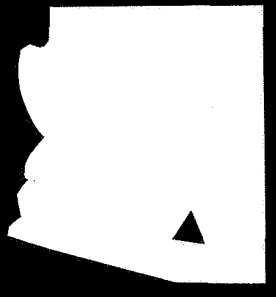
GOSPEL - A GENERAL OPTIMIZATION SOFTWARE PACKAGE FOR ELECTRICAL NETWORK DESIGN

Prepared under Grant NGL-03-002-136 for the
National Aeronautics and Space Administration

by

L. P. Huelsman



ENGINEERING EXPERIMENT STATION
COLLEGE OF ENGINEERING
THE UNIVERSITY OF ARIZONA
TUCSON, ARIZONA

GOSPEL - A GENERAL OPTIMIZATION SOFTWARE PACKAGE FOR ELECTRICAL NETWORK DESIGN

Prepared under Grant NGL-03-002-136 for the
National Aeronautics and Space Administration

by

L. P. Huelsman

Department of Electrical Engineering
University of Arizona
Tucson, Arizona

Abstract: This report describes GOSPEL, a General Optimization Software Package for ELectrical network design. The package may be applied to a wide range of optimization problems. It consists of a series of subprograms each of which implements a particular random or directed search strategy. The problem to be optimized, and any desired constraints are specified by the user as a separate subprogram. The user has the option of selecting an individual optimization strategy or using a sequence of strategies to attack a given problem. Results of the application of the software package to typical problems are given.

September 1968

TABLE OF CONTENTS

I. INTRODUCTION

This is one of a series of reports concerning the use of digital computational techniques in the analysis and synthesis of DLA (distributed-lumped-active) networks. This class of networks consists of three distinct types of elements, namely distributed elements (modeled by partial differential equations), lumped elements (modeled by algebraic equations and ordinary differential equations), and active elements (modeled by algebraic equations). Such a characterization is especially applicable to the broad class of circuits referred to as linear integrated circuits, since the required fabrication techniques readily produce elements which may be referred to as "distributed", as well as producing elements which may be characterized as "lumped" and/or "active". The DLA class of networks is capable of realizing network functions with a wide range of properties. In addition, such realizations usually have fewer components and superior characteristics than realizations using only lumped elements, or realizations using lumped elements and active elements. The analysis problem for this class of networks, however, is considerably more complex than the analysis problem for more restricted classes of networks. The synthesis problem is even more challenging, and the results achieved to date have been far from general.

One of the more promising approaches to the synthesis problem appears to be the use of optimization techniques. The experience of research workers in this field has indicated that in order to successfully apply optimization techniques to a wide range of problems, it is desirable to have available a varied collection of optimization strategies. To be fully useful, the individual strategies of such a collection must be so designed that any one of them can be applied to the same problem, without requiring that the problem be modified. Thus, the individual optimization strategies can be considered as forming the elements of an optimization software package, in which various logical decisions can be incorporated as an "executive monitor" to successfully apply the different strategies in such a way as to obtain the best final results.

This report describes the formulation of a general problem structure, and the development and testing of a series of optimization strategies. The individual strategies are all applicable interchangeably to any problem which can be put into the specified structure. Examples of the application of the various optimization strategies to a pair of test problems are described. The results

of applying this optimization package to the synthesis of networks containing distributed, lumped, and active elements will be covered in a subsequent report.

II. THE GENERAL OPTIMIZATION PROCEDURE

In this section of the report we present a general optimization framework suitable for the expression of a wide range of problems, and capable of implementation by a broad class of optimization strategies. The basic approach that is implemented involves the specification of the parameters that are to be varied, the values of an independent variable at which some weighted requirements are to be met, and a functional relationship which defines the problem. A scalar error criterion is provided as a measure of the success with which the requirements are met. A list of the different variables and a description of their purpose follows:

- n - The number of parameters that are to be varied.
- x_i - The parameters that are to be varied ($i = 1,2,\ldots,n$).
- n_h - The number of values of the independent variable at which some requirements are to be met.
- h_i - The values of some independent variable h at which the requirements are to be met ($i = 1,2,\ldots n_h$).
- g_i - The values of a functional relationship $g(h,\underline{X})$ which defines the problem, where $g_i = g(h_i,\underline{X})$ ($i = 1,2,\ldots,n_h$).
- r_i - The requirements which it is desired to have the g_i meet.
- w_i - The weightings that it is desired to place on the r_i.
- y - The error criteria, $y(\underline{G},\underline{W},\underline{R})$, that it is desired to minimize.

A flow chart of the basic optimization approach using these quantities is shown in Fig. 2.1.

For the example of a network whose elements are to be varied to produce specific values of a magnitude function at a certain numberof frequencies the x_i would be the values of the network elements, the h_i would be the frequencies at which some requirements are to be met, the g_i would be the values of the magnitude of the network function, the r_i would be the desired values of the magnitude at the frequencies h_i, and the error function y might be given in the form

