FINAL REPORT
Grant Number NAG-1-1378

EFFECT OF DESIGN SELECTION ON
RESPONSE SURFACE PERFORMANCE

from

William C. Carpenter
Department of Civil Engineering and Mechanics
University of South Florida
Tampa, Florida 33620

to

Jean-Francois M. Barthelemy
Office of Interdisciplinary Research
Structural Dynamics Division
NASA Langley Research Center
Hampton, Virginia 23681-0001

Table of Contents

1. Introduction

The mathematical formulation of the engineering optimization problem is

$$\begin{gathered} \min \; f(\{x\}) \\ \textit{subject to} \;\; g_i(\{x\}) \le 0, \; i=1,q \end{gathered} \tag{1}$$

where

$\{x\}$ is an nx1 matrix of design variables,

$f(\{x\})$ is the objective function, and

$g_i(\{x\})$ are constraint equations.

Evaluation of the objective function and constraint equations in Equation (1) can be very expensive in a computational sense. Thus, it is desirable to use as few evaluations as possible in obtaining its solution. In solving Equation (1), one approach is to develop approximations to the objective function and/or restraint equations and then to solve Equation (1) using these approximations in place of the original functions. These approximations are referred to as response surfaces.

The desirability of using response surfaces depends upon the number of functional evaluations required to build the response surfaces compared to the number required in the direct solution of Equation (1) without approximations. The present study is concerned with evaluating the performance of response surfaces so that a decision can be made as to their effectiveness in optimization applications. In particular, this study focuses on how the quality of approximations is effected by design selection. Polynomial approximations and neural net approximations are considered.

To provide the groundwork for future discussion, this introductory section discusses:

1. measures of quality of fit at the designs and measures of quality of fit over a region of interest and

2. the methodology used to build the approximations.

1.1 Quality of Fit

Let us consider a problem with n design variables, the components of the vector $\{x\} = \{x_1, x_2, \ldots x_n\}^t$. A total of N designs will be considered: $\{x\}_j$, $j=1,N$. At the designs $\{x\}_j$, let

y_j = the value of the function to be approximated and

$\hat{y}_j$ = the value of the approximating function.

The approximating function, $\hat{y}$, should closely match the function, y, not only at the designs, $\{x\}_j$, but over the entire region of interest.

1.1.1 Fit at the designs

The approximating function $\hat{y}$ closely approximates the function y when s is small where

$$s=\sqrt{\frac{\delta^2}{N}} \tag{2}$$

and where δ^2 is the sum of the squares of the residuals thus

$$\delta^2 = \sum_1^N (y_i - \hat{y}_i)^2 \tag{3}$$

Let $\bar{y}$ be the average value of the designs, y_i. Thus

$$\bar{y} = \frac{\sum_1^N y_i}{N} \tag{4}$$

In this study, one measure of the closeness of fit to be considered is the non-dimensional value v where

$$v = \frac{\sqrt{\dfrac{\sum_1^N (y_i - \hat{y}_i)^2}{N}}}{\bar{y}} * 100 \tag{5}$$

The coefficient v is the non-dimensional root mean square (RMS) error at the designs. Thus, v = 0 is a necessary and sufficient condition that the approximating function fit the actual function at the N design points.

1.1.2 Overall fit

Just because the approximating function exactly fits the function at N designs does not guarantee that it gives a good fit over the region of interest. It is therefore desirable over the region of interest to have a measure of the quality of overall fit. Several examples of this study considers a two dimensional region of interest. For these problems, the rectangular region of interest is overlaid with a 31x31 evenly spaced grid of points. The value of the function and the approximating function is then compared at these NG=961 evenly spaced grid of points. Other examples consider a rectangular n dimensional region of interest. These regions of interest are also overlaid with a evenly spaced grid of points. The value of the function and the approximating function are then compared at these NG grid points. For these examples, a measure of the quality of overall fit is taken as

$$v_G=\frac{\sqrt{\sum_{1}^{NG}(y_i-\hat{y}_i)^2/NG}}{\overline{y_G}} * 100 \tag{6}$$

where $\bar{y}_G$ is the average value of y at the grid points. A small value of v_G indicates that the approximating function did a good job of approximation over the region of interest.

1.2. Polynomial Approximations

With the polynomial response surface approach, the approximating function is taken as an m=k+1 term polynomial expression [1-3] thus

$$\hat{y}=b_o+b_1X_1+\dots b_kX_k \tag{7}$$

where X_j is some expression involving the design variables. For example, a second order polynomial approximation in two variables could be of the form

$$\hat{y}=b_o+b_1x_1+b_2x_2+b_3x_1^2+b_4x_1x_2+b_5x_2^2 \tag{8}$$

The value of the function to be approximated at the N designs can be used to determine the m=k+1 undetermined coefficients in the polynomial expression. For the N designs, Equation (7) yields

$$\begin{Bmatrix} y_1 \\ y_2 \\ .. \\ y_N \end{Bmatrix} = \begin{bmatrix} 1 & X_{1_1} & \cdots & X_{k_1} \\ 1 & X_{1_2} & \cdots & X_{k_2} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & X_{1_N} & \cdots & X_{k_N} \end{bmatrix} \begin{Bmatrix} b_o \\ b_1 \\ .. \\ b_k \end{Bmatrix} \tag{9}$$

or

$$\{Y\}=[Z]\{b\} \tag{10}$$

where {Y} is an Nx1 matrix, [Z] is an Nxm matrix, and {b} is an mx1 matrix.

1.2.1 Exactly-determined approximation

When N=m, the approximation is exactly-determined and the matrix {b} can be determined from Equation (10).

1.2.2 Over-determined approximation

With N>m, Equation (10) can be solved in a least squares sense thus [1-3]

$$[Z]^t\{Y\}=[Z]^t[Z]\{b\} \tag{11}$$

or

$$\{b\}=([Z]^t[Z])^{-1}[Z]^t\{Y\} \tag{12}$$

Equation (12) in effect, chooses the terms of {b} so as to minimize the square of the residual as defined in Equation (2).

1.2.3 Under-determined approximation

When $N<m$, the approximation is under-determined. A solution can be obtained by choosing the terms of {b} so as to minimize the square of the residual as defined in Equation (2). However, a direct solution can be obtained by using the concept of pseudo-inverse [4,5]. Assume that the rank of matrix [Z] is N and define the pseudo-inverse of matrix Z, Z^* thus

$$[Z]^*=[Z]^t([Z][Z]^t)^{-1} \tag{13}$$

where t denotes transpose. Solution of Equation (10) is then

$$\{b\}=[Z]^*\{Y\}+[Q]\{w\} \tag{14}$$

where {w} is an (m-N) column matrix of arbitrary coefficients and [Q] is a mx(m-N) matrix formed from any m-N independent columns of the matrix [R] thus

$$[R]=[I]-[Z]^{*}[Z] \tag{15}$$

One solution to Equation (14) is to take all the arbitrary terms of {w} as zero giving

$$\{b\}=[Z]^{*}\{Y\} \tag{16}$$

The basic solution to Equation (10) is Equation (16). Using that equation, at the designs, $\{\mathbf{x}\}_j$, the value of $\hat{y}_j$ matches the value of y_j. If w_i is the ith term in matrix {w} and $\{q\}_i$ is the ith column of matrix [Q], then at the designs, $\{\mathbf{x}\}_j$, $\hat{y}_j=0$ when

$$\{b\}=w_i\{q\}_i \tag{17}$$

Thus, the last term of the right hand side of Equation (14) gives $\hat{y}_j$ values which match y_j at the designs, $\{\mathbf{x}\}_j$, for any values of w_i.

1.3 Artificial Neural Nets

While the initial motivation for developing artificial neural nets was to develop computer models that could imitate certain brain functions, neural nets can be thought of as another way of developing a response surface. Different types of neural nets are available [6,7], but the type of neural nets considered in this paper are back propagation nets with one hidden layer as shown in Figure 1. This type of neural net has been used previously to develop response surfaces [8-12] and is capable, with enough nodes on the hidden layer, of approximating any continuous function [13].

For the neural net of Figure 1, associated with each node on the hidden layer, node j, and each output node, node k, are coefficients or weights, θ_j and θ_k, respectively. These weights are referred to as the biases. Associated with each path, from an input node i to node j on the hidden layer, is an associated weight, w_{ij} and from node j on the hidden layer to output node k is an associated weight w_{jk}. Let q_i be inputs entered at node i. Node j on the hidden layer receives weighted inputs, $w_{ij}q_i$. It sums these inputs and uses an activation function to yield an output r_j. The activation function considered in this paper is the sigmoid function [6,7]

$$r_j = \frac{1}{1+e^{-\Sigma w_{ij} q_i - \theta_j}} \tag{18}$$

Output node k then receives inputs $w_{jk}r_j$ which are summed and used with an activation function to yield an output s_k. Some variation of the delta-error back propagation algorithm [6,7] is then used to adjust the weights on each learning try so as to reduce the values between the predicted and desired outputs. In this investigation, studies were performed using the program NEWNET [14] which was developed especially for this investigation. NEWNET minimizes the sum of the squares of the residuals in Equation (2) with respect to the weights and biases of the net. Training of the net is thus formulated as an unconstrained minimization problem. Solution of this minimization problem is performed using the method of Davidon, Fletcher, and Powell [15-16]. That algorithm performs a series of one dimensional searches along search directions. Search directions are determined by building an approximation to the inverse Hessian matrix using gradient information. Gradients required by that algorithm are obtained using back-propagation. One-dimensional searches are performed along the search directions using an interval shortening routine.

2. Levels of Designs

2.1 Taylor Series Approximation

The overriding factor which affects the accuracy of an approximation is the levels of the design parameters considered. It is instructive to consider a problem in two design variables. Suppose we wish to make a quadratic approximation of a function thus:

$$y=b_o+b_1x_1+b_2x_2+b_3x_1^2+b_4x_1x_2+b_5x_2^2\ldots \tag{19}$$

Consider that the exact function is evaluated at 6 design points and the information thus generated will be used to determine the 6 undetermined coefficients in Equation (19). Design variables at these design points are taken from the following sets:

$$\begin{aligned} &x_1 \textit{ from the set } \{x_{1_1}\ x_{1_2}..x_{1_p}\}\\ &x_2 \textit{ from the set } \{x_{2_1}\ x_{2_2}..x_{2_q}\}\end{aligned} \tag{20}$$

Here p discrete values are considered for x_1 and q discrete values are considered for x_2. The variable x_1 is said to have p levels and x_2 is said to have q levels. The problem is to determine the minimum levels of the design variables, p and q, required to build the quadratic approximation. In this regard, it is instructive to consider a Taylor series approximation [17] of the function about the point $\{x_1=0,\ x_2=0\}$:

$$\tilde{y}=y(0,0)+\{\nabla y(0,0)\}^t\{\Delta x\}+\{\Delta x\}^t[H(0,0)]\{\Delta x\}+\ldots \tag{21}$$

where

$$\{\Delta x\} = [(x_1 - 0)\ (x_2 - 0)]^t = [x_1\ x_2]^t \tag{22}$$

$$\{\nabla y(0,0)\} = [(\frac{\partial y(0,0)}{\partial x_1}\ \frac{\partial y(0,0)}{\partial x_2}]^t \tag{23}$$

$$[H(0,0)] = \begin{bmatrix} \frac{\partial^2 y(0,0)}{\partial x_1^2} & \frac{\partial y^2(0,0)}{\partial x_1 \partial x_2} \\ \frac{\partial y^2(0,0)}{\partial x_1 \partial x_2} & \frac{\partial^2 y(0,0)}{\partial x_2} \end{bmatrix} \tag{24}$$

Entering Equations (22), (23), and (24) into Equation (21) gives

$$\begin{aligned} \tilde{y} = y(0,0) + \frac{\partial y(0,0)}{\partial x_1} x_1 + \frac{\partial y(0,0)}{\partial x_2} x_2 + \frac{\partial^2 y(0,0)}{\partial x_1^2} x_1^2 + \\ 2\frac{\partial^2 y(0,0)}{\partial x_1 x_2} x_1 x_2 + \frac{\partial^2 y(0,0)}{\partial x_2^2} x_2^2 \end{aligned} \tag{25}$$

The derivatives in Equation (25) can be determined by finite difference equations [18]. The second derivative of y with respect to x_1 can be obtained using information at points indicated in Figure 2 by solid circles, the second derivative of y with respect to x_2 can be obtained using information at points indicated by unfilled circles, and the mixed derivative can be obtained using information at points indicated by unfilled squares.

It can be seen in Figure 2 that at least three levels of both x_1 and x_2 must be used to obtain a quadratic approximation. If three levels are not provided, not information is available to calculate the higher derivatives in Equation (25). A complete 3 factorial design does not have to be used--only 6 selected points from the complete 3 factorial design. Information at those 6 points allow the undetermined coefficients to be exactly determined.

Consider now the design of Figure 3 which are also taken from the 3 factorial design. Even though 6 design points are used, this set of design points does not allow an approximation containing the x_2^2 term of Equation (25). However, with the design of Figure 3, an approximation of the form of Equation (26) could be obtained thus:

$$y=b_o+b_1x_1+b_2x_2+b_3x_1^2+b_4x_1x_2 \tag{26}$$

With the design of Figure 3, if a solution is attempted using Equations (19) and (12), a singular coefficient matrix will be encountered. A solution could be attempted using the pseudo-inverse concept of Equations (13) and (14). However, recent studies [19] have shown that non-unique solutions are obtained with this technique. Non-uniqueness makes these solutions undesirable. Using Equations (26) and (12), a slightly over-determined approximation is obtained.

Recent studies have found that the numerical performance of neural network approximations and polynomial approximations with the same number of associated undetermined parameters is comparable [19]. Thus, it is not expected that neural nets as approximators will perform better than polynomials when there are inadequacies in the training design, as in Figure 3. The next example investigates performance of both polynomial and neural net approximations.

2.2 Example

Consider the function

$$y=1+x_1+x_2+x_3+x_1^2+x_1x_2+x_1x_3+x_2^2+x_2x_3+x_3^2 \tag{27}$$

In the first phase of the investigation, approximations are to be made of this function using the design of Figure 4. The star pattern of design points in Figure 4 does not allow mixed derivatives of the function to be calculated using finite difference type formulae but does permit the other second derivatives to be calculated. Thus, information is available to make a polynomial approximation of the form

$$\tilde{y}=b_o+b_1x_1+b_2x_2+b_3x_3+b_4x_1^2+b_5x_2^2+b_6x_3^2 \tag{28}$$

The function y was evaluated at the design points shown in Figure 4 yielding 7 training pairs for calculating the 7 undetermined parameters in Equation (28). The value of the approximating function $\hat{y}$ was then evaluated at a 5x5x5 grid of designs. These values of $\hat{y}$

were then used to evaluate v_G from Equation (6). The value of v_G obtained is shown in the first line of Table 2.1.

Table 2.1. Performance of Approximations for Various Designs

Number Designs Points	Description	Polynomial Approximation		Neural Net Approximation			
		No. Para.	v_G (%)	ih	No. Para.	No. Apx.	v_G (%)
7	Star--see Figure 4	7	34.6	2	11	10	25.5-97.3
12	Star--see Figure 5	7	34.6	2	11	10	32.9-93.5
10	Computer Generated	10	0.0	2	11	10	36.6-36.9
				3	16	10	21.9-36.7
27	3 factorial	10	0.0	3	16	2	16.6-16.7
				4	21	2	16.6-16.9
125	5 factorial	10	0.0	8	41	1	3.7

A neural net approximation was then considered. Previous studies [19] have indicated that it is desirable to have more training pairs than the number of undetermined parameters (weights and biases) associated with the net. If fewer training pairs than undetermined parameters are used, non-unique approximations should be expected. For a neural net with one hidden layer as shown in Figure 1, there are 6 parameters associated with a net with one node on the hidden layer and 11 parameters associated with a net with two nodes on the hidden layer. It was considered that one node on the hidden layer would yield an inadequate approximation. Thus 2 nodes on the hidden layer were considered. Thus, the neural net approximation is under-determined. That is to say that there are fewer training pairs than there are undetermined parameters associated with the approximation. Non-unique approximations are to be expected. Indeed, this was the case. The 8 training pairs were used to make 10 different approximations by having training commence from a different randomly selected set of weights and biases. Once the nets were trained, the value of the approximating function, $\hat{y}$, was generated at the 5x5x5 set one grid points and the value of v_G was developed. The range of the values obtained is shown in Table 2.1. One can see that a large range of values is obtained. The best neural net approximation is only slightly better than the polynomial approximation while the worst neural net approximation is considerably worse. Just as with the polynomial approximation, the designs used to train the approximation can not yield information necessary to capture essential features of the function to be approximated.

The 12 designs of Figure 5 were next used in the training of a polynomial approximation and a 2 node neural net approximation. Even though more designs are used here than in Figure 4, the additional designs selected do not yield any more information about the nature of the function being approximated. Information is still not available for determining the mixed derivatives of the function to be approximated. Thus, the polynomial approximation of Equation (26) was considered. As there are now more training pairs than there are undetermined parameters, the approximation obtained is over-determined. As no new information is available with the 12 designs, the same polynomial approximation and thus the same v_G as before are obtained. The value of v_G is shown in the second line of Table 2.1.

A neural net with 2 nodes on the hidden layer was then trained with the 12 training pairs. The net was trained 10 times starting from different randomly selected sets of weights and biases. Even thought the number of training pairs, 12, is greater than the number of undetermined parameters associated with the net, 11, non-unique approximations were obtained as can be seen in Table 2.1. Thus, it can be concluded that <u>for neural net approximations, having more training pairs than the number of associated undetermined parameters is only a necessary condition for obtaining a unique approximation but that it is not a sufficient condition</u>. As the 12 designs offered no new information about the function being approximated over that offered by the 8 designs, then just as with the 8 design case, non-unique approximations were obtained.

The program DESIGNS [20], which was developed for this project, was used to generate 10 designs which contain the information necessary for calculating the 10 undetermined coefficients of the complete quadratic approximation of the form:

$$\tilde{y}=b_o+b_1x_1+b_2x_2+b_3x_3+b_4x_1^2+b_5x_2^2+b_6x_3^2+b_7x_1x_2+b_8x_1x_3+b_9x_2x_3 \tag{29}$$

The location of these design points is shown in Figure 6. The polynomial approximation obtained by training the polynomial of Equation (29) with the computer generated designs exactly duplicated the test function of Equation (27). Thus, v_G for the 5x5x5 grid of points was zero as seen in the third line of Table 2.1.

A neural net with 2 nodes on the hidden layer with 6 associated undetermined parameters and a neural net with 3 nodes on the hidden layer and 11 associated undetermined parameters were then trained 10 times with the computer generated training pairs. Each training started from a different randomly selected set of weights and biases. For the case of 2 nodes on the hidden layer, the approximation generated was over-determined and a unique approximation was obtained (the small range of v_G obtained most likely results from the exit criteria employed in the training algorithm). For the case of 3 nodes on the hidden layer, there are 11 associated undetermined parameters but only 10 training pairs. Thus the approximation is under-determined and a non unique approximation is obtained as can be seen in Table 2.1.

The performance of the neural net approximations was much poorer than that of the polynomial approximation on this problem. This poorer performance may be in part because the problem is biased towards the polynomial approximation as the function being approximated is 2 second order polynomial.

A complete 3^3 factorial design and a 5^3 factorial design were considered to see if good results could be obtained with the neural nets if more training pairs were employed. Indeed this was the case. However, many more training pairs were required to get a good approximation than were required with the polynomial approximation. The extra training pairs were wasted on the polynomial approximation. Ten correctly selected training pairs is all that is required to get an exact second order approximation. The additional training pairs offered no new information to the polynomial approximation. The coefficient v_G was zero for training pairs using the 3 and 5 factorial designs and a second order polynomial approximation.

2.3 Conclusion

For a given order of approximation, a good design must use an adequate number of levels of the design variables or a poor approximation will be obtained. Likewise, design points must be located so that information is available for determining all of the undetermined coefficients of the approximating function. In many instances, especially when the region of interest is small, a second order polynomial approximation or neural net equivalent will be sufficient to build a response surface. A second order approximation requires a design containing 3 levels of the design variables. Program DESIGNS has been developed to generate a minimum point design which allows all of the coefficients of a second order polynomial approximating function to be obtained. This minimum point design can be augmented by randomly selected design points or by user selected points.

3. Standard Designs

3.1 Underlying Principle

When making a polynomial approximation of a function, the number of design levels required for each design variable depends upon the order of polynomial approximation being used. Consider for example the problem of approximating a function y, a function of one design variable. As previously discussed, two levels of the design variable would be required to make a linear approximation of the function, three levels of the design variable would be required to make a second order approximation, four levels of the design variable would be required to make a 3rd order approximation, etc. If y is a function of r design variables, a pth order polynomial approximation,$\hat{y}$, requires designs at $p+1$ levels in each design variable.

In response surface methodology, the term <u>factor</u> is used for design variable. A <u>factorial design</u> or <u>factorial experiment</u> is a design in which one uses each of the possible combinations of the levels of each factor. If m is the number of level of each factor and r is the number of factors, then the design would be referred to as a <u>m^r factorial experiment</u>. Table 3.1 gives the number of designs in various factorial experiments.

Table 3.1. Number of designs in a full factorial design

m = level r = factor	2	3	4
2	4	9	16
3	8	27	64
4	16	81	256
10	1024	59049	1.05E06

One can see that even for a small number of factors, complete factorial experiments become impractical if designs are computationally or experimentally expensive to obtain. One then is forced to use some sub-set of the factorial design or alternate designs containing requiring fewer design points. Concepts from statistics are normally used in selecting a sub-set of the factorial design or in developing alternate designs. Thus statistical concepts are reviewed.

3.2 Statistical Concepts

When making an approximation, $\hat{y}$, of a function, y, most approaches used to select design points for a design consider that

1. polynomial approximations are employed and
2. the value of the function, y_i, determined at the designs, $\{x\}_i$, contains some error, ϵ_i.

A measure of the error at point i is the variance of the error, $var(\epsilon_i) = \sigma^2$ where

$$\sigma^2 = \sum_{i=1}^{n} \frac{(y_i - \mu)^2}{n} \tag{30}$$

where

μ is the true mean of all possible observations of y_i and

n is the number of observations made.

In experimental investigations, ϵ_i is experimental error. When making approximations to analytical functions, ϵ_i is zero and the variance of the error at point i is zero. Often approximations are made to a function whose values must be obtained from some numerical algorithm such as the finite element method or finite difference method. Values of y_i obtained from such algorithms depend on control parameters which dictate the level of accuracy of the solution. For example, if y was a stress determined from a finite element analysis, then y could depend on a control parameter which specifies the coarseness of the finite element idealization. In this case, different values of y_i would be obtained for the ith design for different values of the control parameters and ϵ_i could be thought of as a numerical error.

It would be an interesting study to select designs such that approximations developed are insensitive to numerical errors such as finite element idealization error. However, the problem at hand is to find a good approximation to an analytical function or a good approximation for output from a deterministic model. For the problem at hand, for a given design, x_i, one obtains the same functional value, y_i, no matter how many times the function is evaluated. Thus, the problems considered in this report contain no numerical error. However, as all known algorithms with one exception [21] consider that there is some experimental or numerical error, this section now further examines this case.

Errors in the value of y_i used to build an approximation affect the estimation of the undetermined coefficients, b_j, in the polynomial approximation and thus affect $\hat{y}_i$, the values of y_i predicted by the approximation. A measure of the error in b_j resulting from errors in y_i is the variance of b_j. For example, consider that y_i is obtained from a finite element analysis and that a pth order polynomial approximation is employed. The undetermined coefficients in that approximations, b_j, can be determined from Equation (12). If a number of approximations were now made with finite element results, obtained using different idealizations, the coefficient b_j for these approximations would be different. The variance of b_j is a measure of how much the b's change for these different approximations. In like form, the different approximations yield different $\hat{y}_i$ and the variance of $\hat{y}_i$ is a measure of how much the $\hat{y}_i$ values change from approximation to approximation.

From a numerical standpoint, it is desirable to have approximations that are not highly sensitive to the error ϵ_i. Approximations are insensitive to the error, ϵ_i, if the variance of b_j and the variance of $\hat{y}_i$ is small. Most design selection algorithms currently in use attempt in some way to keep these variances small.

The variance of b_j is the j,j term of the variance-covariance matrix cov b where (see Equation 3.11 of [3] or Equation 2.8 of [2])

$$[cov\, b] = \sigma^2([Z]^t[Z])^{-1} \tag{31}$$

and the variance of $\hat{y}_i$ is given by (see Equation 2.11 of [2])

$$var\, \hat{y}_i = \sigma^2\{Z_i\}^t([Z]^t[Z])^{-1}\{Z_i\} \tag{32}$$

where $\{Z_i\}^t$ is the 1xp vector whose elements correspond to the elements of a row of matrix [Z].

Notice that these variance involve the matrix [H] where

$$[H] = ([Z]^t[Z])^{-1} \tag{33}$$

Design selection affects [Z], which from Equation (33) affects [H], which in turn affects the variances of b_j and $\hat{y}_i$. Many design point selection algorithms attempt to select designs which give an [H] matrix which will keep the variances of b_j and $\hat{y}_i$ small.

3.3 Orthogonal Designs

The associated undetermined coefficients of a polynomial approximation function can be found from Equation (12). The solution for these coefficients involve the matrix [Z] (see Equations (9) and (10)). Let $\{Z_i\}$ be the ith column of matrix [Z]. A design is said to be orthogonal if the columns of the [Z] matrix are orthogonal, i.e. $\{Z_i\}^t\{Z_j\}=0$, $i\neq j$. There are interesting properties of orthogonal designs which have prompted there use. Thus orthogonal designs will now to presented in some detail.

3.3.1 Scaling

The discussion of orthogonality is simplified by working with scaled variables. Consider that the approximation in question involves k unscaled design variables $\bar{x}_i$ and contains N design points. Instead of working with $\bar{x}_i$, the variables will be scaled. Let $\bar{x}_{iu}$ be the uth level of unscaled variable i and x_{iu} be the scaled level. The desired scaling is

$$\sum_{u=1}^{N} x_{iu}^2=N, \quad i=1,k \tag{34}$$

$$\sum_{u=1}^{N} x_{iu}=0, \quad i=1,k \tag{35}$$

This scaling can be accomplished by having

$$x_{iu}=\frac{\bar{x}_{iu}-\tilde{x}_i}{S_i} \tag{36}$$

where

$$\tilde{x}_i=\textit{the average of the levels of } \bar{x}_i \tag{37}$$

and

$$S_i^2=\sum_{u=1}^{N}\frac{(\bar{x}_{iu}-\tilde{x}_i)^2}{N} \tag{38}$$

With this scaling, N experimental design points of the orthogonal design give

$$[Z]^t[Z]=N[I] \tag{39}$$

$$([Z]^t[Z])^{-1}=\frac{1}{N}[I] \tag{40}$$

where [I] is the identity matrix.

3.3.1.1 Example of Scaled Designs:

Consider a 2 factorial design with levels of 4 and -4. For that design

$$\tilde{x}_1=0,\quad \tilde{x}_2=0 \tag{41}$$

and

$$S_1^2=S_2^2=\frac{(4-0)^2+(-4-0)^2}{2},\quad or\quad S_1=S_2=4 \tag{42}$$

From Equation (3), the levels of the scaled variables are

$$x_{iu}=\frac{\bar{x}_{iu}-0}{4} \tag{43}$$

or the levels of the scaled variables are 1 and -1.

3.3.2 Bias

Assume that the polynomial approximating function is inadequate. The coefficients of that polynomial can be determined from Equation (12). Let $\{\hat{b}_1\}$ be the coefficients thus obtained and let $[Z_1]$ be the corresponding [Z] matrix. Then from Equation (12)

$$\{\hat{b}_1\}=([Z_1]^t[Z_1])^{-1}[Z_1]^t\{Y\} \tag{44}$$

Assume that the function being approximated can be expressed as

$$\{Y\}=[Z]\{b\} \tag{45}$$

where

$$\{b\}=\begin{Bmatrix}\{b_1\}\\ \{b_2\}\end{Bmatrix},\quad [Z]=[\ [Z_1]\quad [Z_2]\ ] \tag{46}$$

Entering Equations (40), (45), and (46) into Equation (44) gives

$$\{\hat{b}_1\}=\frac{1}{N}[I][Z_1]^t([[Z_1]\quad [Z_2]])\begin{Bmatrix}\{b_1\}\\ \{b_2\}\end{Bmatrix} \tag{47}$$

Entering Equation (39) into Equation (47) gives

$$\{\hat{b}_1\}=\frac{1}{N}(N[I]\{b_1\}+[Z_1]^t[Z_2]\{b_2\}) \tag{48}$$

or

$$\{\hat{b}_1\} = \{b_1\} + \frac{1}{N}[Z_1]^t[Z_2]\{b_2\} = \{b_1\} + [A]\{b_2\} \tag{49}$$

where [A] is called the alias matrix. One can see in Equation (49) that the coefficients $\{\hat{b}_1\}$ will only be correct estimates of $\{b_1\}$ if the columns of $[Z_1]$ are orthogonal to the columns of $[Z_2]$. Special situations where this orthogonality occurs are next discussed.

3.3.2.1 A bias example--linear approximating polynomial but the exact function contains linear terms and cross-product terms:

Consider a linear approximating polynomial

$$\hat{y} = \hat{b}_o + \sum_{i=1}^{k} \hat{b}_i x_i \tag{50}$$

where the exact function is

$$y = b_o + \sum_{i=1}^{k} b_i x_i + \sum_{i=1}^{k} \sum_{j=i}^{k} b_{ij} x_i x_j \tag{51}$$

where b_{ij} are the undetermined coefficients associated with the cross-product terms. For this problem, a full 2^k factorial design gives that the columns of $[Z_1]$ are orthogonal to the columns of $[Z_2]$ and thus

$$\{\hat{b}_1\}=\{b_1\} \tag{52}$$

3.3.2.2 A bias example--linear approximating function but the exact function is a complete quadratic polynomial:

Consider a linear approximating polynomial

$$\hat{y}=\hat{b}_o+\sum_{i=1}^{k}\hat{b}_i x_i \tag{53}$$

where the exact function is a complete second order polynomial thus

$$y=b_o+\sum_{i=1}^{k}+\sum_{i=1}^{k}b_{ii}x_i^2+\sum_{i=1}^{k}\sum_{j=i}^{k}b_{ij}x_i x_j \tag{54}$$

Assume again that a full 2^k factorial design is used. For this problem the alias matrix is such that one obtains

$$\begin{aligned}\hat{b}_o&=b_o+\sum_{i=1}^{k}b_{ii}\\ \hat{b}_j&=b_j,\quad j=1,k\end{aligned} \tag{55}$$

Thus only $\hat{b}_o$ is biased with the other coefficients unbiased or uncorrelated.

3.3.3 Orthogonal Designs for Linear Approximations

For a problem with r design variables, a full 2^r factorial design is an orthogonal design if the approximating function is a first order polynomial. There are several advantages in using such an orthogonal design when the approximating function is assumed to be linear. These advantages are:

1. The solution for the coefficients of the polynomial approximation require a matrix inverse (see Equation (12)). However, when the design is an orthogonal design, that inverse is very easily obtained using Equation (40). Thus there is a small computational advantage in using an orthogonal design.

2. Examples 3.3.2.1 and 3.3.2.2 indicate that under certain conditions, the coefficients obtained using an orthogonal design are unbiased. Obtaining unbiased coefficients is probably more important in developing response surface from experimental results than when developing response surfaces when results are from a deterministic model. With experimental studies, it may be important to ascertain the unbiased values of the linear coefficients. For the deterministic model however, one is looking for an approximating function which gives a good approximation throughout a region of interest. Whether the coefficients of the polynomial approximation are biased or unbiased is of little concern.

3. It can be proven that for linear polynomial approximations, an orthogonal design gives the minimum variance of the coefficients (see page 109 of [3]). It is important when modeling experimental results to obtain a model that is not overly sensitive to experimental error and thus there is an advantage in having a minimum variance of the coefficients.

However, for response surfaces of a deterministic model, variance of the coefficients is not relevant.

3.3.4 Orthogonal Designs for 2nd Order Polynomial Approximations

It is not possible to find an orthogonal design when using a second order polynomial approximating function of the form of Equation (8) (see page 107 of [2]). However, an orthogonal design can be found if one uses as the approximating function a second order orthogonal polynomial (page 130 of [3])

$$\hat{y}=b_o+\sum_{i=1}^{k} b_i x_i+\sum_{i=1}^{k} b_{ii}(x_i^2-\bar{x}_i^2)+\sum_{i=1}^{k}\sum_{j=i}^{k} b_{ij} x_i x_j \tag{56}$$

where

$$\bar{x}_j^2=\frac{\sum_{u=1}^{N} x_{j_u}^2}{N} \tag{57}$$

and where

$$\begin{aligned}&N\text{=the number of design points and}\\&x_{j_u}=x_j \text{ for each of the design points.}\end{aligned} \tag{58}$$

The use of an orthogonal design still gives the small computational advantage that the inverse shown in Equation (12) is an inverse of a diagonal matrix. However, when using second order approximations, it is not clear under what conditions one obtains unbiased coefficients. Also it can not be proven that orthogonal designs any longer give a minimum variance of the coefficients. Thus most of the reasons for using orthogonal designs found for linear approximations are not present when using second order approximations.

3.3.5 General Discussion of Orthogonal Designs

Orthogonal designs offer a small computational advantage that the matrix inverse required in solving for the coefficients of the polynomial approximating function is an inverse of a diagonal matrix. When approximating a deterministic model, properties of orthogonal designs which minimize the variance of the coefficients and which give unbiased coefficients are unimportant. For this case, the use of orthogonal designs can only be justified by how well they perform on test problems. Such test problems are presented later in this report.

3.4 Central Composite Designs--Designs for Fitting Second Order Models

It was shown in Section 2 that at least 3 levels of the design variables are required if one is to make a second order approximation. A workable alternative to using a 3^k factorial design is a class of designs called the central composite design. These types of designs are widely used by workers applying second order response surface techniques [3].

3.4.1 Format of the central composite design

The central composite design is a design composed of the 2^k factorial design augmented by additional points. The augmented design points are as follows:

$$
\begin{array}{ccccc}
x_1 & x_2 & x_3 & \cdots & x_k \\
0 & 0 & 0 & \cdots & 0 \\
-\alpha & 0 & 0 & \cdots & 0 \\
\alpha & 0 & 0 & \cdots & 0 \\
0 & -\alpha & 0 & \cdots & 0 \\
0 & \alpha & 0 & \cdots & 0 \\
\cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & \cdots & -\alpha \\
0 & 0 & 0 & \cdots & \alpha
\end{array}
\tag{59}
$$

Figure 7 shows a central composite design for k=3. The value of α and the number of design points at the center of the design are varied to meet certain conditions. In the following, those conditions are chosen assuming that the approximating polynomial function is given by Equation (56).

3.4.1.1 Single center point rotatable second order experimental designs:

A design is said to be rotatable when the variance of the estimated response--that is, the variance of $\hat{y}$, which in general is a function of position in the design space, is instead only a function of the distance from the center of the design and not on the direction. In other words, a rotatable design is one for which the quality of the estimator $\hat{y}$ is the same for two points that are the same distance from the center of the design [3]. It is possible to develop central composite designs which have a single center point. The value of α which will yield these rotatable second order designs are given in Table 3.2.

Table 3.2. Value of α for single center point rotatable central composite designs

k	α
2	1.414
3	1.682
4	2.000
5	2.378
5 (1/2 rep)	2.000
6	2.828
6 (1/2 rep)	2.378
7	3.364
7 (1/2 rep)	2.828
8	4.000
8 (1/2 rep)	3.364

Note in Table 3.2 that a rotatable second order experimental design can be obtained with a fractional factorial design augmented with additional design points as well as with a augmented full factorial design.

3.4.1.2 Multiple center point rotatable uniform precision designs:

In general, the variance of $\hat{y}$ varies with distance from the center of the design. However, by varying the number of center points, N, the variance at a distance of unity from the center can be made approximately equal to the variance at the center of the design. Such designs are referred to as uniform precision designs. The uniform precision design is based on the philosophy that in the central region of the design space there should be uniform importance as far as the variance of response is concerned, as opposed to, for example, a situation in which the variance is low in the center of the design but increases drastically as one moves away from the design center [3]. The number of center points, m, and the value of α can be varied so as to obtain a rotatable uniform precision designs. Table 3.3 gives those values.

Table 3.3. Values of m and α for multiple center point rotatable uniform precision designs

k	m	α
2	5	1.414
3	6	1.682
4	7	2.000
5	10	2.378
5 (1/2 rep)	6	2.000
6	15	2.828
6 (1/2 rep)	9	2.378
7 (1/2 rep)	14	2.828
8 (1/2 rep)	20	3.364

3.4.1.3 Single center point orthogonal central composite designs:

An orthogonal central composite design can be developed where $[Z]^t[Z]$ is diagonal. To obtain a design of this type a single center point can be used and the α value are taken from Table 3.4.

Table 3.4. Values of α for single center point orthogonal central composite designs

k	α
2	1.000
3	1.216
4	1.414
5	1.596
6	1.761
7	1.910
8	2.045

3.4.1.4 Rotatable orthogonal designs:

By varying the number of designs at the design center, m, and by selecting appropriate values for α, an orthogonal rotatable central composite design can be obtained. Values of m and α for such a design are given in Table 3.5.

Table 3.5. The value of m and α for multiple center point orthogonal rotatable central composite designs

k	m	α
2	8	1.414
3	9	1.682
4	12	2.000
5	17	2.378
5 (1/2 rep)	10	2.000
6	24	2.828
6 (1/2 rep)	15	2.378
7 (1/2 rep)	22	2.828
8 (1/2 rep)	33	3.364

3.4.2 Discussion of the central composite design

Orthogonal central composite designs have been shown to give a variance of response comparable to that obtained with a full 3^k factorial design. Thus, their use is justified when one has experimental error in the response function. Rotatable and uniform precision designs attempt to control the response variance. Thus there use is also justified when one has experimental error in the response function. However, when building a response surface for a deterministic model where there is no experimental error in the response function, their use is justified only by how well they perform of trial problems. Likewise, the designs were developed for the approximating function of Equation (56). If a different second order polynomial approximating function such as in Equation (8) were used or if a neural net was used to develop the response surface, then again the justification for the use of the various central composite designs would have to be based on their performance on trial problems. Performance of various central composite designs on trial problems is next reported.

3.4.3 Example -- Fox's Banana Function

Fox investigated in Reference [16] a function

$$y=10x_1^4-20x_2x_1^2+10x_2^2+x_1^2-2x_1+5 \tag{60}$$

which has banana shaped contours as seen in Figure 8. The region of interest to be considered is ($-1.5<x_1<1.5$, $-.5<x_2<2.0$).

A second order polynomial approximation is to be made of this function using an orthogonal polynomial approximation as in Equation (56). A two variable orthogonal polynomial approximation is of the form

$$\hat{y}=b_o+b_1x_1+b_2x_2+b_{11}(x_1^2-\overline{x}_1^2)+b_{22}(x_2^2-\overline{x}_2^2)+b_{12}x_1x_2 \tag{61}$$

where

$$\overline{x}_j^2=\frac{\sum_{u=1}^{N}x_{j_u}^2}{N} \tag{62}$$

and where

$$N\text{=the number of design points and}$$
$$x_{j_u}=x_j \text{ at the design points} \tag{63}$$

In the first phase of this example, Fox's function was approximated using the second order orthogonal polynomial of Equation (61). The designs used in making the approximation were

1. a full 5^2 factorial design,
2. a full 3^2 factorial design,
3. single center point rotatable central composite design,
4. multiple center point rotatable uniform precision central composite design,
5. single center point orthogonal central composite design,
6. multiple center point rotatable orthogonal central composite design,
7. minimum point design from program DESIGNS,

8-10. minimum point design from program DESIGNS augmented by additional randomly selected design points, and

11. nine randomly selected design points.

Once an approximation was obtained, the approximate function was evaluated at a 31 x 31 grid of points over the region of interest. The approximate function values at these 961 points were used to develop the error parameter v_G from Equation (6). Because there are a differing number of functional evaluations required for each of the sundry designs tested, a comparison of the designs based on v_G is misleading. For example, the full 5^2 factorial design has 25 design points each requiring a functional evaluation where as the multiple center point rotatable orthogonal central composite design has but 16 design points requiring 9 functional evaluations (in the following it is assumed that the function being approximated has no experimental or numerical error and thus the 8 design points at the design center require but one functional evaluation). Thus a comparison of performance based only on quality of fit is not a fair comparison. The 5^2 factorial might do a better job of approximating a function but the computational cost of the 25-9=16 extra functional evaluations might make it a less desirable design.

For each design, design j, a measure of efficiency, E_j, was developed where

$$E_j = \frac{(v_G)_{design\ j}\ T_{design\ j}}{(v_G)_{design\ 1}\ T_{design\ 1}} \tag{64}$$

where T is the number of functional evaluations required for a given design.

