k) fi
    step 5 write  h
```

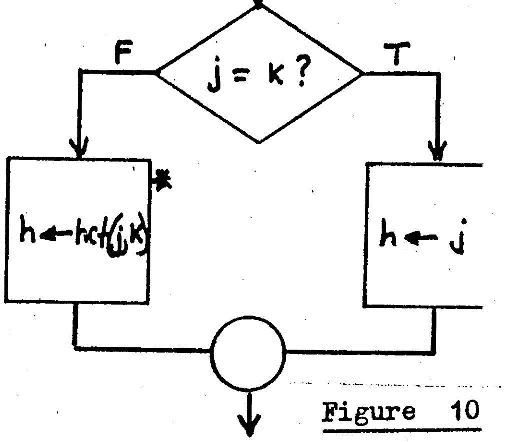


Figure 10

Schoolchildren are taught to find HCF's by finding all the prime factors of both initial values and comparing them. Euclid's algorithm is based on the division theorem which states: If a and b are two positive integers then two integers q and r can be found such that

$$a = bq + r$$

It is clear that the highest common factor of a and b must also be a factor of r. Thus $hcf(a,b) = hcf(b,r)$.
If this division theorem is applied repeatedly, a remainder of zero must ultimately appear and the last positive remainder before that must be the hcf of a and b.
Now, substituting j and k for a and b, the function box marked * in the last flowchart may be analysed as either of the following:

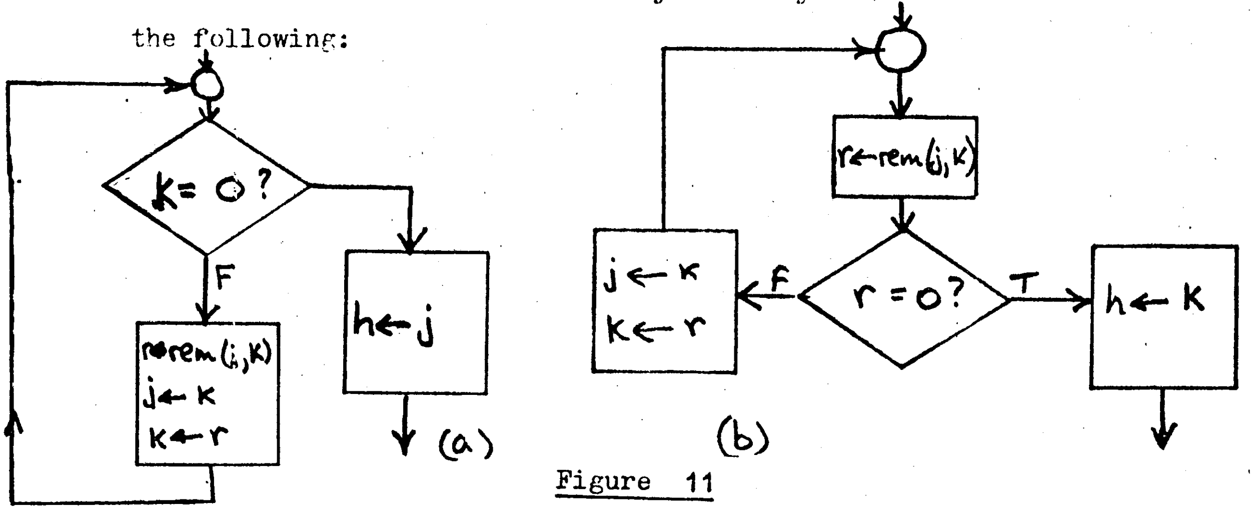


Figure 11

In both of these, the expression rem(j,k) means the remainder when integer j is divided by integer k. The first of the alternatives (a) strictly follows the conventional iteration primitive flowchart, but since this would lead to two more assignments and one more test than the second alternative (b), the latter is preferred.

Alternative (a) would be expressed as:

```
step 4  while k ≠ 0 do step 5 od
        step 5  do r←rem(j,k); j←k; k←r od
step 6  h←j
```

Whereas alternative (b) would be expressed as:

```
step 4  r←rem(j,k)
step 5  if r ≠ 0 do step 6 od fi
        step 6 do j←k; k←r; goto step 4 od
step 7  h←k
```

The solution of rem(j,k) may be left to the final programming stage in the knowledge that in many Fortran function libraries there is a function MOD(J,K) which is assigned the value of the remainder when J is divided by K. Without this function the expression K-(K/J)*J may be used to give the remainder since the first part of the expression to be evaluated (K/J) returns only the partial or integer quotient.

There is one further small refinement to be made to the algorithm. If at some stage the remainder is one, there is clearly no need to repeat the procedure and determine that the remainder at the next stage is zero. We can conclude instead that j and k are mutually prime, that is hcf(j,k) = 1. With this test added, and the steps renumbered, the algorithm and flowchart become:

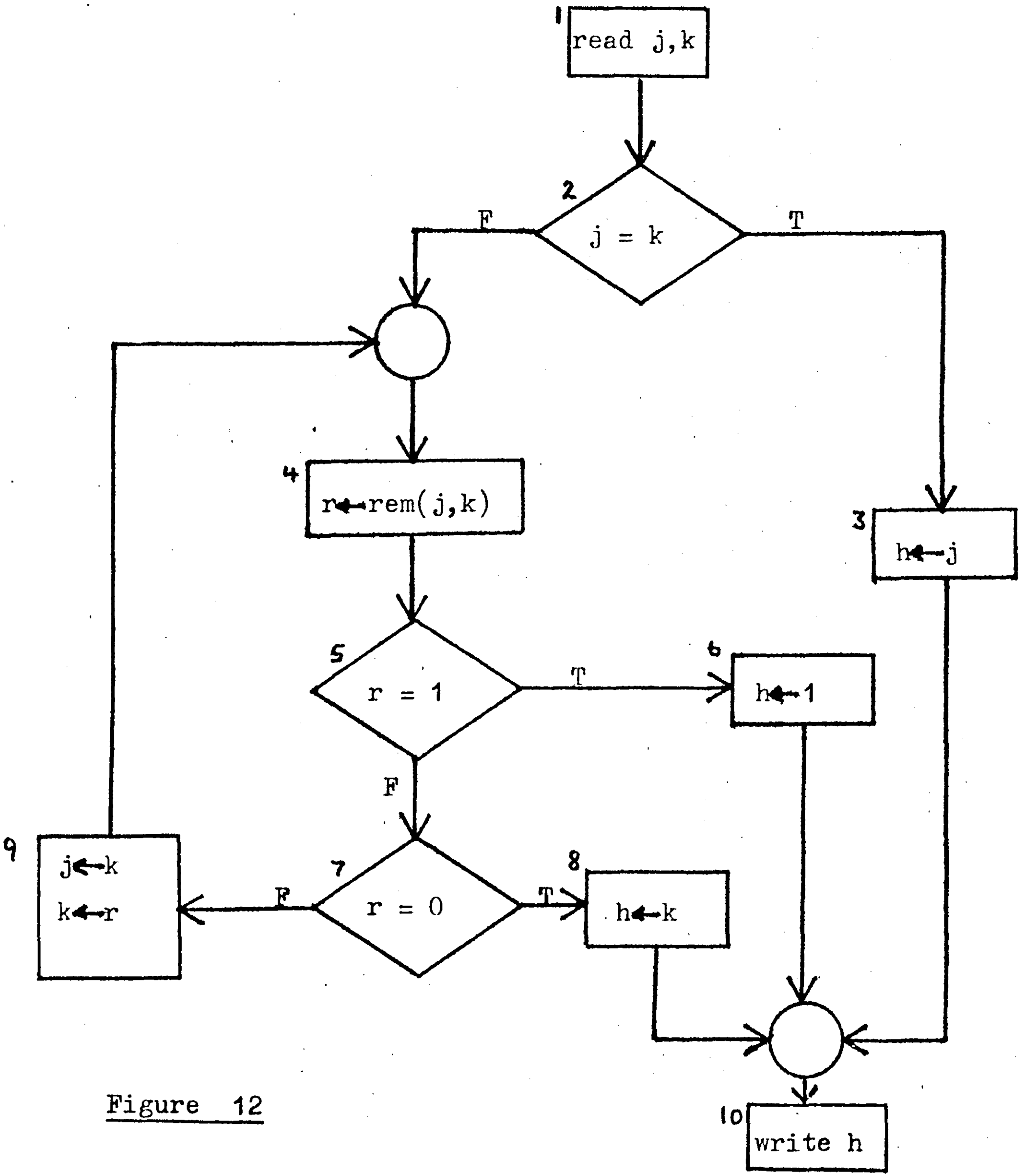


Figure 12

Algorithm HCF (Highest Common Factor) Given two positive integers j and k, find their highest common factor which is the largest positive integer, h, which divides both j and k.

```
Step  1   read  j, k
Step  2   if  j = k  then step 3  h← j
                     else  do step 4;  step 5  od  fi
Step  4   r← rem(j, k)
Step  5   if  r = 1  then step 6  h←1
                     else do step 7  od  fi
Step  7   if  r = 0  then  step 8  h←k
                     else  step 9  do  j← k;  k← r;
                           goto step 4  od fi
Step 10   write h
```

The next stage, writing the program, will be illustrated here although it will normally be left out of the main text and put in an appendix. Coding in Algol after the final algorithmic statement is straightforward. However, since Fortran was not designed with structured programming in view, some departures from the algorithm may be indicated. One useful device in Fortran is the three-way conditional statement IF(X)a,b,c where a, b, and c are three branch labels according to whether X is negative, zero, or positive. This is used in the following:

```
  FUNCTION  IHCF(JJ,KK)
  IF(JJ.EQ.KK) GO TO 5
  J=JJ
  K=KK
1 L=K-(K/J)*J
  IF(L-1)3,4,2
2 K=J
  J=L
  GO TO 1
3 IHCF=J
  RETURN
4 IHCF=1
  RETURN
5 IHCF=JJ
  RETURN
  END
```

There are a few minor points to note in this function routine. The function name has been changed from HCF to IHCF and the remainder variable has been called L so that integer values are implied according to the usual Fortran convention. Also the function does not operate on the integer variables passed to it by the main program but copies them first. This is to avoid corruption of the variables in the main program.

This routine will execute Euclid's algorithm for all integers. The division theorem ensures that even for pairs of very large integers the algorithm will yield the hcf after relatively few iterations. In the application to be developed in chapter 6, however, it will rarely be used with integers greater than, say, 20. This suggests that if a difference is used instead of a remainder, the algorithm will work even more quickly. The computation of a remainder calls for a division and a multiplication which are both computationally slow compared with a subtraction. Thus, reverting to figure 12 and substituting r←j − k in place of r←rem(j,k), and then observing that this calls for j to be greater than k, the flowchart and algorithm may be revised as:

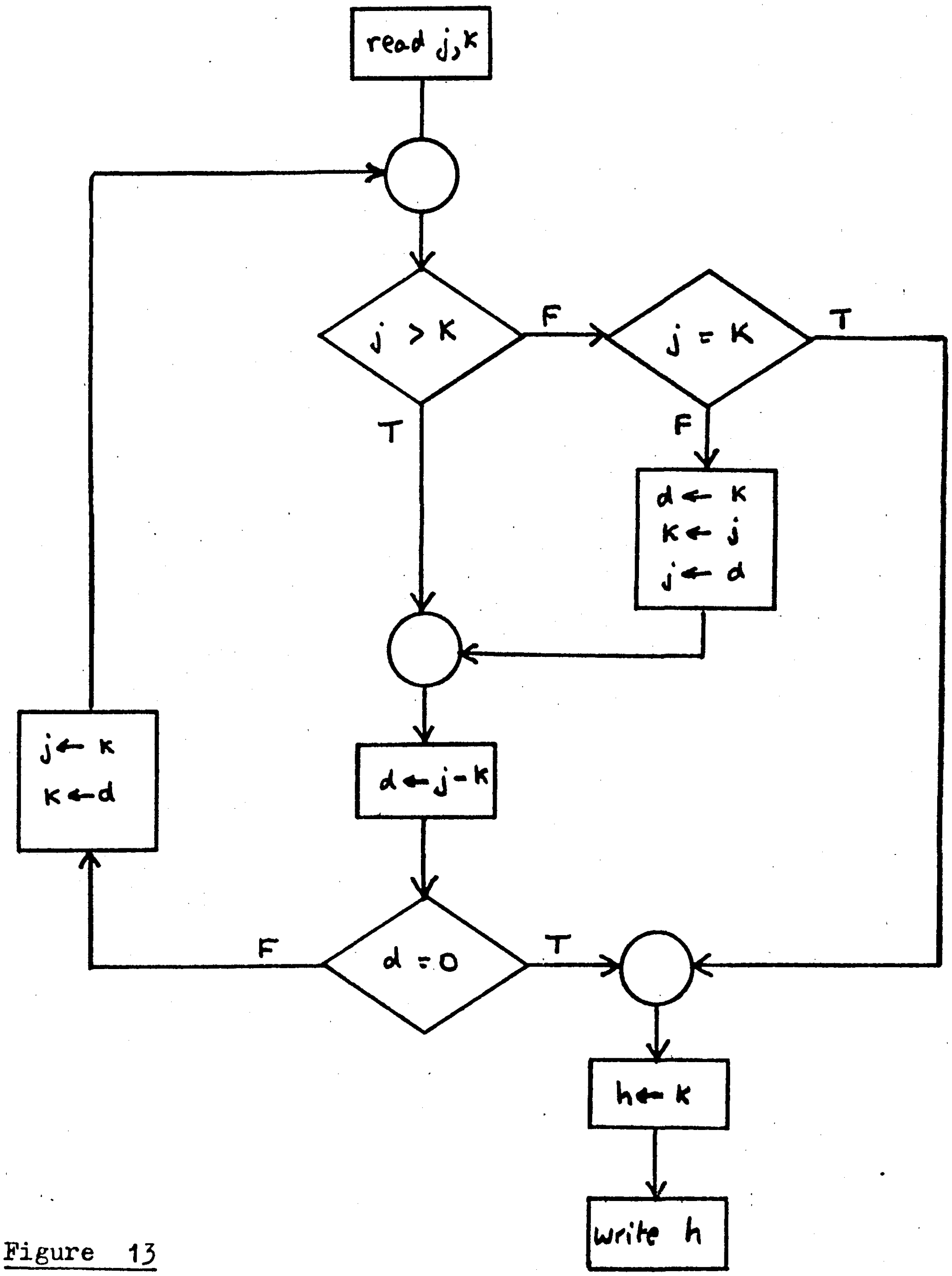


Figure 13

Algorithm HCF (Highest Common Factor) Given two positive integers j and k, find their highest common factor which is the largest positive integer, h, which divides both j and k.

```
step 1  read j, k
step 2  if j > k goto step 5
        else do step 3 od fi
step 3 if j = k goto step 8
       else do step 4 d←k; k←j; j←d od fi
step 5  d←j - k
step 6  if d = 0 goto step 8
        else do step 7  j←k; k←d; goto step 2 od fi
step 8  h←k
step 9  write h
```

Noting that the predicates in steps 2 and 3 may be implemented in Fortran by a three-branch conditional statement, this algorithm may be coded as:

```
  FUNCTION  IHCF(JJ,KK)
  J=JJ
  K=KK
1 IF(J-K)2,4,3
2 D=K
  K=J
  J=D
3 D=J-K
  IF(D.EQ.0)GOTO 4
  J=K
  K=D
  GO TO 1
4 IHCF=K
  RETURN
  END
```

This revision of a function sub-program illustrates that by recording the stages of top-down programming, it becomes easy to make modifications.

The Automatic Design of Experiments

Some Practical Algorithms

CHAPTER TWO

CHOICE OF EXPERIMENTAL CLASSES

The objectives of an experiment can usually be stated in terms of a mathematical model whose parameters are to be estimated. The best experimental design is that set of combinations of values of the control or independent variables which will permit the estimation of those parameters with greatest precision, with least bias, and within allowable cost limitations. A further criterion is expressed in terms of the use to which the fitted model will be put: the design should lead to the estimation of parameters such that the model may be used to predict values of the dependent variables with the greatest possible precision and the least bias, in a pre-specified region of the independent variables.

These criteria are not apparent in the works of Fisher (1925, 1935) whose major objective of experimentation was to test comparisons between treatments. It is felt, however, that Fisher tended to lay undue emphasis on the importance of formal tests of significance in experimental work. In part this emphasis on tests of significance is attributable to the way in which the subject developed in the agricultural and biological sciences, and to the fact that in the simpler types of experiment the treatment means are always efficient estimates. The emphasis on significance tests has had unfortunate consequences, both at a practical level and in theoretical work. Too much effort has been devoted to the investigation of minor points of little real importance. This has resulted in a proliferation of alternative methods of analysis, hedged about with restrictions and qualifications, to the confusion of the practical worker.

In 'Statistical Methods for Research Workers' Fisher actually encouraged the statistician to look around for the test giving the highest significance! It is not surprising that physical scientists sometimes remark that they see little of relevance to their research in standard texts on experimental design and analysis (such as Fisher (1935), Cochran and Cox (1950), and Kempthorne (1952)).

The developing complexity of physical research has called for a different approach to experimental design based upon the estimation of effects rather than upon tests of the significance of their comparisons. Indeed, effects of treatments can no longer be estimated simply, because we are now faced with multi-parameter mathematical models which call for a more subtle approach: usually least squares regression analysis and sometimes with ingenious coding of the variables. Furthermore, the research worker usually knows that these effects, as expressed by parameters or coefficients of the model, exist and what he needs is an efficient estimate of the parameters and reasonably accurate estimates of their errors.

Two types of variable can enter a model: qualitative and quantitative. It may be argued that quantitative variables should be further sub-divided into continuous quantitative and discrete quantitative. For example, in making a cake one might have any continuous measure of sugar or fat, but discretely only one, two, three or four eggs. However, in reality continuous variables are usually measured and controlled in discrete steps. Cooks would not specify sugar more precisely than to the nearest half ounce; steelmakers would not specify carbon content more precisely than the nearest 0.01 per cent.

In industrial research, where the major objective is usually the optimisation of a physical property or the cost or yield of a process, this dependent variable may be represented as the response surface in the space of the independent or control variables. In many cases, the experimenter has sufficient knowledge of his process to know, not only that effects exist, but that he is close enough to the optimum he seeks to be able to assume a response surface that is quadratic in the independent variables.

This situation is so common that it was decided for the present to limit the development of algorithms for the design of continuous variable experiments to those situations which could be represented by quadratic models.

Industrial laboratories frequently arrange experiments based entirely on qualit ative variables for which there is no prior justification for ordering. None of the variables can therefore be coded so as to be analogous to discrete quantitative variables. Such an experiment may be to assess the effects on a chemical estimation of: different laboratories; different apparatuses; different operators; different preparation, cleaning or storage methods; different sources of materials; or different types of atmosphere. Such variables are usually called factors and their values are designate levels. The experimental planners are often faced with the major difficulty that if they wish to examine more than two or three factors, each with several levels, simple multiplication shows that the number of observations to be made is more than is practically possible, limited perhaps by cost, time, and available materials. This problem has therefore been studied and algorithms have been developed to produce fractions of multi-level factorial experiments.

This study then is narrowed to an examination of methods for designing experiments to fit quadratic models in quantitative variables (chapter four) and for designing experiments in qualitative variables (chapters six and seven). Mixed designs, that is designs to deal with independent variables that are both qualitative and quantitative, are mentioned in chapter eight as a subject for further development.

There is, however, a class of experimental design which can be used as a basis for generating both of these other types of design. This is the two-level factorial, or more particularly, the fractional two-level factorial. As will be described in later chapters, the first stage in generating either of the two major designs will be the generation of a fractional two-level factorial. This intersection is illustrated.

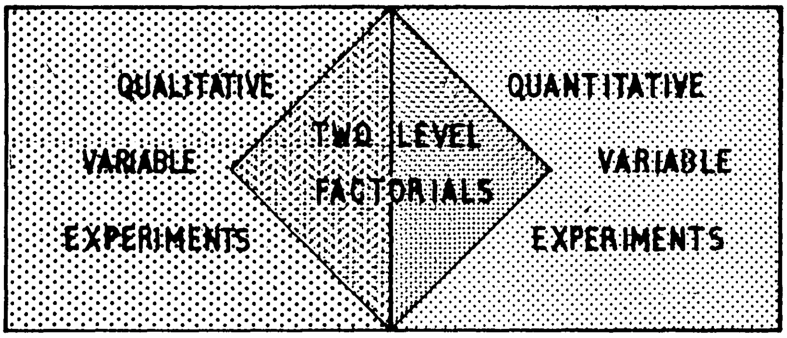


Figure 14

Box and Hunter (1961) make the point succinctly: 'A full 2^k factorial design requires all combinations of two versions of each of k variables. If a variable is continuous, the two versions become the high and low levels of that variable. If a variable is qualitative the two versions correspond to two types, sometimes the presence and absence of the variable.'

There is a further advantage in including the fractional two-level factorial in this study: it is sufficiently simple in concept to have acquired an almost universal adoption among physical, chemical, and metallurgical research workers. They have been familiar with it for some years, due largely to writers like Davies (1954), Duckworth (1968), and Mendenhall (1969). Yet these research workers still have problems and the most frequent is that of generating the best fraction of a factorial to suit the circumstances.

Accordingly, the next chapter of this thesis deals with algorithms for generating fractional two-level experimental designs. Subsequent chapters deal with augmenting these fractional designs to fit quadratic models and with using two level fractional factorials as the start of the procedure for designing asymmetric multi-level fractional factorials. Figure 15 is a simple flowchart relating these procedures.

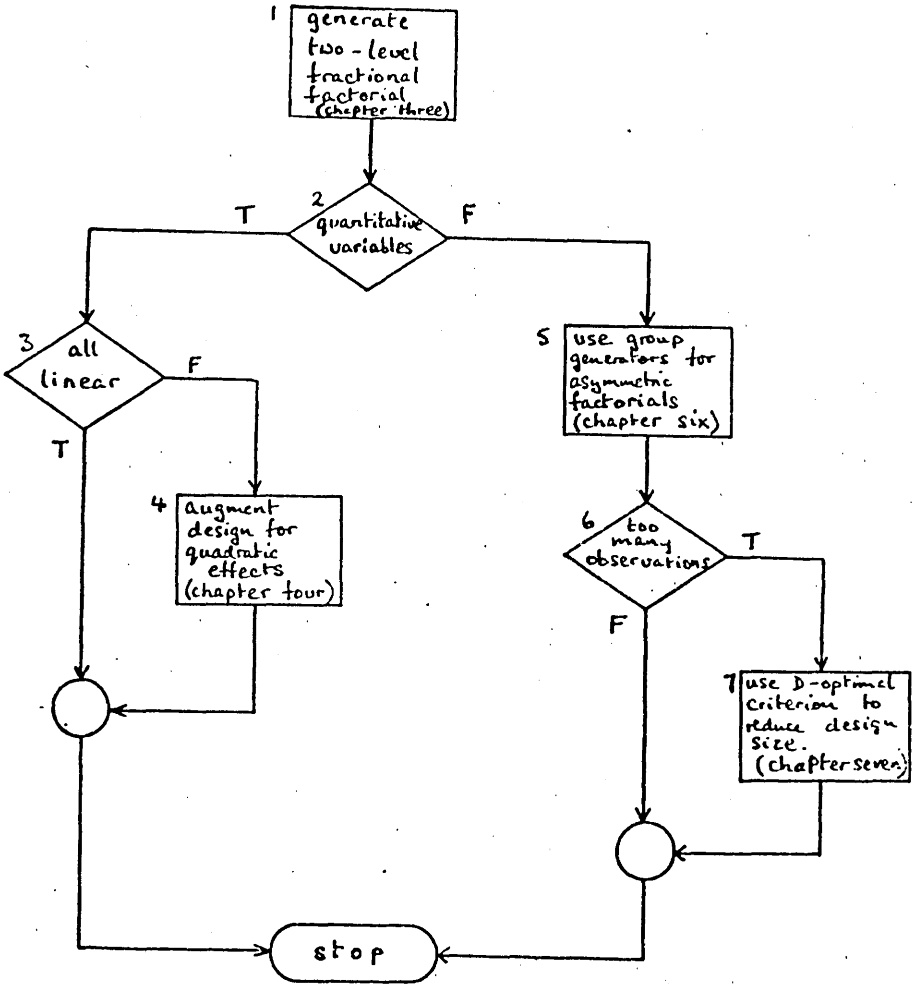

1 generate two-level fractional factorial (chapter three)
2 quantitative variables
T
F
3 all linear
F
T
4 augment design for quadratic effects (chapter four)
5 use group generators for asymmetric factorials (chapter six)
6 too many observations
T
F
7 use D-optimal criterion to reduce design size. (chapter seven)
stop

Figure 15

The first function (box 1) in the flowchart of figure 15 is the generation of a two-level fractional factorial design using the procedure to be developed in chapter three. This follows from the argument that whether the independent variables are qualitative or quantitative, the fractional two-level design will form a base on which the more complex designs will be built.

If the variables are quantitative (box 2) and if only linear main effects and interactions are expected (box 3) then the two-level fractional factorial design satisfies the requirements.

However, if quadratic effects are expected for any of the variables, then the design must be augmented with extra observation points to allow the estimation of those quadratic terms (box 4). The algorithms for augmentation are developed in chapter four.

If the variables are all qualitative but any of them has more than two levels, then the design is classed as an asymmetric factorial (box 5). A procedure for generating balanced fractional asymmetric factorial designs is developed in chapter six.

Sometimes a balanced fractional asymmetric design has more observations than is economically acceptable by the experimenters and is also grossly over-determined (box 6). If this is so, then the criterion of balance is abandoned and a subset of the observations in the balanced fraction is selected using the criterion of D-optimality (box 7). The algorithms for this are developed in chapter seven.

A natural extension of this work would be the development of algorithms for designing mixed experiments: those with both qualitative and quantitative variables. In chapter eight I suggest this among future work to be tackled.

The Automatic Design of Experiments

Some Practical Algorithms

CHAPTER THREE

TWO-LEVEL FACTORIALS

1 Background

The early papers by Fisher (1926 et seq) and Yates (1933 et seq) stimulated a steady flow of papers on both the design and analysis of factorial experiments. Their ability to be divided into blocks by confounding high order interactions with block effects appealed particularly to the agricultural statisticians and they were helped by Barnard (1936) who enumerated a selection of confounded arrangements. These enabled the research worker to choose a design by inspection but they did not give him a uniform procedure for ensuring that his choice would provide the conditions for estimating all the required coefficients of the model to be fitted. Indeed there seems to be little in the literature before the 1950's which discussed explicit mathematical models when considering experimental design.

Finney (1948) drew attention to this in a paper which described the estimation and interpretation of main effects and interactions. He commented: "few things betray the inexperienced statistician more readily than a triumphant presentation of an elaborate analysis of variance table coupled with an almost complete neglect of treatment means."

One of Finney's major contributions was his clear exposition (1945 and 1946) of the relationship between block confounding and fractions of two level factorials. In these papers he explains the notation introduced by Fisher and Yates and which, through common use, has been accepted generally as standard. The notation is that both factors and their effects are represented by capital letters; the high and low levels of the factors are represented by the presence or absence, respectively of lower case letters. Thus if there are three factors, each with two levels, the *factors* would be named A, B, and C. The *effects* of these factors would also be labelled A, B, and C: the first order interaction effects would be labelled AB, AC, and BC; and the second order interaction effect would be labelled ABC. An objective of the experiment would be to estimate these effects together with the mean effect which is denoted by I. Combinations of lower case letters denote firstly *experimental design points*. For example: ac represents the observation point at which factors A and C are both at their high levels and factor B is at its low level. The case where all factors are at their low levels is denoted by (1). This lower case notation is also used, without confusion, to represent the *observed values of the dependent variable* at the corresponding observation points.

As well as a standard notation, there is a *standard order* for listing observation points and factorial effects. The standard order is clear from the following example with three factors:

Observation points	Factorial effects
(1)	I
a	A
b	B
ab	AB
c	C
ac	AC
bc	BC
abc	ABC

This standard notational order will be shown to have value in the next section when the design algorithms are developed.

In the same paper, Finney stated: "In planning a 2^n experiment, using only 2^p treatment combinations in a $1/2^{n-p}$ replicate, the first step is to select a suitable alias subgroup of order 2^{n-p} and then to determine the complete orthogonal sub-group of this as the set of treatment combinations."

The "alias sub-group" to which he refers is also known as the "set of defining contrasts", which will be described later, and their selection constitutes the outstanding problem in designing fractional two-level factorial experiments. Finney gave no formal procedure for choosing the defining contrasts. In his example he arbitrarily chose some with high resolution (interactions between more than three factors) and then tested that they would not lead to aliasing between main effects and low order interaction effects.

He did, however, describe a formal procedure for developing a fractional design once a suitable set of defining contrasts had been chosen. This procedure followed the demonstration by Fisher (1943) of the connection between confounding and the theory of Abelian groups. This connection is shown to be of value in the next section of this chapter when the implementation of the design algorithms as computer programs is described. It is shown to have further value in the algorithms for designing fractional mixed multi-level factorials which are described in chapter seven.

Kempthorne (1947) offered an alternative notation for the design points, using ones and zeros. He also described factors by lower case sub-indexed x's: "If the factors are x_1, x_2, ... , x_n and they take n mutually orthogonal axes y_1 to y_n, then the point (000...0) represents the control treatment(with all the factors at the low levels), the point (100...0) has x_1 at its high level and all other factors at their low levels, and so on." Kempthorne's notation is known as a bit notation in computer terms and this is also shown to be of value when implementing the design algorithms as computer programs.

Kempthorne also illustrated the procedure for designing fractions once a suitable alias sub-group or set of defining contrasts had been chosen; but, he admitted, "no simple method has been found of enumerating such groups."

Box and Hunter (1961) gave a thorough treatment of the notations, design, and analysis of fractional factorial experiments and they suggested a procedure for choosing a set of defining contrasts in a less than wholly arbitrary way. They defined the resolution of a design as the smallest number of factors represented in the design's set of defining contrasts. The resolution of a design would influence the degree of confounding of effects, when they came to be estimated from the observations, as follows:

In designs of resolution 3, no main effect would be confounded with any other main effect, but main effects would be confounded with two-factor interactions.

In designs of resolution 4, no main effect would be confounded with any other main effect or any two-factor interaction, but two-factor interactions would be confounded with each other.

In designs of resolution 5, no main effect or two-factor interaction would be confounded with any other main effect or two-factor interaction, but two-factor interactions would be confounded with three-factor interactions.

With these resolutions in mind they suggested the following procedure for choosing a suitable set of defining contrasts: alias the main effects and required interactions with other interactions assumed to have insignificant effects; this yields possible defining contrasts which must be multiplied together to give the complete set of possible defining contrasts which must then be checked to see if they lead to any undesirable aliasing.

The procedure of Box and Hunter was not, however, a direct path from an explicit statement of the model to be estimated to the choic of a suitable set of defining contrasts. Nor was the similar procedure of Whitwell and Morbey (1961) who dealt specifically with designs of resolution five since these would certainly lead to the estimation of all first order interactions as well as main effects. Their argument rested on the assumption that all first order interactions were needed. They did not consider questioning the experimenter's model to discover if there could be any a priori discarding of first order interactions.

Addelman (1963) reviewed known techniques for constructing fractional designs. He commented: "The crucial part of the specification of a fractional replicate plan is the choice of the defining or identity relationship. One should always attempt to choose interactions for the identity relationship in such a way that those interactions that are completely confounded with the effects or interactions that one wishes to estimate are negligible"

The advocacy of authors such as Cochran and Cox (1950), Kempthorne (1952), Brownlee (1953), Davies (1954), Duckworth (1968), and Mendenhall (1969), led to the two-level factorial becoming the most commonly used type of experimental design in industrial laboratories. Research workers generally and easily appreciated that fractions of these designs would achieve economies in both money and time spent on investigations. The advocates warned, however, that care must be exercised in the choice of these designs so as to avoid the aliasing of main effects and important interactions.

Sometimes, still, this point is ignored by the research worker who, without consultation, uses any available fraction without consideration of the penalty to be paid on analysis. Subsequently, the statistician is expected to extract non-existent information from the experimental results.

The usual procedure is to refer to a standard textbook and try to pick a published design to meet the experimental needs. If a suitable design is not found, a research worker with some understanding of the subject will try a few arbitrary sets of defining contrasts, generating aliasing matrices until a suitable design is found. This introduces an undesirable element of arbitrariness. A paraphrase of the procedure recommended in many texts is:

"First choose a suitable set of defining contrasts. Secondly use these defining contrasts to generate an aliasing matrix and check if all the main effects and interactions that need to be estimated can be estimated without being aliased with any others. If this check fails, start again. If it passes: thirdly use the defining contrasts to generate the fractional design."

The procedure given for the final stage is satisfactory, but that for the first is no more than guesswork. The outstanding problem, that has not previously been solved, is to establish a logical and easy procedure to select a set of defining contrasts that define an aliasing matrix in which the experimental requirements are not aliased.

This is not a trivial problem . It is not uncommon for a research worker to spend several days searching for a suitable set of defining contrasts by trial and error. In view of this, and in view of the historical awareness of the importance of the problem, the simplicity of the solution comes as a surprise. The purpose of this chapter is to explain the solution which has been published in a brief form, Greenfield (1976). After publication, Franklin (1977) identified a case where my algorithm gave an incorrect answer and I immediately submitted a modification for publication (1978). Meanwhile, Franklin and Bailey (1977) developed and published an alternative algorithm which has the added advantage, for agricultural work in which their main interest lies, that it can lead to division of a fractional design into blocks for further confounding. However, as I had already successfully implemented my algorithm and as their extra feature is generally of no particular advantage in an industrial laboratory, I shall not at this stage change my intention to present my original algorithm, duly modified slightly, in the rest of this chapter.

2 Algorithms

The object of the algorithms to be developed in this section is to design a fractional two-level factorial experiment that will admit the estimation ofthe coefficients of a prior stated linear relationship between a dependent variable and a set of independent variables. The latter may be described as the main effects and some of the interactions of a set of factors. In the earliest version of these algorithms interactions were restricted to first order: those between two factors. This was supported by Box and Hunter (1961) who wrote: "With continuous variables it is reasonable to expect the response to vary smoothly. With qualitative variables certain aspects of similarity may be expected in the responses at the different versions. . . . In the conditions of smoothness and similarity commonly encountered, three factor and multi-factor interaction effects are often negligible." However, in subsequent applications there have been several cases where second order interactions, those between three factors, have anticipated for physical reasons. Thus the algorithms must allow the inclusion of interactions of any order. This is a marked departure from the procedures for producing designs of resolution 3, 4 or 5.

The full algorithm may be divided into three main steps:

1. Enter the requirements set: those main effects and interactions which are required to be estimated;
2. Determine the fraction size and find the defining contrasts;
3. Generate and print the design.

Since the second step of the full algorithm has historically been the most taxing, the algorithmic solution will be developed in detail. Subsequently, for the sake of brevity, the first and third steps will simply be expressed in algorithmic form, rather than being developed detail by detail.

The problem is to find a set of defining contrasts which will define a fractional design that will allow the unaliased estimation of all the elements of the requirements set. The solution to the problem is to generate the defining contrasts and the aliasing matrix together, instead of first one and then the other. Effectively this is done by a tree search which is most easily described in terms of an example. The algorithm will first be expressed in straightforward English interspersed with the steps applied to an example. It will then be developed more formally.

Consider a 2^5 experiment in which the variables are labelled A,B,C,D,E. What is required is the smallest possible balanced fractional design that can be used to estimate each of the main effects and also the effects of the first-order interactions AB and AE, assuming that the effects of all other interactions on the dependent variable are known in advance to be negligible. The requirements set is: A,B,AB,C,D,E,AE . The design must also estimate the mean, so in this case there must be a minimum of eight observations.

Algorithm DEFCON (DEFining CONtrasts)

Step 1. Find m such that $2^{n-m} \geqslant 1 + n + p$

where n is the number of factors, p is the number of interactions in the requirements set. If the procedure shows that there is no unaliased design of the size determined by the above expression, it continues to a fraction of double the size. That is m is decreased by one. In the example, the first value of m is 2, so the smallest fraction likely to provide a suitable design is a quarter.

Step 2. Find the (n-m) majority factors in the requirements set. These are the factors that occur most frequently in the requirements set of main effects and first order interactions.

Example: The three majority factors in the requirements set are A, B, E.

Step 3. Write the first column of the aliasing matrix in terms of the (n-m) majority factors. This has 2^{n-m} elements (eight in the example). Mark with an asterisk those elements common to this column and to the requirements set.

Example:

First column of the aliasing matrix:	I	The requirements set:
	A*	(A*,B*,AB*,C,D,E*,AE*)
	B*	
	AB*	
	E*	
	AE*	
	BE	
	ABE	

Step 4. Generate the first defining contrast by taking the product (modulo 2) of the last available element in the first column and the last available element in the requirements set.

Example: ABE x D = ABDE

Step 5. Use this defining contrast to generate the next column of the aliasing matrix. Check at the same time if any of the required effects have become aliased with those already marked in the first column. If not, mark those that have been introduced.

Example:

First column	Second column
I	ABDE
A*	BDE
B*	ADE
AB*	DE
E*	ABD
AE*	BD
BE	AD
ABE	D*

The requirements set is:
(A*,B*,AB*,E*,AE*,C, D*)

between required effects

Step 6. If any aliasing/occurred in step 5, return to step 4, generating a new first defining contrast by taking the product (modulo 2) of the next from last available element in the first column and the last available element in the requirements set. In the present example this does not occur.

Step 7. When there are no more available elements in the first column, and if the requirements have not all been met, decrease m by one and return to step 3.

Step 8. Generate the next defining contrast as in step 4.

Example: BE x C = BCE

As will be explained later, this leads automatically to the third defining contrast as the product of the first and the second.

Example: ABDE x BCE = ACD

Step 9. Generate the full aliasing matrix, mark with asterisks and check for aliasing as in step 5.

Example:

Column 1	Column 2	Column 3	Column 4
I	ABDE	BCE	ACD
A*	BDE	ABCE	CD
B*	ADE	CE	ABCD
AB*	DE	ACE	BCD
E*	ABD	BC	ACDE
AE*	BD	ABC	CDE
BE	AD	C*	ABCDE
ABE	D*	AC	BCDE

The requirements set is (A*,B*,AB*,E*,AE*,C*,D*)

The algorithm described above is sufficient to be followed manually, as several correspondents testified after publication, Greenfield (1976). The algorithm must however be expressed in greater detail if anyone is going to use it to develop a computer program. Some special characteristics of factorials also need to be explained, together with some computing ploys, before the more detailed algorithm can be stated.

The connection between confounding and Abelian groups was described by Fisher (1943). This connection becomes more notable when a binary digit coding is adopted for both the treatment combinations (design points) and main effects and interactions, especially since digital computers use binary integers. In the code adopted here, the binary digits, or bits, are read and counted from right to left. Thus:

00000010 = B or b (bit 2 is described as 'up','set' or 'one')

00000101 = AC or ac (bits 1 and 3 are set)

Since this coding will be used in the programs but alphabetic coding is preferred for human reading, an algorithm will be needed to convert from binary code to aphabetic code. This will be described later as part of algorithm **ALMAT** (print **AL**iasing **MAT**rix).

The binary code permits the direct generation of two-level factorials by counting upwards from zero as follows:

Denary count	Binary code	Treatment combinations
0	00000	(1)
1	00001	a
2	00010	b
3	00011	ab
4	00100	c
5	00101	ac
6	00110	bc
7	00111	abc
8	01000	d
9	01001	ad
10	01010	bd
11	01011	abd
12	01100	cd
.	.	.

etcetera

The table stops at the denary count of $2^n - 1$, where n is the number of factors.

The product of any two elements (modulo 2) when using the binary code is seen by an example:

ABDE x BCE = ACD is equivalent to 11011 x 10110 = 01101

It is convenient that this can be achieved by the use of the exclusive-OR operator (referred to in future simply as <u>eor</u>) which is defined by the following truth table:

A	B	eor (A,B)
0	0	0
1	0	1
0	1	1
1	1	0

As Fisher noted, the full factorial design and the full aliasing matrix (see <u>example</u> after <u>step 9</u> above) are both groups of order 2^n under the product (modulo 2) operator. Similarly if a full design on n factors is expressed as the set of integers from 0 to $2^n - 1$ in binary code, then the set becomes a group under the exclusive-OR operator.

If D = the set $(0,1, \ldots , 2^n-1)$, then for all $x \in D$ and for all $y \in D$ there is a $z \in D$ such that eor(x,y) = z.

The group's identity element is 0, since for all $x \in D$

eor(0,x) = eor(x,0) = x

Also every element x has an inverse which is itself:

eor(x,x) = 0 (the identity)

As an example let n = 2,

then D = ((1),a,b,ab) in alphabetic code

or D = (0,1,2,3) in denary code

or D = (00,01,10,11) in binary code

Application of the operator to all ten pairs is seen to always yield members of D:

eor(00,00) = 00

eor(00,01) = 01

eor(01,10) = 11

etcetera

It is also useful to note that as well as the full aliasing matrix being a group, using the binary notation and the exclusive-OR operator:

- the first column of the aliasing matrix is a sub-group
- the first row of the aliasing matrix (which is the full set of defining contrasts including the identity) is a sub-group
- the fractional design that will be derived using the defining contrasts is a sub-group of the full design group.

One difficulty that has been met in implementing these algorithms on various computers is that in standard Fortran the exclusive-OR operator (and other logical operators) can be used only with logical operands. However, on some scientific computers, such as the IBM 1130 and 1800 and the GA SPC16, logical operators can also be used with integer operands to give an integer result. The integer operands are considered to be in their binary representation as described here. The result reflects whether corresponding bits in the two operands are set or not.

The related types of operator will be distinguished here by different notations, acting on logical operands P and Q and on integer operands I and J:

Logical operators: P.xor.Q P.or.Q P.and.Q
Binary integer operators: eor(I,J) or(I,J) and(I,J)

There should be no objection to using these binary integer operators. Even if they are not provided with the high level language function library, any competent programmer should be able to write them using a machine code or assembler language, and add them to the function library.

A further useful feature of the group property described is that each group, or sub-group, can be generated by a sub-set of the elements of the group or sub-group. Furthermore, this subset can be identified a simple rule: if the elements of the group are expressed in standard order (counting from 0 to 2^n-1 in the case of the full design or its equivalent order for a sub-group as illustrated by the first column of the aliasing matrix in the earlier example) or in the order in which they are created (as in the first row of the aliasing matrix: the defining contrasts), then:

the generators are those elements in the positions with order numbers 2^r+1, where r is any integer 0, 1, 2, . .

As an example, consider the full factorial with n = 3. The elements of the factorial will be expressed alphabetically for reading clarity. The order numbers are set below. The generators are marked with asterisks.

(1)	a	b	ab	c	ac	bc	abc
1	2	3	4	5	6	7	8
	*	*		*			

In this case the answer is obvious because the generators are those elements with single letters. However the rule may not seem so obvious when generating the set of defining contrasts and in the algorithm for doing this the rule is of particular value. It is used as follows: when the elements of a group or sub-group are developed from left to right (in the above example), every time a generator is created the remaining elements of the group up to, but not including, the next generator can be created by taking the product of the new generator with each of its preceding elements in turn.

In the example above, the first element to be written is (1). The first generator to occur is a. Application of the prodedure described creates the single element a.
The second generator to occur is b. Application of the procedure creates the elements b and ab.
The third generator to occur is c. The procedure creates the elements c, ac, bc, abc.

Returning now to the earlier example: consider the first row of the aliasing matrix as a sub-group.

The first element to be written was the identity I.

The first defining contrast to be created (by taking the product ABE x D) was ABDE. This is the first generator and it creates the element of the sub-group ABDE.

The second defining contrast to be created (by taking the product BE x C) was BCE. Application of the procedure described creates two new elements of the sub-group: BCE (which is BCE x I) and ACD (which is BCE x ABDE).

The rule for identifying a new generator is useful in the main algorithm in three ways: First, after the majority factors have been identified, it leads to the use of the above generating procedure for generating the first column of the aliasing matrix.

Second, when a new defining contrast is created it leads to the use of the above procedure for generating consequent defining contrasts.

Third, it helps in designing a marking, or flagging, system so that if aliasing is discovered after a defining contrast has been created the algorithm can backtrack to the previous generator defining contrast. The method and value of this use will become clearer as the algorithm unfolds. It may be noted here that a simple aid in back tracking is:

> if Y is the order number of the current generator (for example, if $r = 5$ then $Y = 2^r+1 = 33$) and if X is the order number of the previous generator, then to determine X it is simpler to write $X \leftarrow (Y-1)/2$ than to compute r from Y and then, by decrementing r, to compute X from 2^r+1.

In the earlier description of the general algorithm DEFCON, asterisks were used to the rows and the defining contrasts. Also backtracking, after discovering aliasing in step 9, was implied and not explicitly described. These two actions, marking and backtracking, are closely related and a more complex procedure than through the use of simple marks like asterisks has to be developed for a programmable algorithm. Discussion of the procedure and the development of the algorithm will be helped by now defining some of the variables to be used. Practical dimensions of arrays are denoted by (*n).

MV(I) = the Ith element of the requirements set (*32)
NV = the number of elements in MV
N = the number of two-level factors
M = the fraction index (the design would be a $1/2^M$ factorial)
K(I,J)= the I,Jth element of the aliasing matrix (*128 x 32)
NF = number of rows in the aliasing matrix
NI = number of columns in the aliasing matrix
KEST = defining contrast being tested for acceptance
JAK = column number of the defining contrast being tested
LNEW = column number of the current generator defining contrast
IN(J) = a marker for the Jth element of the requirements set (*32)
= 0 if accepted
= 100 if currently not being considered
= the value that LNEW had when the Jth element of the requirements set was tentatively accepted
IV(I) = a marker for the Ith row of the aliasing matrix (*128)
= 1 if the row has definitely been assigned
= 0 if the row has been tentatively assigned
= -1 if the row is still available
KR(I) = another marker for the Ith row of the aliasing matrix (*128)
= 0 if the row has definitely been assigned
= 100 if the row is still available
= the value that LNEW had when the Ith row of the aliasing matrix was tentatively assigned
KK(J) = a copy of the Jth element of the first row of the aliasing matrix: to save reference time when computing (*32)
MAJ(I)= the n factors expressed in majority order (*16)
NB(I) = a temporary array used in producing MAJ(I) (*16)

A function subprogram NEW(I) will be called to test if I has a value of $2^r + 1$, where r is any integer.

The above considerations and notations lead to the following more detailed expression of the algorithm:

```
Algorithm  DEFCON (DEFining CONtrasts)
Step 0  (initialise)  given the number of factors, N, the number
        of requirements, NV, and the set of requirements, MV(.),
        find the fraction index, M, such that $2^{N-M} \geq 1 + NV$,
        the number of columns in the aliasing matrix, NI, and
        the number of rows, NF.  Also determine the majority
        order of the factors, MAJ(.).

Step 10 Construct the first column ($JAK \leftarrow 1$) of the aliasing matrix
        using the N - M majority factors as generators and the
        function NEW.  For those elements of the column which
        equal some of the requirements, mark the rows each with
        two markers (IV(.) and KR(.)) and the corresponding
        requirements with one marker (IN(.)).

Step 20 (increment JAK to go to the next column of the aliasing
        matrix) $JAK \leftarrow JAK + 1$
        if  $JAK = 2^r + 1$  then  step 30  $LNEW \leftarrow JAK$, $LL \leftarrow 1$,
        and reset IV(.) row markers for which the KR(.) row markers $\geq$ LNEW
        back to -1 (still available),
        else  goto  step 70  fi

Step 40 (find LE, the first available requirement counting
        from the end of the set)
        if (the search is not successful) then (the allocation
        of requirements to rows is complete, but the aliasing
        matrix must be completed and checked, so:)
        goto  step 20  fi

Step 50 (find LO, the first available row counting back
        from the last row)
        if (the search is not successful)  $LO \leq 0$
        then (the search must backtrack to the previous level
        of LNEW, or if the present value of LNEW is 2 then
        the number of rows must be doubled and the number of
        columns halved)  goto  step 90  fi
```

```
Step 60   set the row marker KR(LO)←LNEW
          create the new defining contrast to be tested
          KEST← eor(LE,K(LO,1))
          goto  step 80

Step 70   create each new defining contrast to be tested in turn
          by eoring each existing defining contrast in turn up to,
          but not including,  the current generating contrast
          KK(LNEW)  with KK(LNEW)
          LL←LL + 1,     KEST← eor(KK(LL), KK(LNEW))

Step 80   (test for aliasing the defining contrast created in
          step 60 or step 70,  and set the markers IV(.) and IN(.)
          to testing values for matching rows and requirements:
          IV(.)←0;  IN(.)←LNEW
          if (the test is     successful)  then KK(JAK)←KEST
          (indicating that the defining contrast has successfully
          passed its  test and is assigned to the set of defining
          contrasts KK(.));   goto step 20
          else  goto step 120    fi

Step 90   (LO≤0 indicates that there is no available row,  so we
          must backtrack to the previous LNEW)
          if  LO = 0 (which indicates it is possible to backtrack)
          then  step 100  JAK←(LNEW - 1)/2 +1  (and reset KR(.)
          markers which are now equal to LNEW back to 100)
          else (LO = -1 implies that LNEW is already at its smallest
          value of 2 so backtracking is not possible,  therefore a
          design of the current fraction is not possible,  so
          double the number of rows and start again)
          step 110  NF←NF*2,  NI←NI/2,  goto step 10,  fi

Step 120  (the defining contrast KEST that has just been tested
               has failed the test so set all requirements markers
          IN(.)≥ LNEW back to 100)
          for  J←1 to NV
          if IN(J)≥ LNEW  then IN(J)←100  fi
          goto  step 50
```

Each of these major steps will now be expanded and also represented as a flowchart.

A problem in <u>step 0</u> is to find M such that $2^{N-M} \geqslant 1 + NV$. Putting NE = N-M, the apparent solution may be $NE \leftarrow ln(1+NV)/ln(2)$. However, since NE would be assigned the integer value of this expression, it would normally take on a value less than that needed. For example, if NV = 6, then the expression would be evaluated as 2.80 and NE would be assigned the integer value 2, whereas the value needed is 3. Addition of integer 1 would correct this, but then there would be cases when NE would be assigned too great a value. For example, if NV = 15, then the expression would be evaluated as 4 and NE would be assigned 5. A compromise which avoids both these errors in assigning integer values is $NE \leftarrow 1 + ln(NV)/ln(2)$.

Another problem in <u>step 0</u> is to determine the majority order of the factors MAJ(.). It is assumed here that elements of the requirements set, MV(.), are expressed in the bit notation described earlier. The problem is to count the incidence of each bit in the requirements set and then to order them. For this purpose an extra array NB(.) is used. Also a function ITEST(I,J) is assumed to be available to test if the Jth bit from the right end of integer I is set or not, returning the value 1 if the bit is set.

In the earlier example the requirements set was (A,B,AB,C,D,E,AE). This would lead to NB(1) = 3, NB(2) = 2, NB(3) = 1, NB(4) = 1, NB(5) = 2, representing the occurrences of the bits corresponding to letters A,B,C,D,E. This would in turn lead, using bit notation, to: MAJ(1) = 0001, MAJ(2) = 00010, MAJ(3) = 10000, MAJ(4) = 00100, MAJ(5) = 01000.

A further function, IONBT(I,J), is assumed to be available which will return the initial value of I with the Jth bit set whether or not it was initially set.

Step 0 now becomes:

```
Step 0    (initialise)  enter N, NV, MV(.)
Step 1    NE←1 + [ln(NV)/ln(2)]  (square brackets indicating
          integer value);  M←N - NE;  NI← 2**M;  NF←2**NE
Step 2    for I←1 to N do step 3 NB(I)←0;  MAJ(I)←0  od
Step 4    for I←1 to NV do step 5  od
Step 5    for J←1 to N  do step 6  od
Step 6    if ITEST(MV(I), J)  =  1  then set NB(J)←NB(J) + 1  fi
Step 7    for I←1 to N  do step 8;  step 801  od
Step 8    MAX←0;  for J←1 to N do step 9 od
Step 801  MAJ(I)←IONBT(MAJ(I), JM);  NB(JM)←0
Step 9    if NB(J) > MAX  then  MAX←NB(J);  JM←J  fi
```

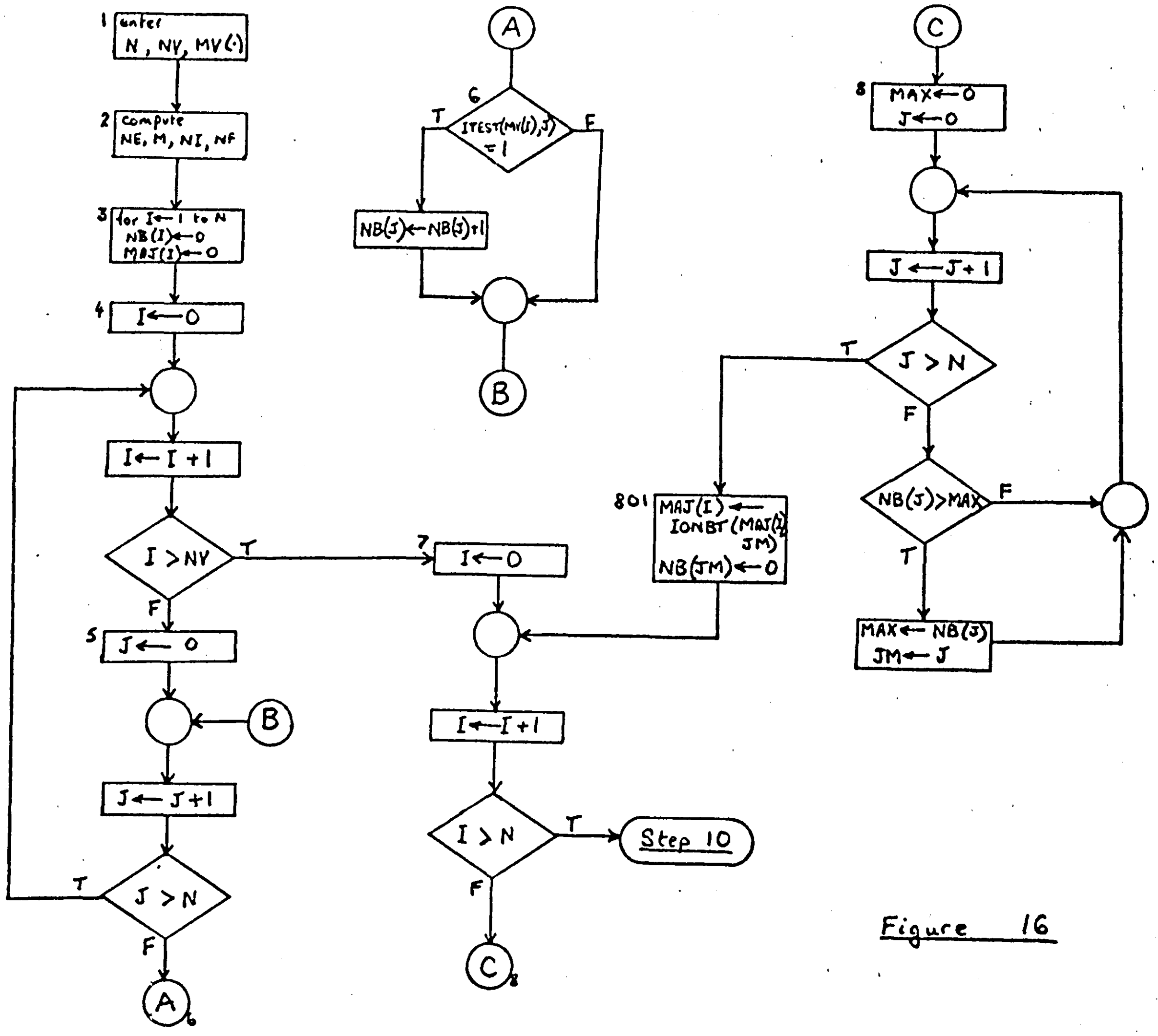


Figure 16

```
Step 10  (construct first column of aliasing matrix and set
markers)  JAK←1; K(1,1)←0;  MM←0;  for I←1 to NV do IN(I)←100 od
Step 11  for I←2 to NF do step 12;  step 15  od
         Step 12  if NEW(I) = 1 then step 13 else step 14 fi
         Step 13  LL←1; MM←MM + 1; K(I,1)←MAJ(MM)
         Step 14  LI←LL + 1;  K(I,1)←eor(K(LL,1),MAJ(MM))
         Step 15  for J←1 to NV do step 16 od
         Step 16  if K(I,1) = MV(J) then do step 17 od fi
         Step 17  IV(I)←1; KR(I)←0;  IN(J)←0
         Step 18  IV(I)←-1;  KR(I)←100
```

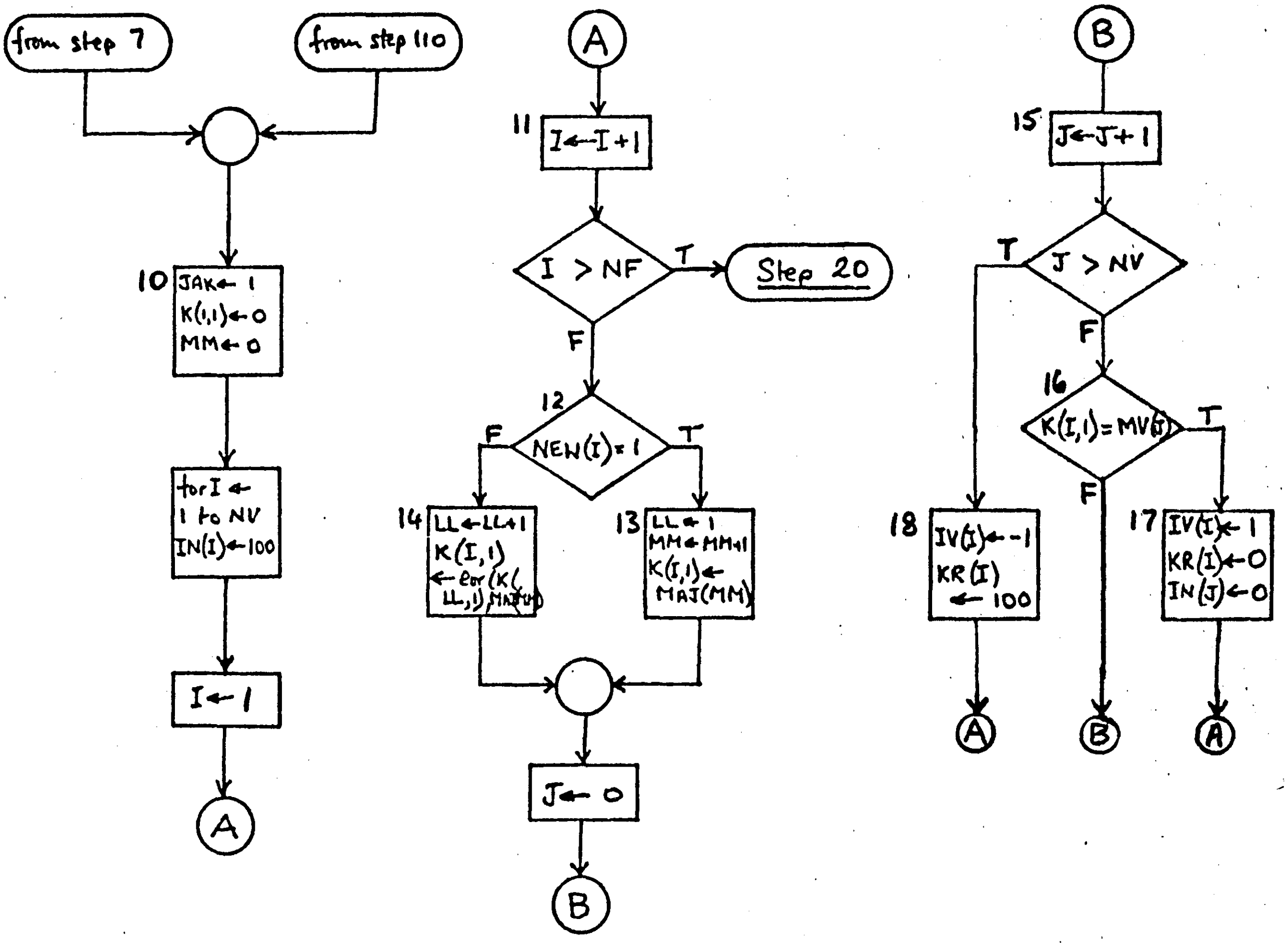


Figure 17