$$y = \sum_i w_i(r_i - g_i)^2 \qquad (2.1)$$

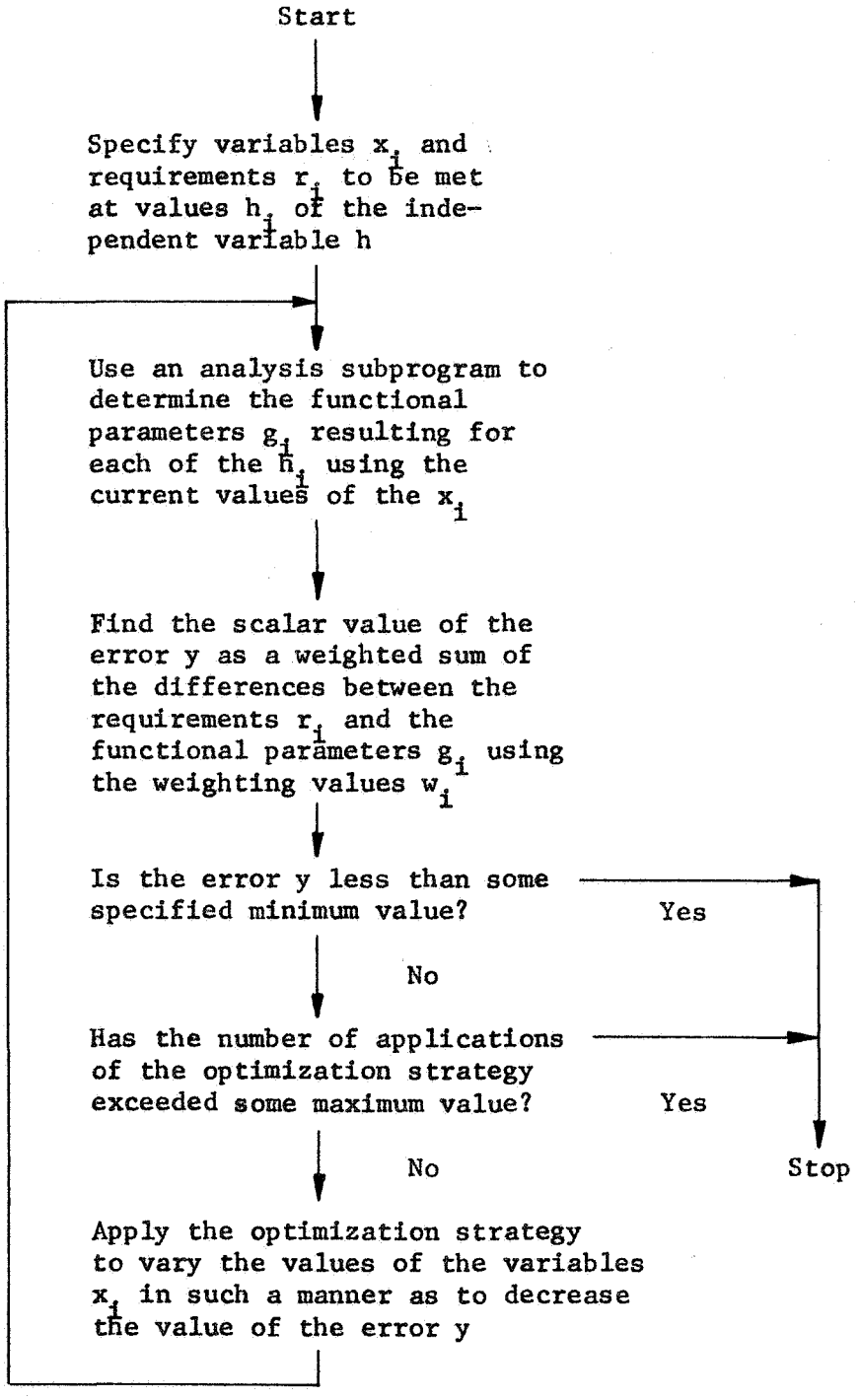


Fig. 2.1 Flow chart of general optimization procedure

This report covers the application of six optimization strategies to the general approach outlined above. These are random grid search, random direction and step size search, pattern search, steepest descent, Newton-Raphson, and Fletcher-Powell. Each of these optimization strategies has been prepared as a separate subroutine. The use of common FORTRAN variables permits any or all of these strategies to be applied to any given problem. In addition, this makes possible the specification of executive monitor schemes in which different optimization strategies are applied to different phases of the optimization of a specified problem. The method of structuring the overall approach to make this possible, and the details of the operation of the different strategies are given in detail in the following sections of this report.

III. GENERAL PROGRAM INFORMATION

The various optimization strategies described in this report have been implemented using FORTRAN as a source language. Each of the strategies has been programmed as a separate subroutine. In order to provide for efficient transfer of information between the various subprograms, and to permit as flexible usage of these component programs as possible, a common set of variable names has been used in writing all the programs. A list of these names follows:

N - The number of variables x_i.

X(I) - The current values of the variable x_i.

XL(I) - The lower bound of the variable x_i.

XU(I) - The upper bound of the variable x_i.

KX(I) - The number of values that it is desired for the variable x_i to take on between its upper and lower limit (for random grid search).

XP(I) - A set of variables which can be used to store the previous values of the variables x_i, the gradient ∇X, etc.

NH - The number of values of the independent variable h.

H(I) - The values h_i of the independent parameter h.

G(I) - The value g_i found by evaluating $g(h,\underline{X})$ for $h = h_i$.

R(I) - The desired value r_i of $g(h,\underline{X})$ evaluated for $h = h_i$.

W(I) - The weighting w_i associated with the value r_i.

YERR - The current value of the error function y.

The names of the FORTRAN variables defined above closely parallel the symbols used for the quantities introduced in connection with the description of the general optimization procedure given in Sec. II of this report. In addition to these variables, it is convenient to define some other variables which have general application to all of the optimization strategies. These are the following:

ALFA - An array used for storing alphanumeric information for problem identification.

ITER - A counter for the number of iterations completed by the optimization program.

ITMAX - The maximum number of iterations desired.

ERMIN - The minimum value desired for the error.