The efficiency of all the designs was compared to design 1, the 5^2 factorial design. Table 3.6 gives, for each design tested, the number of design points, N; for central composite designs, the number of design points at the center of the design, m; the number of functional evaluations required, T; the value of v; the value of v_G; and the value of E_j.

Table 3.6. Performance of various designs on Fox's Banana Function, orthogonal polynomial approximating function, $-1.5 < x_1 < 1.5$, $-.5 < x_2 < 2.0$

Design	N	m	T	v	v_G	E_i
5^2 factorial design	25	...	25	70.76	78.92	1.00
3^2 factorial design	9	...	9	64.07	102.46	.47
single center point rotatable central composite design	9	1	9	54.36	77.34	.35
multiple center point rotatable uniform precision central composite design	13	5	9	53.08	77.34	.35
single center point orthogonal central composite design	9	1	9	64.07	102.46	.47
multiple center point rotatable orthogonal central composite design	16	8	9	51.62	77.34	.35
minimum point design from program DESIGNS	6	...	6	0	162.62	.49
minimum point design from program DESIGNS augmented by 2 randomly selected design points	8	...	8	43.27	105.16	.43
minimum point design from program DESIGNS augmented by 3 randomly selected design points	9	...	9	53.53	88.63	.40
minimum point design from program DESIGNS augmented by 4 randomly selected design points	10	...	10	53.05	86.44	.44
random--9 points	9	...	9	21.05	460.96	2.10

Several items can be noted in Table 3.6:

1. The design composed of 9 randomly selected design points did poorly. Even though the design points were chosen randomly, it turned out that the design points were not well scattered in the design space but were heavily concentrated in one quadrant of the design space. The polynomial approximation fitted the function well at the design points but poorly over the region of interest.

2. The value of v_G was approximately the same for the single center point rotatable central composite design, the multiple center point rotatable uniform precision central composite design, and the multiple center point rotatable orthogonal central composite design. These three designs differ only in the number of design points at the center of the design space. These designs have 1, 5, and 8 designs at the center, respectively. The effect of putting more designs at the center is to translate the response surface toward the center response. For this problem, however, the actual and approximated response were very close at the design center point, even for only 1 design point at the center. Thus, adding more design points at the design center did little to translate the response surface and thus did not material effect the value of v_G.

3. The eleven designs of Table 3.5 were next used to build an approximation using the standard second order polynomial approximation of Equation (8) instead of the orthogonal polynomial approximation of Equation (61). Results identical to those of Table 3.5 were found. The type of approximating polynomial may effect variances but does not affect quality of fit at the design points or over the region of interest. For those problems were there is no experimental or numerical error associated with functional evaluations, one is

not interested in variance. Thus, there is little advantage in using the orthogonal polynomial approximating functions over a standard second order polynomial function.

4. Based on efficiency, the single center point rotatable central composite design, the rotatable uniform precision central composite design, and the rotatable orthogonal central composite design performed the best but none of the designs gave a good approximation over the region of interest. Over a small region of interest, one could expect that a second order polynomial approximation could well approximate the given function. Obviously, here the region of interest is too large for a second order approximation to be a good one. Thus a smaller region of interest was chosen, $-.5 < x_1, .5$, $-.5 < x_2 < .5$. Table 3.7 compares the eleven designs using this region of interest. Notice that over this smaller region of interest, all the designs gave a much better approximation to the function.

5. For the smaller region of interest, based on efficiency, the 3^2 factorial design, the single center point orthogonal central composite design, and the augmented minimum point designs performed the best. Obviously, the optimum choice of design is problem dependent. However, all designs except the randomly selected design performed much better than the 5^2 factorial design.

Table 3.7. Performance of various designs on Fox's Banana Function, orthogonal polynomial approximating function, $-.5<x_1<.5$, $-.5<x_2<.5$

Design	N	m	T	v	v_G	E_j
5^2 factorial design	25	...	25	11.16	8.57	1.00
3^2 factorial design	9	...	9	13.27	10.95	.46
single center point rotatable central composite design	9	1	9	6.58	14.74	.62
multiple center point rotatable uniform precision central composite design	13	5	9	5.88	14.74	.62
single center point orthogonal central composite design	9	1	9	13.27	10.95	.46
multiple center point rotatable orthogonal central composite design	16	8	9	5.47	14.74	.62
minimum point design from program DESIGNS	6	...	6	0	18.66	.52
minimum point design from program DESIGNS augmented by 2 randomly selected design points	8	...	8	5.74	11.82	.44
minimum point design from program DESIGNS augmented by 3 randomly selected design points	9	...	9	6.45	10.53	.44
minimum point design from program DESIGNS augmented by 4 randomly selected design points	10	...	10	6.33	10.29	.48
random--9 points	9	...	9	2.42	47.22	1.98

3.4.4 Conclusion

Second order polynomial approximations or neural net equivalents are often adequate for building response surfaces, especially if the region of interest is small. Central composite designs are convenient for building the second order approximations. They provide the necessary information for determining all of the coefficients of the approximating polynomial and give a good distribution of points in the design space. The approximating function can be made to closely fit the exact function at the design center by using multiple center points. When modeling deterministic systems, each functional evaluation at the design center yields the same function value. Thus, for deterministic models, only one functional evaluation need be performed at the center point even when multiple center points are used. Table 3.8 gives information relevant to central composite designs for various number of design variables, k. Central composite designs give over-determined second order polynomial approximations. In other words, there are more design points in the design than there are undetermined coefficients in a second order polynomial approximation. Table 3.8 also gives the percentage that the approximation is over-determined. Previous studies [19] have indicated that designs which give approximations that are around 20-50% over-determined tend to be efficient designs. One can see that the central composite designs are reasonable for $k<6$. For larger k values, too many design points are being used by the central composite designs. For $k>5$, an augmented minimum point design is a better choice.

Table 3.8. Information relevant to central composite designs for various number of design variables

Number of design variables, k	Number of coefficients in a 2nd order polynomial approximation	Number of functional evaluations required with a central composite design	% over-determined
1	3	4	33
2	6	8	33
3	10	14	40
4	15	24	60
5	21	42	50
6	28	76	171
7	36	142	294
8	45	272	504

4. Optimality Criteria

4.1 D, A, E, G, and V Optimality Criteria

It was pointed out in Section 3 that even for a small number of factors, a complete factorial experiment become impractical if functional evaluations are computationally or experimentally expensive to obtain and thus one is forced to use some sub-set of the factorial design or an alternate design requiring fewer experiments. Section 3 shows that the variances of the coefficients of a polynomial approximation and the variance of the predicted response involve the matrix [H] given in Equation (33) and repeated here:

$$[H]=([Z]^t[Z])^{-1} \tag{65}$$

Schoofs [22] lists five criteria for selecting a sub-set of the factorial designs. These criteria involve the matrix [H]. The criteria, referred to as optimality criteria, attempt to make [H] minimal. However, "the minimum of a matrix is not a well defined concept and a number of operational criteria have been developed" [22]. The optimality criteria for selecting a subset of a full factorial design can be based on selecting the subset satisfying the following criteria:

1. D-optimality, which is achieved if the determinant of [H] is minimal which in term gives that the product of the eigenvalues of [H] is minimal.
2. A-optimality, which is achieved if the trace of [H] is minimal which in term gives that the sum of the eigenvalues of [H] is minimal.
3. E-optimality, which is achieved if the largest eigenvalue of [H] is minimal.

4. G-optimality, which is achieved if the maximum over all candidate points of the estimated response variance is minimal.

5. V-optimality, which is achieved if the estimated response variance, averaged over all candidate points is minimal.

4.1.1 Criteria Applied to a One Dimensional Example

An example is considered here to compare the performance of the 5 optimality criteria. The following test function of one variable was considered:

$$y=2+x+\sin[\frac{3\pi}{2}(x+1)], \quad -1\le x\le 1 \tag{66}$$

This function was approximated with polynomials of order 1-4. The approximations shown in Figure 9 were developed using 13 designs, uniformly spaced in the region of interest. These approximations were then used to generate the functional values at 61 uniformly spaced points in the region of interest which were used to plot the curves of Figure 9.

Further approximations of Equation (66) were developed using various number of design points, n. The designs selected were

1. uniformly spaced design points, n=5,7,9,11,13;
2. randomly selected design points, n=5,7,8,11,13;
3. an n member subset of the 13 uniformly spaced design points, n=5,7,9,11.

Under item 3, the subset of design points was chosen using:

1. D-optimality,
2. A-optimality,
3. E-optimality,
4. G-optimality, and
5. V-optimality.

A FORTRAN program was written to perform the investigation under item 3. The demanding part of the programming was to identify all the possible subsets from the set of thirteen design points. After developing a procedure to identify all combinations, each subset was used to build the [H] matrix. The "optimal" [H] matrix was then determined using the five optimality criteria. The coefficient v_G was then computed for the optimal subset. Figures 10-13 show the value of v_G for the D, A, E, and G optimality criteria when a first, second, third, and fourth order approximation is being made, respectively, versus the number of design points specified in the subset . Also shown in those figures is the value of v_G for designs consisting of design points uniformly spaced in the region of interest.

It was found that for all subsets of size r from a design point set of size n that the estimated response variance, averaged over all candidate points, was invariant. This finding undoubtedly could also be proven theoretically but such a proof was not attempted. From this example, one can conclude that <u>the V optimality criteria, which employees the estimated average response variance, is not a viable criteria for selecting a subset of design points from</u>

a given set. From Figures 10-13, one can see that in most cases there is little difference in the performance of the various optimality criteria with criteria D and G performing slightly better than the other two criteria. As can be seen in Figure 12, on one occasion (when using a third order polynomial approximation and when selecting a subset of 5 design points from the 13 design point set) the G optimality criteria performed poorly while the D criteria did not. Thus, this example indicates that the D optimality criteria may be the criteria of choice. There is a further advantage in using the D optimality criteria. The requirement that the determinant of [H] is minimal is equivalent to a requirement that the determinant of [G] is maximal where

$$[G]=[Z]^t[Z] \tag{67}$$

Thus the D optimality criteria insures that the procedure for determining polynomial coefficients in Equation (12) will be well defined. In other words, Equation (12) uses the inverse of [G]. The D optimality criteria guarantees that [G] is not singular.

One can see in Figures 10-13 that, in most cases, all the optimality criteria performed worst than the uniformly spaced design case. This example indicates that a design picked using an optimality criteria may be no better than a design of the same size in which the design points are uniformly located in the design space.

4.2 S and Q Optimality Criteria

The previous optimality criteria involved only the matrix [H] and did not consider the function to be approximated. Thus for a given number of design variables and level of approximation, the same designs would be selected no matter what the nature of the function to be approximated. Initially it was thought that a superior optimality criteria would have to consider the nature of the function. Thus two additional optimality criteria were examined:

1. S-optimality, which is achieved if the average error of approximation at the design points is minimal and

2. Q-optimality, which is achieved if the maximum error of approximation at the design points is minimal.

Here

$$\textit{average error of approximation} = \frac{\sum_{1}^{r}(y_i - \hat{y}_i)^2}{r} \tag{68}$$

and

$$\textit{maximum error of approximation} = \max\ (y_i - \hat{y}_i)^2, \quad i = 1,..,r \tag{69}$$

where r is the size of the subset of design points to be selected. One can see that with the S and Q optimality criteria, the function to be approximated effects the design points selected.

4.2.2 Criteria Applied to a One Dimensional Example

The one dimensional example problem of Section 4.1.1 was then re-examined. Figures 14-17 show values of v_G using the S and Q optimality criteria and using a first, second, third, and fourth order polynomial approximation, respectively, versus size of the subset of design points. Also shown in these figures are results for uniformly spaced design points. One can see in these figures that terrible approximations were obtained with these criteria when only small subsets of design points were selected from the original set. Figures 18-20 indicate why such bad approximations are obtained with these two criteria.

Figure 18 depicts results obtained by having eleven designs points selected, using the Q optimality criteria, from a set of 13 design points. The Q optimality criteria finds an approximation such that the maximum error of the approximation over eleven design points is minimal. One can see in Figure 18 that the approximating function did indeed well fit the exact function at the 11 design points selected. However, the approximating function did a poor job of approximation at the ends of the region of interest and thus would not yield a low value of v_G. Figure 19 is similar to Figure 18 except that this figure depicts results obtained by having 7 design points selected from the set of 13 design points. One can see that for the optimum design selected, there is an almost perfect approximation at the design points selected but over a much larger region the approximation is poor and thus a large value of v_G would be obtained. In Figure 20, only 5 design points are selected. Again at those design points, an almost perfect approximation is obtained but a terrible approximation is obtained over a large part of the region of interest and thus a large v_G

would be obtained. Thus we can conclude that the S and Q optimality criteria are not operative.

4.3 An Alternate Approach--Random Selection of Designs

The effect of randomly picking design points was next considered. Here designs are picked in the region of interest using a random number generator.

4.3.1 Random Selection of Designs Applied to a One Dimensional Example

For the one dimensional problem under consideration, first, second, third, and fourth order approximations were considered. Design point sets containing 5,7,9,11, and 13 design points were developed by randomly picking design points in the region of interest using a random number generator. Approximations were developed using the design sets. Results using these approximations are compared in Figures 21-24 to results using uniformly spaced design points. One can see in these figures that most of the time results from randomly picked design points are either as good as or not much worst than results from uniformly spaced design points. However, on two occasions, when the number of design points in the design set was small, a relatively poor approximation was obtained. Obviously where one is picking only a small number of points using a random number generator, there is a chance that a bad set of points can be generated and indeed on these two occasion a poor selection of points was made. In general however, when more design points are randomly selected, those points should be scattered throughout the design space and good approximations should be obtained. In conclusion, randomly selecting design points may be a viable method of design selection.

4.4 Larger Problems

Consider a problem in two variables and consider that the potential design points will be taken from a 6 x 6 grid of points. Let

r = total number of design points in the set of potential design points,

c = number of design points in the selected subset of design points,

nc = the number of different combinations of designs in the subset.

For the problem at hand, r=36. Subset sizes of c=15, 20, 25, and 30 are to be considered. The number of possible combinations of design points in the subset, nc, is given by

$$nc = \frac{r!}{(r-c)!\ c!} \tag{70}$$

Table 4.1 summarizes the number of combinations for this study.

Table 4.1 Number of combinations of designs in a two variable study

r Total number of design points	c Number of point in subset	nc Number of combinations
36	15	5,567,902,560
36	20	7,307,872,110
36	25	600,805,296
36	30	1,947,792

One can see that for even small problems, it is infeasible to examine all possible combinations of subsets of size N from a given set of design points. Welch [23], instead of evaluating all possible N-point designs, developed a "branch and bound" algorithm which guarantees global D-optimal designs but which does not generate and evaluate all possible designs. However, even here the computing costs are high. Fedorov [24] developed another technique which neglects the integer character of the components of the design set and obtains a discrete design which is rounded off to an exact design. Reference [22] reports that these designs are considered only approximate. The most popular algorithm seems to be DETMAX by Mitchell [25]. Quoting reference [22], "The algorithm starts with an initial m-point ED (experimental design); the final goal is an optimal N-point ED. During each iteration step that candidate point, which results in the largest increase of det(M), is added to the design, and subsequently that point, which results in the smallest decrease of det(M), is removed from the design. The number m of points in the initial design may be larger or smaller than N. If necessary the algorithm first adds (if $m<N$) or rejects (if $m>N$) points until the number of points in the ED is equal to N. In order to avoid local optima the algorithm is able to perform 'excursions', in which several points are added at one go and subsequently the number of points is reduced to N. If the resulting N-point ED has not been improved, another excursion will be made from the same initial design. If the excursion is successful the resulting ED will be used as starting ED in a further attempt to maximize det(M). The algorithm terminates when, after several excursions, no better ED is found. The algorithm generates high quality EDs against relatively low computing costs."

An attempt is being made to obtain the algorithm DETMAX.

4.5 Optimality Criteria Based on Minimizing Uncertainty

Reference [21] considers problems where there is no experimental error. That reference uses an optimality criteria based on selecting a design which minimizes the uncertainty in the approximating function. That reference was given mixed reviews by a number of leading authorities in the field [21] (reviews follow the paper). The formulation is quite theoretical and difficult to follow. The formulation seems to have promise but requires additional theoretical development before it becomes operative.

4.6 Conclusion

There is little rational for using any of the investigated optimality criteria when building approximations of functions which contain no experimental error. However, the D-optimality criteria can conveniently be used as a heuristic in selecting design points.

Previous investigations have indicated that approximations should be over-determined. That is to say that more training pairs should be used to build an approximations than the number of associated undetermined parameters. It has been suggested that a 20-50% over-determined system might be reasonable. The program DESIGNS, described in Section 2, develops enough designs to exactly determine a quadratic approximation of a given function. The D-optimality criteria can be used as a heuristic for selecting design points to supplement those generated by DESIGNS. The use of the D-optimality criteria to select the supplementary points would guarantee than no singular matrices would be encountered in determining the undetermined parameters associated with the polynomial approximation.

5. Significance Testing of Coefficients

5.1 Introduction

When the training pairs used to build a polynomial response surface contain experimental or numerical error, certain coefficients in the polynomial approximation may not be significant. In other words, even though one calculates a value for some coefficient, b_i, the experimental or numerical error may have such an effect on that coefficient that it could just as well be taken as zero as the value calculated. In situations like this, it may be advantageous to drop that term from the polynomial approximation and redevelop the response surface. Such a procedure is discussed in pages 34-38 of [3] and an automated procedure for performing such an operation was developed in [26]. Testing of significance involves the t-test which is next described.

5.2 t-test

Coefficients of the polynomial approximation are found from Equation (12). The determination of those coefficients involve the matrix [H] where

$$[H]=([Z]^t[Z])^{-1} \tag{71}$$

A number of terms must now be defined:

$$mean\ square\ error=MSE=\frac{\sum_{i=1}^{N}(y_i-\hat{y}_i)^2}{N-m} \tag{72}$$

$$standard\ error\ coefficient = se_i = \sqrt{MSE\ H_{ii}} \tag{73}$$

$$t_i = \left| \frac{b_i}{se_i} \right| \tag{74}$$

where

N = the number of design points and

m = the number of coefficients in the polynomial approximation.

In making the test of significance, t_i from Equation (74) is compared to tabulated values of t_a. The value of t_a is taken from a table of "Percentage Points of the Student's t Distribution" [3]. The value taken depends on the level of significance desired. In lieu of using tabulated values, t_a is often taken as four [26]. If t_i is less than t_a ($t_i < t_a$), then that coefficient's importance in approximating the response is deemed to be insignificant and therefore may be eliminated from the response function.

The primary focus of this study was to examine methods of developing good response surfaces for deterministic models, i.e. for systems that contain no experimental or numerical error. Statistical testing of coefficients presupposes experimental or numerical error and thus is not relevant when approximating response which contains no error. However, the method was thought to perhaps offer a heuristic for improving the quality of a response surface even if experimental or numerical errors are not present. Thus, two examples were examined. Results are next reported.

5.3 Example 1 -- Fox's Banana Function

Example 1 again examines Fox's Banana Function [16]. A complete second order polynomial approximation (m=6) and a complete third order polynomial approximation (m=10) were developed. These approximations were developed using a complete 6^2 factorial design (N=36). A **t-value**, t_i, was calculated for each parameter, b_i, and compared to t_a = 4. Parameter that lack significance ($t_i < t_a$) were eliminated. A new approximation was then developed using only the significant parameters. The values of v and v_G from Equations (5) and (6), respectively , were developed for the complete polynomial and for the polynomial containing only terms deemed significant. Results are shown in Figures 25 and 26. On can see in these figures that eliminating coefficients deemed insignificant had an adverse effect on the quality of the approximation over the region of interest.

5.4 Example 2

The effect of eliminating coefficients deemed insignificant was tested on the function

$$Y=(4+x_1)^3+\sin[\frac{3\pi}{2}(x_1+1)]+2+x_2^4+\sin(\frac{\pi}{2})+7x_2x_1 \tag{75}$$

Again, a complete second order polynomial approximation (m=6) and a complete third order polynomial approximation (m=10) were developed. These approximations were developed using a complete 6^2 factorial design (N=36). A **t-value**, t_i, was calculated for each parameter, b_i, and compared to $t_a = 4$. Parameter that lack significance ($t_i < t_a$) were eliminated. A new approximation was then developed using only the significant parameters. The values of v and v_G from Equations (5) and (6), respectively , were developed for the complete polynomial and for the polynomial containing only terms deemed significant. Results are shown in Figures 27 and 28. On can see in these figures that eliminating coefficients deemed insignificant offered no improvement in the quality of the response surface.

5.5 Conclusion

The applicability of significance testing of polynomial coefficients when modeling deterministic systems was considered. Two examples were examined to see if eliminating terms of polynomial approximations which were deemed to be insignificant by the t-test would improve the quality of the response surfaces developed. Based on these two examples, it was concluded that no improvement in the predictive capability of response surfaces over regions of interest would be obtained with such a procedure. The relevance of significance testing is when modeling systems containing numerical or experimental error.

6. Applicability of the Response Surface Technique

6.1 Introduction

The following study was performed to ascertain under what circumstances could the response surface technique be used to advantage in engineering optimization application. In this regard, assume that a quadratic polynomial approximations is to be made of functions of n variables. The number of undetermined coefficients in that approximation is:

$$\textit{number of coefficients} = \frac{(n+1)(n+2)}{2} \tag{76}$$

Previous studies [19] have shown that the best approximations are obtained when the approximations are over-determined. Thus, the number of functional evaluations required to make the approximation is:

$$\textit{number of functional evaluations} = \frac{\delta(n+1)(n+2}{2} \tag{77}$$

where δ determines the degree that the approximation is over-determined.

The functional evaluations required to build the approximation are initially performed before the start of the optimization process. By using parallel processing, these functional evaluations may be less computationally expensive than evaluations made sequentially in a direct optimization procedure. The number of required evaluations of Equation (77) is then equivalent to a reduced number of sequential evaluations thus:

$$equalivent\ number\ functional\ evaluations = \frac{\delta\,\beta(n+1)(n+2)}{2} \tag{78}$$

where β is a coefficient of efficiency associated with parallel processing.

An optimum solution can be attempted using the response surfaces developed instead of the original functions. However, because of the inexact nature of the approximations, a new set of response surfaces may have to be developed at the most recent approximate solution and another optimal solution attempted. This procedure may have to be repeated α times to reach the optimum solution for the original problem. The total number of equivalent functional evaluations performed in reaching this optimum is:

$$total\ equivalent\ functional\ evaluations = \frac{\alpha\,\beta\,\delta(n+1)(n+2)}{2} \tag{79}$$

If the solutions was attempted by direct optimization techniques instead of using response surfaces, Barthelemy [27] states that a solution can be obtained in most cases using no more than ψ first derivative evaluations. If the first derivatives are obtained by finite difference formulae, an estimate of the number of functional evaluations required by a direct solution procedure is:

$$functional\ evaluations\ direct\ methods \approx \psi(n+1) \tag{80}$$

If the response surface technique is to be competitive with the direct solution technique, then from Equations (4) and (5) one must have:

$$\frac{\alpha\beta\delta(n+1)(n+2)}{2} \leq \gamma\psi(n+1) \tag{81}$$

where γ is a convenience factor associated with using response surfaces. In other words, an investigator may tolerate more functional evaluations with the response surface technique than with the direct solution procedure just for the convenience of using response surfaces. Rearranging Equation (81) gives

$$[n+1][\frac{\alpha\beta\delta(n+2)}{2\gamma} - \psi] \leq 0 \tag{82}$$

Since (n+1) is positive, one obtains

$$\frac{\alpha\beta\delta(n+2)}{2\gamma} - \psi \leq 0 \tag{83}$$

or

$$n \leq \frac{2\psi\gamma}{\alpha\beta\delta} - 2 \tag{84}$$

In review

$$\begin{array}{l} \alpha = \textit{number sequential optimizations} \\ \beta = \textit{parallel processing coefficient} \\ \delta = \textit{overdetermined coefficient} \\ \gamma = \textit{convenience coefficient} \\ \psi = \textit{direct solution coefficient} \end{array} \tag{85}$$

Reasonable ranges of the parameters involved are

$$\begin{array}{l} \alpha = 1.00 \rightarrow 4.00 \\ \beta = 0.10 \rightarrow 1.00 \\ \delta = 1.25 \rightarrow 1.75 \\ \gamma = 1.00 \rightarrow 3.00 \\ \psi = 6.00 \rightarrow 10.0 \end{array} \tag{86}$$

For an approximate upper bound on the number of design variable that could be economical used with the response surface technique take:

$$\begin{array}{l} \alpha = 1.00 \\ \beta = 0.10 \\ \delta = 1.25 \\ \gamma = 3.00 \\ \psi = 10.0 \end{array} \tag{87}$$

giving

$$n \leq 498 \tag{88}$$

Under the most unfavorable set of circumstances, that is:

$$\begin{aligned} \alpha &= 4.00 \\ \beta &= 1.00 \\ \delta &= 1.75 \\ \gamma &= 1.00 \\ \psi &= 6.00 \end{aligned} \tag{89}$$

one obtains

$$n \sim 0 \tag{90}$$

Thus depending upon the problem, one could use the response surface technique for n=0 to n=500 variables. Consider the following reasonable set of parameters

$$\begin{aligned} \alpha &= 3.00 \\ \beta &= 0.50 \\ \delta &= 1.25 \\ \gamma &= 1.50 \\ \psi &= 8.00 \end{aligned} \tag{91}$$

giving

$$n \leq 13 \tag{92}$$

Thus, it is reasonable to assume that the response surface technique could be used for up to 10-15 design variables.

6.2 Conclusion

Under the most advantageous circumstances, the response surface technique applied to engineering optimization application could be used for up to 500 design variables. Under the worst set of circumstances, it is entirely inappropriate. Under normally expected circumstances, this technique might be used to advantage for 10-15 design variables.

7. Additional Examples

7.1 Introduction

The next several examples examine the effect of design selection on the quality of approximations. In each case, a second order polynomial approximation is made of a trial function. Different number of design variables are considered in each example. Thus, for each example different designs are appropriate. In the first example, there are 4 design variables. When there are fewer than 6 design variables, central composite designs are a possible appropriate choice. Other choices are the 3^k factorial design, the minimum point design, the augmented minimum point design, or randomly selected design. All of these designs are considered in that example. In the second and third examples, there are 15 and 20 design variables, respectively. Here, the 3^k factorial design and central composite designs contain too many design points to be practical. For these examples, the minimum point design, the augmented minimum point design, and the randomly selected design are appropriate and are considered.

7.2 The 35 Bar Truss with 4 Design Variables

In many response surface applications, the function to be approximated is a relatively smooth function of the design variables which can be approximated with a lower order polynomial or an artificial neural net with only a few nodes on the hidden layer. A problem of this type is shown in Figure 29. In this example, all loads shown in Figure 29 are in kips, all members of the lower chord of the truss are assumed to have area, A_1, and all members of the upper chord to have area, A_2, all vertical and diagonal members to have area, A_3. The depth of the truss is H. A response surface is to be constructed for the stress in member BC in terms of the design variables, x_i thus

$$x_i = 1/A_i, \quad i=1,3$$
$$x_4 = .09375H - .4375 \tag{93}$$

The region of interest is

$$2\ in^2 \le A_i \le 8\ in^2$$
$$6\ ft \le H \le 10 ft \tag{94}$$

or in terms of the design variables

$$.125 \le x_i \le .5 \tag{95}$$

A number of designs were used to develop a second order polynomial approximation for the stress in member BC. Each approximation was then used to predict stress on a 5 x 5 x 5 x 5 grid of points. The predicted stress and the actual stress on these NG=625 grid of points were then used to develop v_G from Equation (6). The parameter v_G is a measure of the quality of the approximation over the region of interest.

The different designs examined required different numbers of functional evaluation. So as to get a measure of the quality of fit of the approximation over the region of interest which

Table 7.1 The 35 bar truss with 5 design variables, 2nd order polynomial approximation

Description	m	α	T	F	v (%)	v_G (%)	E_j
3^4 factorial design	...	...	81	81	3.34	2.41	1.00
single center point rotatable central composite design	1	2.000	25	25	0.66	2.67	0.34
multiple center point rotatable uniform precision central composite design	7	2.000	31	25	0.59	2.67	0.34
single center point orthogonal central composite design	1	1.414	25	25	1.47	2.37	0.30
multiple center point rotatable orthogonal central composite design	12	2.000	36	25	0.55	2.67	0.34
minimum point design from program DESIGNS	...	...	15	15	0.00	3.99	0.31
minimum point design from program DESIGNS augmented by 3 randomly selected design points	...	...	18	18	0.40	3.86	0.36
minimum point design from program DESIGNS augmented by 6 randomly selected design points	...	...	21	21	0.38	3.91	0.42
minimum point design from program DESIGNS augmented by 9 selected design points	...	...	24	24	0.41	3.77	0.46
randomly selected design	...	...	25	25	0.00	824.2	105

m = number of design points at the center of the design space
T = the total number of design points
F = the number of functional evaluations required
α = parameter which defines location of certain design points

takes into account the number of functional evaluations performed, the efficiency, E_j, from Equation (64) was developed for each design. Table 7.1 reports for each design considered, the efficiency, E_j, as well as other relevant information.

One can see in Table 7.1 that all the designs considered, except the randomly selected design, gave a good approximation over the region of interest. Randomly selected designs, which often work well, can sometimes suffer from the problem that the coefficient matrix used to solve for the approximation's associated parameters is poorly conditioned or that the design points selected are not well scattered throughout the design space. In either case, they can yield a poor approximation over the region of interest as in this example.

The 3^4 factorial design well approximated the trial function. However, because it uses so many design points its efficiency measure is poor and thus is not a design of choice. The single center point orthogonal central composite design and the minimum point design from program DESIGNS performed the best, based of their efficiency. However, excluding the randomly selected design and the 3^4 factorial design, all of the designs considered gave a low value of v_G and had approximately the same value of efficiency.

Under normal circumstances, information is not available to calculate v_G and one must use the parameter v as a measure of the quality of fit over the region of interest. However, the parameter v is only a measure of quality of fit over the region of interest if the approximation is over-determined. Thus, under normal circumstances one would not want to use the minimum point design. This example indicates, that for problems of the size of this example, that any of the central composite designs or the augmented minimum point designs would be appropriate.

7.2 The 35 bar truss with 15 design variables

This example again considers the 35 bar truss of Figure 29. In this example, H is 10 ft., the areas of the 14 bars of the top and bottom chords are A_i, $i=1,14$, and the area of the vertical and diagonal members is A_{15}. The design variables of the problem are taken as

$$x_i = 1/A_i, \quad i=1,15 \tag{96}$$

The region of interest is

$$2\ in^2 \le A_i \le 8\ in^2 \tag{97}$$

or in terms of the design variables

$$.125 \le x_i \le .5 \tag{98}$$

Response surfaces were developed for the stress in member BC using a 2nd order polynomial approximation. The approximation were developed using various designs. To test the quality of the approximations over the region of interest, the function and the approximations were evaluated at NG=500 randomly selected test points over the region of interest. That information was then used to calculate v_G from Equation (6). The random number generator used to develop design points uses, in generating its numbers, an initial seed parameter, IFLAG. A different value of IFLAG was used to generate the 500 test points than was used to generate random points in the randomly selected designs or in the augmented minimum point designs. Thus, the test set of points does not duplicate any of the design points in the designs considered. Results of this investigation are reported in Table 7.2.

One will notice in Table 7.2 that only minimum point designs, augmented minimum point designs, and randomly selected designs are considered. A 3^{15} factorial design contains over 14 million design points. Thus, the use of the 3^{15} factorial design is out of the question. For a problem in k design variables, the central composite design uses a 2^k factorial design augmented by $2k+1$ additional design points. Thus, such a single center point central composite design for this problem contains 32,799 design points. Here again, such a design is impractical. One can develop a central composite design by augmenting only a fraction of the 2^k factorial design. For this problem, a single center point central composite design using only a 1/4 fraction of the 2^{15} factorial design would contain 8,223 design points which is still an impractical design. Thus, for problems of the size of this example, only the minimum point designs, augmented minimum point designs, and randomly selected designs are of reasonable size.

We can see in Table 7.2 that all of the designs with the exception of the "randomly selected--exactly determined design" did a good job of approximating truss behavior. A singular matrix was encountered in Equation (10) for the randomly selected--exactly determined design. With completely randomly selected designs, there is always the possibility of having a poorly conditioned coefficient matrix [Z] in Equation (10) and indeed this occurred in this problem. However, there was no problem with matrix conditioning using randomly selected over-determined designs.

Table 7.2 The 35 bar truss with 15 design variables, 2nd order polynomial approximation

Description	F	v %	v_G %	E_j
minimum point design from program DESIGN-exactly determined	136	0	1.263	1.0
augmented minimum point design--20% over-determined	163	0.083	0.294	0.28
augmented minimum point design--40% over-determined	190	0.087	0.060	0.07
random selection--exactly determined	136	*	*	*
random selection--20% over-determined	163	0.003	0.029	0.03
random selection--40% over-determined	190	0.003	0.010	0.01

* singular coefficient matrix

The efficiency parameter, E_j, is calculated in Table 7.2 but it is rather a meaningless parameter for this problem because all the designs so well fit the exact function. In real life situations, one usually does not have available information for calculating v_G. Thus, the parameter v or like term must be used as a measure of the quality of the approximation. The parameter v is not a meaningful measure of the quality of fit over a region of interest unless the system is over-determined. Thus for this example, the design of choice would be either the 20% over-determined minimum point design or the 20% over-determined randomly selected design.

7.3 Analytical function--20 design variables

This example considers a problem with even more design variables. The function tested is:

$$y=1.+\sum_{i=1}^{20} x_i+\sum_{i=1}^{20}\sum_{j=i}^{20} x_i x_j+\sum_{i=1}^{20}\sum_{j=i}^{20} x_i^2 * x_j \tag{99}$$

A second order polynomial function was used to build the response surface approximating this function. The polynomial approximating function had 231 undetermined coefficients. Because of the large size of this problem, factorial designs and central composite designs are not appropriate. A minimum point design, augmented minimum point designs, and randomly selected designs were considered. Values of the test function and approximate function were evaluated at NG = 1000 randomly selected points and the parameter v_G was developed using this information. The measure of efficiency of the designs examined along with other relevant information is given in Table 7.3.

Table 7.3 Analytical function with 20 design variables, 2nd order polynomial approximation

Description	F	v %	v_G %	E_i
minimum point design from program DESIGN-exactly determined	231	0	88.93	1.0
augmented minimum point design--20% over-determined	277	5.83	49.82	0.67
augmented minimum point design--40% over-determined	323	9.58	18.03	0.28
random selection--exactly determined	231	*	*	*
random selection--20% over-determined	277	0.61	7.21	0.10
random selection--40% over-determined	323	0.46	1.20	0.02

* poorly conditioned coefficient matrix

Just as in Example 7.2, a exactly determined randomly selected design gave a poorly conditioned coefficient matrix. These examples indicate that randomly selected exactly determined designs should be avoided. The 40% over-determined randomly selected design did an excellent job of modeling the test function and was the most efficient design considered. It seems that on problems with a large number of design variables that randomly selected over-determined designs should be expected to work well.

7.4 Conclusion

The examples of this section have shown that design selection depends on the number of design variables. If the number of design variables is less than 6, appropriate designs are:

1. augmented minimum point designs
2. central composite designs
3. over-determined randomly selected designs.

When there are more than 6 design variables, the central composite designs contain too many design point for consideration. For more than 6 design variables, appropriate designs are then

1. augmented minimum point designs
2. over-determined randomly selected designs.

The example examined indicate that in all cases, over-determined designs should be used. They the most efficient designs. Also, when a design is over-determined the coefficient v can be used as a measure of the quality of the approximation over a region of interest. Being able to use v as a measure of the quality of fit over the region of interest is very important because, in general, information is not available to determined the parameter v_G.

8. Augmented Minimum Point Designs

8.1 Introduction

Design selection in the literature concentrates of linear or quadratic response surfaces. This study has also concentrated on quadratic approximations for several reasons:

1. linear approximations, in most instances, will be inadequate to model functions of interest,
2. for many problems, a 2nd order approximation will be adequate to model response especially if the region of interest is limited,
3. there is a scarcity of literature which address design selection for cubic or higher order polynomial approximations, and
4. in optimization process using response surfaces, for moderate size problems, it is more computationally efficient to perform a sequence of quadratic approximations than one cubic or higher order approximation. This fact is next discussed.

The number of terms in a second order polynomial in n design variables is

$$\textit{number terms quadratic} = (n+1) + \frac{n(n+1)}{2} \tag{100}$$

The number of terms in a 3rd order polynomial in n design variables is

$$number\ terms\ cubic = 1 + \frac{3}{2}n(n+1) + \frac{n!}{6(n-3)!} \tag{101}$$

Table 8.1 gives, for various number of design variables, the number of terms in a 2nd order and 3rd order polynomial and their ratio.

Table 8.1 Number of terms in a 2nd and 3rd order polynomial and their ratio

number of design variables, n	number of terms in quadratic	number of terms in cubic	cubic/quadratic
3	10	20	2
6	28	84	3
9	55	220	4
12	91	455	5
15	136	816	6

One can see that for problems with more than 6 design variables, it will probably be more computationally efficient in an optimization algorithm to utilize a sequence of quadratic response surfaces than one 3rd or higher order response surface. When there are 6 or fewer design variables, 3rd or 4th order response surfaces may be used to advantage.

In this report, the term "minimum point design" refers to a design that has just enough design points to allow the determination of coefficients of an approximating polynomial. The term "augmented minimum point design" is a minimum point design which contains additional design points. Thus, augmented minimum point designs are over-determined designs. The studies that have been performed in this report indicate that augmented minimum point designs are competitive with, if not better than, central composite designs for developing a 2nd order response surface. A program DESIGNS [20] was developed for generating augmented minimum point designs for developing a 2nd order response surface. That program is described in Section 8.2.

When there are 6 or fewer design variables, it may be computationally beneficial to use a 3rd order or 4th order response surface. Thus, the program DESIGN4 [28] was developed to generate augmented minimum point designs for a 4th order response surface. The program DESIGN4 is discussed in Section 8.3. The program can also be used to develop a 3rd order response surface. The 3rd order minimum point design is a subset of the 4th order minimum point design. Thus the 4th order minimum point design will give an over-determined 3rd order approximation. Additional randomly selected design points can be added to the 4th order minimum point design to give the desired degree that the 3rd order approximation is to be over-determined.

8.2 Augmented Minimum Point Designs for 2nd Order Approximations

The basic building block for program DESIGNS is the star pattern of design points. Figure 4 shows the star pattern for 3 design variables. This pattern of design points allows one to determine those coefficients of a 2nd order polynomial approximation associated with the terms

$$1,\ x_i,\ x_i^2,\quad i=1,n \tag{102}$$

To be able to determine the coefficients associated with the terms

$$x_i x_j,\quad i \neq j \tag{103}$$

one must supplement the star pattern with one additional design point in the x_i, x_j planes. Figure 30 shows the additional design point in the x_i, x_j plane. Figure 6 shows the total minimum point design for 3 design variables.

Studies of this report indicate that designs should be over-determined. Having a design that is 20-50% over-determined is a good compromise between keeping down the number of design points while still getting a good approximation. The program DESIGNS augments the minimum point design with a user selected number of random design points.

8.2.1 Specifics of program DESIGNS

A listing of the FORTRAN program DESIGNS is found in Appendix 1 and a copy of that program is found in file "designs.f" on the floppy disk accompanying this report. The program should be compiled with a F77 compiler with the compiled program called "design". To run the program just enter "design" from the keyboard. The program prompts the user for

1. the number of design variables,

2. the number of designs points to augment the minimum point design, and

3. a seed parameter, IFLAG, which is used to generate the random numbers (IFLAG can be entered as any positive integer).

The program then generates a design in local coordinates with the maximum range on each design variable of -1 to +1. The program then

4. asks the user to enter an integer which specifies whether design point coordinates are to be also generated in global coordinates. If they are to be calculated in global coordinates, the program then

5. prompts the user to enter the range of design variables in global coordinates.

Results with commentary are written to file "design.res". Design points without commentary are written to file "design.run".

8.3 Augmented Minimum Point Design for 3rd and 4th Order Approximation

A 3^k factorial design is used as the building block of this minimum point design. The 3^k factorial design provides information for calculating the coefficients associated with the terms

$$1,\ x_i,\ x_i x_j,\ x_i^2,\ x_i^2 x_j,\ x_i^2 x_j^2,\quad j \neq i \tag{104}$$

Additional points are then added at -1 and 1 (in local coordinates) along the x_i axis. These points together with the 3^k factorial design point give the star pattern which can be seen in Figure 31. With this arrangement of points, there are 5 design points along the x_i axis which provides information for calculating the coefficient associated with the terms

$$x_i^4 \tag{105}$$

Additional design points are then placed in each x_i, x_j plane which provides information for calculating the coefficient associated with the terms

$$x_i^3 x_j \tag{106}$$

These points are also shown in Figure 31.