```
Step 20   (will next defining contrast be a generator?
           if so,  reset row markers)
           JAK ← JAK + 1
Step 21    if  JAK > NI  then stop  (all defining contrasts found) fi
Step 22    if  NEW(JAK) = 1  then do step 30 od
                             else goto step 70  fi
Step 30    LNEW ← JAK;  LL ← 1
Step 31    for  I ← 2  to  NF  do step 32 od
Step 32    if  KR(I) ≥ LNEW  then  set  IV(I) ← -1  fi
```

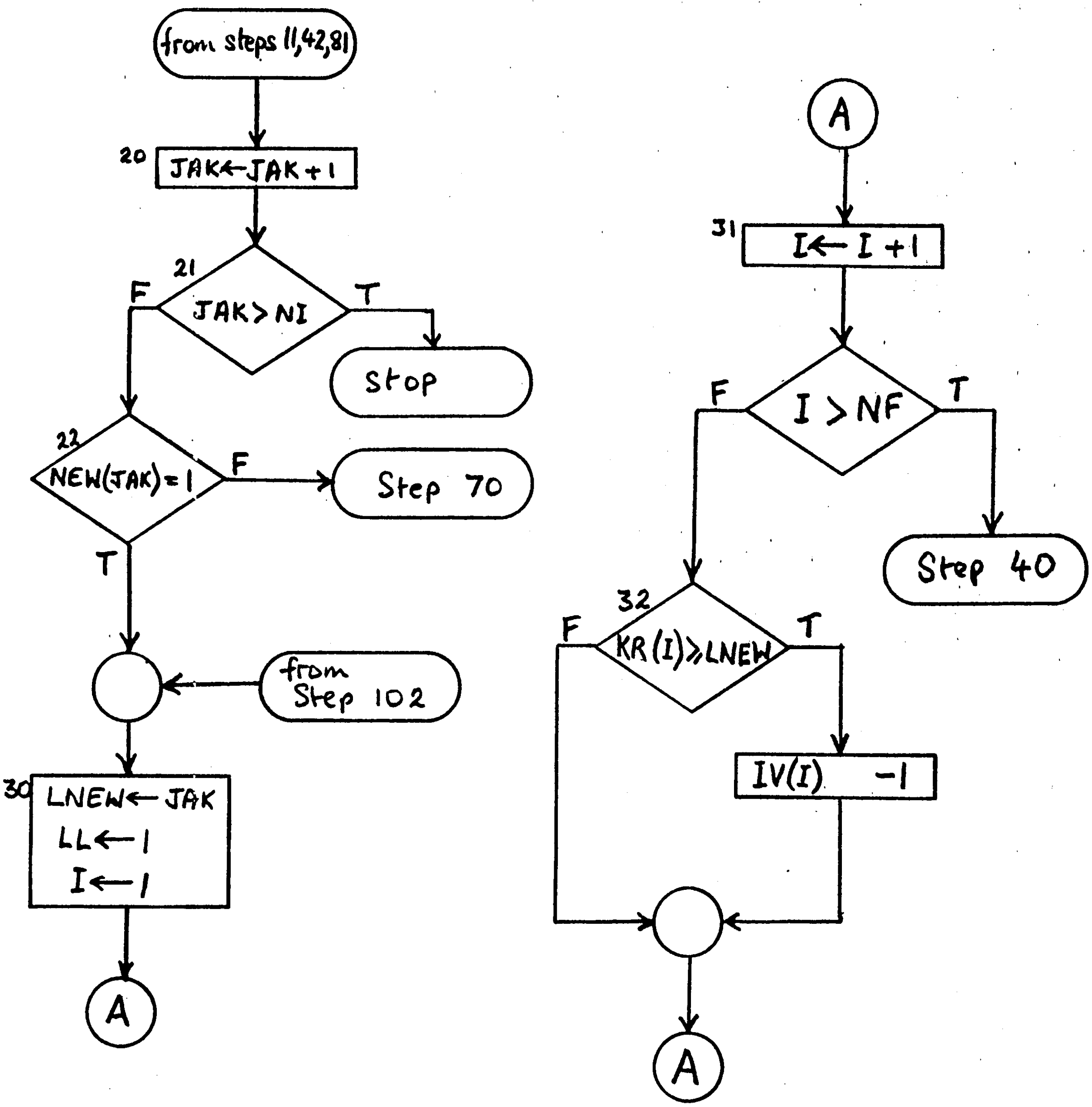


Figure 18

```
Step 40   (find the first available requirement counting from
          the end of the set;  if none,  return to step 20)
          for  I←1 to NV  do step 41;  step 42  od
Step 41   set  J←NV - I + 1
Step 42   if  IN(J) ≥ LNEW then set LE←MV(J); goto step 50 fi
          (at some stage, while a previous KEST was being tested
           in step 80 at the same level of LNEW,  IN(J) may have
          been set equal to LNEW,  so the test in step 42 must
          be ≥ and not just > )
Step 43   goto step 20
```

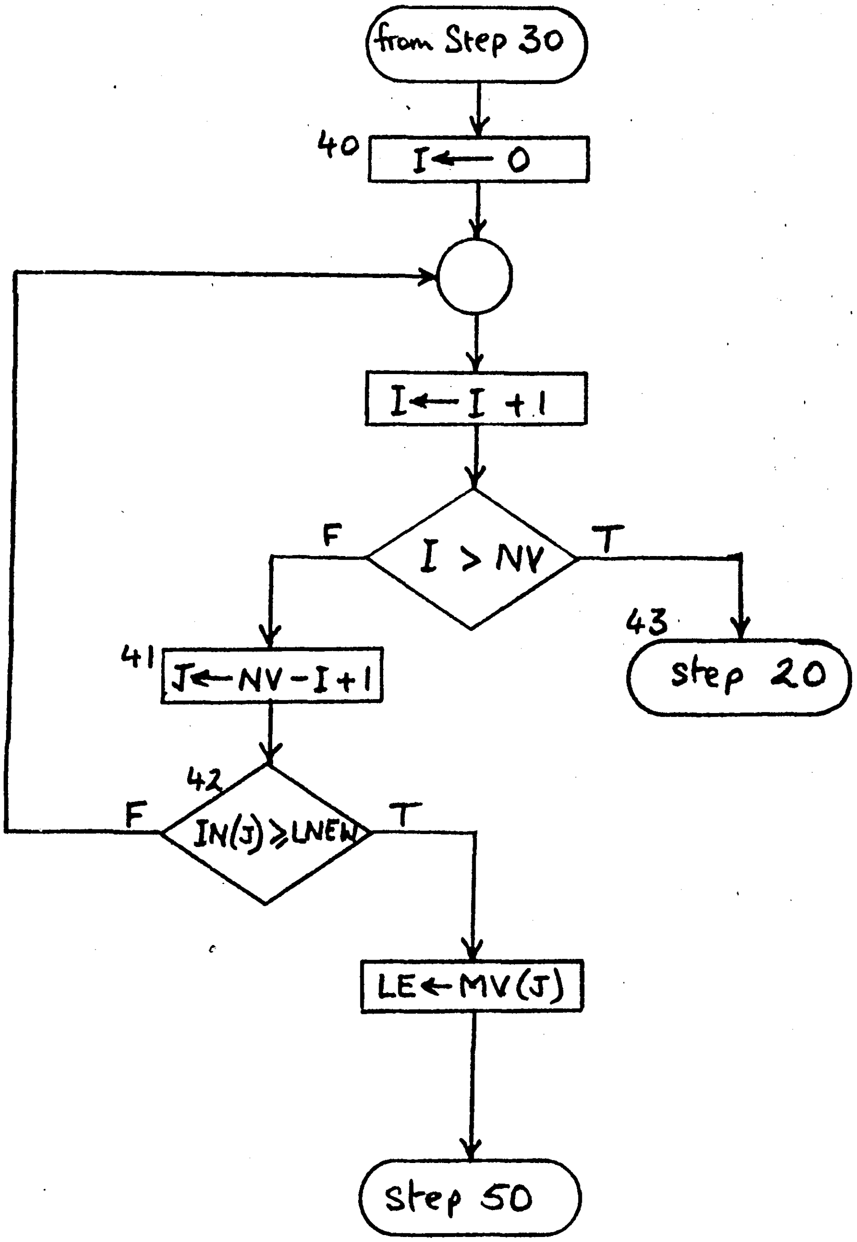


Figure 19

```
Step 50 (find the first available row counting back from the
         last row)
        for I←2 to NF  do step 51 od
        step 51  J←NF - I + 2; if KR(J)>LNEW then step 52 fi
        step 52  LO←J;  goto step 60
Step 53 if LNEW>2 then LO←0  else LO←-1;  goto step 90

Step 60 (mark row and create test defining contrast)
         KR(LO)←LNEW;  KEST←eor(LE,K(LO,1));  goto  step 80

Step 70 (use generator to create test defining contrast)
         LL←LL + 1;  KEST←eor(KK(LL), KK(LNEW))
```

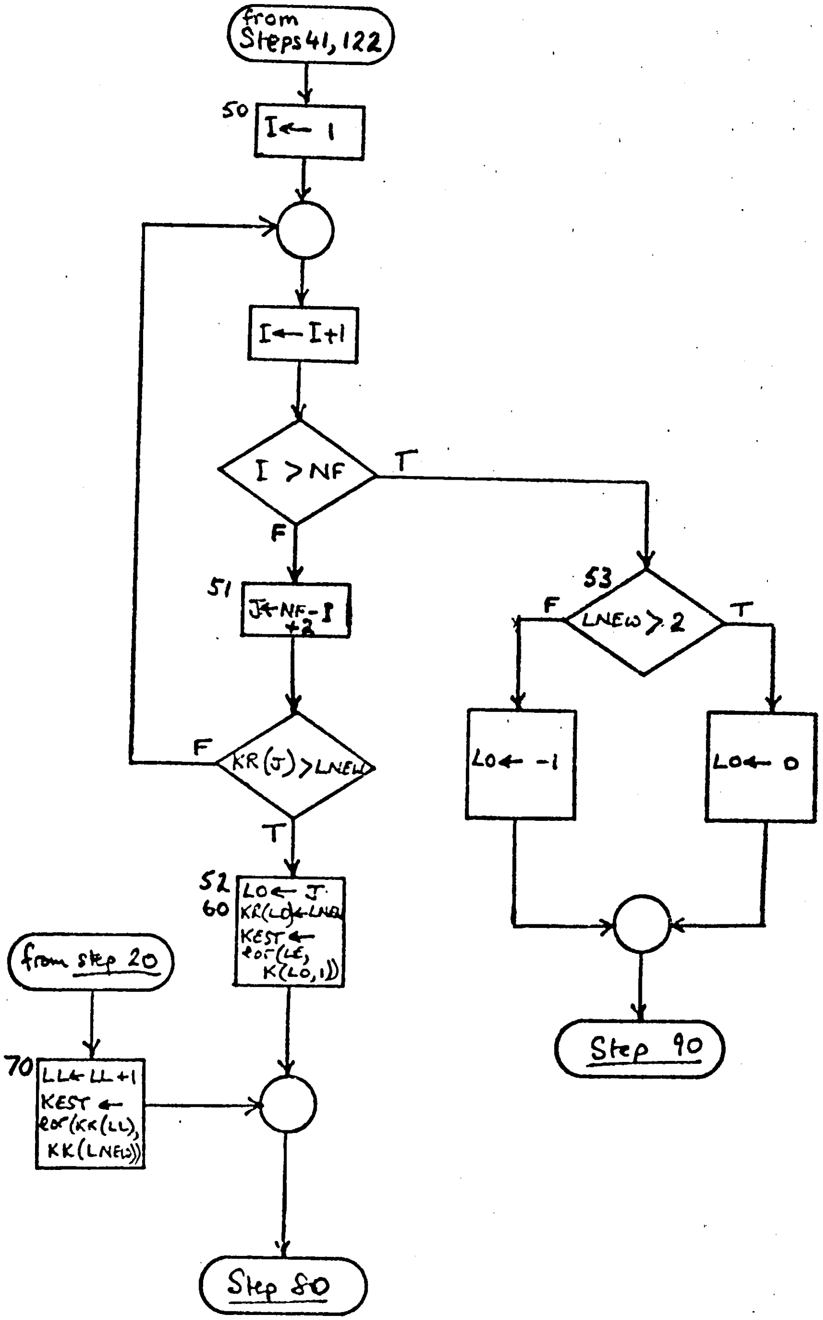


Figure 20

```
Step 80   (test new defining contrast for aliasing and set markers)
          I←1
Step 81   I←I + 1;
          if  I>NF then KK(JAK)←KEST; goto step 20 fi
Step 82   K(I,JAK)←eor(K(I,1),KEST);    J←0
Step 83   J←J + 1;
          if J>NV then goto step 81 fi
Step 84   if MV(J) ≠ K(I,JAK) then goto step 83 fi
Step 85   if IV(I) ≠  -1 then goto step 120 fi
Step 86 IV(I)←0;  IN(J)←LNEW;KR(I)←LNEW;  goto step 83
```

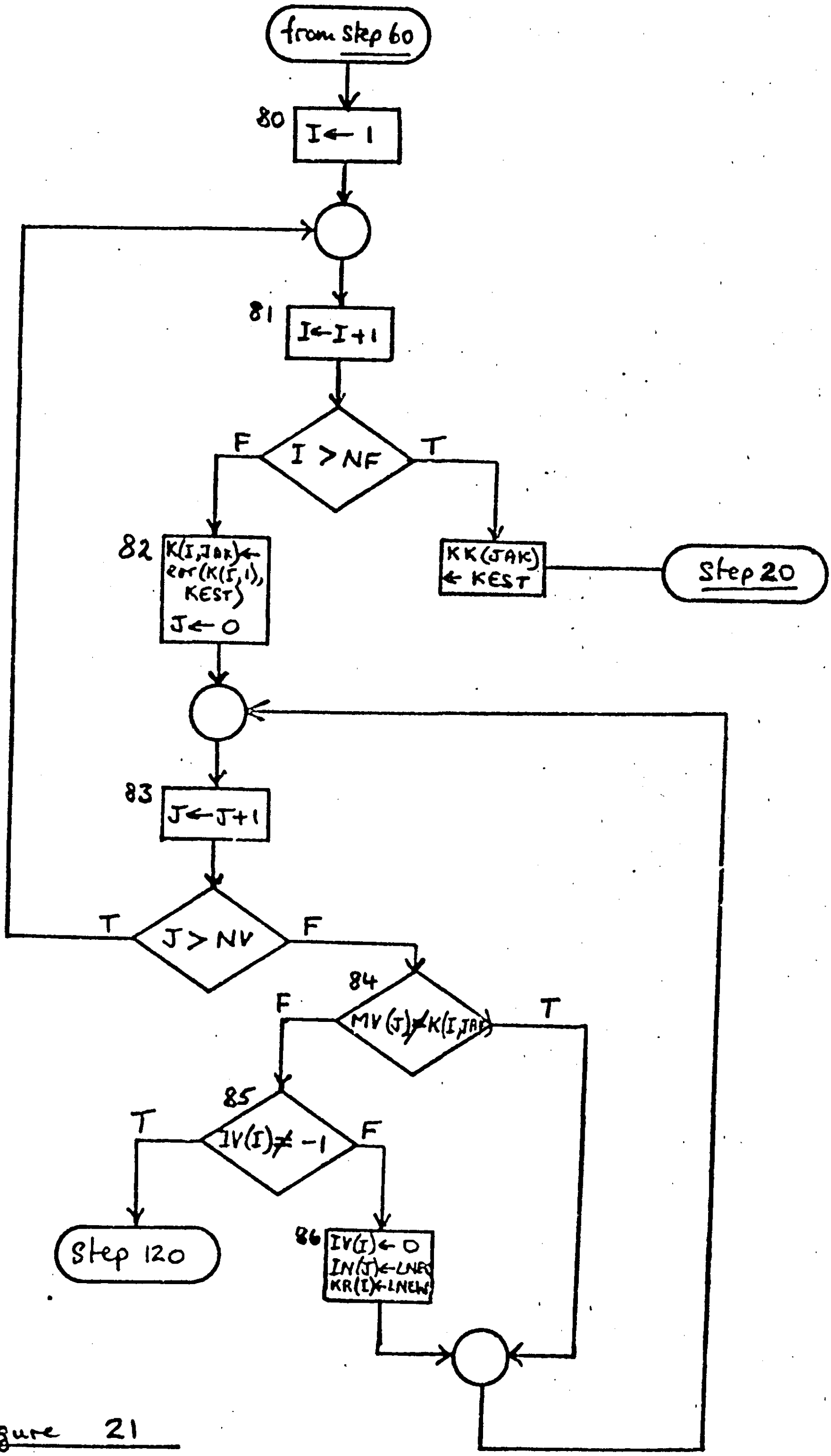


Figure 21

```
Step 90   if LO = 0 then do step 100; step 101 od
                    else do step 110 od fi
          Step 100  JAK←(LNEW-1)/2 + 1;
                    for I←2 to NF do step 102 od;
          Step 101  goto step 30
          Step 102  if KR(I)≥LNEW then KR(I)←100 fi
          Step 110  NF←NF*2;  NI←NI/2;  goto step 10
```

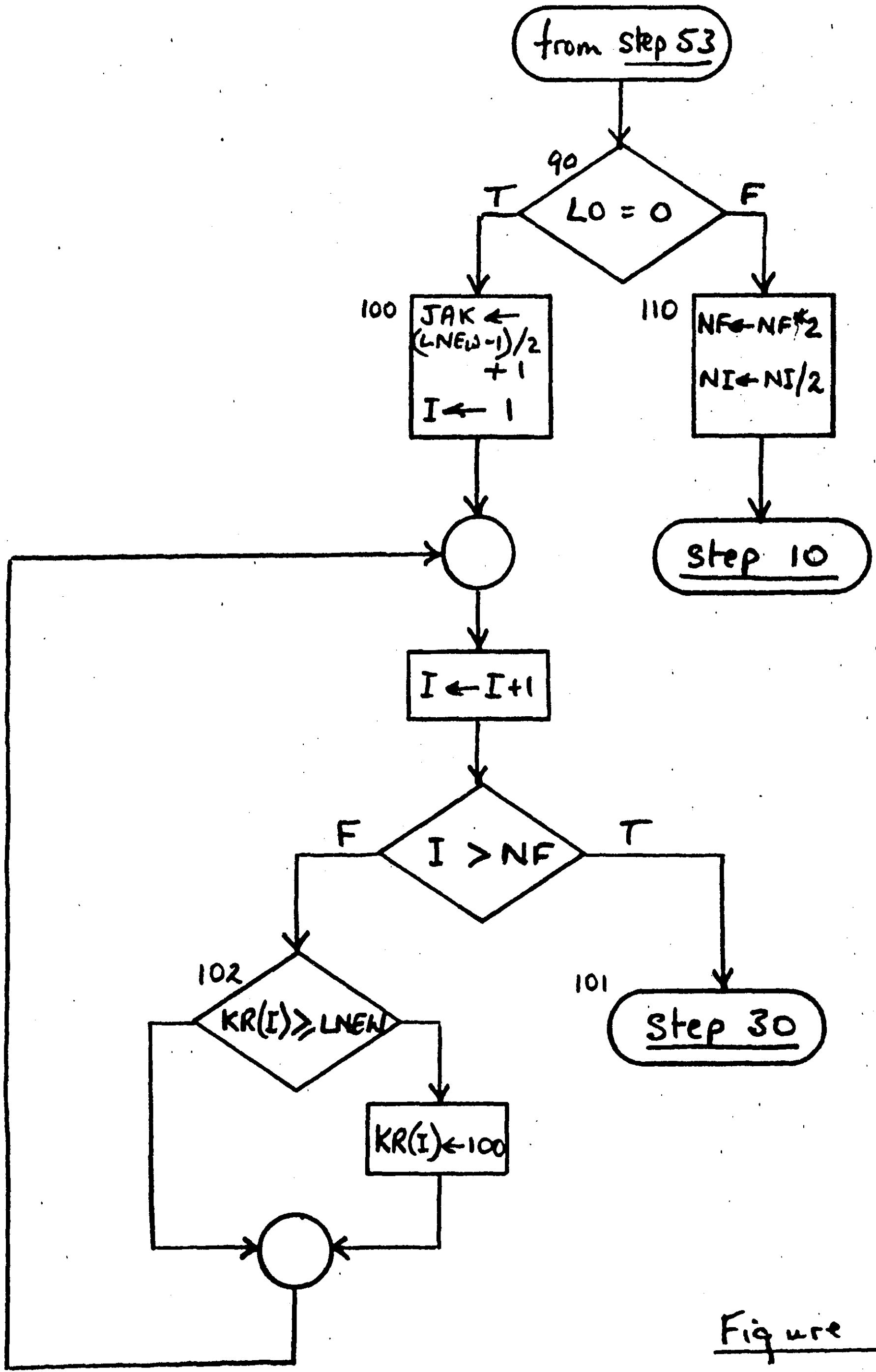


Figure 22

```
Step 120 (since KEST has failed, reset the requirements set markers)
         for J←1 to NV do step 121 od
         step 121  if IN(J)≥LNEW  then  IN(J)←100 fi
Step 122 goto step 50
```

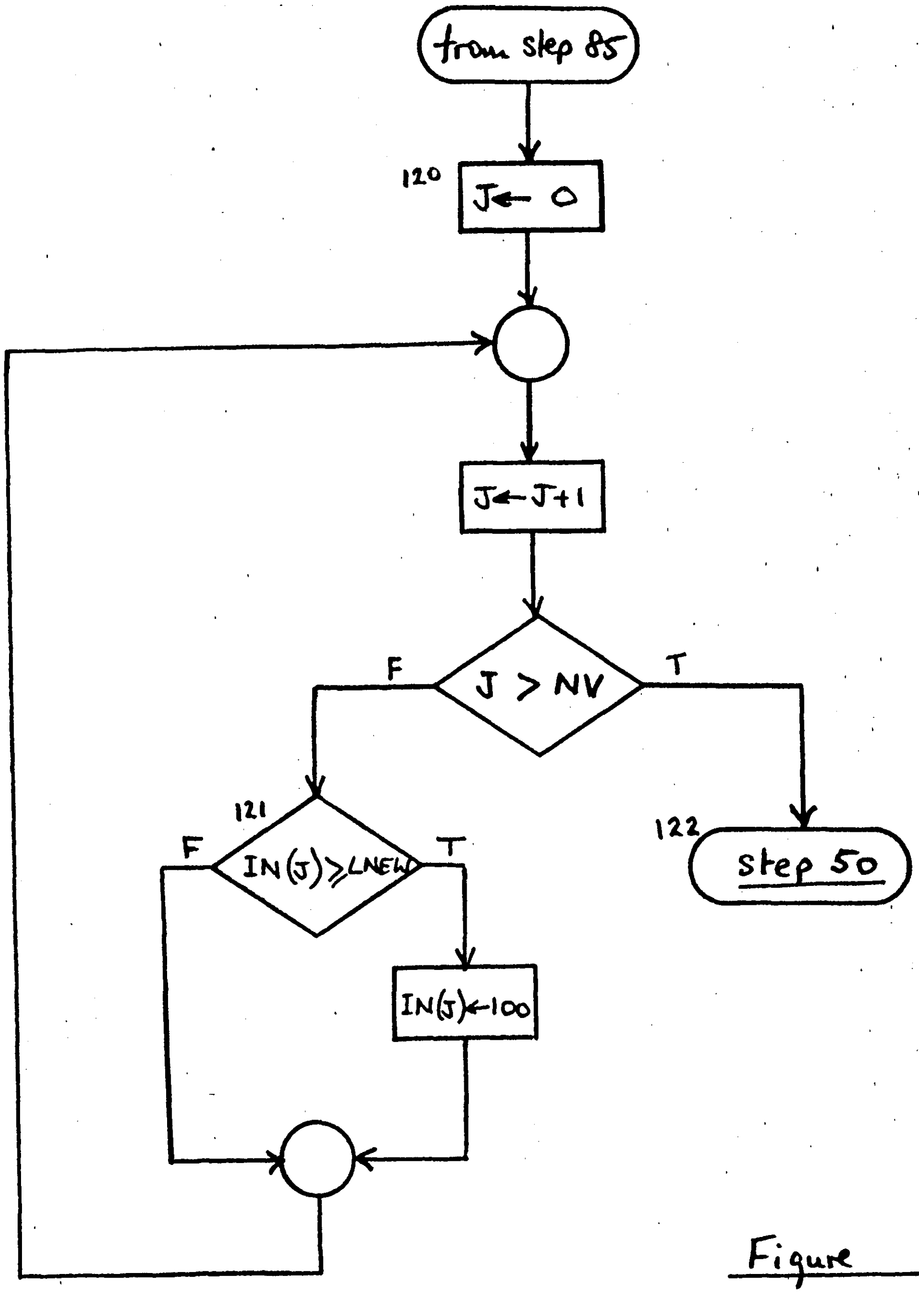


Figure 23

An algorithm for the function subprogram NEW(I), which tests if I is of the form $2^r + 1$, is:

```
Algorithm NEW(I)  (test if I is of the form $2^r + 1$, r an integer)
Step 0  enter I;   step 1 if I = 2 then set NEW←1;  return  fi
Step 2  NEW←0;   J←1;   I1←I - 1
Step 3  J←J*2
Step 4  if J<I1  then goto step 3
        else if J  = I1  then NEW←1 fi fi
Step 5  return
```

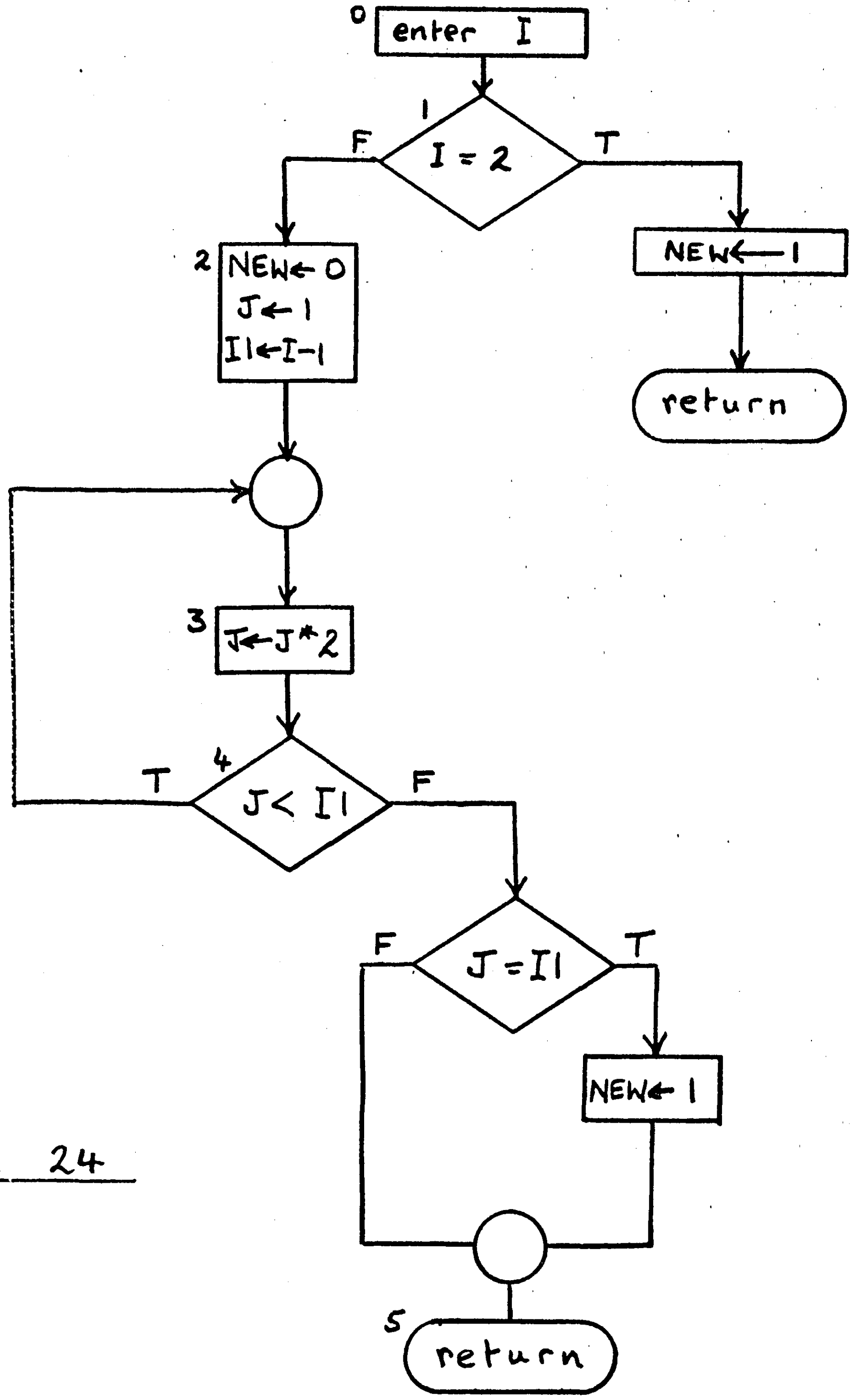


Figure 24

Execution of the algorithm DEFCON will yield an aliasing matrix K(.,.) with NF rows and NI columns. The first row of the aliasing matrix represents the set of defining contrasts and it is also held in the vector KK(.). Only the first column and the first row of the aliasing matrix are needed to produce the fractional design. Experience has shown, however, that users are more confident in the design if they are able to inspect the aliasing matrix. The following algorithm is needed to print the matrix. It includes a procedure to convert the bit notation into alphameric notation. For example, the 16-bit integer 0000000010100101 must be converted into the printed string ACFH and justified right in the column in which it is printed.

If the aliasing matrix has more than eight columns (there may be 16, 32, 64, or 128), the algorithm will divide it into eight-column wide blocks for printing.

The algorithm assumes that an array A(16) has already been assigned the 16 characters A to R, excluding I and O, and a variable BLANK has been assigned the blank character. These assignments can be done in the program with a data statement.

```
Algorithm ALMAT (print ALiasing MATrix)
Step 0   print heading ('Aliasing matrix for (name of experiment)')
Step 1   (number of blocks to be printed) NB←1 + (NI - 1)/8;
         (and number of columns per block) if NI>8 then NS←8
                                                    else NS←NI fi
Step 2   for I1←1 to NB do step 3 od
Step 3   NT←(I1 -1)*8;  for I2←1 to NF do step 4; step 10 od
Step 4   for I3←1 to NS do step 5; step 6; step 8 od
Step 5   NX←NT + I3;  J←K(I2,NX);  L←0
Step 6   for I4←1 to 16 do step 7 od
Step 7   if ITEST(J,I4) = 0 then L←L+1; B(I3,L)←BLANK  fi
Step 8   for I4←1 to 16 do step 9 od
Step 9   if ITEST(J,I4) ≠ 0 then L←L + 1;  B(I3,L)←A(I4)  fi
Step 10  for I3←1 to NS  print (B(I3,L), L←1 to 16)
```

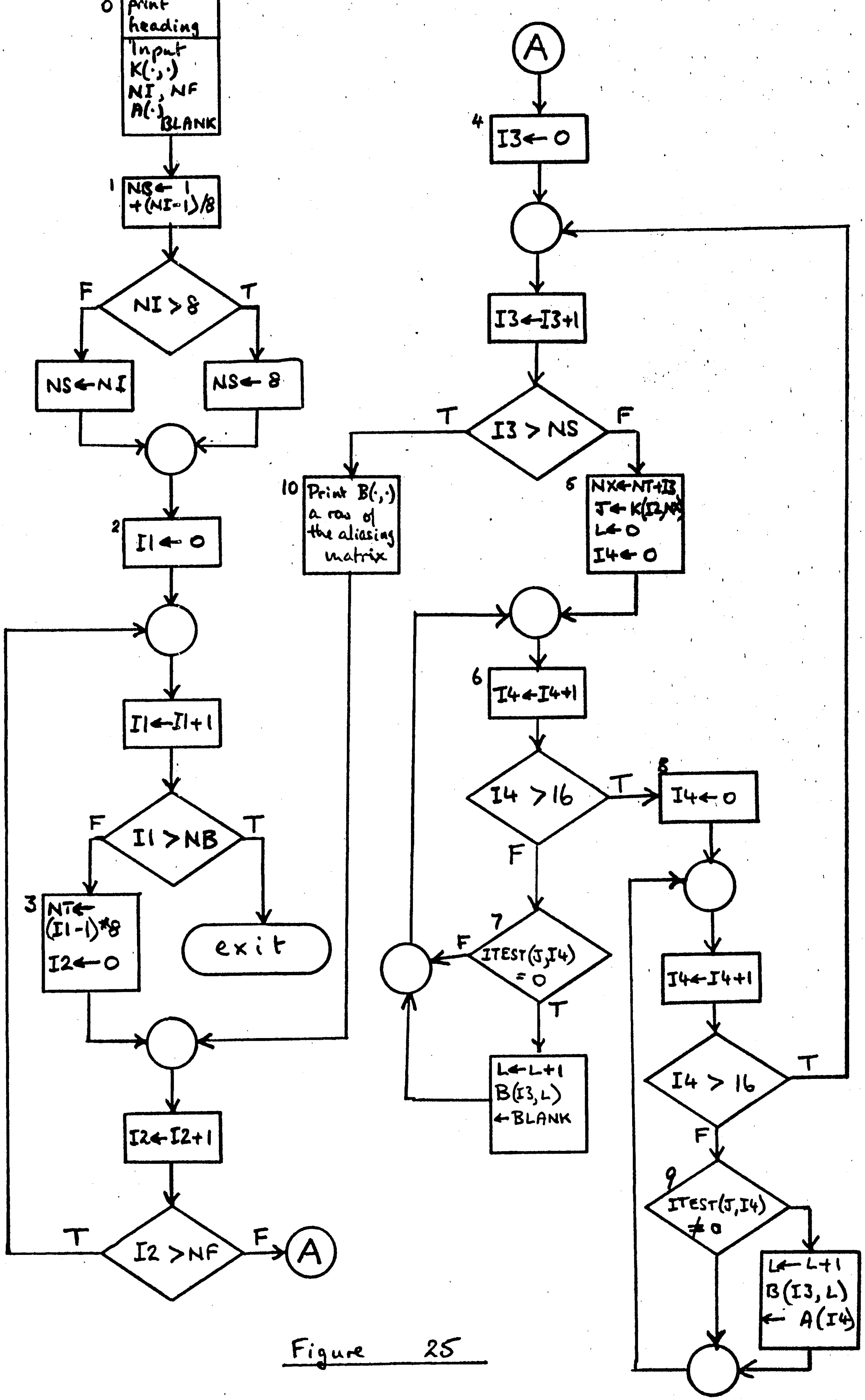

0
print heading
Input K(·,·) NI, NF A(·) BLANK
1
NB← 1 +(NI-1)/8
NI > 8
F
T
NS←NI
NS← 8
2
I1← 0
I1←I1+1
I1 >NB
F
T
3
NT← (I1-1)*8 I2← 0
exit
I2←I2+1
I2 >NF
T
F
A
A
4
I3← 0
I3←I3+1
I3 > NS
T
F
10
Print B(·,·) a row of the aliasing matrix
5
NX←NT+I3 J← K(I2,NX) L← 0 I4← 0
6
I4←I4+1
I4 >16
T
F
8
I4← 0
7
ITEST(J,I4) = 0
F
T
L←L+1 B(I3,L) ←BLANK
I4←I4+1
I4 > 16
T
F
9
ITEST(J,I4) ≠ 0
L←L+1 B(I3,L) ← A(I4)

Figure 25

A practical consideration in implementing the algorithms developed so far is the computer store needed for the program and data. By far the greatest store requirement is for the aliasing matrix. If the program is expected to generate factorial designs with as many as 16 factors, then an array of size 2^{16} would be needed to store the aliasing matrix. As mentioned earlier, however, the aliasing matrix in its entirety is not essential but may be preserved, converted and printed only to support the users' confidence. It is essential to preserve only the first column and the first row of the matrix and this can be done with two one-dimensional arrays, one of which is KK(.). If this is done, then the testing of a defining contrast (steps 80 to 86) would be done by creating and testing for aliasing an integer scalar (say KT) instead of an element of the matrix. That is:

in step 82 KT $\leftarrow$ eor(K(I), KEST)
instead of step 82 K(I,JAK) $\leftarrow$ eor(K(I,1),KEST
and in step 84 if MV(J) $\neq$ KT
instead of step 84 if MV(J) $\neq$ K(I,JAK)

Some corresponding alterations would be needed to ALMAT.

However, in continuing the development of these algorithms that the full aliasing matrix has been carried forward with full dimensions available. The first row is also available as a one dimension array of the defining contrasts, KK(.)

The experimental design is defined by the aliasing matrix. There is a choice of 2^m designs, each being a block of the full 2^n design. That block usually chosen, however, is the principal block containing the identity element (1) which is the point at which all the factors are at there lowest levels. This happens to be a sub-group of the full design, which has advantages that will become apparent in chapter seven. The other possible fractional blocks are cosets of the principal block. Once the defining contrasts have been determined by a well-known procedure which is described in numerous textbooks (see, for example, Duckworth (1968)). In simple algorithmic terms, the procedure is as follows:

Algorithm FRADE (FRActional DEsign)

Step 0 Copy the first column of the aliasing matrix into a design vector, KD(.)

Step 10 Determine the first zero bit from the right in the last element of KD(.). This represents the factor to be added to the design. (J)

Step 20 Find a defining contrast which contains that bit and any of the others already included in the design. (JIP)

Step 30 Copy that defining contrast, removing the bit corresponding to the factor to be added. (NT)

Step 40 For each element of the design, starting from KD(2), determine the number of bits it has in common with the copied contrast.

Step 50 If the number of bits is even return to step 40, but if it is odd add the new factor to the design element being checked and then return to step 40.

Step 60 Return to step 10 until all factors are in the design.

In the example discussed earlier, in which the defining contrasts were: I, ABDE, BCE, ACD, the procedure would be:

Step 0 KD←(I, 00000), (A, 00001), (B, 00010), (AB, 00011)
(E, 10000), (AE, 10001), (BE, 10010), (ABE, 10011)

Step 10 J←(C, 00100)

Step 20 JIP←(BCE, 10110)

Step 30 NT← eor(00100, 10110) = 10010

Step 40 and(00001, 10010) has zero bits (step 50)
and(00010, 10010) has one bit, so add (C,00100) to (00010) to give (BC, 00110)

continuing until

KD = (I, 00000), (A, 00001), (BC, 00110), (ABC, 00111),
(E, 10000), (AE, 10001), (BCE, 10110), (ABCE, 10111)

Then, returning to step 10 J← (D, 01000)
step 20 JIP← (ABDE, 11011)

Passing again through steps 30, 40, and 50, we finally arrive at the following design:

I , AD, BCD, ABC, DE, AE, BCE, ABCDE

To conform with convention, this principal block should be printed in lower case and the identity represented as (1). However, few computer printers have character sets which include lower case so the same capital notation that was used to represent effects will be used to represent the design. In the computer printout, the context will make the distinction quite clear. **The algorithm may now be expressed in more detail:**

```
Algorithm FRADE (FRActional DEsign)
Step 0   for I←1 to NF do KD(I)←K(I,1) od
Step 10  JJ←KD(NF)
Step 15  NE←NE + 1
Step 16  for I←1 to N do step 17 od
Step 17  J←IONBT(0,I); if and(J,JJ)=0 then goto step 20 fi
Step 20  JJ←or(J,JJ)
Step 21  for I←2 to NI do step 22 od
Step 22  JIP←KK(I); if J = and(J,JIP) then if JIP = and(JJ,JIP)
                                    goto step 30 fi fi
Step 30  NT← eor(J,JIP)
Step 40  for I←2 to NF do step 41; step 42; step 50 od
Step 41  NB← and(NT, KD(I)); L←0
Step 42  for II←1 to N do step 43 od
Step 43  if ITEST(NB,II)= 1 then I←L + 1 fi
Step 50  if ITEST(L,1) = 1 then KD(I)←or(KD(I),J) fi
Step 60  if NE<N then goto step 15
                   else goto step 70 (to decode and print design) fi
(step 73 and the following steps are equivalent to steps 6 to 10
in algorithm ALMAT)
Step 70  B(1,16)←'I'; for I←1 to 15 do B(1,I)←BLANK od
Step 71  for I1←2 to NF do step 72; step 73; step 75 od
Step 72  J←KD(I1); I← 0
Step 73  for I2←1 to 16 do step 74 od
Step 74  if ITEST(J,I2) = 0 then I←L + 1; B(I1,L)←BLANK fi
Step 75  for I2←1 to 16 do step 76 od
Step 76  if ITEST(J,I2) ≠ 0 then I←L + 1; B(I1,L)←A(I2) fi
Step 78  print heading ('Design for (name of experiment)')
Step 79  for I1←1 to NF print (B(I1,L), I←1 to 16)
```

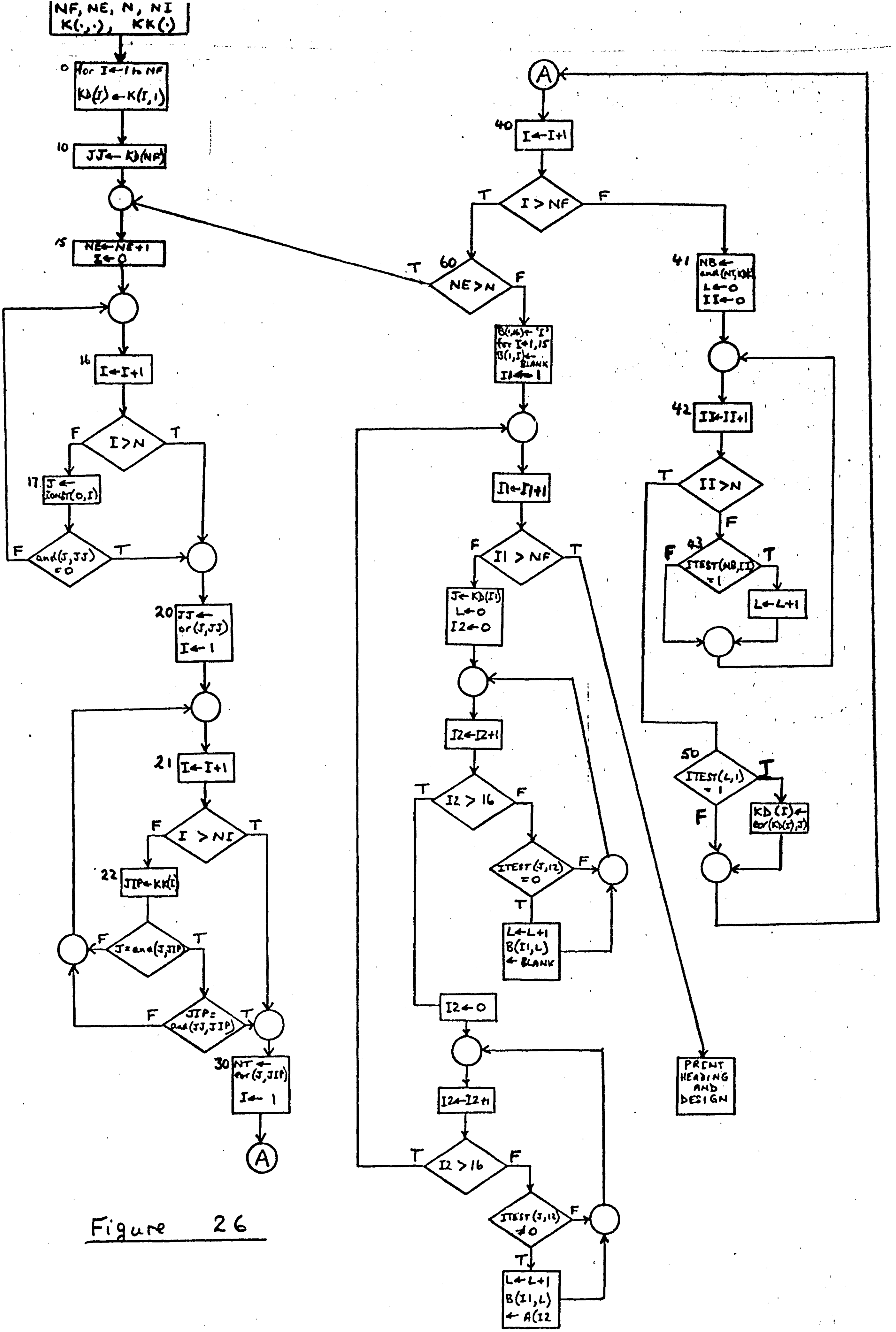

NF, NE, N, NI
K(·,·), KK(·)
0
for I←1 to NF
KD(I) ← K(I,1)
10
JJ← KD(NF)
15
NE←NE+1
I←0
16
I←I+1
I > N
F
T
17
J ← ICNST(0,I)
and(J,JJ) =0
F
T
20
JJ ← or(J,JJ)
I← 1
21
I←I+1
I > NI
F
T
22
JIP← KK(I)
J=and(J,JIP)
F
T
JIP= and(JJ,JIP)
F
T
30
NT ← xor(J,JIP)
I← 1
A
A
40
I←I+1
I > NF
T
F
60
NE > N
T
F
B(1,16)← 'I'
for I←1,15
B(1,I)← BLANK
I1← 1
I1←I1+1
I1 > NF
F
T
J← KD(I1)
L←0
I2←0
I2←I2+1
I2 > 16
T
F
ITEST(J,I2) =0
F
T
L←L+1
B(I1,L) ← BLANK
I2←0
I2←I2+1
I2 > 16
T
F
ITEST(J,I2) ≠0
F
T
L←L+1
B(I1,L) ← A(I2
PRINT HEADING AND DESIGN
41
NB← and(NT,KD(I))
L←0
II←0
42
II←II+1
II > N
T
F
43
ITEST(NB,II) =1
F
T
L←L+1
50
ITEST(L,1) =1
T
F
KD(I)← xor(KD(I),J)

Figure 26

There still remains the first major step of the total algorithm: setting up the initial conditions. That is, entering the number of factors and the requirements set. In a later chapter this will be considered from the broad viewpoint of a conversation between the user and the computer, in which the user will be led through a question and answer routine to establish first the type of experimental design and then the conditions. Only a simple algorithm will be described here for entering the requirements in alphameric form and converting them into the binary form needed by DEFCON. In general terms the algorithm is:

```
Algorithm  ENFAC  (ENter FACtorial requirements)
Step 1     enter N  (the numberof factors) and experiment name
Step 2     generate the first N requirements  (the main effects)
Step 10    enter each required interaction as a set of letters
           and convert to binary form
Step 20    when there are no more entered,  terminate and link
           to DEFCON
```

Developing each step more fully, the algorithm becomes:

```
Algorithm  ENFAC  (ENter FACtorial requirements)
Step 1     Print 'How many factors are there?';  read N
Step 2     for NV←1 to N  do step 3  od (assume all main effects
                                               wanted)
Step 3     MV(NV)←0;  MV(NV)←IONBT(MV(NV), NV)
Step 10    Print 'enter required interaction'
Step 11    for I←1 to 16   read KK(I)   (using in Fortran, a
                                          16A1 format)
Step 12    J←NV + 1;  MV(J)←0;  L←0
Step 13    for I←1 to 16  do  step 14   od
Step 14    for I1←1 to 16  do step 15  od
Step 15    if  KK(I) = A(I1)   then MV(J)←IONBT(MV(J), I1);
                                    L←L + 1  fi
Step 20    if  L>0  then  NV←J;  goto step 10  fi
Step 21    link to DEFCON
```

The algorithm makes use of the array A(.) also used in ALMAT. The 16 characters, A to R, excluding I and 0, have been assigned to the 16 elements of A(.). The algorithm checks each entry for the presence of any of these characters and sets corresponding bits in an element of MV(.). The flowchart is:

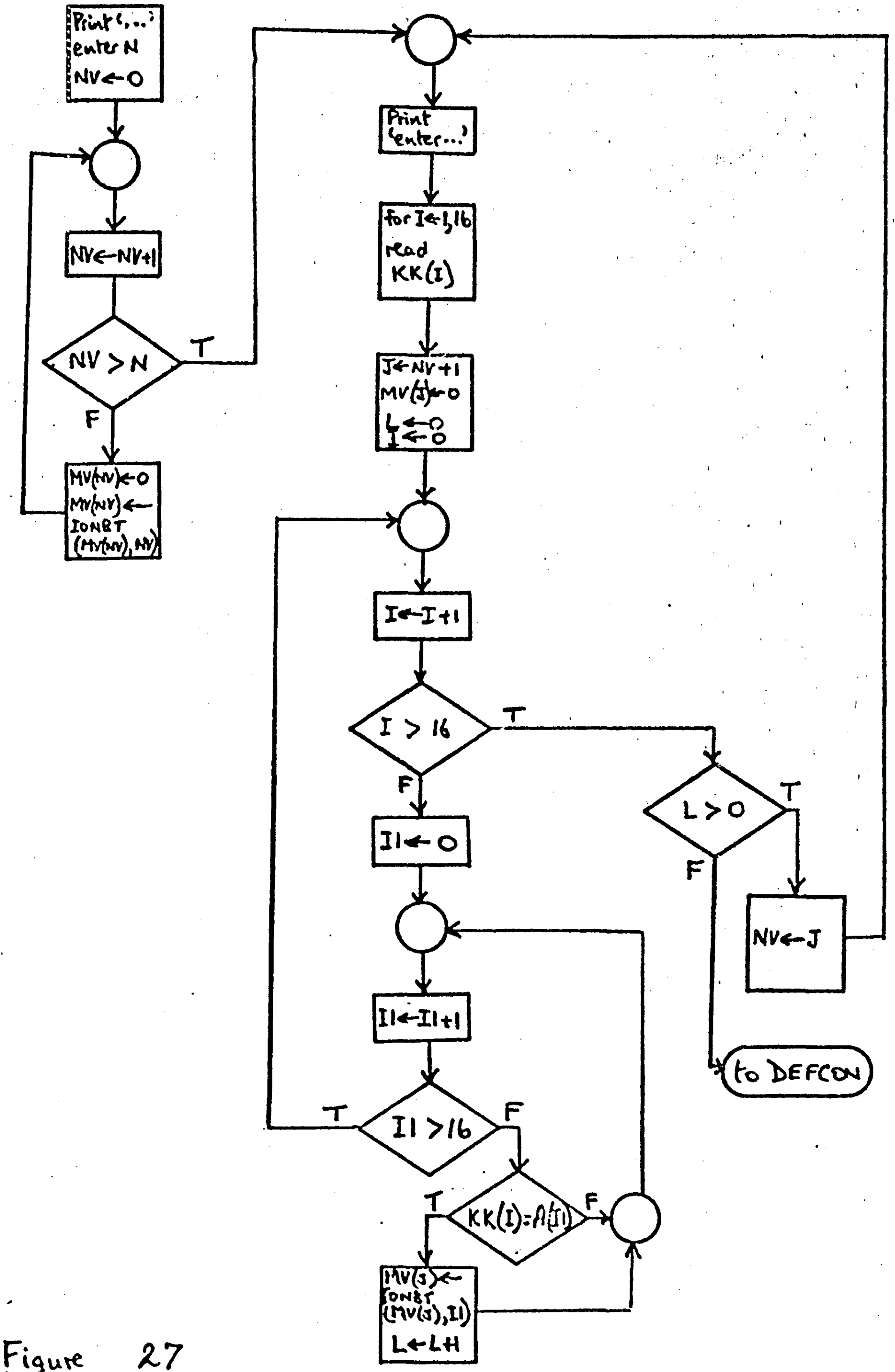


Figure 27

3. Example

An example has been chosen that is simple enough to be used as an illustration but awkward enough to demonstrate the backtracking features of the algorithm.

Consider a 2^5 experiment in which in which it is required to estimate the interactions AC and DE as well as the main effects. (For easy reading, alphameric representation will be used throughout, remembering all the while that the algorithms will actually be converting to, working in, and converting from binary coding).

Algorithm ENFAC

How many factors are there?

5

The first five elements of MV(.) are generated as (A,B,C,D,E)

Enter required interaction

AC

Enter required interaction

DE

Enter required interaction

(blank)

There are now seven elements of MV(.) which are (A,B,C,D,E,AC,DE). These, with the values of N = 5 and NV = 7, are passed to:

Algorithm DEFCON

The fraction index M is computed to have a minimum value of 2, indicating a quarter design.

The requirements, MV(.), are scanned to determine the majority factors, MAJ(.), as (A, C, D, E, B).

The first three elements of MAJ(.) become the generators of the first column of the aliasing matrix K(.,1), which with the help of function NEW, becomes:

$(I, \; A, C, \; AC, D, AD, CD, ACD)^T$ where T indicates that the vector is a column.

The rows containing the elements (A, C, AC, D) are marked with the markers IV(.) and KR(.), and the corresponding elements of MV(.) are marked with markers IN(.).

The column index, JAK, is incremented to the value 2 which, by the function NEW is noted to be of the form $2^r + 1$ so LNEW←2 and LL←1. At this stage no markers need resetting.

The first available requirement, counting from the end, is DE.

The first available row element is ACD in row 8.

The row marker for row 8 is set to the value of LNEW = 2.

The defining contrast to be tested is created by eor(ACD, DE), which is ACE.

This is found to produce aliasing between AC and E in the fourth row. The test has set the value of the seventh defining contrast marker IN(7) to 2, so it is reset to 100 indicating that it is still available.

The next lowest available row element is CD in row 7.

The marker for row 7 is set to the value of LNEW = 2.

The defining contrast to be tested is eor(CD, DE) =CE.

This also produces aliasing, between C and E, and the procedure is repeated.

The next lowest available row element is AD, leading to the defining contrast to be tested: eor (AD, DE) = AE.

This also produces aliasing.

There is no further available row. The value of LO = -1 indicates that LNEW is already at its smallest value of 2 so no backtracking is possible with the current fraction.

The fraction is effectively doubled by doubling the number of rows to 16 and halving the number of columns to 2.

The first four elements of MAJ(.) become the generators of the first column of the aliasing matrix K(.,1), which with the help of function NEW, becomes:

(I, A, C, AC, D, AD, CD, ACD, E, AE, CE, ACE, DE, ADE, CDE, ACDE)^T

All the requirements, except B, now appear in this column. The row markers are set and the corresponding requirements markers are set.

The column index JAK is incremented to 2.
The first available requirement is B.
The first available row is the 16th which contains ACDE.
The defining contrast to be tested is eor(ACDE, B) = ABCDE.
No aliasing is found, so the defining contrasts are (I,ABCDE).

Algorithm ALMAT

The aliasing matrix is printed:

I	ABCDE
A	BCDE
C	ABDE
AC	BDE
D	ABCE
AD	BCE
CD	ABE
ACD	BE
E	ABCD
AE	BCD
CE	ABD
ACE	BD
DE	ABC
ADE	BC
CDE	AB
ACDE	B

Algorithm FRADE

follows the well-established procedure to produce the half-design given the defining contrasts. This is printed in capitals (due to the restriction of most computers) as:

I, AB, BC, AC, BD, AD , CD, ABCD, BE, AE, CE, DE, ABDE, BCDE, ACDE

All that is needed now to complete the exercise is a set of random numbers for the order of observations. However, algorithms for the generation of random numbers will be left to a later chapter dealing with a wider range of experimental designs,

CHAPTER FOUR

QUADRATIC DESIGNS

1 Background

2 Algorithms

3 Examples

1 Background

As I explained in chapter two, most physical processes met in industrial research situations can be described by models which are either linear or quadratic in the independent continuous variables. In my experience this has always been so, provided the ranges of the independent variables were carefully chosen. Since we now have the algorithm developed in chapter three to design the smallest fraction of a two-level factorial for the estimation of the coefficients of explicitly required main effects and interactions, the question arises: can this fractional design be augmented in any way so that the coefficients of specified quadratic terms can be estimated? The answer is that it can, but in several ways. The choice of the method depends upon the criteria used to determine a good design.

Consider the simple case of two independent variables, x_1 and x_2, such that their relationship to a response variable y is:

$$y = a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 \qquad (4.1)$$

then a simple two-level factorial experiment would provide observations suitable for the estimation of the model parameters (a_0, a_1, a_2, a_{12}). These are transformations of the effects and interactions conventionally estimated from a factorial experiment.

If, however, some curvature is suspected so that a more appropriate model would be:

$$y = a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 + a_{11}x_1^2 + a_{22}x_2^2 \qquad (4.2)$$

then the factorial must be augmented by further design points to permit the estimation of the additional parameters (a_{11} and a_{22}).

If the design points of an n-factorial experiment are represented as points in n-space, they occur at the vertices of an n-dimensional hypercube. The question is: where should the further design points be created so as to allow the fitting of a quadratic model?

Criteria considered in the answer to this question relate to the estimates of the above model which can be expressed more generally as:

$$\hat{y} = f(\hat{\underset{\sim}{a}}, \underset{\sim}{x}) \qquad (4.3)$$

where $\hat{\underset{\sim}{a}}$ are the estimates of the model coefficients $\underset{\sim}{a}$, using the available data, and $\hat{y}$ is an estimate of a particular value of the response variable, given the coefficient estimates $\hat{\underset{\sim}{a}}$ and a set of values of the independent variables $\underset{\sim}{x}$.

The criteria are:

1. The estimators $\hat{\underset{\sim}{a}}$ should be unbiased. That is: their expected values should be $\underset{\sim}{a}$.
2. The variance of $\hat{\underset{\sim}{a}}$ should be minimised in relation to the choice of design points.

Considering the first criterion, one approach is to ensure that estimators are not biased by the inclusion of more or less parameters, or coefficients, in the model. Box and Wilson (1951) showed that this could be achieved by making the estimators mutually independent and that this could be done by making the design orthogonal.

A design is said to be orthogonal if all the vectors of the design matrix are mutually orthogonal; that is, if the inner product of every pair of vectors is equal to zero.

Orthogonality also satisfies the second criterion that the estimated coefficients should have minimum variance This was proved by Tocher (1952c) as follows:
Given the design matrix $\underset{\sim}{X}$, Markoff's least squares theorem gives the minimum variance unbiased estimates of the parameters $\underset{\sim}{a}$ as $\hat{\underset{\sim}{a}} = (\underset{\sim}{X}'\underset{\sim}{X})^{-1}\underset{\sim}{X}'\underset{\sim}{y}$ and $\mathrm{var}(\hat{\underset{\sim}{a}}) = (\underset{\sim}{X}'\underset{\sim}{X})^{-1}\sigma^2$.
If the elements of $\underset{\sim}{X}$ are allowed to increase indefinitely, then the elements of $(\underset{\sim}{X}'\underset{\sim}{X})^{-1}$ will decrease indefinitely, so some restraint must be imposed on the elements of $\underset{\sim}{X}$ to ensure a realistic solution. Assume this restraint fixes the sum of squares of the i^{th} column of $\underset{\sim}{X}$ as c_i.
Let $\underset{\sim}{U}$ be an upper triangular matrix such that $\underset{\sim}{X}'\underset{\sim}{X} = \underset{\sim}{U}'\underset{\sim}{U}$.
Since $\underset{\sim}{X}'\underset{\sim}{X}$ is positive definite, $\underset{\sim}{U}$ is non-singular and has an inverse $\underset{\sim}{V}$ which is also upper triangular. Clearly
$v_{ii} = 1/u_{ii}$ and $(\underset{\sim}{X}'\underset{\sim}{X})^{-1} = \underset{\sim}{V}\underset{\sim}{V}'$.
Suppose the i^{th} diagonal element of $(\underset{\sim}{X}'\underset{\sim}{X})^{-1}$ is w_i, then

$$w_i = \sum_j v_{ij}^2 \geqslant v_{ii}^2 = 1/u_{ii}^2 \geqslant 1/\sum_j u_{ji}^2 = 1/c_i$$

For w_i to achieve its lower bound $1/c_i$, both inequalities must reduce to equalities, and for this to occur for all i, all the elements v_{ij} with $i \neq j$ must vanish. Thus $\underset{\sim}{V}$ will be diagonal and so will $\underset{\sim}{U}$. Thus the condition to achieve the lower bounds simultaneously is that $\underset{\sim}{X}'\underset{\sim}{X}$ is diagonal; that is that $\underset{\sim}{X}$ has orthogonal columns.

Returning now to the simple two-factor example (equation 4.1), the values of the variables x_1 and x_2 may be scaled to have the values: -1 for low level and $+1$ for high level. If a dummy variable x_0 with a constant value of $+1$ is attached to the coefficient a_0, then the design matrix $\underset{\sim}{X}$ becomes:

$$\begin{array}{cccc} x_0 & x_1 & x_2 & x_1x_2 \\ \hline \end{array}$$

$$\left[\begin{array}{cccc} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{array}\right]$$

This design is clearly orthogonal since the inner product of any pair of column vectors is equal to zero. If, instead of the above linear model, the quadratic model (equation 4.2) is to be fitted, the design must be augmented so as to maintain orthogonality when the extra rows and columns are added. Box and Wilson (1951) showed how to do this with the following composite designs.

Since curvature, or quadratic effects, needs intermediate points for its estimation, it seems reasonable to put one or more points at the centre of the design, and to put other points on the axes, so that variables will have values midway between +1 and -1. The distance, α, of each axial point from the origin is determined by the orthogonality condition. Thus, in the two factor design, assuming that only one observation is made at the design centre, the design matrix becomes:

$$
\begin{array}{cccccc}
x_0 & x_1 & x_2 & x_1x_2 & x_1^2 & x_2^2 \\
\end{array}
$$

$$
\left[\begin{array}{cccccc}
1 & -1 & -1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 & 1 & 1 \\
1 & 1 & -1 & -1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
\hdashline
1 & -\alpha & 0 & 0 & \alpha^2 & 0 \\
1 & \alpha & 0 & 0 & \alpha^2 & 0 \\
1 & 0 & -\alpha & 0 & 0 & \alpha^2 \\
1 & 0 & \alpha & 0 & 0 & \alpha^2 \\
1 & 0 & 0 & 0 & 0 & 0
\end{array}\right]
$$

The vectors corresponding to x_1^2 and x_2^2 are not orthogonal to the x_0 vector nor to each other, but by transforming the variables to x_1^* and x_2^* such that

$$x_i^* = x_i^2 - \sum_j x_{ij}^2/n = x_i^2 - (4 + 2\alpha^2)/9 \qquad (4.4)$$

orthogonality is achieved when fitting the equivalent model

$$y = b_0 + b_1x_1 + b_2x_2 + b_{12}x_1x_2 + b_{11}x_1^* + b_{22}x_2^* \qquad (4.5)$$

Total orthogonality of the design then demands that

$$\underset{\sim}{x}_1^{*}\,\underset{\sim}{x}_2^{*} = 0 \qquad (4.5)$$

In the two factor case, with only one central point, the value $\alpha = 1$ satisfies this equation so that a simple three level (or 3 x 3) experiment is suitable. However, in the augmentation of fractional designs, and with more factors, the value of α is usually different from 1. This procedure, developed by Box and Wilson, was further illustrated by Davies (1954) and Mendenhall (1968) who gave values for α for designs with more factors but with the assumption that quadratic terms were to be added to the models for all the factors. In practice, however, prior knowledge often enables the experimenter to state that whereas some factors may have quadratic terms, others do not. Thus the design need not be augmented with so many additional observation points for the estimation of quadratic terms in all factors. This is important from the viewpoint of keeping the cost of the experiment as low as possible. On the other hand, Mendenhall illustrated that the addition of some central points will lead to greater uniformity of var($\hat{y}$) without destroying the orthogonality. Furthermore, the addition of some extra central points may sometimes be necessary to provide sufficient degrees of freedom for the estimation of residual variance. In the algorithm to be developed in this chapter, the number of design centre points will be chosen so as to allow a minimum of six degrees of freedom for residual variance estimation.

The objective of this chapter is to present a procedure which avoids the condition of including quadratic terms for all factors and enables the experimenter to define which factors have quadratic effects and which do not.

2 Algorithms

The following theorem is needed to develop the main algorithm.

Theorem If a fractional two level factorial design, in which high and low levels of the factors are scaled ± 1 respectively, is augmented with central and axial points to permit the estimation of quadratic terms in some of the variables, then in order to retain orthogonality the axial points should be at distances $\pm\alpha$ from the origin, where

$$\alpha = \left\{ 0.5*\left(\left\{ nf*(nf + 2*nq + no)\right\}^{0.5} - nf\right)\right\}^{0.5}$$

and nf = number of rows in the basic two-level fractional factorial;

nq = number of factors for which quadratic effects are to be estimated;

no = number of design centre observations.

Let n = total number of rows in the design,

then

$$n = nf + 2*nq + no \qquad 4.6$$

Let x_i and x_j be any two variables for which quadratic effects are to be estimated,

and x_i^* and x_j^* be corresponding second degree transformation as described in the previous section

Then the required tranformations are of the form:

$$x_i^* = x_i^2 - \sum_k x_{ik}^2 / n \qquad 4.7$$

Since x_{ik} takes nf/2 values of +1, nf/2 values of -1, one value of $+\alpha$, one value of $-\alpha$, and all other values of 0, then

$$\sum_k x_{ik}^2 = nf + 2\alpha^2 \qquad 4.8$$

Now let

$$p = \sum_k x_{ik}^2 / n \qquad 4.9$$

then

$$p = (nf + 2\alpha^2) / (nf + 2*nq + no) \qquad 4.10$$

so that $\underset{\sim}{x}_i^* \underset{\sim}{x}_j^*$ (the inner product of the two column vectors $\underset{\sim}{x}_i^*$ and $\underset{\sim}{x}_j^*$)

$$= \sum_k (x_{ik}^2 - p)(x_{jk}^2 - p)$$

$$= \sum_k x_{ik}^2 x_{jk}^2 - 2p\sum_k x_{ik}^2 + n*p^2 \qquad 4.11$$

But from 4.9 $\sum_{K} x_{ik}^{2} = n*p$

and because of the ±1 coding of variables,

$$\sum_{K} x_{ik}^{2} x_{jk}^{2} = nf$$

therefore 4.11 becomes

$$\underset{\sim}{x}_{i}^{*} \underset{\sim}{x}_{j}^{*} = nf - n*p^{2} \tag{4.12}$$

and orthogonality requires this to equal zero.

$$\text{Therefore} \quad p^{2} = nf / n \tag{4.13}$$

Using both 4.10 and 4.13

$$(nf + 2\alpha^{2})^{2} / (nf + 2*nq + no)^{2}$$

$$= nf/(nf + 2*nq + no) \tag{4.14}$$

which is rearranged as

$$\alpha = \left\{ 0.5* \left(\left(\left\{ nf*(nf + 2*nq + no) \right\}^{0.5} - nf \right) \right) \right\}^{0.5} \tag{4.15}$$

and the theorem is proved.

The algorithm to augment the fractional factorial may now be expressed as:

Algorithm AUGFAC (AUGment fractional FACtorial design)

Step 0 (initialise) Given the number of factors, N, the number of requirements, NV, (main effects plus interactions), the number of quadratic terms, NQ, and each factor I identified as linear or quadratic with LO(I) = 'L' or 'Q', if NQ $\geq$ 1 find the number, NO, of design centre points needed (at least one) so that the number of residual degrees of freedom is at least six. Then find the value of ALPHA.

Step 10 Given the least value, R(I,1), the greatest value R(I,2), and the smallest step R(I,3), for each variable, compute the design intervals for each variable. Find the number of increments each side of the mid-range point; find the mid-range point and the points either side of it.

Step 20 Print the unscaled values of the variables in each row of the fractional design.

Step 30 Print the unscaled values of the variables in each row of the augmenting design, including the design centre points.

Step 40 Randomise the order of observations.

Steps 10, 20, and 40 are used even if there are no quadratic terms and the design is therefore not augmented.
A routine, RANDU, borrowed from the IBM 1130 scientific subroutine package, is used in step 40. This returns a random number from the uniform distribution (0,1). The Fortran listing is in the appendix.

In detail the algorithm becomes:

```
Algorithm  AUGFAC  (AUGment fractional FACtorial design)
Step 0     Given N,NV,NQ, LQ(.) ;  print column headings;
           if  NQ≥1  do steps 1 to 5 od  else goto step 10
Step 1     NP←NV + NQ + 1;   ND←NF + 2*NQ;
           M←ND - NP
Step 2     if  M≤5  then  NO←6 - M
                    else  NO←1       fi
step 3     (calculate ALPHA IF  NQ>1)
           if  NQ>1  then do  step 4; step 5  od
                     else  ALPHA←1   fi
Step 4     Z←NF * (ND + NO)
Step 5     ALPHA← sqrt(0.5 * (sqrt(Z)  -  NF))

Step 10    Given R(.,.)  for each variable
           for I←1 to N do  step 11;  step 12  od
Step 11    M←integer((R(I,2) - R(I,1)) / R(I,3))
           M←M/2 + itest(M,1);   P←M * R(I,3);
           X(I,3)←R(I,1) + P
Step 12    if  LQ(I) = "Q"  then do step 13  od
                            else do step 14  od  fi
Step 13    M←integer(P/(ALPHA * R(I,3)) + 0.5)
           Q←M * R(I,3);
           X(I,1)←R(I,1);
           X(I,2)←X(I,3) - Q;
           X(I,4)←X(I,3) + Q;
           X(I,5)←R(I,1) + 2.* P
Step 14    X(I,2)←R(I,1);
           X(I,4)←R(I,2)

Step 20    for I←1 to NF  do  step 21;  step 23  od
Step 21    for J←1 to N  do step 22  od
Step 22    L← 2 *(itest(KD(I), J) + 1);  Y(J)←X(J,L)
Step 23    print  I,  (Y(J),  J←1 to N)
Step 30   JAK←NF
Step 31   if NQ < 1 then goto step 40 fi
Step 32   for I←1 to N do step 33 od
Step 33   if LQ(I) = "Q" then do step 34; step 36 od fi
Step 34   for J←1 to N do step 35 od
Step 35   if I ≠ J then Y(J)←X(J,3) fi
Step 36   JAK←JAK + 1; Y(I)←X(I,1);
          print JAK, (Y(J), J←1 to N);
          JAK←JAK + 1; Y(I)←X(I,5);
          print JAK, (Y(J), J←1 to N)
Step 37   for J←1 to NO do step 38 od
Step 38   JAK←JAK + 1; print JAK, (X(I,3), I←1 to N)
Step 39   if NO > 1 then print 'note that there are' NO
          'design centre points' fi

Step 40   (Given the number of observations in the full design,
           JAK, and the last defining contrast to be tested
           (see chapter 3), KEST, randomise the integers 1 to
           JAK. Repeat if necessary.)
          print 'randomised observation order'
Step 41   IX←1 + 2 * and(8191, KEST)
          (see explanation on next page)
Step 42   XX←JAK        (integer to real)
Step 43   for I←1 to JAK do II(I)←0 od
Step 44   J←0
Step 45   J←J + 1
Step 46   if J > JAK then goto step 56 fi
Step 47   W←1./XX
Step 48   I←0
Step 49   I←I + 1
Step 50   if I > JAK then goto step 48 fi
Step 51   if II(I) ≠ 0 then goto step 49 fi
Step 52   call RANDU(IX,IX, YY)
          (the IBM Fortran subroutine RANDU is listed in appendix one)
Step 53   if YY > W then goto step 49 fi
Step 54   XX←XX - 1; JJ(I)←I; II(I)←1
Step 55   goto step 45
Step 56   print (JJ(J), for J←1 to JAK)
Step 57   if (another random number stream requested)
             then goto step 42   else stop fi
```

The explanation of step 41 is:
The subroutine RANDU for generating a uniformly distributed random number, Y, in the interval (0,1) needs to be seeded with an odd integer, IX, in the interval (0,32767). If a constant starting value were put into the program, the random number streams for all experimental designs with the same number of observations would all be the same. A different starting value for each occasion is created by the assignment in step 41. The last defining contrast to be tested, KEST, provides a different integer for each design. The integer 8191 is an array of bits with the rightmost 12 bits all set to one and the rest set to zero. By anding these two integers we chose the rightmost 12 bits of KEST which when multiplied by two and incremented by one ensures that IX is an odd integer approximately in the middle of the required interval.

The flowchart for algorithm AUGFAC follows.

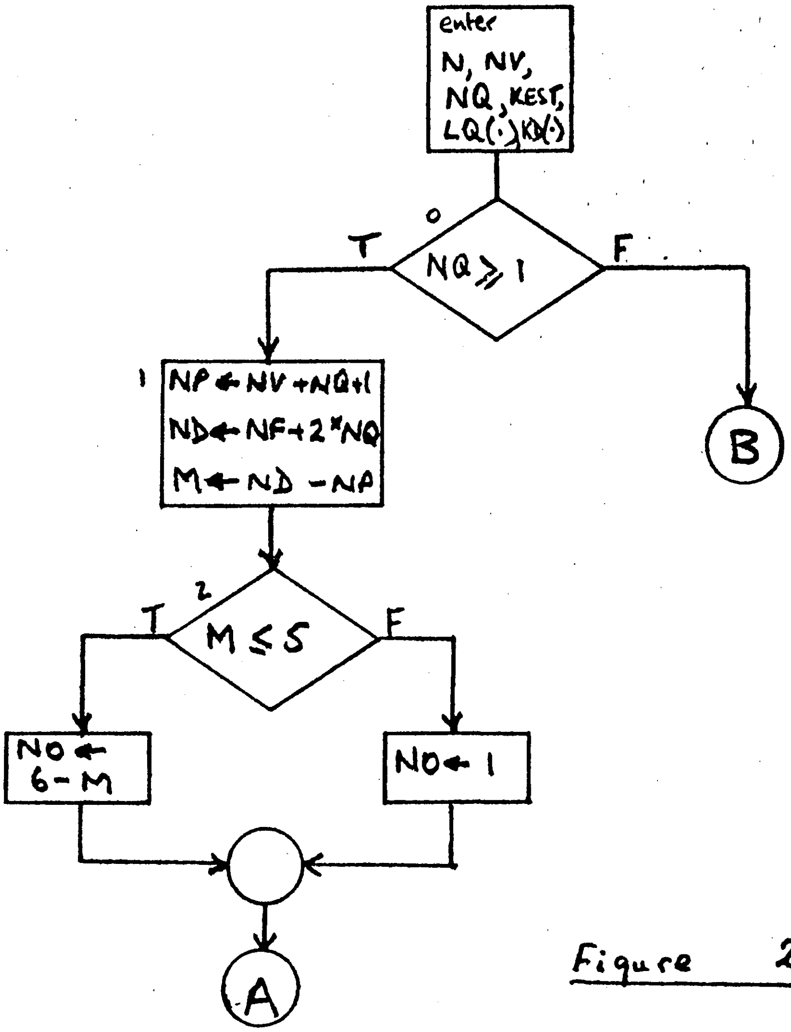


Figure 28

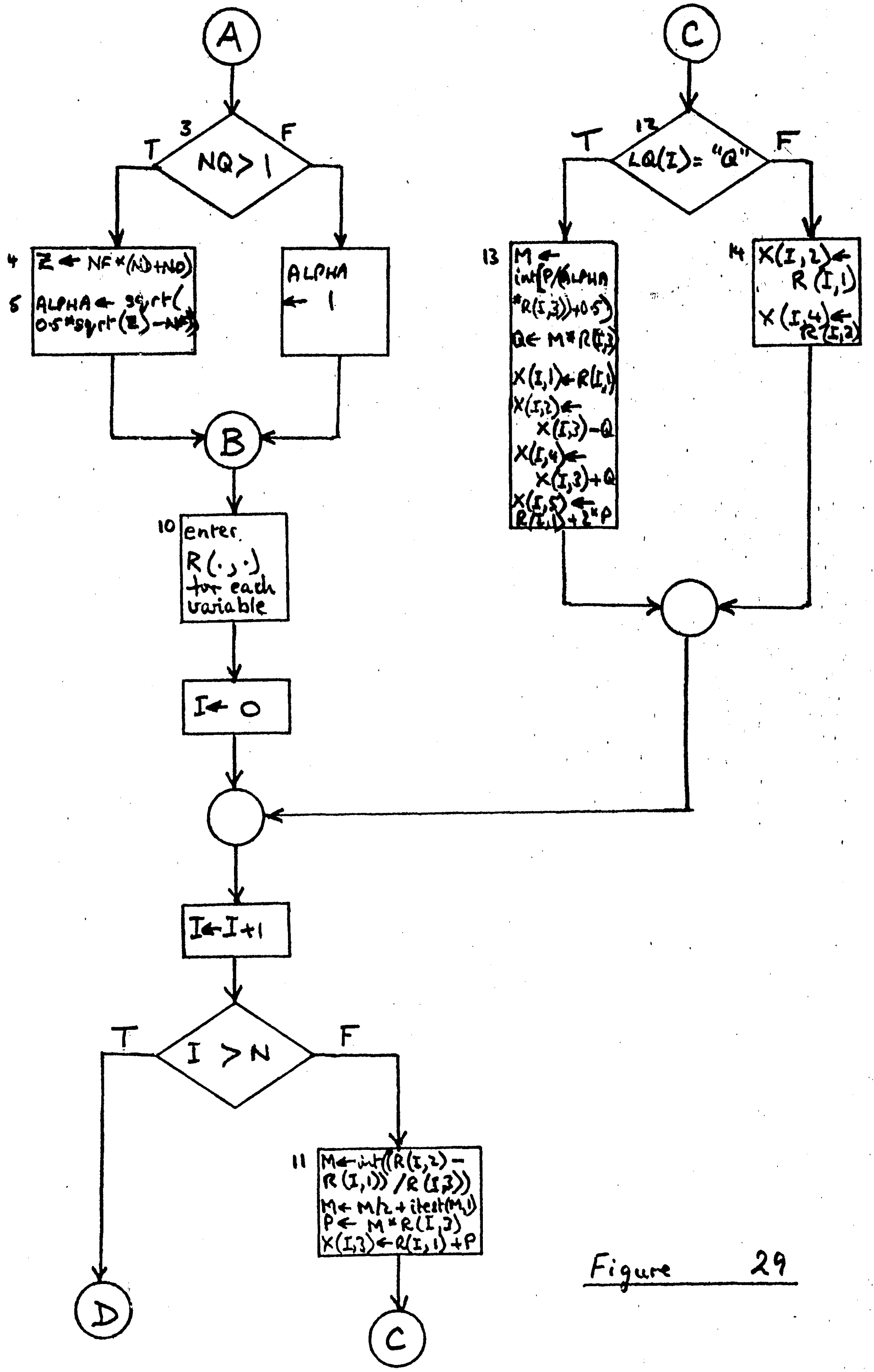

A
3
NQ > 1
T
F
4
Z ← NF*(ND+NO)
5
ALPHA ← sqrt(0.5*sqrt(Z)−NF)
ALPHA ← 1
B
10
enter R(·,·) for each variable
I ← 0
I ← I+1
I > N
T
F
11
M ← int((R(I,2) − R(I,1)) / R(I,3))
M ← M/2 + itest(M,1)
P ← M*R(I,3)
X(I,3) ← R(I,1) + P
D
C
C
12
LQ(I) = "Q"
T
F
13
M ← int[P/(ALPHA*R(I,3))+0.5]
Q ← M*R(I,3)
X(I,1) ← R(I,1)
X(I,2) ← X(I,3) − Q
X(I,4) ← X(I,3) + Q
X(I,5) ← R(I,1) + 2*P
14
X(I,2) ← R(I,1)
X(I,4) ← R(I,2)

Figure 29

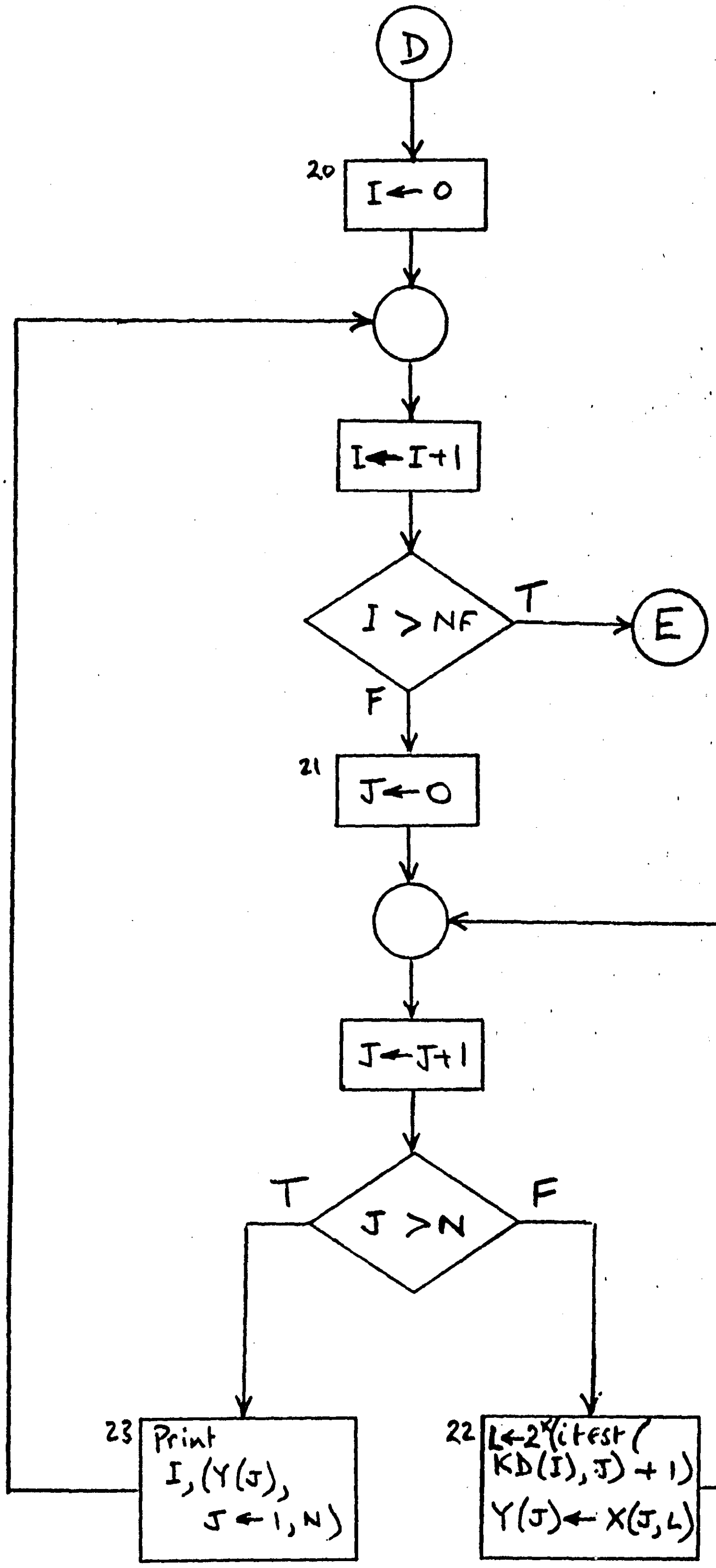

D
20
I ← 0
I ← I+1
I > NF
T
E
F
21
J ← 0
J ← J+1
J > N
T
F
23
Print
I, (Y(J),
J ← 1, N)
22
L ← 2*(itest(
KD(I), J) + 1)
Y(J) ← X(J, L)

Figure 30

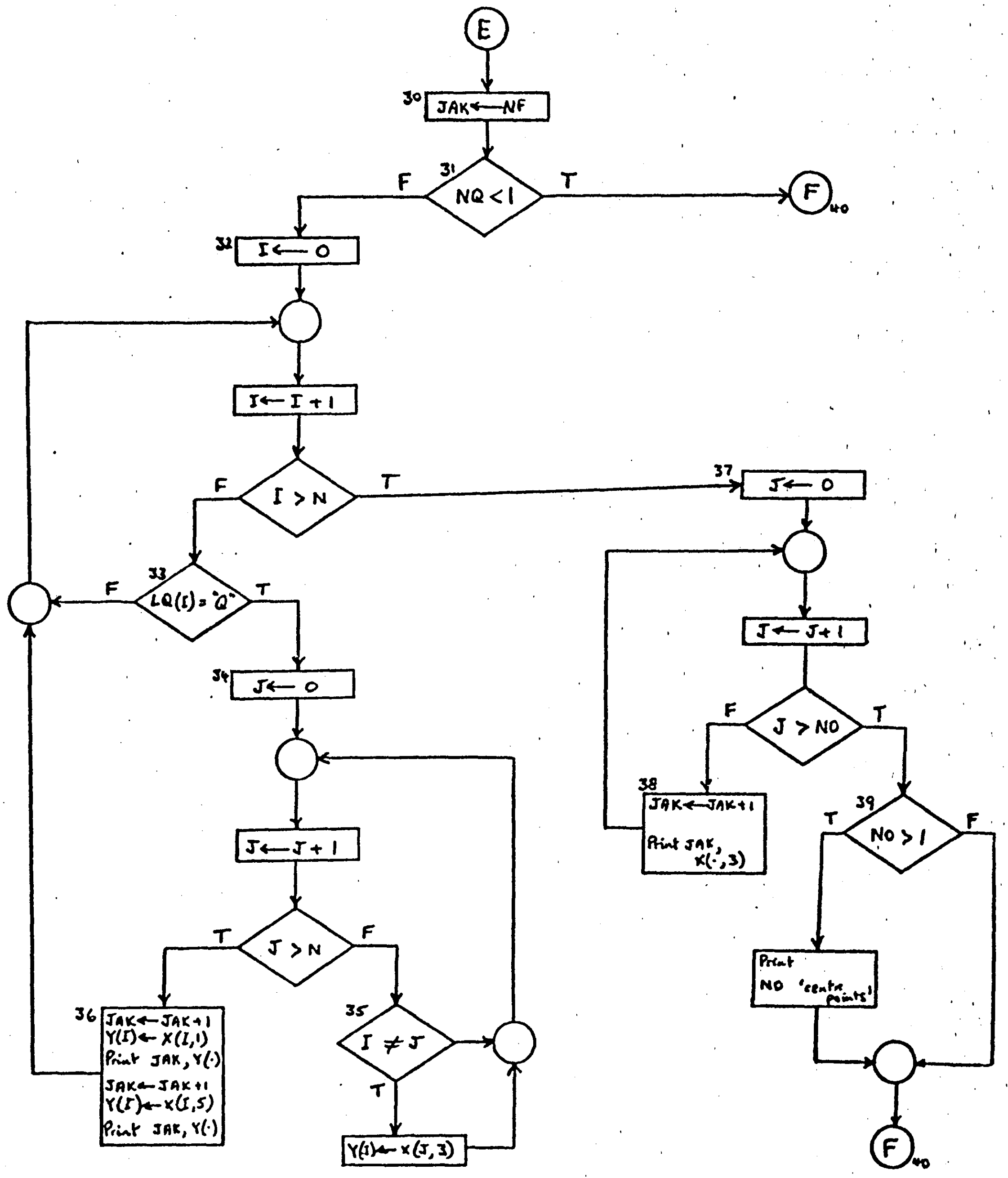
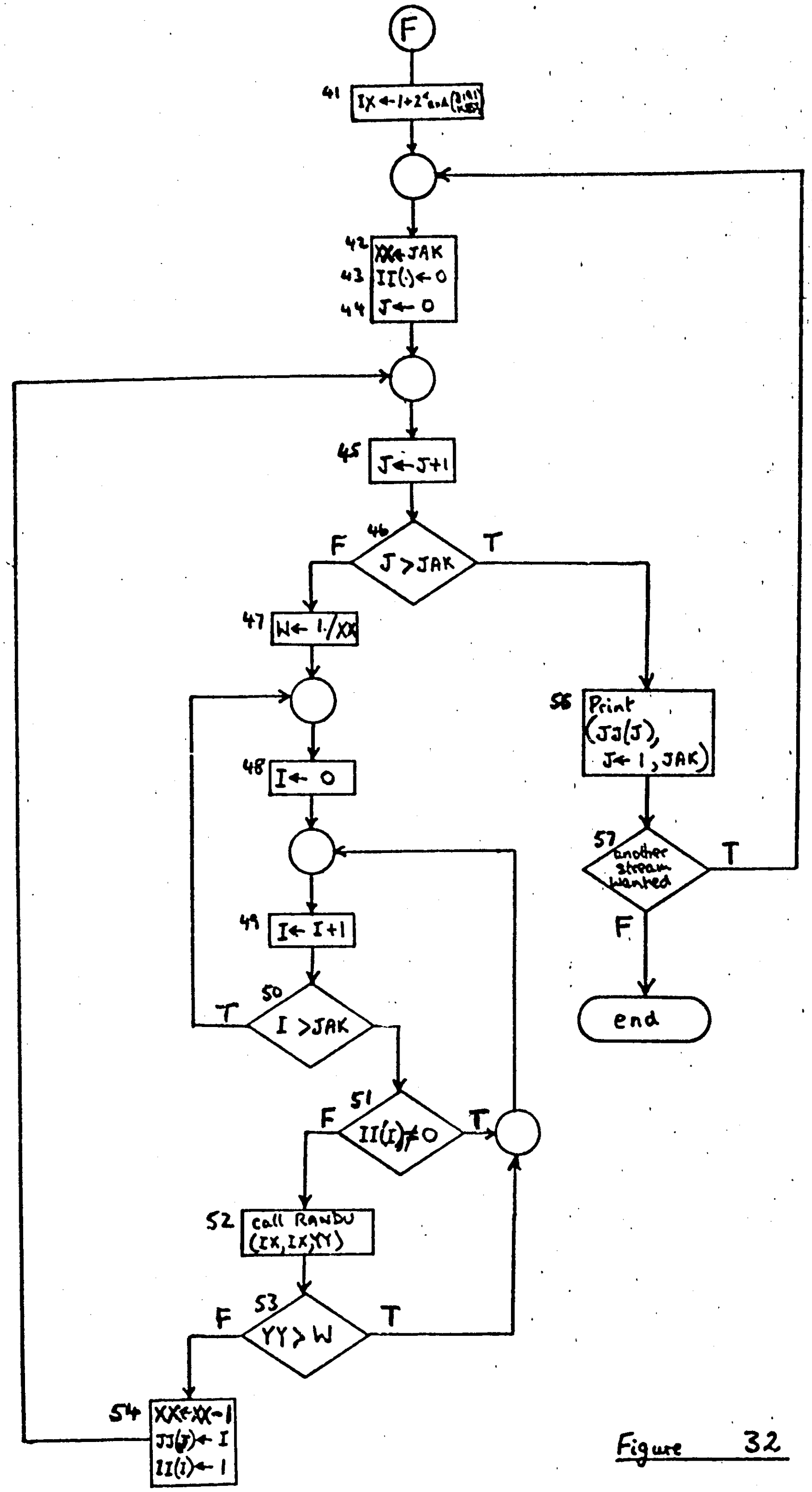

E
30
JAK ← NF
31
NQ < 1
F
T
F
40
32
I ← 0
I ← I + 1
I > N
F
T
37
J ← 0
33
LQ(I) = "Q"
F
T
34
J ← 0
J ← J + 1
J > NO
F
T
38
JAK ← JAK + 1
Print JAK, X(·, 3)
39
NO > 1
T
F
J ← J + 1
J > N
T
F
Print NO 'centre points'
36
JAK ← JAK + 1
Y(I) ← X(I, 1)
Print JAK, Y(·)
JAK ← JAK + 1
Y(I) ← X(I, 5)
Print JAK, Y(·)
35
I ≠ J
T
Y(I) ← X(J, 3)
F
40

Figure 31



F
41
IX ← 1+2^4
42
XX ← JAK
43
II(·) ← 0
44
J ← 0
45
J ← J+1
46
J > JAK
F
T
47
W ← 1./XX
48
I ← 0
49
I ← I+1
50
I > JAK
T
51
II(I) ≠ 0
F
T
52
call RANDU (IX, IX, YY)
53
YY > W
F
T
54
XX ← XX-1
JJ(J) ← I
II(I) ← 1
56
Print (JJ(J), J ← 1, JAK)
57
another stream wanted
T
F
end

Figure 32

A final stage in the development of the sequence of algorithms for quadratic designs is to link them back to those developed in chapter three for generating fractional two-level factorials. This calls for some simple changes to algorithm ENFAC and for algorithm AUGFAC to follow algorithm FRADE.

The alteration needed for algorithm ENFAC is to extend it so that the user will enter, for each variable, the name of the variable, whether or not a quadratic term is to be included (LQ(.) "Q"), and the range and increments of the variable values. The addition is:

```
Step 21   NQ←0
Step 22   for I←1 to N do step 23; step 24; step 25 od
Step 23   Print 'For variable' I 'type variable name,
          L or Q, least value, greatest value, and least interval'
          (the L or Q response referes to whether only linear
          effects are to be included in the model, or if quadratic
          terms are to be added)
Step 24   read (NAMVAR(I,J), for J←1 to 10), LQ(I),
         (R(I,J), for J←1 to 3)
Step 25   if LQ(I) = "Q" then NQ←NQ + 1 fi
Step 26   link to DEFCON
```

Since this modification is so simple the addition to the flowchart of algorithm ENFAC is omitted. A further small change to AUGFAC is needed to print the variable names as column headings.

3 Examples

The algorithms developed in chapters three and four have been implemented in Fortran and applied to many practical experiments involving the development of metallurgical processes and test procedures and the optimisation of material properties. Three examples are presented: the first deals with a simple quadratic model in two independent variables and one dependent variable; the second is much more complicated in that there are seven independent variables and a particular set of interactions and quadratic effects that have been specified for inclusion in the model; the third example shows how to deal with multiple responses.

<u>The first example</u> was chosen to portray a real situation in which there are only two independent variables and little is known about the relationships between them and the dependent variables. In fact, there were more than two independent variables but for the sake of a limited initial experiment all but two were held constant.

The experiment was part of a study of the rolling of powder into strip. The two control variables chosen for variation were:

x_1	roll gap (thousandths of an inch)
x_2	angle of powder feed plates to the vertical (degrees)

There were several dependent variables, representing resultant properties of the product and the behaviour of the mill. In the absence of prior knowledge about the relationships it was assumed that a simple linear effects model would not be adequate and that extra terms would be needed to express the interaction between the two variables and the curvature due to each one of them.

The quadratic model to be fitted was that of equation 4.2. It was agreed in advance with the experimenter that the two independent variables could range as follows:-

<u>GAP</u> from -40 thou to +60 thou in increments of one thou. The negative gap refers to the static screw setting before rolling begins. Clearly the physical gap cannot be less than zero. The increment of one thou is a constraint on the experimental design in that any attempt at specification of a gap with a precision smaller than one thou would not be realisable. The experimental design procedure copes with this constraint.

<u>ANGLE</u> from four degrees to 14 degrees in increments of one degree.

After all the above information was entered on the teletypewriter terminal in response to printed questions, the following results were printed:

```
ALIASING MATRIX FOR POWDER

    I
    A
    B
    AB

DESIGN IS
(1)  A  B  AB

EXPERIMENTAL DESIGN FOR POWDER
OBSERVATION        GAP       ANGLE
     1            -31.0        5.0
     2             51.0        5.0
     3            -31.0       13.0
     4             51.0       13.0
     5            -40.0        9.0
     6             60.0        9.0
     7             10.0        4.0
     8             10.0       14.0
     9             10.0        9.0
    10             10.0        9.0
    11             10.0        9.0
    12             10.0        9.0
NOTE THERE ARE  4  DESIGN CENTRE POINTS
RANDOMISED OBSERVATION ORDER
4 5 7 1 6 12 11 8 3 9 2 10
```

The interpretation of this printed output is:

1) The aliasing matrix demonstrates that all the main linear effects and required first order interactions will be independently estimable from the experiment. In this case there is no doubt since a full factorial is essential with only two independent variables: a fraction is not possible.

2) The factorial design, using standard notation, has the four points (1), a, b, ab. These correspond to the first four of the 12 observations in the fully augmented composite design.

3) Observations 5 to 8 have been added according to the computed value of α, and observations 9 to 12 are design centre points which, as well as contributing to the estimation of quadratic effects, also provide additional degrees of freedom for variance estimation.

Equation 4.15 gives an exact value of 1.21 for α for this design. However, the minimum increment constraints of the two variables detract slightly from perfect orthogonality. Thus, after scaling the variables so that their high and low values in observations 1 to 4 are +1 and -1, adding columns for the interactive and squared values, and applying the further transformations of equation 4.6, the cross products matrix becomes:

$$\begin{array}{lcccccc}
X_0 & 12 & & & & & \\
X_1 & 0 & 6.9743 & & & & \\
X_2 & 0 & 0 & 7.125 & & & \\
X_1X_2 & 0 & 0 & 0 & 4.0 & & \\
X_1{*} & 0 & 0 & 0 & 0 & 4.2982 & \\
X_2{*} & 0 & 0 & 0 & 0 & -0.1174 & 4.6621
\end{array}$$

For simplicity only the lower half of the symmetric matrix is shown. The capital X's denote the transformed variables. This slight loss of orthogonality is the penalty that must be paid for the incremental constraints of practical physical experiments although it can be avoided in computer simulated experiments.

The second example represents a hypothetical situation although it will be recognised as being typical of many real research situations. The problem is to establish a mathematical model to predict the hardness of a low alloy steel given seven independent variables, four of which are alloying elements and three are processing treatments. These variables are given in the following table together with ranges and increments of their values and whether the relationship between each variable and hardness is expected, from technical consideration, to be simply linear or quadratic over the experimental range.

Independent Variable	Q or L	Least Value	Greatest Value	Least Interval	Units
x_1 - Carbon	Q	0.1	0.5	0.05	Wt.%
x_2 - Chromium	L	0.2	3.0	0.01	Wt.%
x_3 - Molybdenum	L	0.01	0.5	0.01	Wt.%
x_4 - Vanadium	L	0.01	0.2	0.01	Wt.%
x_5 - Solution treatment temperature	Q	900.	1200.	5.0	°C
x_6 - Solution treatment time	L	0.5	1.0	0.01	hours
x_7 - Cooling rate to room temperature	Q	50.	6000.	5.0	°C/hour

The following interactions are expected to be effective:

Carbon and chromium	(x_1x_2 or AB)
Carbon and molybdenum	(x_1x_3 or AC)
Carbon and vanadium	(x_1x_4 or AD)
Carbon and cooling rate	(x_1x_7 or AG)
Vanadium and solution temperature	(x_4x_5 or DE)
Vanadium and solution time	(x_4x_6 or DF)

Thus the mathematical model to be fitted by the analysis of experimentally observed data is:

$$\begin{aligned} y = {} & a_0 + a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 + a_6x_6 \\ & + a_7x_7 + a_{11}x_1^2 + a_{22}x_2^2 + a_{33}x_3^2 + a_{44}x_4^2 + a_{55}x_5^2 \\ & + a_{66}x_6^2 + a_{77}x_7^2 + a_{12}x_1x_2 + a_{13}x_1x_3 + a_{14}x_1x_4 \\ & + a_{17}x_1x_7 + a_{45}x_4x_5 + a_{46}x_4x_6 + e \end{aligned}$$

where y represents the hardness, the a are coefficients of the model to be estimated, and e is experimental error. The above information is entered in response to the questions in ENFAC. The printed output is shown in the following tables.

Aliasing matrix

I	ABCF	BCE	AEF	CDG	ABDFG	BDEG	ACDEFG
A	BCF	ABCE	EF	ACDG	BDFG	ABDEG	CDEFG
D	ABCDF	BCDE	ADEF	CG	ABFG	BEG	ACEFG
AD	BCDF	ABCDE	DEF	ACG	BFG	ABEG	CEFG
B	ACF	CE	ABEF	BCDG	ADFG	DEG	ABCDEFG
AB	CF	ACE	BEF	ABCDG	DFG	ADEG	BCDEFG
BD	ACDF	CDE	ABDEF	BCG	AFG	EG	ABCEFG
ABD	CDF	ACDE	BDEF	ABCG	FG	AEG	BCEFG
C	ABF	BE	ACEF	DG	ABCDFG	BCDEG	ADEFG
AC	BF	ABE	CEF	ADG	BCDFG	ABCDEG	DEFG
CD	ABDF	BDE	ACDEF	G	ABCFG	BCEG	AEFG
ACD	BDF	ABDE	CDEF	AG	BCFG	ABCEG	EFG
BC	AF	E	ABCEF	BDG	ACDFG	CDEG	ABDEFG
ABC	F	AE	BCEF	ABDG	CDFG	ACDEG	BDEFG
BCD	ADF	DE	ABCDEF	BG	ACFG	CEG	ABEFG
ABCD	DF	ADE	BCDEF	ABG	CFG	ACEG	BEFG

Design is

I	AF	DG	ADFG	BEF	ABE	BDEFG	ABDEG
CEFG	ACEG	CDEF	ACDE	BCG	ABCFG	BCD	ABCDF

Observation	Carbon	Chromium	Molybdenum	Vanadium	Temperature	Time	Cooling Rate
1	0.15	0.20	0.01	0.01	930.0	0.50	665.0
2	0.45	0.20	0.01	0.01	930.0	1.00	665.0
3	0.15	0.20	0.01	0.20	930.0	0.50	5385.0
4	0.45	0.20	0.01	0.20	930.0	1.00	5385.0
5	0.15	3.00	0.01	0.01	1170.0	1.00	665.0
6	0.45	3.00	0.01	0.01	1170.0	0.50	665.0
7	0.15	3.00	0.01	0.20	1170.0	1.00	5385.0
8	0.45	3.00	0.01	0.20	1170.0	0.50	5385.0
9	0.15	0.20	0.05	0.01	1170.0	1.00	5385.0
10	0.45	0.20	0.05	0.01	1170.0	0.50	5385.0
11	0.15	0.20	0.05	0.20	1170.0	1.00	665.0
12	0.45	0.20	0.05	0.20	1170.0	0.50	665.0
13	0.15	3.00	0.05	0.01	930.0	0.50	5385.0
14	0.45	3.00	0.05	0.01	930.0	1.00	5385.0
15	0.15	3.00	0.05	0.20	930.0	0.50	665.0
16	0.45	3.00	0.05	0.20	930.0	1.00	665.0
17	0.10	1.60	0.03	0.11	1050.0	0.75	3025.0
18	0.50	1.60	0.03	0.11	1050.0	0.75	3025.0
19	0.30	1.60	0.03	0.11	900.0	0.75	3025.0
20	0.30	1.60	0.03	0.11	1200.0	0.75	3025.0
21	0.30	1.60	0.03	0.11	1050.0	0.75	50.0
22	0.30	1.60	0.03	0.11	1050.0	0.75	6000.0
23	0.30	1.60	0.03	0.11	1050.0	0.75	3025.0

Randomised observation order

21 18 1 20 12 14 6 16 11 7 23 17 13 10 15
9 8 4 3 22 19 5 2

The interpretation of the output is as follows:

1) The aliasing matrix in the first table demonstrates that the one-eighth factorial design, using only 16 observations out of a possible total of 128, will be adequate to estimate each of the main linear effects and each of the required interactive effects. The design to do this is expressed in the usual literal notation beneath the aliasing matrix.

2) The full design, with actual values specified for each independent variable at each design point, is displayed in the second table. This design is the basic fractional factorial printed under the first table augmented by six points to make possible the estimation of the three specified quadratic effects (carbon, solution treatment temperature, and cooling rate) and a 23rd point at the design centre to improve the estimation of observational variance.

3) The 23 numbered observations should be made in the order: 21, 18, 1, 20, 12, 14, 6, 16, 11, 7, 23, 17, 13, 10, 15, 9, 8, 4, 3, 22, 19, 5, 2.

Equation 4.15 gives a value for of 1.2616. However, the minimum increment constraints lead to practical values of for carbon, solution temperature, and cooling rate of 1.3333, 1.25, and 1.2606 respectively. The third value is very close the aimed value because the increments (5.0) are very small relative to the full range (50.0 to 6000.0).

After scaling the variables so that their high and low values in observations 1 to 16 are +1 and -1, adding columns for the interactive and squared values, and applying the further transformations of equation 4.6, the cross products matrix has zero off-diagonal elements except for relatively small values in the three positions shown here:

X_1*	5.694		
X_5*	-0.261	4.980	
X_7*	-0.306	0.053	5.059

<u>The third example</u> is not of a complete design problem as were the earlier examples. It simply serves to illustrate the procedure to be adopted when there are several dependent variables and some prior technical knowledge exists about the relationships to be expected between each of the dependent variables and the independent variables. The procedure is to write the model for each dependent variable in terms of the independent variables, and then add the models together.

Suppose there are five independent variables (x_1, x_2, x_3, x_4, x_5) and three dependent variables (y_1, y_2, y_3). The former may be alloying elements of steel, and the latter may be hardness, tensile strength and toughness, each of which could be represented as a function of the set of independent variables, expressed simply as follows:

	x_1	x_2	x_3	x_4	x_5	interactions
y_1	Q	L	L	L	L	x_1x_2, x_4x_5
y_2	L	L	Q	Q	L	x_1x_2, x_1x_3
y_3	Q	L	L	Q	L	x_2x_3

where the relationships L (linear), Q (quadratic) and the interactions are expected from previous experience or from theoretical studies to be significant terms in the respective models.

The way in which to use this set of relationships in the experimental design procedure is to assume a single general dependent variable (Y) with the following relationships:

	x_1	x_2	x_3	x_4	x_5	interactions
Y	Q	L	Q	Q	L	$x_1x_2, x_1x_3, x_2x_3, x_4x_5$

This results from choosing the higher order term in each column. Thus any design based on this composite model will permit the estimation of all parameters in the three separate models.

CHAPTER FIVE

ANALYSIS AND SIMULATION

1 Introduction

The criterion of orthogonality used in chapter four for the generation of quadratic designs has the advantage that the estimation of effects, or coefficients, and of the residual sums of squares, can be reduced to fairly simple formulae.

In practice, the data should eventually be submitted to the full least squares procedure for two reasons: first that continuous variables are measured in discrete steps which are used in the experimental design so that exact orthogonality is lost at that stage (this was shown in the examples of chapter four); second, that during the actual experiment it is rare for target values of control variables to be achieved exactly so that perfect orthogonality is also lost at this stage.

Nevertheless, it is sometimes useful for the experimenter to have a quick method of estimation before submitting his data to a full analysis. Also there is an exceptional situation where perfect orthogonality can be maintained: this is when the experiments are simulated on the computer. Examples of this will be given in the second section of this chapter.

In both these cases it is useful to have formulae for estimation of effects. Furthermore, as will be shown, some parts of the computation can be done at the design stage and therefore can be included in the earlier algorithms so as to save subsequent effort.

If we let $\underset{\sim}{y}$ be an n x 1 vector of observations, and let $\underset{\sim}{X}$ be the n x p design matrix, and $\underset{\sim}{\theta}$ be the p x 1 vector of coefficients to be estimated, then the linear model is:

$$E(\underset{\sim}{y}) = \underset{\sim}{X}\,\underset{\sim}{\theta} \tag{5.1}$$

and the least squares solution is:

$$\underset{\sim}{X}'\underset{\sim}{X}\underset{\sim}{\hat{\theta}} = \underset{\sim}{X}'\underset{\sim}{y} \tag{5.2}$$

where $\underset{\sim}{\hat{\theta}}$ are the least squares estimators of $\underset{\sim}{\theta}$.

Also, since $\underset{\sim}{X}$ is orthogonal, by virtue of the design method, $\underset{\sim}{X}'\underset{\sim}{X} = \underset{\sim}{D}$ a diagonal matrix. So

$$\underset{\sim}{\hat{\theta}} = \underset{\sim}{D}^{-1}\underset{\sim}{X}'\underset{\sim}{y} \tag{5.3}$$

The residual sum of squares, R, is

$$R = \underset{\sim}{y}'\underset{\sim}{y} - \underset{\sim}{y}'\underset{\sim}{X}\,(\underset{\sim}{X}'\underset{\sim}{X})^{-1}\,\underset{\sim}{X}'\underset{\sim}{y} \tag{5.4}$$

$$= \underset{\sim}{y}'\underset{\sim}{y} - \underset{\sim}{y}'\underset{\sim}{X}\,\underset{\sim}{D}^{-1}\,\underset{\sim}{X}'\underset{\sim}{y}$$

$$= \underset{\sim}{y}'\underset{\sim}{y} - \underset{\sim}{y}'\underset{\sim}{X}\,\underset{\sim}{D}^{-1}\,\underset{\sim}{D}\,\underset{\sim}{D}^{-1}\,\underset{\sim}{X}'\underset{\sim}{y} \tag{5.5}$$

Hence, from (5.3) and (5.5)

$$R = \underset{\sim}{y}'\,\underset{\sim}{y} - \underset{\sim}{\hat{\theta}}'\,\underset{\sim}{D}\,\underset{\sim}{\hat{\theta}} \tag{5.6}$$

Formulae may therefore be provided with the experimental design, for estimation of the coefficients in terms of a matrix $\underset{\sim}{A} = \underset{\sim}{D}^{-1}\,\underset{\sim}{X}'$ from equation (5.3), and for estimation of the residual sum of squares in terms of the matrix $\underset{\sim}{D}$ from equation (5.6). The experimental design program may be augmented to print $\underset{\sim}{D}$, $\underset{\sim}{D}^{-1}$, and $\underset{\sim}{A}$.

It should be remembered, from chapter four, that the independent variables were scaled to lie between $\pm\alpha$ or ± 1, depending on whether or not quadratic terms were included, and then the scaled quadratic terms were standardised by subtracting their mean values. Thus the coefficients estimated by equation (5.3) need rescaling if they are to be applied to raw data.

Rewriting $\underset{\sim}{y} = \underset{\sim}{X}\,\underset{\sim}{\hat{\theta}}$

as
$$\underset{\sim}{y} = C + \sum_{1}^{nv} f_i x_i + \sum_{1}^{nv} g_i x_i^* + \sum_{1}^{nv-1} \sum_{i+1}^{nv} h_{ij} x_i x_j \qquad (5.7)$$

where nv is the number of variables and

$$\underset{\sim}{\hat{\theta}} = \left\{ C,\ f_i,\ g_i,\ h_{ij} \right\}$$

and some g_i and some h_{ij} are zero if the quadratic terms and interaction terms to which they are attached are excluded from the model;

also
$$x_i^* = x_i^2 - \beta \qquad (5.8)$$

where β is the mean of the x_i^2, that is

$$\beta = \frac{nf + 2\alpha^2}{nf + 2nq + no} \qquad (5.9)$$

where *nf* is the number of rows in the basic fractional factorial, nq is the number of variables with quadratic terms, no is the number of design centre points, and α is as defined in chapter four.

Then, with obvious summation intervals omitted,

$$y = C + \sum f_i x_i + \sum g_i (x_i^2 - \beta) + \sum\sum h_{ij} x_i x_j \qquad (5.10)$$

$$= C' + \sum f_i x_i + \sum g_i x_i^2 + \sum\sum h_{ij} x_i x_j \qquad (5.11)$$

where
$$C' = C - \beta \sum g_i \qquad (5.12)$$

Now put
$$x_i = p_i + q_i w_i \qquad (5.13)$$

where
$$p_i = \frac{-\alpha(w_i(\max) + w_i(\min))}{w_i(\max) - w_i(\min)} \qquad (5.14)$$

and
$$q_i = \frac{2\alpha}{w_i(\max) - w_i(\min)} \qquad (5.15)$$

and the w_i are the independent variables in unscaled units, and the maxima and minima are data range limits.

Then equation (5.11) becomes:

$$\begin{aligned} y &= C' + \sum f_i(p_i + q_i w_i) + \sum g_i(p_i + q_i w_i)^2 \\ &\quad + \sum_{i<j}\sum h_{ij}(p_i + q_i w_i)(p_j + q_j w_j) \\ &= C' + \sum f_i p_i + \sum g_i p_i^2 + \sum_{i<j}\sum h_{ij} p_i p_j + \sum f_i q_i w_i \\ &\quad + \sum_{i\neq j}\sum h_{ij} p_i q_j w_j + \sum g_i q_i^2 w_i^2 + 2\sum g_i p_i q_i w_i \\ &\quad + \sum_{i<j}\sum h_{ij} q_i q_j w_i w_j \qquad (5.16) \\ &= K + \sum F_i w_i + \sum G_i w_i^2 + \sum_{i<j}\sum H_{ij} w_i w_j \qquad (5.17) \end{aligned}$$

where the rescaled coefficients K, F_i, G_i, H_{ij} can be calculated by comparison between equations (5.16) and (5.17).

In preparation for computing the rescaled coefficients, the experimental design program may be augmented further to print: β, all p_i, all q_i, all $p_i p_j$, all $q_i q_j$, and all $p_i q_j$.

These simple additions to the algorithms of chapter four will not be developed here, but they will be implemented in the program listings in the appendix.

Additionally, the following expressions may be used to compute the elements of the diagonal matrix D.

First element, corresponding to the mean $= n$ (5.18)

Each element corresponding to a main effect $= nf + 2\alpha^2$ (5.19)

Each element corresponding to an interaction $= nf$ (5.20)

Each element corresponding to a transformed quadratic term $= nf(1 - \beta)^2 + 2(\alpha^2 - \beta)^2 + (n - nf - 2)\beta^2$ (5.21)

2 Examples

The intention here is to show how the algorithms of chapters three and four, together with the analysis aids of the first section of this chapter, may be used to estimate the optimal conditions of a physical process which has been modelled mathematically in a form that defies optimisation by analysis.

The first example is a re-presentation of the powder rolling experiment described in chapter four. In fact, this was not computer simulated but it is chosen as a simple example to illustrate the procedure.

Suppose that the powder compaction process has been described dynamically in terms of partial differential equations representing the amount of compaction at every point in space and time as the powder descends between the feed plates and the rolls. Suppose further that these pde's have been translated into numerical procedures so that, given a roll gap setting and an angle of feed plates, the degree of powder compaction (y) as it emerges from between the rolls may be computed. Thus the computer may be used to simulate a real experimental process. We still need an experimental design and the composite designs introduced in chapter four will be suitable. However, we need not concern ourselves with errors of observation so the additional design centre points introduced in the earlier example are not needed. Without these, $\alpha = 1$. Also we need not be constrained by practical increments in the values of independent variables so we can achieve perfect orthogonality. Furthermore, there is no need for randomisation of the order of observations.

With these changes there will only be nine observations and, using the experimental design program with the extra features described in the first part of this chapter, we obtain the following data.

$\alpha = 1 \qquad \beta = \frac{2}{3}$

The scaled design matrix is $\underset{\sim}{X}$ =

observation	x_0	x_1	x_2	x_1x_2	x_1^*	x_2^*
1	1	-1	-1	1	$\frac{1}{3}$	$\frac{1}{3}$
2	1	1	-1	-1	$\frac{1}{3}$	$\frac{1}{3}$
3	1	-1	1	-1	$\frac{1}{3}$	$\frac{1}{3}$
4	1	1	1	1	$\frac{1}{3}$	$\frac{1}{3}$
5	1	-1	0	0	$\frac{1}{3}$	$-\frac{2}{3}$
6	1	1	0	0	$\frac{1}{3}$	$-\frac{2}{3}$
7	1	0	-1	0	$-\frac{2}{3}$	$\frac{1}{3}$
8	1	0	1	0	$-\frac{2}{3}$	$\frac{1}{3}$
9	1	0	0	0	$-\frac{2}{3}$	$-\frac{2}{3}$

The orthogonality is easily checked.

The diagonal cross-products matrix $\underset{\sim}{X}'\underset{\sim}{X}$ is

$$\underset{\sim}{D} = \mathrm{diag}(\ 9,\ 6,\ 6,\ 4,\ 2,\ 2\ )$$

so $\underset{\sim}{D}^{-1} = \mathrm{diag}(\ \frac{1}{9},\ \frac{1}{6},\ \frac{1}{6},\ \frac{1}{4},\ \frac{1}{2},\ \frac{1}{2}\ )$

Using ranges of roll gap and plate angle of (-40,60) and (4,14) the following values are obtained using equations (5.14) and (5.15):

$$p_1 = -0.2 \qquad p_2 = -1.8 \qquad q_1 = 0.02 \qquad q_2 = 0.2$$

Suppose now that the simulation program yields the following values of $\underset{\sim}{y}$, corresponding to observations 1 to 9:

$$\underset{\sim}{y}' = (2.0,\ 1.3,\ 2.0,\ 2.5,\ 2.0,\ 1.9,\ 4.5,\ 5.1,\ 4.8)$$

The vector $\underset{\sim}{X}'\underset{\sim}{y}$ is obtained by taking the inner product of each column of $\underset{\sim}{X}$ in turn with $\underset{\sim}{y}$. Thus

$$\underset{\sim}{X}'\underset{\sim}{y} = (26.1,\ -0.3,\ 1.8,\ 1.2,\ -5.7,\ 0\ )$$

so that from equation (5.3)

$$\hat{\underset{\sim}{\theta}} = (2.9,\ -0.05,\ 0.3,\ 0.3,\ -2.85,\ 0\ )$$

In the notation of equation (5.7)

$$C = 2.9 \qquad f_1 = -0.05 \qquad f_2 = 0.3 \qquad g_1 = -2.85$$

$$g_2 = 0 \qquad h_{12} = 0.3$$

From (5.12) $C' = 2.9 - \frac{2}{3}(-2.85) = 4.8$

The transformations indicated by equations (5.16) and (5.17) are

$$K = C' + f_1p_1 + f_2p_2 + g_1p_1^2 + g_2p_2^2 + h_{12}p_1p_2$$

$$F_1 = f_1q_1 + 2g_1p_1q_1 + h_{12}p_2q_1$$

$$F_2 = f_2q_2 + 2g_2p_2q_2 + h_{12}p_1q_2$$

$$G_1 = g_1q_1^2 \qquad G_2 = g_2q_2^2 \qquad H_{12} = h_{12}q_1q_2$$

which yield the following values:

$K = 4.264 \quad F_1 = 0.011 \quad F_2 = 0.048 \quad G_1 = -0.00114$

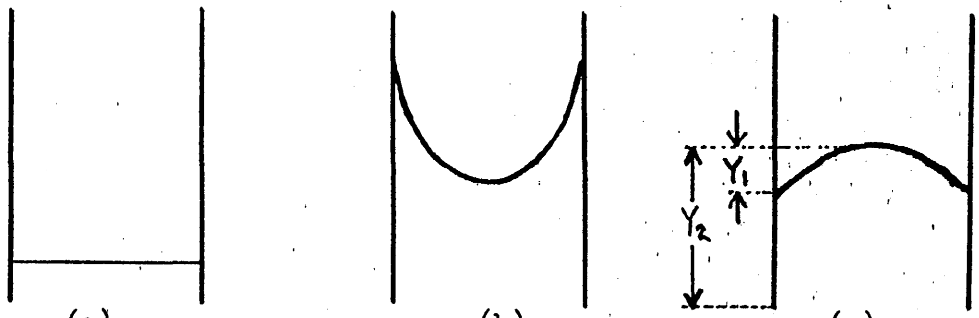
$G_2 = 0 \quad H_{12} = 0.0012$

These are the coefficients of a quadratic function in the two variables which may then be further analysed to predict the values of those two variables at which maximum compaction may be expected to occur in a real trial.

Clearly this is a fictitious simulation because with only two variables we should probably proceed immediately to a real trial rather than resort to the expense of mathematical modelling and computer simulation.

The second example, however, is a practical situation involving six independent, or control, variables. In this case the cost of a large number of observations needed to fit a full quadratic function was considered to be high enough to make preliminary mathematical modelling and simulation worth while.

The physical objective of the research was to produce a carbon distribution profile through the thickness of steel strip, such that the carbon composition at the centre of the strip was higher than that at the surface. Ideally the central composition should be about 0.7 per cent carbon. The purpose was to produce a hard sharp cutting edge when the strip was sharpened. The proposed method of achieving this was by two-stage diffusion. In figure 33 the surfaces of the strip are represented by the two vertical lines. In 33(a), the initial low level uniform distribution of carbon is shown by the horizontal line. After the first stage of diffusion, at a high temperature in an atmosphere of high carbon potential, the carbon distribution is as shown in 33(b). After the second stage, in an atmosphere of lower carbon potential, the carbon distribution should be as shown in 33(c).



(a) (b) (c)

Figure 33

A mathematical model of the process was developed by Pavlossoglou and Clay (1975) using Fick's diffusion law and a computer simulation using this model was developed by Pavlossoglou (1975). These were in terms of the physical properties of the materials and, in the simulation, small finite step lengths and time intervals were used. However, they enabled simulation experiments to be run on the computer to determine the optimal values of the operational variables. The optimal values were those which would make Y_1 in figure 33(c) as great as possible with Y_2 close to 0.7 per cent. The operational variables were:

X1	time of the first stage
X2	time of the second stage
X3	temperature of the first stage
X4	temperature of the second stage
X5	carbon potential of carburising gas, first stage
X6	carbon potential of carburising gas, second stage

As well as main effects and quadratic effects, first order interactions were expected between the following pairs:

X1X3, X1X5, X3X5, X2X4, X2X6, X4X6

The procedures of chapters three and four were used to produce the design tabulated in scaled variables on the next page. The design parameter values were

$$\alpha = 1.724432$$

$$\beta = 0.843274$$

Since the experiment was a simulation it was possible to use increments in the values of the independent variables that were small enough to correspond to the above values of parameters to six decimal places, thus preserving orthogonality. The ranges of the independent variables were:

X1	50,	100	minutes
X2	10,	30	minutes
X3	800,	1000	^{o}C
X4	800,	1000	^{o}C
X5	1.0,	1.5	percent carbon potential
X6	0.5,	1.0	percent carbon potential

These values were used in the simulations according to the scaled values in the tabulated design. The total time for each simulated trial was about half a minute. Each physical trial, had they been done, would have taken half a day plus the time to analyse the material to determine the carbon distributions.

The simulated results are also shown under the columns headed Y1 and Y2. Since most of the values of Y1 are negative, it is clear that the experimental region was misplaced. However, using the methods described earlier in this chapter, quadratic models in the operational variables were fitted to predict the dependent variables. These functions, together with some constraints (such as the need to keep the operational variables positive), were used to determine the conditions for further simulations which yielded the desired results.

row	X1	X2	X3	X4	X5	X6	Y1	Y2
1	-1	-1	-1	-1	-1	-1	-0.163	0.277
2	1	-1	-1	-1	-1	1	-0.184	0.346
3	-1	1	-1	-1	-1	1	-0.175	0.310
4	1	1	-1	-1	-1	-1	-0.089	0.373
5	-1	-1	1	-1	-1	1	-0.101	0.474
6	1	-1	1	-1	-1	-1	-0.037	0.605
7	-1	1	1	-1	-1	-1	-0.042	0.483
8	1	1	1	-1	-1	1	-0.077	0.616
9	-1	-1	-1	1	-1	-1	-0.076	0.389
10	1	-1	-1	1	-1	1	-0.093	0.458
11	-1	1	-1	1	-1	1	-0.047	0.476
12	1	1	-1	1	-1	-1	0.000	0.492
13	-1	-1	1	1	-1	1	-0.058	0.526
14	1	-1	1	1	-1	-1	-0.012	0.638
15	-1	1	1	1	-1	-1	-0.003	0.536
16	1	1	1	1	-1	1	-0.029	0.678
17	-1	-1	-1	-1	1	1	-0.236	0.308
18	1	-1	-1	-1	1	-1	-0.178	0.399
19	-1	1	-1	-1	1	-1	-0.111	0.347
20	1	1	-1	-1	1	1	-0.155	0.436
21	-1	-1	1	-1	1	-1	-0.068	0.569
22	1	-1	1	-1	1	1	-0.069	0.743
23	-1	1	1	-1	1	1	-0.092	0.587
24	1	1	1	-1	1	-1	0.018	0.753
25	-1	-1	-1	1	1	1	-0.117	0.462
26	1	-1	-1	1	1	-1	-0.070	0.534
27	-1	1	-1	1	1	-1	0.011	0.511
28	1	1	-1	1	1	1	-0.022	0.612
29	-1	-1	1	1	1	1	-0.054	0.632
30	1	-1	1	1	1	1	-0.032	0.789
31	-1	1	1	1	1	1	-0.028	0.872
32	1	1	1	1	1	-1	-0.054	0.632
33	$-\alpha$	0	0	0	0	0	-0.059	0.440
34	α	0	0	0	0	0	-0.026	0.614
35	0	$-\alpha$	0	0	0	0	-0.094	0.485
36	0	α	0	0	0	0	-0.063	0.505
37	0	0	$-\alpha$	0	0	0	-0.090	0.363
38	0	0	α	0	0	0	-0.001	0.757
39	0	0	0	$-\alpha$	0	0	-0.123	0.466
40	0	0	0	α	0	0	-0.022	0.594
41	0	0	0	0	$-\alpha$	0	-0.062	0.447
42	0	0	0	0	α	0	-0.039	0.595
43	0	0	0	0	0	$-\alpha$	-0.006	0.510
44	0	0	0	0	0	α	-0.095	0.531
45	0	0	0	0	0	0	-0.050	0.520

The results obtained by optimisation using the regression functions and checked by further simulation were:

X1 = 15 minutes
X2 = 20 minutes
X3 = 1100°C
X4 = 900°C
X5 = 1.3 % C
X6 = 0.1 % C

with simulated results:

Y1 = 0.132 % C
Y2 = 0.696 % C

These results were considered satisfactory in that the use of the recommended procedures had achieved the stated objective. A more detailed description was reported by Pavlossoglou and Greenfield (1975).

The Automatic Design of Experiments

Some Practical Algorithms

CHAPTER SIX

FRACTIONAL ASYMMETRIC MULTI-LEVEL FACTORIALS

1 Background

2 Algorithms

1 Background

As I mentioned in chapter two, industrial laboratories frequently arrange experiments based on qualitative variables for which there is no prior justification for ordering. Such variables are called factors and their different states are called levels.

These experiments are commonly devised by generating the full product of all the different levels of all the factors: the set of all possible combinations. This set may be replicated an arbitrary number of times and the observations may be made in a random order. When these total designs become too massive, the research worker may resort to the classical agricultural research artifacts of graeco-latin squares, lattices, split plots and balanced incomplete blocks.

The aim of this chapter is to present an extension of the methodology developed in chapter three for the generation of fractions of two-level factorials suitable for estimating a pre-specified set of required effects. The approach to be adopted is based on the group properties of factorial designs, which have been used by other workers. Original features here include the link between two-level factorials and multi-level factorials to provide the generators for fractions of the latter, and the general algorithmic approach. The method to be described does not always yield fractions that are small enough in terms of cost of experimentation. A method for further reduction will be developed in chapter seven.

As with two-level factorials, the search for fractional designs was preceded by the related problem of dividing full designs into blocks in such a way that block effects and high order interaction effects were confounded. Fisher (1943 and 1945) realised that there was an association between the theory of abelian groups and the relationships recognisable in the choice of interactions for confounding in 2^n experiments. He extended the association to produce block confounding of p^n experiments, where p is prime. However his method did not lead to asymmetric designs with non-prime levels, nor did it enable one to proceed from a predefined model, expressed as a requirements set, to a fractional design. Finney (1945) used the same notion for developing fractional replicates of p^n experiments, giving particular attention to 3^n arrangements. He showed that the set of defining contrasts was a sub-group and that once this had been determined the fractional design followed. He did not,however, suggest any way of determining the defining contrasts from the requirements set.

Kempthorne (1947) also restricted his attention to symmetric designs (those with the same number of levels for all factors) with a prime number of levels, but he made a useful contribution with the use of a modulo algebra to proceed from the set of defining contrasts to the experimental design. This method was, however, restricted to symmetric designs.

Asymmetric factorials (those which do not have the same number of levels for all factors) were considered by Plackett (1946) who usefully proved that a necessary and sufficient condition for the main effect estimates of two factors to be orthogonal is that the levels of one factor occur with each of the levels of the other factor with proportional frequencies. Thus the widely held assumption that each level of one factor must occur an equal number of times with each level of the other factor is incorrect.

The modulo algebra applied by Kempthorne was a special case of Galois field theory which had been used by Bose (1939) for the construction of graeco-latin squares. This method was used again by Kishen and Srivastava (1959) to generate asymmetric designs. Their method was to generate first a set of symmetric designs, with the number of levels in each not necessarily prime, and then to combine them together.

The algebraic approach was extended to asymmetric designs by White and Hultquist (1965) who used the theory of rings but again the number of levels for each factor was restricted to a prime. Worthley and Bannerjee (1974) went a stage further by combining elements from distinct finite rings which led to block confounded asymmetric designs with factors with non-prime numbers of levels.

A general review of techniques was published by Addelman (1963).

I developed my approach at about the same time as John and Dean (1975) who showed how to generate designs, both symmetric and asymmetric, given a set of group generators which will be described in the next section of this chapter. They showed how the confounding pattern could be determined from the generators and they listed some commonly needed designs with their generators and confounded interactions. Like earlier contributors, however, they did not offer a procedure going logically from model to generators to design.

The object of this chapter is to develop and describe a procedure which moves logically from the model (the experimental requirements) to a set of generators and hence to a balanced fraction of an asymmetric factorial.

2 Algorithms

The general asymmetric factorial experiment may be described by the notation

$$a^{\alpha} \, b^{\beta} \, c^{\gamma} \, . \, . \, . \, .$$

where α is the number of factors with 'a' levels, etcetera. Since, in my experience, there is a continual demand in research laboratories for fractional designs of this type, where the factors are qualitative, an algorithm has been developed to generate them, given first a model in terms of a set of effects that are required to be estimated.

The algorithm makes use of certain group properties which will be introduced with simple examples and then stated as formal requirements in the development of the algorithm.

The simplest class of group is the cyclic group which can be generated by one of its elements. A cyclic group G is said to be of order m if it has m elements. If x is the element which generates the group, then $G = \{1, x, x^2, . . . , x^{m-1}\}$ and $x^m = 1$.

The set of integers, modulo m, form a cyclic group under addition, with order m. For example, the cyclic group of order 2 may be represented by $G_2 = \{0, 1\}$. The element 1 generates the complete group since $1 + 1 = 0 \pmod 2$. Similarly, the cyclic group of order 3 may be represented by $G_3 = \{0, 1, 2\}$. Again, the element 1 generates the complete group since $1 + 1 = 2 \pmod 3$ and $2 + 1 = 0 \pmod 3$.

The cartesian product of these two cyclic group is obtained by taking all possible pairs, one from each group. Thus

$$G_2 \times G_3 = \{00, 10, 01, 11, 02, 12\}$$

where the first integer of each pair is an element of G_2 and the second is an element of G_3. Now, in this case, it can readily be seen that the cartesian product of these two cyclic groups is itself a cyclic group if the element 11 is used as a generator. Using the addition sign to indicate addition modulo 2 for the first integer at the same time as addition modulo 3 for the second integer, the element 11 generates the sequence:

$11 + 11 = 02 \qquad 02 + 11 = 10 \qquad 10 + 11 = 01$

$01 + 11 = 12 \qquad 12 + 11 = 00 \qquad 00 + 11 = 11$

However, it should be noted that the cartesian product of two cyclic groups of orders m and n is itself a cyclic group of order mn only when m and n are relatively prime. Thus, if $m = 3$ and $n = 4$ the cartesian product is a cyclic group of order 12, but if $m = 3$ and $n = 6$ the cartesian product is not a cyclic group of order 18. It is, however, an abelian group.

The notation introduced above is suitable for the representation of multi-factorial experimental designs. Thus, quoting John and Dean (1975), a treatment combination is denoted by the n-tuple $a = a_1a_2.....a_n$ where a_i is an integer between 0 and m_i-1 and where a_i corresponds to the (a_i+1)th level of the ith factor $(i = 1, \ldots, n)$. Addition of treatment combinations is defined as

$$a_1a_2....a_n + b_1b_2....b_n = c_1c_2....c_n$$

where $c_i = a_i + b_i \pmod{m_i}$ for $i = 1, \ldots, n$

In the above example, $n = 2$, $m_1 = 2$, $m_2 = 3$.

A full 2^n factorial can be expressed as the cartesian product of n cyclic groups each of order 2. For example, a 2^3 factorial is the cartesian product of (000, 100) x (000, 010) x (000, 001). The full design is generated by writing the two elements of the first cyclic group (000, 100), adding the generator of the second cyclic group to each of the previous elements in turn giving the sequence (000, 100, 010, 110), then adding the generator of the third cyclic group to each of the previous elements in turn giving the sequence (000, 100, 010, 110, 001, 101, 011, 111). This procedure is given here to introduce the general method and not as a recommendation for generating a full 2^n design which, as explained in chapter three, can most easily be generated by simply counting from 0 to $(2^n - 1)$ in binary.

The method described in chapter three for determining the fraction of a 2^n design for the estimation of a pre-specified set of requirements gives the following half of a 2^3 design for the estimation of main effects only: (000, 110, 101, 011). The element 110 is the order-2 generator of the cylic group (000, 110). Similarly, the element 101 is the order-2 generator of the cyclic group (000, 101). The cartesian product of these two cyclic groups is the set of four elements which constitute the half design.

In general, the generators of a 2^{n-k} design are those elements which are found in the rows of the design numbered $2^r + 1$ for integers $r = 0, \ldots, (n - k - 1)$.

Consider now a factorial with three factors: two with two levels each and the third with four levels. This would be described as a $2^2 \times 4$ factorial. If only main effects are to be estimated, the model would be:

$$y = m + \begin{Bmatrix} a_1 \\ a_2 \end{Bmatrix} + \begin{Bmatrix} b_1 \\ b_2 \end{Bmatrix} + \begin{Bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{Bmatrix} + \text{error}$$

with $a_1 + a_2 = b_1 + b_2 = c_1 + c_2 + c_3 + c_4 = 0$

The total number of effects to be estimated is 6: one mean, one main effect for each of the first two factors, and three main effects for the third factor. Since this is less than half the number of observations in the full factorial (16), it would be economical to find a half design. This may be done by taking the two generators of the half 2^3 design and proceeding as before, except that the third integer will be reduced modulo 4 instead of modulo 2. The two generators and the cyclic groups they generate are:

110 → 000, 110 (order 2)
101 → 000, 101, 002, 103 (order 4)

The cartesian product of these two cyclic groups is obtained by adding all pairs of elements, one from each group, to produce a set of eight elements, which also happens to be a group but not a cyclic group:

000, 110, 101, 011, 002, 112, 103, 013

This half design will permit the estimation of the mean and the main effects.

Consider now another factorial with three factors: each with three levels: a 3^3 factorial. If only main effects are to be estimated, the total number is 7: one mean and two main effects for each of the three factors. This suggests that a suitable fraction of the 3^3 factorial would be a third design. This may be done by taking the two generators of the half 2^3 design and proceeding as before, except that each integer will be reduced modulo 3 instead of modulo 2. The two generators and the cyclic groups they generate are:

110 → 000, 110, 220 (order 3)
101 → 000, 101, 202 (order 3)

The cartesian product of these two cyclic groups is obtained by adding all pairs of elements, one from each group, to produce a set of nine elements, which is also a group but not a cyclic group:

000, 110, 220, 101, 211, 021, 202, 012, 122

This third design will permit the estimation of the mean and the main effects.

The procedure illustrated above is described more generally in the following algorithm:

Algorithm MULFAC (Multi-level asymmetric FACtorial)

Step 0 Enter the number of factors, the number of levels for each factor, and the requirements in terms of the main effects and interactions that need to be estimated.

Step 20 Assume that each factor has only two levels and use the method of chapter three to design a suitable $2^{n - k}$ design.

Step 40 Using algorithm NEW (chapter three) select the (n - k) generators.

Step 60 Generate the (n - k) cyclic groups, using modulo m_i for the ith integer in each generator where m_i is the number of levels for the ith factor.

Step 80 Generate the fractional design by taking the cartesian product of these (n - k) cyclic group.

Since the number of elements in the fractional design will be the product of the orders of the (n - k) cyclic generators, it sometimes happens that the fractional design may still have too many elements to satisfy the required experimental economy. in this case, after applying all of algorithm MULFAC, a further procedure, to be developed in chapter seven, will be applied to select the best subset of the elements in the fractional design. It also sometimes happens that the fractional design generated by algorithm MULFAC is exactly equal to the full design. This situation may be determined after step 40 by calculating the order of each generator and testing if the product of the orders is equal to the product of the numbers of levels of all the factors. In this case the algorithm will immediately switch to the procedure to be developed in chapter seven to select the best subset of the elements in the full design. The criterion for 'best' will be discussed. The reason that the chapter seven procedure is not used generally instead of MULFAC is that it is very much slower.

The following elementary group theory is needed to develop the algorithm MULFAC:

1. A multi-level factorial experiment can be expressed as a product group G, created by the cartesian product of the cyclic groups, each representing a factor. The order of each constituent cyclic group is the number of levels in the factor which that constituent cyclic group represents. The order of the product group G is the product of the orders of the constituent cyclic groups.

$$\begin{aligned} G &= \langle L_1, L_2, \ldots\ldots, L_n \rangle. \\ &= C_{L_1} \times C_{L_2} \times \ldots\ldots \times C_{L_n} \end{aligned}$$

2. The order of any element in a cyclic group is a divisor of the order of the group. Not all elements have the same order. Let the cyclic group be C_L, then an element of C_L is $(x \mid x \in \{0, 1, \ldots, L-1\})$, and the order of x is $O(x)$.

1. if $x = 0$ then $O(x) = 1$
2. if x is prime to L then $O(x) = L$
3. if $x \leqslant \frac{L}{2}$ then $O(x) = \frac{L}{x}$
4. if $x > \frac{L}{2}$ then $O(x) = \frac{L}{L - x}$

These all reduce to: $O(x) = {}^{L}/_{hcf(L,x)}$

It will be remembered here that an algorithm for computing the highest common factor of a pair of integers was developed by way of illustrating the algorithmic procedure in chapter one.

As well as using the notation $O(x)$ to represent the order of an element, the following more precise notation may also be used: $O(x(y))$ to denote the order of an element x from a cyclic group of order y.

3. The order of an element in a cartesian product of cyclic groups, or the order of an element in a cyclic group which has integers representing more than one factor, is the lowest common multiple (lcm) of the orders of the cyclic constituent elements.

Example:

For the group $G = \langle 2, 3, 6 \rangle$ the order of the element $x = (1,2,4)$ is

$$\begin{aligned} O(1,2,4) &= \text{lcm}(O(1(2)), \ O(2(3)), \ O(4(6))) \\ &= \text{lcm}\ (1, 3, 3) \\ &= 3 \end{aligned}$$

4. Given a full mixed factorial design F represented by a product cyclic group $G_\alpha = \langle a, b, c, \ldots \rangle$

where $\alpha = \prod a\ b\ c \ldots$ is the order of G

then all balanced fractions $\{f\}$ of F may be represented by proper sub-groups of G, with properties to be described, or by cosets of those sub-groups which may be regarded spatially as rotations of those sub-groups. Thus the initial problem may be reduced to the search for proper sub-groups with the required properties. I use the notation g_β to represent a sub-group of G_α with order β and with the required properties.

5. The required properties of g_β are:

5.1 The order of g_β is

$$\beta \geqslant 1 + (a-1) + (b-1) + (c-1) + \ldots$$

if we are considering a factorial experiment for the estimation of mean plus main effects only, without interactions. That is: the design must contain at least as many points of observation as the number of independent coefficients to be estimated.

If, for example, there are three factors A, B, C, with a, b, c levels respectively, and the interaction between A and B is to be estimated, then the order of g_β must be:

$$\beta \geqslant 1 + (a-1) + (b-1) + (c-1) + (ab-1)$$

5.2 β divides α

This is Lagrange's theorem: the order of a subgroup g of a finite group G divides the order of G.

5.3 β is divisible by the order of each constituent cyclic group representing each factor. It is obvious from the method of constructing the subgroup g_β that β must be divisible by the order of each constituent cyclic group representing each generator.

Using the notation B/A to indicate that A is divisible by B, then from 5.2 and 5.3:

$$\beta/\alpha, \ a/\beta, \ b/\beta, \ c/\beta \ .$$

or $\quad \beta/\alpha$ and $(x \mid x \in \{a, b, c, \ldots\})/\beta$

Hence β is chosen by first finding the lowest integer that satisfies 5.1 and then finding the nearest integer above or equal to it that is a multiple of the lcm of the factor levels.

5.4 Each factor must be represented by a non-zero integer in at least one of the generators of g_ρ. If this were not so, that factor would not be included in the fractional design except at a single level.

5.5 In at least one of the generators in which a factor is represented by a non-zero integer, the order of that integer must be co-prime/(relatively prime) to the order of the factor. If this were not so, that factor would not be included in the fractional design at all of its defined levels.

For example, for the group $G = \langle 2, 3, 6 \rangle$ there must be at least one generator among the generators of a subgroup balanced in the third factor with a 1 or a 5 as the third integer so as to achieve that balance.

5.6 If several factors have an equal number of levels, then in the set of generators all pairs of those factors must occur at relatively prime levels. If this were not so, the effects of those factors would be confounded. The simplest way to achieve this is to ensure that in the set of generators, all except one of these factors occur at least once at the zero level.

For example, for the group $G = \langle 2, 2, 2 \rangle$ generators for a half design are 110 and 101. Thus of the three factors two of them occur at least once at the zero level.

We have already noted that generators selected at step 40 of algorithm MULFAC may have orders such that their product is too great for experimental economy. Alternative and equivalent generators with smaller orders may however be obtained by examination of the cosets of the initial fractional two-level design which, as we also noted, may be regarded spatially as rotations of the proper sub-group. These cosets are obtained by taking the product of the proper sub-group representing the fractional two-level design and elements of the group not in the sub-group.

For example, consider the group $G_{36} = \langle 2, 3, 6 \rangle$ and the object of obtaining a fractional design that will estimate main effects only. The aim is to find generators for the sub-group g_{β}. First a suitable minimum value of β must be found.

from 5.1 $\beta \geqslant 7$
from 5.2 $\beta = 9$ or 12 or 18
from 5.3 $\beta = 12$ or 18

from paragraph (3) (referring to the order of an element in a cartesian product of cyclic groups) we compute that the cyclic sub-group of highest order has order lcm(2,3,6) which is 6; therefore there is no cyclic sub-group of order 12.

Orders of the generators from the $\frac{1}{2}\ 2^3$ factorial when applied to the (2,3,6) factorial are (from property (3)):

$O(1,1,0) = \mathrm{lcm}(O(1(2)),\ O(1(3)),\ O(0(6))) = 6$
$O(1,0,1) = \mathrm{lcm}(O(1(2)),\ O(0(3)),\ O(1(6))) = 6$
and the product of these is 36 which is the size of the full factorial.

Cosets of the $\frac{1}{2}\ 2^3$ factorial subgroup (000, 110, 101, 011) are obtained by taking the following products:
(100) x (000, 110, 101, 011) ⟶ (100, 010, 001, 111)
(010) x (000, 110, 101, 011) ⟶ (010, 100, 111; 001)
(001) x (000, 110, 101, 011) ⟶ (001, 111, 100, 010)
Clearly these cosets are identical, except in the orders of the elements. Indeed this must be so in the case of a half sub-group since its coset must be the set of elements of the full group when the sub-group is removed. Of importance, however, are the elements in the generator (2nd and 3rd) positions. Those from the first coset are not suitable, by property 5.4. However, those from the second coset are, and their orders, using property (3), are:
$O(1,0,0) = \mathrm{lcm}(O(1(2)),\ O(0(3)),\ O(0(6))) = 2$
$O(1,1,1) = \mathrm{lcm}(O(1(2)),\ O(1(3)),\ O(1(6))) = 6$
and the product of these is 12 which is the desired size of

the fractional factorial. The cyclic groups which these elements generate are:

100 $\rightarrow$ 000, 100 (order 2)
111 $\rightarrow$ 000, 111, 022, 103, 014, 125 (order 6)

The cartesian product is:

000, 111, 022, 103, 014, 125
100, 011, 122, 003, 114, 025

This fractional (one third) design is adequate to estimate the mean and eight (1 + 2 + 5) main effects.

Another useful point in designing the algorithm is that since the factor levels are qualitative, they are order independent. Hence the choice of generators can be simplified to those containing factor levels 0 and 1. On the other hand, other levels with lower orders may be indicated by examination of the cosets as described. The consideration of cosets is used in algorithm SELG (selection of alternative generators) which is developed on a later page.

Apart from the main algorithm, we already have one for hcf, but we also need once for lcm. This may be developed using the following considerations:

1. The lcm of a set of integers must be equal at least to the largest integer in the set.
2. If the lcm is greater than the largest integer in the set, then it must be equal to an integer multiple of the largest integer in the set.
3. Every integer in the set must divide the lcm.

Thus the outline algorithm is:

Algorithm LCM (Lowest Common Multiple)