It is desirable to define two other arrays of FORTRAN variables. These arrays are useful for the following purposes: (1) The specification of certain parameter values that may vary between different applications of a given optimization strategy; and (2) the selection of options giving minor variations in the operation of a given optimization strategy. An example of such a variation is in the quantity of printout that is desired from an optimization subroutine regarding its successful (and unsuccessful) efforts to reduce the error YERR. For purposes of defining these parameters and options, it is desirable to assign each optimization strategy a number and a name. The strategies described in this report are tabulated below:

Subroutine Number	Subroutine Name	Subroutine Optimization Strategy
2	OPT2	Random grid search
3	OPT3	Random direction and step size search
4	OPT4	Pattern search
6	OPT6	Steepest descent
7	OPT7	Newton-Raphson
9	OPT9	Fletcher-Powell

The FORTRAN variables used to specify the values of the parameters and the options to be taken are:

PARAM(I,J) - A set of variables which can be used to store any desired parameters that need to be specified for

optimization subroutine number I. The range of J is from 1-7.

NOPT(I,J) - A set of variables which can be used to store indicators for any options that are included in optimization subroutine number I. The range of J is from 1-10.

The variables defined above are placed in labeled common storage for the entire program. Thus, no input arguments are specified for any of the optimization subroutines. To accomplish this, each subroutine includes the following statements:

```
COMMON /OPT/X(20), XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
```

To provide standardization and avoid duplication of the manner in which the values of the variables are fed into any of the optimization subroutines, a separate subroutine RDOPT has been prepared to provide for the reading of any or all of the above variables. The details of this subroutine are covered in the following section. Similarly, for uniformity, a subroutine PROPT has been provided to record the final results of the use of any of the optimization strategies. The use of these two subroutines make it possible to have an extremely simple basic main program for the use of a given optimization strategy. A flow chart for such a program is shown in Fig. 3.1. The executable statements for this main program (following the labeled common statements) have the form

```
CALL RDOPT
CALL OPTX
CALL PROPT
STOP
END
```

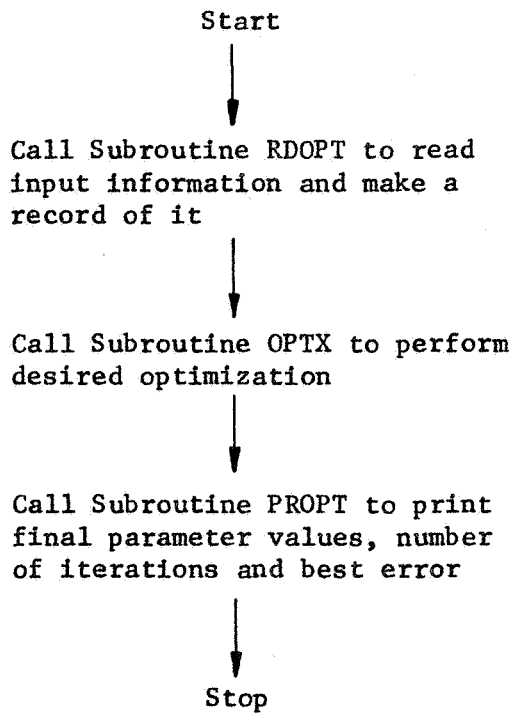


Fig. 3.1 Flow chart for a basic main program

This main program is, of course, a very simple one. Due to the general structure of the optimization software package described in this report, however, it is a simple matter to design complex optimization programs involving more than one optimization strategy. For example, suppose that it was desired to optimize a given problem using up to 200 iterations of steepest descent optimization subprogram OPT6, in addition, if the error was not below the specified value of ERMIN, to try 30 iterations of the Newton-Raphson method incorporated in subroutine OPT7. Assuming that the subroutine RDOPT was used to initially establish a value of 200 for ITMAX, the statements of the main program (following the common statements) would be

```
      CALL RDOPT
      CALL OPT6
      IF (YERR.LT.ERMIN) GO TO 1
      ITMAX=30
      CALL OPT7
    1 CALL PROPT
      STOP
      END
```

Other, more complex executive strategies are easily implemented using the basic software structure described in this report.

The functional relationship $g(h,\underline{X})$ which defines the problem must be implemented by a separate subroutine named ANLYZ. This is the subroutine which the individual optimization strategies call. It must include the labeled common statements given above. The subroutine ANLYZ will contain the necessary FORTRAN statements to generate the NH values of the variables G(I) that result from the use of the NH variables H(I), all evaluations being made for a given set of N values of the variables X(I). Some examples of this subroutine will be found in connection with the discussion of the test problems given in Sec. V.

The error criteria $y(\underline{G},\underline{W},\underline{R})$ is specified by a separate subroutine named ERR which may be set up so as to implement any desired type of error criteria. In addition, this subroutine can be used to incorporate various constraining relations that may be needed for a given problem. The tests of the various optimization strategies documented in this report all used an error criteria of the form given in (2.1). A listing of the subroutine ERR for this relation is given in the Appendix.