8.3.1 Specifics of program DESIGN4

A listing of the FORTRAN program DESIGN4 is found in Appendix 2 and a copy of that program is found in file "design4.f" on the floppy disk accompanying this report. The program should be compiled with a F77 compiler with the compiled program called "design4". To run the program just enter "design4" from the keyboard. The program prompts the user for needed information. Prompts and response are similar to those for the program DESIGNS.

8.4 Conclusion

A minimum point design is a design that has just enough design points to allow the determination of the coefficients of an approximating polynomial. An augmented minimum point design is a minimum point design which contains additional design points. Augmented minimum point designs are competitive with, if not better than, central composite designs for developing a 2nd order response surface. Minimum point designs should be augmented with enough points that the approximation is 20-50% over-determined. A program DESIGNS was developed for generating augmented minimum point designs for developing a 2nd order response surface.

When there are more than 6 design variables, 3rd or higher order approximations require so many design points that it is computationally better to perform a sequence of 2nd order approximations in an optimization process than one higher order approximation. When there are 6 or fewer design variables, a 2nd order approximation may often be satisfactory. However, for those cases where it is desirable to use a higher order approximation, program DESIGN4 was developed. That program generates designs which can be used to develop 3rd or 4th order approximations.

9. Solution Algorithm

9.1 Introduction

In this investigation, the program NEWPSI was used to perform the studies involving polynomial approximations. That program can investigate under-determined, exactly-determined, or over-determined approximations of various orders. The version submitted with this report can handle up to 15 design variables as programmed. The order of polynomial it can handle is as follows:

1. one design variable, up to a 20th order polynomial
2. two design variables, up to a 5th order polynomial
3. for 2-15 design variables, a second order polynomials.

One can use up to 250 designs to train the approximation. In calculating v_G, it can handle up to 2000 grid points.

The program solves for the undetermined parameters associated with the approximation. It then evaluates the approximate function at the design points and calculates the error parameter, v. It then reads in the design points and function value on the test grid. The approximate function is evaluated at the grid points and the error parameter, v_G, is then evaluated.

9.2 Program Specifics

A listing of the FORTRAN program NEWPSI is found in Appendix 3 and a copy of that program is found in file "newpsi.f" on the floppy disk accompanying this report. The program should be compiled with a F77 compiler and the compiled program called "newpsi". To run the program just enter "newpsi" from the keyboard. Data is read from the file "newpsi.dat". Data can be in free format. The program asks for the following data:

1. a value of the print code, ip; (If ip=4, great quantities of output are generated for program checkout. Normally the program is run with ip=0 for normal output).
2. the number of design variable, nd;
3. the order of the polynomial being considered, np;
4. the number of design points in the design, m;
5. the design and function value at the design points, x(i,j), y(i);
6. the number of design points on the grid, ng; and
7. the design and function value at the grid points, xx(i,j), yy(j).

Output is written to the screen and to file "newpsi.res".

10. Conclusion

For a given order of approximation of a function, f, the quality of the approximation is affected by

a. the number of levels of the design variables,

b. the location of the design points, and

c. the degree which the approximation is over-determined.

For an nth order approximation,

1. there must be n+1 levels of the design variables;

2. the design points must be located so that information is available for calculating all of the nth derivatives of f;

3. the approximation should be, at least, 20-50% over-determined.

For example, for a 2nd order approximation in 3 design variables, there must be at least 3 levels of the design variables, design points must be located so that information is available for calculating

$$\frac{\partial f}{\partial x_i}, \frac{\partial^2 f}{\partial x_i \partial x_j}, \quad i=1,3;\ j=1,3 \tag{107}$$

A complete 2nd order polynomial approximation contains 10 undetermined coefficients. Thus, at least 10 design points are required to provide information for calculating these coefficients. To have the approximation 30% over-determined, one would want to use 13 design points.

For second order approximations, when there are fewer than 6 design variables, central composite designs meet requirements 1-3. However, for 6 or more design variables, these designs contain too many design points. A minimum point design is one which contains just enough design points, meeting the derivative requirements of item 1 and 2 above, to exactly-determine the approximation. An augmented minimum point design is a minimum point design supplemented with additional design points. The program DESIGNS was developed to yield augmented minimum point designs for 2nd order approximations. The quality of approximations developed using designs from program DESIGNS was comparable to, if not better than, other standard designs such as the central composite designs.

For more than 6 design variables, 3rd and 4th order approximations require so many design points to determine the coefficients in those approximations that it is more computationally efficient to develop a number of 2nd order approximations than one approximation of 3rd or higher order. For 6 or fewer design points, 2nd order approximations may be quite adequate. However, for those cases where one wishes to use a 3rd or 4th order approximation, the program DESIGN4 was develop. That program generates an augmented minimum point design for developing a 4th order approximation.

Previous studies have shown that the quality of approximations using neural networks is comparable to those using polynomial approximations when the number of undetermined parameters associated with the approximations is the same. Thus, neural networks trained with designs from DESIGNS or DESIGN4 should offer approximations of comparable quality to those obtained using polynomial approximations with the same number of associated undetermined parameters.

11. References

1. Box, G.E.P. and Draper, N.R., Empirical Model-Building and Response Surfaces, John Wiley and Sons, New York (1987).

2. Andre I. Khuri and John A. Cornell, Response Surfaces, Designs and Analyses, Marcel Dekker Inc, New York (1987).

3. Raymond H. Myers, Response Surface Methodology, Allyn and Bacon, Boston (1971).

4. Greville, T.N.E., "The Pseudoinverse of a Rectangular or Singular Matrix and Its Application to the Solution of Systems of Linear Equations," SIAM Review 1, No. 1 (1959) 38-43.

5. Penrose, R., "A Generalized Inverse for Matrices," Proc. Cambridge Philos. Soc. 51 (1955) 406-413.

6. Rumelhart, D. and McClelland, J., Parallel Distributed Processing, Vol I and II, MIT Press, Cambridge, MA (1986).

7. Anderson, J. and Rosenfeld, E., Neurocomputing: Foundations of Research, MIT Press, Cambridge MA (1988).

8. R.D. Vanluchene and Roufei Sun, "Neural Networks in Structural Engineering," Microcomputers in Civil Engineering 5, No. 3 (1990) 207-215.

9. P. Hajela and L. Berke, "Neurobiological Computational Models in Structural Analysis and Design," paper AIAA-90-1133-CP, AIAA/ASME/ASCE/AHS/ASC 31st Structures, Structural Dynamics and Materials Conference, April 2-4 (1990)

10. Swift, R.A. and Batill, S.M., "Application of neural networks to preliminary structural design," AIAA/ASME/AHS/ASC 32nd Structures, Structural Dynamics and Materials Conference, Baltimore, MD, April 8-10 (1991) 335-343.

11. Berke, L. and Hajela, P., "Application of artificial neural nets in structural mechanics," Lecture Series on "Shape and Layout Optimization of Structural Systems' at the International Center for Mechanical Sciences, Udine, Italy, July 16-20 (1990).

12. Rogers, J.L. and LaMarsh, W.J., "Application of a Neural Network to Simulate Analysis in an Optimization Process," Artificial Intelligence in Design 92, John S. Gero editor, Kluwer Academic Publishers, Boston (1992) 739-754.

13. Hornik, Kurt, et. al., "Multilayer Feedforward Networks are Universal

Approximators," Neural Networks 2 (1989) 359-366.

14. Carpenter, W.C., NEWNET User's Guide, University of South Florida, Tampa, Florida (1992).

15. Reklaitis, G.V., Ravindran, A., and Ragsdell, K.M., Engineering Optimization, Methods and Applications, John Wiley & Sons, New York (1983).

16. Fox, R.L., Optimization Methods for Engineering Design, Addison-Wesley Publishing Company, Reading, Mass (1971).

17. Kaplan, Wilfred, Advanced Calculus Addison-Wesley Publishing Company, Reading, Mass (1952).

18. Al-Khafaii, A.W. and Tooley, J.R., Numerical Methods in Engineering Practice, Holt, Rinehart, and Winston, Inc., New York (1986).

19. Carpenter, W.C. and Barthelemy, J.-F. M., "Neural Nets as Approximators in Engineering and Science Applications', manuscript in preparation.

20. Carpenter, W.C., DESIGNS User Manual, University of South Florida, Tampa, Florida (1992).

21. Sacks, J., et. al., "Design and Analysis of Computer Experiments," Statistical Science 4, No. 4, (1989) 409-435.

22. Schoofs, A.J.G., Experimental Design and Structural Optimization, Ph.D Dissertation, Technical University of Eindhoven (1987).

23. Welch, W.J., "Branch and Bound Search for Experimental Designs Based on D-Optimality and Other Criteria;, Technometrics 24 (1982) 41-48.

24. Fedorov, V.V., Theory of Optimal Experiments, Academic Press, New York (1972).

25. Mitchell, T.J., "An Algorithm for the Construction of 'D-Optimal' Experimental Designs," Technometrics 16, No. 2 (1974) 203-210.

26. Efroymson, M.A., "Multiple Regression Analysis," Mathematical Methods for Digital Computers, A. Ralston and H.S. Wilf, ed., John Wiley & Sons, New York (1965).

27. Barthelemy, J.-F. and Dovi, A.R., "Application of multidisciplinary optimization methods to the design of a supersonic transport," Air force/NASA Symposium on Recent Advances in Multidisciplinary Analysis and Optimization, San Francisco, Ca, Sept. 24-26 (1990) 4.1-4.5.