```
Step 0   Enter a set of integers I(.) of length NN
Step 10  Find the largest integer in the set and allocate
         this value to two integer variables M and LCM
Step 20  For each Kth integer, I(K), not yet included in the
         lcm, determine if I(K) divides LCM. If it does, then
         it is included in the lcm. If it does not, then
         increase LCM by M.
         Repeat step 20 until all the integers have been
         included in the lcm.
```

In more detail, the algorithm becomes:

Algorithm LCM (Lowest Common Multiple)

```
Step  0   Enter a set of integers  I(.)  of length  NN
Step  1   set  N←NN;  J←0;  LCM←0;  K←0
Step 10   set  K←K + 1
Step 11   if  K > N  then goto step 20  fi
Step 12   if  I(K) ≤ LCM  then goto step 10  fi
Step 13   set  LCM←I(K);  MM←K;  goto step 10
Step 20   set  K←MM
Step 21   set  J←ionbt(J,K);  M←LCM;  N←N - 1
Step 22   if  N = 0  then return  fi
Step 23   set K←0
Step 25   set K←K + 1
Step 26   if  K > NN  then return  fi
Step 27   if  itest(J,K) = 1  then goto step 25  fi
Step 28   if  LCM/I(K)*I(K) = LCM  (I(K) divides LCM)  then goto step 21 fi
Step 29   set  LCM←LCM + M;  goto step 28
```

A function itest(J,K) is used to test if the Kth bit of J is set to 1; itest returns a value of 1 if it is, or 0 if it is not. The flowchart for algorithm LCM is in figure 34.

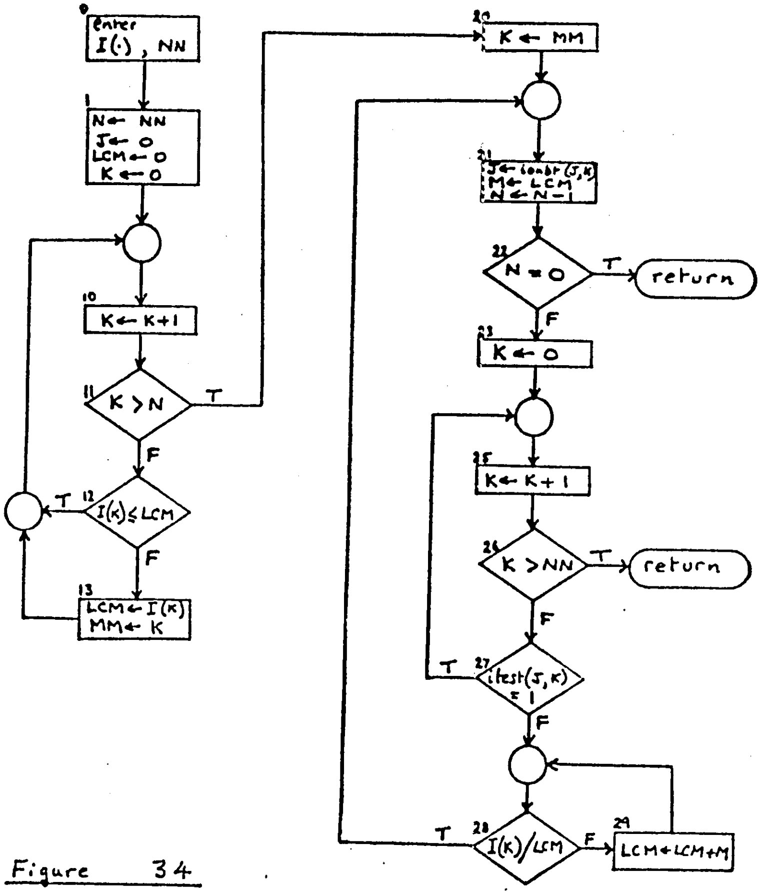


Figure 34

Two further simple functions are needed. One of them (JORD) computes the order of an element L in a cyclic group of order N. The other (MOD) computes the sum of two integers, J and K, modulo L. The algorithms are:

Algorithm JORD (ORDer of an element L from a cyclic group of order N)

```
Step 1   Enter L and N
Step 2   If  L = 0  then  JORD ← 1
                    else  JORD ← N/hcf(L,N)  fi
```

Algorithm MOD (addition of J and K MODulo L)

```
Step 0   enter  J, K and L
Step 1   set  MOD ← J + K
Step 2   if  MOD ≥ L  then set  MOD ← MOD - L  fi
```

A more complex procedure that is needed is to solve the following problem: Given a set of generators and the number of levels of each factor, determine the level of each factor for the Kth observation. This requires products of generators as follows:

Algorithm LEV (determine all factor levels for K-th observation)

Step 100 Suppose the first generator has order IL(1). Then by the time we have reached the K-th observation it will have been cycled L times where
L is the integer value of (K - 1)/IL(1)
The number of observations passed in those L cycles of the first generator is L*IL(1)
Therefore we are now at the M-th level of the first generator, where M is K - L*IL(1)

Step 200 Consider the effect of M increments of the first generator on the levels of the first factor.
The first integer (corresponding to some level of the first factor) which occurs in the first generator is ID(1,1). Usually it has value 0 or 1 but it may be some other integer value.

Hence the unmodulated level of the first factor in the current ((L + 1)th) cycle of the first generator is KK, where KK is M * ID(1,1)
If NL(1) is the number of levels of the first factor, then the modulated level of the first factor is LL, where LL is KK - NL(1) * (KK/NL(1))

The second generator passes through one level for each complete cycle of the first. Thus whereas in step 100 we were considering the K-th observation so far as the first generator was concerned, we may now consider the (L + 1)th observation so far as the second generator is concerned. Therefore at the end of step 100 we could set K ← L + 1 to prepare for dealing with the second generator. This will lead, in step 100, to determining a new value of M, where M is the M-th level of the second generator.

Hence, in step 200, a new modulated level (LL) of the first factor, due to M increments of the second generator, will be determined. This must be combined, using algorithm MOD, with the previously computed modulated level of the first factor due to the first generator. In order to generalise this procedure, the levels of all factors are set to zero before step 100.

The number of observations (NO) in the design is the product of the orders of all the generators. If the procedure described so far is carried through to deal with all the generators, it will lead to the final (NO-th) observation having all factors at the zero level. For a reason which will be stated in chapter seven, it is desirable to have this as the first observation. Therefore in algorithm LEV we shall enter a request for the factor levels of the I-th observation, then use the procedure described to compute the factor levels of the K-th, where K is I - 1. If I = 1 then the algorithm will return all factor levels set to zero.

The full algorithm follows.
The flowchart is in figure 35.

Algorithm LEV (determine all factor LEVels for the I-th observation in the design: return them in array IK(.))

```
Step   1   Enter I; N (number of factors); NG (number of
           generators); IL(.) (the order of each generator);
           ID(.,.) (the generators); NL(.) (the number of
           levels of each factor).
Step   2   for J←1 to N set IK(J)←0
Step   3   if I = 1 then return fi
Step   4   set K1←I - 1

Step 100   set J←0
Step 101   set J←J + 1
Step 102   if J>NG then return fi
Step 103   set L←(K1- 1)/IL(J); M←K1- IL(J)*L; K1←L + 1

Step 200   set JJ←0
Step 201   set JJ←JJ + 1
Step 202   if JJ>N then goto step 101 fi
Step 203   set KK1←ID(J,JJ)*M; L←NL(JJ); LL←KK1- L*(KK1 / L)
Step 204   set IK(JJ)←MOD(IK(JJ), LL, L)
Step 205   goto step 201
```

Another algorithm that will be useful is FASET, which determines the subsets of factors with equal numbers of levels. If the experiment has N1 factors with P1 levels, N2 factors with P2 levels, etcetera, which may have been entered in any order through algorithm ENFAC, then algorithm FASET will give values to an array IX(. , .) as follows;

IX(1,1) = P1   IX(1,2) = N1   IX(1,3) = 0   IX(1,4) = 0
IX(2,1) = P2   IX(2,2) = N2   IX(2,3) = 0   IX(2,4) = 0
etcetera

The zero values in the third and fourth elements of each row will be described in the full description of the main algorithm MULFAC.

The full algorithm FASET follows.
The flowchart is in figure 36.

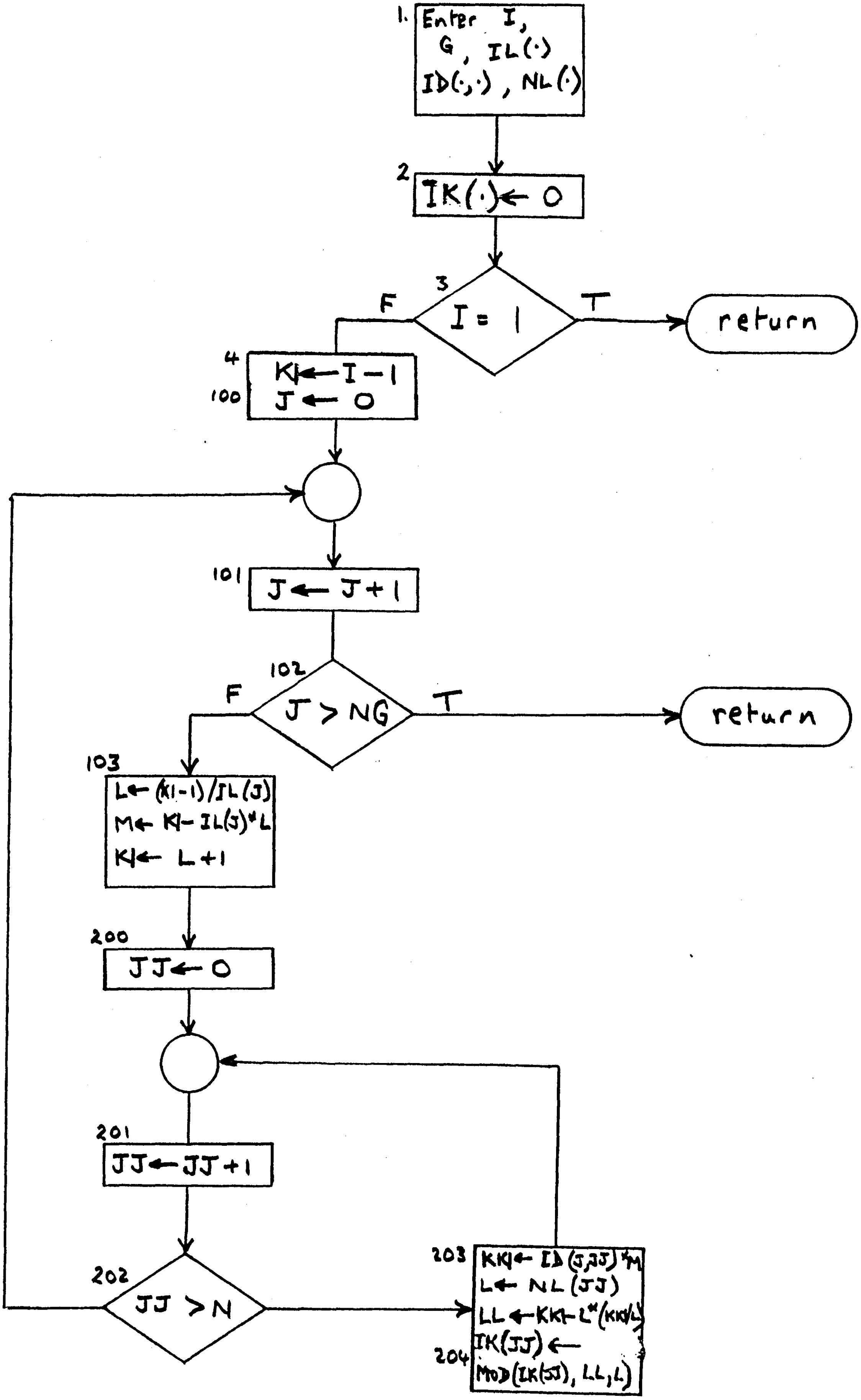

1.
Enter I, G, IL(·) ID(·,·), NL(·)
2
IK(·) ← 0
3
I = 1
F
T
return
4
KI ← I − 1
100
J ← 0
101
J ← J + 1
102
J > NG
F
T
return
103
L ← (KI−1)/IL(J)
M ← KI − IL(J)*L
KI ← L+1
200
JJ ← 0
201
JJ ← JJ+1
202
JJ > N
203
KKI ← ID(J,JJ)*M
L ← NL(JJ)
LL ← KKI−L*(KKI/L)
IK(JJ) ←
204
MOD(IK(JJ), LL, L)

Figure 35

Algorithm FASET (FActor subSETS) (return with IX(I,1) = modulus of I'th subset of factors, IX(I,2) = number of factors in I'th subset, IX(I,3) = 0, IX(I,4) = 0, LI = number of subsets, JG = number of subsets with IX(I,1) > 2)

```
Step 0   enter N, NL(.)
Step 1   set L← 0; LI←0; IXI←0; JG← 0
Step 2   set IXI← IXI + 1
Step 3   set J← 0
Step 4   set J←J + 1
Step 5   if J = IXI then goto step 7 fi
Step 6   if NL(J) = NL(IXI) then goto step 2 else goto step 4 fi
Step 7   set LI← LI + 1; IX(LI,1)←NL(IXI); IX(LI,2)←1;
             IX(LI,3)← 0; IX(LI,4)← 0; L←L + 1
Step 8   if L = N then return fi
Step 9   set J←J + 1
Step 10  if J > N then goto step 14 fi
Step 11  if IX(LI,1) ≠ NL(J) then goto step 9 fi
Step 12  set IX(LI,2)←IX(LI,2) + 1; L← L + 1
Step 13  if L ≠ N then goto step 9 else return fi
Step 14  if IX(LI,2) > 2 then set JG←JG + 1 fi
Step 15  goto step 2
```

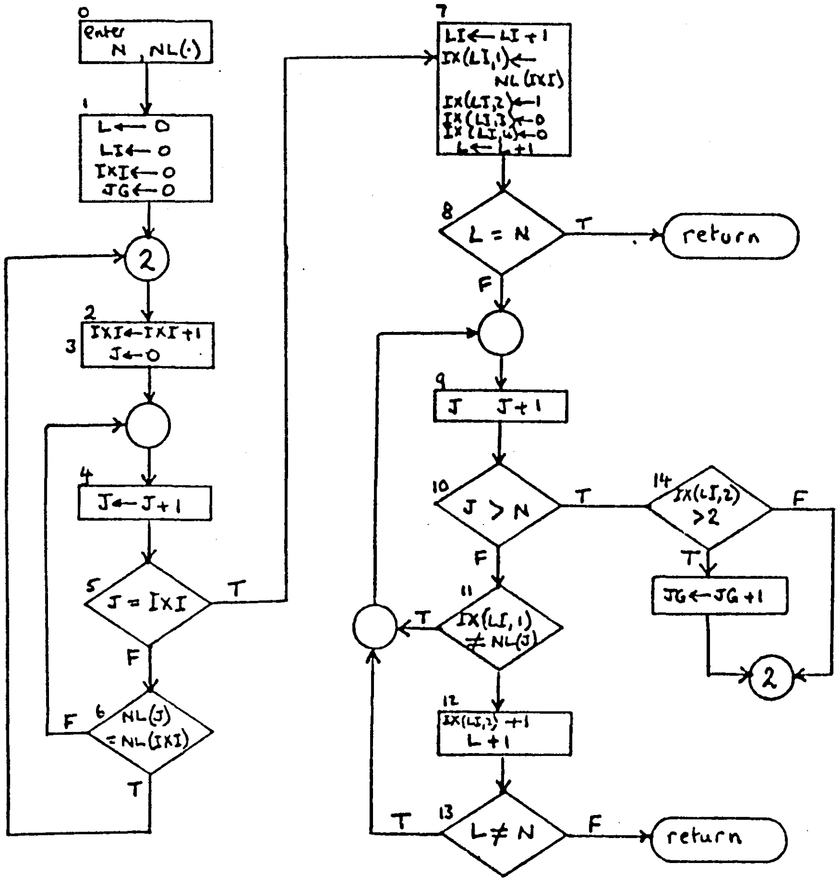


Figure 36

algorithm to construct. This is algorithm SELG which will select better generators, if there are any, than the prime generators taken from the underlying fractional two-level factorial. In MULFAC the algorithm DEFGEN (a slightly modified version of algorithm DEFCON developed in chapter three for the generation of two-level fractional factorials) is used to produce prime generators. At this stage they are in binary form: that is each generator is represented as bit values set to 0 or 1 within an integer. For example, the generators for a 2^{3-1} design may be represented by integers 5 and 6 with bit patterns 101 and 110 respectively. The first task of algorithm SELG is to convert the prime generators from binary to integer form and, according to the generator orders, find how large a design they would generate. Thereafter the generators are cycled, tested for acceptability, according to the criteria specified earlier in this chapter, and for any reduction in the size of design the would generate provided the size would not be too small.

As will be seen later in the development of MULFAC, algorithm SELG will be used only after DEFGEN has been applied to those factors with unique numbers of levels plus those pairs of factors which have equal numbers of levels. For example, in a 2 x 2 x 2 x 3 x 3 x 5 factorial, SELG will be applied to the generators concerned with the last three factors. Some aspects of it will however be introduced to other parts of MULFAC.

The outline algorithm is:

Algorithm SELG (SELect Generators)

Step 0 Enter prime generators in binary form.

Step 10 Convert generators from binary to integer form; compute generator orders and their product.

Step 30 Cycle the generators; compute the orders and products of the cycled generators; test for improvement; for those pairs of factors with equal numbers of levels, check there is at least one generator in which these two factors have integer values that are co-prime; if there is an improvement, note the cycle (the Mth) to which it relates.

Step 70 If no improvement has been found, use the prime generators; if an improvement has been found reconstruct the Mth cycle generators.

The full algorithm follows. The flowchart is figure 37.

Algorithm SELG (SELect Generators for multi-level asymmetric factorial, given the equivalent generators for a two-level factorial in binary form)
(enter with NGI (number of generators), IB(.) (generators in binary form), NIG (number of factors), NLL(.) (number of levels to each factor), LI (number of factor subsets), IX(. , .) (properties of factor subsets): return with IDD(. , .) (generators in integer form))

```
Step  0  (initialise)
         set NOMIN←1;  NB←1 - NIG
Step  1  set I←0
Step  2  set I←I + 1
Step  3  if I > NIG then goto step 6 fi
Step  5  set NOMIN←NOMIN * NLL(I) ; NB←NB + NLL(I);  goto step 2
Step  6  set NO1← NOMIN - 1

Step 10  (convert prime generators from binary to integer form;
         compute generator orders and their products)
         set J← 0;  NO←1
Step 11  set J← J + 1
Step 12  if J > NGI then goto step 21 fi
Step 13  set I←0
Step 14  set I←I + 1
Step 15  if I > NIG then goto step 19 fi
Step 16  set IDD(J,I)← ITEST(IB(J), I)
Step 17  set II(I)←JORD(IDD(J,I), NLL(I))
Step 18  goto step 14
Step 19  set IL(J)←LCM(II,NIG);  NO←NO * IL(J)
Step 20  goto step 11
Step 21  if NO < NB or NO > NOMIN then goto step 30 fi
Step 22  set M← 0;  NOMIN←NO;  NO1←NO -1

Step 30  (cycle generators; compute orders and products;  test
          for improvement)
         set I←0
Step 31  set I←I + 1
Step 32  if I > NO1 then goto step 70 fi
Step 33   for J←1 to NIG  set IKK(J)←0; IK(J)←0
Step 34   set  NO←1;  KJ←I;  IJ←0
Step 35   set  IJ←IJ + 1
Step 36   if  IJ > NGI  then  goto step 47  fi
Step 37   set L←(KJ - 1)/IL(IJ); MM←KJ - IL(IJ) * L; KJ←L + 1
Step 38   set JK←0
Step 39   set JK←JK + 1
Step 40   if  JK > NIG  then  goto step 46  fi
Step 41   set  KKK←IDD(IJ,JK) * MM;  L←NLL(JK);
          IDT(IJ, JK)←KKK - L * (KKK/L);  IT←IDT(IJ, KK);
          II(JK)←JORD(IT, L)
Step 42   if  IT = 0  then goto step 39  fi
Step 43   set  IK(JK)←IK(JK) + IT
Step 44   if  IHCF(IT, L) = 1  then  set  IKK(JK)←1  fi
Step 45   goto  step 39
Step 46   set ILT←LCM(II, NIG);  NO←NO * ILT;  goto step 35
Step 47   set  J←0
Step 48   set  J←J + 1
Step 49   if  J > NIG  then  goto step 52  fi
Step 50   if  IK(J) = 0  then goto step 31  fi
Step 51   if  IKK(J) = 0  then  goto step 31 else goto step 48  fi
Step 52   (for those pairs of factors with equal numbers of levels,
          check there is at least one generator in which these two
          factors have different integer values)
          set  IXJ←0
Step 53   set  IXJ←IXJ + 1
Step 54   if  IXJ > LI  then goto step 67  fi
Step 55   if  IX(IXJ, 2) ≠ 2 then  goto step 53  fi
Step 56   (find first factor in pair)
          set  I3←0
Step 57   set  I3←I3 + 1
Step 58   if  NLL(I3) ≠  IX(IXJ,1)  then  goto  step 57  fi
Step 59   (find second factor in pair)
          set  I4←I3
Step 60   set  I4←I4 + 1
Step 61   if  NLL(I4) ≠  IX(IXJ,1)  then  goto step 60  fi
Step 62   set  J3←0
Step 63   set  J3←J3 + 1
Step 64  if J3 > NGI then goto step 31 fi (no inequality found)
Step 65  if IDT(J3,I3) ≠ IDT(J3,I4) then goto step 53 fi (inequality found)
Step 66  goto step 63
Step 67  (all generators checked)
         if NO < NB or NO > NOMIN then goto step 31 fi
Step 68  set M ← I; NOMIN ← NO; goto step 31

Step 70  (if M = 0 use prime generators; otherwise M indicates
         best set of cycled generators)
         if M > 0 then goto step 75 fi
Step 71  set NO ← NOMIN; return
Step 75  (use M to recompute best set of generators)
         set I ← 0
Step 76  set I ← I + 1
Step 77  if I > NGI then return fi
Step 78  set L ← (M - 1)/IL(I); MM ← IL(I) * L; M ← L + 1
Step 79  set J ← 0
Step 80  set J ← J + 1
Step 81  if J > NIG then goto step 85 fi
Step 82  set KJ ← IDD(I,J) * MM; L ← NLL(J)
Step 83  set IDD(I,J) ← KJ - L * (KJ/L); II(J) ← JORD(IDD(I,J), L)
Step 84  goto step 80
Step 85  set IL(I) ← LCM(II,N)
Step 86  goto step 76
```

I now return to the development of the main algorithm MULFAC and in describing it I refer to the outline flowchart in figure 38. Some of the ideas introduced have arisen in the practical pursuit of an effective algorithm and not simply in the implementation of the theory already described. For example, take the case of six factors of which three have two levels each and the other three have three levels each. If all six factors are considered together in DEFGEN, three generators are produced, each with an order of six. The product of the generator orders is 216: the largest possible number of observations. Alternatively, the three two-level factors yield two generators, each with an order two, and the three three-level factors yield two generators, each with an order three. The

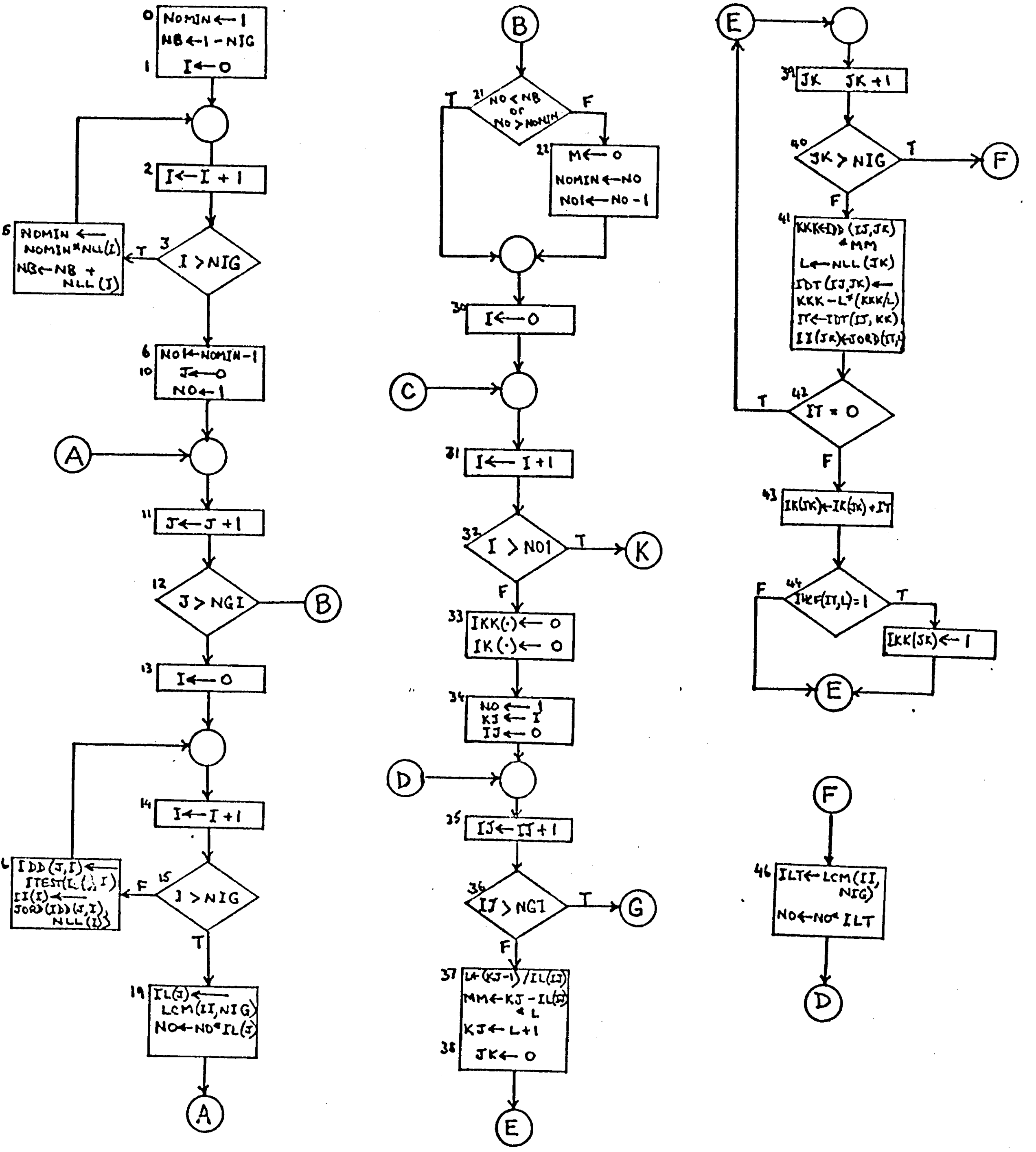
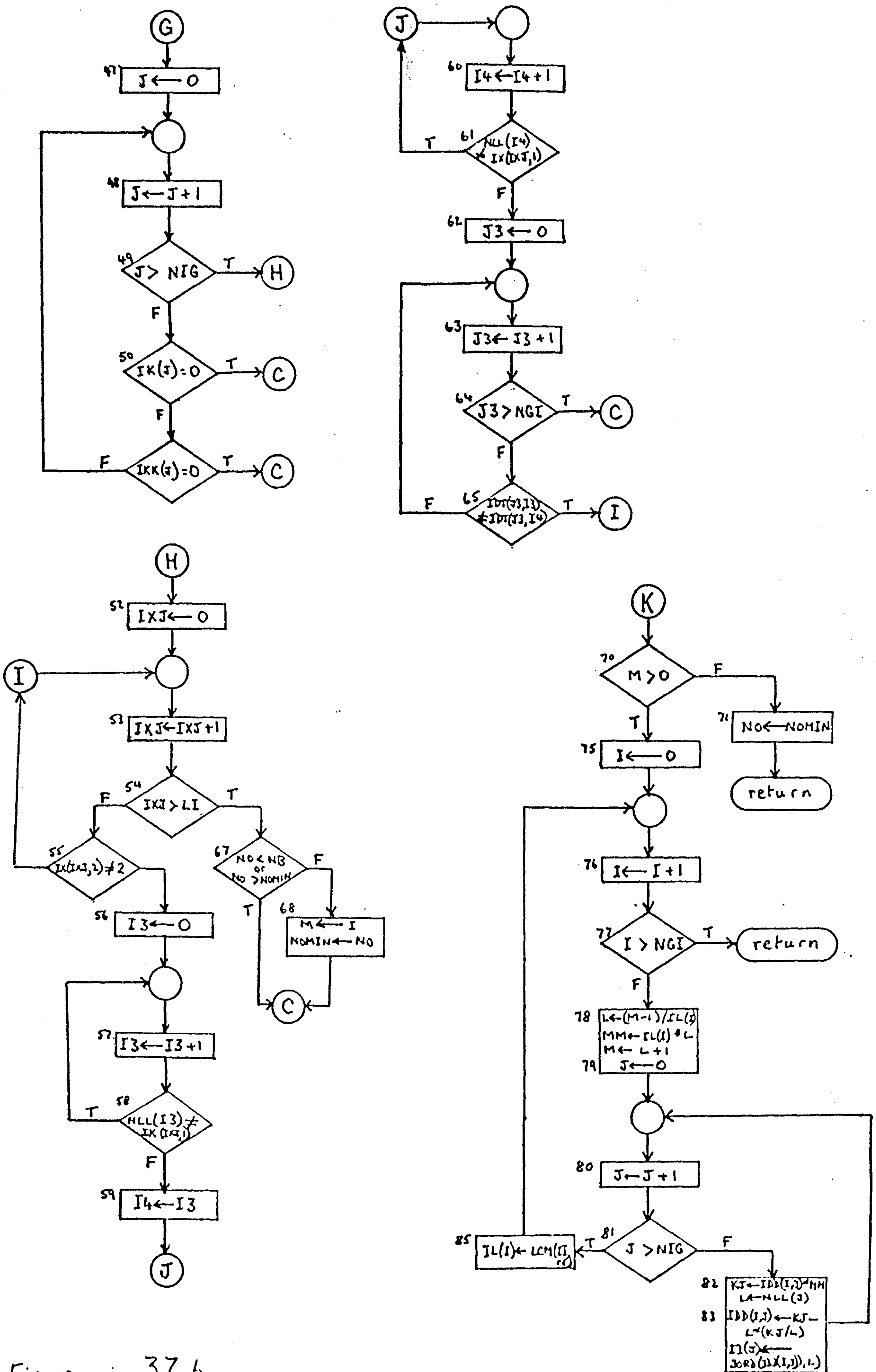

0
NOMIN ← 1
NB ← 1 - NIG
1
I ← 0
2
I ← I + 1
3
I > NIG
T
5
NOMIN ← NOMIN*NLL(I)
NB ← NB + NLL(I)
6
NO1 ← NOMIN - 1
10
J ← 0
NO ← 1
A
11
J ← J + 1
12
J > NGI
B
13
I ← 0
14
I ← I + 1
15
I > NIG
F
T
16
IDD(J,I) ← ITEST(IL(J),I)
II(I) ← JORD(IDD(J,I), NLL(I))
19
IL(J) ← LCM(II,NIG)
NO ← NO*IL(J)
A
B
21
NO < NB or NO > NOMIN
T
F
22
M ← 0
NOMIN ← NO
NO1 ← NO - 1
30
I ← 0
C
31
I ← I + 1
32
I > NO1
T
K
F
33
IKK(·) ← 0
IK(·) ← 0
34
NO ← 1
KJ ← I
IJ ← 0
D
35
IJ ← IJ + 1
36
IJ > NGI
T
G
F
37
L ← (KJ-1)/IL(IJ)
MM ← KJ - IL(IJ)*L
KJ ← L + 1
38
JK ← 0
E
E
39
JK JK + 1
40
JK > NIG
T
F
F
41
KKK ← IDD(IJ,JK)*MM
L ← NLL(JK)
IDT(IJ,JK) ← KKK - L*(KKK/L)
IT ← IDT(IJ, KK)
II(JK) ← JORD(IT,L)
42
IT = 0
T
F
43
IK(JK) ← IK(JK) + IT
44
IKEF(IT,L) = 1
F
T
IKK(JK) ← 1
E
F
46
ILT ← LCM(II, NIG)
NO ← NO*ILT
D

Figure 37a



G
47
J ← 0
48
J ← J + 1
49
J > NIG
T
H
F
50
IK(J) = 0
T
C
F
IKK(J) = 0
F
T
C
J
60
I4 ← I4 + 1
61
NLL(I4) ≠ IX(IXJ,1)
T
F
62
J3 ← 0
63
J3 ← J3 + 1
64
J3 > NGI
T
C
F
65
IDT(J3,I3) ≠ IDT(J3,I4)
F
T
I
H
52
IXJ ← 0
I
53
IXJ ← IXJ + 1
54
IXJ > LI
F
T
55
IX(IXJ,2) ≠ 2
56
I3 ← 0
57
I3 ← I3 + 1
58
NLL(I3) ≠ IX(IXJ,1)
T
F
59
I4 ← I3
J
67
NO < NB or NO > NOMIN
F
T
68
M ← I
NOMIN ← NO
C
K
70
M > 0
F
T
71
NO ← NOMIN
return
75
I ← 0
76
I ← I + 1
77
I > NGI
T
return
F
78
L ← (M-1)/IL(I)
MM ← IL(I) * L
M ← L + 1
79
J ← 0
80
J ← J + 1
81
J > NIG
T
F
85
IL(I) ← LCM(II, L)
82
KJ ← IDD(I,J) * MM
L ← NLL(J)
83
IDD(I,J) ← KJ − L*(KJ/L)
II(J) ← JORD(IDD(I,J)), L)

Figure 37 b

product of the orders of the four generators is now only 36. This is generalised in MULFAC by looking separately at those subsets of factors each with three or more factors with the same number of levels for each. It was for this reason that algorithm FASET, already described, was developed to determine the subsets of factors with equal numbers of levels.

A further practical consideration, related to the one described above, was that sometimes the product of orders of generators pertaining to a subset of factors was equal to the number of levels of another one or two factors. For example, take the case of five factors of which three have two levels each and the other two have four levels each. The three two-level factors yield two generators, each with an order two. The product of the generator orders is four which is equal to the number of levels of each of the other two factors. The practical way to deal with this is to attach one of the four-level factors to one generator and the other factor to the other generator. That is: we have two generators 10100 and 01100, each of order two. The fourth and fifth factors are attached to yield the generators 10110 and 01101, each of order four. The product of the generator orders is 16; the number of observations in the experimental design they will generate.

The structure of the algorithm is now:

Algorithm MULFAC (MULti level asymmetric FACtorials)

```
Step  0  Call ENFAC to enter the experimental requirements in terms
         of the factors, factor levels, and interactions.
Step 10  Call FASET to analyse the requirements in terms of subsets
         of factors with equal numbers of levels.
Step 20  Find a subset of factors with three or more factors with
         equal numbers of levels, or a subset of only one or two
         factors related to another subset as described above.
         If the search is successful  then goto step 26 else goto step 100
Step 26  if it is a small subset related, or linked as described, then
         goto step 66  else goto step 27.
```

Step 27 The chosen subset of factors will now be treated as an independent set of factors for which an experimental design is required. Initialise for a call to DEFGEN by setting up the main effect and interaction requirements for these factors.

Step 50 Call DEFGEN which first finds the smallest two-level fractional factorial design and then from this design extracts the generators.

Step 65 If this is a subset of three or more factors which have been linked to a previously considered subset, then goto step 66 else goto step 70.

Step 66 Merge the linked subsets by setting flags which indicate the merging; goto step 80.

Step 70 If the product of the orders of the generators produced by DEFGEN is equal to the modulus (number of levels per factor) of any remaining factor subset, then link the two subsets.

Step 80 Convert the generators from binary to integer form. (For example a generator may be expressed in binary as an integer with bit values 01011 which, in integer form, would become a set of integers with values 0,1,0,1,1).

Step 90 Return to step 20.

Step 100 If there are no factors left, that is all factors have now been represented in generators, then goto step 200.

Step 106 If there are only one or two factors left then goto step 107 else if there are more than two factors left then goto step 120.

Step 107 Create one or two elementary generators for the one or two remaining factors. An elementary generator is one in which only one factor is represented by 1 and all the other factors are represented by 0. Then goto step 200.

Step 120 Initialise for a call to DEFGEN by setting up the main effect and interaction requirements for the remaining factors.

Step 150 Call DEFGEN.

Step 155 Initialise for a call to SELG.

Step 170 Call SELG to select the generators for a multi-level asymmetric factorial, given the equivalent generators for a two-level factorial.

Step 180 Copy the generators from SELG into the set of generators for all factors.

Step 200 Find the design size (product of generator orders) and call LEV repeatedly to generate the experimental design.

Step 1000 randomise the experimental order, then stop.

The outline flowchart for MULFAC (figure 38) is followed by the detailed algorithm and corresponding flowchart (figure 39).

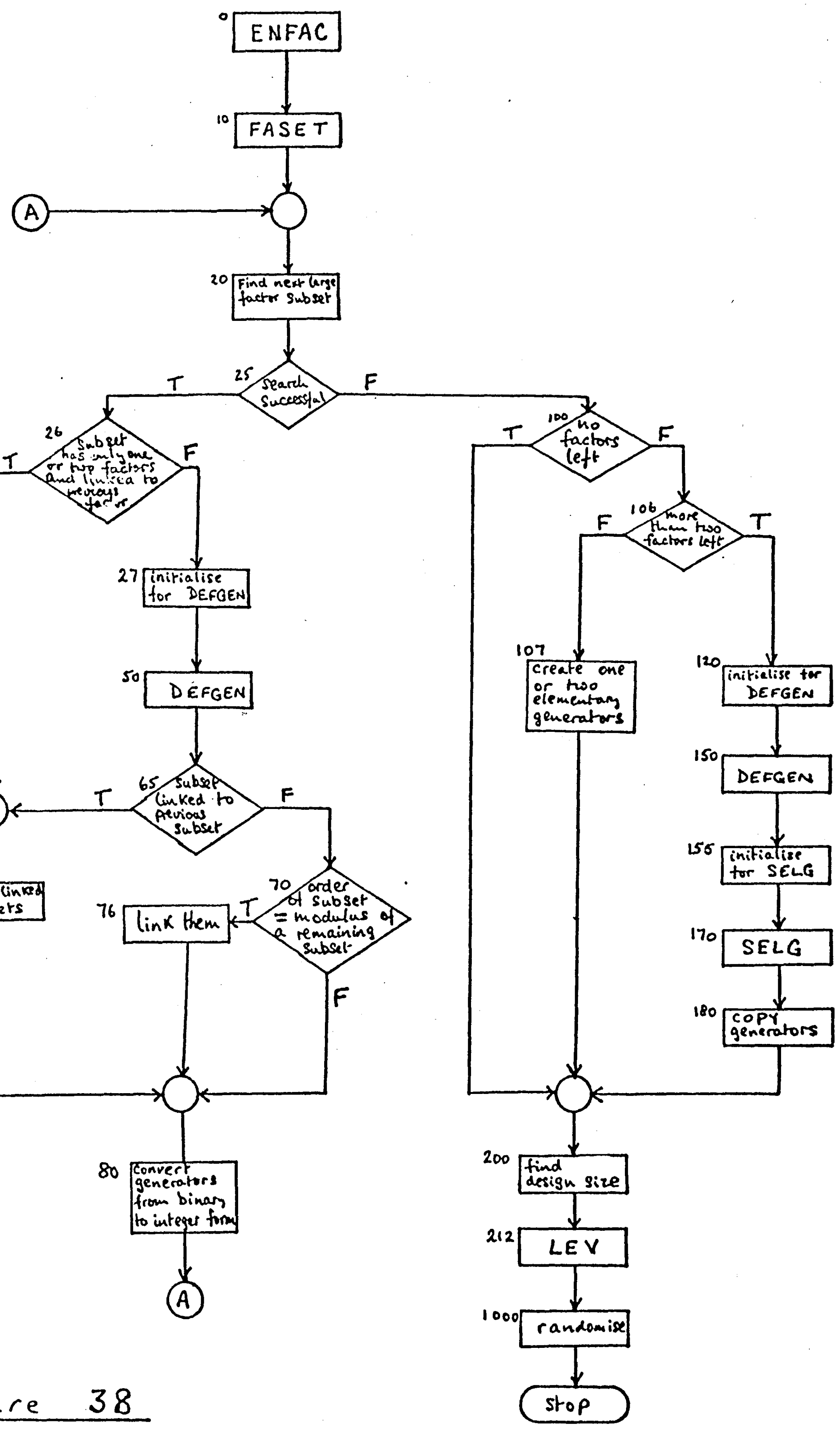

0 ENFAC
10 FASET
A
20 Find next large factor subset
25 Search successful
T
F
26 Subset has only one or two factors and linked to previous factor
T
F
27 initialise for DEFGEN
50 DEFGEN
65 subset linked to previous subset
T
F
66 merge linked subsets
70 order of subset = modulus of a remaining subset
T
F
76 link them
80 convert generators from binary to integer form
A
100 no factors left
T
F
106 more than two factors left
F
T
107 create one or two elementary generators
120 initialise for DEFGEN
150 DEFGEN
155 initialise for SELG
170 SELG
180 COPY generators
200 find design size
212 LEV
1000 randomise
stop

Figure 38

The main variables to be used in algorithm MULFAC are:

N	the total number of factors to be included.
NL(I)	the number of levels of the I'th factor. (*16)
NFULL	the size of the full factorial = the product of the NL(I).
NB	the minimum number of observations needed to estimate all the required coefficients (see section 5.1 earlier in this chapter).
IX(I,1)	the modulus of the I'th subset of factors (number of levels).
IX(I,2)	the number of factors in the I'th subset. (*16,4)
IX(I,3)	a value J (less than I) which points to the J'th subset of factors to which the I'th subset is linked, with the following exception:
IX(I,4)	set to zero if the I'th subset has not yet been used to produce generators, or set to one if it has been used. However, if IX(I,4) is greater than one, then the values of IX(I,3) and IX(I,4) indicate the range (first and final) of generators associated with the I'th subset of factors.
LI	the number of factor subsets.
JG	the number of subsets with moduli greater than two, plus the number of subsets with moduli less than or equal to two that are linked with larger subsets.
NV	the total number of requirements for the design.
MV(I)	the I'th requirement indicated by the bit pattern of the integer. (*32)
NVI	the number of requirements taken into account when entering DEFGEN on behalf of a subset of factors.
MVI(I)	the I'th requirement relative to a subset of factors. (*16)
NG	the total number of generators.
ID(I,J)	the J'th integer in the I'th generator. (*8,16)
NGI	the number of generators relative to a subset of factors.
IDD(I,J)	the J'th integer in the I'th generator for a subset of factors. (*8,16)
IL(I)	the order of the I'th generator. (*8)
NO	the product of the orders of the generators.
IDT(I,J)	the J'th integer of the I'th test generator. (*8,16)
ILT(I)	the order of the I'th test generator. (*8)
NIG	the number of factors considered in a call to DEFGEN.
NLL(I)	the number of levels of the I'th of those factors considered by DEFGEN and SELG. (*16)
LF(I)	the factor in the full set corresponding to the I'th factor in the subset being considered by SELG. (*16)
IB(I)	the I'th generator returned by DEFGEN in binary form. (*8)

Practical dimensions of arrays are denoted by (*n)

Algorithm MULFAC (MULti level asymmetric FACtorials)

```
Step  0   Call ENFAC (to enter experimental name, number of factors,
          number of factor levels (their moduli), and their interactions)

Step  1   set to zero ID(.,.);  NG ← 0
Step 10   call FASET (to determine subsets of factors with equal moduli)
Step 20   (find the next suitable factor subset (see note in text))
          set IG ← 0;   MIN ← 100
Step 21   set IG ← IG + 1
Step 22   if (IX(IG,1) < MIN) and ((IX(IG,2) > 2) or
          (IX(IG,3) ≠ 0))  then do step 23 od  fi
Step 23   set MIN ← IX(IG,1);   IM ← IG
Step 24   if IG = LI  then do step 25 od  else goto step 21  fi
Step 25   if MIN = 100  then goto step 100
                        else if IX(IM,2) > 2 then goto step 27
                        else set NGI ← 0; goto step 26   fi  fi
Step 26   set NGI ← NGI + 1;  IB(NGI) ← 2**(NGI - 1):
          if NGI = IX(IM,2)  then  goto step 66 else goto step 26 fi
Step 27   (initialise for entry to DEFGEN)
          set NIG ← IX(IM,2) + IX(IM,4)
          (IX(IM,4) = 1 if subset linked to previous one, otherwise = 0)
Step 28   set NVI ← IX(IM,4)

Step 29   set NVI ← NVI + 1 ;   MVI(NVI) ← 0
Step 30   set MVI(NVI) ← IONBT(MVI(NVI),NVI)
Step 31   if NVI < NIG  then goto step 29
Step 32   (look for interactions which contain only those factors in
          the current subset;  set up mask first)
          set MASK ← 0;  I ← 0
Step 33   set I ← I + 1
Step 34   if I > N  then goto  step 37  fi
Step 35   if NL(I) = IX(IM,1)  then set MASK ← IONBT(MASK,I)  fi
Step 36   goto step 33
Step 37   set I ← N
Step 38   set I ← I + 1
Step 39   if I > NV  then goto step 50 fi
Step 40   if and(MASK,MV(I)) = 0  then  goto  step 38  fi
Step 41   set I1 ← 0;  J ← 0;  NVI ← NVI + 1;  MVI(NVI) ← 0
Step 42   set J ← J + 1
Step 43   if J > N  then  goto step 38  fi
Step 44   if  ITEST(MASK,J) = 1  then  do  steps 45,46 od  fi
Step 45   set  I1←I1 + 1
Step 46   if  ITEST(MV(I),J) = 1  then set MVI(NVI)←IONBT(MVI(NVI),I1) fi
Step 47   goto step 42

Step 50   oall  DEFGEN (to produce generators from an equivalent
          two level fractional factorial)
          (enter with NIG, NVI, MVI(.); return NGI(number of generators)
          and  IB(.) (generators in binary representation)

Step 60   set  NOG←IX(IM,1) ** NGI
          (the number of levels in each factor of this subset is the
          order of each generator,  thus the full order of the subset
          is the product of the orders of the generators which is the
          same as the order of the generators raised to the power of
          the number of generators)

Step 65   (is this subset linked to a previous one?)
          if IX(IM,3) = 0  then goto step 70  (not linked)
Step 66   set  J←IX(IM,4);  J1←IX(J,3) - 1
Step 67   set  LL←IX(J,4);   goto step 80

Step 70   (if NOG is equal to any of the remaining subset moduli,
           then relate the subsets)
          set  I←0;  L←0;  LL←0;  J1←NG
Step 71   set  I←I + 1
Step 72   if  I > LI  then goto step 80  fi
Step 73   if  I = IM  then goto step 71  fi
Step 74   if  IX(I,4) ≠ 0  then goto step 71  fi
Step 75   if  IX(I,1) ≠ NOG  then goto step 71  fi
Step 76   set  IX(I, 3)←IM;  IX(IM,3)←NG + 1;  JG←JG + 1;
          IX(IM,4)←NG + NGI:  IX(I,4)←1

Step 80   (convert generators from binary to integer form)
           set  I←0
Step 81   set  I←I + 1
Step 82   if  I ≤ NGI  then  goto step 85  fi
Step 83   set  IX(IM,1)←100;   if  J1 > NG  then set NG←J1   fi
Step 84   goto step 20
Step 85   set  J1← J1 + 1;   L←0;  J←0
Step 86   set  J←J + 1
Step  87   if  J > N  then goto  step 81  fi
Step  88   if  NL(J)  ≠  IX(IM,1)  then  goto  step 86  fi
Step  89   set  L←L + 1;   ID(J1,J)←ITEST(IB(I),L)
Step  90   goto  step  86

Step 100   (find how many factors have not been included in generators)
           set  L←0;  I←0
Step 101   set I←I + 1
Step 102   if  I > LI  then  goto step 105  fi
Step 103   if  IX(I, 1)≠ 100  then  set  L←L + IX(I,2)  fi
Step 104   goto  step  101

Step 105   if  L = 0  then  goto step 200  fi
Step 106   if  L > 2  then  goto step 120  fi
Step 107   set I←0
Step 108   set I←I + 1
Step 109   if  I > LI  then  goto step 200 fi
Step 110   if  IX(I,4)  ≠  0  then goto step 108  fi
Step 111   set  I1←IX(I,1);  I2←0;  I3←0
Step 112   set  I2←I2 + 1
Step 113   if  I2 > IX(I,2)  then  goto step 108  fi
Step 114   set  I3←I3  + 1
Step 115   if  I3 > N  then goto step 108  fi
Step 116   if  NL(I3)  ≠  I1  then goto step 114  fi
Step 117   set  NG←NG + 1;  ID(NG,I3)←1;  goto step 112

Step 120   (initialise  for  DEFGEN)
           set  NIG←L;   NVI←0
Step 121   set  NVI←NVI + 1;   MVI(NVI)←0
Step 122   set  MVI(NVI)←IONBT(MVI(NVI),NVI)
Step 123   if  NVI < NIG  then goto  step 121  fi

Step 125   (set up MASK to detect interactions)
           set  MASK←0;  I←0
Step 126   set  I←I + 1
Step 127   if  I > LI  then goto step 135  fi
Step 128   if  IX(I,4)  ≠  0  then  goto step 126  fi
Step 129   set  I1←IX(I,1);  I2←0
Step 130   set  I2←I2 + 1
Step 131  if  I2 > N  then  goto step 126  fi
Step 132  if  NL(I2) ≠ I1  then  goto step 130  fi
Step 133  set  MASK ← IONBT(MASK, I2);  goto step 130  fi

Step 135  set  I ← N
Step 136  set  I ← I + 1
Step 137  if  I > NV  then  goto step 150  fi
Step 138  if  and(MASK, MV(I)) = 0  then goto step 136  fi
Step 139  set  I1 ← 0;  J ← 0;  NVI ← NVI + 1;  MVI(NVI) ← 0
Step 140  set J ← J + 1
Step 141  if  J > N  then goto step  136  fi
Step 142  if  ITEST(MASK,J) = 1  then do steps 143, 144 od fi
Step 143  set I1 ← I1 + 1
Step 144  if  ITEST(MV(I),J) = 1  then set MVI(NVI) ← IONBT(MVI(NVI),I1) fi
Step 145  goto step 140

Step 150  call  DEFGEN  (to produce generators from an equivalent
          two level fractional factorial)
         (enter with NIG,  NVI,  MVI(.);   return with NGI and IB(.))