IV. INPUT AND OUTPUT OF DATA

A subroutine RDOPT has been prepared to provide for reading any or all of the variables defined in the last section. When this subroutine is called, it begins to read data cards. The first card is read in alphanumeric format and is reproduced as the heading for all output data. It may contain any desired information such as the name of the person submitting the program, the date, the purpose of the run, the type of problem being attacked, etc. If it is not desired to reproduce any such information, then a blank card must be used as the first data card. The second data card is used to specify the number of variable parameters N, the number of values NH of the independent variable h, and the values of ITMAX and ERMIN. The format for this information is (2I2,I3,3X, E10.0). Following this information, values of any of the variables listed in Sec. III may be entered. This is done by placing a code number in format (I2) on a card, and following the code number card by a card or cards containing the input information prepared according to the necessary format. A listing of the code numbers, the variables read, and the input format to be used follows:

Code Number	Variables	Input Format	Remarks
1	X(I)	8E10.0	N of these variables are read.
2	XL(I)	8E10.0	N of these variables are read.
3	XU(I)	8E10.0	N of these variables are read.
4	KX(I)	40I2	N of these variables are read.
5	I,PARAM(I,J)	I2,8X, 7E10.0	A maximum of 7 parameters are read. The index I refers to the number of the optimization subroutine.
6	I,NOPT(I,J)	I2,8X, 10I1	A maximum of 10 option indicators are read. The index I refers to the number of the optimization subroutine.
7	H(I)	8E10.0	NH of these variables are read.
8	R(I)	8E10.0	NH of these variables are read.
9	W(I)	8E10.0	NY of these variables are read.

The subroutine RDOPT uses a DATA statement to initialize the arrays PARAM and NOPT to zero and the array W to unity, thus, no input need be made for these arrays unless specific input values are desired. In addition, each of the optimization subroutines tests all of its parameter values at the time it is first called. Any which are zero, i.e., which have not been specifically read in by RDOPT, are initialized to convenient values by the subroutine before proceeding with the execution of the optimization strategy. Thus, in many of the uses of these subroutines, it will not be necessary to provide data cards for code number 5. The code number cards and the associated data cards listed above may be placed in the data card deck in any order, and they need to be used only for the variables for which some input values are to be read. The last data card should be followed by a blank card. Thus, the input card deck will have the following form:

Alphanumeric Identification Card (8A10)
Data Card for N, NH, ITMAX, ERMIN (2I2,I3,3X,E10.0)
This set of cards is repeated as often as necessary:
- Code Number Card (I2)
- First input data card for specified code number
- .
- .
- .
- Last input data card for specified code number

Blank Card

The subroutine RDOPT may also be called additional times in the program to read

values of data. On such subsequent calls it will not normally read in a new alphanumeric identification card or a new data card for N, NH, ITMAX, and ERMIN, i.e., the first data card to be read on such a subsequent call should be a code number card. If it is desired to read a new alphanumeric identification card and a new card for N, NH, ITMAX, and ERMIN, then the call for RDOPT should be preceded by the statement N = 0. The subroutine RDOPT provides a printout of all the data which has been read by it before returning control to the main program.

The major source of output data provided during the execution of an optimization strategy is the optimization subroutine itself. The details of the format used for this vary with the individual subroutine, and will be given in more detail in the discussions of these subroutines. To provide for uniformity in presenting the final results of the optimization process, however, a subroutine PROPT is provided which prints the final values of ITER and YERR, and the final values of the variables X(I). In addition, using these values of X(I), PROPT calls the subroutine ANLYZ, and prints the resulting values of the variables G(I) for the functional relationship defining the problem.

A listing of the subroutines RDOPT and PROPT may be found in the Appendix at the end of this report.

V. TEST PROBLEMS

In order to provide a realistic evaluation of the efficiency of the various optimization strategies in reducing a specified error criteria, two test problems were prepared. These problems are described in this section. The results of applying the various optimization strategies to these test problems is treated in the sections describing the individual optimization strategies.

Test Problem 1

This problem is designed to test the use of optimization strategies to synthesize a low-pass network with a specified topology. The network configuration has the form shown in Fig. 5.1.

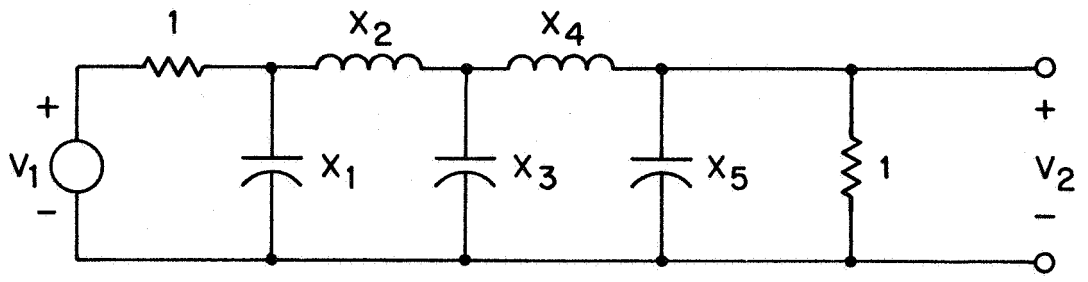


Fig. 5.1 Network for test problem 1

The values of the capacitors are specified in farads and the values of the inductors are specified in henries. The voltage transfer function for this network has the form

$$\frac{V_2}{V_1}(p) = \frac{1}{\begin{array}{l} p^5(x_1x_2x_3x_4x_5) + p^4(x_1x_2x_3x_4 + x_2x_3x_4x_5) \\ \hline + p^3(x_1x_2x_3 + x_1x_2x_5 + x_1x_4x_5 + x_3x_4x_5 + x_2x_4x_4) \\ \hline + p^2(x_1x_2 + x_1x_4 + x_3x_4 + x_2x_3 + x_2x_5 + x_4x_5) \\ \hline + p(x_1 + x_2 + x_3 + x_4 + x_5) + 2 \end{array}} \qquad (5.1)$$

It is desired to match the following specifications on $V_2/V_1(j\omega)$:

Point No.	Frequency (Hz)	$\lvert V_2/V_1(j\omega)\rvert$ (db)
1	1.00E-01	-6.4825E 00
2	2.00E-01	-6.2554E 00
3	5.00E-01	-4.7086E 01
4	1.00E 00	-7.8108E 01
5	2.00E 00	-1.0841E 02
6	5.00E 00	-1.4826E 02
7	1.00E 01	-1.7837E 02