28. Carpenter, W.C., DESIGN4 User Manual, University of South Florida, Tampa, Florida (1992).

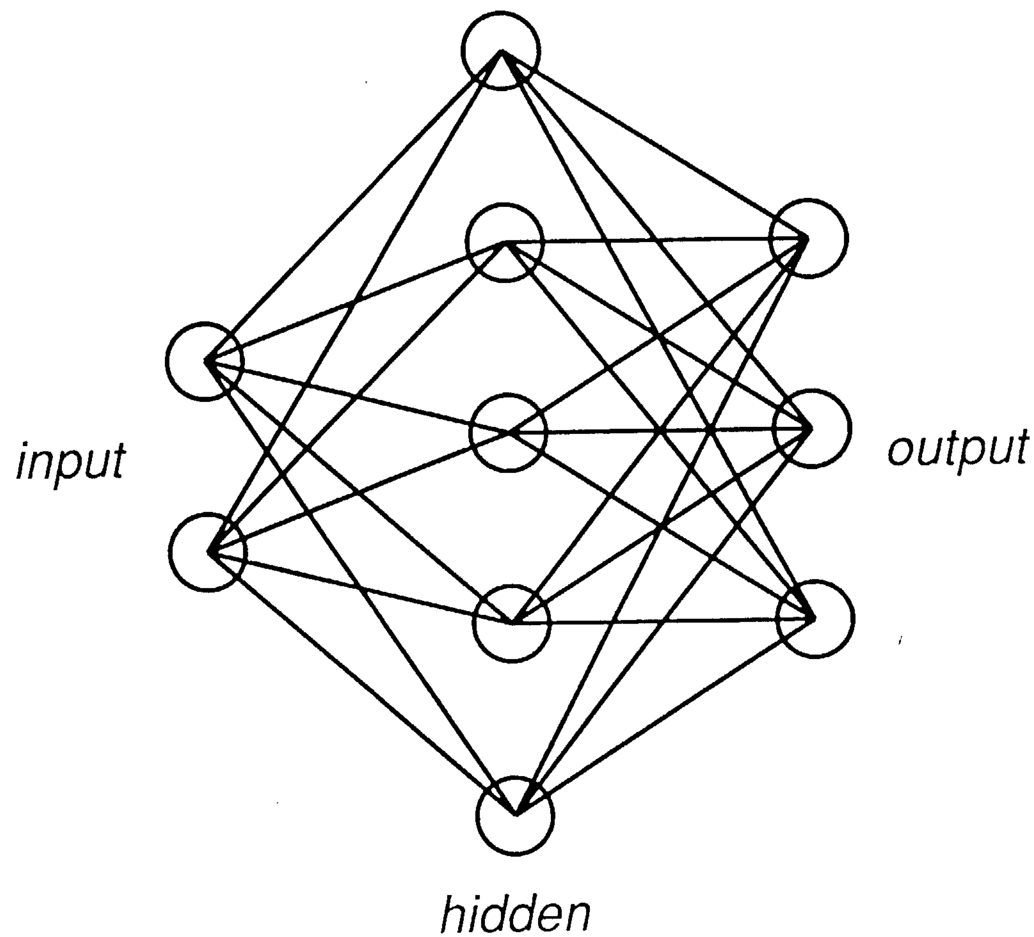

Figure 1. Artificial neural net

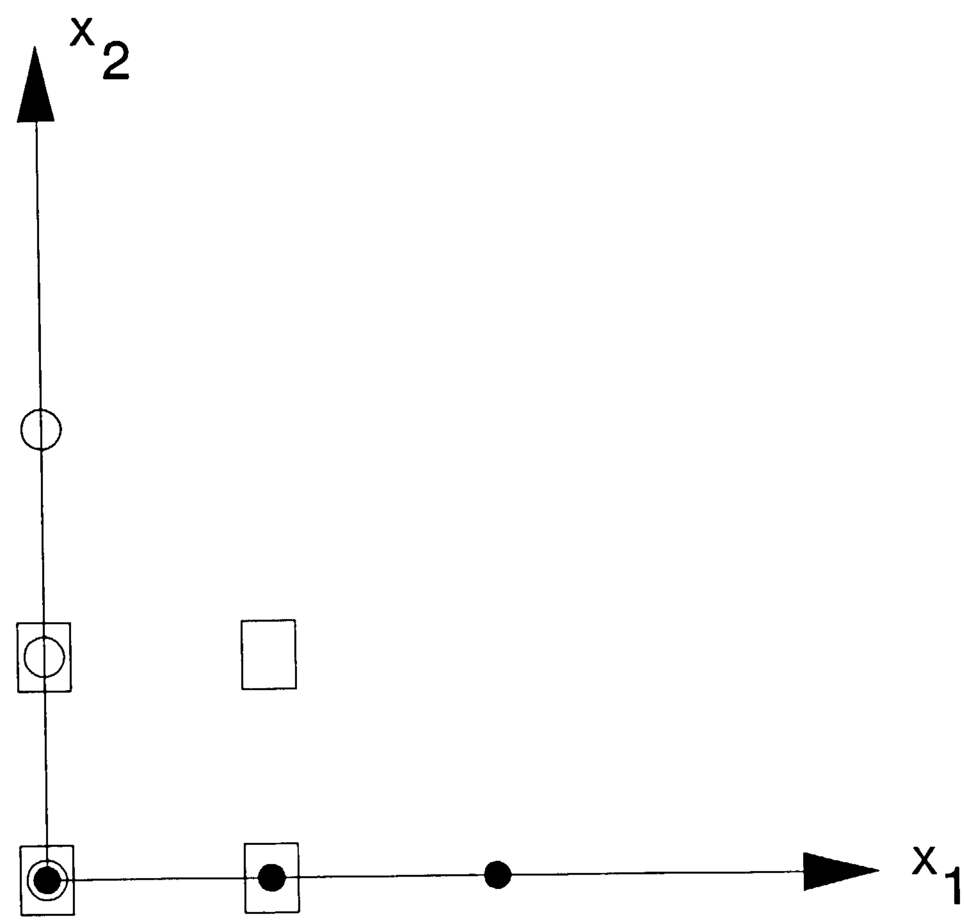

Figure 2. Complete design

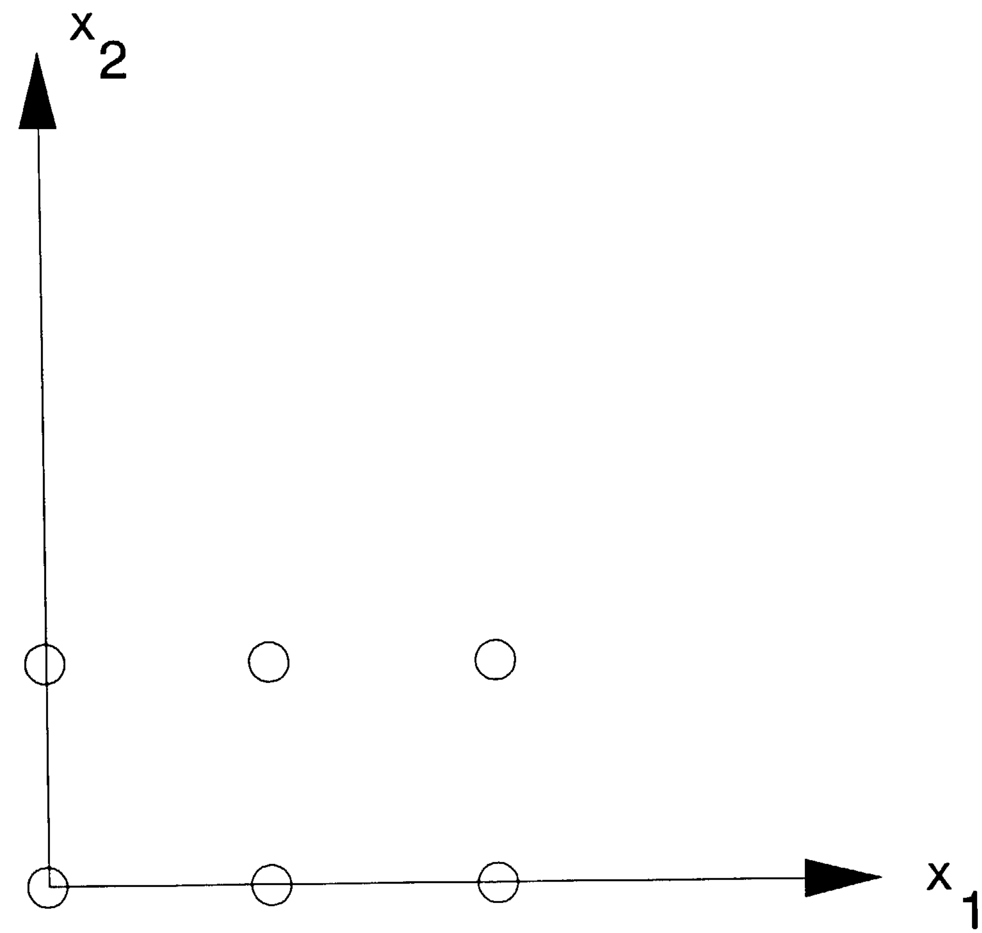

Figure 3. Deficient design

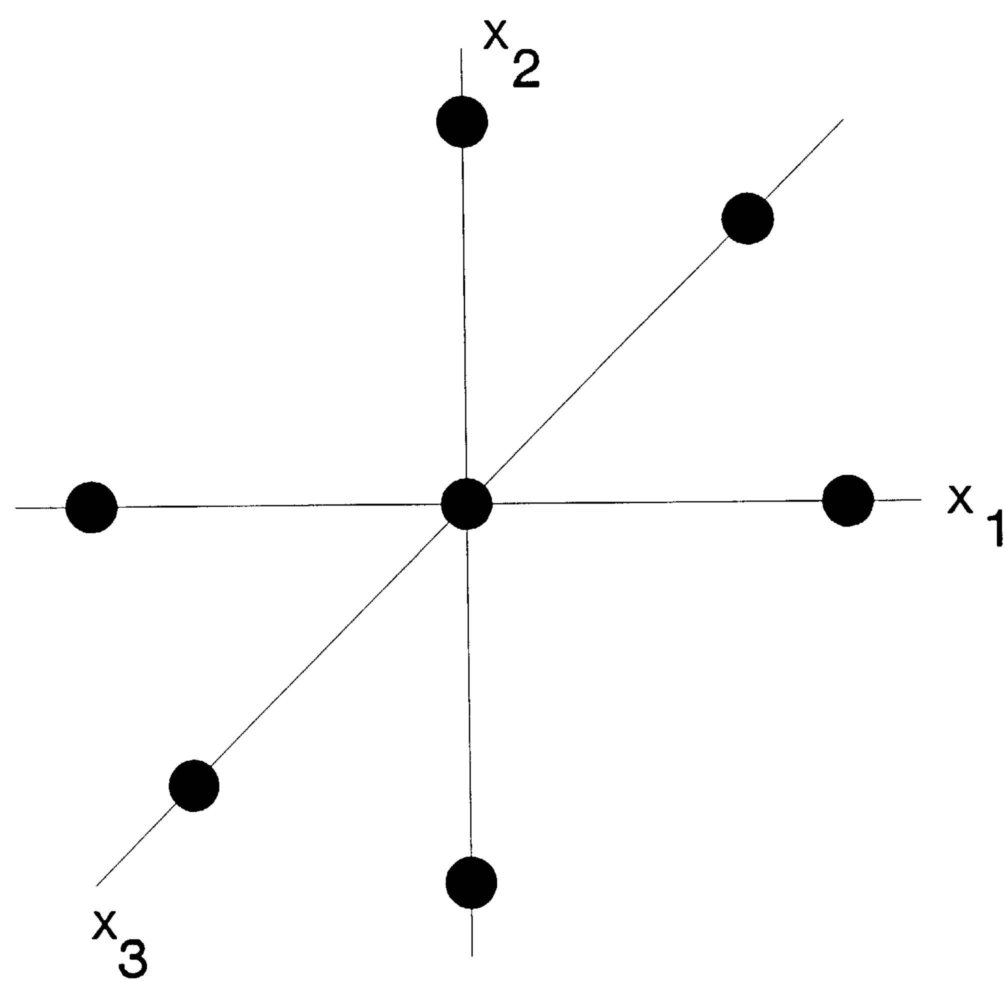

Figure 4. Star pattern of design points--7 design points

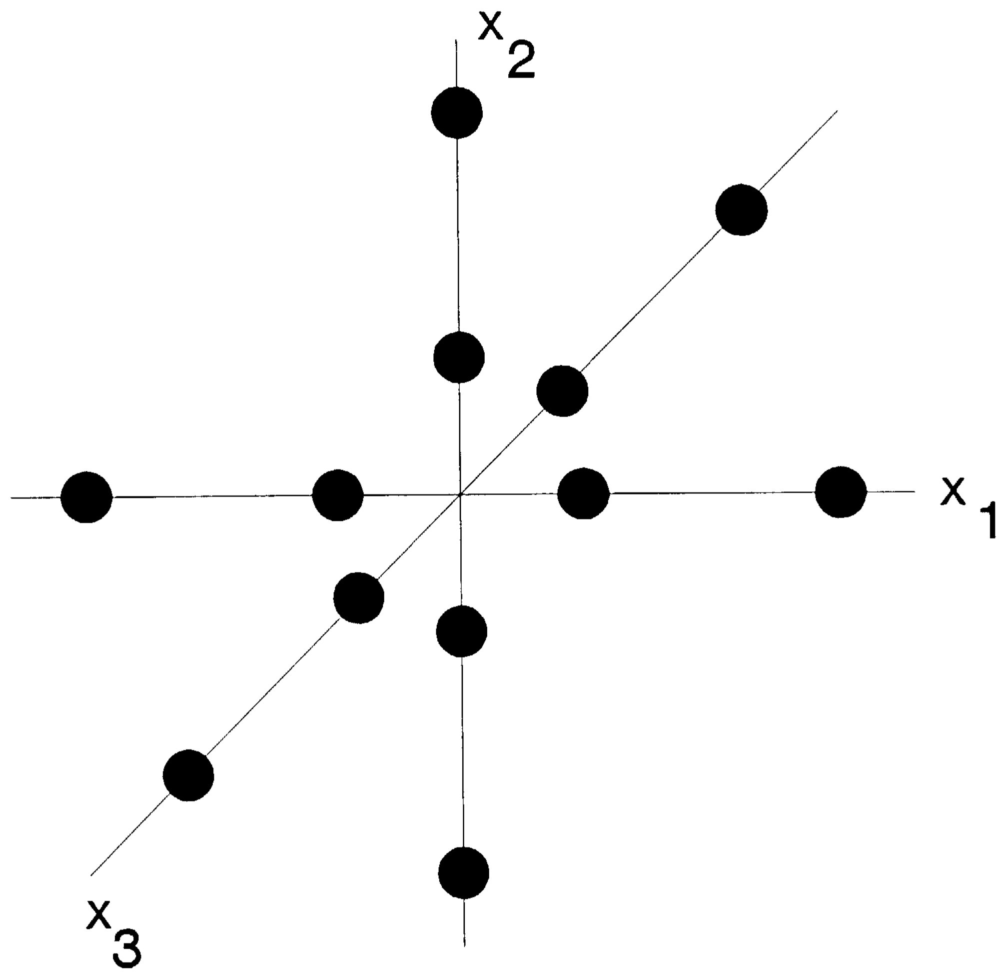

Figure 5. Star pattern of design points--12 design points

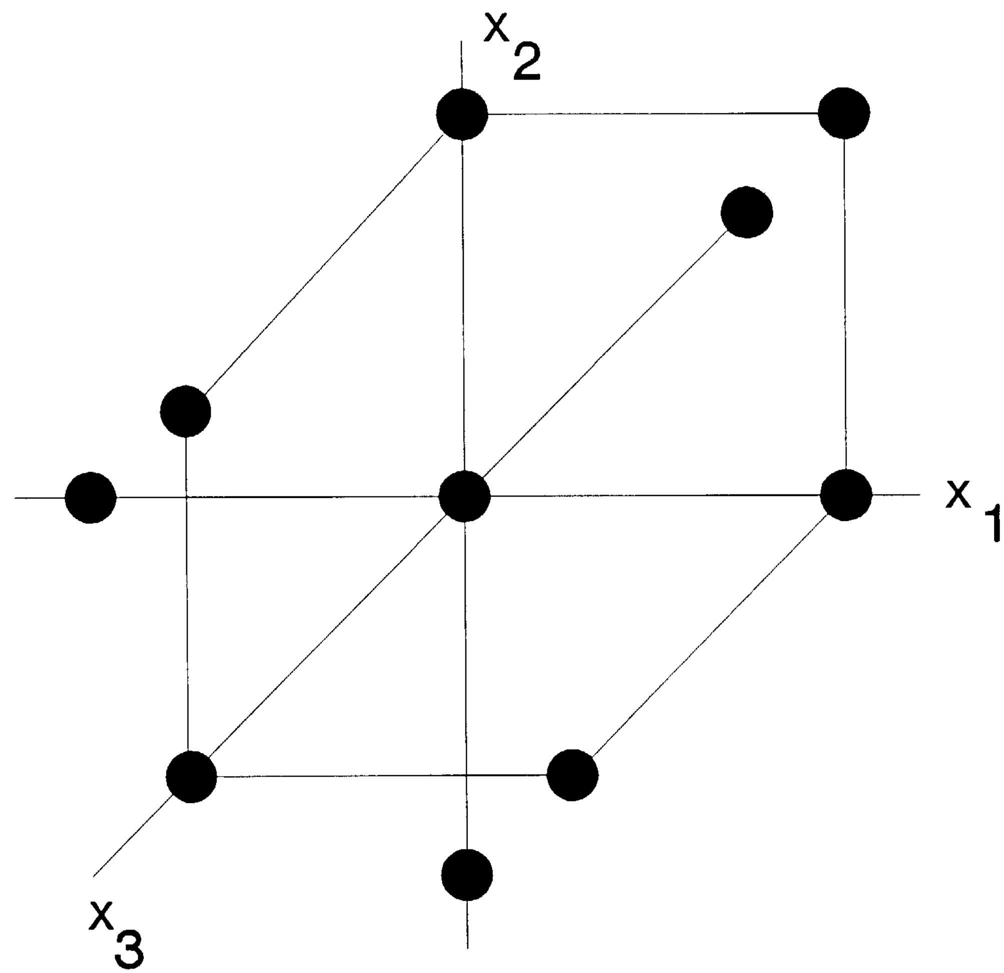

Figure 6. Design from program DESIGNS

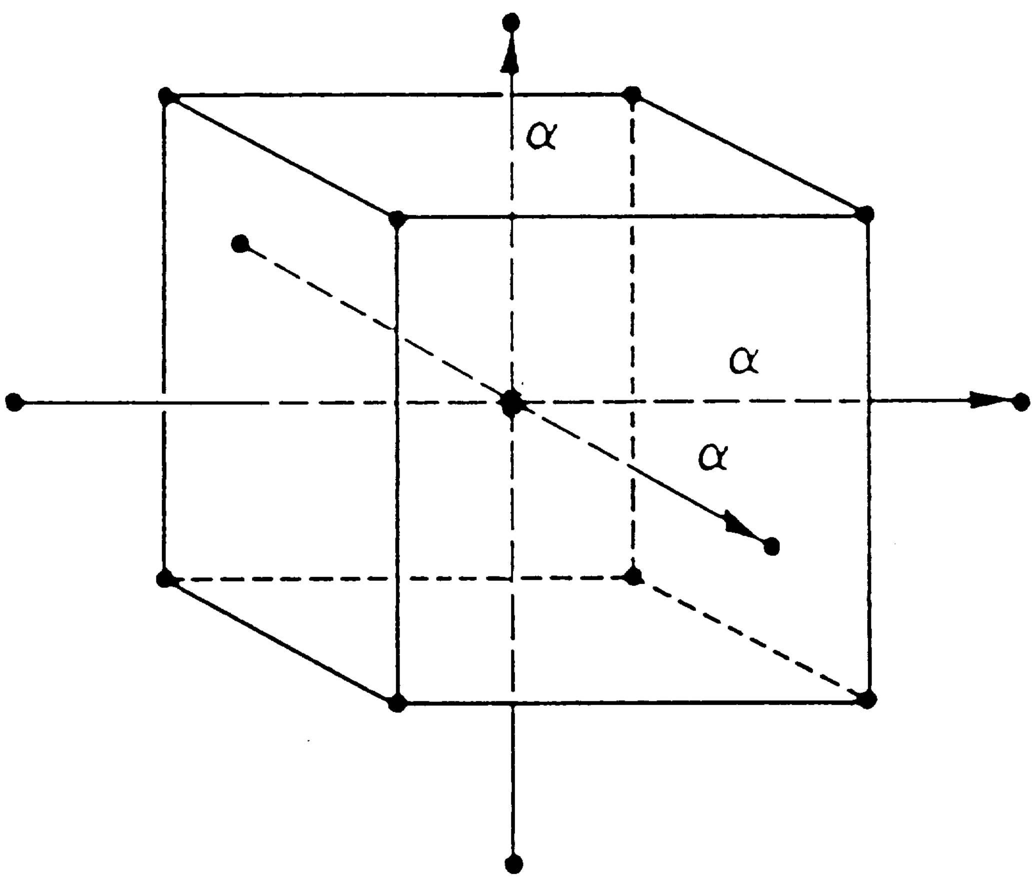

Figure 7. Central composite design for k=3

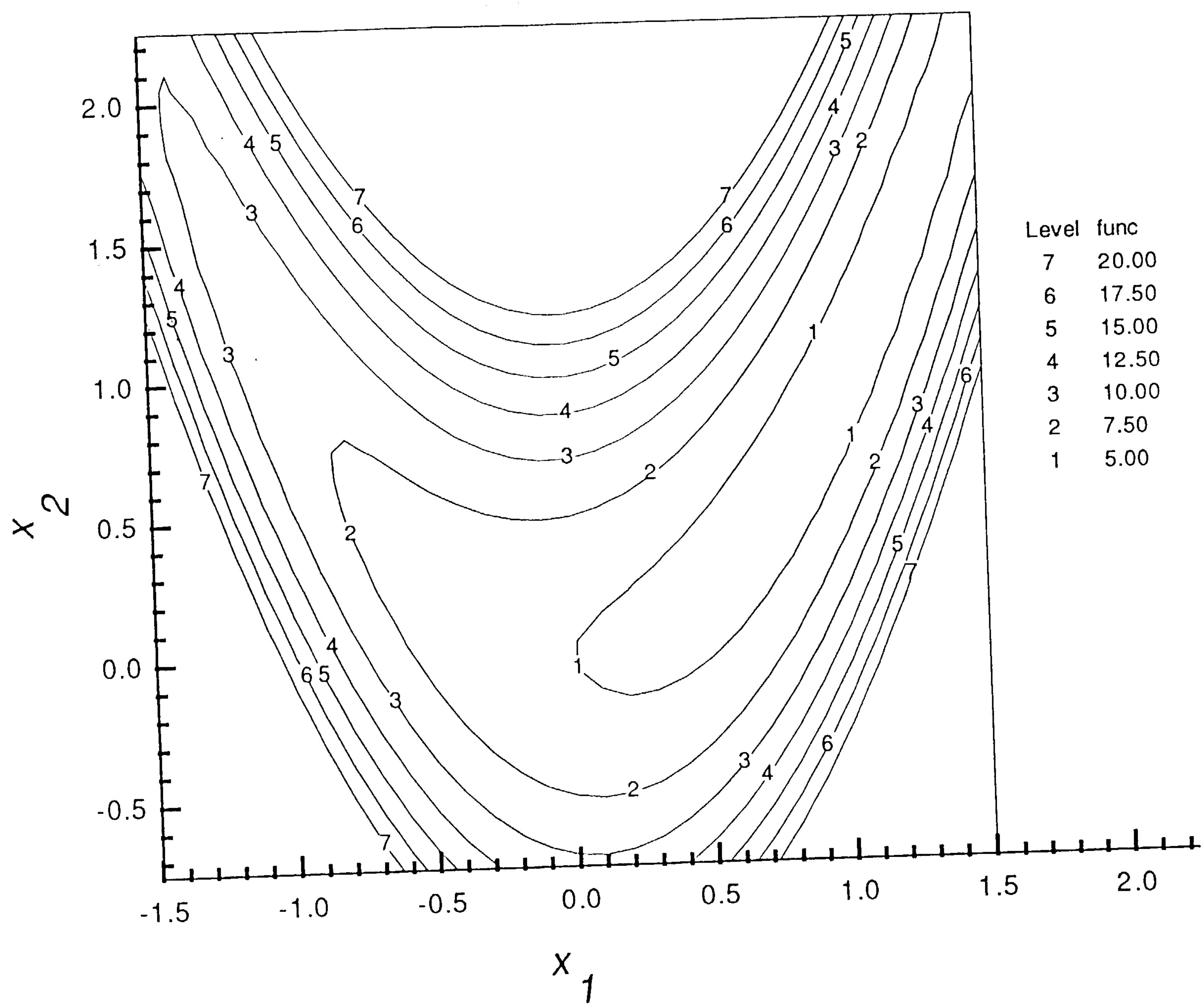

Figure 8. Fox's banana function

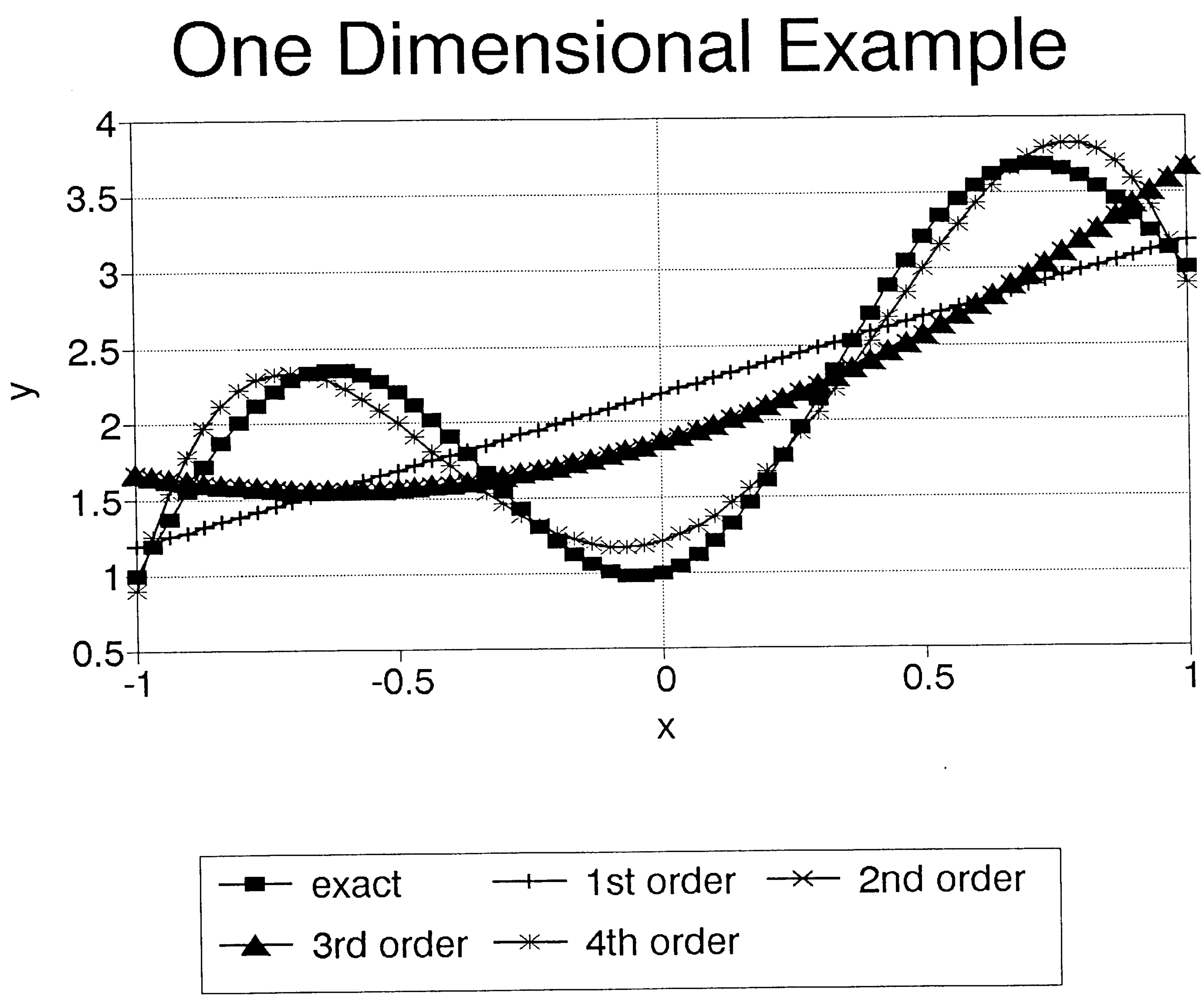

Figure 9. One dimensional example

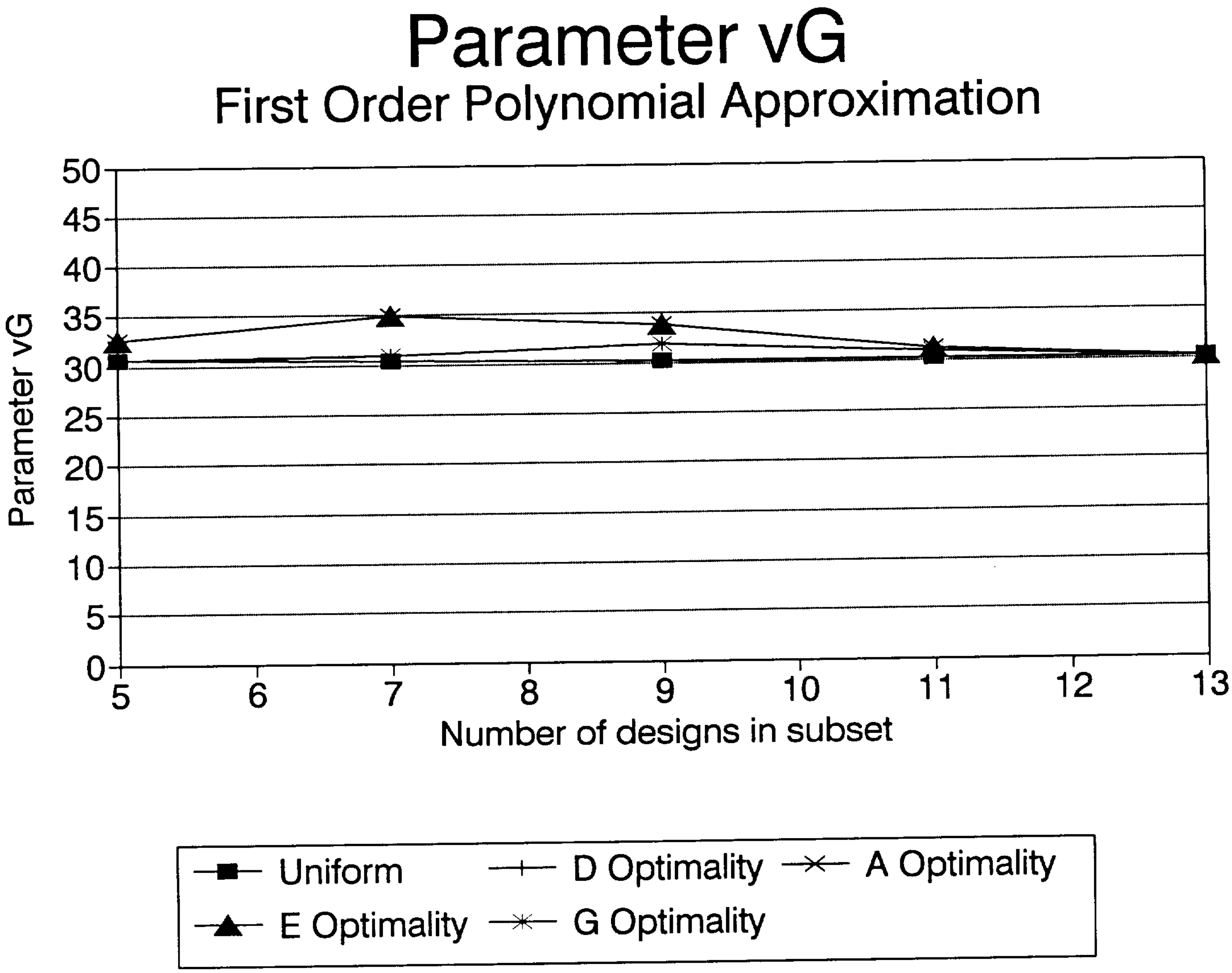

Figure 10. D, A, E, and G optimality, first order approximation

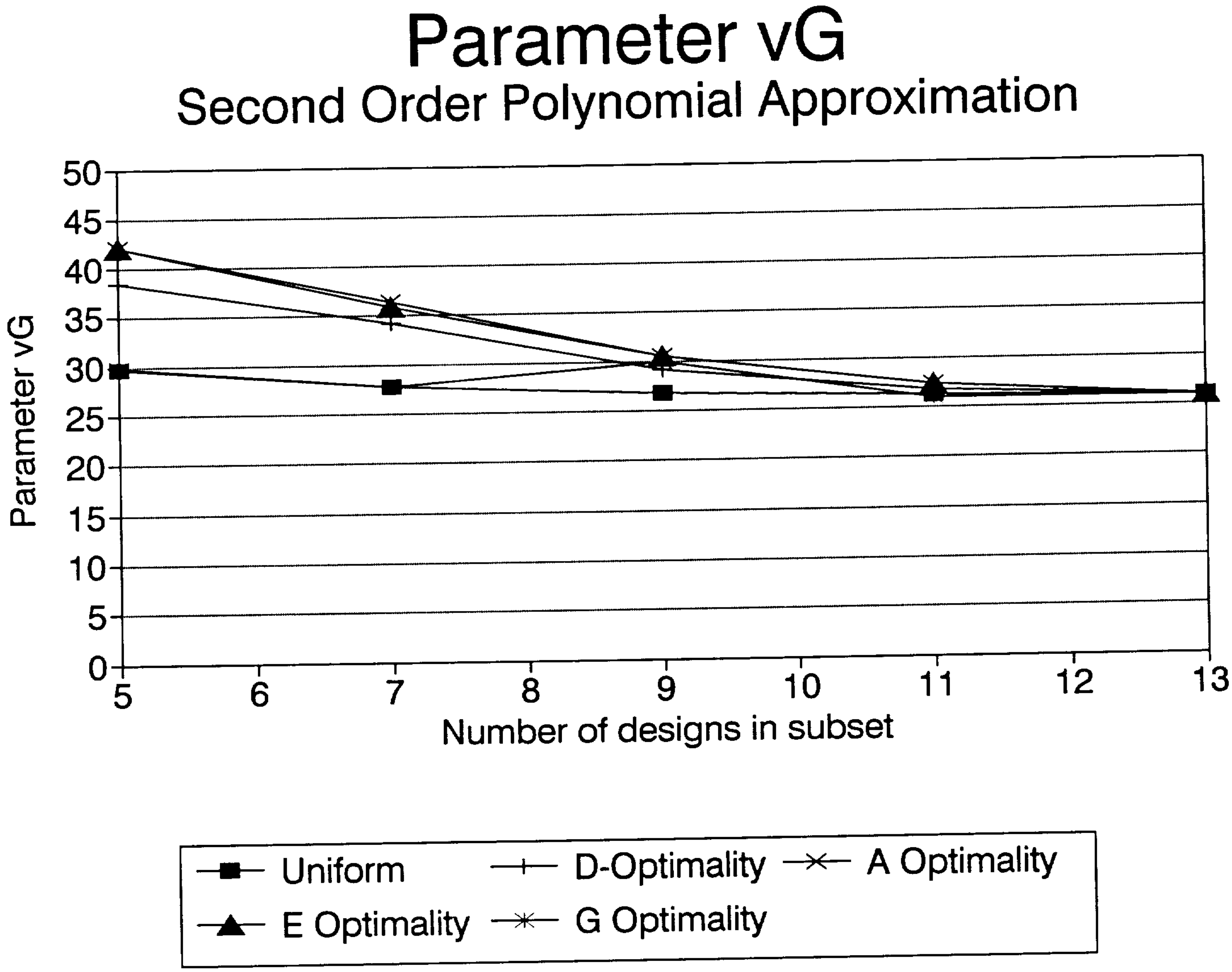

Figure 11. D, A, E, and G optimality, second order approximation

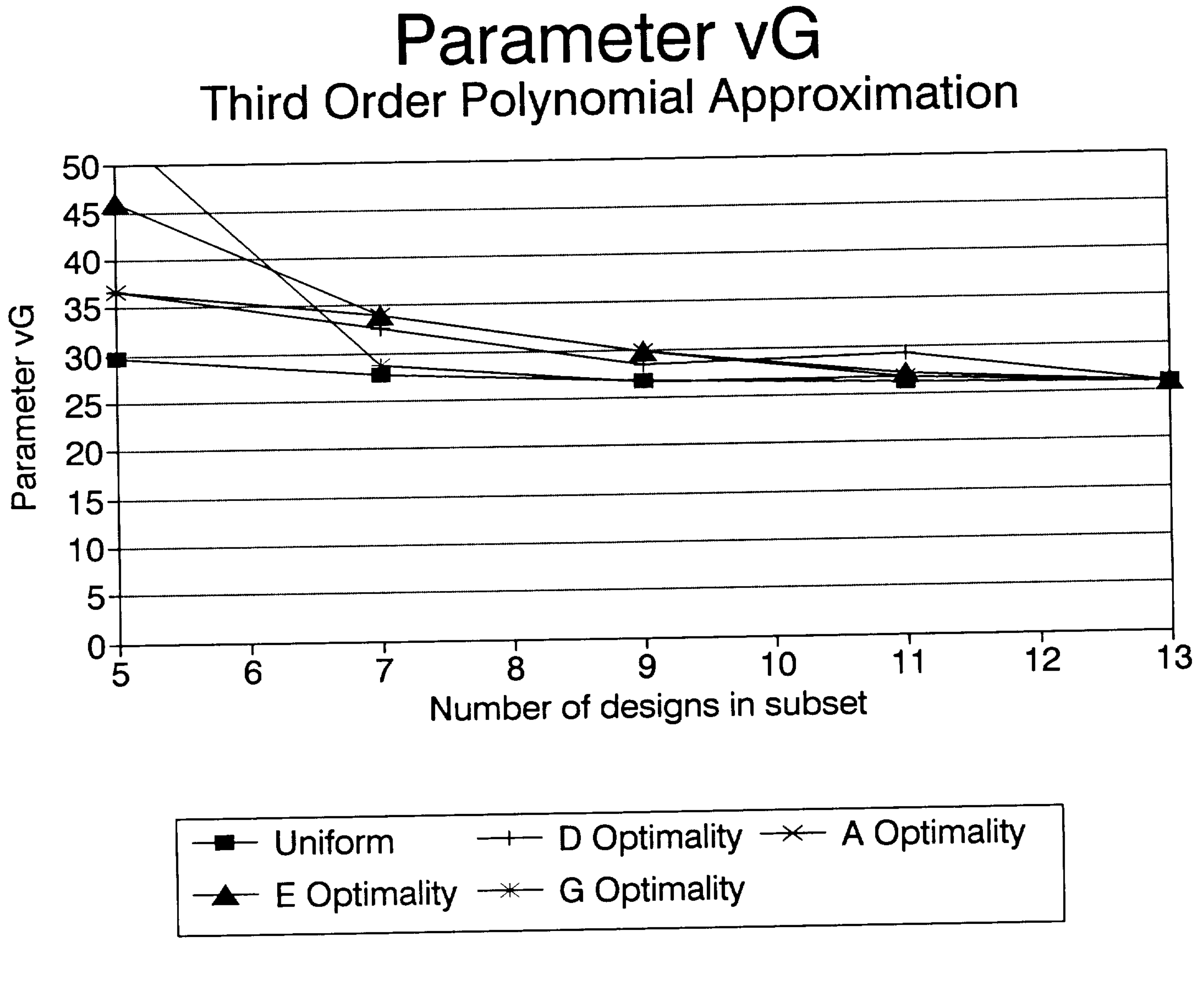

Figure 12. D, A, E, and G optimality, third order approximation

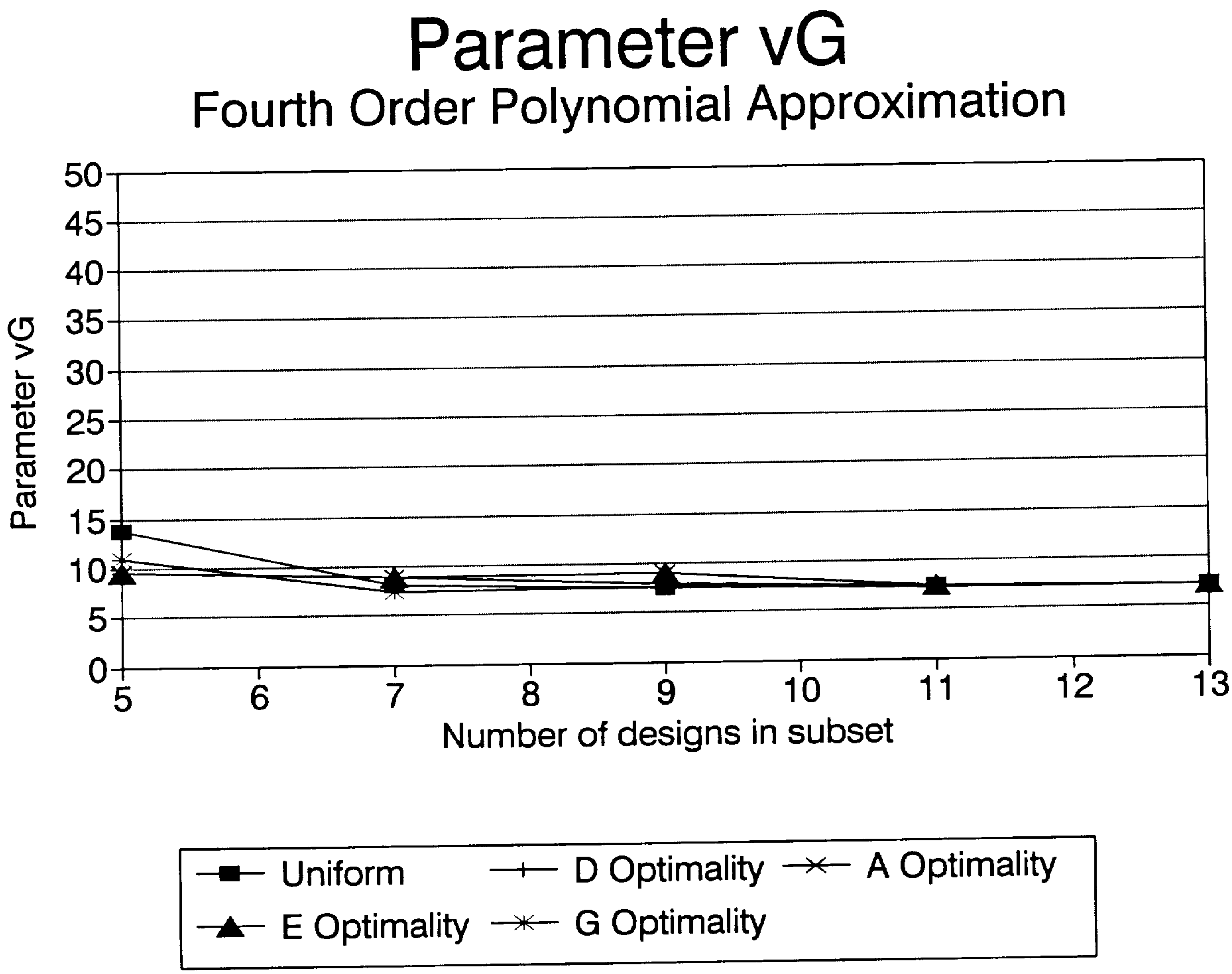

Figure 13. D, A, E, and G optimality, fourth order approximation

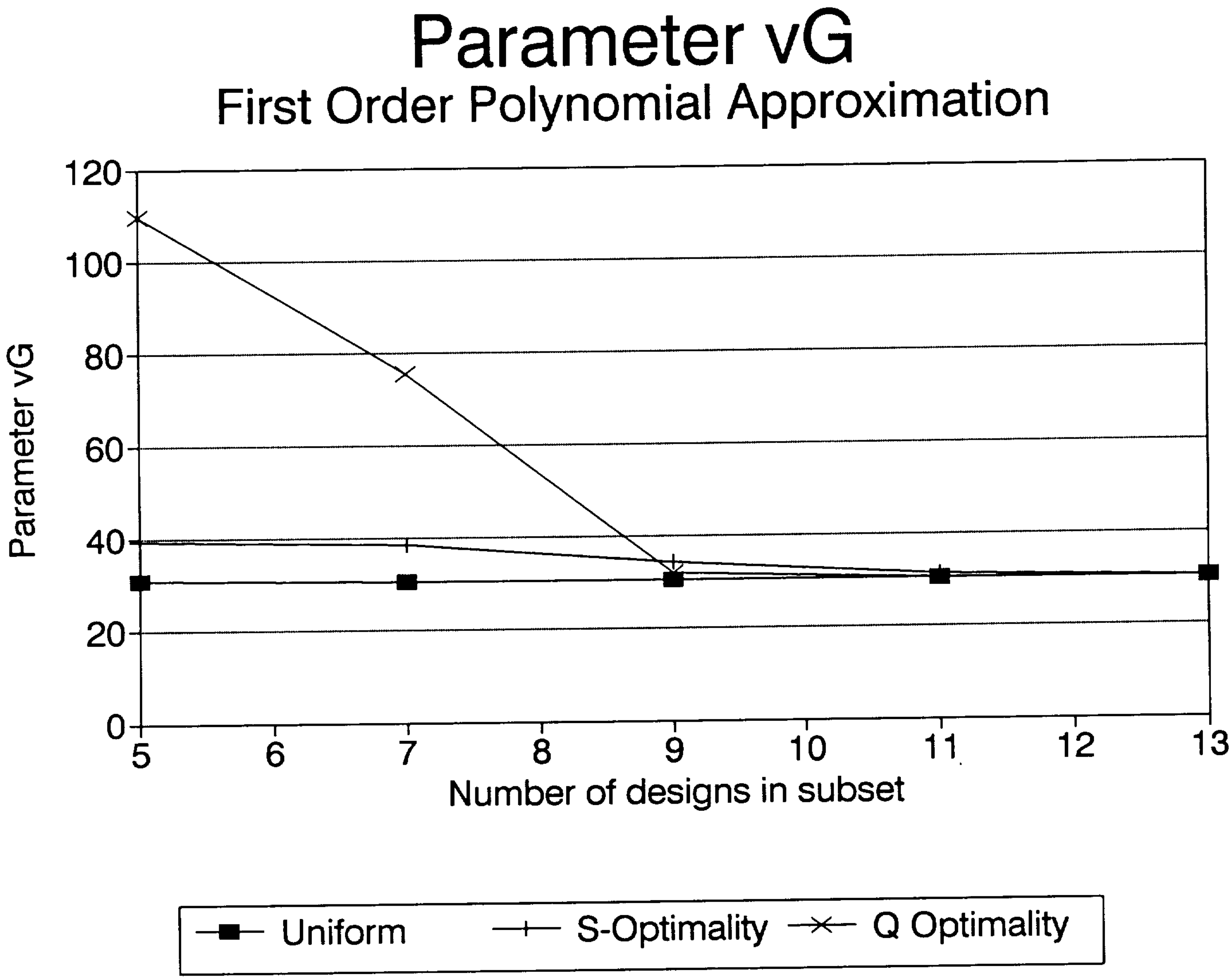

Figure 14. S and Q optimality, first order approximation

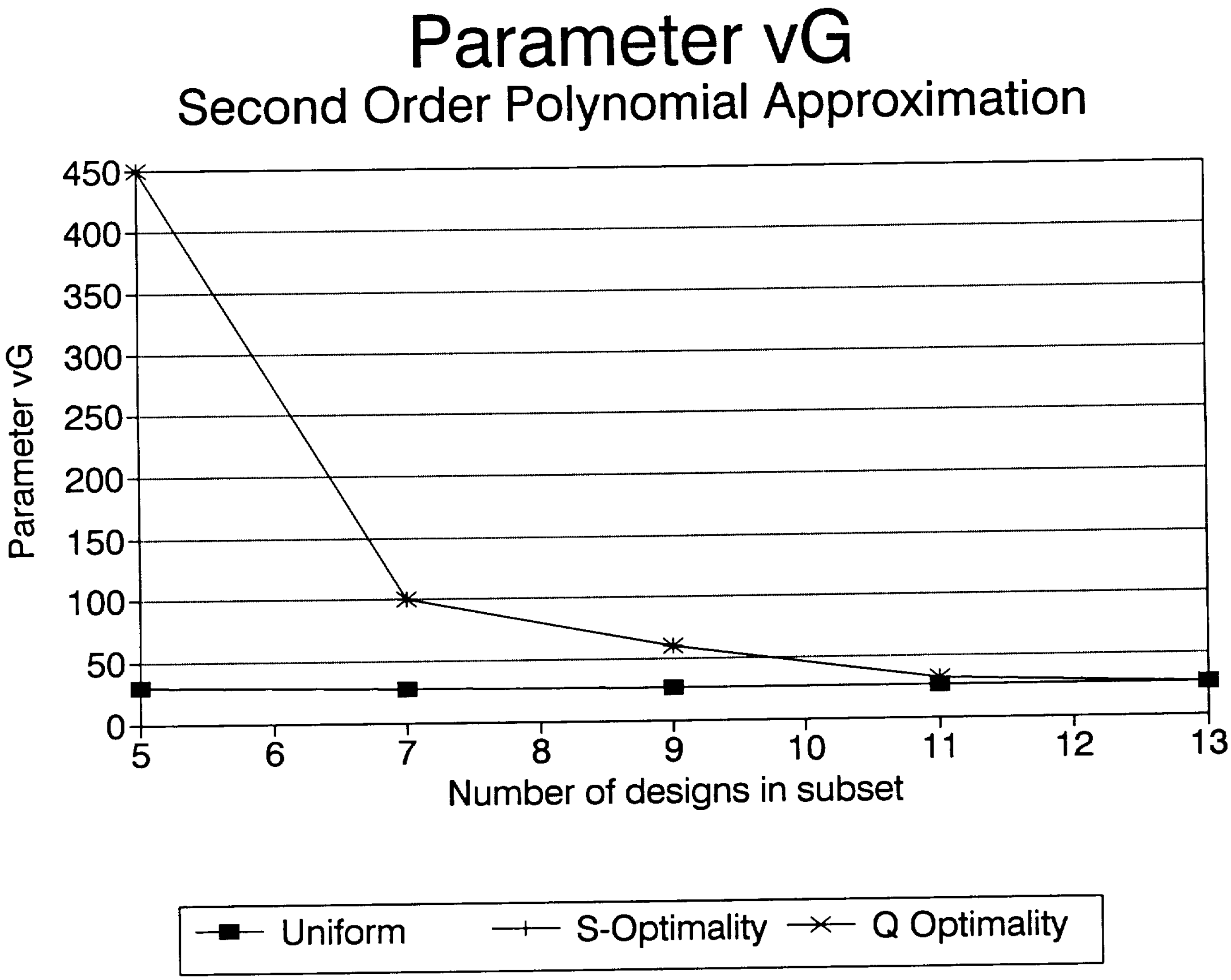

Figure 15. S and Q optimality, second order approximation

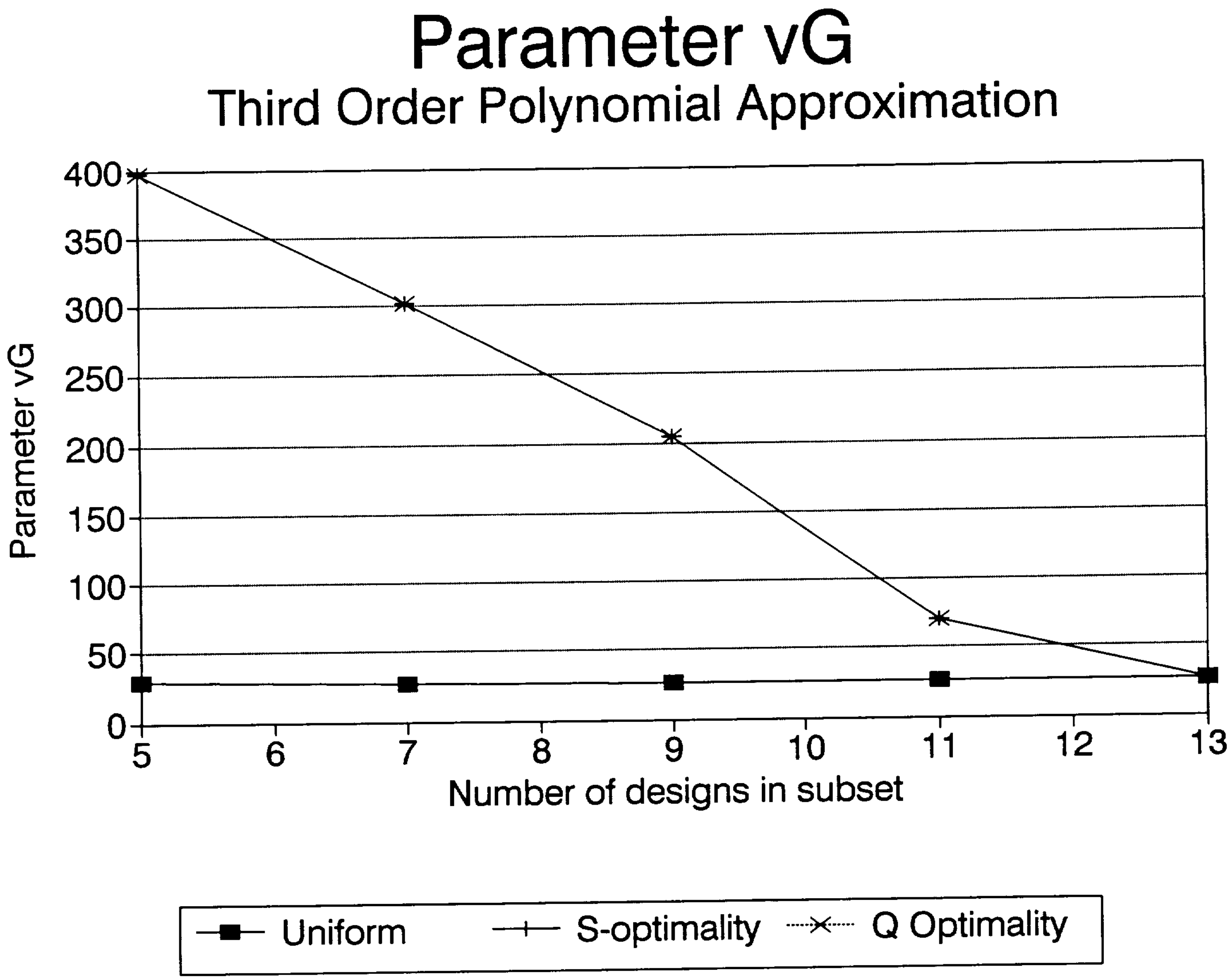

Figure 16. S and Q optimality, third order approximation

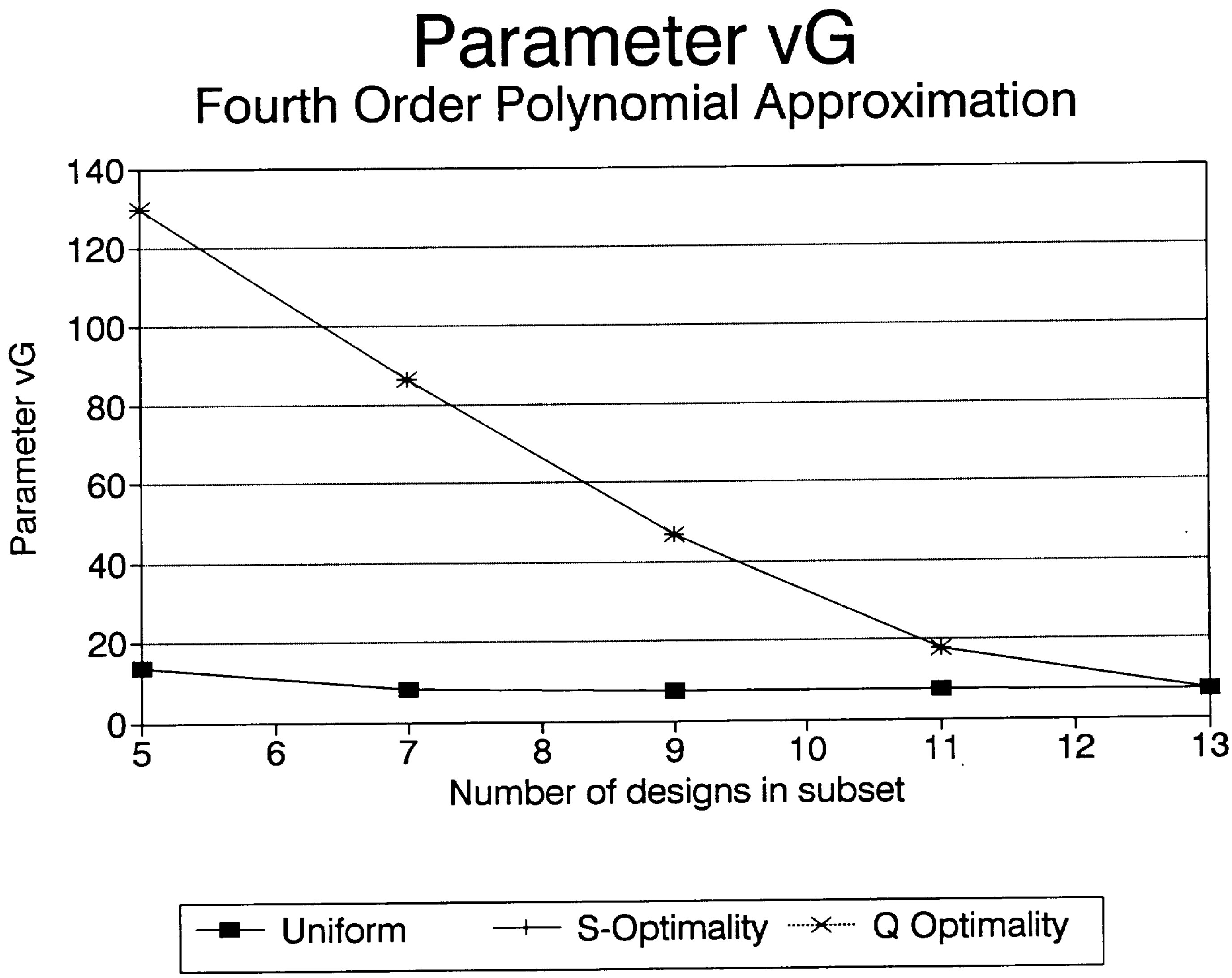

Figure 17. S and Q optimality, fourth order approximation

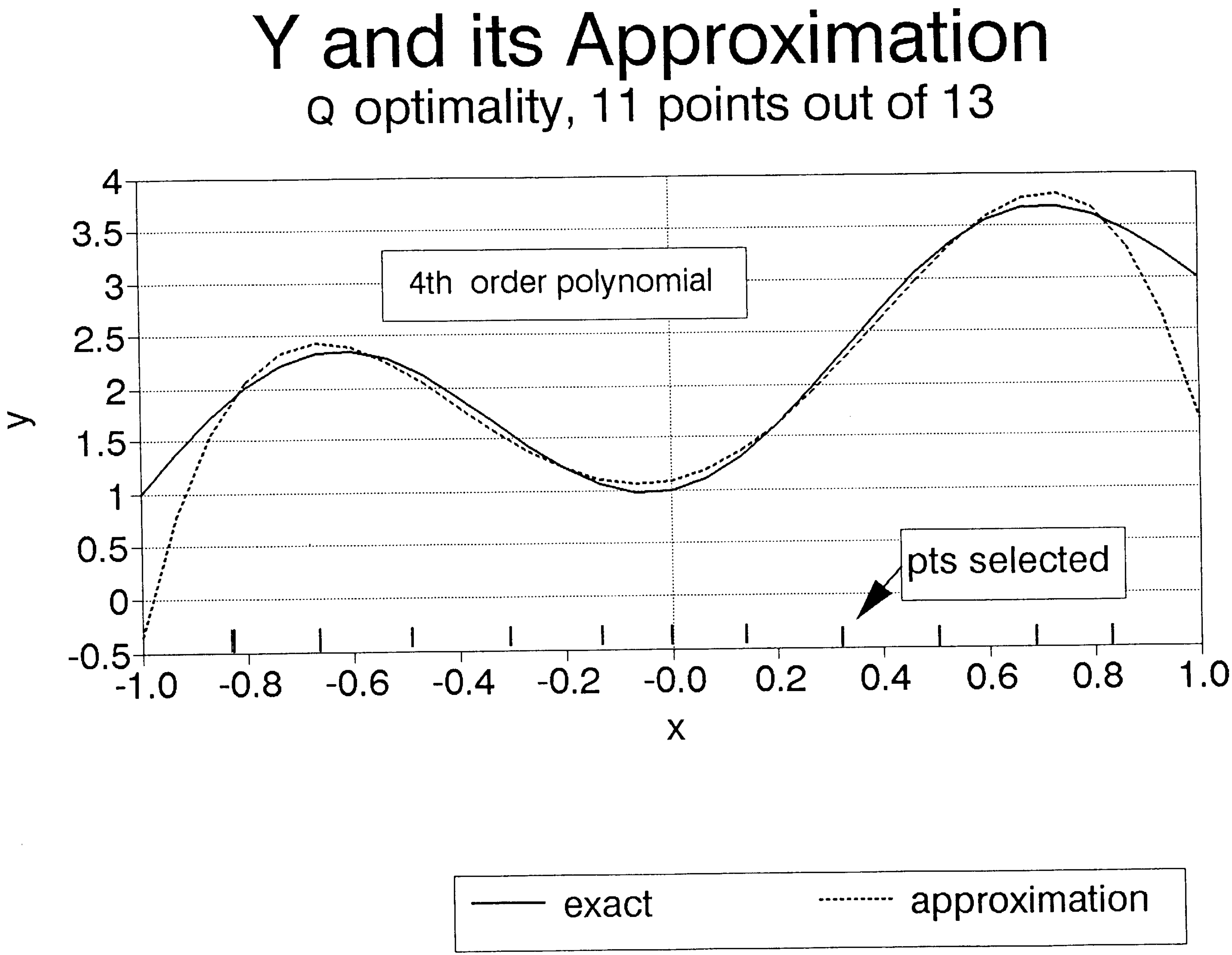

Figure 18. Q optimality, 11 out of 13 points selected

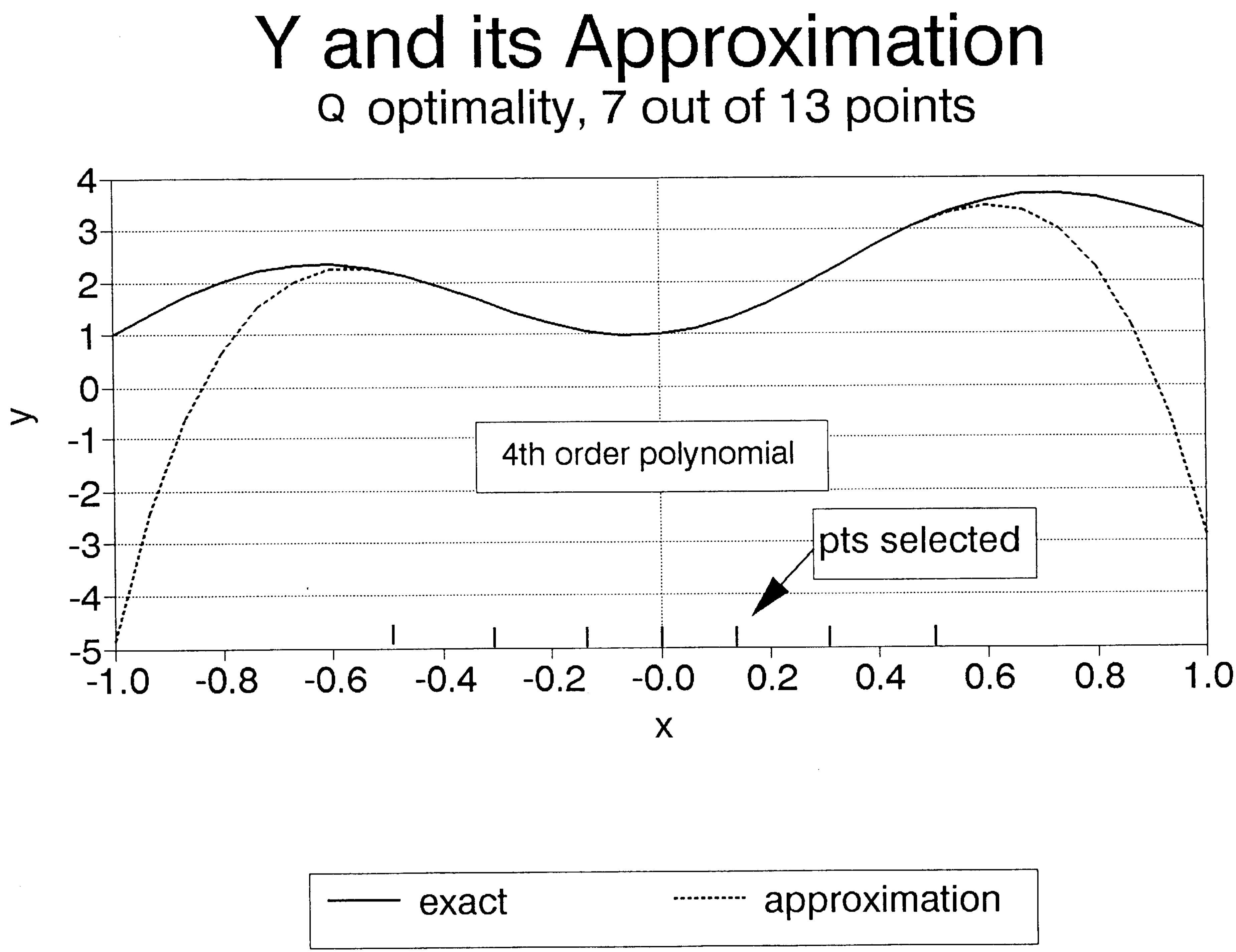

Figure 19. Q optimality, 7 out of 13 points selected

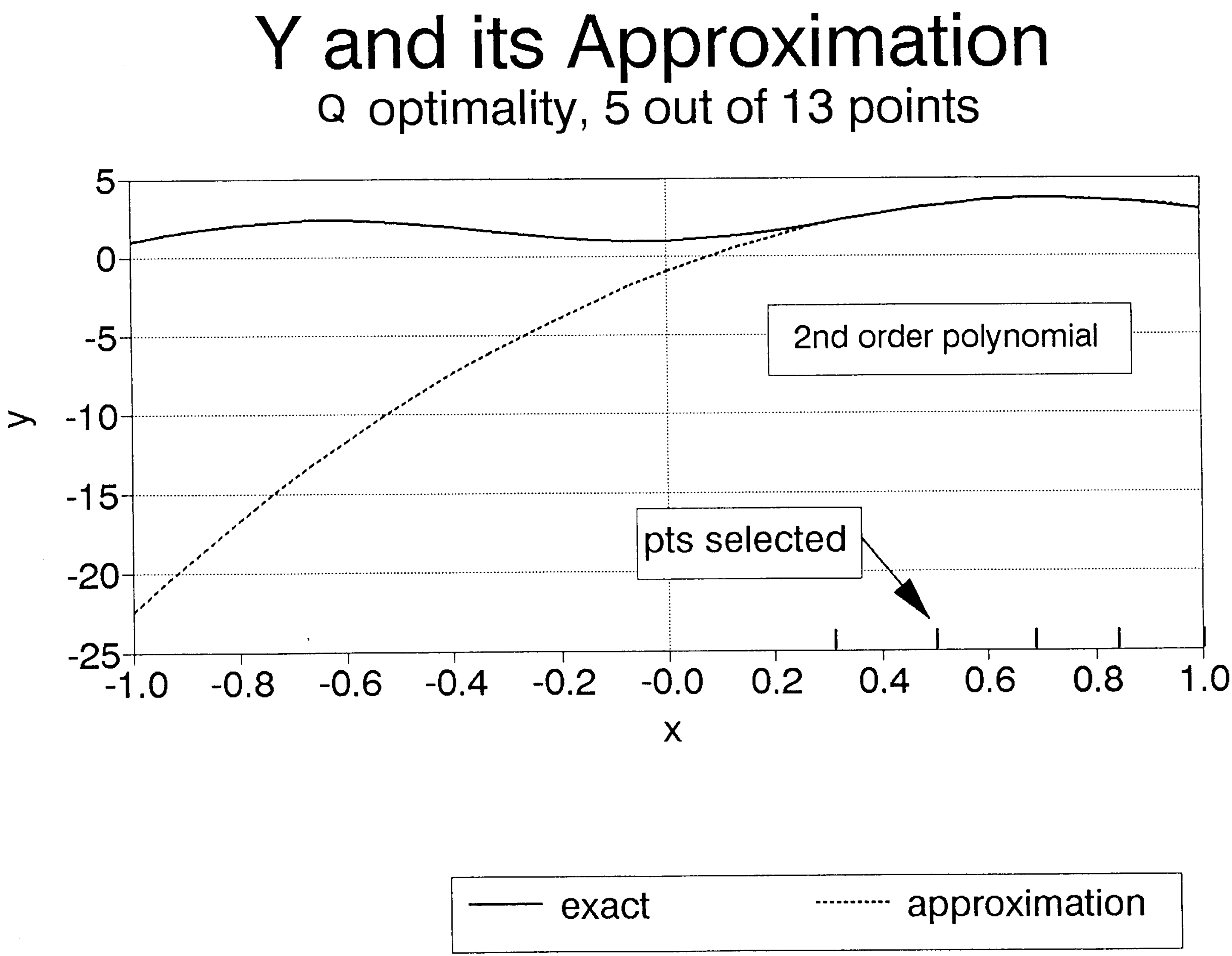

Figure 20. Q optimality, 5 out of 13 points selected

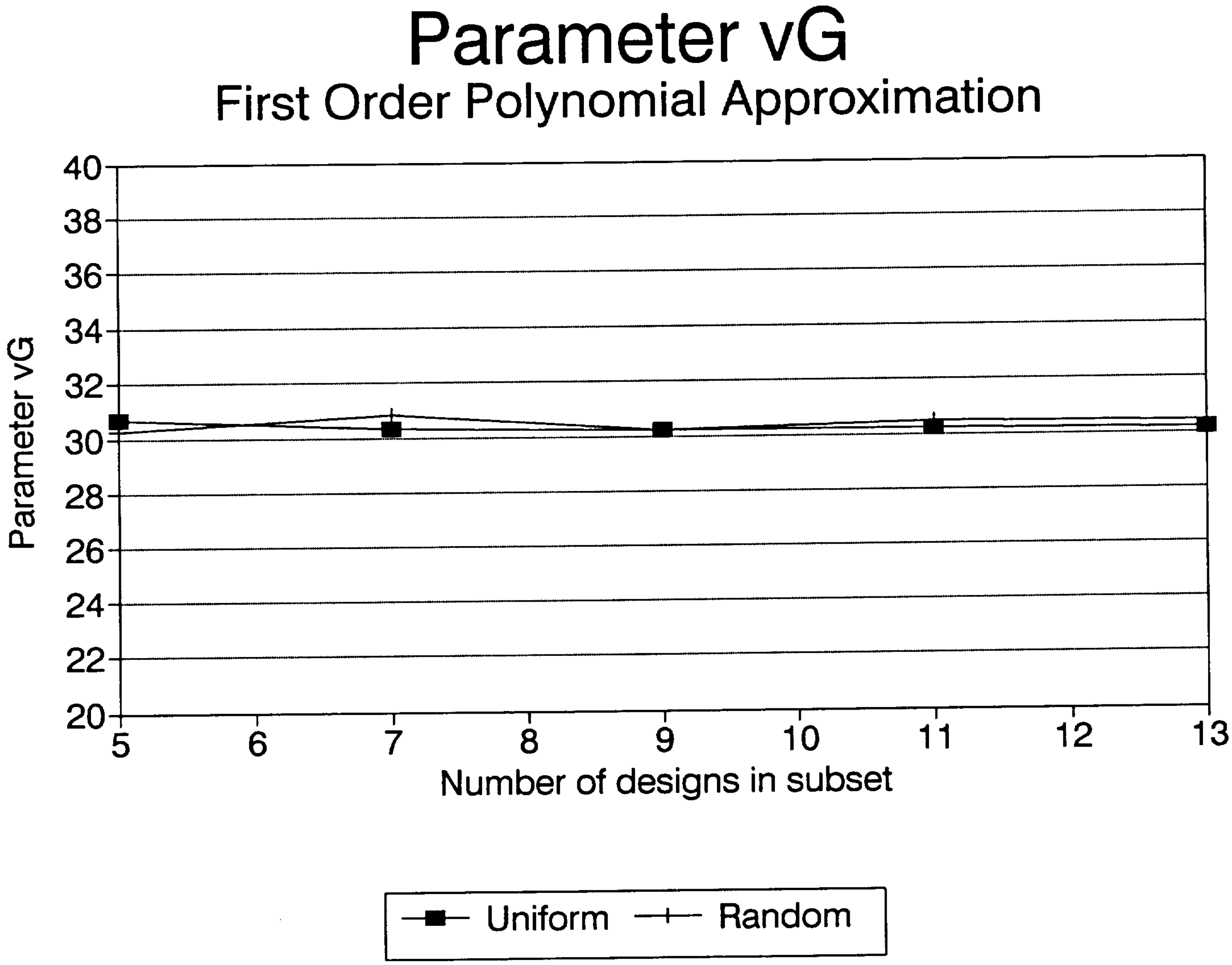

Figure 21. Random points, first order approximation

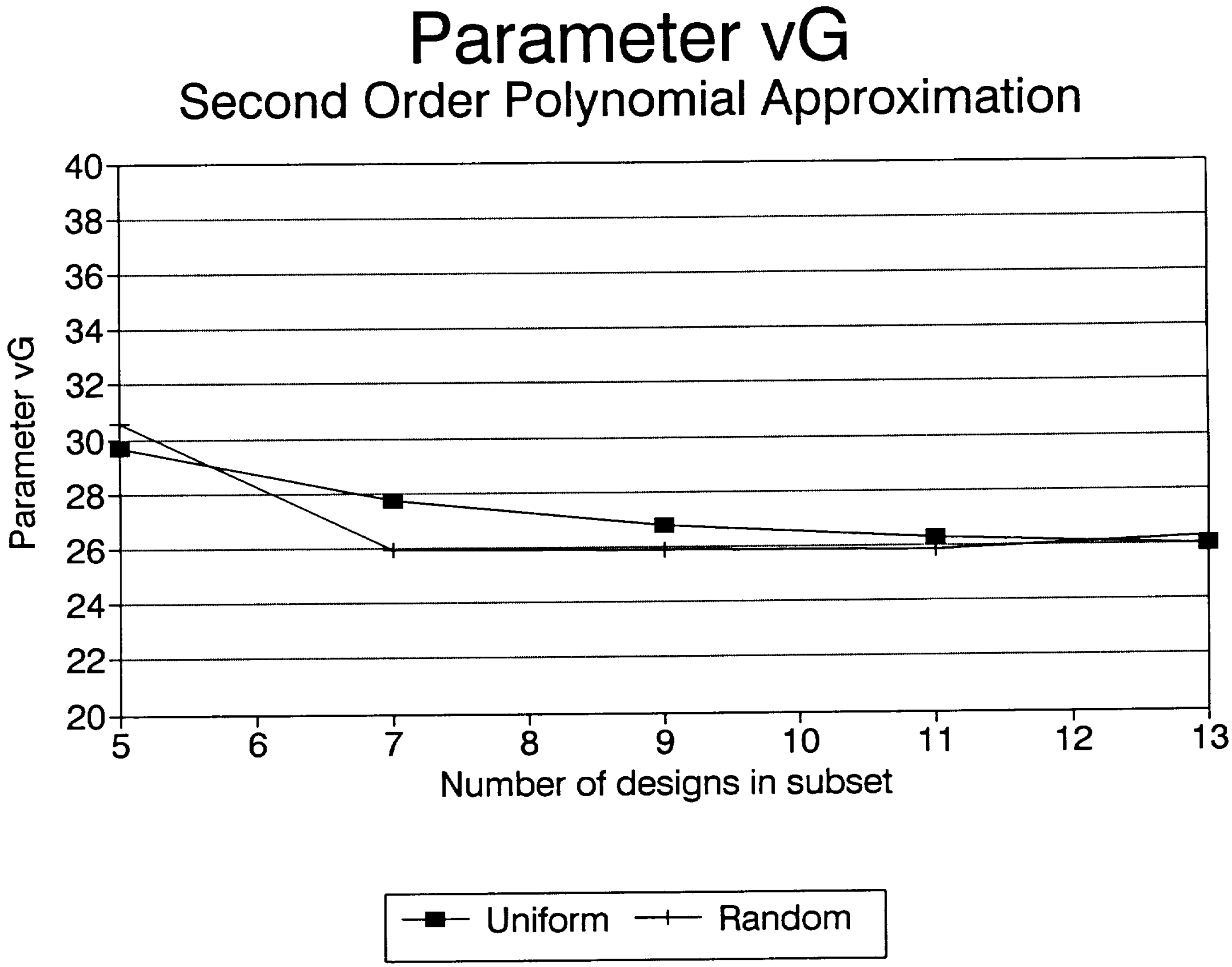

Figure 22. Random points, second order approximation

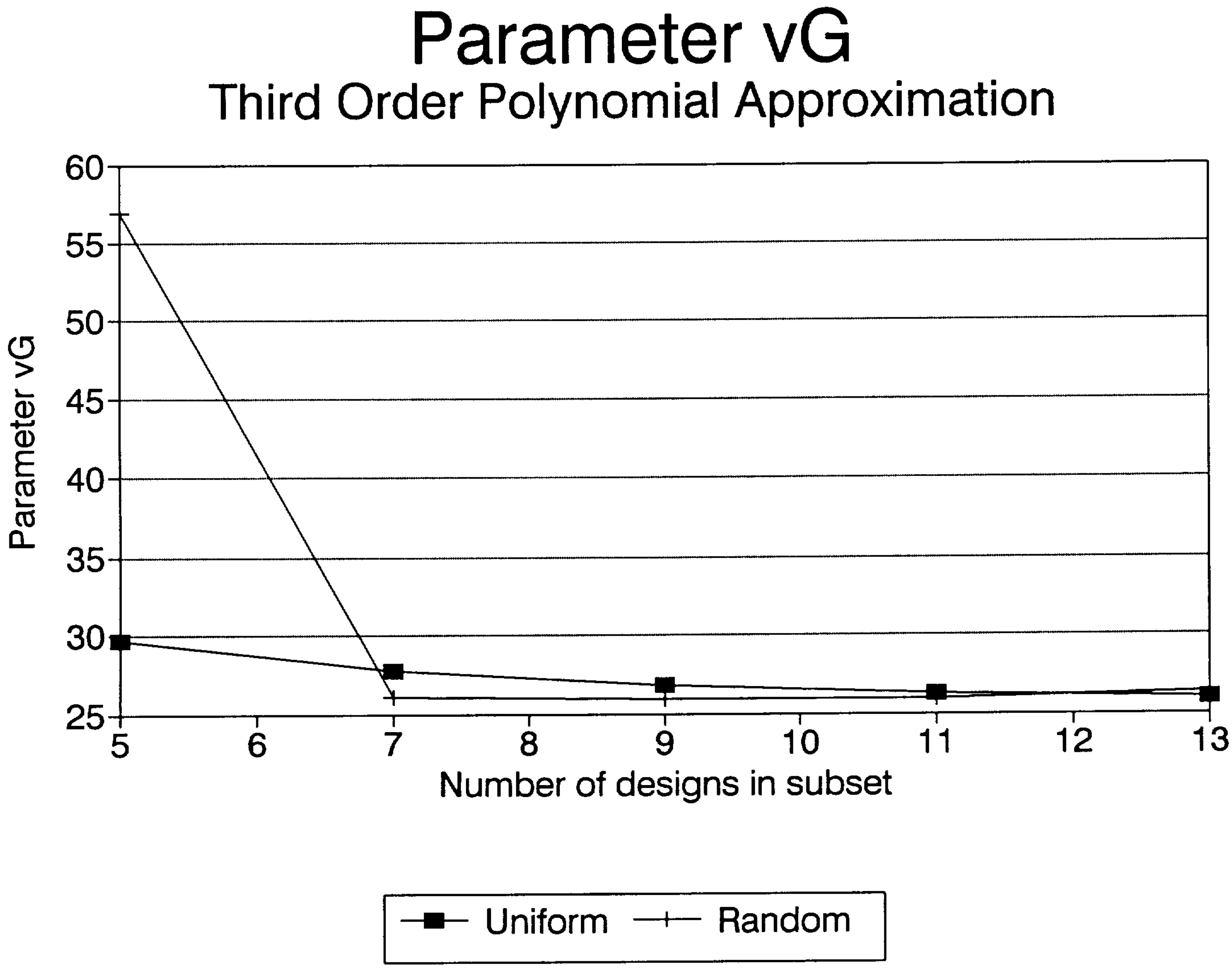

Figure 23. Random points, third order approximation

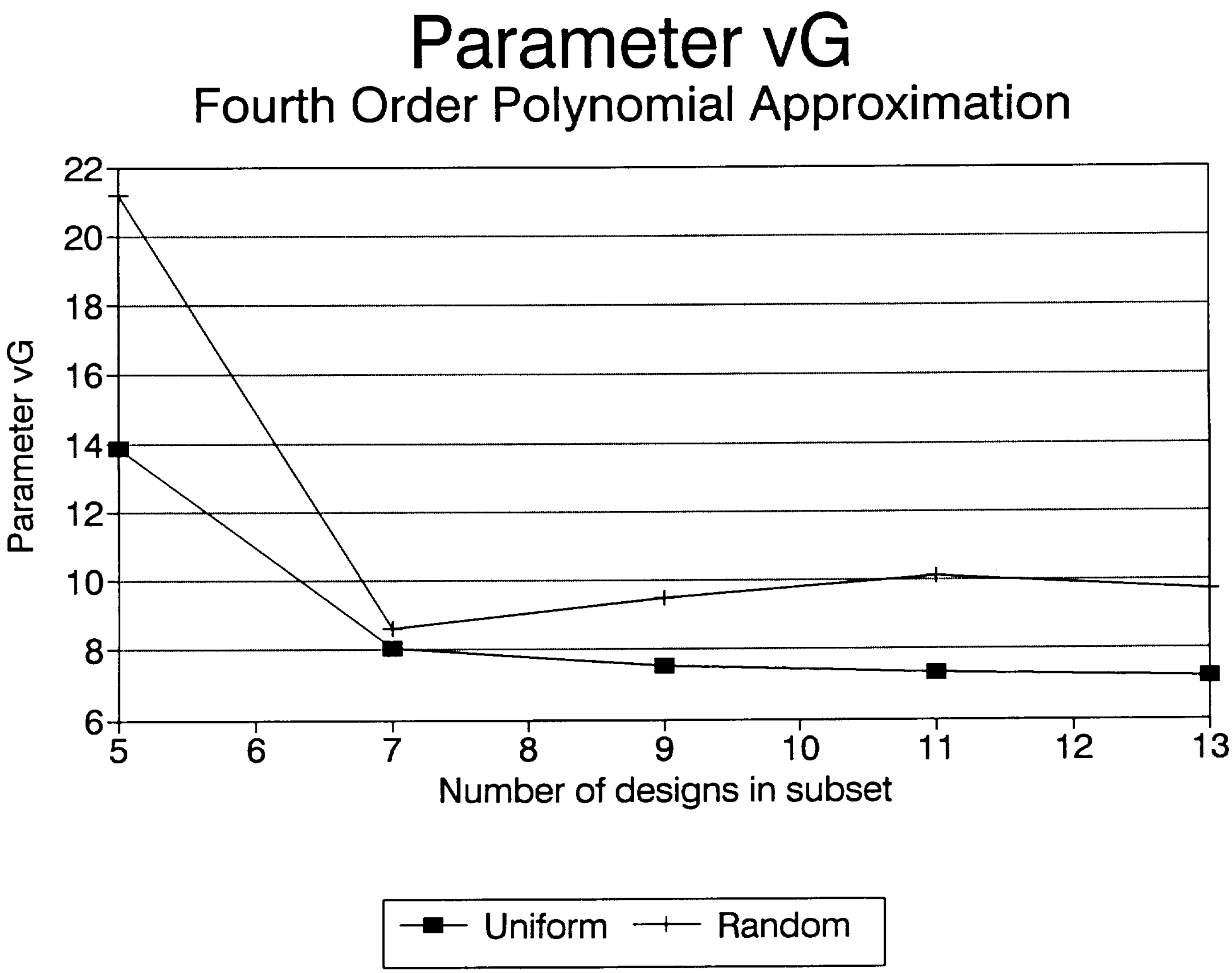

Figure 24. Random points, fourth order approximation

MEASUREMENTS OF QUALITY OF FIT
BEFORE AND AFTER t-test
PERFORMED ON

$$y = 10x_1^4 - 20x_2x_1^2 + 10x_2^2x_1^2 + x_1^2 - 2x_1 + 5$$

"Fox's Banana Function"

SECOND ORDER APPROXIMATION

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_1^2 + b_4x_1x_2 + b_5x_2^2$$

Before t-test	After t-test
ν : 26.8	ν : 41.8
V_G : 102.11	V_G : 175.82
Solution of Coefficients	Solution of Coefficients
$b = \begin{Bmatrix} 121.2 \\ -836.3 \\ 66.7 \\ 393.9 \\ -100 \\ 10 \end{Bmatrix}$	$b = \begin{Bmatrix} 0 \\ -814.0 \\ 0 \\ 352.6 \\ 0 \\ 0 \end{Bmatrix}$

Figure 25. Significance testing, Example 1, 2nd order approximation

MEASUREMENTS OF QUALITY OF FIT
BEFORE AND AFTER t-test
PERFORMED ON

$$y = 10x_1^4 - 20x_2x_1^2 + 10x_2^2x_1^2 + x_1^2 - 2x_1 + 5$$

"Fox's Banana Function"

THIRD ORDER APPROXIMATION

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_1^2 + b_4x_1x_2 + b_5x_2^2 + b_6x_1^3 + b_7x_1^2x_2 + b_8x_1x_2^2 + b_9x_2^3$$

Before t-test

ν: 2.9

V_G: 53.71

Solution of Coefficients

$$b = \begin{Bmatrix} -12.1 \\ 283.7 \\ 0 \\ -306.1 \\ 0 \\ 10 \\ 100 \\ -20 \\ 0 \\ 0 \end{Bmatrix}$$

After t-test

ν: 6.4

V_G: 112.38

Solution of Coefficients

$$b = \begin{Bmatrix} 0 \\ 385.0 \\ 0 \\ -349.3 \\ 0 \\ 0 \\ 103.8 \\ -17.2 \\ 0 \\ 0 \end{Bmatrix}$$

Figure 26. Significance testing, Example 1, 3rd order approximation

MEASUREMENTS OF QUALITY OF FIT
BEFORE AND AFTER t-test
PERFORMED ON

$$Y=(4+x_1)^3+\sin\left[\frac{3\pi}{2}*(x_1+1)\right]+2+x_2^4+\sin\left(\frac{\pi}{2}\right)+7x_2x_1$$

SECOND ORDER APPROXIMATION

$$Y=b_0+b_1x_1+b_2x_2+b_3x_1^2+b_4x_1x_2+b_5x_2^2$$

Before t-test

ν : 6.2

V_G : 90.02

Solution of Coefficients

$$b=\begin{Bmatrix} 97.6 \\ 35.0 \\ -108.4 \\ 19.4 \\ 7 \\ 44.3 \end{Bmatrix}$$

After t-test

ν : 8.6

V_G : 123.67

Solution of Coefficients

$$b=\begin{Bmatrix} 96.4 \\ 0 \\ -90.9 \\ 29.0 \\ 0 \\ 44.3 \end{Bmatrix}$$

Figure 27. Significance testing, Example 2, 2nd order approximation

MEASUREMENTS OF QUALITY OF FIT
BEFORE AND AFTER t-test
PERFORMED ON

$$Y = (4 + x_1)^3 + \sin\left[\frac{3\pi}{2} * (x_1 + 1)\right] + 2 + x_2^4 + \sin\left(\frac{\pi}{2}\right) + 7 x_2 x_1$$

THIRD ORDER APPROXIMATION

$$Y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_1^2 + b_4 x_1 x_2 + b_5 x_2^2 + b_6 x_1^3 + b_7 x_1^2 x_2 + b_8 x_1 x_2^2 + b_9 x_2^3$$

Before t-test

ν: 0.7

V_G: 27.87

Solution of Coefficients

$$b = \begin{Bmatrix} 64.1 \\ 50.7 \\ 28.6 \\ 10.8 \\ 7 \\ -30.7 \\ 1.2 \\ 0 \\ 0 \\ 10 \end{Bmatrix}$$

After t-test

ν: 0.7

V_G: 29.92

Solution of Coefficients

$$b = \begin{Bmatrix} 64.1 \\ 50.8 \\ 28.6 \\ 10.8 \\ 7 \\ -30.7 \\ 0 \\ 0 \\ 0 \\ 10 \end{Bmatrix}$$

Figure 28. Significance testing, Example 2, 3rd order approximation

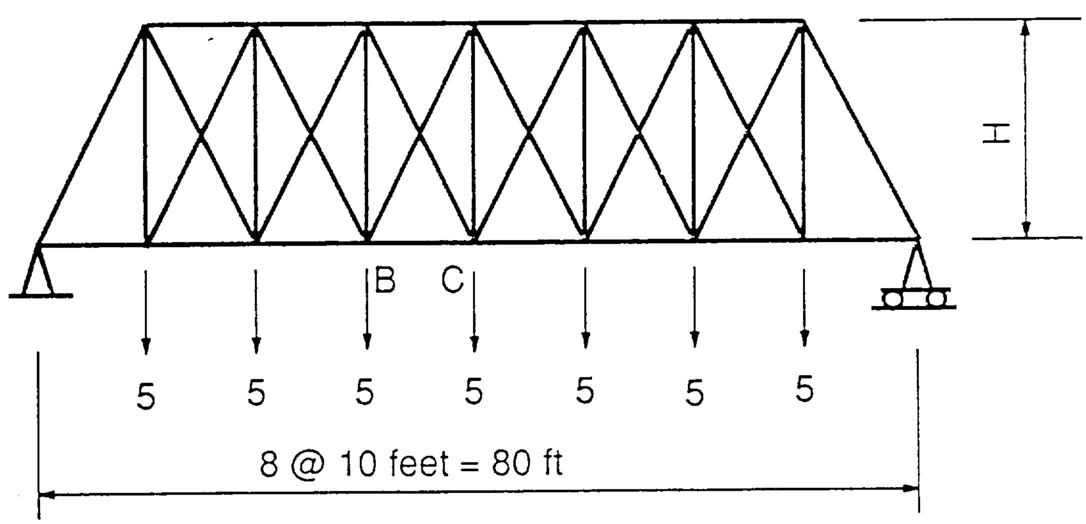

Figure 29. The 35 bar truss

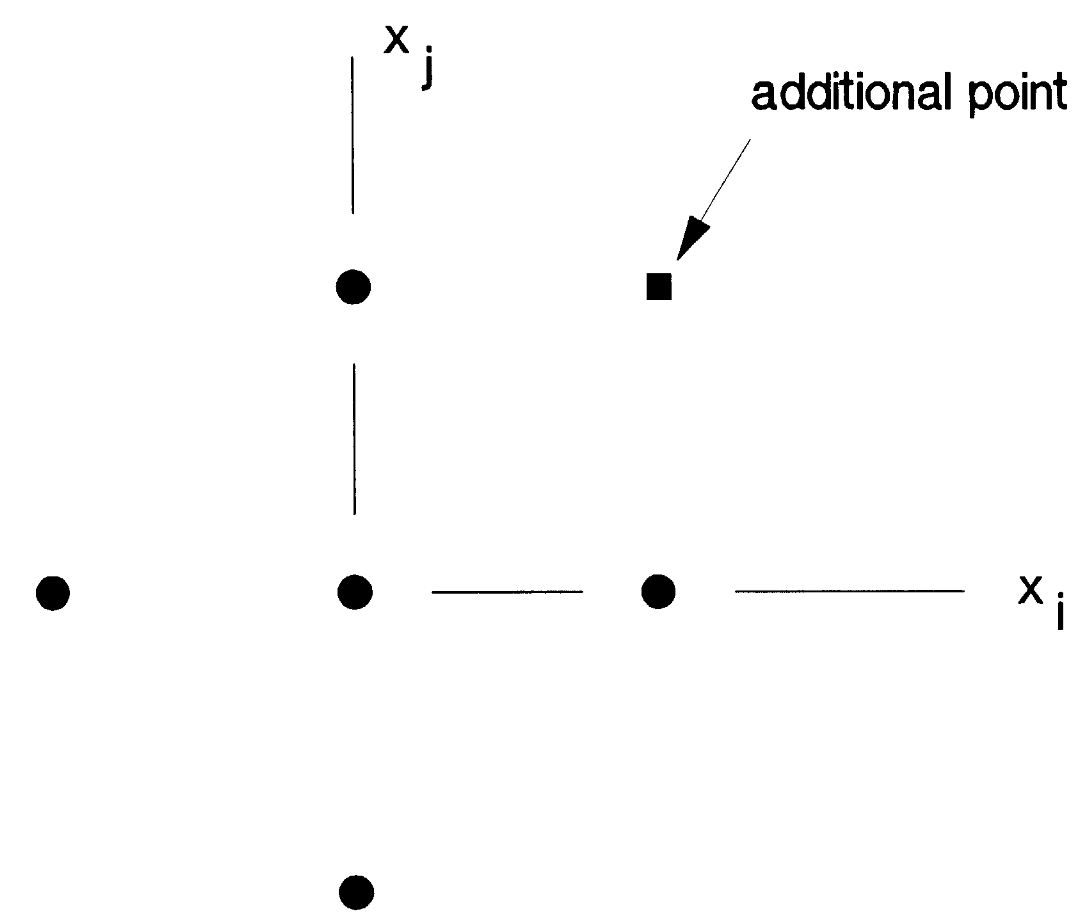

Figure 30. Additional points to complete a second order design

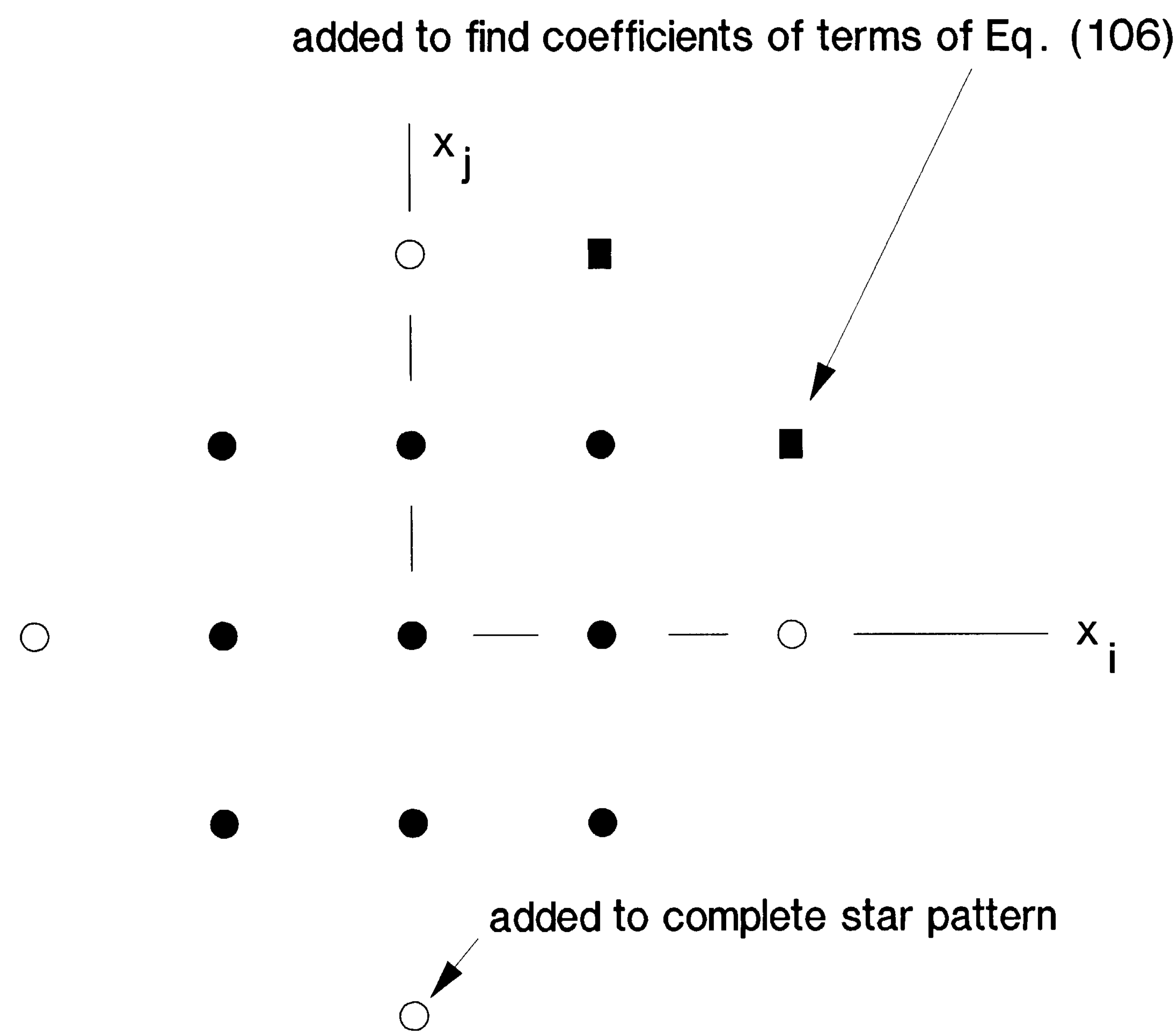

Figure 31. Additional points to complete a fourth order design

Appendix 1
Program DESIGNS

```
      PROGRAM DESIGNS
C
C     PROGRAM TO GENERATE DESIGNS FOR 2ND ORDER POLYNOMIAL
C     PROGRAM DIMENSIONED FOR UP TO 20 VARIABLES
C     RESULTS TO SCREEN AND TO FILE designs.res
C     DESIGN IN GLOBAL COORDINATES TO FILE designs.run
C
C     DEFINITIONS
C     N          = NUMBER OF DESIGN VARIABLES
C     M          = NUMBER OF RANDOM DESIGNS POINTS
C
      DIMENSION X(2000,20)
      DIMENSION XBB(20),XBE(20),A(20),B(20)
    1 FORMAT(I5,6F10.6)
    2 FORMAT(' PROGRAM GENERATES DESIGNS FOR FITTING 2ND ORDER',
     X' POLYNOMIAL')
    3 FORMAT(' ENTER NUMBER OF DESIGN VARIABLES')
    4 FORMAT(' NUMBER OF DESIGN VARIABLES = N =', I3)
   11 FORMAT(6F10.6)
      OPEN(UNIT=7,FILE='designs.res')
      OPEN(UNIT=8,FILE='designs.run')
      WRITE(6,2)
      WRITE(6,3)
      READ(5,*)N
      WRITE(6,4)N
C     SET UP TERMS
      NP1=N+1
      NM1=N-1
      M=(N*N+3*N+2)/2
      MP1=M+1
C
C     ZERO DESIGN MATRIX
      DO100I=1,M
      DO100J=1,N
  100 X(I,J)=0.
      II=0
C....................................
C
C     GENERATE THE FIRST N+1 POINTS FOR FITTING A LINEAR FUNCTION
C     THE FIRST POINT IS WHEN ALL X'S ZERO, ALREADY DONE
C     GENERATE NEXT N POINTS
      DO101I=1,N
      II=I+1
  101 X(II,I)=1.
C
C....................................
C     GENERATE NEXT N POINTS
C    THE 2N+1 POINTS THUS GENERATED WILL ALLOW ADDING SQUARED TERMS
C
      DO102I=1,N
      II=I+N+1
  102 X(II,I)=-1.
C
C....................................
C
C     GENERATE NEXT N(N-1)/2 POINTS
C     THE (N*N+3*N+2)/2 POINTS THUS GENERATED WILL ALOW ADDING CROSS
C     PRODUCT TERMS.  WE WILL THEN HAVE COMPLETE 2ND ORDER POLYNOMIAL
C     APPROXIMATION
C
```

```
      ILAST=2*N+1
      IDO=N-1
      J=1
      JJ=2
  103 CONTINUE
      DO104I=1,IDO
      II=I+ILAST
      X(II,J)=1.
      X(II,JJ)=1.
      JJ=JJ+1
  104 CONTINUE
      ILAST=ILAST+IDO
      IDO=IDO-1
      J=J+1
      JJ=J+1
      IF(J.LE.NM1)GOTO103
C
C     IF WE GOT HERE WE HAVE DEVELOPED THE MINIMUM POINT DESIGN
C
      WRITE(6,*)' WE HAVE GENERATED ',II,' POINTS IN THE MIN PT DESIGN'
      WRITE(7,*)' WE HAVE GENERATED ',II,' POINTS IN THE MIN PT DESIGN'
      WRITE(6,*)' DESIGN POINTS WRITTEN TO FILE designs.res'
C
C     DEVELOP DESIGN POINTS TO AUGMENT THE MINIMUM POINT DESIGN
C     READ IN THE NUMBER OF RANDOM DESIGN POINTS TO BE DEVELOPED
      WRITE(6,*)' ENTER THE NUMBER OF RANDOM GENERATED DESIGN PTS',
     X' DESIRED=M'
      READ(5,*)M
      WRITE(6,*)' NUMBER OF RANDOM DESIGN POINTS M=',M
      WRITE(7,*)' NUMBER OF RANDOM DESIGN POINTS M=',M
      WRITE(6,*)' IFLAG IS ANY POSITIVE INTEGER USED TO START RANDOM',
     X' PROCESS'
      WRITE(6,*)' ENTER IFLAG'
      READ(5,*)IFLAG
      WRITE(6,*)' IFLAG=',IFLAG
      WRITE(7,*)' IFLAG=',IFLAG
      DO850I=1,M
      II=II+1
      DO851J=1,N