Step 155  (set number of factor levels for entry to SELG)
          set  I ← 0;  I2 ← 0
Step 156  set  I ← I + 1
Step 157  if  I > N  then goto step 170  fi
Step 158  set  I1 ← 0
Step 159  set  I1 ← I1 + 1
Step 160  if  I1 > LI  then  goto step 156  fi
Step 161  if  IX(I1,4) ≠ 0  then goto step 159  fi
Step 162  if  NL(I) ≠ IX(I1,1)  then goto step 159 fi
Step 163  set  I2 ← I2 + 1; NLL(I2) ← NL(I);  LF(I2) ← I;  goto step 159

Step 170  call  SELG (to select generators for multi-level asymmetric
           factorial, given the equivalent generators for a two-level
           factorial)
           (enter with NGI, IB(.), NIG,NLL(.),  LI (number of factor
            subsets),  IX(. , .)  (properties of factor subsets);
            return with IDD(. , .) (generators in integer form))
```

```
Step 180    (copy generators from SELG into generators for all factors)
            set  I←0
Step 181    set  I←I + 1;   NG←NG + 1
Step 182    if  I > NGI  then goto step 200  fi
Step 183    set  J←0
Step 184    set  J←J + 1
Step 185    if  J > NIG  then goto step 181  fi
Step 186    set  I1←LF(J); ID(NG, I1)←IDD(I, J);  goto step 184

Step 200    (find full design size NO = product of generator orders)
            set  I←0;   NO←1
Step 201    set  I←I + 1
Step 202    if  I > NG  then goto step 208  fi
Step 203    set  J←0
Step 204    set  J← J + 1
Step 205    if  J > N  then goto step 207
Step 206    set  II(J)←JORD(ID(I,J), NL(J));  goto step 204
Step 207    set  IL(I)←LCM(II, N);  NO←NO*IL(I);  goto step 201

Step 208    set  NFULL←1;   I←0
Step 209    set  I←I + 1;  if  I> N  goto step 2091  fi
Step 2090   set  NFULL←NFULL * NL(I);  goto step 209
Step 2091   if  NO≤ NFULL  then goto step 210
                           else set NO← NFULL; NG←N;  I←0   fi
Step 2092   set  I←I + 1;  if  I > N  then  goto step 210
                                     else set J←0;  IL(I)←NL(I)  fi
Step 2093   set  J←J + 1;  if  J > N then goto step 2092 fi
Step 2094   set  ID(I,J)← 0
Step 2095   if  I = J  then set  ID(I,J)←1     fi
Step 2096   goto step 2093
Step 210    print heading;   set  I←0
Step 211    set  I←I + 1;  if  I > NO  then  goto step 1000 fi
Step 212    call  LEV  (to find levels of all factors for I'th observation)
            (enter with I, N, NG, IL, ID;   return with IK)
Step 213    print  N values of IK(.);   goto step 211

Step 1000   randomise order of NO observations;   stop
```

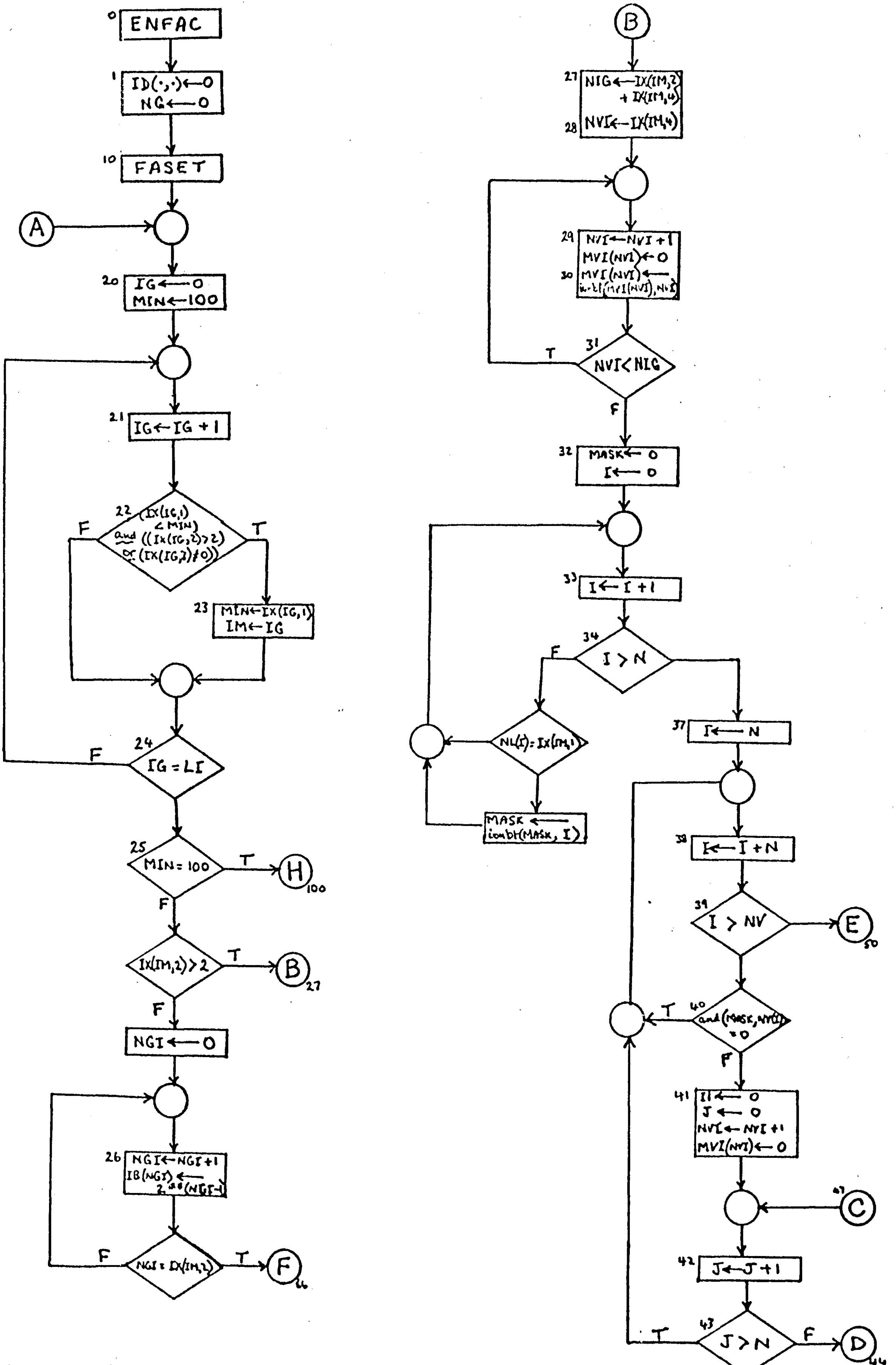

0
ENFAC
1
ID(·,·)←0
NG←0
10
FASET
A
20
IG←0
MIN←100
21
IG←IG+1
22
(IX(IG,1)
< MIN)
and ((IX(IG,2)>2)
or (IX(IG,2)≠0))
F
T
23
MIN←IX(IG,1)
IM←IG
24
IG = LI
F
25
MIN = 100
T
H
100
F
IX(IM,2)>2
T
B
27
F
NGI←0
26
NGI←NGI+1
IB(NGI)←
2**(NGI-1)
F
NGI = IX(IM,2)
T
F
66
B
27
NIG←IX(IM,2)
+ IX(IM,4)
28
NVI←IX(IM,4)
29
NVI←NVI+1
MVI(NVI)←0
30
MVI(NVI)←
ior-bl(MVI(NVI), NVI)
31
NVI < NIG
T
F
32
MASK←0
I←0
33
I←I+1
34
I > N
F
NL(I) = IX(IM,1)
MASK←
ionbt(MASK, I)
37
I←N
38
I←I+N
39
I > NV
E
50
40
and(MASK, NV(I))
= 0
T
F
41
II←0
J←0
NVI←NVI+1
MVI(NVI)←0
47
C
42
J←J+1
43
J > N
T
F
D
44

Figure 39a

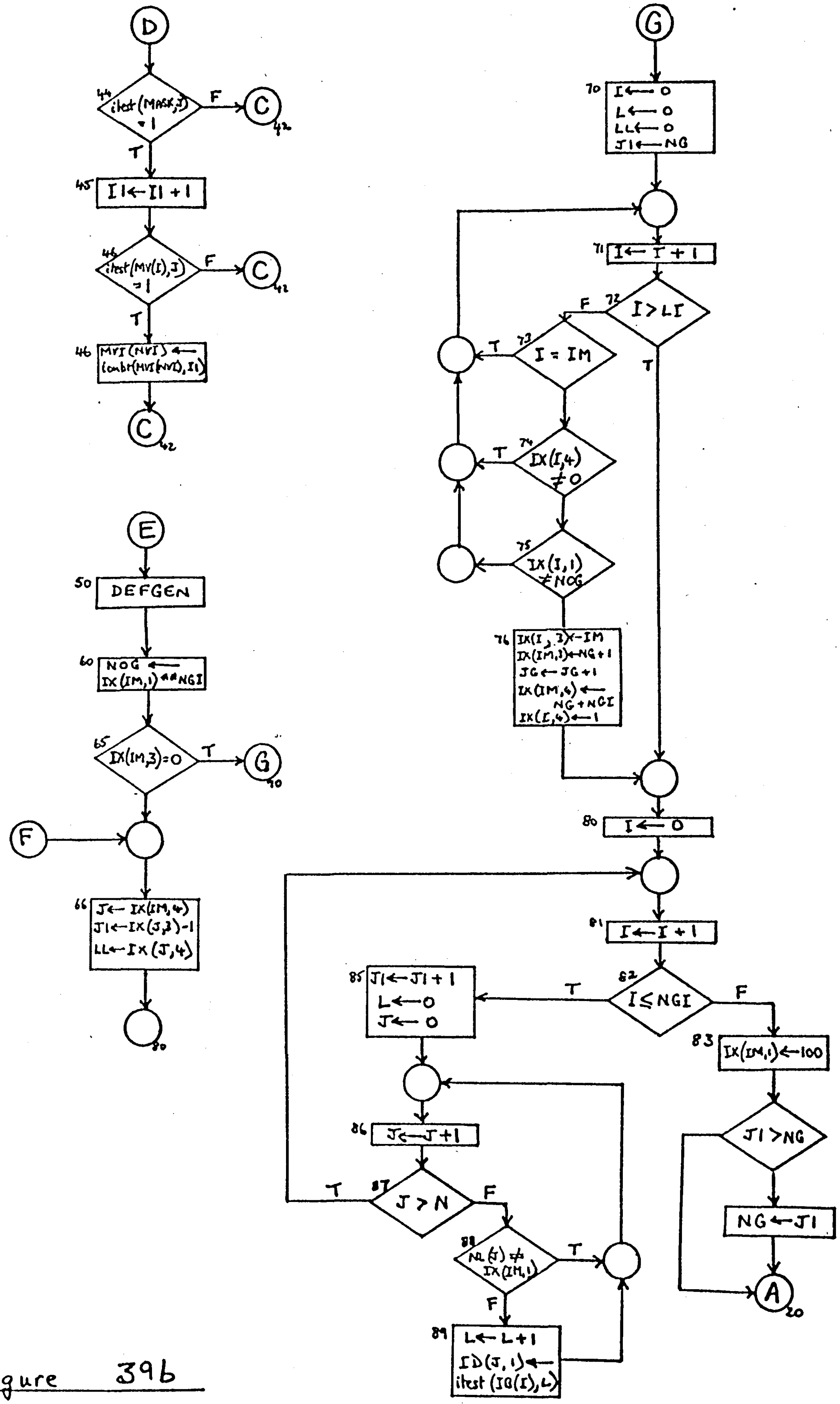

D
44
itest(MASK,J) = 1
F
C
42
T
45
I1 ← I1 + 1
46
itest(MV(I),J) = 1
F
C
42
T
46
MVI(NVI) ← Combr(MVI(NVI), I1)
C
42
E
50
DEFGEN
60
NOG ← IX(IM,1) ** NGI
65
IX(IM,3)=0
T
G
70
F
66
J ← IX(IM,4)
J1 ← IX(J,3)-1
LL ← IX(J,4)
80
G
70
I ← 0
L ← 0
LL ← 0
J1 ← NG
71
I ← I + 1
72
I > LI
F
T
73
I = IM
T
74
IX(I,4) ≠ 0
T
75
IX(I,1) ≠ NOG
76
IX(I,3) ← IM
IX(IM,3) ← NG+1
JG ← JG+1
IX(IM,4) ← NG+NGI
IX(I,4) ← 1
80
I ← 0
81
I ← I + 1
82
I ≤ NGI
T
F
83
IX(IM,1) ← 100
J1 > NG
NG ← J1
A
20
85
J1 ← J1 + 1
L ← 0
J ← 0
86
J ← J+1
87
J > N
T
F
88
NL(J) ≠ IX(IM,1)
T
F
89
L ← L+1
ID(J,1) ← itest(IB(I),L)

Figure 39b

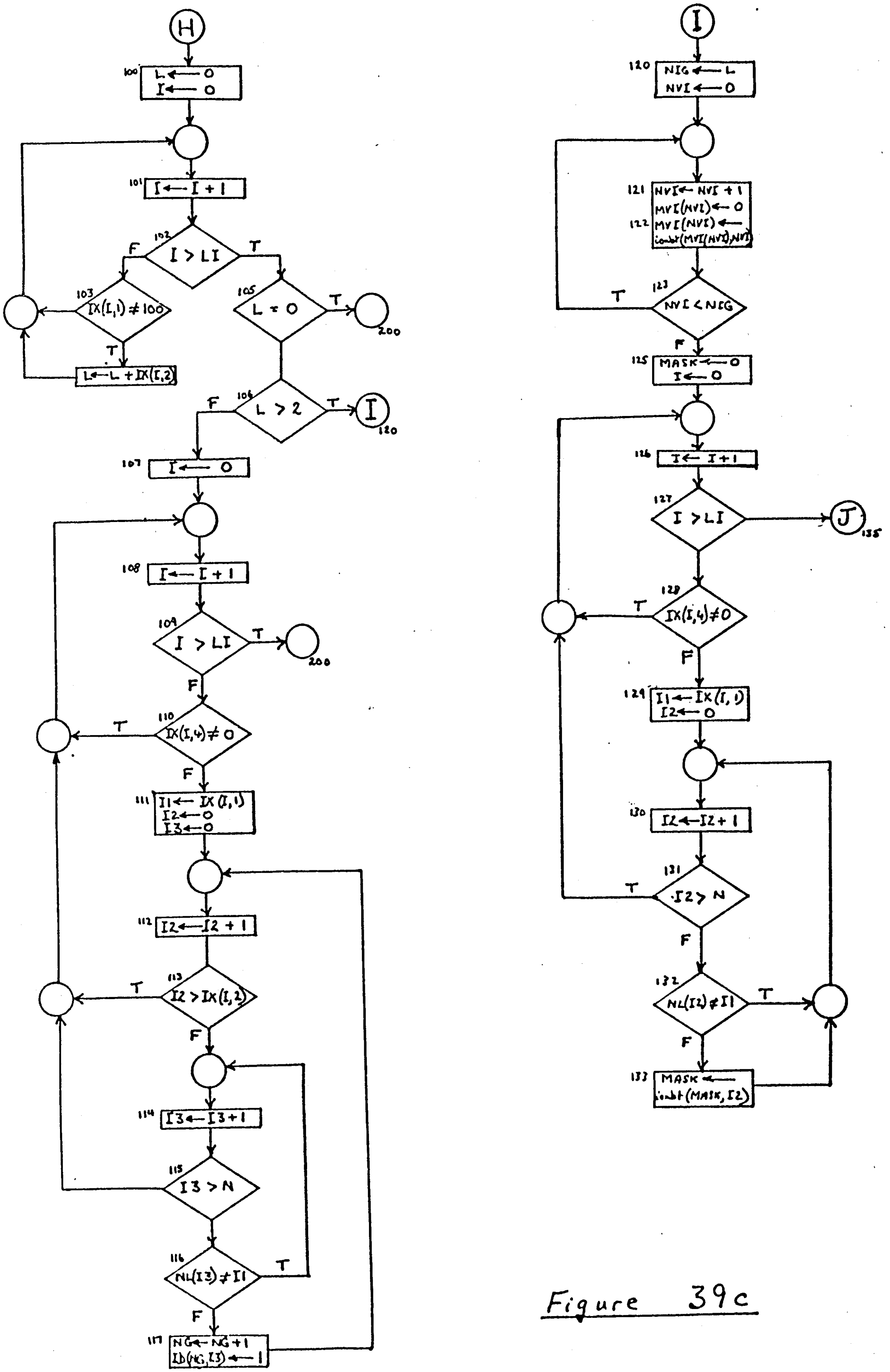
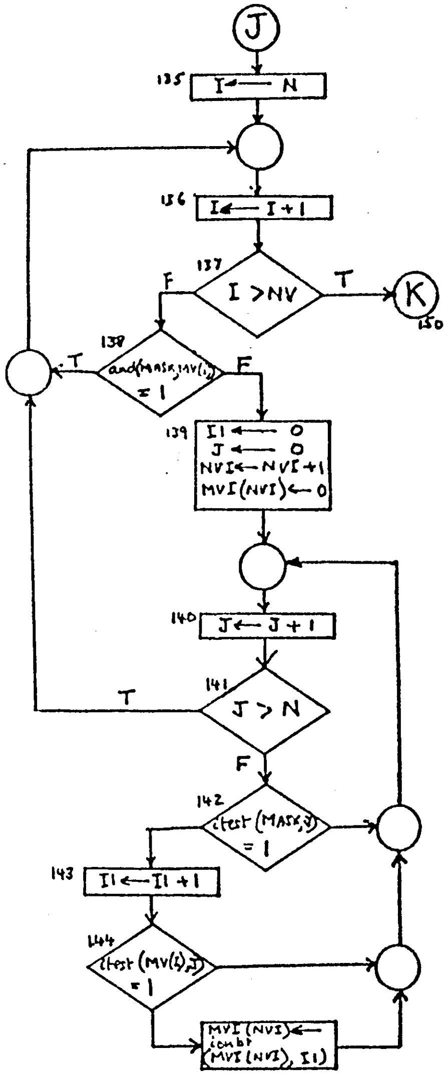
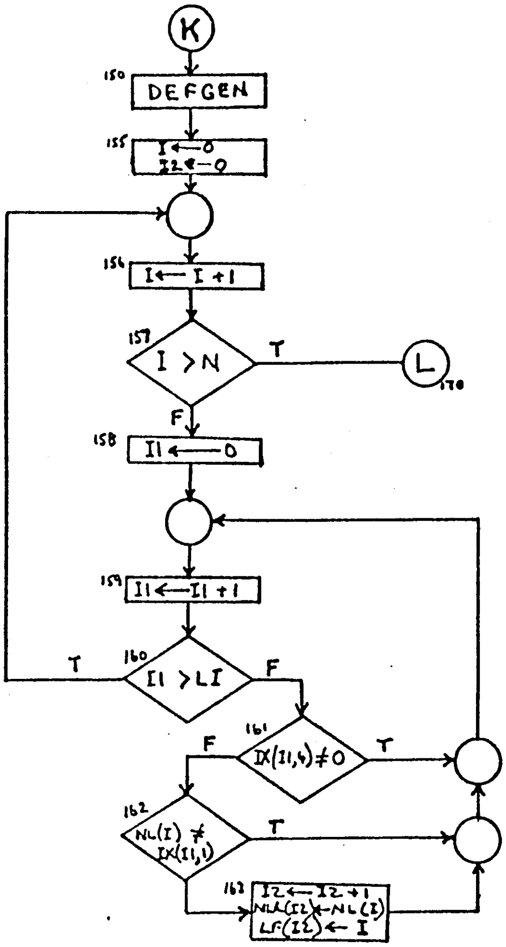

H
100
L ← 0
I ← 0
101
I ← I + 1
102
I > LI
F
T
103
IX(I,1) ≠ 100
T
L ← L + IX(I,2)
105
L = 0
T
200
106
L > 2
F
T
I
120
107
I ← 0
108
I ← I + 1
109
I > LI
T
200
F
110
IX(I,4) ≠ 0
T
F
111
I1 ← IX(I,1)
I2 ← 0
I3 ← 0
112
I2 ← I2 + 1
113
I2 > IX(I,2)
T
F
114
I3 ← I3 + 1
115
I3 > N
116
NL(I3) ≠ I1
T
F
117
NG ← NG + 1
ID(NG,I3) ← 1
I
120
NIG ← L
NVI ← 0
121
NVI ← NVI + 1
MVI(NVI) ← 0
122
MVI(NVI) ←
ionbt(MVI(NVI),NVI)
123
NVI < NIG
T
F
125
MASK ← 0
I ← 0
126
I ← I + 1
127
I > LI
J
135
128
IX(I,4) ≠ 0
T
F
129
I1 ← IX(I,1)
I2 ← 0
130
I2 ← I2 + 1
131
I2 > N
T
F
132
NL(I2) ≠ I1
T
F
133
MASK ←
ionbt(MASK, I2)

Figure 39c


J
135
I ← N
136
I ← I + 1
137
I > NV
T
K
150
F
138
and(MASK, MV(I))
= 1
T
F
139
I1 ← 0
J ← 0
NVI ← NVI + 1
MVI(NVI) ← 0
140
J ← J + 1
141
J > N
T
F
142
ctest (MASK, J)
= 1
143
I1 ← I1 + 1
144
ctest (MV(I), J)
= 1
MVI (NVI) ←
ionbr
(MVI (NVI), I1)


K
150
DEFGEN
155
I ← 0
I2 ← 0
156
I ← I + 1
157
I > N
T
L
170
F
158
I1 ← 0
159
I1 ← I1 + 1
160
I1 > LI
T
F
161
IX(I1,4) ≠ 0
F
T
162
NL(I) ≠
IX(I1,1)
T
163
I2 ← I2 + 1
NLA(I2) ← NL(I)
LF(I2) ← I

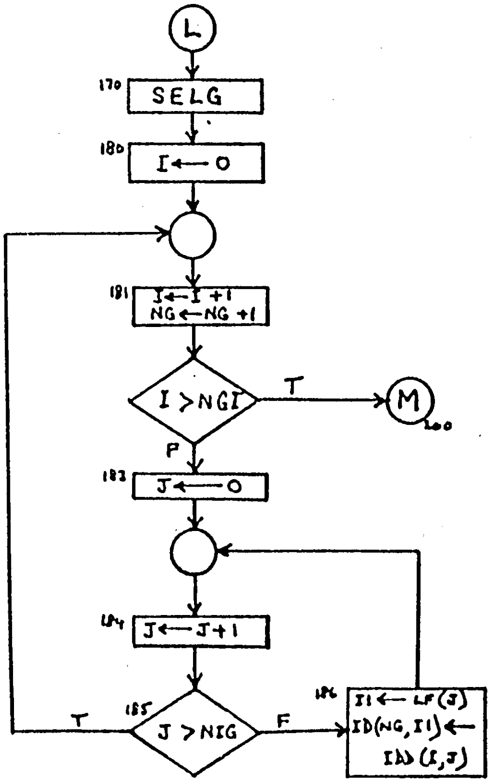

L
170
SELG
180
I ← 0
181
I ← I + 1
NG ← NG + 1
I > NGI
T
M
200
F
183
J ← 0
184
J ← J + 1
185
J > NIG
T
F
186
I1 ← LF(J)
ID(NG, I1) ←
IW (I, J)

Figure 39d

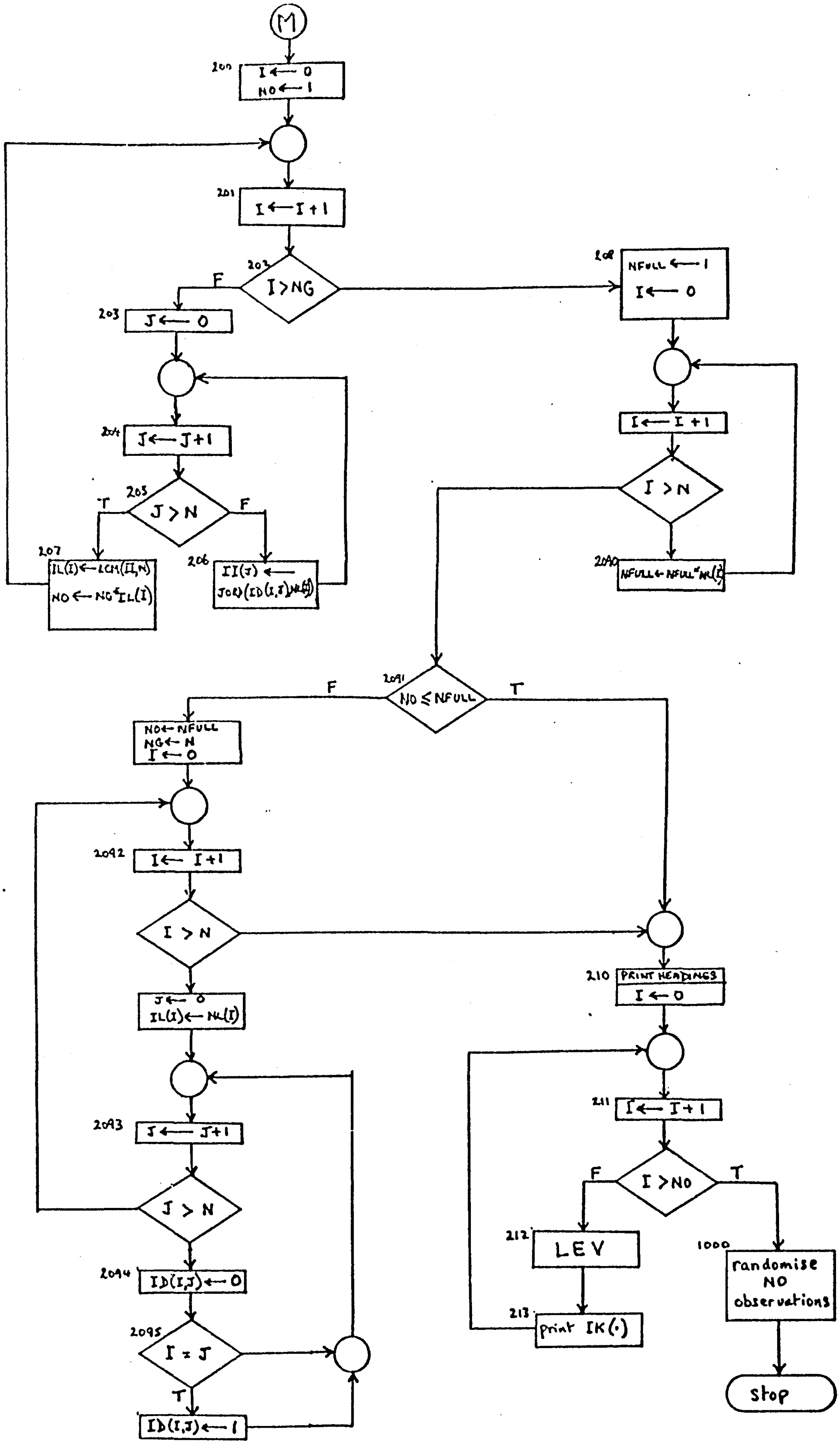

M
200
I ← 0
NO ← 1
201
I ← I + 1
202
I > NG
F
203
J ← 0
204
J ← J + 1
205
J > N
T
F
207
IL(I) ← LCM(II, N)
NO ← NO*IL(I)
206
II(J) ←
JOR(ID(I,J)NL(J))
208
NFULL ← 1
I ← 0
I ← I + 1
I > N
2090
NFULL ← NFULL*NL(I)
2091
NO ≤ NFULL
F
T
NO ← NFULL
NG ← N
I ← 0
2092
I ← I + 1
I > N
J ← 0
IL(I) ← NL(I)
2093
J ← J + 1
J > N
2094
ID(I,J) ← 0
2095
I = J
T
ID(I,J) ← 1
210
PRINT HEADINGS
I ← 0
211
I ← I + 1
I > NO
F
T
212
LEV
213
print IK(·)
1000
randomise
NO
observations
stop

Figure 39e

The Automatic Design of Experiments

Some Practical Algorithms

CHAPTER SEVEN

REDUCING THE BALANCED ASYMMETRIC FRACTION

1 Background

The method developed in chapter six leads to the generation of balanced fractional designs which are adequate for estimating all specified main effects and interactions. Sometimes however the generated designs are more than adequate: the number of observations exceeds the number of contrasts to be estimated by many more than the few extra needed for error estimation. In these cases the costs of practical experimentation dictate that the size of the design should be reduced further. In this chapter I present two criteria that are widely applied in reducing experimental designs. See Fedorov (1972). The two criteria are:

A-optimality: The trace of the inverted information (cross-product) matrix is minimised. In the case of a discrete design, such as an asymmetric factorial using qualitative variables, the use of the A-optimality criterion means choosing a subset of r points from a design with n points ($n > r$) such that the trace of the inverted information matrix of the r-subdesign is no greater than that of any other r-subdesign.

D-optimality: The determinant of the inverted information matrix is minimised. In the case of a discrete design, the use of the D-optimality criterion means choosing a subset of r points from a design with n points ($n > r$) such that the determinant of the inverted information matrix of the r-subdesign is no greater than that of any other r-subdesign.

Several papers have been published on procedures for sequentially designing experiments using the criterion of D-optimality. See, for example, Goldsmith (1974) and Wynn (1970). A common problem, however, was that the sequence had to start with a basic subdesign: one whose information matrix is non-singular. Published procedures were not helpful in choosing the best basic subdesign. It was in tackling this problem that I developed the algebra of section 2 of this chapter and hence to an algorithm based on the criterion of A-optimality. Unfortunately, this led to a computing problem: the computing time was too long for the algorithm to be of practical use. I therefore abandoned it and returned to the D-optimality criterion. However, I am including a report of the algebraic development for the record, in case it may be useful elsewhere, and an outline of suggested algorithms.

Box and Draper (1971) pose practical arguments in favour of D-optimality:

1. It forces experimenters to give some thought to the model to be postulated before experiments are actually done.
2. The number of observations is not restricted in any way so long as it is sufficient (the information matrix is non-singular). Extra observations can be added to the design so long as the experimenter chooses.
3. Since the search for the best design can be made over any specified region in the design variables, regions of special shape can be handled.

These arguments are particularly apposite with respect to asymmetric factorials. The first argument has been considered in earlier chapters when dealing with the design requirements (see algorithm ENFAC). The second answers the problem with which we opened this chapter: so long as we can find the best smallest basic subdesign, then we can add any number of points we like to it. The third argument also suits us. The special shape of the region specified by our design variables, is that it must have as many dimensions as there are contrasts to be estimated and the variable represented in each of these dimensions can be set at one of only two values. This matter of coding the contrasts, and an algorithm to effect it, will be described fully in section three of this chapter. In that section I shall also present a new contribution to this field: an algorithm for choosing the best smallest basic subdesign. This will be followed by the full sequential algorithm for building on to the basic subdesign and, in section four, some examples. Fortran listings are in appendix three.

2 Using the trace

The sum of the diagonal elements of a square matrix is called the trace of that matrix. A practical argument for the A-optimality criterion is: When a set of linear coefficients is estimated by least squares, the variance of each estimated coefficient is proportional to the corresponding diagonal element of the inverted information matrix. Hence if the experiment is designed by choosing r observations such that the trace of the inverted information matrix is no greater than the corresponding trace for any other subset of r observations, we may fairly assume that the variance of each estimated coefficient is reasonably close to its minimum. Thus we may take the trace as a measure of the precision with which the coefficients may be estimated from the experimental results.

This leads to the simply stated algorithm: given that a basic subdesign has already been chosen, search among all observation points not yet included in the design for the point whose inclusion would cause the maximum reduction in the trace.

This still leaves the problem of choosing the best smallest basic subdesign, since the above algorithm relies upon the non-singularity of the information matrix so that it can have an inverse. I had an idea, however, that a generalised inverse might be used so that the search could begin as soon as a single row had been chosen as an anchor point for the design. If this were possible, as it proved to be, then the algorithm would become:

1. choose any point on the periphery of the design region (for example, if all the variables are coded (0,1) then choose the point $\underset{\sim}{0}$);

2. although the information matrix of one design point is singular so that the trace of its inverse is non-existent, assume that the trace does exist and let it be T;

3. using the concept of a generalised inverse, test every non-included point in turn and find the point whose inclusion would cause the maximum reduction (ΔT) in T (it turns out that ΔT is a real quantity);

4. repeat step 3 until a basic design is achieved;

5. use a modification of step 3 with a normal inverse, instead of a generalised inverse, to augment the basic design until it has the required number of points.

In developing the algebra to support this algorithm, I also considered the possibility of removing points from the design. There are now four possibilities:

1. Stepping into a non-basic design;
2. Stepping out of a non-basic design;
3. Stepping into a basic design;
4. Stepping out of the basic design.

A further requirement is clearly a test for the basicity of a design. This emerges from the theory developed to provide the four stepping procedures. The following initial relationships are needed:

1. Partitioned inverse

It is well established and easy to demonstrate that if a square matrix $\underset{\sim}{A}$ is partitioned as

$$\underset{\sim}{A} = \begin{bmatrix} \underset{\sim}{A}_{11} & \underset{\sim}{A}_{12} \\ \underset{\sim}{A}_{21} & \underset{\sim}{A}_{22} \end{bmatrix} \qquad 7.1$$

then, if $\underset{\sim}{A}$ has an inverse $\underset{\sim}{A}^{-1}$,

$$\underset{\sim}{A}^{-1} = \left[\begin{array}{c|c} \underset{\sim}{A}_{11}^{-1} + \underset{\sim}{A}_{11}^{-1}\underset{\sim}{A}_{12}\underset{\sim}{M}^{-1}\underset{\sim}{A}_{21}\underset{\sim}{A}_{11}^{-1} & -\underset{\sim}{A}_{11}^{-1}\underset{\sim}{A}_{12}\underset{\sim}{M}^{-1} \\ \hline -\underset{\sim}{M}^{-1}\underset{\sim}{A}_{21}\underset{\sim}{A}_{11}^{-1} & \underset{\sim}{M}^{-1} \end{array}\right] \qquad 7.2$$

where $\underset{\sim}{M} = \underset{\sim}{A}_{22} - \underset{\sim}{A}_{21}\underset{\sim}{A}_{11}^{-1}\underset{\sim}{A}_{12}$

2. Generalised inverse

If a non-basic, or singular, matrix $\underset{\sim}{A}$ can be factored symmetrically as

$$\underset{\sim}{A} = \underset{\sim}{u}\underset{\sim}{u}' \qquad 7.3$$

where $\underset{\sim}{A}$ is n-square and $\underset{\sim}{u}$ is n by m, then a generalised inverse $\underset{\sim}{A}^*$ is defined as

$$\underset{\sim}{A}^* \hat{=} \underset{\sim}{u}\,(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\,\underset{\sim}{u}' \qquad 7.4$$

so that

$$\underset{\sim}{A}^*\underset{\sim}{A} = \underset{\sim}{u}\,(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\,\underset{\sim}{u}' \qquad 7.5$$

It will be noted that although the rank of $\underset{\sim}{A}$ is less than n, since $\underset{\sim}{A}$ is singular, provided the rank of $\underset{\sim}{A}$ is greater than or equal to m then the rank of $(\underset{\sim}{u}'\underset{\sim}{u})$ will be full so its inverse will exist. If, in fact, $\underset{\sim}{A}$ is basic, or non-singular, it can be shown that the above definition (equation 7.4) satisfies the usual algebra of inversion. Suppose in this case that $\underset{\sim}{A}$ can be similarly factored symmetrically:

$$\underset{\sim}{A} = \underset{\sim}{u}\underset{\sim}{u}' \qquad 7.3$$

so that
$$\underset{\sim}{A}^{-1} = (\underset{\sim}{u}\underset{\sim}{u}')^{-1} \qquad 7.6$$

Let
$$\underset{\sim}{A}{*}\underset{\sim}{A} = \underset{\sim}{A}^{-1}\underset{\sim}{A} \qquad 7.7$$

or
$$\underset{\sim}{u}\,(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\,\underset{\sim}{u}' = \underset{\sim}{I} \qquad 7.8$$

then
$$\underset{\sim}{u}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\,\underset{\sim}{u}' = \underset{\sim}{u}' \qquad 7.9$$

or
$$\underset{\sim}{u}' = \underset{\sim}{u}' \qquad 7.10$$

Thus the left hand side of 7.7 conforms with the right hand side so that the relationships 7.6 and 7.7 apply equally well to all factorable square matrices whether basic or non-basic.

3. Trace of a product

It is well known that if a square matrix $\underset{\sim}{A}$ is a product of two square matrices $\underset{\sim}{B}$ and $\underset{\sim}{C}$ such that

$$\underset{\sim}{A} = \underset{\sim}{B}\,\underset{\sim}{C} \qquad 7.11$$

then, although $\underset{\sim}{A} \neq \underset{\sim}{C}\,\underset{\sim}{B}$ necessarily,

$$\text{trace }(\underset{\sim}{A}) = \text{trace }(\underset{\sim}{B}\,\underset{\sim}{C}) = \text{trace }(\underset{\sim}{C}\,\underset{\sim}{B}) \qquad 7.12$$

That is: the components of a matrix product may be rotated when computing the trace.

4. Trace of a generalised inverse

Applying equation 7.12 to rotate the components of equation 7.4

$$\begin{aligned}\text{trace }(\underset{\sim}{A}^*) &= \text{tr}\left\{\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\right\}\\ &= \text{tr}\left\{\underset{\sim}{u}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\right\}\end{aligned}$$

Thus
$$\text{tr }(\underset{\sim}{A}^*) = \text{tr}\left\{(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\right\} \qquad 7.13$$

We now consider each of the four sequential design possibilities stated earlier and prove four corresponding theorems.

Theorem one: When a row is stepped into a non-basic design, the trace of the inverted information matrix is decreased by

$$\frac{\utilde{x}'\,\utilde{A}^*\,\utilde{x} + 1}{\utilde{x}'\,\utilde{A}^*\,\utilde{A}\,\utilde{x} - \utilde{x}'\utilde{x}}$$

where $\utilde{x}'$ is the row stepped in, $\utilde{A}$ is the information matrix, and $\utilde{A}^*$ is its generalised inverse.

Proof Let $\utilde{u}'$ be the non-basic sub-matrix of the full design matrix; that is, $\utilde{u}'$ represents the non-basic design.
Let $\utilde{x}'$ be a single row of the full design matrix, not yet included in the sub-design $\utilde{u}'$.
Let $\utilde{U}'$ be the augmented design: the sub-design plus the row $\utilde{x}'$.
Then

$$\utilde{U}' = \begin{bmatrix} \utilde{u}' \\ \utilde{x}' \end{bmatrix} \qquad 7.14$$

If $\utilde{A}$ is the cross products (or information) matrix of the initial sub-design $\utilde{u}'$

$$\utilde{A} = \utilde{u}\utilde{u}' \qquad 7.15$$

and the cross products matrix of the augmented design is

$$(\utilde{A} + \utilde{x}\utilde{x}') = \begin{bmatrix} \utilde{u} & \utilde{x} \end{bmatrix} \begin{bmatrix} \utilde{u}' \\ \utilde{x}' \end{bmatrix} = \utilde{U}\utilde{U}' \qquad 7.16$$

The generalised inverse of the augmented design cross products matrix is

$$\begin{aligned}(\utilde{A} + \utilde{x}\utilde{x}')^* &= \utilde{U}(\utilde{U}'\utilde{U})^{-1}(\utilde{U}'\utilde{U})^{-1}\utilde{U}' \qquad \text{(from 7.4)} \\ &= (\utilde{u} \quad \utilde{x})\begin{bmatrix} \utilde{u}'\utilde{u} & \utilde{u}'\utilde{x} \\ \utilde{x}'\utilde{u} & \utilde{x}'\utilde{x} \end{bmatrix}^{-1} \begin{bmatrix} \utilde{u}'\utilde{u} & \utilde{u}'\utilde{x} \\ \utilde{x}'\utilde{u} & \utilde{x}'\utilde{x} \end{bmatrix}^{-1} \begin{bmatrix} \utilde{u}' \\ \utilde{x}' \end{bmatrix} \\ &= \utilde{U}\,\utilde{D}^{-1}\,\utilde{D}^{-1}\,\utilde{U}' \qquad 7.17\end{aligned}$$

where $\utilde{D}$ is the inner product of the augmented design matrix $\utilde{U}$

$$\utilde{D} = \utilde{U}'\utilde{U} \qquad 7.18$$

Let

$$\utilde{D}^{-1} = \begin{bmatrix} \utilde{D}_1 & \utilde{D}_2 \\ \utilde{D}_2' & \utilde{D}_4 \end{bmatrix} \qquad 7.19$$

and, noting that $\utilde{D}_4$ is a scalar, apply equation 7.2 to give:

$$\utilde{D}_1 = (\utilde{u}'\utilde{u})^{-1} + (\utilde{u}'\utilde{u})^{-1}\utilde{u}'\utilde{x}\utilde{x}'\utilde{u}\,(\utilde{u}'\utilde{u})^{-1}\,M^{-1} \qquad 7.20$$

$$\utilde{D}_2 = -\,(\utilde{u}'\utilde{u})^{-1}\utilde{u}'\utilde{x}\,M^{-1} \qquad 7.21$$

$$D_4 = M^{-1} \qquad 7.22$$

where $M = \utilde{x}'\utilde{x} - \utilde{x}'\utilde{u}(\utilde{u}'\utilde{u})^{-1}\utilde{u}'\utilde{x}$

$$= \utilde{x}'\utilde{x} - \utilde{x}'\,\utilde{A}^*\,\utilde{A}\,\utilde{x} \qquad \text{(from 7.5)} \qquad 7.23$$

and M is a scalar.

Application to equation 7.17 first of equation 7.14 and then of equations 7.19 to 7.23 yields:

$$\begin{aligned}
\mathrm{tr}\left\{(\utilde{A} + \utilde{x}\utilde{x}')^*\right\} &= \mathrm{tr}\left\{(\utilde{U}'\utilde{U})^{-1}\right\} \\
&= \mathrm{tr}\left\{\utilde{D}^{-1}\right\} \\
&= \mathrm{tr}\left\{\utilde{D}_1\right\} + \mathrm{tr}\left\{\utilde{D}_4\right\} \\
&= \mathrm{tr}\left\{(\utilde{u}\utilde{u}')^{-1}\right\} + \mathrm{tr}\left\{(\utilde{u}'\utilde{u})^{-1}\utilde{u}'\utilde{x}\utilde{x}'\utilde{u}(\utilde{u}'\utilde{u})^{-1}M^{-1}\right\} + M^{-1} \\
&= \mathrm{tr}(\utilde{A}^*) + M^{-1}\left[\mathrm{tr}\left\{\utilde{u}(\utilde{u}'\utilde{u})^{-1}(\utilde{u}'\utilde{u})^{-1}\utilde{u}'\utilde{x}\utilde{x}'\right\} + 1\right] \\
&\qquad \text{(from 7.12 and 7.13)} \\
&= \mathrm{tr}(\utilde{A}^*) + M^{-1}\left[\mathrm{tr}(\utilde{A}^*\utilde{x}\utilde{x}') + 1\right] \qquad 7.24 \\
&\qquad \text{(from 7.4)}
\end{aligned}$$

Therefore the trace is decreased by

$$\mathrm{tr}(\utilde{A}^*) - \mathrm{tr}((\utilde{A} + \utilde{x}\utilde{x}')^*) = -M^{-1}\left[\mathrm{tr}(\utilde{A}^*\utilde{x}\utilde{x}') + 1\right] \qquad 7.25$$

But, by equation 7.12, $\mathrm{tr}(\utilde{A}^*\utilde{x}\utilde{x}') = \mathrm{tr}(\utilde{x}'\utilde{A}^*\utilde{x})$ 7.26

and since $\utilde{x}$ is a vector then $\utilde{x}'\utilde{A}^*\utilde{x}$ is a scalar

thus $\mathrm{tr}(\utilde{x}'\utilde{A}^*\utilde{x}) = \utilde{x}'\utilde{A}^*\utilde{x}$ 7.27

From 7.23, 7.25 and 7.27 the decrease in trace is

$$\Delta T = \frac{\utilde{x}'\utilde{A}^*\,\utilde{x} + 1}{\utilde{x}'\utilde{A}^*\utilde{A}\,\utilde{x} - \utilde{x}'\utilde{x}} \qquad 7.28$$

which was to be proved.

Note that in order to minimise the trace of $(\utilde{A} + \utilde{x}\utilde{x}')^*$ it is necessary to choose a row $\utilde{x}'$ to step into the design such that the quantity ΔT is a maximum.

Theorem two: When a row is stepped out of a non-basic design, the trace of the inverted insformation matrix is decreased by

$$\frac{\mathbf{x}'\ \mathbf{A}^*\ \mathbf{x}\ -\ 1}{\mathbf{x}'\ \mathbf{A}^*\ \mathbf{A}\ \mathbf{x}\ -\ \mathbf{x}'\mathbf{x}}$$

where $\mathbf{x}'$ is the row stepped out, $\mathbf{A}$ is the information matrix, and $\mathbf{A}^*$ is its generalised inverse.

Proof Let $\mathbf{u}'$ be the non-basic sub-matrix of the full design matrix; that is, $\mathbf{u}'$ represents the non-basic design, with a cross-products matrix $\mathbf{A} = \mathbf{uu}'$.

Let $\mathbf{x}'$ be a row contained in $\mathbf{u}'$ which may be stepped out of $\mathbf{u}'$ to form a decremented non-basic sub-matrix $\mathbf{U}'$.

The analysis requires a partitioning of $\mathbf{U}$ in terms of $\mathbf{u}$ and $\mathbf{x}$ so that the new cross-products matrix is:

$$\mathbf{UU}' = \mathbf{A} - \mathbf{xx}' \qquad 7.29$$

This may be done by using the operator i to represent the square root of minus one and partitioning as:

$$\mathbf{U} = (\mathbf{u} \quad i\mathbf{x}) \qquad 7.30$$

Thus

$$(\mathbf{A} - \mathbf{xx}') = \mathbf{UU}' = (\mathbf{u} \quad i\mathbf{x}) \begin{bmatrix} \mathbf{u}' \\ i\mathbf{x}' \end{bmatrix} \qquad 7.31$$

and from 7.11

$$\begin{aligned} \operatorname{tr}\left\{(\mathbf{A} - \mathbf{xx}')^*\right\} &= \operatorname{tr}\left\{(\mathbf{U}'\mathbf{U})^{-1}\right\} \\ &= \operatorname{tr}\left\{\begin{bmatrix} \mathbf{uu}' & i\mathbf{u}'\mathbf{x} \\ i\mathbf{x}'\mathbf{u} & -\mathbf{x}'\mathbf{x} \end{bmatrix}^{-1}\right\} \qquad 7.32 \\ &= \operatorname{tr}(\mathbf{D}_1) + D_4 \quad \text{(as in proof of theorem one)} \end{aligned}$$

$$\text{where} \quad \mathbf{D}_1 = (\mathbf{u}'\mathbf{u})^{-1} + (\mathbf{u}'\mathbf{u})^{-1}\mathbf{u}'(i\mathbf{x})(i\mathbf{x})'\ \mathbf{u}(\mathbf{u}'\mathbf{u})^{-1}\ M^{-1} \qquad 7.33$$

$$D_4 = M^{-1} \qquad 7.34$$

$$\begin{aligned} \text{and} \quad M &= (i\mathbf{x}')(i\mathbf{x}) - (i\mathbf{x}')\mathbf{u}(\mathbf{u}'\mathbf{u})^{-1}\ \mathbf{u}'(i\mathbf{x}) \\ &= -\ (\mathbf{x}'\mathbf{x} - \mathbf{x}'\mathbf{A}^*\mathbf{Ax}) \qquad 7.34 \end{aligned}$$

Therefore

$$\begin{aligned} \operatorname{tr}\left\{(\mathbf{A} - \mathbf{xx}')^*\right\} &= \operatorname{tr}\left\{(\mathbf{u}'\mathbf{u})^{-1}\right\} + M^{-1}\left[\operatorname{tr}\left\{-(\mathbf{u}'\mathbf{u})^{-1}\mathbf{u}'\mathbf{xx}'(\mathbf{u}'\mathbf{u})^{-1}\right\} + 1\right] \\ &= \operatorname{tr}\left\{\mathbf{A}^*\right\} - \Delta T \qquad 7.35 \end{aligned}$$

$$\text{where} \quad \Delta T = \frac{\mathbf{x}'\ \mathbf{A}^*\ \mathbf{x}\ -\ 1}{\mathbf{x}'\ \mathbf{A}^*\ \mathbf{A}\ \mathbf{x}\ -\ \mathbf{x}'\mathbf{x}} \qquad 7.36$$

which was to be proved.

Theorem three: When a row is stepped into a basic design, the trace of the inverted information matrix is decreased by

$$\frac{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{A}^{-1}\underset{\sim}{x}}{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{x} + 1}$$

Proof Let $\underset{\sim}{u}'$ be the basic sub-matrix of the full design matrix; that is, $\underset{\sim}{u}'$ represents the basic design.
Let $\underset{\sim}{x}'$ be a single row of the full design matrix, not yet included in the sub-design $\underset{\sim}{u}'$, and let $\underset{\sim}{U}'$ be the augmented design, as in 7.14.
Let $\underset{\sim}{A}$ be the information matrix of the design $\underset{\sim}{u}'$, so that

$$\underset{\sim}{A} = \underset{\sim}{u}\underset{\sim}{u}' \qquad \text{which is non-singular} \tag{7.37}$$

and the information matrix of the augmented design is

$$\underset{\sim}{W} = \underset{\sim}{A} + \underset{\sim}{x}\underset{\sim}{x}' \tag{7.38}$$

There is a well known theorem, for which a novel proof is presented in the next section of this chapter, that:

$$\begin{aligned}\underset{\sim}{W}^{-1} &= (\underset{\sim}{A} + \underset{\sim}{x}\underset{\sim}{x}')^{-1} \\ &= \underset{\sim}{A}^{-1} - \underset{\sim}{A}^{-1}\underset{\sim}{x}\,(\underset{\sim}{I} + \underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x})^{-1}\underset{\sim}{x}'\underset{\sim}{A}^{-1}\end{aligned} \tag{7.39}$$

Since $\underset{\sim}{x}'$ is a single row $\underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x}$ is a scalar

thus, from 7.39,

$$\operatorname{tr}\left\{(\underset{\sim}{A} + \underset{\sim}{x}\underset{\sim}{x}')^{-1}\right\} = \operatorname{tr}\left\{\underset{\sim}{A}^{-1}\right\} - \frac{\operatorname{tr}\left\{\underset{\sim}{A}^{-1}\underset{\sim}{x}\underset{\sim}{x}'\underset{\sim}{A}^{-1}\right\}}{1 + \underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x}} \tag{7.40}$$

Hence, applying 7.12 to the numerator of the second term, the trace is decreased by

$$\frac{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x}}{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x} + 1} \tag{7.41}$$

which proves the theorem.

Theorem four: When a row is stepped out of a basic design, the trace of the inverted information matrix is decreased by

$$\frac{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{A}^{-1}\,\underset{\sim}{x}}{\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{x} - 1} \tag{7.42}$$

The proof follows from the proof of theorem three by changing signs appropriately.

The four trace changes demonstrated in the four theorems may be used in an algorithm for stepping rows into and out of a design, provided it is known when the design is basic or non-basic. Since the denominators in 7.28 and 7.36 are finite only when $\underset{\sim}{A}^*\underset{\sim}{A} \neq \underset{\sim}{I}$, a suitable test for basicity is to test each diagonal element of $\underset{\sim}{A}^*\underset{\sim}{A}$ (or $\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'$) against unity.

Examination of the four trace changes shows that quantities to be computed are:

$$\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{x} \tag{7.43}$$

$$\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{A}\,\underset{\sim}{x} \tag{7.44}$$

$$\underset{\sim}{x}'\,\underset{\sim}{x} \tag{7.45}$$

$$\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x} \tag{7.46}$$

$$\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x} \tag{7.47}$$

The easiest of these quantities to compute is $\underset{\sim}{x}'\,\underset{\sim}{x}$. Although I leave a full description of the coding of quantitative variables until the next section of this chapter (section 3: D-optimal algorithms) it is useful to note at this stage that the coding leads to $\underset{\sim}{x}'$ being represented as a row with a one in the first element, a one somewhere for each contrast, and zeros elsewhere. In section 3, algorithms CONTRA and DROW are developed to produce this coding. The result of this coding is that

$$\underset{\sim}{x}'\,\underset{\sim}{x} = NC + 1 \tag{7.48}$$

where NC is the number of contrasts to be estimated.

From equations 7.3 to 7.10 where it was shown that the generalised inverse satisfies the requirements of a non-singular inverse, 7.43 and 7.46 are equivalent. Expressing these in terms of the sub-design matrix $\underset{\sim}{u}$:

$$\begin{aligned}\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{x} &= \underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x}\\ &= \underset{\sim}{x}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\underset{\sim}{x}\end{aligned} \tag{7.49}$$

Similarly, expressing 7.44 in terms of $\underset{\sim}{u}$:

$$\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{A}\,\underset{\sim}{x} = \underset{\sim}{x}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\underset{\sim}{x} \qquad 7.50$$

And, expressing 7.47 in terms of $\underset{\sim}{u}$:

$$\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\,\underset{\sim}{A}^{-1}\,\underset{\sim}{x} = \underset{\sim}{x}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\underset{\sim}{x}$$

$$= \left[\underset{\sim}{x}'\underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\right](\underset{\sim}{u}'\underset{\sim}{u})^{-1}\left[(\underset{\sim}{u}'\underset{\sim}{u})^{-1}\underset{\sim}{u}'\underset{\sim}{x}\right] \qquad 7.51$$

Thus, by putting

$$\underset{\sim}{v} = (\underset{\sim}{u}'\underset{\sim}{u})^{-1} \qquad 7.52$$

$$\underset{\sim}{w} = \underset{\sim}{u}(\underset{\sim}{u}'\underset{\sim}{u})^{-1} = \underset{\sim}{u}\underset{\sim}{v} \qquad 7.53$$

The quantities to be computed are:

$$\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{x} = \underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{x} = (\underset{\sim}{x}'\underset{\sim}{w})(\underset{\sim}{w}'\underset{\sim}{x}) \qquad 7.54$$

$$\underset{\sim}{x}'\,\underset{\sim}{A}^*\,\underset{\sim}{A}\,\underset{\sim}{x} = (\underset{\sim}{x}'\underset{\sim}{w})(\underset{\sim}{u}'\underset{\sim}{x}) \qquad 7.55$$

$$\underset{\sim}{x}'\,\underset{\sim}{A}^{-1}\underset{\sim}{A}^{-1}\,\underset{\sim}{x} = (\underset{\sim}{x}'\underset{\sim}{w})\,\underset{\sim}{v}\,(\underset{\sim}{w}'\underset{\sim}{x}) \qquad 7.56$$

and, in the same terms, the test for basicity is to test in turn against unity the diagonal elements of $\underset{\sim}{w}\,\underset{\sim}{u}'$ 7.57

One problem that arises in computing these quantities is that when the design become basic,and the number of rows is hence equal to or greater than the number of columns, the dimensions of the matrices $\underset{\sim}{v}$ and $\underset{\sim}{w}$ continue to increase. However, instead of computing the quantities in terms of $\underset{\sim}{v}$ and $\underset{\sim}{w}$, they may then be computed in terms of $\underset{\sim}{A}^{-1}$ which will be a square matrix of dimension NC1. When a new row $\underset{\sim}{x}'$ is entered into the design, $\underset{\sim}{A}^{-1}$ is augmented by applying equation equation 7.39.

Using these computed quantities, the algorithm to select a reduced fraction of the balanced fraction produced by **MULFAC** may be described. This is the algorithm **TRADES** (to produce a **TRA**ce based **DES**ign), as follows:

Algorithm TRADES (TRAce based DESign)

```
Step 0  enter with number of factors N, number of levels for
        each factor NL(.), number of generators NG, the
        set of generators ID(.,.), generated design size NO,
        number of contrasts to be estimated NC, the set of
        contrasts ICON(.,.), and NC1 (NC + 1)
Step 1  enter ND (design size wanted $\geqslant$ NC1)
Step 2  for I ← 1 to NO set BITS(I) ← .FALSE. (to indicate that
        no rows have yet been entered into the design)
Step 3  choose row 1 as the first row in the design
        set NI ← 1
        the design matrix at this stage is a single row
        $u' = (1,0, . . . , 0)$
        so the matrix $v$ (equation 7.52) has a single element = 1
        and the matrix $w$ (equation 7.53) = $u$
        set BITS(1) ← .TRUE.