Thus, the problem is simply to find the values of the elements x_i ($i = 1,2,\ldots,5$) so that the low-pass characteristic specified by the data in the table given above

is realized as closely as possible. The various quantities defined in Sec. II may be identified as follows: n equals 5; n_h equals 7; x_1, x_3, and x_5 are the values of the capacitors; x_2 and x_4 are the values of the inductors; the h_i ($i = 1,2,\ldots,7$) are the values of frequency (in Hz); the r_i ($i = 1,2,\ldots7$) are the specified values of the magnitude of the transfer function (in db); and the w_i ($i = 1,2,\ldots,7$) are the unity-valued weighting factors. The functional relationship $g(h,\underline{X})$ is given by (5.1) with the substitution $p = j2\pi h$. The data tabulated above corresponds with the following values for the parameters:

$x_1 = .7$ farad $\qquad x_2 = 1.6$ henries
$x_3 = .9$ farad $\qquad x_4 = 1.4$ henries
$x_5 = .6$ farad

A flow chart and a listing for the subroutine ANLYZ which computes $g(h,\underline{X})$ is given in the Appendix of this report.

Test Problem 2

This problem is designed to test the use of optimization strategies to solve an approximation problem. It is desired to find the coefficients x_i ($i = 1,2,\ldots,5$) in the network function

$$N(p) = \frac{x_5 p^2}{(p^2 + x_1 p + x_2)(p^2 + x_3 p + x_4)} \qquad (5.2)$$

in such a manner that the following magnitude and phase specifications are met:

Point No.	Frequency (rad/sec)	Magnitude	Phase (deg)
1	0.8	5.0389	
2	0.9	20.9585	
3	1.0	50.0000	
4	1.1	23.6463	
5	1.2	7.2198	
6	0.8		153.03
7	0.9		117.75
8	1.0		0.00
9	1.1		-115.46
10	1.2		-148.03

The various quantities defined in Sec. II may be identified as follows: n equals 5;

n_h equals 10; x_1 and x_2 determine one pair of pole locations; x_3 and x_4 determine a second pair of pole locations; x_5 is an overall magnitude constant: the h_i ($i = 1,2,\ldots,10$) are the values of frequency (note that the first five values and the last five values are the same); the r_i ($i = 1,2,\ldots,5$) are the desired values for the magnitude of $N(j\omega)$; the r_i ($i = 6,7,\ldots,10$) are the desired values of the phase of $N(j\omega)$; and the w_i ($i = 1,2,\ldots,10$) are set to unity. The functional relationship $g(h,\underline{X})$ is given by (5.2) with the substitution $p = jh$. The data tabulated above corresponds with the following values for the parameters:

$$x_1 = 0.1 \qquad x_4 = 0.9$$
$$x_2 = 1.1 \qquad x_5 = 1.0$$
$$x_3 = 0.1$$

A flow chart and a listing for the subroutine ANLYZ which computes $g(h,\underline{X})$ for this problem is given in the Appendix of this report.

VI. OPTIMIZATION STRATEGIES

In this portion of the report a description is given of the various optimization strategies. Each of these descriptions is subdivided into the following classifications:

1. Theory. In this section, the basic theory of the particular optimization strategy is described, and the algorithm which is used to implement the method is defined.

2. Discussion of the Program. In this section, the details of the program which implements the particular optimization strategy are discussed, with reference to the theory developed in section 1. A description of the different parameters and options provided in the program are given.

3. Results. In this section, the results obtained from applying the particular optimization strategy to the two test problems defined in Sec. V are presented. These results should be interpreted as typical in the sense that they represent initial efforts to apply the optimization strategy to the test problems and illustrate the use of different program options and the effect of different initial starting points. In general, no attempt has been made to exhaustively attack a given problem using any one optimization strategy. Thus, the results described, in many cases, could be improved by additional optimization runs using information gained from earlier ones to effectively direct them. A listing of

the pertinent values of variables, parameters, and options is given in connection with the discussion presented in each of the strategies.

A summary of the results obtained from all the optimization strategies, and some conclusions with respect to the particular utility of each of them is given in Sec. VII. Flow charts and listings for the different subroutines may be found in the Appendix of this report.

A. RANDOM GRID SEARCH SUBROUTINE OPT2

Theory

A random grid search is made by first defining a set of upper and lower bounds for the values of the variables. This in effect defines an n-dimensional volume which is to be searched. There are then two alternatives which may be used: (1) select a certain finite number of possible values spaced throughout the defined interval of each variable and randomly select the actual value from among these possible choices; or (2) select the value of each variable randomly from the continuous range of possible values in the defined interval. The first alternative is especially useful when the physical nature of the variable is such as to permit only a certain number of values. As a simple example, an (ideal) diode may be treated as a resistor with a value of zero or infinity. Thus, if the resistance of the diode is one of the variables of the problem, it is helpful to constrain it to having either of two values, namely, a very low value or a very high one, the selection between the two being made randomly. The second alternative is useful when a continuous range of values of the physical variable are available. A simple example of such a variable is the resistance of a potentiometer. When upper and lower bounds for the variables have been determined, and one of the alternatives defined above has been selected, then a random grid search is achieved by a cycle of operations in which a set of values for the variables are randomly chosen, and the scalar error criteria is calculated for the values of these variables. The error criteria is then compared with the lowest value found previously. If it is less, then the current values of the variables are retained as a new "best" point. Otherwise they are discarded and the cycle is repeated. If desired, after a predetermined number of cycles, the upper and lower limits of the variables may be redefined to more closely surround the best point which has been found, and a detailed random search made of this smaller n-dimensional volume.