      IFLAG=IFLAG+1
      XDUM=RAND(IFLAG)
      X(II,J)=2.*XDUM-1.
  851 CONTINUE
  850 CONTINUE
C
C     IF WE GOT HERE WE HAVE FINISHED GENERATING THE RANDOM DESIGN PTS
      WRITE(6,*)' RANDOM DESIGN POINTS WRITTEN TO FILE designs.res'
C
C     PRINT OUT THE MINIMUM POINT MATRIX IN LOCAL COORDINATES
C
      WRITE(7,*)' DESIGN MATRIX IN LOCAL COORDINATES'
      ITOTAL=II
      DO700I=1,ITOTAL
      WRITE(7,1)I,(X(I,J),J=1,N)
  700 CONTINUE
C     SEE IF WE ARE TO GENERATE DESIGNS IN GLOBAL COORDINATES
C
      WRITE(6,*)' ITEST=1 IF DESIGN POINTS ARE TO BE IN GLOBAL',
     X' COORDINATES'
      WRITE(6,*)' OTHERWISE, ITEST=0'
```

```
      WRITE(6,*)' ENTER ITEST'
      READ(5,*)ITEST
      IF(ITEST.NE.1)GOTO860
C     IF WE GOT HERE WE ARE TO GENERATE DESIGNS IN GLOBAL COORDINATES
      WRITE(6,*)' ENTER LOWER AND UPPER RANGE ON EACH DESIGN VARIABLE'
      WRITE(6,*)' i.e. ENTER XBB(I) TO XBE(I)'
      DO861I=1,N
      READ(5,*)XBB(I),XBE(I)
      WRITE(6,*)' I,XBB(I),XBE(I)=',I,XBB(I),XBE(I)
      WRITE(7,*)' I,XBB(I),XBE(I)=',I,XBB(I),XBE(I)
  861 CONTINUE
      GOTO862
  860 CONTINUE
C
C     IF WE GOT HERE LOWER BOUND VARIABLE IN GLOBAL COORDINATES IS -1
C     IF WE GOT HERE UPPER BOUND VARIABLE IN GLOBAL COORDINATES IS  1
      DO863I=1,N
      XBB(I)=-1.
      XBE(I)=1.
  863 CONTINUE
  862 CONTINUE
      WRITE(7,*)' I,XBB(I),XBE(I),A(I),B(I)'
      DO1301I=1,N
      A(I)=(XBE(I)-XBB(I))/2.
      B(I)=(XBE(I)+XBB(I))/2.
      WRITE(7,*)I,XBB(I),XBE(I),A(I),B(I)
 1301 CONTINUE
      DO1202I=1,ITOTAL
      DO1202J=1,N
 1202 X(I,J)=A(J)*X(I,J)+B(J)
      WRITE(6,*)' DESIGN IN GLOBAL COORDINATES WRITEN TO designs.res'
      WRITE(6,*)' DESIGN IN GLOBAL COORDINATES WRITEN TO designs.run'
      WRITE(7,*)' DESIGN IN GLOBAL COORDINATES'
      WRITE(8,*)ITOTAL
      DO970I=1,ITOTAL
      WRITE(7,1)I,(X(I,J),J=1,N)
      WRITE(8,11)(X(I,J),J=1,N)
  970 CONTINUE
      STOP
      END
```

Appendix 2
Program DESIGN4

```
      PROGRAM DESIGN4
C
C     PROGRAM TO GENERATE DESIGNS FOR 4TH ORDER POLYNOMIAL
C     PROGRAM DIMENSIONED FOR UP TO 6 VARIABLES
C     RESULTS TO SCREEN AND TO FILE design4.res
C     DESIGN IN GLOBAL COORDINATES TO FILE design4.run
C
C     DEFINITIONS
C     N          = NUMBER OF DESIGN VARIABLES
C     M          = NUMBER OF RANDOM DESIGNS POINTS
C
      DIMENSION X(2000,6)
      DIMENSION XBB(10),XBE(10),A(10),B(10)
    1 FORMAT(I5,6F10.6)
    2 FORMAT(' PROGRAM GENERATES DESIGNS FOR FITTING 4TH ORDER',
     X' POLYNOMIAL')
    3 FORMAT(' ENTER NUMBER OF DESIGN VARIABLES')
    4 FORMAT(' NUMBER OF DESIGN VARIABLES = N =', I3)
   11 FORMAT(6F10.6)
      OPEN(UNIT=7,FILE='design4.res')
      OPEN(UNIT=8,FILE='design4.run')
      WRITE(6,2)
      WRITE(6,3)
      READ(5,*)N
      WRITE(6,4)N
      IF(N.EQ.6)GOTO601
      IF(N.EQ.5)GOTO501
      IF(N.EQ.4)GOTO401
      IF(N.EQ.3)GOTO301
      IF(N.EQ.2)GOTO201
      IF(N.EQ.1)GOTO101
      WRITE(6,*)' PROGRAM CAN NOT DO MORE THAN 6 DESIGN VARIABLES'
      WRITE(7,*)' PROGRAM CAN NOT DO MORE THAN 6 DESIGN VARIABLES'
      STOP
C
C     DEVELOP 3 FACTORIAL DESIGN TO GET 4 DESIGN VARIABLE PRODUCT TERMS
  101 CONTINUE
      II=0
      DO100I1=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
  100 CONTINUE
      GOTO701
  201 CONTINUE
      II=0
      DO200I1=1,101,50
      DO200I2=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
      X(II,2)=FLOAT(I2-51)/100.
  200 CONTINUE
      GOTO701
  301 CONTINUE
      II=0
      DO300I1=1,101,50
      DO300I2=1,101,50
      DO300I3=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
      X(II,2)=FLOAT(I2-51)/100.
```

```
      X(II,3)=FLOAT(I3-51)/100.
  300 CONTINUE
      GOTO701
  401 CONTINUE
      II=0
      DO400I1=1,101,50
      DO400I2=1,101,50
      DO400I3=1,101,50
      DO400I4=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
      X(II,2)=FLOAT(I2-51)/100.
      X(II,3)=FLOAT(I3-51)/100.
      X(II,4)=FLOAT(I4-51)/100.
  400 CONTINUE
      GOTO701
  501 CONTINUE
      II=0
      DO500I1=1,101,50
      DO500I2=1,101,50
      DO500I3=1,101,50
      DO500I4=1,101,50
      DO500I5=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
      X(II,2)=FLOAT(I2-51)/100.
      X(II,3)=FLOAT(I3-51)/100.
      X(II,4)=FLOAT(I4-51)/100.
      X(II,5)=FLOAT(I5-51)/100.
  500 CONTINUE
      GOTO701
C
  601 CONTINUE
      II=0
      DO600I1=1,101,50
      DO600I2=1,101,50
      DO600I3=1,101,50
      DO600I4=1,101,50
      DO600I5=1,101,50
      DO600I6=1,101,50
      II=II+1
      X(II,1)=FLOAT(I1-51)/100.
      X(II,2)=FLOAT(I2-51)/100.
      X(II,3)=FLOAT(I3-51)/100.
      X(II,4)=FLOAT(I4-51)/100.
      X(II,5)=FLOAT(I5-51)/100.
      X(II,6)=FLOAT(I6-51)/100.
  600 CONTINUE
      GOTO701
  701 CONTINUE
C
C     ENTER REST OF POINTS IN THE STAR FORMATION
C
      DO702I=1,N
      II=II+1
      DO703J=1,N
  703 X(II,J)=0.
      X(II,I)=1.
  702 CONTINUE
      DO704I=1,N
```

```
      II=II+1
      DO705J=1,N
  705 X(II,J)=0.
      X(II,I)=-1.
  704 CONTINUE
C
C     ENTER TERMS TO CALCULATE COEFFICIENT ASSOCIATED WITH THE TERM
C     X(I)**3*X(J)
C
      NM1=N-1
      IDO=N-1
      J=1
      JJ=2
  803 CONTINUE
      DO804I=1,IDO
      II=II+1
      X(II,J)=1.
      X(II,JJ)=.5
      II=II+1
      X(II,J)=.5
      X(II,JJ)=1.
      JJ=JJ+1
  804 CONTINUE
      IDO=IDO-1
      J=J+1
      JJ=J+1
      IF(J.LE.NM1)GOTO803
C
C     IF WE GOT HERE WE HAVE DEVELOPED THE MINIMUM POINT DESIGN
C
      WRITE(6,*)' WE HAVE GENERATED ',II,' POINTS IN THE MIN PT DESIGN'
      WRITE(7,*)' WE HAVE GENERATED ',II,' POINTS IN THE MIN PT DESIGN'
      WRITE(6,*)' DESIGN POINTS WRITTEN TO FILE design4.res'
C
C     DEVELOP DESIGN POINTS TO AUGMENT THE MINIMUM POINT DESIGN
C     READ IN THE NUMBER OF RANDOM DESIGN POINTS TO BE DEVELOPED
      WRITE(6,*)' ENTER THE NUMBER OF RANDOM GENERATED DESIGN PTS',
     X' DESIRED=M'
      READ(5,*)M
      WRITE(6,*)' NUMBER OF RANDOM DESIGN POINTS M=',M
      WRITE(7,*)' NUMBER OF RANDOM DESIGN POINTS M=',M
      WRITE(6,*)' IFLAG IS ANY POSITIVE INTEGER USED TO START RANDOM',
     X' PROCESS'
      WRITE(6,*)' ENTER IFLAG'
      READ(5,*)IFLAG
      WRITE(6,*)' IFLAG=',IFLAG
      WRITE(7,*)' IFLAG=',IFLAG
      DO850I=1,M
      II=II+1
      DO851J=1,N
      IFLAG=IFLAG+1
      XDUM=RAND(IFLAG)
      X(II,J)=2.*XDUM-1.
  851 CONTINUE
  850 CONTINUE
C
C     IF WE GOT HERE WE HAVE FINISHED GENERATING THE RANDOM DESIGN PTS
      WRITE(6,*)' RANDOM DESIGN POINTS WRITTEN TO FILE design4.res'
C
C     PRINT OUT THE MINIMUM POINT MATRIX IN LOCAL COORDINATES
```

```
C
      WRITE(7,*)' DESIGN MATRIX IN LOCAL COORDINATES'
      ITOTAL=II
      DO700I=1,ITOTAL
      WRITE(7,1)I,(X(I,J),J=1,N)
  700 CONTINUE
C     SEE IF WE ARE TO GENERATE DESIGNS IN GLOBAL COORDINATES
C
      WRITE(6,*)' ITEST=1 IF DESIGN POINTS ARE TO BE IN GLOBAL',
     X' COORDINATES'
      WRITE(6,*)' OTHERWISE, ITEST=0'
      WRITE(6,*)' ENTER ITEST'
      READ(5,*)ITEST
      IF(ITEST.NE.1)GOTO860
C     IF WE GOT HERE WE ARE TO GENERATE DESIGNS IN GLOBAL COORDINATES
      WRITE(6,*)' ENTER LOWER AND UPPER RANGE ON EACH DESIGN VARIABLE'
      WRITE(6,*)' i.e. ENTER XBB(I) TO XBE(I)'
      DO861I=1,N
      READ(5,*)XBB(I),XBE(I)
      WRITE(6,*)' I,XBB(I),XBE(I)=',I,XBB(I),XBE(I)
      WRITE(7,*)' I,XBB(I),XBE(I)=',I,XBB(I),XBE(I)
  861 CONTINUE
      GOTO862
  860 CONTINUE
C
C     IF WE GOT HERE LOWER BOUND VARIABLE IN GLOBAL COORDINATES IS -1
C     IF WE GOT HERE UPPER BOUND VARIABLE IN GLOBAL COORDINATES IS  1
      DO863I=1,N
      XBB(I)=-1.
      XBE(I)=1.
  863 CONTINUE
  862 CONTINUE
      WRITE(7,*)' I,XBB(I),XBE(I),A(I),B(I)'
      DO1301I=1,N
      A(I)=(XBE(I)-XBB(I))/2.
      B(I)=(XBE(I)+XBB(I))/2.
      WRITE(7,*)I,XBB(I),XBE(I),A(I),B(I)
 1301 CONTINUE
      DO1202I=1,ITOTAL
      DO1202J=1,N
 1202 X(I,J)=A(J)*X(I,J)+B(J)
      WRITE(6,*)' DESIGN IN GLOBAL COORDINATES WRITEN TO design4.res'
      WRITE(6,*)' DESIGN IN GLOBAL COORDINATES WRITEN TO design4.run'
      WRITE(7,*)' DESIGN IN GLOBAL COORDINATES'
      WRITE(8,*)ITOTAL
      DO970I=1,ITOTAL
      WRITE(7,1)I,(X(I,J),J=1,N)
      WRITE(8,11)(X(I,J),J=1,N)
  970 CONTINUE
      STOP
      END
```