Step 4  (design is still non-basic)
        set NI ← NI + 1;
        for J ← 1 to NO
           if BITS(J) = .FALSE. then use algorithms DROW and LEV
           to find a design row IX(.)            $(x')$
           and compute DELT ← $((x'w)(w'x) + 1)/((x'w)(u'x) - NC1)$
           and hence find the value of J (JMAX) such that
           DELT is maximum
Step 5  set BITS(JMAX) ← .TRUE.
        increment dimension of matrix $v$
        and compute new matrix $v$   $(u'u)^{-1}$     (by equation 7.52)
              and new matrix $w$   $(uv)$         (by equation 7.53)
        test diagonal elements of $wu'$ for equality to unity
        if design still non-basic then goto step 3 else goto step 6
Step 6  (design is now basic)

        Let $u'$ = set of rows for which BITS(J) = .TRUE.
               (those rows that are now in the design)
        compute $A^{-1}$ ← $(uu')^{-1}$
```

```
Step 7   set NI ← NI + 1;  if NI > ND then goto step 9
         for J ← 1 to NO
             if BITS(J) = .FALSE. then use algorithms DROW and LEV
             to find a design row IX(•)        ($\underset{\sim}{x}'$)
             and compute DELT ← $(\underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{A}^{-1}\underset{\sim}{x})/(\underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x} + 1)$
             and hence find the value of J (JMAX) such that
             DELT is maximum

Step 8   set BITS(JMAX) ← .TRUE.
         compute augmented $\underset{\sim}{A}^{-1}$ by applying equation (7.39)
         goto step 7

Step 9   for J ← 1 to NO
             if BITS(J) = .TRUE. then use algorithm DROW
             to compute levels of factors for each design row,
             hence print design with ND rows, as required.

         randomise order
```

3 D-optimal algorithms

In this section I shall develop all the detail needed for algorithm REDDES (REDduce DESign) based on the D-optimality criterion. In outline, the algorithm must have the following steps:

1. Given the set of generators computed in MULFAC (see chapter six), find the subset of all possible design rows that will represent the best smallest basic sub-design.
2. Augment the sub-design, one row at a time from the remaining rows, until the specified design size has been achieved.

Several problems remain to be solved. The first is to develop an algorithm which will map a design row into a fully coded row of values of variables representing all contrasts to be estimated. The two alternative codings that may be used for the least squares analysis of categorical data are described fully by Scheffé (1959). Both use zeros and ones to represent absence or presence of a factor level. One method is to have a dummy variable to represent every level of each factor or required interaction and then to add dummy rows to the observation matrix to represent the constraints that the effects of the levels of each factor, or interaction, must sum to zero. The second, algebraically equivalent, method deals with the constraints by having one less variable for each factor than the number of levels for that factor. The second method is computationally preferable.

For example, consider a 2 x 3 x 4 design with the three factors labelled A,B, and C respectively and the requirement to estimate three main effects and the AB interaction, the second method would need one dummy variable to represent the contrast between the two levels of factor A: also called the main effect of factor A. Two dummy variables are needed for the main effect of factor B: one represents the contrast between level 2 and level 1, and the second represents the contrast between level 3 and level 1. Similarly three dummy variables are needed for the main effect of factor C: one to represent the contrast between level 2 and level 1; the second to represent the contrast between level 3 and level 1; and the third for the contrast between level 4 and level 1.

The interaction AB needs two dummy variables: one to represent the comparison of the effect of A with the first B contrast, and the second to represent the comparison of the effect of A with the second B contrast. One further dummy variable, with a constant value of one, is needed to represent the general mean of all observed values of the dependent variable. Thus, even without requiring the AC and BC interactions, this design calls for eight dummy variables, representing the contrasts, plus one for the mean.

Two algorithms are needed to deal with this coding. One of these, CONTRA, determines how many contrasts are needed, given the number of levels for each factor, and the requirements of the design. It also sets up arrays (ICON(I,J)) to specify the contrasts, where I = 1 to NC (the number of contrasts) and J = 1 to N (number of factors).

In the example, the first contrast, the main effect of A, would be represented by: ICON(1,1) = 1 ICON(1,2) = 0 ICON(1,3) = 0
The second and third contrasts, those of factor B, are represented
by: ICON(2,1) = 0 ICON(2,2) = 1 ICON(2,3) = 0
ICON(3,1) = 0 ICON(3,2) = 2 ICON(3,3) = 0

There are three similar rows of ICON(.,.) to represent the contrasts of factor C.
The two remaining contrasts, for the AB interaction, are represented
by: ICON(7,1) = 1 ICON(7,2) = 1 ICON(7,3) = 0
ICON(8,1) = 1 ICON(8,2) = 2 ICON(8,3) = 0

The pattern of values is similar that of the observation matrix generated by MULFAC. The similarity led to the use of generators to generate the values of ICON(.,.) in algorithm CONTRA. The full algorithm follows with a flowchart (figure 40).

The second algorithm, DROW, uses the values of ICON(.,.) to convert a design row, IK(.), into a fully coded row of values of variables representing all the contrasts to be estimated II(J), J = 1 to NC. The method is to count the number of times a non-zero value of ICON(I,J) is equal to the value of IK(J) for all J = 1 to N. If this value is odd then the value of II(I) is coded 1. If it is even then the value of II(I) is coded 0.

In the example, consider the design row, or vector, 1 2 0. This indicates, as described in chapter six, that the factor A is at its second level, the factor B is at its third level, and the factor C is at its first level. It is represented by the vector IK(.) as: IK(1) = 1 IK(2) = 2 IK(3) = 0.
The first contrast, represented by the first row of the array ICON(.,.), has the following equalities and inequalities with IK(.):

ICON(1,1) = IK(1) ICON(1,2) $\neq$ IK(2) ICON(1,3) = 0
There is one non-zero equality: an odd number. Therefore code II(1) = 1.

Further similar comparisons give the complete design row codes, apart from the dummy variable representing the mean, as follows:

II(1) = 1 II(2) = 0 II(3) = 1 II(4) = 0
II(5) = 0 II(6) = 0 II(7) = 1 II(8) = 0

This illustrates a feature of using an odd number of equalities as the criterion for coding a value of 1. The value assigned to the dummy variable representing an interaction is contrary to the value that may have been expected from the earlier decription. Thus the interaction indicated by the factor levels would be AB_2 rather than AB_1; hence we may have expected II(7) = 0 and II(8) = 1. The fact that the odd number criterion leads to contrary values is not important since the choice of values 0 and 1 to represent the absence and presence of a contrast was arbitrary: contrary values are just as satisfactory.

The dummy variable representing the mean is introduced in the main algorithm REDDES by using an EQUIVALENCE statement. In the Fortran program: EQUIVALENCE(IY(2),II(1)) and IY(1) = 1 ensure that in a vector IY(.), representing the full set of dummy variables, the first value is permanently set to 1 and the subsequent values are those of the II(.) vector.

The full algorithm, DROW, follows with a flowchart (figure 41).

Algorithm CONTRA (establish the CONTRAsts corresponding to the required main effects and interactions)

```
Step 0    enter with the number of factors N, the number of levels
          for each factor NL(.), the number of requirements NV,
          and the requirements set MV(.)

Step 10   set I ← 0; I1 ← 0   (find main effect contrasts)
Step 11   set I ← I + 1; L ← 0
Step 12   if I > N then goto step 30 fi
Step 13   set L ← L + 1
Step 14   if L = NL(I) then goto step 11 fi
Step 15   set I1 ← I1 + 1; J ← 0
Step 16   set J ← J + 1
Step 17   if J > N then goto step 19 fi
Step 18   set ICON(I1,J) ← 0; goto step 16
Step 19   set ICON(I1,I) ← L; goto step 13

Step 30   (find number of contrasts NN related to an interaction MV(I) )
          if I > NV then goto step 80
                    else set J ← 0; NN ← 1   fi
Step 31   set J ← J + 1
Step 32   if J > N then goto step 35 fi
Step 33   if itest(MV(I), J) = 0 then goto step 31 fi
Step 34   set NN ← NN * (NL(I) - 1); goto step 31

Step 35   (set up contrast generators IDT(.,.) and their orders ILT(.)
          set KG ← 0; J ← 0
Step 36   set J ← J + 1
Step 37   if J > N then goto step 46 fi
Step 38   if itest(MV(I), J) = 0 then goto step 36 fi
Step 39   set KG ← KG + 1; L ← 0
Step 40   set L ← L + 1
Step 41   if L > N then goto step 36 fi
Step 42   if L ≠ J then goto step 45 fi
Step 44   set IDT(KG,L) ← 1; ILT(KG) ← NL(J); goto step 40
Step 45   set IDT(KG,L) ← 0; goto step 40
Step 46  (use generators to create interaction contrasts)
         set IJ← 0
Step 47  set IJ←IJ + 1
Step 48  if IJ > NN then goto step 30 fi
Step 49  set I1←I1 + 1 ;  J←0
Step 50  set J←J + 1
Step 51  if J > N then goto step 55 fi
Step 52  set ICON(I1, J)←0 ;  goto step 50
Step 55  set K←IJ;   J← 0
Step 56  set J←J + 1
Step 57  if J > KG then goto step 69 fi
Step 58  set L←(K - 1)/ILT(J) ;  M←K - ILT(J) * L
Step 60  set K←L + 1 ;  JJ←0
Step 62  set JJ←JJ + 1
Step 63  if JJ > N then goto step 56 fi
Step 64  set KK←IDT(J,JJ) * M ;  L←NL(JJ) ;  LL←KK - L * (KK/L) ;
         ICON(I1,JJ)← MOD(ICON(I1,JJ), LL, L) ;  goto step 62
Step 69  set J← 0
Step 70  set J←J + 1
Step 71  if J > N then goto step 75 fi
Step 72  if itest(MV(I), J) = 0 then goto step 70 fi
Step 73  if ICON(I1, J) ≠ 0 then goto step 70 fi
Step 74  set I1← I1 - 1 ;  goto step 47
Step 75  set I←I + 1 ;  goto step 30

Step 80  set NC←I1 ;  NC1←NC + 1 ;  return
```

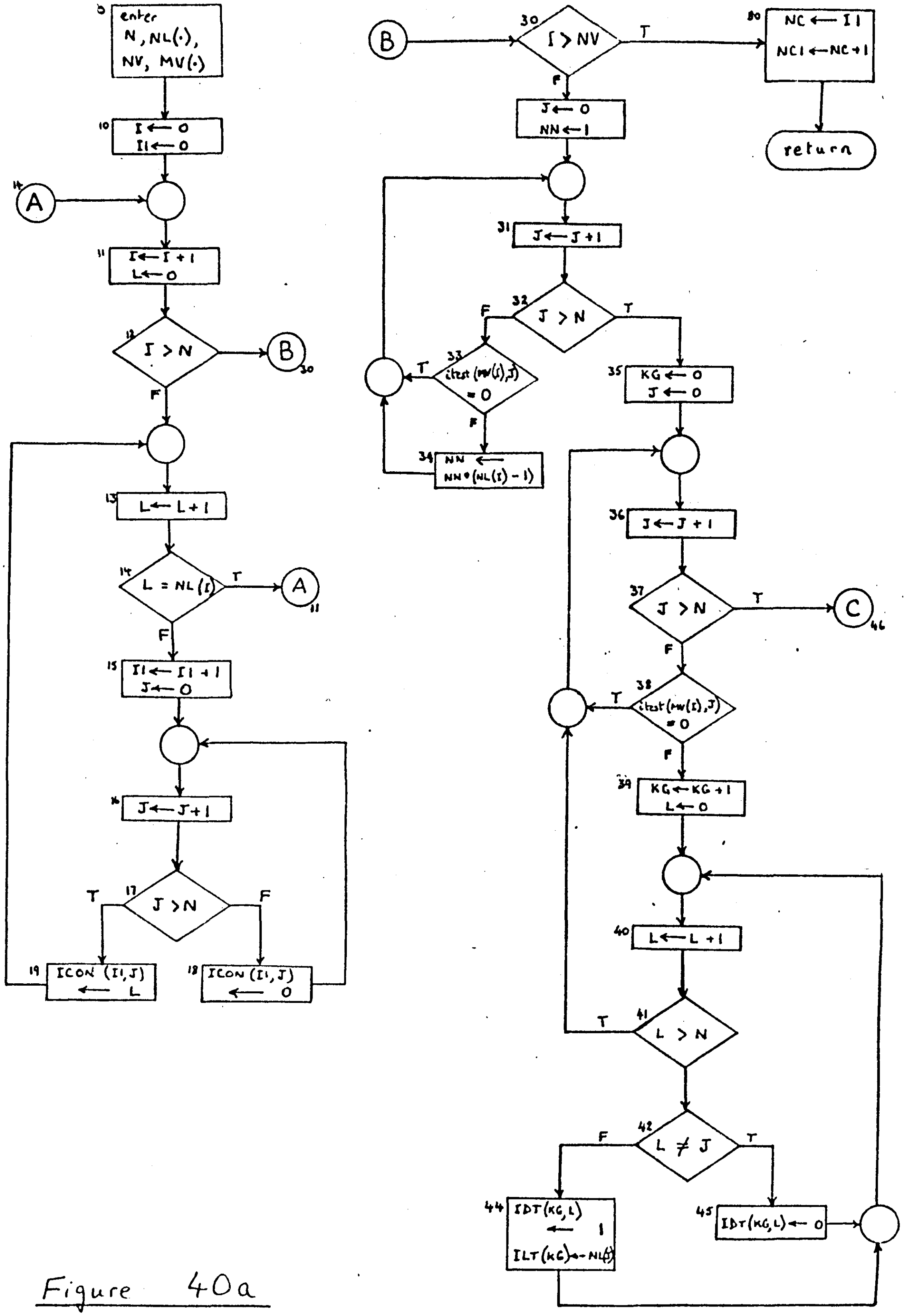
Algorithm DROW (find the complete Design ROW, II(.), corresponding to the observation vector at a point, IK(.), given the matrix of contrasts, ICON(.,.))

```
Step  0   enter with  design row  IK(.),  number of factors  N,
          matrix of contrasts  ICON(.,.),  and number of contrasts NC

Step  1   set  I←0
Step  2   set  I←I + 1
Step  3   if  I > NC  then  return  fi
Step  4   set  II(I)←0 ;  L←0;  J←0
Step  5   set  J←J + 1
Step  6   if  J > N  then  goto step 11  fi
Step  7   if  ICON (I,J) = 0  then  goto  step 5  fi
Step  8   if  IK(J) = 0  then goto  step 5  fi
Step  9   if  ICON(I,J) ≠  IK(J)  then  goto step 2  fi
Step 10   set  L←L + 1 ;   goto step 5
Step 11   set  II(I)←ITEST(L,1);  goto step 2
```

Whereas the two algorithms, CONTRA, and DROW, are elements of the set of experimental design algorithms, they would clearly be useful in the automatic coding of dummy variables for the analysis of the experimental results. This application will be a component of further work.

The algorithm CONTRA needs to be used only once in the complete design procedure. One of its results is the value of NC, the number of contrasts. The smallest number of observations needed to construct a basic design is NC + 1. Since this value is needed early in algorithm REDDES, it was more convenient to call CONTRA before entering REDDES. I have therefore changed the Fortran program representing algorithm MULFAC so that it calls CONTRA. This small change is shown in the new listing of the program MULFAC in appendix three.



0
enter
N, NL(·),
NV, MV(·)
10
I ← 0
I1 ← 0
14
A
11
I ← I + 1
L ← 0
12
I > N
B
30
F
13
L ← L + 1
14
L = NL(I)
T
A
11
F
15
I1 ← I1 + 1
J ← 0
16
J ← J + 1
17
J > N
T
F
19
ICON (I1, J)
← L
18
ICON (I1, J)
← 0
B
30
I > NV
T
F
80
NC ← I1
NC1 ← NC + 1
return
J ← 0
NN ← 1
31
J ← J + 1
32
J > N
F
T
33
itest (MV(I), J)
= 0
T
F
34
NN ←
NN * (NL(J) − 1)
35
KG ← 0
J ← 0
36
J ← J + 1
37
J > N
T
C
46
F
38
itest (MV(I), J)
= 0
T
F
39
KG ← KG + 1
L ← 0
40
L ← L + 1
41
L > N
T
42
L ≠ J
F
T
44
IDT(KG, L)
← 1
ILT(KG) ← NL(J)
45
IDT(KG, L) ← 0

Figure 40a

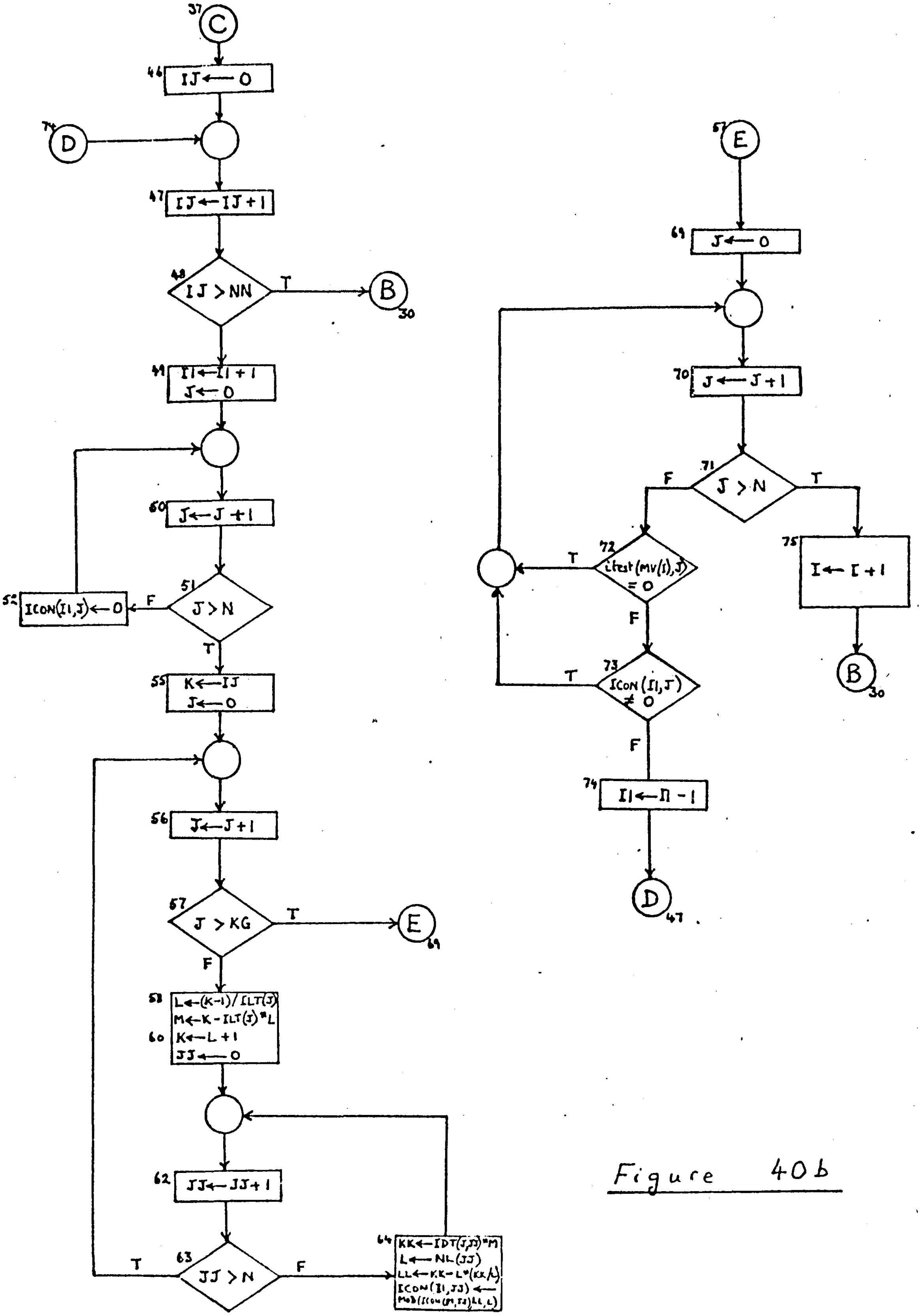

37
C
46
IJ ← 0
74
D
47
IJ ← IJ + 1
48
IJ > NN
T
B
30
49
I1 ← I1 + 1
J ← 0
50
J ← J + 1
51
J > N
F
52
ICON(I1,J) ← 0
T
55
K ← IJ
J ← 0
56
J ← J + 1
57
J > KG
T
E
69
F
58
L ← (K-1)/ILT(J)
M ← K - ILT(J)*L
60
K ← L + 1
JJ ← 0
62
JJ ← JJ + 1
63
JJ > N
T
F
64
KK ← IDT(J,JJ)*M
L ← NL(JJ)
LL ← KK - L*(KK/L)
ICON(I1,JJ) ←
MOD(ICON(I1,JJ),LL,L)
57
E
69
J ← 0
70
J ← J + 1
71
J > N
F
T
72
Itest(MV(I),J) = 0
T
F
73
ICON(I1,J) ≠ 0
T
F
74
I1 ← I1 - 1
D
47
75
I ← I + 1
B
30

Figure 40b

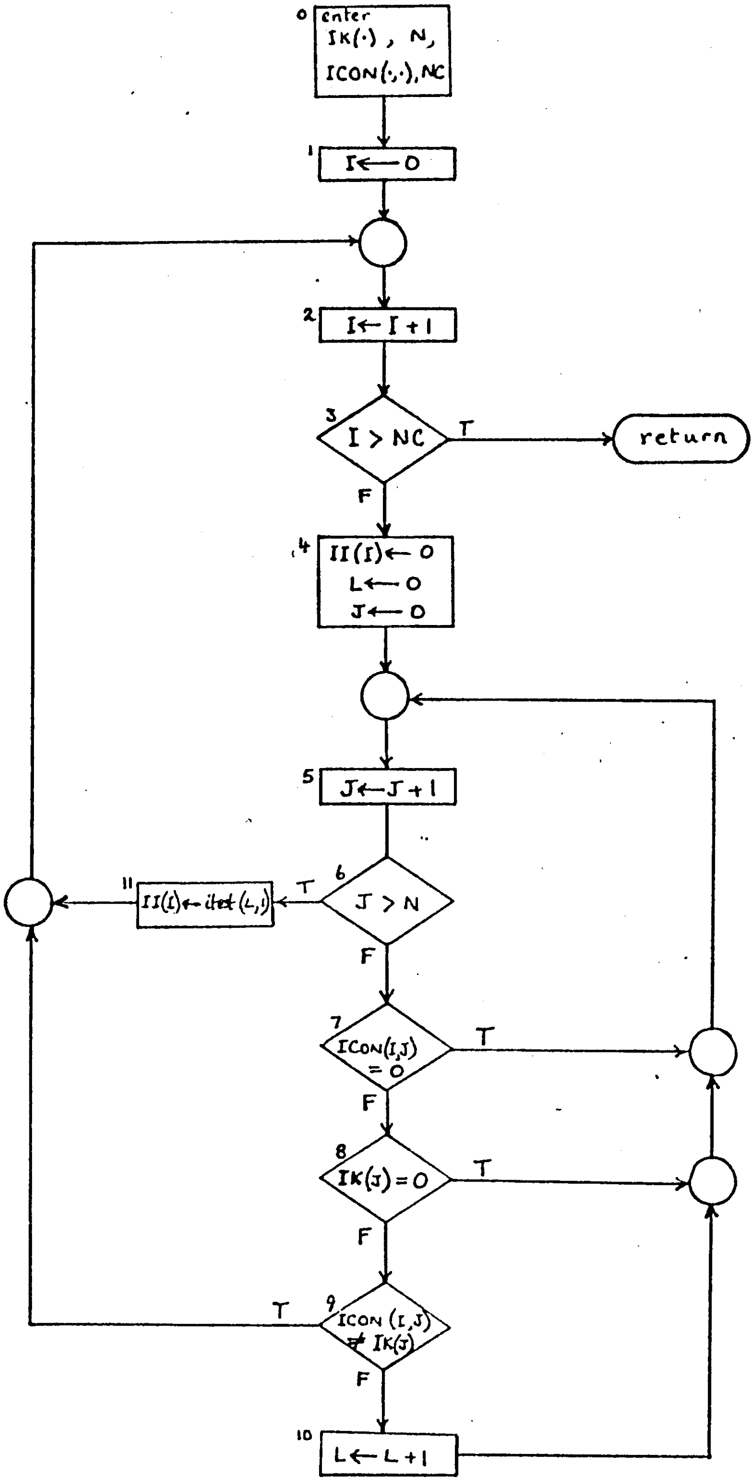

0
enter
IK(·) , N,
ICON(·,·), NC
1
I ← 0
2
I ← I + 1
3
I > NC
T
return
F
4
II(I) ← 0
L ← 0
J ← 0
5
J ← J + 1
6
J > N
T
11
II(I) ← itet (L,1)
F
7
ICON(I,J) = 0
T
F
8
IK(J) = 0
T
F
9
ICON (I,J) ≠ IK(J)
T
F
10
L ← L + 1

Figure 41

We need an algorithm to select the subset of all possible design rows that will represent the best smallest basic sub-design. That is, we need to find NC + 1 rows, where NC is the number of contrasts to be estimated, such that when all dummy variables have been fully coded, using algorithm DROW, a square matrix $\mathbf{X}$ is formed. The determinant of $(\mathbf{X}'\mathbf{X})^{-1}$ must be a minimum: that is, there must be no other (NC + 1)-subset of rows for which the corresponding determinant is less.

Since the matrix $\mathbf{X}$ will be square, this criterion is equivalent to the criterion that the determinant of $\mathbf{X}$ must be a maximum. In order to meet this criterion, we need the following theorem:

<u>Theorem five</u>: When the r-th row, $\mathbf{x}'_r$, of a non-singular matrix $\mathbf{X}$, is replaced by a new row, $\mathbf{a}$, then the determinant of the new matrix $\mathbf{X}_a$ is equal to the determinant of the matrix $\mathbf{X}$ times y_r where y_r is the r-th element of the vector $\mathbf{y}$ such that $\mathbf{a} = \mathbf{X}'\mathbf{y}$. Also, the inverse of $\mathbf{X}_a$ is equal to the inverse of $\mathbf{X}$ times $\mathbf{E}'$ where $\mathbf{E}$ is the identity matrix with the r-th column replaced by the vector $\boldsymbol{\eta} = \left(\frac{-y_1}{y_r}, \ldots, \frac{1}{y_r}, \ldots, \frac{-y_k}{y_r}\right)'$ and k is the matrix order.

<u>Proof</u> Given a square matrix $\mathbf{X}$ of order k, with an inverse $\mathbf{X}^{-1}$, replace the r-th row $\mathbf{x}'_r$ by a new row $\mathbf{a}$. Let the vector $\mathbf{y}$ be such that

$$\mathbf{a} = \mathbf{X}'\mathbf{y} \qquad 7.43$$

that is: $\mathbf{a} = \sum y_i \mathbf{x}_i$ with summation over i 7.44

and each $\mathbf{x}_i$ is the i-th column of $\mathbf{X}$.

or:

$$\mathbf{a}' = \mathbf{y}'\mathbf{X} \qquad 7.45$$

then, from 7.44,

$$y_r \mathbf{x}_r = \mathbf{a} - \sum_{i \neq r} y_i \mathbf{x}_i \qquad 7.46$$

and

$$\mathbf{x}_r = -\frac{y_1}{y_r}\mathbf{x}_1 - \ldots + \frac{1}{y_r}\mathbf{a} - \ldots - \frac{y_k}{y_r}\mathbf{x}_k \qquad 7.47$$

$$= \mathbf{X}'_a \boldsymbol{\eta} \qquad 7.48$$

where

$$\boldsymbol{\eta} = \left(\frac{-y_1}{y_r}, \ldots, \frac{1}{y_r}, \ldots, \frac{-y_k}{y_r}\right)' \qquad 7.49$$

Let $\underset{\sim}{E} = \underset{\sim}{e}_1, \underset{\sim}{e}_2, \ldots, \underset{\sim}{\eta}, \ldots, \underset{\sim}{e}_k$ 7.50

where $\underset{\sim}{e}_i$ is a column vector with 1 in the i-th position and 0 elsewhere and $\underset{\sim}{\eta}$ is the r-th column vector.

then, from 7.48 and 7.50,

$$\underset{\sim}{X}' = \underset{\sim}{X}'_a \underset{\sim}{E} \qquad 7.51$$

hence $$\underset{\sim}{X}_a^{-1} = \underset{\sim}{X}^{-1} \underset{\sim}{E}' \qquad 7.52$$

and $$\det(\underset{\sim}{X}_a) = \det(\underset{\sim}{X})/\det(\underset{\sim}{E}) \qquad 7.53$$

now $\det(\underset{\sim}{E})$ = sum of products of elements of the vector and their cofactors. 7.54

Let C_i = cofactor of η_i the i-th element of $\underset{\sim}{\eta}$

$$= (-1)^{i+r}\det(\underset{\sim}{M}_{ir}) \qquad 7.55$$

where $\det(\underset{\sim}{M}_{ir})$ is the minor of η_i

hence $$\det(\underset{\sim}{E}) = \eta_i C_i \qquad 7.56$$

Since the columns of $\underset{\sim}{E}$ are the vectors $\underset{\sim}{e}_i$, except the r-th column which is $\underset{\sim}{\eta}$, then the cofactors of all elements of $\underset{\sim}{\eta}$, except for that in which i = r, are zero, and the cofactor of the r-th element of $\underset{\sim}{\eta}$ is 1.

So, from 7.56, $$\det(\underset{\sim}{E}) = \eta_r = \frac{1}{y_r} \qquad 7.57$$

and, from 7.53 and 7.57,

$$\det(\underset{\sim}{X}_a) = \det(\underset{\sim}{X})\, y_r \qquad 7.58$$

Thus the theorem is proved.

It is useful to note that, from 7.45, $\underset{\sim}{y} = (\underset{\sim}{X}^{-1})' \underset{\sim}{a}$ 7.59

which is a step in calculating $\underset{\sim}{\eta}$ from 7.49.

Also note that r may be chosen such that y_r and hence $\det(\underset{\sim}{X}_a)$ are maxima.

If we are now able to choose a (NC + 1)-subset of rows from the full design matrix such that they form a non-singular square matrix $\underset{\sim}{X}$, we may apply the theorem to exchange a row that is in the sub-design with a row that is not in. Repeated application will lead to the best smallest basic sub-design. We can be sure of a non-singular starting matrix by choosing rows such that the j-th choice has a 1 in the j-th element and zeros to the right of it. An example will be given. Thus the outline algorithm for finding the best NC + 1 rows is:

step 1 choose a non-singular starting matrix $\underset{\sim}{X}$, set $\underset{\sim}{A} \leftarrow \underset{\sim}{X}$

step 2 set $\underset{\sim}{B} \leftarrow \underset{\sim}{X}^{-1}$ and DETA $\leftarrow$ det($\underset{\sim}{X}$)
(an algorithm will be given for this)

step 3 find the next row to enter (one that has not yet been in the subdesign); let this vector $\underset{\sim}{a}'$ be the array IX(.)

step 4 calculate the vector $\underset{\sim}{y} = (\underset{\sim}{X}^{-1})' \underset{\sim}{a}$; call this array Y(.)

step 5 find the maximum element of $\underset{\sim}{y}$; the r-th element y_r

step 6 exchange the r-th row of $\underset{\sim}{X}$ with the vector $\underset{\sim}{a}'$

step 7 find the new determinant: set DETA $\leftarrow y_r *$ DETA

step 8 compute η using 7.49, and hence $\underset{\sim}{E}$

step 9 find the new matrix inverse using 7.52

step 10 return to step 3 unless there are no more rows to be tested.

At this stage it is instructive to consider a simple example; this helps to identify some of the minor computational nuances.

Consider the full 2^3 factorial design with dummy variables representing the mean and the three main effect contrasts. For simplicity I exclude interactions:

$$\begin{array}{cccc|l}
1 & 0 & 0 & 0 & \text{row 1} \\
1 & 1 & 0 & 0 & \text{row 2} \\
1 & 0 & 1 & 0 & \text{row 3} \\
1 & 1 & 1 & 0 & \text{row 4} \\
1 & 0 & 0 & 1 & \text{row 5} \\
1 & 1 & 0 & 1 & \text{row 6} \\
1 & 0 & 1 & 1 & \text{row 7} \\
1 & 1 & 1 & 1 & \text{row 8}
\end{array}$$

The procedure described above leads to the choice of rows 1,2,3, and 5 as the non-singular starting matrix.

Thus the algorithm leads through the following steps:

step 1 $$\underset{\sim}{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

step 2 $$\underset{\sim}{X}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \qquad \det(\underset{\sim}{X}) = 1$$

step 3 next row to try is row 4 = 1 1 1 0
set IN ← 4

step 4 $\underset{\sim}{y}' = (-1 \quad 1 \quad 1 \quad 0)$

step 5 first maximum value from the left is $y_2 = 1$
set OUT ← 2

step 6 $$\underset{\sim a}{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}$$

step 7 $\det(\underset{\sim a}{X}) = y_2 * \det(\underset{\sim}{X}) = 1$

step 8 $\underset{\sim}{\eta}' = (1 \quad 1 \quad -1 \quad 0)$

step 9 $$\underset{\sim a}{X}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix}$$

step 3 (matrices $\underset{\sim a}{X}$ and $\underset{\sim a}{X}^{-1}$ now become $\underset{\sim}{X}$ and $\underset{\sim}{X}^{-1}$)

next row to try is row 6 = 1 1 0 1
set IN ← 6

step 4 $\underset{\sim}{y}' = (0 \quad 1 \quad -1 \quad 1)$

step 5 first maximum value from the left is $y_2 = 1$, but the second row has already been changed once, so look for the next : $y_4 = 1$

set OUT ← 4

(this suggests an improvement to the algorithm: keep a tally of the times a row has been moved out of the sub-matrix, and if two rows otherwise equally qualify then move out that for which the tally is lesser)

step 6 $$\underset{\sim a}{X} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

step 7 $\det(\underset{\sim a}{X}) = y_4 * \det(\underset{\sim}{X}) = 1$

step 8 $\underset{\sim}{\eta}' = (0 \quad -1 \quad 1 \quad 1)$

step 9 $$\underset{\sim}{X}_a^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 1 \end{pmatrix}$$

step 3 next row to try is row 7 = 1 0 1 1
set IN ⟵ 7

step 4 $$\underset{\sim}{y}' = (-1 \quad -1 \quad 2 \quad 1)$$

step 5 maximum value is $y_3 = 2$
set OUT ⟵ 3

step 6 $$\underset{\sim}{X}_a = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

step 7 $$\det(\underset{\sim}{X}_a) = 2$$

step 8 $$\underset{\sim}{\eta}' = \left(\tfrac{1}{2} \quad \tfrac{1}{2} \quad \tfrac{1}{2} \quad -\tfrac{1}{2} \right)$$

step 9 $$\underset{\sim}{X}_a^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

step 3 next row to try is row 8 = 1 1 1 1

step 4 $$\underset{\sim}{y}' = \left(-\tfrac{1}{2} \quad \tfrac{1}{2} \quad \tfrac{1}{2} \quad \tfrac{1}{2} \right)$$

thus any row chosen to move out would lead to a decrease in the determinant, so stop

(the test for stopping to be included in the algorithm is that there should be no value in $\underset{\sim}{y}$ greater than or equal to 1)

In this example it is encouraging to note that the final half-design at step 6 is the balanced half design that would be achieved by the method of chapter three. In general, however, the best NC + 1 rows will not constitute a balanced fraction.

The second stage of the main algorithm REDDES, to augment the sub-design one row at a time, follows the procedure used by Goldsmith (1974) in which the row chosen is such that the increase in det(X'X) is a maximum. The algorithm uses the following two well known theorems for which a novel joint proof is given.

<u>Theorem six</u>: If a square matrix $\underset{\sim}{A}$, whose inverse is $\underset{\sim}{A}^{-1}$, is augmented by the vector product $\underset{\sim}{x}\underset{\sim}{x}'$, then the inverse of the augmented matrix is

$$(\underset{\sim}{A} + \underset{\sim}{x}\underset{\sim}{x}')^{-1} \quad = \quad \underset{\sim}{A}^{-1}(\underset{\sim}{I} - \underset{\sim}{x}\underset{\sim}{x}'\underset{\sim}{A}^{-1}/d)$$

$$\text{where} \quad d \quad = \quad (1 + \underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x})$$

<u>Theorem seven</u>: If a square matrix $\underset{\sim}{A}$, whose determinant is $\det(\underset{\sim}{A})$ and inverse is $\underset{\sim}{A}^{-1}$, is augmented by the vector product $\underset{\sim}{x}\underset{\sim}{x}'$, then the determinant of the augmented matrix is

$$\det(\underset{\sim}{A} + \underset{\sim}{x}\underset{\sim}{x}') \quad = \quad d*\det(\underset{\sim}{A})$$

$$\text{where} \quad d \quad = \quad (1 + \underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x})$$

<u>Proof</u> Dempster (1969) uses the sweep operator for matrix inversion. He shows that if a matrix $\underset{\sim}{A}$ is partitioned as

$$\underset{\sim}{A} \quad = \quad \left(\begin{array}{c:c} \underset{\sim}{a}_{11} & \underset{\sim}{a}_{12} \\ \hdashline \underset{\sim}{a}_{21} & \underset{\sim}{a}_{22} \end{array}\right) \tag{7.60}$$

where the $\underset{\sim}{a}_{ij}$ are sub-matrices, then the matrix may be inverted in a succession of sweeps defined as follows

$$\left(\begin{array}{c:c} \underset{\sim}{a}_{11} & \underset{\sim}{a}_{12} \\ \hdashline \underset{\sim}{a}_{21} & \underset{\sim}{a}_{22} \end{array}\right) \xrightarrow{(k)} \left(\begin{array}{c:c} \underset{\sim}{b}_{11} & \underset{\sim}{b}_{12} \\ \hdashline \underset{\sim}{b}_{21} & \underset{\sim}{b}_{22} \end{array}\right) \tag{7.61}$$

where

$$\left.\begin{aligned} \underset{\sim}{b}_{kk} &= -\underset{\sim}{a}_{kk}^{-1} \\ \underset{\sim}{b}_{ik} &= \underset{\sim}{a}_{ik}\underset{\sim}{a}_{kk}^{-1} \\ \underset{\sim}{b}_{kj} &= \underset{\sim}{a}_{kk}^{-1}\underset{\sim}{a}_{kj} \\ \underset{\sim}{b}_{ij} &= \underset{\sim}{a}_{ij} - \underset{\sim}{a}_{ik}\underset{\sim}{a}_{kk}^{-1}\underset{\sim}{a}_{kj} \end{aligned}\right\} \tag{7.62}$$

where k may be given successive values of 1,2 or 2,1 with the same resultant inverse.

Consider now a matrix partitioned as $\left(\begin{array}{c|c} \utilde{A} & \utilde{x} \\ \hline \utilde{x}' & 1 \end{array}\right)$ and invert it using the sweep operator in each of the two sequences. Thus

$$\left(\begin{array}{c|c} \utilde{A} & \utilde{x} \\ \hline \utilde{x}' & 1 \end{array}\right) \xrightarrow{(1)} \left(\begin{array}{c|c} -\utilde{A}^{-1} & \utilde{A}^{-1}\utilde{x} \\ \hline \utilde{x}'\utilde{A}^{-1} & 1 - \utilde{x}'\utilde{A}^{-1}\utilde{x} \end{array}\right)$$

$$\xrightarrow{(2)} \left(\begin{array}{c|c} -\utilde{A}^{-1} - \utilde{A}^{-1}\utilde{x}(1 + \utilde{x}'\utilde{A}^{-1}\utilde{x})^{-1}\utilde{x}'\utilde{A}^{-1} & -\utilde{A}^{-1}\utilde{x}(1 - \utilde{x}'\utilde{A}^{-1}\utilde{x})^{-1} \\ \hline (1 + \utilde{x}'\utilde{A}^{-1}\utilde{x})^{-1}\utilde{x}'\utilde{A}^{-1} & -(1 + \utilde{x}'\utilde{A}^{-1}\utilde{x})^{-1} \end{array}\right) \qquad 7.63$$

and

$$\left(\begin{array}{c|c} \utilde{A} & \utilde{x} \\ \hline \utilde{x}' & 1 \end{array}\right) \xrightarrow{(2)} \left(\begin{array}{c|c} \utilde{A} + \utilde{x}\utilde{x}' & -\utilde{x} \\ \hline \utilde{x}' & -1 \end{array}\right)$$

$$\xrightarrow{(1)} \left(\begin{array}{c|c} -(\utilde{A} + \utilde{x}\utilde{x}')^{-1} & -(\utilde{A} + \utilde{x}\utilde{x}')^{-1}\utilde{x} \\ \hline \utilde{x}'(\utilde{A} + \utilde{x}\utilde{x}')^{-1} & -1 + \utilde{x}'(\utilde{A} + \utilde{x}\utilde{x}')^{-1}\utilde{x} \end{array}\right) \qquad 7.64$$

Now, since the two inverses are identical, the top left component of one may be equated with the top left component of the other. Hence

$$\begin{aligned} (\utilde{A} + \utilde{x}\utilde{x}')^{-1} &= \utilde{A}^{-1} - \utilde{A}^{-1}\utilde{x}(1 + \utilde{x}'\utilde{A}^{-1}\utilde{x})^{-1}\utilde{x}'\utilde{A}^{-1} \\ &= \utilde{A}^{-1}(\utilde{I} - \utilde{x}\utilde{x}'\utilde{A}^{-1}/d) \qquad 7.65 \end{aligned}$$

$$\text{where } d = (1 + \utilde{x}'\utilde{A}^{-1}\utilde{x})$$

which proves theorem six.

The determinant of the starting matrix is equal to the product of the determinants of the pivoting sub-matrices. Hence by equating determinants by inspection of the two sequences:

$$\begin{aligned} \det(\utilde{A} + \utilde{x}\utilde{x}') &= \det(\utilde{A})*\det(1 + \utilde{x}'\utilde{A}^{-1}\utilde{x}) \\ &= d*\det(\utilde{A}) \qquad 7.66 \end{aligned}$$

which proves theorem seven.

These theorems now complete the development needed to construct algorithm REDDES, which follows. The flowchart is in figure 42. An algorithm INVERT, to invert a square non-singular matrix, also follows with a flowchart in figure 43.

The main variables to be used in algorithm REDDES are:

N	the total number of factors to be included
BITS(I)	a bit array indicating inclusion or exclusion of a row in the design (*3200)
NL(I)	the number of levels of the I'th factor (*16)
NG	the number of generators
ID(I,J)	the J'th integer of the I'th generator (*8,16)
NO	generated design size = product of orders of generators
NC	number of contrasts to be estimated
NC1	NC + 1
ICON(I,J)	the J'th integer of the I'th contrast (*20,20)
ND	design size (number of rows) wanted
IY(I)	I'th integer of fully (dummy) coded design row (*20)
II(I)	I'th integer of row of contrasts (see algorithm step 85 for equivalence with IY(.)) (*20)
IK(I)	level of I'th factor in a design row (*16)
A(I,J)	the (I,J)th element of a square matrix (*20,20)
B(I,J)	the (I,J)th element of the inverse of matrix A (*20,20)
C(I,J)	a copy of B(I,J) (*20,20)
ETA(I)	the I'th element of vector ETA (*20)
DETA	determinant of matrix A
Y(I)	the I'th value in the vector Y used in the row exchange procedure (*20)
IC(I)	the I'th value of the vector IC used in the row exchange procedure to keep a tally of the number of times a row has been exchanged (*20)
IA(I)	the I'th integer in the vector IA, used in the row exchange procedure to record which design row is in the I'th row of the initial basic design. (*20)

Practical dimensions of arrays are denoted by (*n)

Algorithm REDDES (REDuce DESign)

Step 0 enter with number of factors N, number of levels for each factor NL(.), number of generators NG, the set of generators ID(.,.), generated design size (product of generator orders) NO, number of contrasts to be estimated NC, the set of contrasts ICON(.,.), and NC1 = (NC + 1)

Step 1 initialise and enter the design size wanted ND

Step 80 find a subset of NC1 rows from the full set of NO rows such that the NC1 square matrix is non-singular: A(.,.)

Step 120 invert the matrix into B(.,.)

Step 125 find the next row to enter, noting that bits have been set to 1 if rows have already been in the sub-design; use algorithms LEV and DROW to put the row into vector IY(.)

Step 130 calculate the vector Y(.) (B' * IY)

Step 140 find the maximum value of Y(.); if there are several equal maximum values, choose the first for which the tally IC(.) is the least; increment that IC(.)

Step 150 compute ETA ($\underset{\sim}{\eta}$) and DETA, the determinant of the new matrix

Step 160 compute the new matrix inverse C(.,.); copy C(.,.) into B(.,.); return to step 125, but if there are no more rows to test go to step 180

Step 180 write the basic design

Step 200 compute the cross-products matrix (set $\underset{\sim}{A} \longleftarrow \underset{\sim}{A}'\underset{\sim}{A}$) and the inverse cross-products matrix (set $\underset{\sim}{B} \longleftarrow \underset{\sim}{B}\underset{\sim}{B}'$) and the determinant of the cross products matrix (set DETA $\longleftarrow$ DETA*DETA)

Step 215 find the row for which the test quantity ($\underset{\sim}{x}'\underset{\sim}{A}^{-1}\underset{\sim}{x}$ from equation 7.66) is greatest. (computation is speeded by noting that all elements of a row (IY(.)) are either 0 or 1

Step 240 compute the new inverse using equation 7.65, then return to step 215 unless the required number of rows has been reached

Step 250 randomise the order of observations, then stop

The detailed algorithm follows.

Algorithm REDDES (REDuce DESign)

```
Step 0     enter with number of factors N,  number of levels for
           each factor NL(.),  number of generators NG,  the
           set of generators ID(.,.),  generated design size NO,
           number of contrasts to be estimated NC,  the set of
           contrasts ICON(.,.),  and  NC1  (NC + 1)
Step 1     enter ND  (design size wanted > NC1)
Step 5     for I ← 1 to NO  set BITS(I) ← .FALSE.
Step 80    set IY(1) ← 1;   JJJ ← 1;   K ← 0
Step 81    set I ← 0;  K ← K + 1;  if K > NO then goto step 98 fi
Step 82    set I ← I + 1;  if I > NO  then goto step 81  fi
Step 83    if BITS(I) = .TRUE.  then goto step 82  fi
Step 84    call algorithm LEV (to find vector IK(.) corresponding
           to the I'th row)
Step 85    call algorithm DROW (to map the vector IK(.) into the
           full row of contrasts II(.)  and hence,  by an equivalence
           statement  (in Fortran:  EQUIVALENCE(IY(2), II(1))) into IY(.))
Step 86    if IY(JJJ) ≠ 1  then goto step 82  fi
Step 87    set L ← JJJ
Step 88    set L ← L + 1
Step 89    if L > NC1  then goto step 92  fi
Step 90    if IY(L) ≠ 0  then goto step 82  else goto step 88  fi
           (steps 86 to 90 ensure that the rows selected constitute a
            non-singular matrix)
Step 92    set L ← 0
Step 93    set L ← L + 1
Step 94    if L > NC1  then goto step 96  fi
Step 95    set A(JJJ,L) ← IY(L);   goto step 93
Step 96    set IA(JJJ) ← I;   JJJ ← JJJ + 1;   BITS(I) ← .TRUE.
Step 97    if JJJ ≤ NC1  then goto step 81  fi
Step 98    set IC(1) ← 1000   (a high tally value ensures that the
           first row is never a candidate for removing from the design)
Step 99    for I ← 2 to NC1  set IC(I) ← 0
Step 100   (if any extra rows are needed to complete the square matrix,
            find those for which the diagonal element will be 1)
           if JJJ ≥ NC1  then goto step 120  fi
Step 101   set I ← 1
Step 102   set I←I + 1 ; if I > NO then goto step 101 fi
Step 103   set K←0
Step 104   set K←K + 1 ; if K > NC1 then goto step 109 fi
Step 105   if I = IA(K) then goto step 104 fi
Step 109   set BITS(I)←.FALSE.
Step 110   call algorithm LEV(I,IK)
Step 111   call algorithm DROW(IK,II)
Step 112   if IY(JJJ) ≠ 1 then goto step 102 fi
Step 113   set K←0
Step 114   set K←K + 1
Step 115   if K > NC1 then goto step 117 fi
Step 116   set A(JJJ,K)←IY(K); goto step 114
Step 117   set IA(JJJ)←I; BITS(I)←.TRUE. ; JJJ←JJJ + 1
Step 118   if JJJ ≤ NC1 then goto step 102 fi
Step 120   call algorithm INVERT (to invert square array A of
           size NC1 into array B and compute DETA, the determinant
           of A)
Step 125   set IN←1
Step 126   set IN←IN + 1
Step 127   if IN > NO then goto step 180 fi
Step 128   if BITS(IN) = .TRUE. then goto step 126 fi
Step 129   call algorithm LEV (IN,IK)
Step 130   call algorithm DROW(IK,II)
Step 131   set I←0; YMAX←-1000. ; ICMIN←1000
Step 133   set I←I + 1
Step 134   if I > NC1 then goto step 146 fi
Step 135   set Y(I)←0. ; J←0
Step 136   set J←J + 1
Step 137   if J > NC1 then goto step 140 fi
Step 138   set Y(I)←Y(I) + B(J,I)*IY(J); goto step 136
Step 140   if Y(I) < YMAX or Y(I) < 1 then goto step 133 fi
Step 141   if I = 1 then goto step 133 fi
Step 142   if Y(I) > YMAX then goto step 144 fi
Step 143   if IC(I) < ICMIN then goto step 145 else goto step 133 fi
Step 144   set ICMIN←IC(I)
Step 145   set YMAX←Y(I); IOUT←I; goto step 133
Step 146   if YMAX = -1000. then goto step 126 fi
Step 147   set J←IA(IOUT); IA(IOUT)←IN ; IC(IOUT)←IC(IOUT) + 1
Step 150   set DETA←YMAX * DETA; I←0
Step 151   set I←I + 1
Step 152   if I > NC1 then goto step 160 fi
Step 153   set ETA(I)← - Y(I)/YMAX
Step 154   if I = IOUT  then set ETA(I)←1.0/YMAX fi ; goto step 151
Step 160   set I←0
Step 161   set I←I + 1
Step 162   if I > NC1  then goto step 175 fi
Step 163   set  J←0
Step 164   set J←J + 1
Step 165   if   J > NC1  then goto step 161 fi
Step 166   set  C(I,J)←0.;  K←0
Step 167   set  K←K + 1
Step 168   if  K > NC1  then goto step 164 fi
Step 169   if K ≠  IOUT  then goto step 172  fi
Step 170   set  C(I,J)←C(I,J) + B(I,K)*ETA(J) ;  goto step 167
Step 172   if  K  ≠  J  then  goto step 167 fi
Step 173   set  C(I,J)←C(I,J) + B(I,K) ;  goto step 167
Step 175   call algorithm SWAP(B,C,NC1) ;   goto step 125
           (to copy the NC1 order C array into the B array)
Step 180   write heading  'Basic Design'
Step 181   set  NI← 0
Step 182   set  NI←NI + 1
Step 183   if  NI > NC1  then  goto step 190
Step 184   set  J← IA(NI)
Step 185   call  algorithm  LEV (J, IK)   (find the J-th row)
Step 186   write IK(L), L←1 to N         (print the J-th row)
Step 187   goto step 182
Step 190   write heading  'Extra Rows'
Step 191   set  I←0
Step 192   set  I←I + 1
Step 193   if  I > NO   then  goto step 200  fi
Step 194   set  BITS(I)← .FALSE.  ;   J← 0
Step 195   set  J←J + 1
Step 196   if  J > NC1  then  goto step 192  fi
Step 197   if  I  =  IA(J)  then do step 198 od else goto step 195 fi
Step 198   set  BITS(I)← .TRUE. ;   goto step 192
           (In the early part of the algorithm the array BITS(.) was
           used to record if a row had been in the basic sub-design
           at any time during the stepping in/out procedure.  Now the
           array records that a row is either in or out of the design)
Step 200   (set  A←A'A)  for  I←1 to NC1  do step 201  od
Step 201   for  J←1 to NC1  do step 202;  step 203  od
Step 202  set  C(I,J)←0.
Step 203  for  K←1 to NC1  do step 205  od
Step 205  set  C(I,J)← C(I,J) + A(K,I)*A(K,J)
Step 206  call algorithm  SWAP(A,C,NC1)
Step 207  (set  B←BB')  for I←1 to NC1  do step 208  od
Step 208  for  J←1 to NC1  do  step 209;  step 210 od
Step 209  set  C(I,J)←0
Step 210  for  K←1 to NC1  do  step  211  od
Step 211  set  C(I,J)← C(I,J) + B(I,K)*B(J,K)
Step 212  call algorithm  SWAP(B,C,NC1)
Step 213  set  DETA←DETA*DETA
Step 215  (find best row to enter)  set  I←1;  DTMAX←-1
Step 216  set  I←I + 1;  if  I > NO  then  goto step 233 fi
Step 217  if  BITS(I) = .TRUE.  then goto step 216  fi
Step 218  call algorithm  LEV(I,IK)
Step 219  call algorithm  DROW(IK,II)
Step 220  set  J←0;  DTEST←0
Step 222  set  J←J + 1;  if  J > NC1  then goto step  231  fi
Step 223  if  IY(J) = 0 then goto step 222  fi
Step 225  set  K←J;  DTEST← DTEST + B(J,J)
Step 227  set  K←K + 1;  if  K > NC1  then  goto step 222  fi
Step 228  if  IY(K) = 0  then  goto step  227  fi
Step 229  set  DTEST←DTEST + 2*B(J,K);  goto step  227
Step 231  if  DTEST < DTMAX  then  goto step 216  fi
Step 232  set  DTMAX←DTEST;  IN←I;  goto step 216
Step 233  call algorithm  LEV(IN,IK)
Step 234  set  BITS(IN)←.TRUE.
Step 235  write IK(L),  L←1 to N  (print the IN-th row)
Step 236  if  NI ≥ ND  then  goto step 250  fi
Step 237  set  NI← NI + 1;  D←1  +  DTMAX;  DETA←D * DETA
Step 238  call algorithm  DROW(IK,II)
Step 239  if  D ≤ 0.0001  then goto step  215  fi
Step 240  for  I←1 to NC1  do  set  Y(I)←0.  od
Step 241  set  I← 0
Step 242  set  I←I + 1;  if  I > NC1  then  goto step 246  fi
Step 243  if  IY(I)  = 0  then  goto step 242  fi
Step 244  for  J←1 to NC1  do  set  Y(J)←Y(J) + B(I,J) od; goto step 242
Step 246  for  I←1 to NC1  do step 247 od
Step 247  for  J←1 to NC1 do  set B(I,J)←B(I,J) - Y(I)*Y(J)/D  od
Step 248  goto  step 215
Step 250  call  algorithm  RANDOM ;    stop
```

```
Algorithm  INVERT   (INVERT matrix A of order M into matrix B
                      and compute derterminant of A:  D)