The random grid search optimization subroutine requires only a single

evaluation of the functional relationship $g(h,\underline{X})$ per cycle of operation. This gives it a large advantage over strategies which require the computation of either the gradient or the Jacobian, since such computations are usually made by perturbation techniques, thus requiring respectively $n + 1$ and $n^2 + 1$ evaluations of the functional relationship $g(h,\underline{X})$ per cycle. In addition, the simultaneous nature of the random grid search strategy means that it is unaffected by difficult terrain features in the n-dimensional space in which the error criteria is defined. Thus, despite its relatively unsophisticated basic nature, random grid search frequently provides valuable information about problems in which little information is available concerning what constitutes reasonable initial values for the variables.

A major disadvantage of the random search strategy lies in the fact that, for large dimensional problems, the n-dimensional volume which must be searched is extremely large. For example, consider a unit segment of line, which is search into a final interval of uncertainty of 10%, i.e., a length of 0.1 units. Now consider a unit square. 10% of its area could be considered as a smaller square $(0.1)^{1/2}$ on a side (it could be considered in many other ways also). This says that each of the two variables could be anywhere within a 31.6% interval of their total range even though only 10% of the given area is being considered. Continuing this logic, we see that if a function of 50 variables is specified, a 10% volume of the space would encompass $(0.1)^{1/50}$ range of each of the variable values, about 93%! Thus we see that cutting down our multidimensional volume to a small percentage can still mean that great ranges of individual variables remain to be searched.

The probablistic aspects of random search as applied to the multivariable situation are interesting. Consider a case with three variables, each with a normalized range of 1.0. The hyperplane of experiments for this case is a cube. If the range of each variable is divided into ten intervals of (0.1) units in length then the cube may be considered as being divided into 1000 smaller cubes or subcubes. More general we may say that if there are m divisions of each variable and n variables; then there will be $(m)^n$ subcubes in the n-dimensional space. Now consider the function y which is to be minimized. Assign to each subcube the average value that y has in the subcube. Suppose that we wish to find any 1 of the best 100 subcubes out of the thousand. By best, we mean the subcubes with the minimum value of y. If the subcubes are chosen at random, the probability of each choice being in the best 100 is 100/1000, i.e., 0.1. The probability of

not being in the best 100 is 1 - 0.1 = 0.9. For two choices, the probability of both subcubes not being in the best 100 is $(0.9)^2$ = .81. Thus, the probability of finding one good subcube in 2 tries is 1 - .81 = .19. In general, the probability p(0.1), i.e., the probability of finding a cell in the best 10% is 1 - $(0.9)^n$, where n is the number of tries. More generally,

$$p(f) = 1 - (1 - f)^n \qquad (6A.1)$$

where f is the fraction of the total possibility. For example, for a value of f of 0.1, 16 trials give a probability of 0.8, 44 trials give a probability of .99, etc. A table giving some other cases is given below[1]

Table for values of n

		p(f)			
		.8	.9	.95	.99
f	.1	16	22	29	44
	.05	32	45	59	90
	.025	64	91	119	182
	.01	161	230	299	459

Note that the probability does not depend on the number of dimensions.

Discussion of the Program

The digital computer program used to implement the random grid search strategy described above consists of two subroutines. The first of these, OPT2, initializes the parameters, keeps track of successful and unsuccessful steps, provides for printout, and tests for the maximum number of iterations and the minimum value of the error. The second subroutine, RAND, provides for the generation of the randomly chosen values of the variables X(I). The subroutine OPT2 has several options designed to permit greater flexibility in its use. First of all, it provides for the use of different initializations for the random number generator of the CDC 6400 computer for which the program was written. Secondly, as described above, it provides for the option of constraining the values of the variables to a certain number of possible values over the given range, or the use of a continuous range of values. Finally, it has an option

[1] Samuel H. Brooks, "A Discussion of Random Methods for Seeking Maxima", Operations Research, Vol. 6, pp. 244-251, March 1958.

permitting a search to be made of a reduced volume of the n-dimensional space in the vicinity of the best point previously found. An option by means of which the subroutine will run for a number of iterations sufficient to provide a specified probability that an option point is located within a certain volume of the n-dimensional space using the relation defined in (6A.1) is also provided. The options and their defining parameters are given below:

NOPT (2,1) - If this option indicator is set to 1, the maximum iterations as determined by ITMAX will be computed by the subroutine using the probability formula given in (6A.1), and using values specified by PARAM (2,1) and PARAM (2,2).

NOPT (2,2) - If this option indicator is set to 1., a local random search will be performed around the best point found from the originial random search. PARAM (2,3) defines the area to be searched in this case. The local search will be initiated either when the maximum iterations specified by ITMAX have been made, or when the error is less than that specified by ERMIN.

NOPT (2,3) - If this option indicator is set to 1, the results of every iteration will be printed out. If it is set to 0, only the results of those iterations which yield an improvement in the error will be printed.

NOPT (2,4) - If this option indicator is set to 1, the values of each variable X(I) will be chosen randomly between the limits XL(I) and XU(I). If it is set to 0, the values of the variables X(I) will be constrained to KX(I) possible values equally spaced between XL(I) and XU(I), and randomly selected.

NOPT (2,5) - This option determines the initialization point for the random number generator in the CDC 6400 computer. If it is not read in, or if it is read in as 0, an initialization of 1 is selected for the random number generator. This option indicator may be set to any other integer value to obtain other initializations of the random number generator.