Appendix 3
Program NEWPSI

```
      PROGRAM newpsi
c
c     *************************************************************
c     *************************************************************
c
c     the program develops a polynomial approximation which
c     may be either under, exactly, or over determined
c     it can handle up to 15 design variables as programmed.
c     The order of polynomial it can handle is as follows:
c     1.  one one design variable, up to a 20th order polynomial
c     2.  two design variables, up to 5th order polynomial
c     3.  for 2-15 design variables, a 2nd order polynomial
c     One can use up to 250 designs to train the approximation.
c     It can handle up to 2000 grid points
c
c     ************************************************************
c     ************************************************************
c
      IMPLICIT REAL*8 (A-H,O-Z)
      dimension x(250,15),y(250),a(250,136)
      dimension yhat(250)
      dimension b(136)
      dimension xx(2000,15),yy(2000),abig(2000,136)
      dimension yyhat(2000)
    1 FORMAT(9F8.4)
    2 FORMAT(3F12.6)
    3 FORMAT(F10.6,1H,,F10.6,1H,,F10.6,1H,,F10.6,1H,,F10.6,1H,,F10.6,
     X1H,,F10.6)
      OPEN(UNIT=5,FILE='newpsi.dat')
      OPEN(UNIT=7,FILE='newpsi.res')
      OPEN(UNIT=8,FILE='newpsi.plot')
C
c     **************************************************************
c
C     read in data
c
c     read in the print code
      read(5,*)ip
c
c     enter number of design variables, nd
      read(5,*)nd
c
c     enter THE DEGREE OF POLYNOMIAL TO BE CONSIDERED, np
      READ(5,*)np
c
c     ENTER NUMBER OF DESIGNS FOR PROBLEM,M
      READ(5,*)M
c
      write(6,*)' print code ip=',ip
      write(6,*)' number of design variables, nd=',nd
```

```
      write(6,*)' degree of polynomial being considered=np=',np
      write(6,*)' number of designs m=',m
      write(7,*)' print code ip=',ip
      write(7,*)' number of design variables, nd=',nd
      write(7,*)' degree of polynomial being considered=np=',np
      write(7,*)' number of designs m=',m
c
c     read in designs and set up matrix a
c
      write(7,*)' x(i,j),y(i)'
      DO101I=1,M
      read(5,*)(x(i,j),j=1,nd),y(i)
      write(7,*)(x(i,j),j=1,nd),y(i)
  101 continue
c
c     set up the coefficient matrix, a, in the matrix equation
c     y=a x
c
      call geta(ip,m,nd,np,n,x,a)
c
c     SEE WHETHER SYSTEM IS UNDER,EXACTLY, OR OVER DETERMINED
c
      IF(M.GE.N)GOTO400
c     IF WE GOT HERE WE ARE UNDER-DETERMINED
      WRITE(6,*)' SYSTEM IS UNDER-DETERMINED'
      WRITE(7,*)' SYSTEM IS UNDER-DETERMINED'
      CALL PSI(ip,M,N,A,Y,B)
      GOTO402
  400 CONTINUE
      IF(M.GT.N)GOTO401
c     IF WE GOT HERE WE ARE EXACTLY DETERMINED
      WRITE(6,*)' SYSTEM IS EXACTLY DETERMINED'
      WRITE(7,*)' SYSTEM IS EXACTLY DETERMINED'
      CALL EXACT(ip,M,A,Y,B)
      GOTO402
  401 CONTINUE
c     IF WE GOT HERE WE ARE OVER-DETERMINED
      WRITE(6,*)' SYSTEM IS OVER-DETERMINED'
      WRITE(7,*)' SYSTEM IS OVER-DETERMINED'
      CALL OVER(ip,M,N,A,Y,B)
  402 CONTINUE
c
c
c     EVALUATE APPROXIMATION AT DESIGNS
c
      WRITE(6,*)' MATRIX OF COEFFICIENTS, B(I)'
      WRITE(7,*)' MATRIX OF COEFFICIENTS, B(I)'
      WRITE(6,*)(B(I),I=1,N)
      WRITE(7,*)(B(I),I=1,N)
      WRITE(7,*)' MATRICES Y(I) AND YHAT(I)'
c
c     recalculate matrix a
      call geta(ip,m,nd,np,n,x,a)
c
c     calculate approximation at designs and print results
c
      write(7,*)' y(i),yhat(i)'
      DO102I=1,M
      YHAT(I)=0.
      DO103J=1,N
```

```
      yhat(i)=yhat(i)+a(i,j)*b(j)
  103 CONTINUE
      WRITE(7,*)Y(I),YHAT(I)
  102 CONTINUE
C
c     evaluate function at grid
      read(5,*)ng
      write(6,*)' number of designs on grid = ngn',ng
      write(7,*)' number of designs on grid = ngn',ng
      write(7,*)' xx(i,j),yy(i)'
      DO601I=1,ng
      read(5,*)(xx(i,j),j=1,nd),yy(i)
      write(7,*)(xx(i,j),j=1,nd),yy(i)
  601 continue
      call getabg(ip,ng,nd,np,n,xx,abig)
      write(7,*)' yy(i),yyhat(i) at grid'
      DO602I=1,ng
      YYHAT(I)=0.
      DO603J=1,N
      yyhat(i)=yyhat(i)+abig(i,j)*b(j)
  603 CONTINUE
      WRITE(7,*)YY(I),YYHAT(I)
c
c     write the plot file
      write(8,*)(xx(i,j),j=1,nd),yyhat(i)
c
  602 CONTINUE
c
c     calculate statistical terms
c
      call statit(m,y,yhat,ng,yy,yyhat)
c
      STOP
      END
      subroutine geta(ip,m,nd,np,n,x,a)
c
c     ************************************************************
c     ************************************************************
c
c     This subroutine generates the matrix a where the matrix
c     equation is y= a b.  Here y are the training functions,
c     b are undetermined coefficients.  The algorithm is programmed
c     to handle
c     1.  any level of approximation for one design variable
c     2.  up to 5th order polynomial in two design variables
c     3.  quadratic approximation in more than two design variabaales
c
c     ************************************************************
c     ************************************************************
c
      IMPLICIT REAL*8 (A-H,O-Z)
      dimension x(250,15),a(250,136)
c
c     do for each design
c
      do300i=1,m
c
c     *******************************************************
c
c     if nd is not equal to 1 go to 400
```

```
      if(nd.ne.1)goto400
c
c     ********************************************************
c     ********************************************************
c
c     here we have nd=1, i.e. one design variable
c     we will develp a's for all np's
c
      a(i,1)=1.
      j=1
      do201k=1,np
      j=j+1
      a(i,j)=x(i,1)**k
  201 continue
      n=np+1
      goto301
c
c
c     ************************************
  400 continue
c
c     if nd is not equal to 2 go to 500
      if(nd.ne.2)goto500
c
c     **************************************************************
c     **************************************************************
c
c     if we got here we have 2 design variables
c
      x1=x(i,1)
      x2=x(i,2)
c
c     *********************
c
c     add the constant and linear terms
c
      a(i,1)=1.
      a(i,2)=x1
      a(i,3)=x2
      n=3
      if(np.lt.2)goto301
c
c     ********************
c
c     add the 2nd order terms
c
      a(i,4)=x1**2
      a(i,5)=x1*x2
      a(i,6)=x2**2
      n=6
      if(np.lt.3)goto301
c
c     *******************
c
c     add the cubic terms
c
      a(i,7)=x1**3
      a(i,8)=x1**2*x2
      a(i,9)=x1*x2**2
      a(i,10)=x2**3
```

```
      n=10
      if(np.lt.4)goto301
c
c     ******************
c
c     add the 4th order terms
c
      a(i,11)=x1**4
      a(i,12)=x1**3*x2
      a(i,13)=x1**2*x2**2
      a(i,14)=x1*x2**3
      a(i,15)=x2**4
      n=15
      if(np.lt.5)goto301
c
c     ******************
c
c     add the 5th order terms
c
      a(i,16)=x1**5
      a(i,17)=x1**4*x2
      a(i,18)=x1**3*x2**2
      a(i,19)=x1**2*x2**3
      a(i,20)=x1*x2**4
      a(i,21)=x2**5
      n=21
      if(np.lt.6)goto301
c
c     *****************
c
c     algorithm not programed for polynomials of order larger than 5
c
      write(6,*)' for two design variables, algorithm not programed for'
      write(6,*)' polynomials of order larger than 5'
      write(7,*)' for two design variables, algorithm not programed for'
      write(7,*)' polynomials of order larger than 5'
      stop
c
c     ******************************************************
c     ******************************************************
c
  500 continue
c
c     if we got here number of design variables >2
c
c     *******************
c
c     enter constant and linear terms
c
      a(i,1)=1.
      j=1
      do501k=1,nd
      j=j+1
      a(i,j)=x(i,k)
  501 continue
      n=j
      if(np.lt.2)goto301
c
c     *******************
c
```

```
c     enter the quadratic terms
c
      do502k=1,nd
      do502L=k,nd
      j=j+1
      a(i,j)=x(i,k)*x(i,L)
  502 continue
      n=j
      if(np.lt.3)goto301
c
c     ********************
c
c     algorithm not programmed for more than quadratic approximation
c     when number of design variables >2
c
      write(6,*)' algorithm not programmed for more than quadratic'
      write(6,*)' approximation when number of design variables >2'
      write(7,*)' algorithm not programmed for more than quadratic'
      write(7,*)' approximation when number of design variables >2'
      stop
c
c     ***********************************************************
c     ***********************************************************
c
c     print out some results
c
  301 continue
      if(ip.lt.4)goto302
      write(6,*)' a(i,j)',(a(i,j),j=1,n)
      write(6,*)' '
      write(7,*)' a(i,j)',(a(i,j),j=1,n)
      write(7,*)' '
  302 continue
c
c     *******************************************************************
c
  300 continue
      write(6,*)' number of undetermined coef=n=',n
      write(7,*)' number of undetermined coef=n=',n
c
      return
      end
      subroutine getabg(ip,m,nd,np,n,x,a)
c
c     ***********************************************************
c     ***********************************************************
c
c     This subroutine generates the matrix a where the matrix
c     equation is y= a b.  Here y are the training functions,
c     b are undetermined coefficients.  The algorithm is programmed
c     to handle
c     1.  any level of approximation for one design variable
c     2.  up to 5th order polynomial in two design variables
c     3.  quadratic approximation in more than two design variabaales
c
c     ***********************************************************
c     ***********************************************************
c
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION A(2000,136)
```

```
      DIMENSION X(2000,15)
c
c     do for each design
c
      do300i=1,m
c
c     ********************************************************
c
c     if nd is not equal to 1 go to 400
      if(nd.ne.1)goto400
c
c     ********************************************************
c     ********************************************************
c
c     here we have nd=1, i.e. one design variable
c     we will develp a's for all np's
c
      a(i,1)=1.
      j=1
      do201k=1,np
      j=j+1
      a(i,j)=x(i,1)**k
  201 continue
      n=np+1
      goto301
c
c
c     ************************************
  400 continue
c
c     if nd is not equal to 2 go to 500
      if(nd.ne.2)goto500
c
c     ***************************************************************
c     ***************************************************************
c
c     if we got here we have 2 design variables
c
      x1=x(i,1)
      x2=x(i,2)
c
c     *********************
c
c     add the constant and linear terms
c
      a(i,1)=1.
      a(i,2)=x1
      a(i,3)=x2
      n=3
      if(np.lt.2)goto301
c
c     ********************
c
c     add the 2nd order terms
c
      a(i,4)=x1**2
      a(i,5)=x1*x2
      a(i,6)=x2**2
      n=6
      if(np.lt.3)goto301
```

```
c
c     ******************
c
c     add the cubic terms
c
      a(i,7)=x1**3
      a(i,8)=x1**2*x2
      a(i,9)=x1*x2**2
      a(i,10)=x2**3
      n=10
      if(np.lt.4)goto301
c
c     ******************
c
c     add the 4th order terms
c
      a(i,11)=x1**4
      a(i,12)=x1**3*x2
      a(i,13)=x1**2*x2**2
      a(i,14)=x1*x2**3
      a(i,15)=x2**4
      n=15
      if(np.lt.5)goto301
c
c     ******************
c
c     add the 5th order terms
c
      a(i,16)=x1**5
      a(i,17)=x1**4*x2
      a(i,18)=x1**3*x2**2
      a(i,19)=x1**2*x2**3
      a(i,20)=x1*x2**4
      a(i,21)=x2**5
      n=21
      if(np.lt.6)goto301
c
c     ******************
c
c     algorithm not programed for polynomials of order larger than 5
c
      write(6,*)' for two design variables, algorithm not programed for'
      write(6,*)' polynomials of order larger than 5'
      write(7,*)' for two design variables, algorithm not programed for'
      write(7,*)' polynomials of order larger than 5'
      stop
c
c     ******************************************************
c     ******************************************************
c
  500 continue
c
c     if we got here number of design variables >2
c
c     *******************
c
c     enter constant and linear terms
c
      a(i,1)=1.
      j=1
```

```
      do501k=1,nd
      j=j+1
      a(i,j)=x(i,k)
  501 continue
      n=j
      if(np.lt.2)goto301
c
c     ********************
c
c     enter the quadratic terms
c
      do502k=1,nd
      do502L=k,nd
      j=j+1
      a(i,j)=x(i,k)*x(i,L)
  502 continue
      n=j
      if(np.lt.3)goto301
c
c     ********************
c
c     algorithm not programmed for more than quadratic approximation
c     when number of design variables >2
c
      write(6,*)' algorithm not programmed for more than quadratic'
      write(6,*)' approximation when number of design variables >2'
      write(7,*)' algorithm not programmed for more than quadratic'
      write(7,*)' approximation when number of design variables >2'
      stop
c
c     ****************************************************************
c     ***************************************************************
c
c     print out some results
c
  301 continue
      if(ip.lt.4)goto302
      write(6,*)' a(i,j)',(a(i,j),j=1,n)
      write(6,*)' '
      write(7,*)' a(i,j)',(a(i,j),j=1,n)
      write(7,*)' '
  302 continue
c
c     *****************************************************************
c
  300 continue
      write(6,*)' number of undetermined coef=n=',n
      write(7,*)' number of undetermined coef=n=',n
c
      return
      end
      SUBROUTINE PSI(IP,M,N,DUMA,Y,XX)
      IMPLICIT REAL*8 (A-H,O-Z)
C
      DIMENSION DUMa(250,136)
      DIMENSION A(21,21),B(21,21),D(21,21),DI(21,21),BPI(21,21)
      DIMENSION C(21,21),FI(21,21),CPI(21,21),H(21,21),HI(21,21)
      DIMENSION API(21,21)
      DIMENSION F(21,21)
      DIMENSION IPIVOT(21),IWK(21,2)
```

```
      DIMENSION y(250)
      DIMENSION XX(21)
C
C     THIS SUBROUTINE CALCULATES PSEUDO INVERSE OF MATRIX A
C     M     = ROW DIMENSION OF A LESS THAN N
C     N     = COLUMN DIMENSION OF A
C
C     COPY DUMA TO A
C
      DO90I=1,M
      DO90J=1,N
   90 A(I,J)=DUMA(I,J)
C
C
C     PRINT MATRIX A
      if(ip.lt.4)goto50
      WRITE(6,*)' MATRIX A'
      WRITE(7,*)' MATRIX A'
      CALL WRITIT(M,N,A)
   50 continue
C
C     SET UP MATRIX B
C
      DO100I=1,M
      DO100J=1,M
  100 B(I,J)=A(I,J)
      if(ip.lt.4)goto51
      WRITE(6,*)' MATRIX B'
      WRITE(7,*)' MATRIX B'
      CALL WRITIT(M,M,B)
   51 continue
C
C     GET D= B TRAN * B
C
      DO101I=1,M
      DO101J=1,M
      D(I,J)=0.
      DO101K=1,M
  101 D(I,J)=D(I,J)+B(K,I)*B(K,J)
      if(ip.lt.4)goto52
      WRITE(6,*)' MATRIX D'
      WRITE(7,*)' MATRIX D'
      CALL WRITIT(M,M,D)
   52 continue
C
C     GET INVERSE OF D=DI
      MAX=21
      MDUM=0
      IOP=0
      CALL MATINV(MAX,M,D,MDUM,DI,IOP,DETERM,ISCALE,IPIVOT,IWK)
      WRITE(6,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      WRITE(7,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      DO300I=1,M
      DO300J=1,M
  300 DI(I,J)=D(I,J)
      if(ip.lt.4)goto53
      WRITE(6,*)' MATRIX DI'
      WRITE(7,*)' MATRIX DI'
      CALL WRITIT(M,M,DI)
   53 continue
```

```
C
C     GET PSEUDO INVERSE OF B = BPI = DI * B TRANS
C
      DO102I=1,M
      DO102 JQ=1,M
      BPI(I,JQ)=0.
      DO102J=1,M
  102 BPI(I,JQ)=BPI(I,JQ)+DI(I,J)*B(JQ,J)
      if(ip.lt.4)goto54
      WRITE(6,*)' MATRIX BPI'
      WRITE(7,*)' MATRIX BPI'
      CALL WRITIT(M,M,BPI)
   54 continue
C
C     SET UP MATRIX C = A
C
      DO103I=1,M
      DO103J=1,N
  103 C(I,J)=A(I,J)
      if(ip.lt.4)goto55
      WRITE(6,*)' MATRIX C'
      WRITE(7,*)' MATRIX C'
      CALL WRITIT(M,N,C)
   55 continue
C
C     SET UP MATRIX F = C * C TRANS
C
      DO104I=1,M
      DO104J=1,M
      F(I,J)=0.
      DO104K=1,N
  104 F(I,J)=F(I,J)+C(I,K)*C(J,K)
      if(ip.lt.4)goto56
      WRITE(6,*)' MATRIX F'
      WRITE(7,*)' MATRIX F'
      CALL WRITIT(M,M,F)
   56 continue
C
C     GET THE INVERSE OF F = FI
C
      CALL MATINV(MAX,M,F,MDUM,FI,IOP,DETERM,ISCALE,IPIVOT,IWK)
      WRITE(6,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      WRITE(7,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      DO301I=1,M
      DO301J=1,M
  301 FI(I,J)=F(I,J)
      if(ip.lt.4)goto57
      WRITE(6,*)' MATRIX FI'
      WRITE(7,*)' MATRIX FI'
      CALL WRITIT(M,M,FI)
   57 continue
C
C      GET THE PSEUDO INVERSE OF C = CPI = C TRANS * FI
C
      DO105IQ=1,N
      DO105J=1,M
      CPI(IQ,J)=0.
      DO105I=1,M
  105 CPI(IQ,J)=CPI(IQ,J)+C(I,IQ)*FI(I,J)
      if(ip.lt.4)goto58
```

```
      WRITE(6,*)' MATRIX CPI'
      WRITE(7,*)' MATRIX CPI'
      CALL WRITIT(N,M,CPI)
   58 continue
C
C     SET UP MATRIX H = PSEUDO INVERSE OF B = BPI
C
      DO106I=1,M
      DO106J=1,M
  106 H(I,J)=BPI(I,J)
      if(ip.lt.4)goto59
      WRITE(6,*)' MATRIX H'
      WRITE(7,*)' MATRIX H'
      CALL WRITIT(M,M,H)
   59 continue
C
C     GET INVERSE OF H = HI
      CALL MATINV(MAX,M,H,MDUM,HI,IOP,DETERM,ISCALE,IPIVOT,IWK)
      WRITE(6,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      WRITE(7,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      DO302I=1,M
      DO302J=1,M
  302 HI(I,J)=H(I,J)
      if(ip.lt.4)goto60
      WRITE(6,*)' MATRIX HI'
      WRITE(7,*)' MATRIX HI'
      CALL WRITIT(M,M,HI)
   60 continue
C
C     GET PSEUDO INVERSE OF A = API = CPI * HI * BPI
C
      DO107IQ=1,N
      DO107J=1,M
      API(IQ,J)=0.
      DO107I=1,M
      DO107K=1,M
  107 API(IQ,J)=API(IQ,J)++CPI(IQ,I)*HI(I,K)*BPI(K,J)
      if(ip.lt.4)goto61
      WRITE(6,*)' MATRIX API'
      WRITE(7,*)' MATRIX API'
      CALL WRITIT(N,M,API)
   61 continue
C
C    GET XX = API * Y
C
      DO108IQ=1,N
      XX(IQ)=0.
      DO108J=1,M
  108 XX(IQ)=XX(IQ)+API(IQ,J)*Y(J)
      JDUM=1
      if(ip.lt.4)goto62
      WRITE(6,*)' MATRIX XX'
      WRITE(7,*)' MATRIX XX'
      CALL WRITIT(N,JDUM,XX)
   62 continue
C
      RETURN
      END
      SUBROUTINE WRITIT(MM,NN,XX)
      IMPLICIT REAL*8 (A-H,O-Z)
```

```
      DIMENSION XX(21,1)
    1 FORMAT(1X)
    2 FORMAT(10F7.2)
      WRITE(6,1)
      DO100I=1,MM
      WRITE(6,2)(XX(I,J),J=1,NN)
      WRITE(7,2)(XX(I,J),J=1,NN)
  100 CONTINUE
      RETURN
      END
      SUBROUTINE EXACT(IP,M,A,Y,B)
      IMPLICIT REAL*8 (A-H,O-Z)
C
      DIMENSION a(250,136),b(136),y(250)
      DIMENSION IPIVOT(250),IWK(250,2)
      DIMENSION C(136,1)
      DO100I=1,M
  100 C(I,1)=Y(I)
      MAX=250
      MDUM=1
      IOP=0
      CALL MATINV(MAX,M,A,MDUM,C,IOP,DETERM,ISCALE,IPIVOT,IWK)
      WRITE(6,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      WRITE(7,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      DO101I=1,M
      B(I)=C(I,1)
  101 CONTINUE
      if(ip.lt.4)goto50
      WRITE(6,*)' MATRIX B',(B(I),I=1,M)
      WRITE(7,*)' MATRIX B',(B(I),I=1,M)
   50 continue
      RETURN
      END
      SUBROUTINE OVER(IP,M,N,A,Y,B)
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION a(250,136),b(136),y(250)
      DIMENSION IPIVOT(136),IWK(136,2)
      DIMENSION ATA(136,136),ATY(136,1)
      DO200I=1,N
      DO200J=1,N
      ATA(I,J)=0.
      DO200K=1,M
  200 ATA(I,J)=ATA(I,J)+A(K,I)*A(K,J)
      DO201I=1,N
      ATY(I,1)=0.
      DO201K=1,M
  201 ATY(I,1)=ATY(I,1)+A(K,I)*Y(K)
      MAX=136
      MDUM=1
      IOP=0
      CALL MATINV(MAX,N,ATA,MDUM,ATY,IOP,DETERM,ISCALE,IPIVOT,IWK)
      WRITE(6,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      WRITE(7,*)' DETERM=',DETERM,' ISCALE=',ISCALE
      DO101I=1,N
      B(I)=ATY(I,1)
  101 CONTINUE
      if(ip.lt.4)goto50
      WRITE(6,*)' MATRIX B',(B(I),I=1,N)
      WRITE(7,*)' MATRIX B',(B(I),I=1,N)
   50 continue
```

```
      RETURN
      END
      subroutine statit(m,y,yhat,ng,yy,yyhat)
      implicit real*8 (a-h,o-z)
c
c     ***********************************************************
c
c     This subroutine calculates quality of approximation measures
c     this subroutine calculates v, r2, and vg
c
c     ***********************************************************
c
      dimension y(250),yhat(250)
      dimension yy(2000),yyhat(2000)
      yb=0.
      do100id=1,m
      yb=yb+y(id)
  100 continue
      yb=yb/float(m)
      error=0.
      do101id=1,m
      error=error+(y(id)-yhat(id))**2
  101 continue
      v=sqrt(error/float(m))/yb*(100.)
      write(7,*)' error over designs=error = ',error
      write(7,*)' average y over design = yb =',yb
      write(6,*)' coefficient v (as %)= ',v
      write(7,*)' coefficient v (as %)= ',v
      dn=0.
      dd=0.
      do7769id=1,m
      dn=dn+(yhat(id)-yb)**2
      dd=dd+(y(id)-yb)**2
 7769 continue
      r2=dn/dd*(100.)
      write(6,*)' coefficient r2 (as%) = ',r2
      write(7,*)' coefficient r2 (as%) = ',r2
c     get vg
      perror=0.
      yg=0.
      do155id=1,ng
      yg=yg+yy(id)
      perror=perror+(yy(id)-yyhat(id))**2
  155 continue
      yg=yg/float(ng)
      vg=sqrt(perror/float(ng))/yg*(100.)
      write(7,*)' sum of residuals squared=perror=',perror
      write(7,*)' average y over grid = yg =',yg
      write(6,*)' coefficient vg = ',vg
      write(7,*)' coefficient vg = ',vg
      return
      end
      SUBROUTINE MATINV(MAX,N,A,M,B,IOP,DETERM,ISCALE,IPIVOT,IWK)       MATINV 2
      implicit real*8 (a-h,o-z)

C     F1.3                                                              MATINV 3
C***********************************************************************MATINV 4
C                                                                       MATINV 5
C     PURPOSE - MATINV INVERTS A REAL SQUARE MATRIX A.                  MATINV 6
C                IN ADDITION THE ROUTINE SOLVES THE MATRIX              MATINV 7
```

```
C           EQUATION AX=B,WHERE B IS A MATRIX OF CONSTANT
C           VECTORS. THERE IS ALSO AN OPTION TO HAVE THE
C           DETERMINANT EVALUATED. IF THE INVERSE IS NOT
C           NEEDED, USE GELIM TO SOLVE A SYSTEM OF SIMULTANEOUS
C           EQUATIONS AND DETFAC TO EVALUATE A DETERMINANT
C           FOR SAVING TIME AND STORAGE.
C
C   USE       - CALL MATINV(MAX,N,A,M,B,IOP,DETERM,ISCALE,IPIVOT,IWK)
C
C             MAX - THE MAXIMUM ORDER OF A AS STATED IN THE
C                   DIMENSION STATEMENT OF THE CALLING PROGRAM.
C
C             N   - THE ORDER OF A, 1.LE.N.LE.MAX.
C
C             A   - A TWO-DIMENSIONAL ARRAY OF THE COEFFICIENTS.
C                   ON RETURN TO THE CALLING PROGRAM, A INVERSE
C                   IS STORED IN A.
C                   A MUST BE DIMENSIONED IN THE CALLING PROGRAM
C                   WITH FIRST DIMENSION MAX AND SECOND DIMENSION
C                   AT LEAST N.
C
C             M   - THE NUMBER OF COLUMN VECTORS IN B.
C                   M=0 SIGNALS THAT THE SUBROUTINE IS
C                   USED SOLELY FOR INVERSION,HOWEVER,
C                   IN THE CALL STATEMENT AN ENTRY CORRE-
C                   SPONDING TO B MUST BE PRESENT.
C
C             B   - A TWO-DIMENSIONAL ARRAY OF THE CONSTANT
C                   VECTOR B. ON RETURN TO CALLING PROGRAM,
C                   X IS STORED IN B. B SHOULD HAVE ITS FIRST
C                   DIMENSION MAX AND ITS SECOND AT LEAST M.
C
C             IOP - COMPUTE DETERMINANT OPTION.
C                    IOP=0 COMPUTES THE MATRIX INVERSE AND
C                          DETERMINANT.
C                    IOP=1 COMPUTES THE MATRIX INVERSE ONLY.
C
C           DETERM- FOR IOP=0-IN CONJUNCTION WITH ISCALE
C                   REPRESENTS THE VALUE OF THE DETERMINANT
C                   OF A, DET(A),AS FOLLOWS.
C                    DET(A)=(DETERM)(10**100(ISCALE))
C                   THE COMPUTATION DET(A) SHOULD NOT BE
C                   ATTEMPTED IN THE USER PROGRAM SINCE IF
C                   THE ORDER OF A IS LARGER AND/OR THE
C                   MAGNITUDE OF ITS ELEMENTS ARE LARGE(SMALL),
C                   THE DET(A) CALCULATION MAY CAUSE OVERFLOW
C                 (UNDERFLOW). DETERM SET TO ZERO FOR
C                 SINGULAR MATRIX CONDITION, FOR EITHER
C                 IOP=1,OR 0. SHOULD BE CHECKED BY PROGRAMER
C                 ON RETURN TO MAIN PROGRAM.
C
C         ISCALE  - A SCALE FACTOR COMPUTED BY THE
C                   SUBROUTINE TO AVOID OVERFLOW OR
C                   UNDERFLOW IN THE COMPUTATION OF
C                   THE QUANTITY,DETERM.
C
C         IPIVOT  - A ONE DIMENSIONAL INTEGER ARRAY
C                   USED BY THE SUBPROGRAM TO STORE
C                   PIVOTOL INFORMATION. IT SHOULD BE
C                   DIMENSIONED AT LEAST N. IN GENERAL
```

```
C                         THE USER DOES NOT NEED TO MAKE USE                  MATINV68
C                         OF THIS ARRAY.                                      MATINV69
C                                                                             MATINV70
C                  IWK  - A TWO-DIMENSIONAL INTEGER ARRAY OF                  MATINV71
C                         TEMPORARY STORAGE USED BY THE ROUTINE.              MATINV72
C                         IWK SHOULD HAVE ITS FIRST DIMENSION                 MATINV73
C                         MAX, AND ITS SECOND 2.                              MATINV74
C                                                                             MATINV75
C     REQUIRED ROUTINES-                                                      MATINV76
C                                                                             MATINV77
C     REFERENCE         -FOX,L, AN INTRODUCTION TO NUMERICAL                  MATINV78
C                               LINEAR ALGEBRA                                MATINV79
C                                                                             MATINV80
C     STORAGE           - 542 OCTAL LOCATIONS                                 MATINV81
C                                                                             MATINV82
C     LANGUAGE          -FORTRAN                                              MATINV83
C     LIBRARY FUNCTIONS -ABS                                                  MATINV84
C                                                                             MATINV85
C     RELEASED           - JULY 1973                                          MATINV86
C                                                                             MATINV87
C     LATEST REVISION    - JULY 29, 1981                                      MATINV88
C                          COMPUTER SCIECES CORPORATION                       MATINV89
C                          HAMPTON, VA                                        MATINV90
C*****************************************************************************MATINV91
C                                                                             MATINV92
      DIMENSION IPIVOT(N),A(MAX,N),B(MAX,N),IWK(MAX,2)                        MATINV93
      EQUIVALENCE (IROW,JROW), (ICOLUM,JCOLUM), (AMAX, T, SWAP)               MATINV94
C                                                                             MATINV98
C     INITIALIZATION                                                          MATINV99
C                                                                             MATIN100
      ISCALE=0                                                                MATIN101
      R1=(10.0d+00)**32                                                       MATIN102
      R2=1.0d+00/R1                                                           MATIN103
      DETERM=1.0d+00                                                          MATIN104
      DO 20 J=1,N                                                             MATIN105
        IPIVOT(J)=0                                                           MATIN106
 20   CONTINUE                                                                MATIN107
      DO 550 I=1,N                                                            MATIN108
C                                                                             MATIN109
C       SEARCH FOR PIVOT ELEMENT                                              MATIN110
C                                                                             MATIN111
        AMAX=0.0d+00                                                          MATIN112
        DO 105 J=1,N                                                          MATIN113
          IF (IPIVOT(J)-1) 60, 105, 60                                        MATIN114
   60     DO 100 K=1,N                                                        MATIN115
            IF (IPIVOT(K)-1) 80, 100, 740                                     MATIN116
   80       TMAX = ABS(A(J,K))                                                MATIN117
            IF(AMAX-TMAX) 85,100,100                                          MATIN118
   85       IROW=J                                                            MATIN119
            ICOLUM=K                                                          MATIN120
            AMAX=TMAX                                                         MATIN121
  100     CONTINUE                                                            MATIN122
  105   CONTINUE                                                              MATIN123
        IF (AMAX) 740,106,110                                                 MATIN124
  106   DETERM=0.0d+00                                                        MATIN125
        ISCALE=0                                                              MATIN126
        GO TO 740                                                             MATIN127
  110   IPIVOT(ICOLUM) = 1                                                    MATIN128
C                                                                             MATIN129
C       INTERCHANGE ROWS TO PUT PIVOT ELEMENT ON DIAGONAL                     MATIN130
```

```
C
      IF (IROW-ICOLUM) 140, 260, 140
  140 DETERM=-DETERM
      DO 200 L=1,N
        SWAP=A(IROW,L)
        A(IROW,L)=A(ICOLUM,L)
        A(ICOLUM,L)=SWAP
  200 CONTINUE
      IF(M) 260, 260, 210
  210 DO 250 L=1, M
        SWAP=B(IROW,L)
        B(IROW,L)=B(ICOLUM,L)
        B(ICOLUM,L)=SWAP
  250 CONTINUE
  260 IWK(I,1)=IROW
      IWK(I,2)=ICOLUM
      PIVOT=A(ICOLUM,ICOLUM)
      IF(IOP) 740,1000,321
C
C     SCALE THE DETERMINANT
C
 1000 PIVOTI=PIVOT
      IF(ABS(DETERM)-R1)1030,1010,1010
 1010 DETERM=DETERM/R1
      ISCALE=ISCALE+1
      IF(ABS(DETERM)-R1)1060,1020,1020
 1020 DETERM=DETERM/R1
      ISCALE=ISCALE+1
      GO TO 1060
 1030 IF(ABS(DETERM)-R2)1040,1040,1060
 1040 DETERM=DETERM*R1
      ISCALE=ISCALE-1
      IF(ABS(DETERM)-R2)1050,1050,1060
 1050 DETERM=DETERM*R1
      ISCALE=ISCALE-1
 1060 IF(ABS(PIVOTI)-R1)1090,1070,1070
 1070 PIVOTI=PIVOTI/R1
      ISCALE=ISCALE+1
      IF(ABS(PIVOTI)-R1)320,1080,1080
 1080 PIVOTI=PIVOTI/R1
      ISCALE=ISCALE+1
      GO TO 320
 1090 IF(ABS(PIVOTI)-R2)2000,2000,320
 2000 PIVOTI=PIVOTI*R1
      ISCALE=ISCALE-1
      IF(ABS(PIVOTI)-R2)2010,2010,320
 2010 PIVOTI=PIVOTI*R1
      ISCALE=ISCALE-1
  320 DETERM=DETERM*PIVOTI
C
C     DIVIDE PIVOT ROW BY PIVOT ELEMENT
C
  321 A(ICOLUM,ICOLUM)=1.0d+00
      DO 350 L=1,N
  350   A(ICOLUM,L)=A(ICOLUM,L)/PIVOT
      IF(M) 380, 380, 360
  360 DO 370 L=1,M
  370   B(ICOLUM,L)=B(ICOLUM,L)/PIVOT
C
C     REDUCE NON-PIVOT ROWS
```

```
C                                                                       MATIN191
  380   DO 550 L1=1,N                                                   MATIN192
          IF(L1-ICOLUM) 400, 550, 400                                   MATIN193
  400     T=A(L1,ICOLUM)                                                MATIN194
          A(L1,ICOLUM)=0.0d+00                                          MATIN195
          DO 450 L=1,N                                                  MATIN196
  450       A(L1,L)=A(L1,L)-A(ICOLUM,L)*T                               MATIN197
          IF(M) 550, 550, 460                                           MATIN198
  460     DO 500 L=1,M                                                  MATIN199
  500       B(L1,L)=B(L1,L)-B(ICOLUM,L)*T                               MATIN200
  550 CONTINUE                                                          MATIN201
C                                                                       MATIN202
C     INTERCHANGE COLUMNS                                               MATIN203
C                                                                       MATIN204
      DO 710 I=1,N                                                      MATIN205
        L=N+1-I                                                         MATIN206
        IF (IWK(L,1)-IWK(L,2))630,710,630                               MATIN207
  630   JROW=IWK(L,1)                                                   MATIN208
        JCOLUM=IWK(L,2)                                                 MATIN209
        DO 705 K=1,N                                                    MATIN210
          SWAP=A(K,JROW)                                                MATIN211
          A(K,JROW)=A(K,JCOLUM)                                         MATIN212
          A(K,JCOLUM)=SWAP                                              MATIN213
  705   CONTINUE                                                        MATIN214
  710 CONTINUE                                                          MATIN215
  740 RETURN                                                            MATIN216
      END                                                               MATIN217
C        ROUTINE NAME   - HC318=EPSLON                                  EPSLON 2
C        FROM EISPACK                                                   EPSLON 3
C                                                                       EPSLON 4
C-----------------------------------------------------------------      EPSLON 5
C                                                                       EPSLON 6
C   LATEST REVISION     - AUGUST 1,1984                                 EPSLON 7
C                         COMPUTER SCIENCES CORP., HAMPTON, VA.         EPSLON 8
C                                                                       EPSLON 9
C   PURPOSE             - THE FORTRAN FUNCTION EPSLON ESTIMATES UNIT    EPSLON10
C                         ROUNDOFF IN QUANTITIES OF SIZE X.             EPSLON11
C                                                                       EPSLON12
C   USAGE               - VARIABLE = EPSLON(X)                          EPSLON13
C                                                                       EPSLON14
C   ARGUMENTS    X      - IS A REAL INPUT VARIABLE WHICH REPRESENTS THE EPSLON15
C                         QUANTITIES OF SIZE IN WHICH UNIT ROUNDOFF     EPSLON16
C                         WILL BE ESTIMATED.                            EPSLON17
C                                                                       EPSLON18
C   REQUIRED ROUTINES   - NONE                                          EPSLON19
C                                                                       EPSLON20
C   REMARKS  1.  IT SHOULD BE NOTED THAT EPSLON IS A FUNCTION           EPSLON21
C                DESIGNED TO BE CALLED BY ROUTINES IN THE               EPSLON22
C                EISPACK VERSION 3.                                     EPSLON23
C                                                                       EPSLON24
C                THIS PROGRAM SHOULD FUNCTION PROPERLY  ON  ALL         EPSLON25
C                SYSTEMS  SATISFYING  THE   FOLLOWING   TWO             EPSLON26
C                ASSUMPTIONS,                                           EPSLON27
C                                                                       EPSLON28
C                A.  THE BASE  USED  IN  REPRESENTING  FLOATING         EPSLON29
C                POINT NUMBERS IS NOT A POWER OF THREE.                 EPSLON30
C                                                                       EPSLON31
C                B.   THE  QUANTITY  A  IN  STATEMENT   10   IS         EPSLON32