Step  0    enter with matrix A of order M
Step  1    set  $D \leftarrow 1.$ ;  $TH \leftarrow 10^{-7}$
Step  2    call algorithm SWAP (to copy A into B)
Step  3    set  $I \leftarrow 0$
Step  4    set  $I \leftarrow I + 1$ ;  if $I > M$ then return  fi
Step  5    set  $D \leftarrow D * B(I,I)$
Step  6    if $B(I,I) < TH$ then do set $D \leftarrow 0$; return od fi
Step  7    set  $B(I,I) \leftarrow 1./B(I,I)$;  $J \leftarrow 0$
Step  8    set  $J \leftarrow J + 1$;  if $J > M$ then goto step 11 fi
Step  9    if $J \neq I$ then set $B(I,J) \leftarrow B(I,J)*B(I,I)$ fi
Step 10    goto step 8
Step 11    set  $II \leftarrow 0$
Step 12    set  $II \leftarrow II + 1$ ;  if $II > M$ then goto step 18 fi
Step 13    if $II = I$ then goto step 12 fi
Step 14    set  $J \leftarrow 0$
Step 15    set  $J \leftarrow J + 1$;  if $J > M$ then goto step 12 fi
Step 16    if $J = I$ then goto step 15 fi
Step 17    set  $B(II,J) \leftarrow B(II,J) - B(I,J)*B(II,I)$;  goto step 15
Step 18    set  $J \leftarrow 0$
Step 19    set  $J \leftarrow J + 1$ ;  if $J > M$ then goto step 4 fi
Step 20    if $J \neq I$ then set $B(J,I) \leftarrow - B(J,I) * B(I,I)$ fi
Step 21    goto step 19
```

The full set of programs representing the algorithms of this chapter are in appendix three.

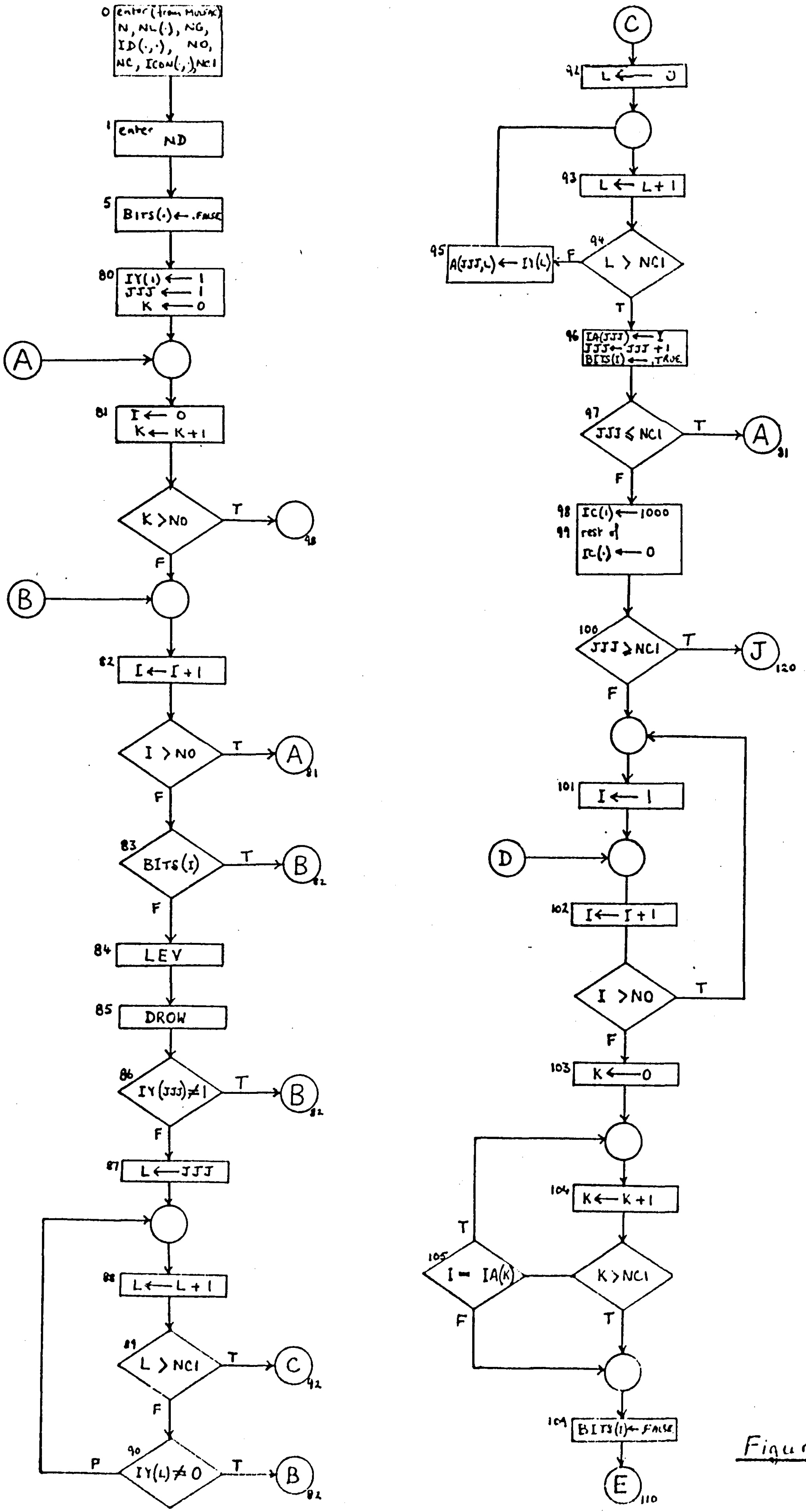

0
enter (from MUSAC)
N, NL(·), NG,
ID(·,·), NO,
NC, ICON(·,·), NC1
1
enter
ND
5
BITS(·) ← .FALSE.
80
IY(1) ← 1
JJJ ← 1
K ← 0
A
81
I ← 0
K ← K+1
K > NO
T
98
F
B
82
I ← I+1
I > NO
T
A
81
F
83
BITS(I)
T
B
82
F
84
LEY
85
DROW
86
IY(JJJ) ≠ 1
T
B
82
F
87
L ← JJJ
88
L ← L+1
89
L > NC1
T
C
92
F
90
IY(L) ≠ 0
F
T
B
82
C
92
L ← 0
93
L ← L+1
94
L > NC1
F
95
A(JJJ,L) ← IY(L)
T
96
IA(JJJ) ← I
JJJ ← JJJ+1
BITS(I) ← .TRUE.
97
JJJ ≤ NC1
T
A
81
F
98
IC(1) ← 1000
99
rest of
IC(·) ← 0
100
JJJ ≥ NC1
T
J
120
F
101
I ← 1
D
102
I ← I+1
I > NO
T
F
103
K ← 0
104
K ← K+1
K > NC1
105
I = IA(K)
T
F
T
109
BITS(I) ← .FALSE.
E
110

Figure 4.2 a

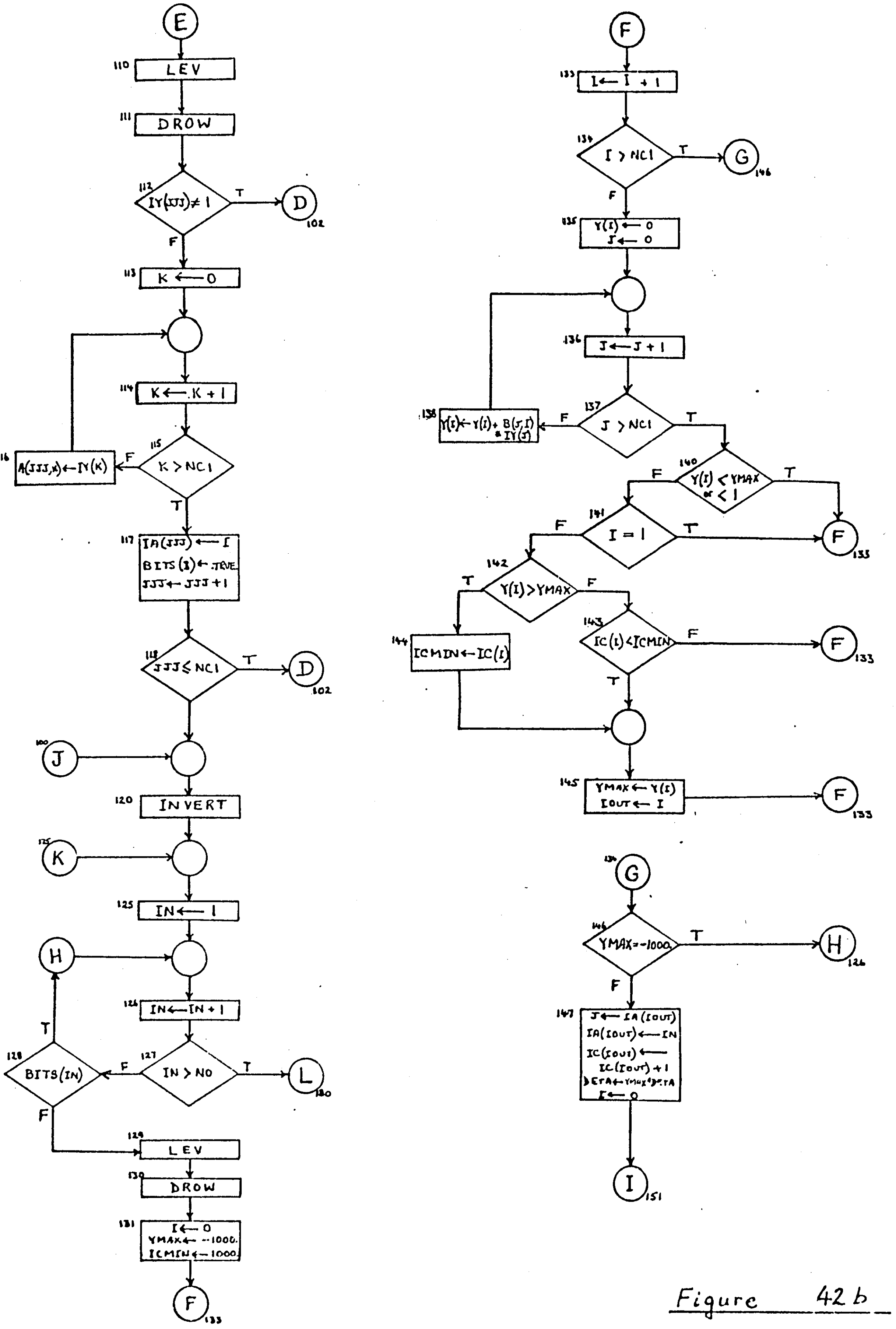

E
110
LEV
111
DROW
112
IY(JJJ)≠1
T
D
102
F
113
K ← 0
114
K ← K + 1
115
K > NC1
F
116
A(JJJ,K) ← IY(K)
T
117
IA(JJJ) ← I
BITS(I) ← .TRUE.
JJJ ← JJJ + 1
118
JJJ ≤ NC1
T
D
102
100
J
120
INVERT
125
K
125
IN ← 1
H
126
IN ← IN + 1
127
IN > NO
T
L
180
F
128
BITS(IN)
T
F
129
LEV
130
DROW
131
I ← 0
YMAX ← -1000.
ICMIN ← 1000.
F
133
F
133
I ← I + 1
134
I > NC1
T
G
146
F
135
Y(I) ← 0
J ← 0
136
J ← J + 1
137
J > NC1
F
138
Y(I) ← Y(I) + B(J,I) * IY(J)
T
140
Y(I) < YMAX or < 1
F
T
141
I = 1
F
T
F
133
142
Y(I) > YMAX
T
F
144
ICMIN ← IC(I)
143
IC(I) < ICMIN
F
F
133
T
145
YMAX ← Y(I)
IOUT ← I
F
133
134
G
146
YMAX = -1000.
T
H
126
F
147
J ← IA(IOUT)
IA(IOUT) ← IN
IC(IOUT) ← IC(IOUT) + 1
DETA ← YMAX*DETA
I ← 0
I
151

Figure 42b

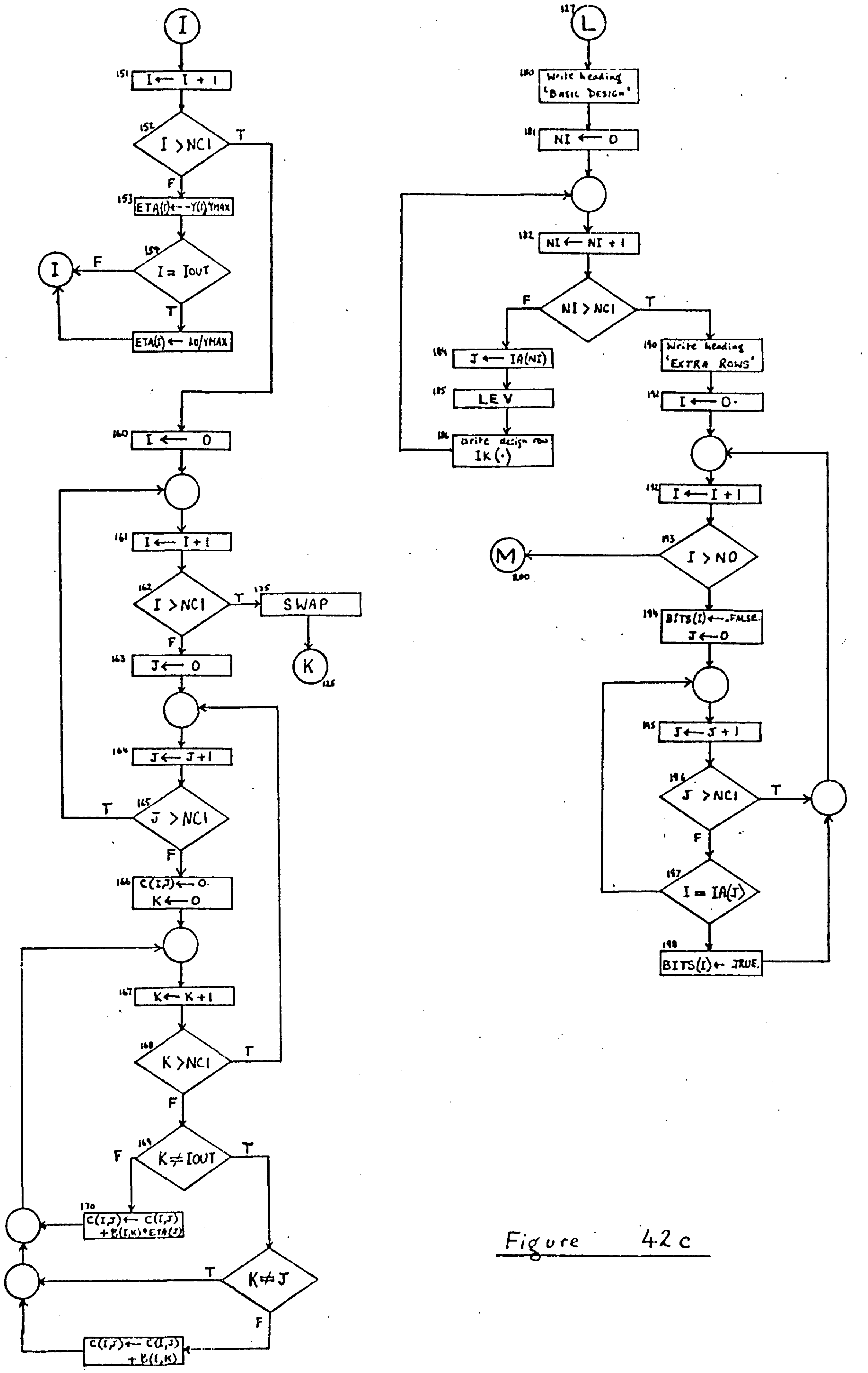

I
151
I ← I + 1
152
I > NC1
T
F
153
ETA(I) ← -Y(I)/YMAX
154
I = IOUT
F
I
T
ETA(I) ← 10/YMAX
160
I ← 0
161
I ← I + 1
162
I > NC1
T
175
SWAP
K
125
F
163
J ← 0
164
J ← J + 1
165
J > NC1
T
F
166
C(I,J) ← 0.
K ← 0
167
K ← K + 1
168
K > NC1
T
F
169
K ≠ IOUT
F
T
170
C(I,J) ← C(I,J) + B(I,K)*ETA(J)
K ≠ J
T
F
C(I,J) ← C(I,J) + B(I,K)
127
L
180
Write heading 'BASIC DESIGN'
181
NI ← 0
182
NI ← NI + 1
NI > NC1
F
T
184
J ← IA(NI)
185
LEV
186
write design row IK(·)
190
Write heading 'EXTRA ROWS'
191
I ← 0.
192
I ← I + 1
193
I > NO
M
200
194
BITS(I) ← .FALSE.
J ← 0
195
J ← J + 1
196
J > NC1
T
F
197
I = IA(J)
198
BITS(I) ← .TRUE.

Figure 42c

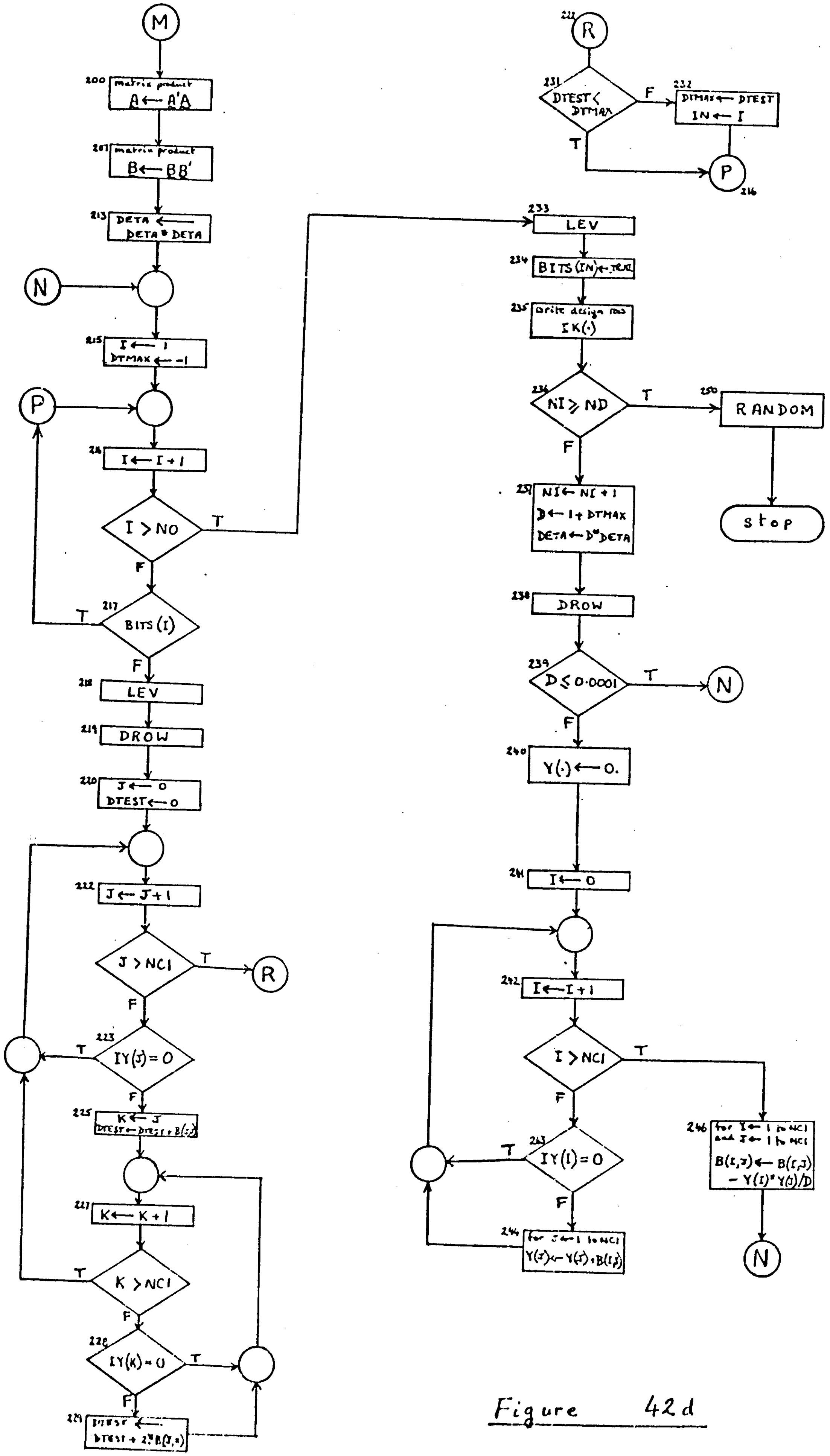

M
200
matrix product
A ← A'A
207
matrix product
B ← BB'
213
DETA ←
DETA * DETA
N
215
I ← 1
DTMAX ← -1
P
216
I ← I + 1
I > NO
T
F
217
BITS (I)
T
F
218
LEV
219
DROW
220
J ← 0
DTEST ← 0
222
J ← J + 1
J > NCI
T
R
F
223
IY(J) = 0
T
F
225
K ← J
DTEST ← DTEST + B(J,J)
227
K ← K + 1
K > NCI
T
F
228
IY(K) = 0
T
F
229
DTEST ←
DTEST + 2*B(J,K)
222
R
231
DTEST < DTMAX
F
232
DTMAX ← DTEST
IN ← I
T
P
216
233
LEV
234
BITS (IN) ← TRUE
235
write design row
IK(·)
236
NI ≥ ND
T
250
RANDOM
stop
F
237
NI ← NI + 1
D ← 1 + DTMAX
DETA ← D*DETA
238
DROW
239
D ≤ 0.0001
T
N
F
240
Y(·) ← 0.
241
I ← 0
242
I ← I + 1
I > NCI
T
F
243
IY(I) = 0
T
F
244
for J ← 1 to NCI
Y(J) ← Y(J) + B(I,J)
246
for I ← 1 to NCI
and J ← 1 to NCI
B(I,J) ← B(I,J)
− Y(I)*Y(J)/D
N

Figure 42d

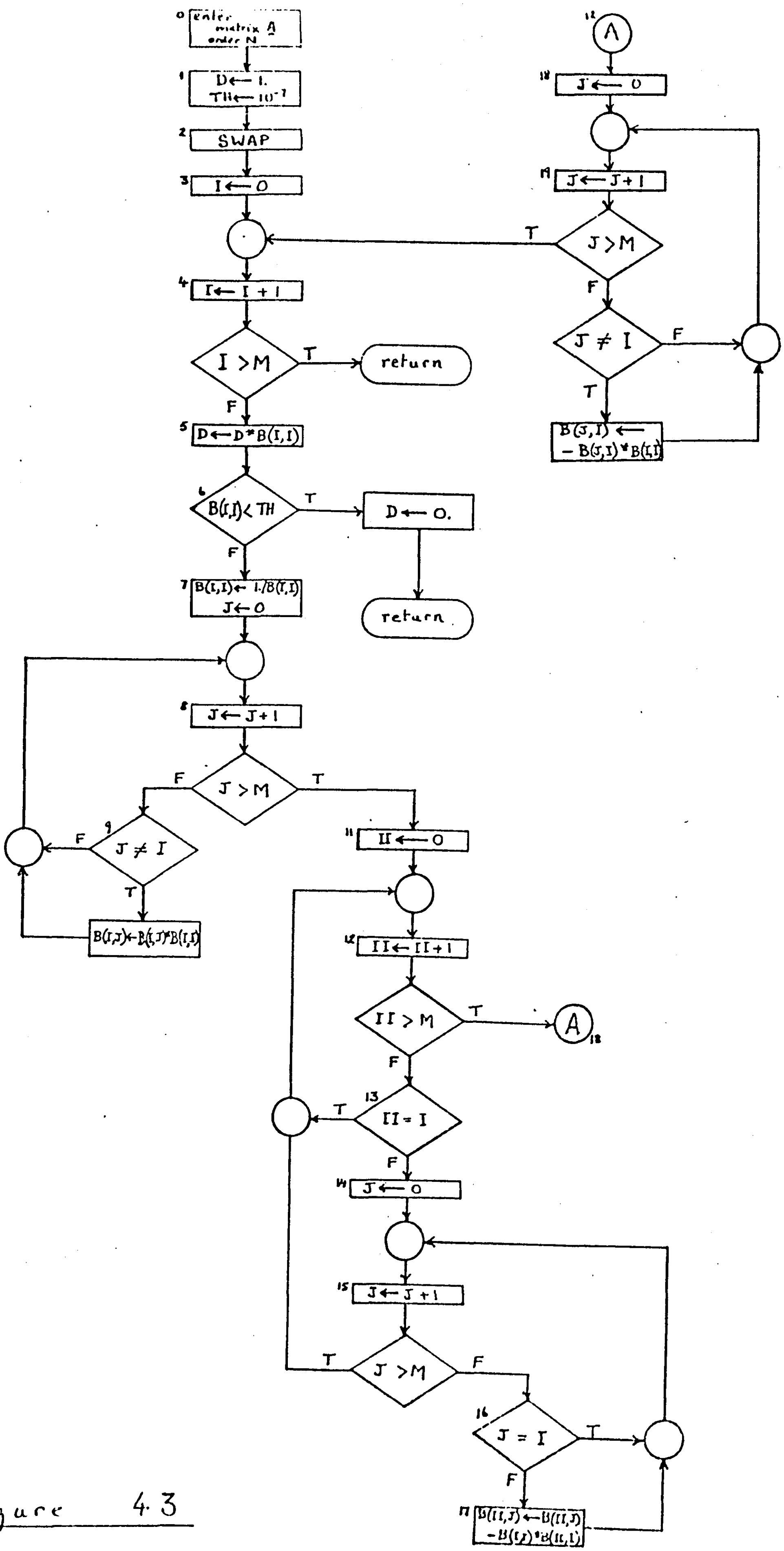

0 enter matrix A order N
1 D ← 1. TH ← 10^-7
2 SWAP
3 I ← 0
4 I ← I + 1
I > M
T
return
F
5 D ← D * B(I,I)
6 B(I,I) < TH
T
D ← 0.
return
F
7 B(I,I) ← 1./B(I,I) J ← 0
8 J ← J + 1
J > M
F
T
9 J ≠ I
F
T
B(I,J) ← B(I,J)*B(I,I)
11 II ← 0
12 II ← II + 1
II > M
T
A 18
F
13 II = I
T
F
14 J ← 0
15 J ← J + 1
J > M
T
F
16 J = I
T
F
17 B(II,J) ← B(II,J) - B(I,J)*B(II,I)
18 A
18 J ← 0
19 J ← J + 1
J > M
T
F
J ≠ I
F
T
B(J,I) ← - B(J,I)*B(I,I)

Figure 4.3

4 Examples

In this section I present a few examples chosen to illustrate the results of using the combined methods of chapters six and seven.

As a simple illustration, consider a 2 x 4 experiment without interactions. That is: there are two factors, one with two levels and one with four levels. Thus for the first factor there is one contrast to estimate and for the second there are three. The assumption that there are no interactions means that the contrasts of one factor are assumed to be the same at all levels of the other factor. The conversation at the computer terminal proceeds as:

```
Computer:    Enter title of experiment
User:        T24
Computer:    How many factors are there?
User:        2
Computer:    Enter required interaction
User:        (blank)
Computer:    For factor 1 type the number of levels
User:        2
Computer:    For factor 2 type the number of levels
User:        4
Computer:    Generators for T24
                           1 0
                           0 1
             Balanced design has 8 points.  At least 5 are needed.
             Type Y for balanced design,  else  N.
User:        N
Computer:    Enter ND: design size wanted
User:        6
Computer:    Basic design
                           0  0
                           1  0
                           0  1
                           0  2
                           0  3
             Extra rows
                           1  3
```

```
Computer:   Randomised order
                    5 4 1 3 2 6
            Is another random number stream wanted?  Type Y or N.
User:       N
```

Further examples are illustrated more briefly with a statement of the requirements, the full design size (product of the factor levels), the balanced fraction size (product of the generator orders), the design size wanted, the least number of rows in a basic design (number of contrasts plus one), and the extra rows. For the sake of brevity, the conversation and the random number stream are omitted.

Requirements: 2 x 6, no interactions.
Full design: 12 points. Balanced fraction: 12 points.
Design size wanted: 8 points. At least 5 needed.
Generators: 1 0 ; 0 1

```
Basic design:   0 0
                1 0
                0 1
                0 2
                0 3
                0 4
                0 5
Extra rows:     1 5
```

Requirements: 2 x 2 x 4, no interactions.
Full design: 16 points. Balanced fraction: 16 points.
Design size wanted: 8 points. At least 6 needed.
Generators: 1 0 3 ; 0 1 3

```
Basic design:   0 0 0
                1 1 0
                1 0 1
                1 0 3
                0 0 2
                0 1 3
Extra rows:     0 1 1
                1 1 2
```

Requirements: 2 x 2 x 2 x 3, no interactions.

Full design: 24 points. Balanced fraction: 12 points.

Design size wanted: 9 points. At least 6 needed.

Generators: 1 0 1 0 ; 0 1 1 0 ; 0 0 0 1

Basic design:
0 0 0 0
1 1 0 1
0 1 1 0
1 0 1 0
0 0 0 2
1 1 0 0

Extra rows:
1 0 1 2
0 1 1 1
0 1 1 2

Requirements: 2 x 2 x 2 x 4, no interactions.

Full design: 32 points. Balanced fraction: 8 points.

Design size wanted: 8 points.. At least 7 needed.

Generators: 1 0 1 1 ; 0 1 1 0

Balanced design:
0 0 0 0
1 1 0 1
0 1 1 2
1 1 0 3
0 1 1 0
1 0 1 1
0 0 0 2
1 0 1 3

Requirements: 2 x 2 x 3 x 3, no interactions.

Full design: 36 points. Balanced fraction: 36 points.

Design size wanted: 12 points. At least 7 needed.

Generators: 1 0 0 0 ; 0 1 0 2 ; 0 0 1 1

Basic design:
0 0 0 0
1 0 1 1
1 1 0 2
0 1 1 0
1 0 2 0
0 0 2 1
0 0 1 2

Extra rows:
0 1 2 2
0 1 0 1
1 1 1 0
1 0 0 2
1 1 2 1

Requirements: 2 x 2 x 2 x 4 x 4, no interactions.

Full design: 64 points. Balanced fraction: 16 points.

Design size wanted: 12 points. At least 10 needed.

Generators: 1 0 1 1 0 ; 0 1 1 0 1

Basic design:
0 0 0 0 0
1 1 0 1 1
0 1 1 2 1
1 0 1 3 2
1 0 1 1 2
0 1 1 0 3
1 0 1 3 0
1 1 0 1 1
0 0 0 2 2
1 0 1 1 0

Extra rows:
1 1 0 3 3
0 1 1 2 3

Requirements: 2 x 3 x 6, no interactions.

Full design: 36 points. Balanced fraction: 12 points.

Design size wanted: 12 points. At least 9 needed.

Generators: 1 0 3 ; 0 2 5

Balanced design:
0 0 0
1 2 2
0 2 5
1 1 1
0 1 4
1 0 0
0 0 3
1 2 5
0 2 2
1 1 4
0 1 1
1 0 3

Requirements: 2 x 2 x 2 x 3 x 3, CD interaction (that is, the interaction between the third and fourth factors).

Full design: 72 points. Balanced fraction: 72 points.

Design size wanted: 12 points. At least 10 needed.

Generators: 0 0 1 0 0 ; 1 0 0 0 0 ; 0 1 0 0 0 ; 0 0 0 1 0 ; 0 0 0 0 1

Basic design:
0 0 0 0 0
0 1 1 2 2
1 1 0 0 1
1 0 1 1 0
0 1 0 1 0
1 1 1 2 1
0 0 0 2 1
0 1 1 0 2
1 1 0 1 1

Extra rows:
1 0 0 1 2
1 1 0 2 2
1 0 1 0 1

Requirements: 2 x 2 x 4, AB interaction.

Full design: 16 points. Balanced fraction: 16 points.

Design size wanted: 8 points. At least 7 needed.

```
Generators:    1 0 0 ;  0 1 0 ;  0 0 3

Basic design:    0 0 0
                 1 0 0
                 1 1 0
                 0 0 1
                 0 0 2
                 0 1 0
                 0 1 3

Extra rows:      0 0 3
```

Requirements: 2 x 2 x 2 x 3 x 3 x 3 x 4 x 4, no interaction.

Full design: 3456 points. Balanced fraction: 144 points.

Design size wanted: 24 point. At least 16 needed.

```
Generators:    1 0 1 0 0 0 1 0 ;  0 1 1 0 0 0 0 1 ;
               0 0 0 1 0 1 0 0 ;  0 0 0 0 1 1 0 0

Basic design:    0 0 0 0 0 0 0 0
                 1 1 0 1 1 2 1 1
                 0 0 0 2 2 1 0 2
                 0 1 1 2 0 2 0 3
                 0 1 1 1 2 0 2 3
                 0 1 1 2 1 0 0 1
                 0 1 1 2 1 0 2 1
                 1 0 1 0 1 1 1 2
                 1 0 1 2 1 0 3 0
                 1 1 0 1 0 1 3 3
                 1 1 0 0 2 2 3 1
                 0 0 0 2 2 1 2 0
                 1 1 0 1 0 1 1 1
                 0 0 0 1 1 2 2 2
                 1 1 0 2 1 0 1 3
                 1 0 1 1 2 0 1 0

Extra rows:      0 1 1 1 0 1 2 1
                 1 0 1 0 0 0 3 2
                 0 1 1 0 1 1 2 3
                 1 0 1 2 0 2 1 0
                 0 0 0 1 1 2 0 0
                 1 1 0 0 0 0 1 1
                 0 1 1 0 2 2 0 3
                 0 0 0 2 0 2 2 2
```

This example illustrates the power of the combined procedures. Since the full design has 3456 points, the rapid reduction to a balanced fraction of only 144 points by using the generators (algorithm MULFAC) before switching to the further reduction to 24 points using algorithm REDDES, effects a substantial saving in computing time.

The following examples illustrate the changes made by varying interaction requirements.

Requirements: 2 x 3 x 3, AC interaction.

Full design: 18 points. Balanced fraction: 18 points.

Design size wanted: 12 points. At least 8 needed.

Generators: 1 0 0 ; 0 0 2 ; 0 2 0

Basic design:
0 0 0
1 2 0
1 0 0
0 1 0
0 0 1
1 0 2
0 2 2
1 1 0

Extra rows:
1 2 1
0 1 1
1 1 2
0 1 2

Requirements: 2 x 3 x 3, BC interaction.

Full design: 18 points. Balanced fraction: 18 points.

Design size wanted: 12 points. At least 10 needed.

Generators: 0 2 0 ; 0 0 2 ; 1 0 0

Basic design:
0 0 0
1 0 0
1 1 2
0 2 1
0 1 2
1 2 0
1 1 0
1 0 1
0 0 2
1 0 2

Extra rows:
1 2 2
1 1 1

Requirements: 2 x 3 x 3, AC and BC interactions.

Full design: 18 points. Balanced fraction: 18 points.

Design size wanted: 12 points. At least 12 needed.

Generators: 0 0 2 ; 1 0 0 ; 0 2 0

Balanced design:
0 0 0
0 1 2
1 1 1
1 2 2
0 0 1
0 2 2
0 2 1
0 2 0
0 1 0
1 0 1
0 0 2
1 0 2

CHAPTER EIGHT

CONCLUSIONS

1 Work done

In chapter one I briefly outlined the history of experimental design. Although this could not be complete because so much literature exists on the subject, it led to the objective of the study: to develop a methodology, represented as a set of programmable algorithms, for the design of experiments of the types that are generally likely to be useful in the physical sciences. The chapter concluded with a diversion into the relatively new subject of designing algorithms. This was needed to set the scene for subsequent chapters in which the design of algorithms was a major theme.

In chapter two I discussed in more detail my choice of types of experimental design for the study. Here, I am confident of the choice because it is based on eleven years' experience of experimental design and analysis in industrial research. The experimental designs chosen are intended to lead to the fitting of linear and quadratic models with quantitative variables, and of factorial models with qualitative variables. However, in both cases, fractional two-level factorial designs form a base on which the more complex designs may be built.

Chapter three was accordingly devoted to the development of algorithms for designing fractional two-level factorials. An important feature of these fractional designs is that in the physical situations to which they are applied, it is usual to assume that some first order interactions and most higher order interactions are negligible. The usual design procedure is to design a fractional experiment and then to check, by way of the aliasing matrix, if all the required main effects and interactions are aliased only with interactions assumed negligible. If they are found to be aliased with each other, then the experiment is redesigned and re-checked. My simple but practically important contribution has been a new algorithm which leads directly from the requirements set to a set of defining contrasts and hence to a fractional design in which none of the required effects is aliased with any other.

In chapter four I developed algorithms for augmenting a fractional two level factorial experiment with points that would enable the fitting of quadratic terms as well as linear and interaction terms with quantitative variables. Although this was based on standard theory leading to estimators of quadratic terms orthogonal to estimators of linear and interactive terms, I modified the usual procedure which imposes the condition of including quadratic terms for all factors. This enables the experimenter to define which factors have quadratic effects and which do not, using prior knowledge of the physical situation.

A note on the analysis of these designs was introduced in chapter five, followed by an example. This example was chosen to illustrate in detail the usefulness of methodology developed so far and applied successfully to the computer simulation of a physical experiment.

Fractional designs of asymmetric factorial experiments have long been a problem. The literature abounds with studies of factorials in which the numbers of levels of factors are prime. Indeed the subject seems to have become a purely theoretical branch of combinatorial mathematics: the needs of practical experimentation have largely been ignored. In chapter six, therefore, I provided the theory and the consequent algorithms for constructing balanced fractional asymmetric factorial designs using group generators. These were not always successful in producing balanced fractions that were as small as desirable from an economic viewpoint. The well-known criterion of minimising the determinant of the inverted cross-products matrix was used in chapter seven for developing algorithms for further reduction in the size of a fractional experimental design. There were several important innovations here. One was a procedure for ensuring that the smallest basic design, to which other points would be added according to the determinant criterion, would be the best in terms of that criterion. Another valuable contribution was the algorithm for automatically coding as dummy variables the contrasts representing main effects and interactions. Since this was developed as a step in constructing the cross-products matrix which is also needed in analysing experimental data, the algorithm should be useful to anybody writing an analysis program. A new joint proof of two known theorems in matrix algebra was presented. Chapter seven concluded with a set of examples of experiments designed by the linked methods of chapters six and seven.

Finally, the algorithms developed in this study have been fully implemented using standard Fortran with a few specified exceptions. These programs are listed in three appendices.

I understand from correspondence that the programs listed in appendix one, implementing the algorithms of chapters three and four, have been implemented successfully at: the United States Army Logistics Center in Virginia; the Department of Metallurgy in The University of Newcastle, New South Wales; the research laboratories of Comalco Aluminium (Bell Bay)Limited in Tasmania; and at the Union Carbide Corporation in West Virginia.

They have also been used intensively at the Sheffield laboratories and at the Teesside laboratories of the British Steel Corporation.

2. Further work

Despite careful checking, I fully expect that in such a complex set of algorithms and their program implementations there must be some mistakes. There must also be room for improved efficiency. Further work must therefore include corrections of mistakes and improvements in programming efficiency of work done so far.

The question of efficiency should also be studied in relation to the experimental designs themselves. We may, for example, define the efficiency of a design as the determinant of the information matrix divided by the number of observations. With such a definition we could compare designs with the same requirements and answer questions like: When we lose orthogonality (for example, by using rough approximations to α in algorithm AUGFAC, or by making unbalanced experiments as with algorithm REDDES) do we lose so much in efficiency as to be important?

A natural extension of the work done will be the development of algorithms for designing mixed experiments: those with both qualitative and quantitative variables: allocating two levels to those with linear effects and three levels to those with quadratic effects. However, a general technique calls for more thought and development than I have been able to give.

Other extensions include consideration of experimental constraints, as suggested in chapter one, figure one; inclusion of prior data for consideration when applying the determinant criterion; dealing with multi-variate situations when more than one dependent variable is present (one approach was suggested in the example at the end of chapter four); sequential design and analysis; and experimental simulation using probabilistic models.

Another major problem to be tackled is that of the psychology and language needed to develop reliable conversations between the computer and the experimental research worker who does not have the benefit of a statistician to guide him. This was discussed briefly at the end of section two, chapter one. It is a problem of such magnitude that deserves a complete research study.

3 Acknowledgements

I am grateful to Dr A Huitson and Dr W G Gilchrist of Sheffield City Polytechnic and to Mr S E Siday of the British Steel Corporation for their guidance during this research program.

The Automatic Design of Experiments

Some Practical Algorithms

REFERENCES

Addelman S. (1963a) Techniques for constructing fractional replicate plans. JASA, March 45-71.

Addelman S. (1963b) Techniques for constructing fractional replicate plans in factorial experiments. Biometrika, 34, 255-272.

Barnard M. M. (1936) An enumeration of the confounded arrangements in the 2^n factorial designs. JRSS supplement, 3, 195-202.

Bose R. C. (1939) On the application of Galois fields to the problem of the construction of hyper-graeco-latin squares. Sankhya, 3, 323-338.

Box G. E. P. and Behnken D. W. (1960) Some new three level designs for the study of quantitative variables. Technometrics, 2, 4, 455-476.

Box G. E. P. and Hunter J. S. (1961) The 2^{k-p} fractional factorial designs. Part 1: Technometrics, 3, 3, 311-351. Part 2: Technometrics, 3, 4, 449-458.

Box G. E. P. and Wilson K. B. (1951) On the experimental attainment of optimum conditions. JRSS(B), 13, 1, 1-45.

Box M. J. and Draper N. R. (1971) Factorial designs, the det(X'X) criterion and some related matters. Technometrics, 13, 4, 731-742.

Brownlee K. A. (1953) Industrial experimentation. (Chemical Publishing Co., N.Y.).

Cochran W. G. and Cox G. M. (1950) Experimental designs. (Wiley).

Davies O. L. (1954) Design and analysis of industrial experiments. (Oliver and Boyd).

Dempster A. P. (1969) Elements of continuous multivariate analysis. (Addison-Wesley)

Dijkstra E. W. (1973) Notes on structured programming. (Academic Press).

Draper N. R. and Mitchell T. J. (1967) The construction of saturated 2_R^{k-p} designs. Ann. Math. Stat., 38, 1110-1126.

Duckworth W. E. (1968) Statistical techniques in technological research. (Methuen).

Ehrenfeld S. (1953) On the efficiency of experimental designs. Ann. Math. Stat., 26, 247-255.

Fedorov V. V. (1972) Theory of optimal experiments. (Academic Press).

Finney D. J. (1945) The fractional replication of factorial experiments. Ann. Eugen., 12, 4, 291-301.

Finney D. J. (1946) Recent developments in the design of field experiments, part 3: fractional replication. Jnl. Agr. Sci., 36, 184-191.

Finney D. J. (1948) Main effects and interactions. JASA, 566-571.

Fisher R. A. (1925) Statistical methods for research workers. (Oliver and Boyd).

Fisher R. A. (1926) The arrangement of field experiments. Jnl. Ministry of Agriculture, 33, 503-513.

Fisher R. A. (1935) The design of experiments. (Oliver and Boyd).

Fisher R. A. (1943) The theory of confounding in factorial experiments in relation to the theory of groups. Ann. Eugen., 11, 4, 341-353.

Fisher R. A. (1945) A system of confounding for factors with more than two alternatives, giving completely orthogonal cubes and high powers. Ann. Eugen., 12, 4, 283-290.

FORTRAN 4 manuals for the IBM 1130 and 1800 computers and the General Automation SPC 16 computer: all editions published between about 1965 and 1978.

Franklin M. F. (1977) Private communication, March 17, from the ARC Unit, Edinburgh.

Franklin M. F. and Bailey R. A. (1977). Selection of defining contrasts and confounded effects in two-level experiments. JRSS(C), Applied Statistics, 26, 3, 321-326.

Gauss C. F. (1809) Theoria Motus Corporum Coelestium, book 2, section 3, articles 174 to 189.

Gauss C. F. (1821) Theoria Combinationis Observationum Erroribus Minimis Obnoxiae.

Goldsmith P. L. (1974) Computer-aided design of experiments using cells from a pre-specified repertoire. A report of the ICI conversational computing service.

Goodman S. E. and Hedetniemi S. T. (1977) Introduction to the design and analysis of algorithms. (McGraw Hill)

Greenfield A. A. (1972) Automatic design of experiments with qualitative variables. British Steel Corporation open report MG/44/72.

Greenfield A. A. (1974) Design of balanced experiments with continuous variables - mathematical and programming aspects. British Steel Corporation open report MG/57/71 revised.

Greenfield A. A. (1976) Selection of defining contrasts in two-level experiments. JRSS(C), Applied Statistics, 25, 1, 64-67.

Greenfield A. A. (1978) Selection of defining contrasts in two-level experiments - a modification. JRSS(C), Applied Statistics, 27, 1, p78.

Herzberg A. M. and Cox D. R. (1969) Recent work on the design of experiments. JRSS(A), 132, 29-67.

John J. A. and Dean A. M. (1975) Single replicate factorial experiments in generalised cyclic designs.
JRSS(B), 37, 1,
Part 1: Symmetrical arrangements. 63-71.
Part 2: Asymmetrical arrangements. 72-76.

Kendall M. G. and Stuart A. (1966) The advanced theory of statistics. Volume 3, 158-160. (Griffin).

Kempthorne O. (1947) A simple approach to confounding and fractional replication in factorial experiments. Biometrika, 34 255-272.

Kempthorne O. (1952) Design and analysis of experiments. (Wiley).

Kieffer J. and Wolfovitz J. (1959) Optimum designs in regression problems. Ann. Math. Stat., 30, 271-295.

Kieffer J. (1959a) On the non-randomised optimality and randomised non-optimality of symmetrical designs. Ann. Math. Stat., 29, 675-699.

Kieffer J. (1959b) Optimum experimental designs. JRSS(B), 21, 2, 272-319.

Kishen K. and Srivastava J. N. (1959) Mathematical theory of confounding in asymmetrical and symmetrical factorial designs. Jnl. Indian Soc. Agr. Stat., 11, 73-110.

Knuth D. E. (1973) Fundamental algorithms. Volume one: the art of computing. (Adison Wesley).

Mendenhall W. (1969) The design and analysis of experiments. (Wordsworth).

Pavlossoglou J. and Clay W. (1975) Mathematical modelling for the razor strip (13% Cr, low carbon steel) carburising processes. British Steel Corporation open report CDL/MT/105/75.

Pavlossoglou J. (1975) Computer simulation of the razor strip two stage carburising processes. British Steel Corporation open report CDL/MT/108/75.

Pavlossoglou J. and Greenfield A. A. (1975) Optimisation of the razor strip two stage processes. British Steel Corporation open report CDL/MT/113/75.

Plackett R. L. (1946) Some generalisations in the multifactorial design. Biometrika, 33, 328-332.

Smith K. (1918) On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations. Biometrika, 12, 1, 1-85.

Tocher K. D. (1952a) The design and analysis of block experiments. JRSS(B) 45-100.

Tocher K. D. (1952b) The design and analysis of experiments. Ph.D. thesis, London.

Tocher K. D. (1952c) A note on the design problem. Biometrika, 39, p189.

Wald A. (1943) On the efficient design of statistical investigations. Ann. Math. Stat., 14, 134-140.

White D. and Hultquist R. A. (1965) Construction of confounding plans for mixed factorial designs. Ann. Math. Stat., 36, 1256-1271.

Whitwell J. C. and Morbey G. K. (1961) Reduced designs of resolution five. Technometrics, 3, 459-477.

Worthley R. and Banerjee K. S. (1974) A general approach to confounding plans in mixed factorial experiments when the number of levels of a factor is any positive integer. Ann. Stat., 2, 3, 579-585.

Wynn H. P. (1970) The sequential generation of D-optimum experimental designs. Ann. Math. Stat., 41, 1655-1664.

Wynn H. P. (1972) Results in the theory and construction of D-optimum experimental designs. JRSS(B), 34, 2, 133-147.

Yates F. (1933) The principles of orthogonality and confounding in replicated experiments. Jnl Ag. Sci., 23, 108-145.

Yates F. (1935) Complex experiments. Suppl. JRSS, 2, 181-223

Yates F. (1937) The design and analysis of factorial experiments. Tech. Comm. Bur. Soil. Sci., Harpenden, number 35.

The Automatic Design of Experiments

Some Practical Algorithms

GLOSSARY

Abelian group	A group G with binary operation @ is abelian if, for all $g,h \in G$, $g@h = h@g$.
Algorithm	A sequence of rules for solving a problem.
Alias	Two effects are said to be aliased with one another in an experiment if they cannot be distinguished from each other.
Aliasing matrix	The aliasing matrix of an experiment is an array of all effects and interactions that could be estimated from the experimental results, such that all the effects and interactions on any row of the array are aliased with one another.
Asymmetric factorial	A factorial experiment in which at least two of the factors have unequal numbers of levels.
Balanced experiment	An experiment in which, for every factor, the numbers of observations at every level of the factor are equal.
Confounding	An interaction is said to be confounded when it is aliased with the mean effect and is used as a basis for dividing or taking a fraction of an experiment.
Contrast	The difference between the mean values observed at two different levels of a factor.
Control variable	A variable whose values can be specified by the experimenter. (see Independent variable).
Co-prime	Two integers are co-prime if they have no factors in common other than 1.
Coset	If H is a subgroup in a group G with binary operation @ , then a coset of H is a set of elements $\{x$ such that $x = h@g$ for all $h \in H$ and some $g \in G\}$.

Cross products matrix	If $\underset{\sim}{X}$ is a matrix of values of independent variables in an experiment, then the product $\underset{\sim}{X}'\underset{\sim}{X}$ is the cross-products matrix, where $\underset{\sim}{X}'$ is the transpose of $\underset{\sim}{X}$.
Cyclic group	A group G with binary operation @ is cyclic if it can be generated completely by some element g.
Defining contrast	The interactions confounded in any system of confounding are the defining contrasts.
Dependent variable	A variable whose observed value depends on the values of the control (or independent) variables.
Design matrix	The matrix of values of independent variables, including specified functions of the independent variables, in an experiment.
Design point	A single combination of specified values of independent variables at which the dependent variable(s) will be observed. A row of the observation matrix.
Discrete step	A distinct interval.
Effect	A contrast. The difference between the mean values observed at two different levels of a factor. Equivalently, the coefficient of a term in a linear model.
Experimental design	A specified set of conditions for an experiment.
Experimental design point	see design point
Factor	An independent variable. Usually a qualitative variable, but may refer to a quantitative variable.

Factorial experiment	An experiment in which observations are made at different combinations of levels, or values, of factors.
Fractional replication	The selection of only a fraction of all possible combinations of levels of factors.
Fractional two-level factorial	A fractionally replicated two-level factorial experiment
Group	A group is a non-empty set S/with a binary operation (@, say) such that there is an identity element in S, every element in S has an inverse in S, and the operation is associative.
Group generator	Some element in a group G that can generate the group, either on its own (see cyclic group) or together with other generators.
Independent variable	A variable whose values can be specified by the experimenter; or a variable on whose value the value of the response or dependent variable depends.
Information matrix	see cross products matrix
Interaction	The difference in effect of a control variable on a response variable at difference levels of another control variable. Higher order interactions are defined as the differences of the next lower order interactions at different levels of another control variable. If a dependent variable is represented as a function of independent variables, then an interaction is the partial derivative of the function with respect to two or more of the independent variables.

Level	A distinct value of an independent variable or factor. If the variable is qualitative such that an ordinal value cannot be ascribed to it, then the level is only a nominal label.
Multi-level factorial	A factorial experiment in which each factor has more than two levels.
Observation matrix	A matrix $\underset{\sim}{X}$ in which each row represents a vector of values of the independent variables at which dependent, or response, variable is to be observed.
Observation point	A point in the space of independent variables defined by a row in the observation matrix.
Observed value	The value of the dependent, or response, variable observed at an observation point.
Optimisation	The determination of conditions at which some criterion is at its best.
Optimal design	An experimental design at which some criterion is at its best.
Orthogonal	Two unequal vectors are orthogonal if their inner product is equal to zero. An experimental design is orthogonal if the cross products matrix is a diagonal matrix.
Prime	An integer is prime if it is divisible by no integer other than itself and 1.
Quadratic design	An experimental design in quantitative variables which will allow for the estimation of coeeficients of quadratic terms.

Qualitative variable	A variable, or factor, whose levels are nominal qualities or categories.
Quantitative variable	A variable whose values can be designated by reference to some measure.
Relatively prime	see Co-prime .
Requirements	The effects and interactions that are required to be estimated from the observations to be made in an experiment. Also: experimental requirements, and requirements set.
Response	The observed value of a dependent variable at an observation point.
Subgroup	A subset of a group which is also a group with the same operator.
Two-level factorial	A factorial experiment in which each factor is at exactly two levels.
Standard order	The order in which effects or treatment combinations in two level factorials may be stated, in alphabetic notation, introducing one letter at a time in alphabetic order.

The Automatic Design of Experiments

Some Practical Algorithms

APPENDICES

The Automatic Design of Experiments

Some Practical Algorithms

APPENDIX ONE

The programs listed in this appendix represent a full implementation of the algorithms developed in chapters three and four. They have been written in standard Fortran 4 with the following exceptions and extra functions:

The logical operators .AND., .OR., .XOR. have been used with integer operands to give integer results, as described in chapter three, section two.

The function ITEST(I,J) returns a value of 1 if the J'th bit from the right in the integer I is set to 1, and a value of 0 if the J'th bit is set to 0.

The function IONBT(I,J) sets to 1 the J'th bit from the right in the integer I.

The subroutine FREMAT which is called from subroutine ENFAC is a system subroutine supplied by General Automation for their SPC-16 computers to allow data to be entered in free format by reference in READ statements to FORMAT(V). Alternative statements are available on most other computers for allowing data to be read in free format.

Users of these programs should carefully check that the dimension statements in all the programs are adequate for their problems.

Main Program DESIGN

```
      DIMENSION IV(128),IA(16),NB(16),MAJ(16),IN(32),KR(128)
      COMMON/DES/ N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32)
     1,LQ(16),R(16,3),NQ,NAMVAR(16,5),KEST
      DATA IA/'A ','B ','C ','D ','E ','F ','G ','H ','J ',
     1'K ','L ','M ','N ','P ','Q ','R '/
C     *  *  *  *
C
C     STEP 0  (INITIALISE)
C
C     *  *  *  *
      CALL ENFAC(IA)
      NE=1.+ALOG(FLOAT(NV))/ALOG(2.0)
      M=N-NE
      IF(M.GT.9)M=9
      NE=N-M
      NI=2**M
      NF=2**NE
      DO 1 I=1,N
      NB(I)=0
    1 MAJ(I)=0
      DO 3 I=1,NV
      DO 2 J=1,N
      IF(ITEST(MV(I),J).EQ.1)NB(J)=NB(J)+1
    2 CONTINUE
    3 CONTINUE
      DO 4 I=1,N
      MAX=0
      DO 5 J=1,N
      IF(NB(J).LE.MAX)GO TO 5
      MAX=NB(J)
      JM=J
    5 CONTINUE
      MAJ(I)=IONBT(MAJ(I),JM)
      NB(JM)=0
    4 CONTINUE
C     *   *   *   *
C
C     STEP  10  (CONSTRUCT FIRST COLUMN OF ALIASING MATRIX AND
C                SET MARKERS)
C
C     *   *   *   *
  210 JAK=1
      K(1)=0
      MM=0
      DO 6 I=1,NV
    6 IN(I)=100
      DO 7 I=2,NF
      IF(NEW(I).NE.1)GO TO 8
      LL=1
      MM=MM+1
      K(I)=MAJ(MM)
      GO TO 11
    8 LL=LL+1
      K(I)=K(LL).XOR.MAJ(MM)
   11 DO 9 J=1,NV
      IF(K(I).NE.MV(J))GO TO 9
      IV(I)=1
      KR(I)=0
      IN(J)=0
      GO TO 7
    9 CONTINUE
      KR(I)=100
      IV(I)=-1
    7 CONTINUE
      IF(M.EQ.0)GO TO 1001
C     *   *   *   *
C
C     STEP  20  (WILL NEXT DEFINING CONTRAST BE A GENERATOR?
C                IF SO,  RESET ROW MARKERS)
C
C     *   *   *   *
   20 JAK=JAK+1
      IF(JAK.GT.NI)GO TO 1000
      IF(NEW(JAK).NE.1)GO TO 70
   30 LNEW=JAK
      LL=1
      DO 13 I=2,NF
      IF(KR(I).GE.LNEW)IV(I)=-1
   13 CONTINUE
C     *   *   *   *
C
C     STEP  40   (FIND FIRST AVAILABLE REQUIREMENT COUNTING FROM
C                 END OF SET.  IF NONE,  RETURN TO STEP 20)
C
C     *   *   *   *
      DO 14 I=1,NV
      J=NV-I+1
      IF(IN(J).GE.LNEW)GO TO 15
   14 CONTINUE
      NI=JAK-1
      GO TO 1000
   15 LE=MV(J)
C     *   *   *   *
C
C     STEP 50  (FIND FIRST AVAILABLE ROW AND CREATE TEST
C               DEFINING CONTRAST  (KEST))
C
C     *   *   *   *
   50 DO 16 I=2,NF
      J=NF-I+2
      IF(KR(J).GT.LNEW)GO TO 52
      GO TO 16
   52 LO=J
      GO TO 60
   16 CONTINUE
      IF(LNEW.GT.2)GO TO 17
      LO=-1
      GO TO 90
   17 LO=0
      GO TO 90
C     *   *   *   *
C
C     STEP  60  (MARK ROW AND CREATE TEST DEFINING CONTRAST)
C
C     *   *   *   *
   60 KR(LO)=LNEW
      KEST=LE.XOR.K(LO)
      GO TO 80
C     *   *   *   *
C
C     STEP  70  (USE GENERATOR TO CREATE DEFINING CONTRAST)
C
C     *   *   *   *
   70 LL=LL+1
      KEST=KK(LL).XOR.KK(LNEW)
C     *   *   *   *
C
C     STEP 80  (TEST NEW DEFINING CONTRAST FOR ALIASING AND
C                SET MARKERS)
C
C     *   *   *   *
   80 I=1
   81 I=I+1
      IF(I.GT.NF)GO TO 18
      KJAK=K(I).XOR.KEST
      J=0
   83 J=J+1
      IF(J.GT.NV)GO TO 81
      IF(MV(J).NE.KJAK)GO TO 83
      IF(IV(I).NE.-1)GO TO 120
      IV(I)=0
      IN(J)=LNEW
      KR(I)=LNEW
      GO TO 83
   18 KK(JAK)=KEST
      GO TO 20
```

```
C     *    *    *    *
C
C     STEP  90  (BACKTRACK TO PREVIOUS LNEW)
C
C     *    *    *    *
   90 IF(LO.EQ.0)GO TO 19
      NF=NF*2
      NI=NI/2
      GO TO 210
   19 JAK=(LNEW-1)/2+1
      DO 21 I=2,NF
      IF(KR(I).GE.LNEW)KR(I)=100
   21 CONTINUE
      GO TO 30
C     *    *    *    *
C
C     STEP  120  (SINCE KEST HAS FAILED, RESET REQUIREMENTS
C                 SET MARKERS)
C
C     *    *    *    *
  120 DO 22 J=1,NV
      IF(IN(J).GE.LNEW)IN(J)=100
   22 CONTINUE
      GO TO50
 1000 KK(1)=0
      CALL ALMAT(IA)
      CALL FRADE(IA)
 1001 CALL AUGFAC
      STOP
      END
      SUBROUTINE ENFAC(IA)
      DIMENSION IA(16)
      COMMON/DES/ N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32)
     1,LQ(16),R(16,3),NQ,NAMVAR(16,5),KEST
C     *   *   *   *
C
C     ENTER  EXPERIMENTAL  REQUIREMENTS
C
C     *   *   *   *
      CALL FREMAT
      WRITE(5,20)
   20 FORMAT('ENTER TITLE FOR THIS EXPERIMENT')
      READ(5,108)NAMEX
  108 FORMAT(10A2)
  602 WRITE(5,1)
    1 FORMAT('HOW MANY FACTORS ARE THERE')
      READ(5,100)N
      IF(N.GT.16)GO TO 600
  100 FORMAT(V)
      DO 2 NV=1,N
      MV(NV)=0
    2 MV(NV)=IONBT(MV(NV),NV)
   10 WRITE(5,3)
    3 FORMAT('ENTER REQUIRED INTERACTION')
    4 READ(5,60)(KK(I),I=1,16)
   60 FORMAT(16A1)
      J=NV+1
      IF(J.GT.32)GO TO 700
      MV(J)=0
      L=0
      DO 5 I=1,16
      DO 6 I1=1,16
      IF(KK(I).NE.IA(I1))GO TO 6
      MV(J)=IONBT(MV(J),I1)
      L=L+1
    6 CONTINUE
    5 CONTINUE
      IF(L.LE.0)GO TO 7
      NV=J
      GO TO 10
    7 NQ=0
      DO 13 I=1,N
      WRITE(5,12)I
   12 FORMAT('FOR VARIABLE ',I6,'  TYPE '/'VARIABLE NAME')
      READ(5,108)(NAMVAR(I,J),J=1,5)
      WRITE(5,50)
   50 FORMAT('L OR Q ')
      READ(5,60)LQ(I)
      WRITE(5,51)
   51 FORMAT('LEAST VALUE,GREATEST VALUE AND INTERVAL ')

      READ(5,100)(R(I,J),J=1,3)
      IF(LQ(I).EQ.'Q')NQ=NQ+1
   13 CONTINUE
       RETURN
  600 WRITE(5,601)
  601 FORMAT('NO MORE THAN 16 FACTORS ALLOWED TRY AGAIN')
      GO TO 602
  700 WRITE(5,701)
  701 FORMAT('NO MORE THAN 31 FACTORS+INTERACTIONS ALLOWED ,TRY AGAIN')
      GO TO 602
      END
      SUBROUTINE ALMAT(IA)
      DIMENSION IA(16)
      DIMENSION IB(8,16)
      COMMON/DES/ N,NF,KD(128),NI,K(128),NAMEX(10)
     1,KK(512),NE,NV,MV(32),LQ(16),R(16,3),NQ,NAMVAR(16,5),KEST
      DATA IBLANK/'  '/
   10 FORMAT (8(16A1))
C     *    *    *    *
C
C     PRINT  ALIASING  MATRIX
C
C     *    *    *    *
   11 FORMAT(/////)
      WRITE(5,1)NAMEX
    1 FORMAT('ALIASING MATRIX FOR ',10A2)
      NB=1+(NI-1)/8
      IF(NI.GT.8)GO TO 2
      NS=NI
      GO TO 3
    2 NS=8
    3 DO 8 I1=1,NB
      NT=(I1-1)*8
      DO 7 I2=1,NF
      DO 5 I3=1,NS
      NX=NT+I3
      J=K(I2).XOR.KK(NX)
      L=0
      DO 4 I4=1,16
      IF(ITEST(J,I4).NE.0)GO TO 4
      L=L+1
      IB(I3,L)=IBLANK
    4 CONTINUE
      DO 6 I4=1,16
      IF(ITEST(J,I4).EQ.0)GO TO 6
      L=L+1
      IB(I3,L)=IA(I4)
    6 CONTINUE
    5 CONTINUE
      WRITE(5,10)((IB(I3,L),L=1,16),I3=1,NS)
    7 CONTINUE
      WRITE(5,11)
    8 CONTINUE
      RETURN
      END
      SUBROUTINE FRADE(IA)
      DIMENSION IB(128,16),IA(16)
      COMMON/DES/ N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE
     1,NV,MV(32),LQ(16),R(16,3),NQ,NAMVAR(16,5),KEST
      DATA IBLANK/'  '/
C     *   *   *   *
C
C     STEP  0  (COPY FIRST COLUMN OF ALIASING MATRIX
C                TO DESIGN VECTOR)
C
C     *   *   *   *
      DO 1 I=1,NF
    1 KD(I)=K(I)
C     *   *   *   *
C
C     STEP  10  (FIND 'J' : FACTOR TO BE ADDED)
C
C     *   *   *   *
   10 JJ=KD(NF)
   15 NE=NE+1
      DO 2 I=1,N
      J=IONBT(0,I)
      JTEST=J.AND.JJ
      IF(JTEST.EQ.0)GO TO 20
    2 CONTINUE
C     *   *   *   *
C
C     STEP  20  (FIND 'JIP' : DEFINING CONTRAST WITH
C                 FACTOR TO BE ADDED)
C
C     *   *   *   *
   20 JJ=J.OR.JJ
      DO 22 I=2,NI
      JIP=KK(I)
      JTEST=J.AND.JIP
      IF(J.NE.JTEST)GO TO 22
      JTEST=JJ.AND.JIP
      IF(JIP.EQ.JTEST)GO TO 30
   22 CONTINUE
C     *   *   *   *
C
C     STEP  30  (COPY 'JIP',  REMOVING 'J')
C
C     *   *   *   *
   30 NT=J.XOR.JIP
```