PARAM (2,1) - This constant specifies the fraction (with the lowest error) of the total volume in which it is desired that the best point found by the random search be located. It is used only if NOPT (2,1) is set to 1. The parameter is initialized to 0.05 if not read in by RDOPT.

PARAM (2,2) - This parameter determines the desired probability that the best point found by the random search be within the fraction of the total volume defined by PARAM (2,1). It is used only if NOPT (2,1) is set to 1. The parameter is initialized to .95 if a value is not provided by RDOPT.

PARAM (2,3) - This parameter determines the reduced volume to be searched if the fine search option is selected by setting NOPT (2,2) to 1. It does this by redefining the values of XL(I) by the statement

```
XL(I) = XP(I) - PARAM(2,3)*(XU(I) - XL(I))
```

where XP(I) is the best value found for X(I) in the original search. Similarly, the value of XU(I) is redefined by the statement

```
XU(I) = XP(I) + PARAM(2,3)*(XU(I) - XL(I))
```

where the value of XL(I) is the original one, not the redefined one. It should be noted that if any of the XP(I) are close to the original values of the corresponding XL(I) or XU(I), the new volume to be searched may include an area outside the area defined by the original XL(I) and XU(I). This parameter is initialized to 0.2 if a value for it is not previously established by the main program or by RDOPT.

PARAM (2,4) - This parameter specifies the ratio between the minimum error used in the original search and the minimum error specified for the local search. It defines a minimum error ERMN 2 using the statement

```
ERMN2 = ERMIN*PARAM(2,4)
```

It is initialized to 0.1 if a value is not read in.

PARAM (2,5) - This parameter establishes an arbitrary initial value for the error, against which the error realized by the initial iterations of the program can be tested to determine whether or not they have made an improvement. It is initialized to 1.E+20 by the program, unless some other value is read in.

PARAM (2,6) - This parameter specifies the new value of the maximum number of iterations permitted for the local search. It defines a new maximum iteration constant ITMX2 using the statement

```
ITMX2 = ITMAX*PARAM(2,6)
```

It is initialized to 1.0 unless some other value is read in.

A flow chart and a listing for the subroutine OPT2 and RAND is given in the Appendix.

Results

Tests were made to determine the effective ness of the program in solving test problems 1 and 2. The data for these tests is itemized in Table 6A. Some comments on the different runs follow:

Run 1 - This and the following run were made on problem 1. The run illustrates the effects of convergence from a poorly chosen starting point with an initial error of 5169. After 80 iterations the error was reduced to 15.62. No further improvement was made in iterations 81-100.

Run 2 - In this run, the points have been chosen randomly rather than being constrained to 30 values of each parameter as was done in run 1. Very little difference was noted in the points found for the first 100 iterations. The error at iteration 80 was 14.98, compared with an error of 15.62 at this point in run 1. Only two additional improvements were made in the second 100 iterations, and these reduced the error only slightly to 9.243. A local search was next made around the "best" point having the values

X(1) = 1.039	X(4) = 1.109
X(2) = 1.178	X(5) = .5304
X(3) = 1.244	

In 200 iterations, only 6 improvements were found, but these reduced the error to 0.5969.

Run 3 - This and the following run were made on problem 3. This run illustrates the vastness of hyperspace, and the problems of matching many requirement points. The initial error was greater than 10^5 and was not recorded. The starting point was poorly chosen, and after 400 iterations the error was still 2403.

Run 4 - A better starting point was chosen for this run. Even so, the initial error was 125,484. After 400 iterations, the error had been reduced to 163. If further reduction of the error was required, additional runs could be made using the final point of this run as a starting point, and further reducing the volume to be searched.

TABLE 6A - TEST RESULTS FOR OPT2

Run No.	1	2	3	4
Problem	1	1	2	2
X(1)	.5	.5	1.	.05
X(2)	.5	.5	1.	1.0
X(3)	.5	.5	1.	.05
X(4)	.5	.5	1.	1.0
X(5)	.5	.5	1.	1.0
XL(1)	.01	.01	.01	0.
XL(2)	.01	.01	.01	.8
XL(3)	.01	.01	.01	0.
XL(4)	.01	.01	.01	.8
XL(5)	.01	.01	.01	.8
XU(1)	1.5	1.5	1.5	.2
XU(2)	1.5	1.5	1.5	1.2
XU(3)	1.5	1.5	1.5	.2
XU(4)	1.5	1.5	1.5	1.2
XU(5)	1.5	1.5	1.5	1.2
KX(1)	30	30	0	0
KX(2)	30	30	0	0
KX(3)	30	30	0	0
KX(4)	30	30	0	0
KX(5)	30	30	0	0
PARAM(2,1)	.05	.05	.05	.05
PARAM(2,2)	.95	.95	.95	.95
PARAM(2,3)	.2	.2	.2	.2
PARAM(2,4)	1	1	1	1
PARAM(2,5)	10000.	10000.	10000.	1.E+20
PARAM(2,6)	1.	1.	1.	1.
NOPT(2,1)	0	0	0	0
NOPT(2,2)	0	1	1	1
NOPT(2,3)	0	0	0	0
NOPT(2,4)	0	1	1	1
NOPT(2,5)	0	0	0	0
ITMAX	100	200	200	200
ERMIN	.001	.001	.001	.001
Original Error	5169	5169		125484
Final Error	15.62	.5970	2403	163
Iterations	100	400	400	400
X(1)	.7293	1.189	.0540	.0896
X(2)	.8834	1.158	1.017	1.081
X(3)	1.294	1.137	.1723	.1317
X(4)	1.140	1.313	.9051	.9021
X(5)	.9862	.4203	.1827	1.152
CP time	3.272	4.416	4.202	3.829