C                REPRESENTED  TO  THE ACCURACY USED IN FLOATING         EPSLON33
C                POINT VARIABLES THAT ARE STORED IN MEMORY.             EPSLON34
```

```
C
C              THE  STATEMENT  NUMBER 10 AND THE GO TO 10 ARE
C              INTENDED  TO  FORCE  OPTIMIZING  COMPILERS  TO
C              GENERATE CODE SATISFYING ASSUMPTION 2.
C
C              UNDER THESE ASSUMPTIONS,  IT  SHOULD  BE  TRUE
C              THAT,
C
C              A IS NOT EXACTLY EQUAL TO FOUR-THIRDS,
C
C              B HAS A ZERO FOR ITS LAST BIT OR DIGIT,
C
C              C IS NOT EXACTLY EQUAL TO ONE,
C
C              EPS  MEASURES  THE  SEPARATION OF 1.0 FROM THE
C              NEXT LARGER FLOATING POINT NUMBER.
C
C   EXAMPLE :
C     PROGRAM TR(OUTPUT,TAPE6=OUTPUT)
C     REAL X
C     X = 4.
C     A = EPSLON(X)
C     WRITE(6,100) A
C100  FORMAT(5H0A = ,G22.14)
C     STOP
C     END
C     OUTPUT :
CA =     .56843418860808E-13
C
C-------------------------------------------------------------------
C*F45V1P0*
      REAL*8 FUNCTION EPSLON (X)
C
      REAL*8 X
      REAL*8 A,B,C,EPS
      A = 4.0E0/3.0E0
   10 B = A - 1.0E0
      C = B + B + B
      EPS = ABS(C-1.0E0)
      IF (EPS .EQ. 0.0E0) GO TO 10
      EPSLON = EPS*ABS(X)
      RETURN
C**  THIS PROGRAM VALID ON FTN4 AND FTN5 **
      END
C        ROUTINE NAME   - PF260=QZHES
C        FROM EISPACK
C
C--------------------------------------------------------------------------
C
C   LATEST REVISION     - AUGUST 1,1984
C                         COMPUTER SCIENCES CORP., HAMPTON, VA.
C
C
C   PURPOSE             - THIS SUBROUTINE ACCEPTS A PAIR OF REAL
C                         GENERAL MATRICES AND REDUCES ONE OF THEM TO
C                         UPPER HESSENBERG FORM AND THE OTHER TO UPPER
C                         TRIANGULAR FORM USING ORTHOGONAL
C                         TRANSFORMATIONS.  IT IS USUALLY FOLLOWED BY
C                         QZIT(PF261), QZVAL(PF262) AND, POSSIBLY,
C                         QZVEC(PF263).
```

```
C
C
C   USAGE              - CALL QZHES(NM,N,A,B,MATZ,Z)
C
C   ARGUMENTS    NM    - ON INPUT NM MUST BE SET TO THE ROW DIMENSION
C                        OF TWO-DIMENSIONAL ARRAY PARAMETERS AS
C                        DECLARED IN THE CALLING PROGRAM DIMENSION
C                        STATEMENT.
C
C                N     - ON INPUT N IS THE ORDER OF THE MATRICES.
C
C                A     - ON INPUT A CONTAINS A REAL GENERAL MATRIX.
C                        MUST BE OF DIMENSION NM X N.
C
C                        ON OUTPUT A HAS BEEN REDUCED TO UPPER
C                        HESSENBERG FORM.  THE ELEMENTS BELOW THE FIRST
C                        SUBDIAGONAL HAVE BEEN SET TO ZERO.
C
C                B     - ON INPUT B CONTAINS A REAL GENERAL MATRIX.
C                        MUST BE OF DIMENSION NM X N.
C
C                        ON OUTPUT B HAS BEEN REDUCED TO UPPER
C                        TRIANGULAR FORM.  THE ELEMENTS BELOW THE MAIN
C                        DIAGONAL HAVE BEEN SET TO ZERO.
C
C                MATZ  - ON INPUT MATZ SHOULD BE SET TO .TRUE.  IF THE
C                        RIGHT HAND TRANSFORMATIONS ARE TO BE
C                        ACCUMULATED FOR LATER USE IN COMPUTING
C                        EIGENVECTORS, AND TO .FALSE.  OTHERWISE.
C
C                Z     - ON OUTPUT Z CONTAINS THE PRODUCT OF THE RIGHT
C                        HAND TRANSFORMATIONS IF MATZ HAS BEEN SET TO
C                        .TRUE. OTHERWISE, Z IS NOT REFERENCED.
C                        MUST BE OF DIMENSION NM X N.
C
C
C   REQUIRED ROUTINES  - NONE
C
C   REMARKS  1.  THIS SUBROUTINE IS THE FIRST STEP OF THE QZ
C                ALGORITHM FOR SOLVING GENERALIZED MATRIX
C                EIGENVALUE PROBLEMS, SIAM J. NUMER. ANAL. 10,
C                241-256(1973) BY MOLER AND STEWART.
C
C   EXAMPLE :
C     PROGRAM TQZHES(OUTPUT,TAPE6=OUTPUT)
C     DIMENSION A(5,5),Z(5,5),B(5,5)
C     LOGICAL MATZ
C
C     N = 5
C     NM = 5
C     MATZ = .TRUE.
C
C     DATA A /10.,2.,3.,2*1.,2.,12.,1.,2.,1.,3.,1.,11.,
C    *        1.,-1.,1.,2.,1.,9.,3*1.,-1.,1.,15.   /
C
C     DATA B /12.,1.,-1.,2.,2*1.,14.,1.,-1.,1.,-1.,1.,
C    *        16.,-1.,1.,2.,-1.,-1.,12.,-1.,3*1.,-1.,11.  /
C
C     CALL  QZHES(NM,N,A,B,MATZ,Z)
C     WRITE(6,100) ((A(I,J),I=1,5),J=1,5),((B(I,J),I=1,5),J=1,5),
```

```
C     *          ((Z(I,J),I=1,5),J=1,5)
C100  FORMAT(1H ,5H A = /5(1H ,5(G8.2,2X)/))
C     *        5H B = /5(1H ,5(G8.2,2X)/)
C     *        5H Z = /5(1H ,5(G8.2,2X)/))
C      STOP
C      END
C
C      OUTPUT :
C
C      A =
C     -9.9       4.1       0.        0.        0.
C     -2.4       11.       -3.0      0.        0.
C      .91       .26       -13.       3.3      0.
C     -3.8       2.0        1.7      -11.       2.6
C      2.7      -1.5       -.99       1.4      -11.
C      B =
C      -12.      0.        0.        0.        0.
C       2.3       16.      0.        0.        0.
C      -.34     -3.0      -12.       0.        0.
C      -3.8       .80     -1.5      -10.       0.
C       2.5     -1.4      -1.5      -1.5      -13.
C      Z =
C      1.0      0.        0.        0.        0.
C      0.         .26       .95      -.14      -.70E-01
C      0.         .87E-01  -.24E-01   .43      -.90
C      0.         .24E-01   .16       .89       .43
C      0.       -.96        .26       .22E-01  -.89E-01
C
C-----------------------------------------------------------------------
      SUBROUTINE QZHES(NM,N,A,B,MATZ,Z)
C
      implicit real*8 (a-h,o-z)
      INTEGER I,J,K,L,N,LB,L1,NM,NK1,NM1,NM2
      REAL*8 A(NM,N),B(NM,N),Z(NM,N)
      REAL*8 R,S,T,U1,U2,V1,V2,RHO
      LOGICAL MATZ
      IF (.NOT. MATZ) GO TO 10
C
      DO 3 J = 1, N
C
         DO 2 I = 1, N
            Z(I,J) = 0.0E0
    2    CONTINUE
C
         Z(J,J) = 1.0E0
    3 CONTINUE
C     .......... REDUCE B TO UPPER TRIANGULAR FORM ..........
   10 IF (N .LE. 1) GO TO 170
      NM1 = N - 1
C
      DO 100 L = 1, NM1
         L1 = L + 1
         S = 0.0E0
C
         DO 20 I = L1, N
            S = S + ABS(B(I,L))
   20    CONTINUE
C
         IF (S .EQ. 0.0E0) GO TO 100
         S = S + ABS(B(L,L))
```

```
         R = 0.0E0
C
         DO 25 I = L, N
            B(I,L) = B(I,L) / S
            R = R + B(I,L)**2
   25    CONTINUE
C
         R = SIGN(SQRT(R),B(L,L))
         B(L,L) = B(L,L) + R
         RHO = R * B(L,L)
C
         DO 50 J = L1, N
            T = 0.0E0
C
            DO 30 I = L, N
               T = T + B(I,L) * B(I,J)
   30       CONTINUE
C
            T = -T / RHO
C
            DO 40 I = L, N
               B(I,J) = B(I,J) + T * B(I,L)
   40       CONTINUE
C
   50    CONTINUE
C
         DO 80 J = 1, N
            T = 0.0E0
C
            DO 60 I = L, N
               T = T + B(I,L) * A(I,J)
   60       CONTINUE
C
            T = -T / RHO
C
            DO 70 I = L, N
               A(I,J) = A(I,J) + T * B(I,L)
   70       CONTINUE
C
   80    CONTINUE
C
         B(L,L) = -S * R
C
         DO 90 I = L1, N
            B(I,L) = 0.0E0
   90    CONTINUE
C
  100 CONTINUE
C     .......... REDUCE A TO UPPER HESSENBERG FORM, WHILE
C                KEEPING B TRIANGULAR ..........
      IF (N .EQ. 2) GO TO 170
      NM2 = N - 2
C
      DO 160 K = 1, NM2
         NK1 = NM1 - K
C     .......... FOR L=N-1 STEP -1 UNTIL K+1 DO -- ..........
         DO 150 LB = 1, NK1
            L = N - LB
            L1 = L + 1
C     .......... ZERO A(L+1,K) ..........
```

```
            S = ABS(A(L,K)) + ABS(A(L1,K))
            IF (S .EQ. 0.0E0) GO TO 150
            U1 = A(L,K) / S
            U2 = A(L1,K) / S
            R = SIGN(SQRT(U1*U1+U2*U2),U1)
            V1 =  -(U1 + R) / R
            V2 = -U2 / R
            U2 = V2 / V1
C
            DO 110 J = K, N
               T = A(L,J) + U2 * A(L1,J)
               A(L,J) = A(L,J) + T * V1
               A(L1,J) = A(L1,J) + T * V2
  110       CONTINUE
C
            A(L1,K) = 0.0E0
C
            DO 120 J = L, N
               T = B(L,J) + U2 * B(L1,J)
               B(L,J) = B(L,J) + T * V1
               B(L1,J) = B(L1,J) + T * V2
  120       CONTINUE
C     .......... ZERO B(L+1,L) ..........
            S = ABS(B(L1,L1)) + ABS(B(L1,L))
            IF (S .EQ. 0.0E0) GO TO 150
            U1 = B(L1,L1) / S
            U2 = B(L1,L) / S
            R = SIGN(SQRT(U1*U1+U2*U2),U1)
            V1 =  -(U1 + R) / R
            V2 = -U2 / R
            U2 = V2 / V1
C
            DO 130 I = 1, L1
               T = B(I,L1) + U2 * B(I,L)
               B(I,L1) = B(I,L1) + T * V1
               B(I,L) = B(I,L) + T * V2
  130       CONTINUE
C
            B(L1,L) = 0.0E0
C
            DO 140 I = 1, N
               T = A(I,L1) + U2 * A(I,L)
               A(I,L1) = A(I,L1) + T * V1
               A(I,L) = A(I,L) + T * V2
  140       CONTINUE
C
            IF (.NOT. MATZ) GO TO 150
C
            DO 145 I = 1, N
               T = Z(I,L1) + U2 * Z(I,L)
               Z(I,L1) = Z(I,L1) + T * V1
               Z(I,L) = Z(I,L) + T * V2
  145       CONTINUE
C
  150    CONTINUE
C
  160 CONTINUE
C
  170 RETURN
C**  THIS PROGRAM VALID ON FTN4 AND FTN5 **
```

```
      END
C        ROUTINE NAME    - PF261=QZIT
C        FROM EISPACK
C
C-----------------------------------------------------------------------
C
C    LATEST REVISION     - AUGUST 1,1984
C                          COMPUTER SCIENCES CORP., HAMPTON, VA.
C
C
C    PURPOSE             - THIS SUBROUTINE ACCEPTS A PAIR OF REAL
C                          MATRICES, ONE OF THEM IN UPPER HESSENBERG
C                          FORM AND THE OTHER IN UPPER TRIANGULAR FORM.
C                          IT REDUCES THE HESSENBERG MATRIX TO
C                          QUASI-TRIANGULAR FORM USING ORTHOGONAL
C                          TRANSFORMATIONS WHILE MAINTAINING THE
C                          TRIANGULAR FORM OF THE OTHER MATRIX.  IT IS
C                          USUALLY PRECEDED  QZHES(PF260) AND FOLLOWED
C                          BY QZVAL(PF262) AND, POSSIBLY, QZVEC(PF263).
C
C
C    USAGE               - CALL QZIT(NM,N,A,B,EPS1,MATZ,Z,IERR)
C
C    ARGUMENTS    NM     - ON INPUT NM MUST BE SET TO THE ROW DIMENSION
C                          OF TWO-DIMENSIONAL ARRAY PARAMETERS AS
C                          DECLARED IN THE CALLING PROGRAM DIMENSION
C                          STATEMENT.
C
C                 N      - ON INPUT N IS THE ORDER OF THE MATRICES.
C
C                 A      - ON INPUT A CONTAINS A REAL UPPER HESSENBERG
C                          MATRIX.
C                          MUST BE OF DIMENSION NM X N.
C
C                          ON OUTPUT A HAS BEEN REDUCED TO
C                          QUASI-TRIANGULAR FORM.  THE ELEMENTS BELOW THE
C                          FIRST SUBDIAGONAL ARE STILL ZERO AND NO TWO
C                          CONSECUTIVE SUBDIAGONAL ELEMENTS ARE NONZERO.
C
C
C                 B      - ON INPUT B CONTAINS A REAL UPPER TRIANGULAR
C                          MATRIX.
C                          MUST BE OF DIMENSION NM X N.
C
C                          ON OUTPUT B IS STILL IN UPPER TRIANGULAR
C                          FORM, ALTHOUGH ITS ELEMENTS HAVE BEEN ALTERED.
C                          THE LOCATION B(N,1) IS USED TO STORE EPS1
C                          TIMES THE NORM OF B FOR LATER USE BY QZVAL
C                          QZVAL(PF262) AND QZVEC(PF263).
C
C                 EPS1   - ON INPUT EPS1 IS A TOLERANCE USED TO DETERMINE
C                          NEGLIGIBLE ELEMENTS.  EPS1 = 0.0 (OR NEGATIVE)
C                          MAY BE INPUT, IN WHICH CASE AN ELEMENT WILL BE
C                          NEGLECTED ONLY IF IT IS LESS THAN ROUNDOFF
C                          ERROR TIMES THE NORM OF ITS MATRIX.  IF THE
C                          INPUT EPS1 IS POSITIVE, THEN AN ELEMENT WILL
C                          BE CONSIDERED NEGLIGIBLE IF IT IS LESS THAN
C                          EPS1 TIMES THE NORM OF ITS MATRIX.  A POSITIVE
C                          VALUE OF EPS1 MAY RESULT IN FASTER EXECUTION,
C                          BUT LESS ACCURATE RESULTS.
```

```
C
C               MATZ    - ON INPUT MATZ SHOULD BE SET TO .TRUE.  IF THE
C                         RIGHT HAND TRANSFORMATIONS ARE TO BE
C                         ACCUMULATED FOR LATER USE IN COMPUTING
C                         EIGENVECTORS, AND TO .FALSE.  OTHERWISE.
C
C               Z       - ON INPUT Z CONTAINS, IF MATZ HAS BEEN SET TO
C                         .TRUE., THE TRANSFORMATION MATRIX PRODUCED IN
C                         THE REDUCTION BY QZHES(PF260), IF PERFORMED,
C                         OR ELSE THE IDENTITY MATRIX.  IF MATZ HAS BEEN
C                         SET TO .FALSE., Z IS NOT REFERENCED.
C                         MUST BE OF DIMENSION NM X N.
C
C                         ON OUTPUT Z CONTAINS THE PRODUCT OF THE
C                         RIGHT HAND TRANSFORMATIONS (FOR BOTH STEPS) IF
C                         MATZ HAS BEEN SET TO .TRUE..
C
C               IERR   - ON OUTPUT IERR IS SET TO
C                        ZERO FOR NORMAL RETURN.
C                        J IF THE LIMIT OF 30*N ITERATIONS IS EXHAUSTED
C                        WHILE THE J-TH EIGENVALUE IS BEING SOUGHT.
C
C
C   REQUIRED ROUTINES    - HC318=EPSLON
C
C   REMARKS  1.  THIS SUBROUTINE IS THE SECOND STEP OF THE QZ
C                ALGORITHM FOR SOLVING GENERALIZED MATRIX
C                EIGENVALUE PROBLEMS, SIAM J. NUMER. ANAL. 10,
C                241-256(1973) BY MOLER AND STEWART, AS
C                MODIFIED IN TECHNICAL NOTE NASA TN
C                D-7305(1973) BY WARD.
C
C   EXAMPLE :
C     PROGRAM TQZIT(OUTPUT,TAPE6=OUTPUT)
C     DIMENSION A(5,5),B(5,5),Z(5,5)
C     LOGICAL MATZ
C
C     N = 5
C     NM = 5
C     MATZ = .TRUE.
C     EPS1 = 0.0E0
C
C     DATA A /10.,2.,3.,2*1.,2.,12.,1.,2.,1.,3.,1.,11.,
C    *       1.,-1.,1.,2.,1.,9.,3*1.,-1.,1.,15.   /
C
C     DATA B /12.,1.,-1.,2.,2*1.,14.,1.,-1.,1.,-1.,1.,
C    *       16.,-1.,1.,2.,-1.,-1.,12.,-1.,3*1.,-1.,11.  /
C
C     CALL  QZHES(NM,N,A,B,MATZ,Z)
C     CALL  QZIT(NM,N,A,B,EPS1,MATZ,Z,IERR)
C     WRITE(6,99) IERR
C99   FORMAT(1H1,8H IERR = ,I4)
C     WRITE(6,100) ((A(I,J),I=1,5),J=1,5),((B(I,J),I=1,5),J=1,5),
C    *           ((Z(I,J),I=1,5),J=1,5)
C100  FORMAT(1H ,5H A = /5(1H ,5(G8.2,2X)/))
C    *       5H B = /5(1H ,5(G8.2,2X)/)
C    *       5H Z = /5(1H ,5(G8.2,2X)/))
C     STOP
C     END
C
```

```
C     OUTPUT :
C
C     IERR =    0
C     A =
C    -15.        -1.3       0.         0.         0.
C     1.1         7.4       0.         0.         0.
C     1.5        -1.5      -16.        0.         0.
C    -2.2         .96        1.0      -10.        0.
C    -2.6        -.31        1.2        1.7      -8.6
C     B =
C    -9.9        0.         0.         0.          .31E-12
C    -.29         17.       0.         0.         0.
C     1.3        -2.1      -14.        0.         0.
C    -1.9         1.7        .96      -11.        0.
C    -2.6        -.32        1.3        2.1      -13.
C     Z =
C     .28        -.71E-01    .16       -.24       -.91
C     .52        -.24       -.66        .48       -.64E-01
C     .49         .56        .49        .45        .75E-01
C    -.60         .48       -.29        .44       -.38
C    -.25        -.63        .45        .57       -.94E-01
C
C---------------------------------------------------------------------
      SUBROUTINE QZIT(NM,N,A,B,EPS1,MATZ,Z,IERR)
C
      implicit real*8 (a-h,o-z)
      INTEGER I,J,K,L,N,EN,K1,K2,LD,LL,L1,NA,NM,ISH,ITN,ITS,KM1,LM1,
     X        ENM2,IERR,LOR1,ENORN
      REAL*8 A(NM,N),B(NM,N),Z(NM,N)
      REAL*8 R,S,T,A1,A2,A3,EP,SH,U1,U2,U3,V1,V2,V3,ANI,A11,
     X       A12,A21,A22,A33,A34,A43,A44,BNI,B11,B12,B22,B33,B34,
     X       B44,EPSA,EPSB,EPS1,ANORM,BNORM,EPSLON
      LOGICAL MATZ,NOTLAS
      IERR = 0
C     .......... COMPUTE EPSA,EPSB ..........
      ANORM = 0.0E0
      BNORM = 0.0E0
C
      DO 30 I = 1, N
         ANI = 0.0E0
         IF (I .NE. 1) ANI = ABS(A(I,I-1))
         BNI = 0.0E0
C
         DO 20 J = I, N
            ANI = ANI + ABS(A(I,J))
            BNI = BNI + ABS(B(I,J))
   20    CONTINUE
C
         IF (ANI .GT. ANORM) ANORM = ANI
         IF (BNI .GT. BNORM) BNORM = BNI
   30 CONTINUE
C
      IF (ANORM .EQ. 0.0E0) ANORM = 1.0E0
      IF (BNORM .EQ. 0.0E0) BNORM = 1.0E0
      EP = EPS1
      IF (EP .GT. 0.0E0) GO TO 50
C     .......... USE ROUNDOFF LEVEL IF EPS1 IS ZERO ..........
      EP = EPSLON(1.0E0)
   50 EPSA = EP * ANORM
      EPSB = EP * BNORM
```

```
C     .......... REDUCE A TO QUASI-TRIANGULAR FORM, WHILE
C                KEEPING B TRIANGULAR ..........
      LOR1 = 1
      ENORN = N
      EN = N
      ITN = 30*N
C     .......... BEGIN QZ STEP ..........
   60 IF (EN .LE. 2) GO TO 1001
      IF (.NOT. MATZ) ENORN = EN
      ITS = 0
      NA = EN - 1
      ENM2 = NA - 1
   70 ISH = 2
C     .......... CHECK FOR CONVERGENCE OR REDUCIBILITY.
C                FOR L=EN STEP -1 UNTIL 1 DO -- ..........
      DO 80 LL = 1, EN
         LM1 = EN - LL
         L = LM1 + 1
         IF (L .EQ. 1) GO TO 95
         IF (ABS(A(L,LM1)) .LE. EPSA) GO TO 90
   80 CONTINUE
C
   90 A(L,LM1) = 0.0E0
      IF (L .LT. NA) GO TO 95
C     .......... 1-BY-1 OR 2-BY-2 BLOCK ISOLATED ..........
      EN = LM1
      GO TO 60
C     .......... CHECK FOR SMALL TOP OF B ..........
   95 LD = L
  100 L1 = L + 1
      B11 = B(L,L)
      IF (ABS(B11) .GT. EPSB) GO TO 120
      B(L,L) = 0.0E0
      S = ABS(A(L,L)) + ABS(A(L1,L))
      U1 = A(L,L) / S
      U2 = A(L1,L) / S
      R = SIGN(SQRT(U1*U1+U2*U2),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      U2 = V2 / V1
C
      DO 110 J = L, ENORN
         T = A(L,J) + U2 * A(L1,J)
         A(L,J) = A(L,J) + T * V1
         A(L1,J) = A(L1,J) + T * V2
         T = B(L,J) + U2 * B(L1,J)
         B(L,J) = B(L,J) + T * V1
         B(L1,J) = B(L1,J) + T * V2
  110 CONTINUE
C
      IF (L .NE. 1) A(L,LM1) = -A(L,LM1)
      LM1 = L
      L = L1
      GO TO 90
  120 A11 = A(L,L) / B11
      A21 = A(L1,L) / B11
      IF (ISH .EQ. 1) GO TO 140
C     .......... ITERATION STRATEGY ..........
      IF (ITN .EQ. 0) GO TO 1000
      IF (ITS .EQ. 10) GO TO 155
```

```
C     .......... DETERMINE TYPE OF SHIFT ..........
      B22 = B(L1,L1)
      IF (ABS(B22) .LT. EPSB) B22 = EPSB
      B33 = B(NA,NA)
      IF (ABS(B33) .LT. EPSB) B33 = EPSB
      B44 = B(EN,EN)
      IF (ABS(B44) .LT. EPSB) B44 = EPSB
      A33 = A(NA,NA) / B33
      A34 = A(NA,EN) / B44
      A43 = A(EN,NA) / B33
      A44 = A(EN,EN) / B44
      B34 = B(NA,EN) / B44
      T = 0.5E0 * (A43 * B34 - A33 - A44)
      R = T * T + A34 * A43 - A33 * A44
      IF (R .LT. 0.0E0) GO TO 150
C     .......... DETERMINE SINGLE SHIFT ZEROTH COLUMN OF A ..........
      ISH = 1
      R = SQRT(R)
      SH = -T + R
      S = -T - R
      IF (ABS(S-A44) .LT. ABS(SH-A44)) SH = S
C     .......... LOOK FOR TWO CONSECUTIVE SMALL
C                SUB-DIAGONAL ELEMENTS OF A.
C                FOR L=EN-2 STEP -1 UNTIL LD DO -- ..........
      DO 130 LL = LD, ENM2
         L = ENM2 + LD - LL
         IF (L .EQ. LD) GO TO 140
         LM1 = L - 1
         L1 = L + 1
         T = A(L,L)
         IF (ABS(B(L,L)) .GT. EPSB) T = T - SH * B(L,L)
         IF (ABS(A(L,LM1)) .LE. ABS(T/A(L1,L)) * EPSA) GO TO 100
  130 CONTINUE
C
  140 A1 = A11 - SH
      A2 = A21
      IF (L .NE. LD) A(L,LM1) = -A(L,LM1)
      GO TO 160
C     .......... DETERMINE DOUBLE SHIFT ZEROTH COLUMN OF A ..........
  150 A12 = A(L,L1) / B22
      A22 = A(L1,L1) / B22
      B12 = B(L,L1) / B22
      A1 = ((A33 - A11) * (A44 - A11) - A34 * A43 + A43 * B34 * A11)
     X     / A21 + A12 - A11 * B12
      A2 = (A22 - A11) - A21 * B12 - (A33 - A11) - (A44 - A11)
     X     + A43 * B34
      A3 = A(L1+1,L1) / B22
      GO TO 160
C     .......... AD HOC SHIFT ..........
  155 A1 = 0.0E0
      A2 = 1.0E0
      A3 = 1.1605E0
  160 ITS = ITS + 1
      ITN = ITN - 1
      IF (.NOT. MATZ) LOR1 = LD
C     .......... MAIN LOOP ..........
      DO 260 K = L, NA
         NOTLAS = K .NE. NA .AND. ISH .EQ. 2
         K1 = K + 1
         K2 = K + 2
```

```
      KM1 = MAX0(K-1,L)
      LL = MIN0(EN,K1+ISH)
      IF (NOTLAS) GO TO 190
C     .......... ZERO A(K+1,K-1) ..........
      IF (K .EQ. L) GO TO 170
      A1 = A(K,KM1)
      A2 = A(K1,KM1)
  170 S = ABS(A1) + ABS(A2)
      IF (S .EQ. 0.0E0) GO TO 70
      U1 = A1 / S
      U2 = A2 / S
      R = SIGN(SQRT(U1*U1+U2*U2),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      U2 = V2 / V1
C
      DO 180 J = KM1, ENORN
         T = A(K,J) + U2 * A(K1,J)
         A(K,J) = A(K,J) + T * V1
         A(K1,J) = A(K1,J) + T * V2
         T = B(K,J) + U2 * B(K1,J)
         B(K,J) = B(K,J) + T * V1
         B(K1,J) = B(K1,J) + T * V2
  180 CONTINUE
C
      IF (K .NE. L) A(K1,KM1) = 0.0E0
      GO TO 240
C     .......... ZERO A(K+1,K-1) AND A(K+2,K-1) ..........
  190 IF (K .EQ. L) GO TO 200
      A1 = A(K,KM1)
      A2 = A(K1,KM1)
      A3 = A(K2,KM1)
  200 S = ABS(A1) + ABS(A2) + ABS(A3)
      IF (S .EQ. 0.0E0) GO TO 260
      U1 = A1 / S
      U2 = A2 / S
      U3 = A3 / S
      R = SIGN(SQRT(U1*U1+U2*U2+U3*U3),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      V3 = -U3 / R
      U2 = V2 / V1
      U3 = V3 / V1
C
      DO 210 J = KM1, ENORN
         T = A(K,J) + U2 * A(K1,J) + U3 * A(K2,J)
         A(K,J) = A(K,J) + T * V1
         A(K1,J) = A(K1,J) + T * V2
         A(K2,J) = A(K2,J) + T * V3
         T = B(K,J) + U2 * B(K1,J) + U3 * B(K2,J)
         B(K,J) = B(K,J) + T * V1
         B(K1,J) = B(K1,J) + T * V2
         B(K2,J) = B(K2,J) + T * V3
  210 CONTINUE
C
      IF (K .EQ. L) GO TO 220
      A(K1,KM1) = 0.0E0
      A(K2,KM1) = 0.0E0
C     .......... ZERO B(K+2,K+1) AND B(K+2,K) ..........
  220 S = ABS(B(K2,K2)) + ABS(B(K2,K1)) + ABS(B(K2,K))
```

```
      IF (S .EQ. 0.0E0) GO TO 240
      U1 = B(K2,K2) / S
      U2 = B(K2,K1) / S
      U3 = B(K2,K) / S
      R = SIGN(SQRT(U1*U1+U2*U2+U3*U3),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      V3 = -U3 / R
      U2 = V2 / V1
      U3 = V3 / V1
C
      DO 230 I = LOR1, LL
         T = A(I,K2) + U2 * A(I,K1) + U3 * A(I,K)
         A(I,K2) = A(I,K2) + T * V1
         A(I,K1) = A(I,K1) + T * V2
         A(I,K) = A(I,K) + T * V3
         T = B(I,K2) + U2 * B(I,K1) + U3 * B(I,K)
         B(I,K2) = B(I,K2) + T * V1
         B(I,K1) = B(I,K1) + T * V2
         B(I,K) = B(I,K) + T * V3
  230 CONTINUE
C
      B(K2,K) = 0.0E0
      B(K2,K1) = 0.0E0
      IF (.NOT. MATZ) GO TO 240
C
      DO 235 I = 1, N
         T = Z(I,K2) + U2 * Z(I,K1) + U3 * Z(I,K)
         Z(I,K2) = Z(I,K2) + T * V1
         Z(I,K1) = Z(I,K1) + T * V2
         Z(I,K) = Z(I,K) + T * V3
  235 CONTINUE
C     .......... ZERO B(K+1,K) ..........
  240 S = ABS(B(K1,K1)) + ABS(B(K1,K))
      IF (S .EQ. 0.0E0) GO TO 260
      U1 = B(K1,K1) / S
      U2 = B(K1,K) / S
      R = SIGN(SQRT(U1*U1+U2*U2),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      U2 = V2 / V1
C
      DO 250 I = LOR1, LL
         T = A(I,K1) + U2 * A(I,K)
         A(I,K1) = A(I,K1) + T * V1
         A(I,K) = A(I,K) + T * V2
         T = B(I,K1) + U2 * B(I,K)
         B(I,K1) = B(I,K1) + T * V1
         B(I,K) = B(I,K) + T * V2
  250 CONTINUE
C
      B(K1,K) = 0.0E0
      IF (.NOT. MATZ) GO TO 260
C
      DO 255 I = 1, N
         T = Z(I,K1) + U2 * Z(I,K)
         Z(I,K1) = Z(I,K1) + T * V1
         Z(I,K) = Z(I,K) + T * V2
  255 CONTINUE
C
```

```
  260 CONTINUE
C     .......... END QZ STEP ..........
      GO TO 70
C     .......... SET ERROR -- ALL EIGENVALUES HAVE NOT
C                CONVERGED AFTER 30*N ITERATIONS ..........
 1000 IERR = EN
C     .......... SAVE EPSB FOR USE BY QZVAL AND QZVEC ..........
 1001 IF (N .GT. 1) B(N,1) = EPSB
      RETURN
C**  THIS PROGRAM VALID ON FTN4 AND FTN5 **
      END
C        ROUTINE NAME   - PF262=QZVAL
C        FROM EISPACK
C
C-----------------------------------------------------------------------
C
C   LATEST REVISION     - AUGUST 1,1984
C                         COMPUTER SCIENCES CORP., HAMPTON, VA.
C
C
C   PURPOSE             - THIS SUBROUTINE ACCEPTS A PAIR OF REAL
C                         MATRICES, ONE OF THEM IN QUASI-TRIANGULAR
C                         FORM AND THE OTHER IN UPPER TRIANGULAR FORM.
C                         IT REDUCES THE QUASI-TRIANGULAR MATRIX
C                         FURTHER, SO THAT ANY REMAINING 2-BY-2 BLOCKS
C                         CORRESPOND TO PAIRS OF COMPLEX EIGENVALUES,
C                         AND RETURNS QUANTITIES WHOSE RATIOS GIVE THE
C                         GENERALIZED EIGENVALUES.  IT IS USUALLY
C                         PRECEDED BY QZHES(PF260) AND QZIT(PF261) AND
C                         MAY BE FOLLOWED BY QZVEC(PF263).
C
C
C   USAGE               - CALL QZVAL(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z)
C
C   ARGUMENTS    NM     - ON INPUT NM MUST BE SET TO THE ROW DIMENSION
C                         OF TWO-DIMENSIONAL ARRAY PARAMETERS AS
C                         DECLARED IN THE CALLING PROGRAM DIMENSION
C                         STATEMENT.
C
C                N      - ON INPUT N IS THE ORDER OF THE MATRICES.
C
C                A      - ON INPUT A CONTAINS A REAL UPPER QUASI-
C                         TRIANGULAR MATRIX.
C                         MUST BE OF DIMENSION NM X N.
C
C                         ON OUTPUT A HAS BEEN REDUCED FURTHER TO A
C                         QUASI-TRIANGULAR MATRIX IN WHICH ALL NONZERO
C                         SUBDIAGONAL ELEMENTS CORRESPOND TO PAIRS OF
C                         COMPLEX EIGENVALUES.
C
C                B      - ON INPUT B CONTAINS A REAL UPPER TRIANGULAR
C                         MATRIX.
C                         MUST BE OF DIMENSION NM X N.
C                         IN ADDITION, LOCATION B(N,1) CONTAINS THE
C                         TOLERANCE QUANTITY (EPSB) COMPUTED AND SAVED
C                         IN QZIT(PF261).
C
C                         ON OUTPUT B IS STILL IN UPPER TRIANGULAR
C                         FORM, ALTHOUGH ITS ELEMENTS HAVE BEEN ALTERED.
C                         B(N,1) IS UNALTERED.
```

```
C
C
C            ALFR    - ON OUTPUT ALFR CONTAINS THE REAL PART OF THE
C                      DIAGONAL ELEMENTS OF THE TRIANGULAR MATRIX
C                      THAT WOULD BE OBTAINED IF A WERE REDUCED
C                      COMPLETELY TO TRIANGULAR FORM BY UNITARY
C                      TRANSFORMATIONS.  NON-ZERO VALUES OF ALFI
C                      OCCUR IN PAIRS, THE FIRST MEMBER POSITIVE AND
C                      THE SECOND NEGATIVE.
C                      MUST BE OF DIMENSION N.
C
C            ALFI    - ON OUTPUT ALFI CONTAINS THE IMAGINARY PART
C                      OF THE DIAGONAL ELEMENTS OF OF THE TRIANGULAR
C                      MATRIX THAT WOULD BE OBTAINED IF A WERE
C                      REDUCED COMPLETELY TO TRIANGULAR FORM BY
C                      UNITARY TRANSFORMATIONS.  NON-ZERO VALUES
C                      OF ALFI OCCUR IN PAIRS, THE FIRST MEMBER
C                      POSITIVE AND THE SECOND NEGATIVE.
C                      MUST BE OF DIMENSION N.
C
C            BETA    - ON OUTPUT BETA CONTAINS THE DIAGONAL ELEMENTS
C                      OF THE CORRESPONDING B, NORMALIZED TO BE REAL
C                      AND NON-NEGATIVE.  THE GENERALIZED EIGENVALUES
C                      ARE THEN THE RATIOS ((ALFR+I*ALFI)/BETA).
C                      MUST BE OF DIMENSION N.
C
C
C            MATZ    - ON INPUT MATZ SHOULD BE SET TO .TRUE.  IF
C                      THE RIGHT HAND TRANSFORMATIONS ARE TO BE
C                      ACCUMULATED FOR LATER USE IN COMPUTING
C                      EIGENVECTORS, AND TO .FALSE.  OTHERWISE.
C
C            Z       - ON INPUT Z CONTAINS, IF MATZ HAS BEEN SET
C                      TO .TRUE., THE TRANSFORMATION MATRIX PRODUCED
C                      IN THE REDUCTIONS BY QZHES(PF260) AND QZIT
C                      (PF261) IF PERFORMED, OR ELSE THE IDENTITY
C                      MATRIX.  IF MATZ HAS BEEN SET TO .FALSE., Z
C                      IS NOT REFERENCED.
C                      MUST BE OF DIMENSION NM X N.
C
C                      ON OUTPUT Z CONTAINS THE PRODUCT OF THE
C                      RIGHT HAND TRANSFORMATIONS (FOR ALL THREE
C                      STEPS) IF MATZ HAS BEEN SET TO .TRUE.
C
C   REQUIRED ROUTINES    - NONE
C
C   REMARKS   1.  THIS SUBROUTINE IS THE THIRD STEP OF THE QZ
C                 ALGORITHM FOR SOLVING GENERALIZED MATRIX
C                 EIGENVALUE PROBLEMS, SIAM J. NUMER. ANAL. 10,
C                 241-256(1973) BY MOLER AND STEWART.
C
C   EXAMPLE :
C     PROGRAM TQZVAL(OUTPUT,TAPE6=OUTPUT)
C     DIMENSION A(5,5),B(5,5),ALFR(5),ALFI(5),BETA(5),Z(5,5)
C     LOGICAL MATZ
C
C     N = 5
C     NM = 5
C     MATZ = .TRUE.
C     EPS1 = 0.0E0
```

```
C                                                                       QZVAL111
C     DATA A /10.,2.,3.,2*1.,2.,12.,1.,2.,1.,3.,1.,11.,                 QZVAL112
C    *       1.,-1.,1.,2.,1.,9.,3*1.,-1.,1.,15.   /                     QZVAL113
C                                                                       QZVAL114
C     DATA B /12.,1.,-1.,2.,2*1.,14.,1.,-1.,1.,-1.,1.,                  QZVAL115
C    *       16.,-1.,1.,2.,-1.,-1.,12.,-1.,3*1.,-1.,11.  /              QZVAL116
C                                                                       QZVAL117
C     CALL  QZHES(NM,N,A,B,MATZ,Z)                                      QZVAL118
C     CALL  QZIT(NM,N,A,B,EPS1,MATZ,Z,IERR)                             QZVAL119
C     CALL  QZVAL(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z)                       QZVAL120
C     WRITE(6,99) IERR                                                  QZVAL121
C     WRITE(6,100) ALFR,ALFI,BETA,((Z(I,J),I=1,5),J=1,5)                QZVAL122
C99   FORMAT(1H1,8H IERR = ,I4)                                         QZVAL123
C100  FORMAT(1H ,8H ALFR = /1H ,5(G8.2,2X)/                             QZVAL124
C    *       8H ALFI = /1H ,5(G8.2,2X)/                                 QZVAL125
C    *       8H BETA = /1H ,5(G8.2,2X)/                                 QZVAL126
C    *       5H Z = /5(1H ,5(G8.2,2X)/))                                QZVAL127
C     STOP                                                              QZVAL128
C     END                                                               QZVAL129
C                                                                       QZVAL130
C     OUTPUT :                                                          QZVAL131
C                                                                       QZVAL132
C     IERR =    0                                                       QZVAL133
C     ALFR =                                                            QZVAL134
C     15.       7.2       16.       10.       8.6                       QZVAL135
C     ALFI =                                                            QZVAL136
C     0.        0.        0.        0.        0.                        QZVAL137
C     BETA =                                                            QZVAL138
C     9.9       17.       14.       11.       13.                       QZVAL139
C     Z =                                                               QZVAL140
C      .24      -.54E-01   .21      -.27      -.91                      QZVAL141
C     -.54       .25       .65      -.46       .13                      QZVAL142
C      .49       .56       .49       .45       .75E-01                  QZVAL143
C     -.60       .48      -.29       .44      -.38                      QZVAL144
C     -.25      -.63       .45       .57      -.94E-01                  QZVAL145
C                                                                       QZVAL146
C---------------------------------------------------------------------- QZVAL147
      SUBROUTINE QZVAL(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z)                  EISP7123
C
      implicit real*8 (a-h,o-z)                                         EISP7124
      INTEGER I,J,N,EN,NA,NM,NN,ISW                                     EISP7125
      REAL*8 A(NM,N),B(NM,N),ALFR(N),ALFI(N),BETA(N),Z(NM,N)            EISP7126
      REAL*8 C,D,E,R,S,T,AN,A1,A2,BN,CQ,CZ,DI,DR,EI,TI,TR,U1,           EISP7127
     X       U2,V1,V2,A1I,A11,A12,A2I,A21,A22,B11,B12,B22,SQI,SQR,      EISP7128
     X       SSI,SSR,SZI,SZR,A11I,A11R,A12I,A12R,A22I,A22R,EPSB         EISP7129
      LOGICAL MATZ                                                      EISP7130
      EPSB = B(N,1)                                                     EISP7131
      ISW = 1                                                           EISP7132
C     .......... FIND EIGENVALUES OF QUASI-TRIANGULAR MATRICES.         EISP7133
C                FOR EN=N STEP -1 UNTIL 1 DO -- ..........              EISP7134
      DO 510 NN = 1, N                                                  EISP7135
         EN = N + 1 - NN                                                EISP7136
         NA = EN - 1                                                    EISP7137
         IF (ISW .EQ. 2) GO TO 505                                      EISP7138
         IF (EN .EQ. 1) GO TO 410                                       EISP7139
         IF (A(EN,NA) .NE. 0.0E0) GO TO 420                             EISP7140
C     .......... 1-BY-1 BLOCK, ONE REAL ROOT ..........                 EISP7141
  410    ALFR(EN) = A(EN,EN)                                            EISP7142
         IF (B(EN,EN) .LT. 0.0E0) ALFR(EN) = -ALFR(EN)                  EISP7143
         BETA(EN) = ABS(B(EN,EN))                                       EISP7144
```

```
      ALFI(EN) = 0.0E0
      GO TO 510
C     .......... 2-BY-2 BLOCK ..........
  420 IF (ABS(B(NA,NA)) .LE. EPSB) GO TO 455
      IF (ABS(B(EN,EN)) .GT. EPSB) GO TO 430
      A1 = A(EN,EN)
      A2 = A(EN,NA)
      BN = 0.0E0
      GO TO 435
  430 AN = ABS(A(NA,NA)) + ABS(A(NA,EN)) + ABS(A(EN,NA))
     X   + ABS(A(EN,EN))
      BN = ABS(B(NA,NA)) + ABS(B(NA,EN)) + ABS(B(EN,EN))
      A11 = A(NA,NA) / AN
      A12 = A(NA,EN) / AN
      A21 = A(EN,NA) / AN
      A22 = A(EN,EN) / AN
      B11 = B(NA,NA) / BN
      B12 = B(NA,EN) / BN
      B22 = B(EN,EN) / BN
      E = A11 / B11
      EI = A22 / B22
      S = A21 / (B11 * B22)
      T = (A22 - E * B22) / B22
      IF (ABS(E) .LE. ABS(EI)) GO TO 431
      E = EI
      T = (A11 - E * B11) / B11
  431 C = 0.5E0 * (T - S * B12)
      D = C * C + S * (A12 - E * B12)
      IF (D .LT. 0.0E0) GO TO 480
C     .......... TWO REAL ROOTS.
C                ZERO BOTH A(EN,NA) AND B(EN,NA) ..........
      E = E + (C + SIGN(SQRT(D),C))
      A11 = A11 - E * B11
      A12 = A12 - E * B12
      A22 = A22 - E * B22
      IF (ABS(A11) + ABS(A12) .LT.
     X    ABS(A21) + ABS(A22)) GO TO 432
      A1 = A12
      A2 = A11
      GO TO 435
  432 A1 = A22
      A2 = A21
C     .......... CHOOSE AND APPLY REAL Z ..........
  435 S = ABS(A1) + ABS(A2)
      U1 = A1 / S
      U2 = A2 / S
      R = SIGN(SQRT(U1*U1+U2*U2),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      U2 = V2 / V1
C
      DO 440 I = 1, EN
         T = A(I,EN) + U2 * A(I,NA)
         A(I,EN) = A(I,EN) + T * V1
         A(I,NA) = A(I,NA) + T * V2
         T = B(I,EN) + U2 * B(I,NA)
         B(I,EN) = B(I,EN) + T * V1
         B(I,NA) = B(I,NA) + T * V2
  440 CONTINUE
C
```

```
      IF (.NOT. MATZ) GO TO 450
C
      DO 445 I = 1, N
         T = Z(I,EN) + U2 * Z(I,NA)
         Z(I,EN) = Z(I,EN) + T * V1
         Z(I,NA) = Z(I,NA) + T * V2
  445 CONTINUE
C
  450 IF (BN .EQ. 0.0E0) GO TO 475
      IF (AN .LT. ABS(E) * BN) GO TO 455
      A1 = B(NA,NA)
      A2 = B(EN,NA)
      GO TO 460
  455 A1 = A(NA,NA)
      A2 = A(EN,NA)
C     .......... CHOOSE AND APPLY REAL Q ..........
  460 S = ABS(A1) + ABS(A2)
      IF (S .EQ. 0.0E0) GO TO 475
      U1 = A1 / S
      U2 = A2 / S
      R = SIGN(SQRT(U1*U1+U2*U2),U1)
      V1 = -(U1 + R) / R
      V2 = -U2 / R
      U2 = V2 / V1
C
      DO 470 J = NA, N
         T = A(NA,J) + U2 * A(EN,J)