```
C     *   *   *   *
C
C     STEP 40  (FIND NUMBER OF BITS IN COMMON BETWEEN
C                EACH DESIGN ELEMENT AND 'NT')
C
C     *   *   *   *
      DO 4 I=2,NF
      NB=NT.AND.KD(I)
      L=0
      DO 3 II=1,N
      IF(ITEST(NB,II).EQ.1)L=L+1
    3 CONTINUE
C     *   *   *   *
C
C     STEP  50  (ADD FACTOR TO DESIGN ELEMENT IF NUMBER
C                OF BITS (L)  IS ODD)
C
C     *   *   *   *
      IF(ITEST(L,1).EQ.1)KD(I)=KD(I).OR.J
    4 CONTINUE
      IF(NE.LT.N)GO TO 15
C     *   *   *   *
C
C     PRINT  DESIGN
C
C     *   *   *   *
      IB(1,16)='I '
      DO 5 I=1,15
    5 IB(1,I)=IBLANK
      DO 9 I1=2,NF
      J=KD(I1)
      L=0
      DO 7 I2=1,16
      IF(ITEST(J,I2).NE.0)GO TO 7
      L=L+1
      IB(I1,L)=IBLANK
    7 CONTINUE
      DO 8 I2=1,16
      IF(ITEST(J,I2).EQ.0)GO TO 8
      L=L+1
      IB(I1,L)=IA(I2)
    8 CONTINUE
    9 CONTINUE
      WRITE(5,110)NAMEX
  110 FORMAT('DESIGN FOR ',10A2)
   11 FORMAT(16A1)
      DO 12 I1=1,NF
   12 WRITE(5,11)(IB(I1,L),L=1,16)
      RETURN
      END
      SUBROUTINE AUGFAC
      DIMENSION X(16,5),Y(16),II(200),JJ(200)
      COMMON/DES/ N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32)
     1,LQ(16),R(16,3),NQ,NAMVAR(16,5),KEST
C     *   *   *   *
C
C     STEP  0  (INITIALISE AND PRINT COLUMN HEADINGS
C               FIND  ALPHA)
C
C     *   *   *   *
      WRITE(5,600)NAMEX
  600 FORMAT('EXPERIMENTAL DESIGN FOR ',10A2)
      WRITE(5,90)((NAMVAR(I,J),J=1,5),I=1,N)
   90 FORMAT('OBSERVATIONS ',3X,10(5A2,1X))
      IF(NQ.LT.1)GO TO 10
      NP=NV+NQ+1
      ND=NF+2*NQ
      M=ND-NP
      IF(M.LE.5)GO TO 1
      NO=1
      GO TO 2
    1 NO=6-M
    2 IF(NQ.GT.1)GO TO 3
      ALPHA=1
      GO TO 10
    3 Z=NF*(ND+NO)
      ALPHA=SQRT(0.5*(SQRT(Z)-NF))
C     *   *   *   *
C
C     STEP  10  (COMPUTE DESIGN INTERVALS)
C
C     *   *   *   *
   10 DO 4 I=1,N
      M=IFIX((R(I,2)-R(I,1))/R(I,3))
      M=M/2+ITEST(M,1)
      P=M*R(I,3)
      X(I,3)=R(I,1)+P
      IF(LQ(I).EQ.'Q')GO TO 13
      GO TO 14
   13 M=IFIX(P/(ALPHA*R(I,3))+0.5)
      Q=M*R(I,3)
      X(I,1)=R(I,1)
      X(I,2)=X(I,3)-Q
      X(I,4)=X(I,3)+Q
      X(I,5)=R(I,1)+2.0*P
      GO TO 4
   14 X(I,2)=R(I,1)
      X(I,4)=R(I,2)
    4 CONTINUE
```

```
C
C     STEP  20  (PRINT TWO-LEVEL FRACTIONAL PART
C                OF DESIGN)
C
C     *   *   *   *
      DO 7 I=1,NF
      DO 5 J=1,N
      L=2*(ITEST(KD(I),J)+1)
    5 Y(J)=X(J,L)
      WRITE(5,6)I,(Y(J),J=1,N)
    7 CONTINUE
C     *   *   *   *
C
C     STEP  30  (PRINT AUGMENTING PART OF DESIGN)
C
C     *   *   *   *
      JAK=NF
      IF(NQ.LT.1)GO TO 40
      DO 8 I=1,N
      IF(LQ(I).NE.'Q')GO TO 8
      DO 9 J=1,N
      IF(I.NE.J)Y(J)=X(J,3)
    9 CONTINUE
      JAK=JAK+1
      Y(I)=X(I,1)
      WRITE(5,200)JAK,(Y(J),J=1,N)
      JAK=JAK+1
      Y(I)=X(I,5)
      WRITE(5,200)JAK,(Y(J),J=1,N)
    8 CONTINUE
      DO 15 J=1,NO
      JAK=JAK+1
      WRITE(5,200)JAK,(X(I,3),I=1,N)
   15 CONTINUE
      IF(NO.GT.1)WRITE(5,11)NO
   11 FORMAT('NOTE THAT THERE ARE ',I6,' DESIGN CENTRE POINTS')
C     *   *   *   *
C
C     STEP  40  (RANDOMISE ORDER OF OBSERVATIONS)
C
C     *   *   *   *
   40 IX=1+2*(8191.AND.KEST)
      WRITE(5,601)
  601 FORMAT('RANDOMISED OBSERVATION ORDER')
  604 XX=JAK
      DO 12 I=1,JAK
   12 II(I)=0
      DO 17 J=1,JAK
      W=1./XX
  200 FORMAT(I6,6X,10(F10.4,1X))
```

```
201 FORMAT(10I6)
  6 FORMAT(I6,6X,10(F10.4,1X))
 18 DO 16 I=1,JAK
    IF(II(I).NE.0)GO TO 16
    CALL RANDU(IX,IX,YY)
    IF(YY.GT.W)GO TO 16
    XX=XX-1
    JJ(J)=I
    II(I)=1
    GO TO 17
 16 CONTINUE
    GO TO 18
 17 CONTINUE
    WRITE(5,201)(JJ(J),J=1,JAK)
    WRITE(5,602)
602 FORMAT('IS ANOTHER RANDOM NUMBER STREAM WANTED,TYPE YES OR NO')
    READ(5,603)IY
603 FORMAT(1A1)
    IF(IY.EQ.'Y')GO TO 604
    RETURN
    END
```

```
      FUNCTION NEW(I)
C     *    *    *    *
C
C     TEST IF "I" IS OF FORM 2**R + 1
C
C     *    *    *    *
      IF(I.EQ.2)GO TO 4
      NEW=0
      J=1
      I1=I-1
    3 J=J*2
      IF(J.LT.I1)GO TO 3
      IF(J.EQ.I1)NEW=1
      RETURN
    4 NEW=1
      RETURN
      END
```

```
      SUBROUTINE RANDU(IX,IY,YFL)
      IY=IX*899
      IF(IY)5,6,6
    5 IY=IY+32767+1
    6 YFL=IY
      YFL=YFL/32767.0
      RETURN
      END
```

(note: the value 32767 is used because this is the largest positive integer on a 16-bit word computer: $2^{15} - 1 = 32767$.

If these programs are implemented on a computer with a different word length, this integer must be altered accordingly).

The Automatic Design of Experiments

Some Practical Algorithms

APPENDIX TWO

The programs listed in this appendix represent a full implementation of the algorithms developed in chapter six. Algorithms DEFCON (renamed DEFGEN) and ENFAC, whose program implementations were listed in appendix one, have been modified to link with the multi-factorial algorithms as described in chapter six.

The programs have been written in standard Fortran 4 with the same exceptions and extra functions as were described in appendix one.

Again, users should carefully check that the dimension statements are adequate for their problems.

Main Program MULFAC

```
C MULFAC
      BIT BITS(3200)
      DIMENSION ID(8,16),IX(16,4),MVI(16),IA(16),IK(16),LF(16)
     1,IL(8),IDD(8,16),IB(8),JJ(200),II(16),NLL(16)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32),
     1NL(16),KEST
      DATA IA/'A ','B ','C ','D ','E ','F ','G ','H ','J ',
     1'K ','L ','M ','N ','P ','Q ','R '/
      CALL ENFAC(IA)
      DO 1 I=1,8
      DO 1 J=1,16
    1 ID(I,J)=0
      NG=0
      CALL FASET(IX,JG,LI)
C     *   *   *   *
C
C     FIND NEXT SUITABLE  FACTOR SUBSET
C
C     *   *   *   *
   20 IG=0
      MIN=100
   21 IG=IG+1
      IF((IX(IG,1).LT.MIN)
     1.AND.((IX(IG,2).GT.2).OR.
     1(IX(IG,3).NE.0)))GO TO 23
      GO TO 24
   23 MIN=IX(IG,1)
      IM=IG
   24 IF(IG.EQ.LI)GO TO 25
      GO TO 21
   25 IF(MIN.EQ.100)GO TO 100
      IF(IX(IM,2).GT.2)GO TO 27
      NGI=0
   26 NGI=NGI+1
      IB(NGI)=2**(NGI-1)
      IF(NGI.EQ.IX(IM,2))GO TO 66
      GO TO 26
C     *   *   *   *
C
C     INITIALISE FOR ENTRY TO DEFGEN
C
C     *   *   *   *
   27 NIG=IX(IM,2)+IX(IM,4)
      NVI=IX(IM,4)
   29 NVI=NVI+1
      MVI(NVI)=0
      MVI(NVI)=IONBT(MVI(NVI),NVI)
      IF(NVI.LT.NIG)GO TO 29
      MASK=0
```

```
C
C      FIND INTERACTIONS BETWEEN FACTORS IN CURRENT SUBSET
C
C      *    *    *    *
       I=0
   33  I=I+1
       IF(I.GT.N)GO TO 37
       IF(NL(I).EQ.IX(IM,1))MASK=IONBT(MASK,I)
       GO TO 33
   37  I=N
   38  I=I+1
       IF(I.GT.NV)GO TO 50
       ITT=MASK.AND.MV(I)
       IF(ITT.EQ.0)GO TO 38
       I1=0
       J=0
       NVI=NVI+1
       MVI(NVI)=0
   42  J=J+1
       IF(J.GT.N)GO TO 38
       IF(ITEST(MASK,J).NE.1)GO TO 47
       I1=I1+1
       IF(ITEST(MV(I),J).EQ.1)MVI(NVI)=IONBT(MVI(NVI),I1)
   47  GO TO 42
   50  CALL DEFGEN(NGI,IB,NIG,NVI,MVI)
       NOG=IX(IM,1)**NGI
C      *    *    *    *
C
C      IS SUBSET LINKED TO A PREVIOUS ONE?
C
C      *    *    *    *
       IF(IX(IM,3).EQ.0)GO TO 70
   66  J=IX(IM,4)
       J1=IX(J,3)-1
       LL=IX(J,4)
       GO TO 80
   70  I=0
       L=0
       LL=0
       J1=NG
   71  I=I+1
       IF(I.GT.LI)GO TO 80
       IF(I.EQ.IM)GO TO 71
       IF(IX(I,4).NE.0)GO TO 71
       IF(IX(I,1).NE.NOG)GO TO 71
       IX(I,3)=IM
       IX(IM,3)=NG+1
       JG=JG+1
       IX(IM,4)=NG+NGI
       IX(I,4)=1
```

```
C     *   *   *   *
C
C     CONVERT GENERATORS FROM BINARY  TO INTEGER FORM
C
C     *   *   *   *
   80 I=0
   81 I=I+1
      IF(I.LE.NGI)GO TO 85
      IX(IM,1)=100
      IF(J1.GT.NG)NG=J1
      GO TO 20
   85 J1=J1+1
      L=0
      J=0
   86 J=J+1
      IF(J.GT.N)GO TO 81
      IF(NL(J).NE.IX(IM,1))GO TO 86
      L=L+1
      ID(J1,J)=ITEST(IB(I),L)
      GO TO 86
C     *   *   *   *
C
C     HOW MANY FACTORS NOT YET IN GENERATORS
C
C     *   *   *   *
  100 L=0
      I=0
  101 I=I+1
      IF(I.GT.LI)GO TO 105
      IF(IX(I,1).NE.100)L=L+IX(I,2)
      GO TO 101
  105 IF(L.EQ.0)GO TO 200
      IF(L.GT.2)GO TO 120
      I=0
  108 I=I+1
      IF(I.GT.LI)GO TO 200
      IF(IX(I,4).NE.0)GO TO 108
      I1=IX(I,1)
      I2=0
      I3=0
  112 I2=I2+1
      IF(I2.GT.IX(I,2))GO TO 108
  114 I3=I3+1
      IF(I3.GT.N)GO TO 108
      IF(NL(I3).NE.I1)GO TO 114
      NG=NG+1
      ID(NG,I3)=1
      GO TO 112
```

```
C     INITIALISE FOR DEFGEN
C
C     *    *    *    *
  120 NIG=L
      NVI=0
  121 NVI=NVI+1
      MVI(NVI)=0
      MVI(NVI)=IONBT(MVI(NVI),NVI)
      IF(NVI.LT.NIG)GO TO 121
      MASK=0
      I=0
  126 I=I+1
      IF(I.GT.LI)GO TO 135
      IF(IX(I,4).NE.0)GO TO 126
      I1=IX(I,1)
      I2=0
  130 I2=I2+1
      IF(I2.GT.N)GO TO 126
      IF(NL(I2).NE.I1)GO TO 130
      MASK=IONBT(MASK,I2)
      GO TO 130
  135 I=N
  136 I=I+1
      IF(I.GT.NV)GO TO 150
      ITT=MASK.AND.MV(I)
      IF(ITT.EQ.0)GO TO 136
      I1=0
      J=0
      NVI=NVI+1
      MVI(NVI)=0
  140 J=J+1
      IF(J.GT.N)GO TO 136
      IF(ITEST(MASK,J).NE.1)GO TO 140
      I1=I1+1
      IF(ITEST(MV(I),J).EQ.1)MVI(NVI)=IONBT(MVI(NVI),I
      GO TO 140
  150 CALL DEFGEN(NGI,IB,NIG,NVI,MVI)
      I=0
      I2=0
  156 I=I+1
      IF(I.GT.N)GO TO 170
      I1=0
  159 I1=I1+1
      IF(I1.GT.LI)GO TO 156
      IF(IX(I1,4).NE.0)GO TO 159
      IF(NL(I).NE.IX(I1,1))GO TO 159
      I2=I2+1
      NLL(I2)=NL(I)
      LF(I2)=I
      GO TO 159
  170 CALL SELG(NIG,NGI,IB,NLL,LI,IX,IDD)
C     *    *    *    *
C
C     COPY GENERATORS FROM SELG INTO GENERATORS
C          FOR ALL FACTORS
C
C     *    *    *    *
      I=0
  181 I=I+1
      IF(I.GT.NGI)GO TO 200
      NG=NG+1
      J=0
  184 J=J+1
      IF(J.GT.NIG)GO TO 181
      I1=LF(J)
      ID(NG,I1)=IDD(I,J)
      GO TO 184
C     *    *    *    *
C
C     FIND FULL DESIGN  SIZE
C
C     *    *    *    *
  200 I=0
      WRITE(5,500)NAMEX
  500 FORMAT(///'    *    *    *    *    *'///'GENERATORS FOR   ',10A2
     1///)
      NO=1
  201 I=I+1
      IF(I.GT.NG)GO TO 208
      J=0
  204 J=J+1
      IF(J.GT.N)GO TO 207
      II(J)=JORD(ID(I,J),NL(J))
      GO TO 204
  207 IL(I)=LCM(II,N)
      WRITE(5,501)(ID(I,J),J=1,N)
  501 FORMAT(10X,16I2)
      NO=NO*IL(I)
      GO TO 201
  208 NFULL=1
      I=0
  209 I=I+1
      IF(I.GT.N)GO TO 2091
      NFULL=NFULL*NL(I)
      GO TO 209
 2091 IF(NO.LE.NFULL)GO TO 210
      NO=NFULL
      NG=N
      I=0
 2092 I=I+1
```

```
      IF(I.GT.N)GO TO 210
      J=0
      IL(I)=NL(I)
 2093 J=J+1
      IF(J.GT.N)GO TO 2092
      ID(I,J)=0
      IF(I.EQ.J)ID(I,J)=1
      GO TO 2093
  210 I=0
      WRITE(5,350)NAMEX
  350 FORMAT(///'   *    *    *    *    *'//'DESIGN FOR ',10A2
     1///)
  211 I=I+1
      IF(I.GT.NO)GO TO 1000
      CALL LEV(I,NG,IL,ID,IK)
      WRITE(5,301)(IK(LL),LL=1,N)
      GO TO 211
C     *    *    *    *
C
C     RANDOMISE ORDER OF OBSERVATIONS
C
C     *    *    *    *
 1000 IXX=1+2*(8191.AND.KEST)
      WRITE(5,601)
  601 FORMAT('RANDOMISED ORDER')
  604 XX=NO
      DO 312 I=1,NO
  312 BITS(I)=.FALSE.
      L=0
      JO=NO
      IF(NO.LE.200)GO TO 605
      JO=200
      JK=0
  605 L=L+1
      IF(NO.GT.L*200)GO TO 606
      JO=NO-(L-1)*200
      JK=1
  606 DO 317 J=1,JO
      W=1./XX
  318 DO 316 I=1,NO
      IF(BITS(I))GO TO 316
      CALL RANDU(IXX,IXX,YY)
  301 FORMAT(10I6)
      IF(YY.GT.W)GO TO 316
      XX=XX-1
      JJ(J)=I
      BITS(I)=.TRUE.
      GO TO 317
  316 CONTINUE
      GO TO 318
  317 CONTINUE
      WRITE(5,301)(JJ(J),J=1,JO)
      IF(JK.NE.1)GO TO 605
      WRITE(5,602)
  602 FORMAT('IS ANOTHER RANDOM NUMBER STREAM WANTED,TYPE YES
     1OR NO')
      READ(5,603)IY
  603 FORMAT(1A1)
      IF(IY.EQ.'Y')GO TO 604
      STOP
      END
      SUBROUTINE ENFAC(IA)
      DIMENSION IA(16)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32)
     1,NL(16)
C     *   *   *   *
C
C     ENTER EXPERIMENTAL REQUIREMENTS
C
C     *   *   *   *
      CALL FREMAT
      WRITE(5,20)
   20 FORMAT('ENTER TITLE FOR THIS EXPERIMENT')
      READ(5,108)NAMEX
  108 FORMAT(10A2)
  602 WRITE(5,1)
    1 FORMAT('HOW MANY FACTORS ARE THERE')
      READ(5,100)N
      IF(N.GT.16)GO TO 600
  100 FORMAT(V)
      DO 2 NV=1,N
      MV(NV)=0
    2 MV(NV)=IONBT(MV(NV),NV)
   10 WRITE(5,3)
    3 FORMAT('ENTER REQUIRED INTERACTION')
    4 READ(5,60)(KK(I),I=1,16)
   60 FORMAT(16A1)
      J=NV+1
      IF(J.GT.32)GO TO 700
      MV(J)=0
      L=0
      DO 5 I=1,16
      DO 6 I1=1,16
      IF(KK(I).NE.IA(I1))GO TO 6
      MV(J)=IONBT(MV(J),I1)
      L=L+1
    6 CONTINUE
    5 CONTINUE
      IF(L.LE.0)GO TO 7
      NV=J
      GO TO 10
    7 DO 13 I=1,N
      WRITE(5,12)I
   12 FORMAT('FOR FACTOR ',I6,' TYPE THE NUMBER OF LEVELS')
      READ(5,100)NL(I)
   13 CONTINUE
       RETURN
  600 WRITE(5,601)
  601 FORMAT('NO MORE THAN 16 FACTORS ALLOWED TRY AGAIN')
      GO TO 602
  700 WRITE(5,701)

  701 FORMAT('NO MORE THAN 31 FACTORS+INTERACTIONS ALLOWED ,TRY AGAIN')
      GO TO 602
      END
      SUBROUTINE FASET(IX,JG,LI)
      DIMENSION IX(16,4)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32),
     1NL(16)
C     *    *    *    *
C
C     DETERMINE FACTOR SUBSETS
C
C     *    *    *    *
      L=0
      LI=0
      IXI=0
      JG=0
    2 IXI=IXI+1
      J=0
    4 J=J+1
      IF(J.EQ.IXI)GO TO 7
      IF(NL(J).EQ.NL(IXI))GO TO 2
      GO TO 4
    7 LI=LI+1
      IX(LI,1)=NL(IXI)
      IX(LI,2)=1
      IX(LI,3)=0
      IX(LI,4)=0
      L=L+1
      IF(L.EQ.N)RETURN
    9 J=J+1
      IF(J.GT.N)GO TO 14
      IF(IX(LI,1).NE.NL(J))GO TO 9
      IX(LI,2)=IX(LI,2)+1
      L=L+1
      IF(L.NE.N)GO TO 9
      RETURN
   14 IF(IX(LI,2).GT.2)JG=JG+1
      GO TO 2
      END
      SUBROUTINE DEFGEN(NGI,IB,NIG,NVI,MVI)
      DIMENSION MVI(16)
      DIMENSION IB(8),NW(8)
      DIMENSION IV(128),NB(16),MAJ(16),IN(32),KR(128)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32
     1,NL(16),KEST
      DATA NW/2,3,5,9,17,33,65,129/
C     *   *   *   *
C
C     STEP 0 (INITIALISE)
C
C     *   *   *   *
      NE=1.+ALOG(FLOAT(NVI))/ALOG(2.0)
      M=NIG-NE
      IF(M.GT.9)M=9
      NE=NIG-M
      NI=2**M
      NF=2**NE
      DO 1 I=1,NIG
      NB(I)=0
    1 MAJ(I)=0
      DO 3 I=1,NVI
      DO 2 J=1,NIG
      IF(ITEST(MVI(I),J).EQ.1)NB(J)=NB(J)+1
    2 CONTINUE
    3 CONTINUE
      DO 4 I=1,NIG
      MAX=0
      DO 5 J=1,NIG
      IF(NB(J).LE.MAX)GO TO 5
      MAX=NB(J)
      JM=J
    5 CONTINUE
      MAJ(I)=IONBT(MAJ(I),JM)
      NB(JM)=0
    4 CONTINUE
C     *   *   *   *
C
C     STEP 10 (CONSTRUCT FIRST COLUMN OF ALIASING MATRIX AND
C              SET MARKERS)
C
C     *   *   *   *
  210 JAK=1
      K(1)=0
      MM=0
      DO 6 I=1,NVI
    6 IN(I)=100
      DO 7 I=2,NF
      IF(NEW(I).NE.1)GO TO 8
      LL=1
      MM=MM+1
      K(I)=MAJ(MM)
      GO TO 11
    8 LL=LL+1
      K(I)=K(LL).XOR.MAJ(MM)
   11 DO 9 J=1,NVI
      IF(K(I).NE.MVI(J))GO TO 9
      IV(I)=1
      KR(I)=0
      IN(J)=0
      GO TO 7
    9 CONTINUE
      KR(I)=100
      IV(I)=-1
    7 CONTINUE
      IF(M.GT.0)GO TO 20
      DO 12 I=2,NF
   12 KD(I)=K(I)
      GO TO 1001
C     *   *   *   *
C
C     STEP 20 (WILL NEXT DEFINING CONTRAST BE A GENERATOR?
C              IF SO, RESET ROW MARKERS)
C
C     *   *   *   *
   20 JAK=JAK+1
      IF(JAK.GT.NI)GO TO 1000
      IF(NEW(JAK).NE.1)GO TO 70
   30 LNEW=JAK
      LL=1
      DO 13 I=2,NF
      IF(KR(I).GE.LNEW)IV(I)=-1
   13 CONTINUE
C     *   *   *   *
C
C     STEP 40 (FIND FIRST AVAILABLE REQUIREMENT COUNTING FROM
C              END OF SET. IF NONE, RETURN TO STEP 20)
C
C     *   *   *   *
      DO 14 I=1,NVI
      J=NVI-I+1
      IF(IN(J).GE.LNEW)GO TO 15
   14 CONTINUE
      NI=JAK-1
      GO TO 1000
   15 LE=MVI(J)
```

```
C      *    *    *    *
C
C      STEP 50 (FIND FIRST AVAILABLE ROW AND CREATE TEST
C                 DEFINING CONTRAST (KEST))
C
C      *    *    *    *
   50 DO 16 I=2,NF
      J=NF-I+2
      IF(KR(J).GT.LNEW)GO TO 52
      GO TO 16
   52 LO=J
      GO TO 60
   16 CONTINUE
      IF(LNEW.GT.2)GO TO 17
      LO=-1
      GO TO 90
   17 LO=0
      GO TO 90
C      *    *    *    *
C
C      STEP 70 (USE GENERATOR TO CREATE DEFINING CONTRAST)
C
C      *    *    *    *
   60 KR(LO)=LNEW
      KEST=LE.XOR.K(LO)
      GO TO 80
   70 LL=LL+1
      KEST=KK(LL).XOR.KK(LNEW)
C      *    *    *    *
C
C      STEP 80 (TEST NEW DEFINING CONTRAST FOR ALIASING AND
C                SET MARKERS)
C
C      *    *    *    *
   80 I=1
   81 I=I+1
      IF(I.GT.NF)GO TO 18
      KT=K(I).XOR.KEST
      J=0
   83 J=J+1
      IF(J.GT.NVI)GO TO 81
      IF(MVI(J).NE.KT)GO TO 83
      IF(IV(I).NE.-1)GO TO 120
      IV(I)=0
      IN(J)=LNEW
      KR(I)=LNEW
      GO TO 83
   18 KK(JAK)=KEST
      GO TO 20
```

```
C       *    *    *    *
C
C       STEP 90 (BACKTRACK TO PREVIOUS LNEW)
C
C       *    *    *    *


   90 IF(LO.EQ.0)GO TO 19
      NF=NF*2
      NI=NI/2
      GO TO 210
   19 JAK=(LNEW-1)/2+1
      DO 21 I=2,NF
      IF(KR(I).GE.LNEW)KR(I)=100
   21 CONTINUE
      GO TO 30
C       *    *    *    *
C
C       STEP 120 (SINCE KEST HAS FAILED, RESET REQUIREMENTS
C                 SET MARKERS)
C
C       *    *    *    *
  120 DO 22 J=1,NVI
      IF(IN(J).GE.LNEW)IN(J)=100
   22 CONTINUE
      GO TO50
 1000 CALL FRADE(NIG)
 1001 NGI=0
      J=0
  140 J=J+1
      JG=NW(J)
      IF(JG.GT.NF)RETURN
      NGI=NGI+1
      IB(NGI)=KD(JG)
      GO TO 140
      END
      SUBROUTINE FRADE(NIG)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE
C     *    *    *    *
C
C     STEP 0 (COPY FIRST COLUMN OF ALIASING MATRIX
C            TO DESIGN VECTOR)
C
C     *    *    *    *
      DO 1 I=1,NF
    1 KD(I)=K(I)
   10 JJ=KD(NF)
   15 NE=NE+1
      DO 2 I=1,NIG
      J=IONBT(0,I)
      JTEST=J.AND.JJ
      IF(JTEST.EQ.0)GO TO 20
    2 CONTINUE
C     *    *    *    *
C
C     STEP 20 (FIND 'JIP' : DEFINING CONTRAST WITH
C             FACTOR TO BE ADDED)
C
C     *    *    *    *
   20 JJ=J.OR.JJ
      DO 22 I=2,NI
      JIP=KK(I)
      JTEST=J.AND.JIP
      IF(J.NE.JTEST)GO TO 22
      JTEST=JJ.AND.JIP
      IF(JIP.EQ.JTEST)GO TO 30
   22 CONTINUE
C     *    *    *    *
C
C     STEP 30 (COPY 'JIP', REMOVING 'J')
C
C     *    *    *    *
   30 NT=J.XOR.JIP
C     *    *    *    *
C
C     STEP 40 (FIND NUMBER OF BITS IN COMMON BETWEEN
C              EACH DESIGN ELEMENT AND 'NT')
C
C     *    *    *    *
      DO 4 I=2,NF
      NB=NT.AND.KD(I)
      L=0
      DO 3 II=1,NIG
      IF(ITEST(NB,II).EQ.1)L=L+1
    3 CONTINUE
C     *    *    *    *
C
C     STEP 50 (ADD FACTOR TO DESIGN ELEMENT IF NUMBER
C              OF BITS (L) IS ODD)
C
C     *    *    *    *
      IF(ITEST(L,1).EQ.1)KD(I)=KD(I).OR.J
    4 CONTINUE
      IF(NE.LT.NIG)GO TO 15
      RETURN
      END
      SUBROUTINE SELG(NIG,NGI,IB,NLL,LI,IX,IDD)
      DIMENSION IB(8),IX(16,4),NLL(16),IDD(8,16),II(16),IL(8),IKK(16)
     1,IDT(8,16),IK(16)
C     *   *   *   *
C
C     INITIALISE
C
C     *   *   *   *
      NOMIN=1
      NB=1-NIG
      I=0
    2 I=I+1
      IF(I.GT.NIG)GO TO 6
      NOMIN=NOMIN*NLL(I)
      NB=NB+NLL(I)
      GO TO 2
    6 NO1=NOMIN-1
C     *   *   *   *
C
C     CONVERT PRIME GENERATORS FROM BINARY TO
C     INTEGER FORM : FIND ORDERS AND PRODUCTS
C
C     *   *   *   *
      J=0
      NO=1
   11 J=J+1
      IF(J.GT.NGI)GO TO 21
      I=0
   14 I=I+1
      IF(I.GT.NIG)GO TO 19
      IDD(J,I)=ITEST(IB(J),I)
      II(I)=JORD(IDD(J,I),NLL(I))
      GO TO 14
   19 IL(J)=LCM(II,NIG)
      NO=NO*IL(J)
      GO TO 11
   21 IF(NO.LT.NB.OR.NO.GT.NOMIN)GO TO 30
      M=0
      NOMIN=NO
      NO1=NO-1
C     *   *   *   *
C
C     CYCLE GENERATORS : COMPUTE ORDERS AND
C     PRODUCTS : TEST FOR IMPROVEMENT
C
C     *   *   *   *
   30 I=0
   31 I=I+1
      IF(I.GT.NO1)GO TO 70
      DO 33 J=1,NIG
      IKK(J)=0
   33 IK(J)=0
      NO=1
      KJ=I
      IJ=0
   35 IJ=IJ+1
      IF(IJ.GT.NGI)GO TO 47
      L=(KJ-1)/IL(IJ)
      MM=KJ-IL(IJ)*L
      KJ=L+1
      JK=0
   39 JK=JK+1
      IF(JK.GT.NIG)GO TO 46
      KKK=IDD(IJ,JK)*MM
      L=NLL(JK)
      IDT(IJ,JK)=KKK-L*(KKK/L)
      IT=IDT(IJ,JK)
      II(JK)=JORD(IT,L)
      IF(IT.EQ.0)GO TO 39
      IK(JK)=IK(JK)+IT
      IF(IHCF(IT,L).EQ.1)IKK(JK)=1
      GO TO 39
   46 ILT=LCM(II,NIG)
      NO=NO*ILT
      GO TO 35
   47 J=0
   48 J=J+1
      IF(J.GT.NIG)GO TO 52
      IF(IK(J).EQ.0)GO TO 31
      IF(IKK(J).EQ.0)GO TO 31
      GO TO 48
C     *    *    *    *
C
C     FOR THOSE PAIRS OF FACTORS WITH EQUAL NUMBERS
C     OF LEVELS, CHECK THERE IS AT LEAST ONE GENERATOR
C     IN WHICH THESE TWO FACTORS HAVE  DIFFERENT INTEGER
C     VALUES
C
C     *    *    *    *
   52 IXJ=0
   53 IXJ=IXJ+1
      IF(IXJ.GT.LI)GO TO 67
      IF(IX(IXJ,2).NE.2)GO TO 53
      I3=0
   57 I3=I3+1
      IF(NLL(I3).NE.IX(IXJ,1))GO TO 57
      I4=I3
   60 I4=I4+1
      IF(NLL(I4).NE.IX(IXJ,1))GO TO 60
      J3=0
   63 J3=J3+1
      IF(J3,GT.NGI)GO TO 31
      IF(IDT(J3,I3).NE.IDT(J3,I4))GO TO 53
      GO TO 63
C     *   *   *   *
C
C     ALL TESTS PASSED : IS IT AN IMPROVEMENT?
C
C     *   *   *   *
   67 IF(NO.LT.NB.OR.NO.GT.NOMIN)GO TO 31
      M=I
      NOMIN=NO
      GO TO 31
C     *   *   *   *
C
C     IF M=0 USE PRIME GENERATORS
C
C     *   *   *   *
   70 IF(M.GT.0)GO TO 75
      NO=NOMIN
      RETURN
C     *   *   *   *
C
C     USE M TO RECOMPUTE BEST SET OF GENERATORS
C
C     *   *   *   *
   75 I=0
   76 I=I+1
      IF(I.GT.NGI)RETURN
      L=(M-1)/IL(I)
      MM=M-IL(I)*L
      M=L+1
      J=0
   80 J=J+1
      IF(J.GT.NIG)GO TO 85
      KJ=IDD(I,J)*MM
      L=NLL(J)
      IDD(I,J)=KJ-L*(KJ/L)
      II(J)=JORD(IDD(I,J),L)
      GO TO 80
   85 IL(I)=LCM(II,NIG)
      GO TO 76
      END
      FUNCTION LCM(I,NN)
      DIMENSION I(16)
C     *    *    *    *
C
C     FIND LOWEST COMMON MULTIPLE OF SET I(.)
C     OF  NN  INTEGERS
C
C     *    *    *    *
      N=NN
      J=0
      LCM=0
      K=0
   10 K=K+1
      IF(K.GT.N)GO TO 20
      IF(I(K).LE.LCM)GO TO 10
      LCM=I(K)
      MM=K
      GO TO 10
   20 K=MM
   21 J=IONBT(J,K)
      M=LCM
      N=N-1
      IF(N.EQ.0)RETURN
      K=0
   25 K=K+1
      IF(K.GT.NN)RETURN
      IF(ITEST(J,K).EQ.1)GO TO 25
   28 IF((LCM/I(K))*I(K).EQ.LCM)GO TO 21
      LCM=LCM+M
      GO TO 28
      END
```

```
      FUNCTION MOD(J,K,L)
C     *    *    *    *
C
C     ADD  J  AND  K,   MODULO  M
C
C     *    *    *    *
      MOD=J+K
    1 IF(MOD.LT.L)RETURN
      MOD=MOD-L
      GO TO 1
      END
      SUBROUTINE LEV(I,NG,IL,ID,IK)
      DIMENSION IL(8),ID(8,16),IK(16)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV
     1,MV(32),NL(16)
C     *    *    *    *
C
C     DETERMINE ALL FACTOR LEVELS FOR I'TH OBSERVATION
C
C     *    *    *    *
      DO 2 J=1,N
    2 IK(J)=0
      IF(I.EQ.1)RETURN
      K1=I-1
      J=0
  101 J=J+1
      IF(J.GT.NG)RETURN
      L=(K1-1)/IL(J)
      M=K1-IL(J)*L
      K1=L+1
      JJ=0
  201 JJ=JJ+1
      IF(JJ.GT.N)GO TO 101
      KK1=ID(J,JJ)*M
      L=NL(JJ)
      LL=KK1-L*(KK1/L)
      IK(JJ)=MOD(IK(JJ),LL,L)
      GO TO 201
      END
```

```
      FUNCTION IHCF(JJ,KK)
      J=JJ
C     *    *    *    *
C
C     FIND HIGHEST COMMON FACTOR OF JJ AND KK
C
C     *    *    *    *
      K=KK
    2 IF(J.GT.K)GO TO 5
      IF(J.EQ.K)GO TO 8
      ID=K
      K=J
      J=ID
    5 ID=J-K
      IF(ID.EQ.0)GO TO 8
      J=K
      K=ID
      GO TO 2
    8 IHCF=K
      RETURN
      END
      FUNCTION NEW(I)
C     *   *   *   *
C
C     TEST IF "I" IS OF FORM 2**R + 1
C
C     *   *   *   *
      IF(I.EQ.2)GO TO 4
      NEW=0
      J=1
      I1=I-1
    3 J=J*2
      IF(J.LT.I1)GO TO 3
      IF(J.EQ.I1)NEW=1
      RETURN
    4 NEW=1
      RETURN
      END
```

```
      FUNCTION JORD(L,N)
C     *    *    *    *
C
C     FIND ORDER OF ELEMENT  L
C     IN CYCLIC GROUP OF ORDER  N
C
C     *    *    *    *
      IF(L.EQ.0)JORD=1
      IF(L.NE.0)JORD=N/IHCF(L,N)
      RETURN
      END
```

```
  SUBROUTINE RANDU(IX,IY,YFL)
  IY=IX*899
  IF(IY)5,6,6
5 IY=IY+32767+1
6 YFL=IY
  YFL=YFL/32767.0
  RETURN
  END
```

The Automatic Design of Experiments

Some Practical Algorithms

APPENDIX THREE

The programs listed in this appendix represent a full implementation of the algorithms developed in chapter seven except for those subroutines already listed in appendix two and for which there are no changes. These subroutines are: ENFAC, FASET, DEFGEN, FRADE, MOD, SELG, LEV, LCM, IHCF, NEW, JORD, RANDU.

The main program MULFAC has been modified, as described in chapter seven, and links with the second main program REDDES.

The programs have been written in standard Fortran 4 with the same exceptions and extra functions as were described in appendix one.

Again, users should carefully check that the dimension statements are adequate for their problems.

```
C MULFAC
      DIMENSION IX(16,4),MVI(16),IK(16),LF(16)
     1,IDD(8,16),IB(8),II(16),NLL(16)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32),
     1NL(16),KEST,NG,NO,NC,NC1,ID(8,16),IL(8),ICON(20,20)
      CALL ENFAC
      DO 1 I=1,8
      DO 1 J=1,16
    1 ID(I,J)=0
      NG=0
      CALL FASET(IX,JG,LI)
C     *    *    *    *
C
C     FIND NEXT SUITABLE  FACTOR SUBSET
C
C     *    *    *    *
   20 IG=0
      MIN=100
   21 IG=IG+1
      IF((IX(IG,1).LT.MIN)
     1.AND.((IX(IG,2).GT.2).OR.
     1(IX(IG,3).NE.0)))GO TO 23
      GO TO 24
   23 MIN=IX(IG,1)
      IM=IG
   24 IF(IG.EQ.LI)GO TO 25
      GO TO 21
   25 IF(MIN.EQ.100)GO TO 100
      IF(IX(IM,2).GT.2)GO TO 27
      NGI=0
   26 NGI=NGI+1
      IB(NGI)=2**(NGI-1)
      IF(NGI.EQ.IX(IM,2))GO TO 66
      GO TO 26
C     *    *    *    *
C
C     INITIALISE FOR ENTRY TO DEFGEN
C
C     *    *    *    *
   27 NIG=IX(IM,2)+IX(IM,4)
      NVI=IX(IM,4)
   29 NVI=NVI+1
      MVI(NVI)=0
      MVI(NVI)=IONBT(MVI(NVI),NVI)
      IF(NVI.LT.NIG)GO TO 29
      MASK=0
C     *    *    *    *
C
C     FIND INTERACTIONS BETWEEN FACTORS IN CURRENT SUBSET
C
C     *    *    *    *
      I=0
   33 I=I+1
      IF(I.GT.N)GO TO 37
      IF(NL(I).EQ.IX(IM,1))MASK=IONBT(MASK,I)
      GO TO 33
   37 I=N
   38 I=I+1
      IF(I.GT.NV)GO TO 50
      ITT=MASK.AND.MV(I)
      IF(ITT.EQ.0)GO TO 38
      I1=0
      J=0
      NVI=NVI+1
      MVI(NVI)=0
   42 J=J+1
      IF(J.GT.N)GO TO 38
      IF(ITEST(MASK,J).NE.1)GO TO 47
      I1=I1+1
      IF(ITEST(MV(I),J).EQ.1)MVI(NVI)=IONBT(MVI(NVI),I1)
   47 GO TO 42
   50 CALL DEFGEN(NGI,IB,NIG,NVI,MVI)
      NOG=IX(IM,1)**NGI
C     *   *   *   *
C
C     IS SUBSET LINKED TO A PREVIOUS ONE?
C
C     *   *   *   *
      IF(IX(IM,3).EQ.0)GO TO 70
   66 J=IX(IM,4)
      J1=IX(J,3)-1
      LL=IX(J,4)
      GO TO 80
   70 I=0
      L=0
      LL=0
      J1=NG
   71 I=I+1
      IF(I.GT.LI)GO TO 80
      IF(I.EQ.IM)GO TO 71
      IF(IX(I,4).NE.0)GO TO 71
      IF(IX(I,1).NE.NOG)GO TO 71
      IX(I,3)=IM
      IX(IM,3)=NG+1
      JG=JG+1
      IX(IM,4)=NG+NGI
      IX(I,4)=1
C     *   *   *   *
C
C     CONVERT GENERATORS FROM BINARY  TO INTEGER FORM
C
C     *   *   *   *
   80 I=0
   81 I=I+1
      IF(I.LE.NGI)GO TO 85
      IX(IM,1)=100
      IF(J1.GT.NG)NG=J1
      GO TO 20
   85 J1=J1+1
      L=0
      J=0
   86 J=J+1
      IF(J.GT.N)GO TO 81
      IF(NL(J).NE.IX(IM,1))GO TO 86
      L=L+1
      ID(J1,J)=ITEST(IB(I),L)
      GO TO 86
```

```
C     *   *   *   *
C
C     HOW MANY FACTORS NOT YET IN GENERATORS
C
C     *   *   *   *
  100 L=0
      I=0
  101 I=I+1
      IF(I.GT.LI)GO TO 105
      IF(IX(I,1).NE.100)L=L+IX(I,2)
      GO TO 101
  105 IF(L.EQ.0)GO TO 200
      IF(L.GT.2)GO TO 120
      I=0
  108 I=I+1
      IF(I.GT.LI)GO TO 200
      IF(IX(I,4).NE.0)GO TO 108
      I1=IX(I,1)
      I2=0
      I3=0
  112 I2=I2+1
      IF(I2.GT.IX(I,2))GO TO 108
  114 I3=I3+1
      IF(I3.GT.N)GO TO 108
      IF(NL(I3).NE.I1)GO TO 114
      NG=NG+1
      ID(NG,I3)=1
      GO TO 112
C     *   *   *   *
C
C     INITIALISE FOR DEFGEN
C
C     *   *   *   *
  120 NIG=L
      NVI=0
  121 NVI=NVI+1
      MVI(NVI)=0
      MVI(NVI)=IONBT(MVI(NVI),NVI)
      IF(NVI.LT.NIG)GO TO 121
      MASK=0
      I=0
  126 I=I+1
      IF(I.GT.LI)GO TO 135
      IF(IX(I,4).NE.0)GO TO 126
      I1=IX(I,1)
      I2=0
  130 I2=I2+1
      IF(I2.GT.N)GO TO 126
      IF(NL(I2).NE.I1)GO TO 130
      MASK=IONBT(MASK,I2)
      GO TO 130
  135 I=N
  136 I=I+1
      IF(I.GT.NV)GO TO 150
      ITT=MASK.AND.MV(I)
      IF(ITT.EQ.0)GO TO 136
      I1=0
      J=0
      NVI=NVI+1
      MVI(NVI)=0
  140 J=J+1
      IF(J.GT.N)GO TO 136
      IF(ITEST(MASK,J).NE.1)GO TO 140
      I1=I1+1
      IF(ITEST(MV(I),J).EQ.1)MVI(NVI)=IONBT(MVI(NVI),I1)
      GO TO 140
  150 CALL DEFGEN(NGI,IB,NIG,NVI,MVI)
      I=0
      I2=0
  156 I=I+1
      IF(I.GT.N)GO TO 170
      I1=0
  159 I1=I1+1
      IF(I1.GT.LI)GO TO 156
      IF(IX(I1,4).NE.0)GO TO 159
      IF(NL(I).NE.IX(I1,1))GO TO 159
      I2=I2+1
      NLL(I2)=NL(I)
      LF(I2)=I
      GO TO 159
  170 CALL SELG(NIG,NGI,IB,NLL,LI,IX,IDD)
C     *   *   *   *
C
C     COPY GENERATORS FROM SELG INTO GENERATORS
C          FOR ALL FACTORS
C
C     *   *   *   *
      I=0
  181 I=I+1
      IF(I.GT.NGI)GO TO 200
      NG=NG+1
      J=0
  184 J=J+1
      IF(J.GT.NIG)GO TO 181
      I1=LF(J)
      ID(NG,I1)=IDD(I,J)
      GO TO 184
C     *   *   *   *
C
C     FIND FULL DESIGN  SIZE
C
C     *   *   *   *
  200 I=0
      WRITE(5,500)NAMEX
  500 FORMAT(///'   *   *   *   *   *'///'GENERATORS FOR  ',10A
     1///)
      NO=1
  201 I=I+1
      IF(I.GT.NG)GO TO 208
      J=0
  204 J=J+1
      IF(J.GT.N)GO TO 207
      II(J)=JORD(ID(I,J),NL(J))
      GO TO 204
  207 IL(I)=LCM(II,N)
  501 FORMAT(10X,16I2)
      NO=NO*IL(I)
      GO TO 201
  208 NFULL=1
      I=0
  209 I=I+1
      IF(I.GT.N)GO TO 2091
      NFULL=NFULL*NL(I)
      GO TO 209
 2091 IF(NO.LE.NFULL)GO TO 2095
      NO=NFULL
      NG=N
      I=0
 2092 I=I+1
      IF(I.GT.N)GO TO 2095
      J=0
      IL(I)=NL(I)
 2093 J=J+1
      IF(J.GT.N)GO TO 2092
      ID(I,J)=0
      IF(I.EQ.J)ID(I,J)=1
      GO TO 2093
 2095 DO 20951 I=1,NG
20951 WRITE(5,501)(ID(I,J),J=1,N)
      CALL CONTRA
      WRITE(5,2096)NO,NC1
 2096 FORMAT(//'BALANCED DESIGN HAS',I5,' POINTS.  AT LEAST',I4,
     1' NEEDED.'/'TYPE Y FOR BALANCED DESIGN ELSE N')
      READ(5,2097)IY
 2097 FORMAT(1A1)
      IF(IY.EQ.'Y')GO TO 210
      CALL LINK('REDDES')
  210 I=0
      WRITE(5,350)NAMEX
  350 FORMAT(///'   *   *   *   *   *'//'DESIGN FOR ',10A2
     1///)
  211 I=I+1
      IF(I.GT.NO)GO TO 1000
      CALL LEV(I,IK)
      WRITE(5,301)(IK(LL),LL=1,N)
  301 FORMAT(10I6)
      GO TO 211
 1000 CALL RANDOM(NO)
      STOP
      END
```

```
C REDDES
C     *    *    *    *
C
C     REDUCE DESIGN
C
C     *    *    *    *
      BIT BITS(3200)
      DIMENSION II(20),IY(20),IA(20)
     1,IC(20),IK(16),Y(20),A(20,20),B(20,20),C(20,20)
     1,ETA(20)
      COMMON N,NF,KD(128),NI,K1(128),NAMEX(10),KK1(512),NE,NV,MV(32)
     1,NL(16),KEST,NG,NO,NC,NC1,ID(8,16),IL(8),ICON(20,20)
      EQUIVALENCE(IY(2),II(1))
C     *    *    *    *
C
C     INITIALISE
C
C     *    *    *    *
      CALL FREMAT
      WRITE(5,1)
    1 FORMAT('ENTER ND:DESIGN SIZE WANTED')
      READ(5,2)ND
    2 FORMAT(V)
      DO 5 I=1,NO
    5 BITS(I)=.FALSE.
      IY(1)=1
      JJJ=1
      K=0
   81 I=0
      K=K+1
      IF(K.GT.NO)GO TO 98
   82 I=I+1
      IF(I.GT.NO)GO TO 81
      IF(BITS(I))GO TO 82
      CALL LEV(I,IK)
      CALL DROW(IK,II)
      IF(IY(JJJ).NE.1)GO TO 82
      L=JJJ
   88 L=L+1
      IF(L.GT.NC1)GO TO 92
      IF(IY(L).NE.0)GO TO 82
      GO TO 88
   92 L=0
   93 L=L+1
      IF(L.GT.NC1)GO TO 96
      A(JJJ,L)=IY(L)
      GO TO 93
   96 IA(JJJ)=I
      JJJ=JJJ+1
      BITS(I)=.TRUE.
      IF(JJJ.LE.NC1)GO TO 81
   98 IC(1)=1000
      DO 99 I=2,NC1
   99 IC(I)=0
```

```
C     *    *    *    *
C
C     FIND EXTRA ROWS,IF NEEDED,IN ORDER
C          OF DIAGONAL VALUES=1
C
C     *    *    *    *
      IF(JJJ.GE.NC1)GO TO 120
  101 I=1
  102 I=I+1
      IF(I.GT.NO)GO TO 101
      K=0
  105 K=K+1
      IF(K.GT.NC1)GO TO 109
      IF(I.EQ.IA(K))GO TO 105
  109 BITS(I)=.FALSE.
      CALL LEV(I,IK)
      CALL DROW(IK,II)
      IF(IY(JJJ).NE.1)GO TO 102
      K=0
  114 K=K+1
      IF(K.GT.NC1)GO TO 117
      A(JJJ,K)=IY(K)
      GO TO 114
  117 IA(JJJ)=I
      BITS(I)=.TRUE.
      JJJ=JJJ+1
      IF(JJJ.LE.NC1)GO TO 102
  120 CALL INVERT(A,B,DETA,NC1)
C     *    *    *    *
C
C     FIND NEXT ROW TO ENTER
C     BITS SET IF ROWS HAVE BEEN IN
C
C     *    *    *    *
  125 IN=1
  126 IN=IN+1
      IF(IN.GT.NO)GO TO 180
      IF(BITS(IN))GO TO 126
      BITS(IN)=.TRUE.
      CALL LEV(IN,IK)
      CALL DROW(IK,II)
C     *    *    *    *
C
C     COMPUTE Y=B'*IY
C
C     *    *    *    *
      I=0
      YMAX=-1000.
      ICMIN=1000
  133 I=I+1
      IF(I.GT.NC1)GO TO 146
      Y(I)=0.
      J=0
  136 J=J+1
      IF(J.GT.NC1)GO TO 140
      Y(I)=Y(I)+B(J,I)*IY(J)
      GO TO 136
  140 IF(Y(I).LT.YMAX)GO TO 133
      IF(Y(I).LT.1)GO TO 133
      IF(I.EQ.1)GO TO 133
```

```
C     *    *    *    *
C
C     IF THERE ARE SEVERAL EQUAL MAXIMUM VALUES
C        OF Y(I),CHOOSE THE FIRST FOR WHICH
C           IC(I) IS LEAST.
C
C     *    *    *    *
      IF(Y(I).GT.YMAX)GO TO 144
      IF(IC(I).LT.ICMIN)GO TO 145
      GO TO 133
  144 ICMIN=IC(I)
  145 YMAX=Y(I)
      IOUT=I
      GO TO 133
  146 IF(YMAX.EQ.-1000.)GO TO 126
      J=IA(IOUT)
      IA(IOUT)=IN
      IC(IOUT)=IC(IOUT)+1
C     *    *    *    *
C
C     DETERMINANT OF NEW MATRIX
C
C     *    *    *    *
      DETA=YMAX*DETA
C     *    *    *    *
C
C     COMPUTE ETA
C
C     *    *    *    *
      I=0
  151 I=I+1
      IF(I.GT.NC1)GO TO 160
      ETA(I)=-Y(I)/YMAX
      IF(I.EQ.IOUT)ETA(I)=1.0/YMAX
      GO TO 151
C     *    *    *    *
C
C     COMPUTE NEW MATRIX
C
C     *    *    *    *
  160 I=0
  161 I=I+1
      IF(I.GT.NC1)GO TO 175
      J=0
  164 J=J+1
      IF(J.GT.NC1)GO TO 161
      C(I,J)=0.
      K=0
  167 K=K+1
      IF(K.GT.NC1)GO TO 164
      IF(K.NE.IOUT)GO TO 172
      C(I,J)=C(I,J)+B(I,K)*ETA(J)
      GO TO 167
  172 IF(K.NE.J)GO TO 167
      C(I,J)=C(I,J)+B(I,K)
      GO TO 167
  175 CALL SWAP(B,C,NC1)
      GO TO 125
  180 WRITE(5,780)
  780 FORMAT(//'BASIC DESIGN'//)
      NI=0
  182 NI=NI+1
      IF(NI.GT.NC1)GO TO 190
      J=IA(NI)
      CALL LEV(J,IK)
      WRITE(5,781)(IK(L),L=1,N)
  781 FORMAT(16I6)
      GO TO 182
  190 WRITE(5,782)
  782 FORMAT(//'EXTRA ROWS'//)
C     *    *    *    *
C
C     SET A=A'A
C
C     *    *    *    *
      DO 201 I=1,NO
      BITS(I)=.FALSE.
      DO 199 J=1,NC1
      IF(I.EQ.IA(J))GO TO 200
  199 CONTINUE
      GO TO 201
  200 BITS(I)=.TRUE.
  201 CONTINUE
      DO 202 I=1,NC1
      DO 203 J=1,NC1
      C(I,J)=0.
      DO 205 K=1,NC1
  205 C(I,J)=C(I,J)+A(K,I)*A(K,J)
  203 CONTINUE
  202 CONTINUE
      IF(NI.GT.ND)GO TO 250
      CALL SWAP(A,C,NC1)
C     *    *    *    *
C
C     SET B=BB'
C
C     *    *    *    *
      DO 208 I=1,NC1
      DO 209 J=1,NC1
      C(I,J)=0.
      DO 211 K=1,NC1
  211 C(I,J)=C(I,J)+B(I,K)*B(J,K)
  209 CONTINUE
  208 CONTINUE
      CALL SWAP(B,C,NC1)
      DETA=DETA*DETA
```

```
C     *    *    *    *
C
C     LOOK FOR AVAILABLE ROW AND TEST IT.
C
C     *    *    *    *
  215 I=1
      DTMAX=-1.
  216 I=I+1
      IF(I.GT.NO)GO TO 233
      IF(BITS(I))GO TO 216
      CALL LEV(I,IK)
      CALL DROW(IK,II)
      J=0
      DTEST=0.
  222 J=J+1
      IF(J.GT.NC1)GO TO 231
      IF(IY(J).EQ.0)GO TO 222
      K=J
      DTEST=DTEST+B(J,J)
  227 K=K+1
      IF(K.GT.NC1)GO TO 222
      IF(IY(K).EQ.0)GO TO 227
      DTEST=DTEST+2*B(J,K)
      GO TO 227
  231 IF(DTEST.LT.DTMAX)GO TO 216
      DTMAX=DTEST
      IN=I
      GO TO 216
  233 CALL LEV(IN,IK)
      BITS(IN)=.TRUE.
      WRITE(5,781)(IK(L),L=1,N)
      IF(NI.GE.ND)GO TO 250
      NI=NI+1
      D=1+DTMAX
      DETA=D*DETA
      CALL DROW(IK,II)
C     *    *    *    *
C
C     COMPUTE NEW INVERSE
C
C     *    *    *    *
      IF(D.LE.0.0001)GO TO 215
      DO 239 I=1,NC1
  239 Y(I)=0.
      I=0
  241 I=I+1
      IF(I.GT.NC1)GO TO 246
      IF(IY(I).EQ.0)GO TO 241
      DO 244 J=1,NC1
  244 Y(J)=Y(J)+B(I,J)
      GO TO 241
  246 DO 247 I=1,NC1
      DO 248 J=1,NC1
  248 B(I,J)=B(I,J)-Y(I)*Y(J)/D
  247 CONTINUE
      GO TO 215
  250 CALL RANDOM(ND)
      STOP
      END
      SUBROUTINE INVERT(A,B,D,M)
      DIMENSION A(20,20),B(20,20)
      D=1.
      TH=1.E-7
      CALL SWAP(B,A,M)
      DO 20 I=1,M
      D=D*B(I,I)
      IF(B(I,I).LT.TH)D=0.
      IF(D.EQ.0.)RETURN
      B(I,I)=1./B(I,I)
      DO 10 J=1,M
      IF(J.NE.I)B(I,J)=B(I,J)*B(I,I)
   10 CONTINUE
      DO 16 II=1,M
      IF(II.EQ.I)GO TO 16
      DO 15 J=1,M
      IF(J.EQ.I)GO TO 15
      B(II,J)=B(II,J)-B(I,J)*B(II,I)
   15 CONTINUE
   16 CONTINUE
      DO 18 J=1,M
      IF(J.NE.I)B(J,I)=-B(J,I)*B(I,I)
   18 CONTINUE
   20 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DROW(IK,II)
      DIMENSION IK(16),II(20)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32),
     1NL(16),KEST,NG,NO,NC,NC1,ID(8,16),IL(8),ICON(20,20)
C     *    *    *    *
C
C     DESIGN ROW
C
C     *    *    *    *
      I=0
    2 I=I+1
      IF(I.GT.NC)RETURN
      II(I)=0
      L=0
      J=0
    5 J=J+1
      IF(J.GT.N)GO TO 11
      IF(ICON(I,J).EQ.0)GO TO 5
      IF(IK(J).EQ.0)GO TO 5
      IF(ICON(I,J).NE.IK(J))GO TO 2
      L=L+1
      GO TO 5
   11 II(I)=ITEST(L,1)
      GO TO 2
      END
      SUBROUTINE CONTRA
      DIMENSION IDT(8,20),ILT(8)
      COMMON N,NF,KD(128),NI,K1(128),NAMEX(10),KK1(512),NE,NV,MV(32),
     1NL(16),KEST,NG,NO,NC,NC1,ID(8,16),IL(8),ICON(20,20)
C     *    *    *    *
C
C     SET UP MAIN EFFECT CONTRASTS
C
C     *    *    *    *
      I=0
      I1=0
   11 I=I+1
      L=0
      IF(I.GT.N)GO TO 30
   13 L=L+1
      IF(L.EQ.NL(I))GO TO 11
      I1=I1+1
      J=0
   16 J=J+1
      IF(J.GT.N)GO TO 19
      ICON(I1,J)=0
      GO TO 16
   19 ICON(I1,I)=L
      GO TO 13
C     *    *    *    *
C
C     FIND NN=NUMBER OF CONTRASTS RELATED TO
C          AN INTERACTION MV(I)
C
C     *    *    *    *
   30 IF(I.GT.NV)GOTO 80
      J=0
      NN=1
   31 J=J+1
      IF(J.GT.N)GO TO 35
      IF(ITEST(MV(I),J).EQ.0)GO TO 31
      NN=NN*(NL(I)-1)
      GO TO 31
C     *    *    *    *
C
C     SET UP CONTRAST GENERATORS "IDT" AND THEIR ORDERS
C         'ILT' GIVEN MV(I)
C
C     *    *    *    *
   35 KG=0
      J=0
   36 J=J+1
      IF(J.GT.N)GO TO 46
      IF(ITEST(MV(I),J).EQ.0)GO TO 36
      KG=KG+1
      L=0
   40 L=L+1
      IF(L.GT.N)GO TO 36
      IF(L.NE.J)GO TO 45
   44 IDT(KG,L)=1
      ILT(KG)=NL(J)
      GO TO 40
   45 IDT(KG,L)=0
      GO TO 40
C     *    *    *    *
C
C     USE GENERATORS TO FIND CONTRASTS 'ICON'
C
C     *    *    *    *
   46 IJ=0
   47 IJ=IJ+1
      IF(IJ.GT.NN)GO TO 30
      I1=I1+1
      J=0
   50 J=J+1
      IF(J.GT.N)GO TO 55
      ICON(I1,J)=0
      GO TO 50
   55 K=IJ
      J=0
   56 J=J+1
      IF(J.GT.KG)GO TO 69
      L=(K-1)/ILT(J)
      M=K-ILT(J)*L
   60 K=L+1
      JJ=0
   62 JJ=JJ+1
      IF(JJ.GT.N)GO TO 56
      KK=IDT(J,JJ)*M
      L=NL(JJ)
      LL=KK-L*(KK/L)
      ICON(I1,JJ)=MOD(ICON(I1,JJ),LL,L)
      GO TO 62
   69 J=0
   70 J=J+1
      IF(J.GT.N)GO TO 75
      IF(ITEST(MV(I),J).EQ.0)GO TO 70
      IF(ICON(I1,J).NE.0)GO TO 70
      I1=I1-1
      GO TO 47
   75 I=I+1
      GO TO 30
   80 NC=I1
      NC1=NC+1
      RETURN
      END
      BIT BITS(3200)
      DIMENSION JJ(200)
      COMMON N,NF,KD(128),NI,K(128),NAMEX(10),KK(512),NE,NV,MV(32)
     1,NL(16),KEST
C     *    *    *    *
C
C     RANDOMISE ORDER OF OBSERVATIONS
C
C     *    *    *    *
 1000 IXX=1+2*(8191.AND.KEST)
      WRITE(5,601)
  601 FORMAT('RANDOMISED ORDER')
  604 XX=NO
      DO 312 I=1,NO
  312 BITS(I)=.FALSE.
      L=0
      JO=NO
      IF(NO.LE.200)GO TO 605
      JO=200
      JK=0
  605 L=L+1
      IF(NO.GT.L*200)GO TO 606
      JO=NO-(L-1)*200
      JK=1
  606 DO 317 J=1,JO
      W=1./XX
  318 DO 316 I=1,NO
      IF(BITS(I))GO TO 316
      CALL RANDU(IXX,IXX,YY)
  301 FORMAT(10I6)
      IF(YY.GT.W)GO TO 316
      XX=XX-1
      JJ(J)=I
      BITS(I)=.TRUE.
      GO TO 317
  316 CONTINUE
      GO TO 318
  317 CONTINUE
      WRITE(5,301)(JJ(J),J=1,JO)
      IF(JK.NE.1)GO TO 605
      WRITE(5,602)
  602 FORMAT('IS ANOTHER RANDOM NUMBER STREAM WANTED,TYPE YES
     1OR NO')
      READ(5,603)IY
  603 FORMAT(1A1)
      IF(IY.EQ.'Y')GO TO 604
      RETURN
      END


      SUBROUTINE SWAP(A,B,N)
C     *    *    *    *
C
C     COPIES N SQUARE MATRIX B INTO A
C
C     *    *    *    *
      DIMENSION A(20,20),B(20,20)
      DO 1 I=1,N
      DO 2 J=1,N
    2 A(I,J)=B(I,J)
    1 CONTINUE
      RETURN
      END
```