         A(NA,J) = A(NA,J) + T * V1
         A(EN,J) = A(EN,J) + T * V2
         T = B(NA,J) + U2 * B(EN,J)
         B(NA,J) = B(NA,J) + T * V1
         B(EN,J) = B(EN,J) + T * V2
  470 CONTINUE
C
  475 A(EN,NA) = 0.0E0
      B(EN,NA) = 0.0E0
      ALFR(NA) = A(NA,NA)
      ALFR(EN) = A(EN,EN)
      IF (B(NA,NA) .LT. 0.0E0) ALFR(NA) = -ALFR(NA)
      IF (B(EN,EN) .LT. 0.0E0) ALFR(EN) = -ALFR(EN)
      BETA(NA) = ABS(B(NA,NA))
      BETA(EN) = ABS(B(EN,EN))
      ALFI(EN) = 0.0E0
      ALFI(NA) = 0.0E0
      GO TO 505
C     .......... TWO COMPLEX ROOTS ..........
  480 E = E + C
      EI = SQRT(-D)
      A11R = A11 - E * B11
      A11I = EI * B11
      A12R = A12 - E * B12
      A12I = EI * B12
      A22R = A22 - E * B22
      A22I = EI * B22
      IF (ABS(A11R) + ABS(A11I) + ABS(A12R) + ABS(A12I) .LT.
     X    ABS(A21) + ABS(A22R) + ABS(A22I)) GO TO 482
      A1 = A12R
      A1I = A12I
      A2 = -A11R
      A2I = -A11I
```

```
      GO TO 485                                                         EISP7265
  482 A1 = A22R                                                         EISP7266
      A1I = A22I                                                        EISP7267
      A2 = -A21                                                         EISP7268
      A2I = 0.0E0                                                       EISP7269
C     .......... CHOOSE COMPLEX Z ..........                            EISP7270
  485 CZ = SQRT(A1*A1+A1I*A1I)                                          EISP7271
      IF (CZ .EQ. 0.0E0) GO TO 487                                      EISP7272
      SZR = (A1 * A2 + A1I * A2I) / CZ                                  EISP7273
      SZI = (A1 * A2I - A1I * A2) / CZ                                  EISP7274
      R = SQRT(CZ*CZ+SZR*SZR+SZI*SZI)                                   EISP7275
      CZ = CZ / R                                                       EISP7276
      SZR = SZR / R                                                     EISP7277
      SZI = SZI / R                                                     EISP7278
      GO TO 490                                                         EISP7279
  487 SZR = 1.0E0                                                       EISP7280
      SZI = 0.0E0                                                       EISP7281
  490 IF (AN .LT. (ABS(E) + EI) * BN) GO TO 492                         EISP7282
      A1 = CZ * B11 + SZR * B12                                         EISP7283
      A1I = SZI * B12                                                   EISP7284
      A2 = SZR * B22                                                    EISP7285
      A2I = SZI * B22                                                   EISP7286
      GO TO 495                                                         EISP7287
  492 A1 = CZ * A11 + SZR * A12                                         EISP7288
      A1I = SZI * A12                                                   EISP7289
      A2 = CZ * A21 + SZR * A22                                         EISP7290
      A2I = SZI * A22                                                   EISP7291
C     .......... CHOOSE COMPLEX Q ..........                            EISP7292
  495 CQ = SQRT(A1*A1+A1I*A1I)                                          EISP7293
      IF (CQ .EQ. 0.0E0) GO TO 497                                      EISP7294
      SQR = (A1 * A2 + A1I * A2I) / CQ                                  EISP7295
      SQI = (A1 * A2I - A1I * A2) / CQ                                  EISP7296
      R = SQRT(CQ*CQ+SQR*SQR+SQI*SQI)                                   EISP7297
      CQ = CQ / R                                                       EISP7298
      SQR = SQR / R                                                     EISP7299
      SQI = SQI / R                                                     EISP7300
      GO TO 500                                                         EISP7301
  497 SQR = 1.0E0                                                       EISP7302
      SQI = 0.0E0                                                       EISP7303
C     .......... COMPUTE DIAGONAL ELEMENTS THAT WOULD RESULT            EISP7304
C                IF TRANSFORMATIONS WERE APPLIED ..........             EISP7305
  500 SSR = SQR * SZR + SQI * SZI                                       EISP7306
      SSI = SQR * SZI - SQI * SZR                                       EISP7307
      I = 1                                                             EISP7308
      TR = CQ * CZ * A11 + CQ * SZR * A12 + SQR * CZ * A21              EISP7309
     X   + SSR * A22                                                    EISP7310
      TI = CQ * SZI * A12 - SQI * CZ * A21 + SSI * A22                  EISP7311
      DR = CQ * CZ * B11 + CQ * SZR * B12 + SSR * B22                   EISP7312
      DI = CQ * SZI * B12 + SSI * B22                                   EISP7313
      GO TO 503                                                         EISP7314
  502 I = 2                                                             EISP7315
      TR = SSR * A11 - SQR * CZ * A12 - CQ * SZR * A21                  EISP7316
     X   + CQ * CZ * A22                                                EISP7317
      TI = -SSI * A11 - SQI * CZ * A12 + CQ * SZI * A21                 EISP7318
      DR = SSR * B11 - SQR * CZ * B12 + CQ * CZ * B22                   EISP7319
      DI = -SSI * B11 - SQI * CZ * B12                                  EISP7320
  503 T = TI * DR - TR * DI                                             EISP7321
      J = NA                                                            EISP7322
      IF (T .LT. 0.0E0) J = EN                                          EISP7323
      R = SQRT(DR*DR+DI*DI)                                             EISP7324
```

```
      BETA(J) = BN * R
      ALFR(J) = AN * (TR * DR + TI * DI) / R
      ALFI(J) = AN * T / R
      IF (I .EQ. 1) GO TO 502
  505 ISW = 3 - ISW
  510 CONTINUE
      B(N,1) = EPSB
C
      RETURN
C**  THIS PROGRAM VALID ON FTN4 AND FTN5 **
      END
C        ROUTINE NAME      - PF263=QZVEC
C        FROM EISPACK
C
C-----------------------------------------------------------------------
C
C   LATEST REVISION        - AUGUST 1,1984
C                            COMPUTER SCIENCES CORP., HAMPTON, VA.
C
C
C   PURPOSE                - THIS SUBROUTINE ACCEPTS A PAIR OF REAL
C                            MATRICES, ONE OF THEM IN QUASI-TRIANGULAR
C                            FORM (IN WHICH EACH 2-BY-2 BLOCK CORRESPONDS
C                            TO A PAIR OF COMPLEX EIGENVALUES) AND THE
C                            OTHER IN UPPER TRIANGULAR FORM.  IT COMPUTES
C                            THE EIGENVECTORS OF THE TRIANGULAR PROBLEM
C                            AND TRANSFORMS THE RESULTS BACK TO THE
C                            ORIGINAL COORDINATE SYSTEM.  IT IS USUALLY
C                            PRECEDED BY QZHES(PF260), QZIT(PF261), AND
C                            QZVAL(PF262).
C
C
C   USAGE                  - CALL QZVEC(NM,N,A,B,ALFR,ALFI,BETA,Z)
C
C   ARGUMENTS    NM        - ON INPUT NM MUST BE SET TO THE ROW DIMENSION
C                            OF TWO-DIMENSIONAL ARRAY PARAMETERS AS
C                            DECLARED IN THE CALLING PROGRAM DIMENSION
C                            STATEMENT.
C
C                N         - ON INPUT N IS THE ORDER OF THE MATRICES.
C
C                A         - ON INPUT A CONTAINS A REAL UPPER QUASI-
C                            TRIANGULAR MATRIX.
C                            MUST BE OF DIMENSION NM X N.
C
C                          - ON OUTPUT A IS UNALTERED.  ITS SUBDIAGONAL
C                            ELEMENTS PROVIDE INFORMATION ABOUT THE STORAGE
C                            OF THE COMPLEX EIGENVECTORS.
C
C                B         - ON INPUT B CONTAINS A REAL UPPER TRIANGULAR
C                            MATRIX.  IN ADDITION, LOCATION B(N,1) CONTAINS
C                            THE TOLERANCE QUANTITY (EPSB) COMPUTED AND
C                            SAVED IN QZIT(PF261).
C                            MUST BE OF DIMENSION NM X N.
C
C                            ON OUTPUT B HAS BEEN DESTROYED.
C
C                ALFR      - ON INPUT ALFR IS A VECTOR SUCH THAT THE
C                            RATIOS ((ALFR+I*ALFI)/BETA) ARE THE
C                            GENERALIZED EIGENVALUES.  THEY ARE USUALLY
```

```
C                       OBTAINED FROM QZVAL(PF262).
C                       MUST BE OF DIMENSION N.
C
C             ALFI    - ON INPUT ALFI IS A VECTOR SUCH THAT THE RATIOS
C                       ((ALFR+I*ALFI)/BETA) ARE THE GENERALIZED
C                       EIGENVALUES.  THEY ARE USUALLY OBTAINED FROM
C                       QZVAL(PF262).
C                       MUST BE OF DIMENSION N.
C
C             BETA    - ON INPUT BETA IS A VECTOR SUCH THAT THE RATIOS
C                       ((ALFR+I*ALFI)/BETA) ARE THE GENERALIZED
C                       EIGENVALUES.  THEY ARE USUALLY OBTAINED FROM
C                       QZVAL(PF262).
C                       MUST BE OF DIMENSION N.
C
C             Z       - ON INPUT Z CONTAINS THE TRANSFORMATION MATRIX
C                       PRODUCED IN THE REDUCTIONS BY QZHES(PF260),
C                       QZIT(PF261), AND QZVAL(PF262), IF PERFORMED.
C                       IF THE EIGENVECTORS OF THE TRIANGULAR PROBLEM
C                       ARE DESIRED, Z MUST CONTAIN THE IDENTITY
C                       MATRIX.
C                       MUST BE OF DIMENSION NM X N.
C
C                       ON OUTPUT Z CONTAINS THE REAL AND IMAGINARY
C                       PARTS OF THE EIGENVECTORS. IF ALFI(I) .EQ.
C                       0.0, THE I-TH EIGENVALUE IS REAL AND THE I-TH
C                       COLUMN OF Z CONTAINS ITS EIGENVECTOR.  IF
C                       ALFI(I) .NE. 0.0, THE I-TH EIGENVALUE IS
C                       COMPLEX.  IF ALFI(I) .GT. 0.0, THE EIGENVALUE
C                       IS THE FIRST OF A COMPLEX PAIR AND THE I-TH
C                       AND (I+1)-TH COLUMNS OF Z CONTAIN ITS EIGEN-
C                       VECTOR.  IF ALFI(I) .LT. 0.0, THE EIGEN-
C                        VALUE IS THE SECOND OF A COMPLEX PAIR AND THE
C                       (I-1)-TH AND I-TH COLUMNS OF Z CONTAIN THE
C                       CONJUGATE OF ITS EIGENVECTOR.  EACH EIGEN-
C                       VECTOR IS NORMALIZED SO THAT THE MODULUS
C                       OF ITS LARGEST COMPONENT IS 1.0 .
C
C
C   REQUIRED ROUTINES   - NONE
C
C   REMARKS   1.   THIS SUBROUTINE IS THE OPTIONAL FOURTH STEP
C                  OF THE QZ ALGORITHM FOR SOLVING GENERALIZED
C                  MATRIX EIGENVALUE PROBLEMS, SIAM J. NUMER.
C                  ANAL. 10, 241-256(1973) BY MOLER AND STEWART.
C
C
C   EXAMPLE :
C     PROGRAM TQZVEC(OUTPUT,TAPE6=OUTPUT)
C     DIMENSION A(5,5),B(5,5),ALFR(5),ALFI(5),BETA(5),Z(5,5)
C     LOGICAL MATZ
C
C     N = 5
C     NM = 5
C     MATZ = .TRUE.
C     EPS1 = 0.0E0
C
C     DATA A /10.,2.,3.,2*1.,2.,12.,1.,2.,1.,3.,1.,11.,
C    *        1.,-1.,1.,2.,1.,9.,3*1.,-1.,1.,15.   /
C
C     DATA B /12.,1.,-1.,2.,2*1.,14.,1.,-1.,1.,-1.,1.,
```

```
C     *        16.,-1.,1.,2.,-1.,-1.,12.,-1.,3*1.,-1.,11.  /                QZVEC111
C                                                                          QZVEC112
C      CALL  QZHES(NM,N,A,B,MATZ,Z)                                        QZVEC113
C      CALL  QZIT(NM,N,A,B,EPS1,MATZ,Z,IERR)                               QZVEC114
C      CALL  QZVAL(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z)                         QZVEC115
C      CALL QZVEC(NM,N,A,B,ALFR,ALFI,BETA,Z)                               QZVEC116
C      WRITE(6,99) IERR                                                    QZVEC117
C      WRITE(6,100) ((Z(I,J),I=1,5),J=1,5)                                 QZVEC118
C99    FORMAT(1H1,7HIERR = ,I4)                                            QZVEC119
C100   FORMAT(5H Z = /5(1H ,5(G8.2,2X)/))                                  QZVEC120
C      STOP                                                                QZVEC121
C      END                                                                 QZVEC122
C                                                                          QZVEC123
C      OUTPUT :                                                            QZVEC124
C                                                                          QZVEC125
C      IERR =    0                                                         QZVEC126
C      Z =                                                                 QZVEC127
C      .26      -.59E-01   .23       -.30      -1.0                        QZVEC128
C     -.85       .39       1.0       -.69       .26                        QZVEC129
C      1.0       1.0       .85        .88       .54E-01                    QZVEC130
C     -1.0       .83      -.39        .72      -.46                        QZVEC131
C     -.45      -.84       .65        1.0      -.19E-01                    QZVEC132
C                                                                          QZVEC133
C------------------------------------------------------------------------- QZVEC134
      SUBROUTINE QZVEC(NM,N,A,B,ALFR,ALFI,BETA,Z)                          EISP7336
C
      implicit real*8 (a-h,o-z)                                            EISP7337
      INTEGER I,J,K,M,N,EN,II,JJ,NA,NM,NN,ISW,ENM2                         EISP7338
      REAL*8 A(NM,N),B(NM,N),ALFR(N),ALFI(N),BETA(N),Z(NM,N)               EISP7339
      REAL*8 D,Q,R,S,T,W,X,Y,DI,DR,RA,RR,SA,TI,TR,T1,T2,W1,X1,             EISP7340
     X       ZZ,Z1,ALFM,ALMI,ALMR,BETM,EPSB                                EISP7341
      EPSB = B(N,1)                                                        EISP7342
      ISW = 1                                                              EISP7343
C     .......... FOR EN=N STEP -1 UNTIL 1 DO -- ..........                 EISP7344
      DO 800 NN = 1, N                                                     EISP7345
         EN = N + 1 - NN                                                   EISP7346
         NA = EN - 1                                                       EISP7347
         IF (ISW .EQ. 2) GO TO 795                                         EISP7348
         IF (ALFI(EN) .NE. 0.0E0) GO TO 710                                EISP7349
C     .......... REAL VECTOR ..........                                    EISP7350
         M = EN                                                            EISP7351
         B(EN,EN) = 1.0E0                                                  EISP7352
         IF (NA .EQ. 0) GO TO 800                                          EISP7353
         ALFM = ALFR(M)                                                    EISP7354
         BETM = BETA(M)                                                    EISP7355
C     .......... FOR I=EN-1 STEP -1 UNTIL 1 DO -- ..........               EISP7356
         DO 700 II = 1, NA                                                 EISP7357
            I = EN - II                                                    EISP7358
            W = BETM * A(I,I) - ALFM * B(I,I)                              EISP7359
            R = 0.0E0                                                      EISP7360
C                                                                          EISP7361
            DO 610 J = M, EN                                               EISP7362
  610       R = R + (BETM * A(I,J) - ALFM * B(I,J)) * B(J,EN)              EISP7363
C                                                                          EISP7364
            IF (I .EQ. 1 .OR. ISW .EQ. 2) GO TO 630                        EISP7365
            IF (BETM * A(I,I-1) .EQ. 0.0E0) GO TO 630                      EISP7366
            ZZ = W                                                         EISP7367
            S = R                                                          EISP7368
            GO TO 690                                                      EISP7369
  630       M = I                                                          EISP7370
```

```
            IF (ISW .EQ. 2) GO TO 640                                   EISP7371
C     .......... REAL 1-BY-1 BLOCK ..........                           EISP7372
            T = W                                                       EISP7373
            IF (W .EQ. 0.0E0) T = EPSB                                  EISP7374
            B(I,EN) = -R / T                                            EISP7375
            GO TO 700                                                   EISP7376
C     .......... REAL 2-BY-2 BLOCK ..........                           EISP7377
  640       X = BETM * A(I,I+1) - ALFM * B(I,I+1)                       EISP7378
            Y = BETM * A(I+1,I)                                         EISP7379
            Q = W * ZZ - X * Y                                          EISP7380
            T = (X * S - ZZ * R) / Q                                    EISP7381
            B(I,EN) = T                                                 EISP7382
            IF (ABS(X) .LE. ABS(ZZ)) GO TO 650                          EISP7383
            B(I+1,EN) = (-R - W * T) / X                                EISP7384
            GO TO 690                                                   EISP7385
  650       B(I+1,EN) = (-S - Y * T) / ZZ                               EISP7386
  690       ISW = 3 - ISW                                               EISP7387
  700    CONTINUE                                                       EISP7388
C     .......... END REAL VECTOR ..........                             EISP7389
         GO TO 800                                                      EISP7390
C     .......... COMPLEX VECTOR ..........                              EISP7391
  710    M = NA                                                         EISP7392
         ALMR = ALFR(M)                                                 EISP7393
         ALMI = ALFI(M)                                                 EISP7394
         BETM = BETA(M)                                                 EISP7395
C     .......... LAST VECTOR COMPONENT CHOSEN IMAGINARY SO THAT         EISP7396
C                EIGENVECTOR MATRIX IS TRIANGULAR ..........            EISP7397
         Y = BETM * A(EN,NA)                                            EISP7398
         B(NA,NA) = -ALMI * B(EN,EN) / Y                                EISP7399
         B(NA,EN) = (ALMR * B(EN,EN) - BETM * A(EN,EN)) / Y             EISP7400
         B(EN,NA) = 0.0E0                                               EISP7401
         B(EN,EN) = 1.0E0                                               EISP7402
         ENM2 = NA - 1                                                  EISP7403
         IF (ENM2 .EQ. 0) GO TO 795                                     EISP7404
C     .......... FOR I=EN-2 STEP -1 UNTIL 1 DO -- ..........            EISP7405
         DO 790 II = 1, ENM2                                            EISP7406
            I = NA - II                                                 EISP7407
            W = BETM * A(I,I) - ALMR * B(I,I)                           EISP7408
            W1 = -ALMI * B(I,I)                                         EISP7409
            RA = 0.0E0                                                  EISP7410
            SA = 0.0E0                                                  EISP7411
C                                                                       EISP7412
            DO 760 J = M, EN                                            EISP7413
               X = BETM * A(I,J) - ALMR * B(I,J)                        EISP7414
               X1 = -ALMI * B(I,J)                                      EISP7415
               RA = RA + X * B(J,NA) - X1 * B(J,EN)                     EISP7416
               SA = SA + X * B(J,EN) + X1 * B(J,NA)                     EISP7417
  760       CONTINUE                                                    EISP7418
C                                                                       EISP7419
            IF (I .EQ. 1 .OR. ISW .EQ. 2) GO TO 770                     EISP7420
            IF (BETM * A(I,I-1) .EQ. 0.0E0) GO TO 770                   EISP7421
            ZZ = W                                                      EISP7422
            Z1 = W1                                                     EISP7423
            R = RA                                                      EISP7424
            S = SA                                                      EISP7425
            ISW = 2                                                     EISP7426
            GO TO 790                                                   EISP7427
  770       M = I                                                       EISP7428
            IF (ISW .EQ. 2) GO TO 780                                   EISP7429
C     .......... COMPLEX 1-BY-1 BLOCK ..........                        EISP7430
```

```
            TR = -RA                                                    EISP7431
            TI = -SA                                                    EISP7432
  773       DR = W                                                      EISP7433
            DI = W1                                                     EISP7434
C     .......... COMPLEX DIVIDE (T1,T2) = (TR,TI) / (DR,DI) .......... EISP7435
  775       IF (ABS(DI) .GT. ABS(DR)) GO TO 777                         EISP7436
            RR = DI / DR                                                EISP7437
            D = DR + DI * RR                                            EISP7438
            T1 = (TR + TI * RR) / D                                     EISP7439
            T2 = (TI - TR * RR) / D                                     EISP7440
            GO TO (787,782), ISW                                        EISP7441
      CALL GOTOER                                                       EISP7442
  777       RR = DR / DI                                                EISP7443
            D = DR * RR + DI                                            EISP7444
            T1 = (TR * RR + TI) / D                                     EISP7445
            T2 = (TI * RR - TR) / D                                     EISP7446
            GO TO (787,782), ISW                                        EISP7447
      CALL GOTOER                                                       EISP7448
C     .......... COMPLEX 2-BY-2 BLOCK ..........                        EISP7449
  780       X = BETM * A(I,I+1) - ALMR * B(I,I+1)                       EISP7450
            X1 = -ALMI * B(I,I+1)                                       EISP7451
            Y = BETM * A(I+1,I)                                         EISP7452
            TR = Y * RA - W * R + W1 * S                                EISP7453
            TI = Y * SA - W * S - W1 * R                                EISP7454
            DR = W * ZZ - W1 * Z1 - X * Y                               EISP7455
            DI = W * Z1 + W1 * ZZ - X1 * Y                              EISP7456
            IF (DR .EQ. 0.0E0 .AND. DI .EQ. 0.0E0) DR = EPSB            EISP7457
            GO TO 775                                                   EISP7458
  782       B(I+1,NA) = T1                                              EISP7459
            B(I+1,EN) = T2                                              EISP7460
            ISW = 1                                                     EISP7461
            IF (ABS(Y) .GT. ABS(W) + ABS(W1)) GO TO 785                 EISP7462
            TR = -RA - X * B(I+1,NA) + X1 * B(I+1,EN)                   EISP7463
            TI = -SA - X * B(I+1,EN) - X1 * B(I+1,NA)                   EISP7464
            GO TO 773                                                   EISP7465
  785       T1 = (-R - ZZ * B(I+1,NA) + Z1 * B(I+1,EN)) / Y             EISP7466
            T2 = (-S - ZZ * B(I+1,EN) - Z1 * B(I+1,NA)) / Y             EISP7467
  787       B(I,NA) = T1                                                EISP7468
            B(I,EN) = T2                                                EISP7469
  790    CONTINUE                                                       EISP7470
C     .......... END COMPLEX VECTOR ..........                          EISP7471
  795    ISW = 3 - ISW                                                  EISP7472
  800 CONTINUE                                                          EISP7473
C     .......... END BACK SUBSTITUTION.                                 EISP7474
C                TRANSFORM TO ORIGINAL COORDINATE SYSTEM.               EISP7475
C                FOR J=N STEP -1 UNTIL 1 DO -- ..........               EISP7476
      DO 880 JJ = 1, N                                                  EISP7477
         J = N + 1 - JJ                                                 EISP7478
C                                                                       EISP7479
         DO 880 I = 1, N                                                EISP7480
            ZZ = 0.0E0                                                  EISP7481
C                                                                       EISP7482
            DO 860 K = 1, J                                             EISP7483
  860       ZZ = ZZ + Z(I,K) * B(K,J)                                   EISP7484
C                                                                       EISP7485
            Z(I,J) = ZZ                                                 EISP7486
  880 CONTINUE                                                          EISP7487
C     .......... NORMALIZE SO THAT MODULUS OF LARGEST                   EISP7488
C                COMPONENT OF EACH VECTOR IS 1.                         EISP7489
C                (ISW IS 1 INITIALLY FROM BEFORE) ..........            EISP7490
```

```
      DO 950 J = 1, N
         D = 0.0E0
         IF (ISW .EQ. 2) GO TO 920
         IF (ALFI(J) .NE. 0.0E0) GO TO 945
C
         DO 890 I = 1, N
            IF (ABS(Z(I,J)) .GT. D) D = ABS(Z(I,J))
  890    CONTINUE
C
         DO 900 I = 1, N
  900    Z(I,J) = Z(I,J) / D
C
         GO TO 950
C
  920    DO 930 I = 1, N
            R = ABS(Z(I,J-1)) + ABS(Z(I,J))
            IF (R .NE. 0.0E0) R = R * SQRT((Z(I,J-1)/R)**2
     X                                     +(Z(I,J)/R)**2)
            IF (R .GT. D) D = R
  930    CONTINUE
C
         DO 940 I = 1, N
            Z(I,J-1) = Z(I,J-1) / D
            Z(I,J) = Z(I,J) / D
  940    CONTINUE
C
  945    ISW = 3 - ISW
  950 CONTINUE
C
      RETURN
      END
C        ROUTINE NAME    - PF266=RGG
C        FROM EISPACK
C
C-----------------------------------------------------------------------
C
C   LATEST REVISION      - AUGUST 1,1984
C                          COMPUTER SCIENCES CORP., HAMPTON, VA.
C
C
C   PURPOSE              - THIS SUBROUTINE CALLS THE RECOMMENDED
C                          SEQUENCE OF SUBROUTINES FROM THE EIGENSYSTEM
C                          SUBROUTINE PACKAGE (EISPACK) TO FIND THE
C                          EIGENVALUES AND EIGENVECTORS (IF DESIRED) FOR
C                          THE REAL GENERAL GENERALIZED EIGENPROBLEM AX
C                          = (LAMBDA)BX.
C
C
C   USAGE                - CALL RGG(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z,IERR)
C
C   ARGUMENTS     NM     - ON INPUT NM MUST BE SET TO THE ROW DIMENSION
C                          OF THE TWO-DIMENSIONAL ARRAY PARAMETERS AS
C                          DECLARED IN THE CALLING PROGRAM DIMENSION
C                          STATEMENT.
C
C                 N      - ON INPUT N IS THE ORDER OF THE MATRICES A
C                          AND B.
C
C                 A      - ON INPUT A CONTAINS A REAL GENERAL MATRIX.
C                          MUST BE OF DIMENSION NM X N.
```

```
C
C         B       - ON INPUT B CONTAINS A REAL GENERAL MATRIX.
C                   MUST BE OF DIMENSION NM X N.
C
C         ALFR    - ON OUTPUT ALFR CONTAINS THE REAL PART OF THE
C                   NUMERATORS OF THE EIGENVALUES.
C                   MUST BE OF DIMENSION N.
C
C         ALFI    - ON OUTPUT ALFI CONTAINS THE IMAGINARY PART OF
C                   THE NUMERATORS OF THE EIGENVALUES.
C                   MUST BE OF DIMENSION N.
C
C         BETA    - ON OUTPUT BETA CONTAINS  THE DENOMINATORS OF
C                   THE EIGENVALUES, WHICH ARE THUS GIVEN
C                   BY THE RATIOS (ALFR+I*ALFI)/BETA.
C                   COMPLEX CONJUGATE PAIRS OF EIGENVALUES APPEAR
C                   CONSECUTIVELY  WITH THE EIGENVALUE
C                   HAVING THE POSITIVE IMAGINARY PART FIRST.
C                   MUST BE OF DIMENSION N.
C
C         MATZ    - ON INPUT MATZ IS AN INTEGER VARIABLE SET EQUAL
C                   TO ZERO IF  ONLY EIGENVALUES ARE
C                   DESIRED.  OTHERWISE IT IS SET TO
C                   ANY NON-ZERO INTEGER FOR BOTH EIGENVALUES AND
C                   EIGENVECTORS.
C
C         Z       - ON OUTPUT Z CONTAINS THE REAL AND IMAGINARY
C                   PARTS OF THE EIGENVECTORS IF MATZ IS NOT
C                   ZERO.  IF THE J-TH EIGENVALUE IS REAL, THE
C                   J-TH COLUMN OF Z CONTAINS ITS
C                   EIGENVECTOR.  IF THE J-TH
C                   EIGENVALUE IS COMPLEX WITH POSITIVE IMAGINARY
C                   PART, THE J-TH AND (J+1)-TH
C                   COLUMNS OF Z CONTAIN THE REAL AND
C                   IMAGINARY PARTS OF ITS EIGENVECTOR.  THE
C                   CONJUGATE OF THIS VECTOR IS THE
C                   EIGENVECTOR FOR THE CONJUGATE EIGENVALUE.
C                   MUST BE OF DIMENSION NM X N.
C
C         IERR    - ON OUTPUT IERR IS AN INTEGER OUTPUT VARIABLE
C                   SET EQUAL TO AN ERROR COMPLETION CODE
C                   DESCRIBED IN THE DOCUMENTATION FOR QZIT
C                   PF261). THE NORMAL COMPLETION CODE IS ZERO.
C
C
C  REQUIRED ROUTINES    - PF260=QZHES,PF261=QZIT,PF262=QZVAL,PF263=QZVEC
C                   HC318=EPSLON
C
C  REMARKS  1.  REFERENCES
C
C           FROM THE EISPACK PACKAGE OF EIGENSYSTEM ROUTINES.
C
C        2.  SUBROUTINE RGG IS A DRIVER ROUTINE WHICH CALLS ROUTINES
C            QZHES(PF260), QZIT(PF261), QZVAL(PF262), AND
C            QZVEC(PF263).
C
C            QZHES(PF260) ACCEPTS A PAIR OF  REAL  GENERAL  MATRICES
C            AND REDUCES ONE OF THEM TO UPPER  HESSENBERG  FORM  AND
C            THE  OTHER  TO  UPPER  TRIANGULAR FORM USING ORTHOGONAL
C            TRANSFORMATIONS.
```

```
C
C              QZIT(PF261)  ACCEPTS  A  PAIR  OF REAL MATRICES, ONE OF
C              THEM IN UPPER HESSENBERG FORM AND THE  OTHER  IN  UPPER
C              TRIANGULAR  FORM.   IT REDUCES THE HESSENBERG MATRIX TO
C              QUASI-TRIANGULAR FORM USING ORTHOGONAL  TRANSFORMATIONS
C              WHILE  MAINTAINING  THE  TRIANGULAR  FORM  OF THE OTHER
C              MATRIX.
C
C              QZVAL(PF262)  ACCEPTS  A  PAIR OF REAL MATRICES, ONE OF
C              THEM IN QUASI-TRIANGULAR FORM AND THE  OTHER  IN  UPPER
C              TRIANGULAR   FORM.   IT  REDUCES  THE  QUASI-TRIANGULAR
C              MATRIX FURTHER, SO THAT  ANY  REMAINING  2-BY-2  BLOCKS
C              CORRESPOND TO PAIRS OF COMPLEX EIGENVALUES, AND RETURNS
C              QUANTITIES   WHOSE   RATIOS   GIVE   THE    GENERALIZED
C              EIGENVALUES.
C
C              QZVEC(PF263) ACCEPTS A PAIR OF REAL  MATRICES,  ONE  OF
C              THEM  IN  QUASI-TRIANGULAR  FORM  (IN WHICH EACH 2-BY-2
C              BLOCK CORRESPONDS TO A PAIR OF COMPLEX EIGENVALUES) AND
C              THE  OTHER  IN  UPPER TRIANGULAR FORM.  IT COMPUTES THE
C              EIGENVECTORS OF THE TRIANGULAR PROBLEM  AND  TRANSFORMS
C              THE RESULTS BACK TO THE ORIGINAL COORDINATE SYSTEM.
C
C   EXAMPLE :
C
C
C     PROGRAM TRGG(OUTPUT,TAPE6=OUTPUT)
C
C
C     DIMENSION A(5,5),B(5,5),ALFR(5),ALFI(5),BETA(5),Z(5,5)
C
C     N = 5
C     NM = 5
C     MATZ = 1
C
C     DATA A /10.,2.,3.,2*1.,2.,12.,1.,2.,1.,3.,1.,11.,
C    *       1.,-1.,1.,2.,1.,9.,3*1.,-1.,1.,15.   /
C
C     DATA B /12.,1.,-1.,2.,2*1.,14.,1.,-1.,1.,-1.,1.,
C    *       16.,-1.,1.,2.,-1.,-1.,12.,-1.,3*1.,-1.,11.  /
C
C     CALL RGG(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z,IERR)
C
C     WRITE(6,99) IERR
C     WRITE(6,100) ALFR,ALFI,BETA,((Z(I,J),I=1,5),J=1,5)
C99   FORMAT(1H1,7HIERR = ,I4)
C100  FORMAT(1H0,7HALFR = /1H ,5(G8.2,2X)/
C    *       8H0ALFI = /1H ,5(G8.2,2X)/
C    *       8H0BETA = /1H ,5(G8.2,2X)/
C    *       5H0Z = /5(1H ,5(G8.2,2X)/))
C     STOP
C     END
C
C     OUTPUT :
C
C     IERR =    0
C     ALFR =
C     15.       7.2       16.       10.       8.6
C     ALFI =
C     0.        0.        0.        0.        0.
```

```
C     BETA =                                                            RGG  151
C     9.9        17.        14.        11.        13.                   RGG  152
C     Z =                                                               RGG  153
C     .26       -.59E-01   .23       -.30       -1.0                    RGG  154
C    -.85        .39       1.0       -.69        .26                    RGG  155
C     1.0        1.0       .85        .88        .54E-01                RGG  156
C    -1.0        .83      -.39        .72       -.46                    RGG  157
C    -.45       -.84       .65        1.0       -.19E-01                RGG  158
C                                                                       RGG  159
C---------------------------------------------------------------------- RGG  160
      SUBROUTINE diverg(NM,N,A,B,ALFR,ALFI,BETA,MATZ,Z,IERR)               EISP7
C
      implicit real*8 (a-h,o-z)                                         EISP7613
      INTEGER N,NM,IERR,MATZ                                            EISP7614
      REAL*8 A(NM,N),B(NM,N),ALFR(N),ALFI(N),BETA(N),Z(NM,N)            EISP7615
      LOGICAL TF                                                        EISP7616
      zero = 0.0e+00
      IF (N .LE. NM) GO TO 10                                           EISP7617
      IERR = 10 * N                                                     EISP7618
      GO TO 50                                                          EISP7619
C                                                                       EISP7620
   10 IF (MATZ .NE. 0) GO TO 20                                         EISP7621
C     .......... FIND EIGENVALUES ONLY ..........                       EISP7622
      TF = .FALSE.                                                      EISP7623
      CALL  QZHES(NM,N,A,B,TF,Z)                                        EISP7624
      CALL  QZIT(NM,N,A,B,zero ,TF,Z,IERR)                              EISP7625
      CALL  QZVAL(NM,N,A,B,ALFR,ALFI,BETA,TF,Z)                         EISP7626
      GO TO 50                                                          EISP7627
C     .......... FIND BOTH EIGENVALUES AND EIGENVECTORS ..........      EISP7628
   20 TF = .TRUE.                                                       EISP7629
      CALL  QZHES(NM,N,A,B,TF,Z)                                        EISP7630
      CALL  QZIT(NM,N,A,B,zero ,TF,Z,IERR)                              EISP7631
      CALL  QZVAL(NM,N,A,B,ALFR,ALFI,BETA,TF,Z)                         EISP7632
      IF (IERR .NE. 0) GO TO 50                                         EISP7633
      CALL  QZVEC(NM,N,A,B,ALFR,ALFI,BETA,Z)                            EISP7634
   50 RETURN                                                            EISP7635
C**  THIS PROGRAM VALID ON FTN4 AND FTN5 **                             EISP7636
      END
      subroutine gotoer
      write(6,10)
   10 format('there is an error in calculating subroutine')
      return
      end                                                               EISP7637
c
```