B. RANDOM DIRECTION AND STEP SIZE SEARCH SUBROUTINE OPT3

Theory

The random direction search optimization strategy is characterized by an extremely simple basic philosophy. To initiate such a search, an initial set of values for the variables is selected, and the value of the error criterion is computed for this point in the n-dimensional space. All the variables are then simultaneously changed using independent positive and negative random increments obtained from a random number source, and the error criterion is again evaluated at the new point. If the new value of the error is less than the original value, the new point is used as the basis for future random moves. If an improvement has not been made in the error, then the original point is used as the base for another random move. Although the philosophy of this procedure is extremely simple, this type of an optimization strategy has the advantage that it is able to follow quite difficult terrain features with excellent results. Thus, it is useful in many optimization problems where other search strategies fail. Another advantage is that this type of search does not require the computation of gradient information, which frequently must be obtained by a perturbation technique. Thus, the number of evaluations of the functional relationship defining the problem is considerably reduced. As a result, the random direction and step size search optimization strategy can be permitted to operate for relatively large numbers of cycles without using up large quantities of computing time.

A modification of the basic strategy which has been found to be of value in many problems is to reduce the magnitude of the random variations after a predetermined number of unsuccessful moves have been attempted. In effect, this initiates a "local" search in the vicinity of the best point previously found. Such a step is frequently of assistance in more accurately locating sharply defined minima.

Discussion of the Program

The digital computer program used to implement the random direction and step size search described above consists of two subroutines. The first of these, OPT3, initializes the parameters, counts successful and unsuccessful steps, provides for printout of data, and tests for maximum iterations and minimum error. The second subroutine, RANS, computes the random changes in the variables X(I).

The subroutine OPT3 has several options designed to permit greater flexibility in its use. First of all, it has provision for the use of a set of externally

provided random numbers, for cases in which it is not desired to use the random number source provided by the computer on which the program is being run. Second, for the case where the program is being run on a CDC 6400 computer, an option is provided to permit the use of different initializations for the random number generator of this computer. Finally, parameter choices are available to specify the details of the manner in which a local search is made. The options and the related parameters are described as follows:

NOPT(3,1) - If this option indicator is set to 1, the random numbers used in determining the random changes in the variables are read in as data when the subroutine OPT3 is first called. In this case 572 random numbers must be supplied in FORMAT (16F6.3). These numbers should have a normal distribution with zero mean and variance equal to one. They should have a maximum magnitude of 2.7 in both the positive and the negative directions. If this indicator is not set to 1, the random numbers are taken from the computer random number generator. In this case it is assumed that the numbers are uniformly distributed over a range of 0 - 1.

NOPT(3,3) - This option determines the initialization for the random number generator of the CDC 6400 computer. If it is not read in, or if it is read in as zero, an initialization of 1 is selected for the random number generator. This indicator may be set to any other integer value to obtain other initializations for the random number generator.

PARAM(3,1) - This parameter controls the maximum size of the steps which are permitted in the initial phase of the random search, i.e., before the program has switched to the local mode. The actual variation in the variable X(I) is determined by adding the quantity DELTX(I) to X(I). The quantity DELTX(I) is determined by the relation

$$DELTX(I) = PARAM(3,1)*(XU(I) - XL(I))\ ARN/50.$$

where the quantity ARN takes on random values lying between -50 and 50. The parameter is initialized to 0.25 unless some other value is provided by the user.

PARAM(3,2) - This parameter controls the maximum size of the steps which are permitted in the second phase of the random search, after the program has switched to the local mode. The expression for determining DELTX(I) is similar to that used for determining the previous parameter, namely,

DELTX(I) = PARAM(3,2)*(XU(I) - XL(I)) ARN/50.

This parameter is initialized to 0.05 unless some other value is provided by the user.

PARAM(3,3) - This parameter specifies the number of unsuccessful steps which must be attempted in the initial mode of the search strategy before the program changes to the local mode. It should be noted that the variable NFAIL used in the program to count the number of unsuccessful steps is initialized to 0 after each successful step. Thus, the program will change to its local mode only after a number of successive unsuccessful steps equal to PARAM(3,3) have been made. This parameter is initialized to 40.0 unless some other value is specified by the user.

A listing and a flow chart for subroutines OPT3 and RANS is given in the Appendix of this report.

Results

Tests were made to determine the effectiveness of this optimization strategy in solving test problems 1 and 2. Some comments on some of the trial runs which were made are given below. The numerical data for these runs is given in Table 6B. It should be noted that in some of the early runs PARAM(3,2) was made larger than PARAM(3,1). This in effect changed the mode of the program from its initial search mode to a larger "global" mode. In general, this approach was not as successful as changing to a smaller "local" mode.

Run 1 - This run was a broad search mode on problem 1 using the internal computer random number generator initialized to 1. The initial error was 5173. After 45 iterations the error had been reduced to 3.523. At iteration 80 (after 35 unsuccessful iterations) the program changed to a global mode encompassing 3 times as much space. No further successes were observed through iteration 200.

Run 2 - This run was similar to the first run except for the use of an initialization of 5 for the computer random number generator. After 31 iterations the error had been reduced from 5173 to 2.559. At iteration 66 the program changed to a global mode, and no further successes were observed through iteration 200. The final point reached in this run was quite different from the one reached in run 1. This was especially notable in the values of x_1 and x_5. Thus it appears feasible to use different initializations for the random number generator to obtain different minima in the hyperspace.

Run 3 - In this run, also made on problem 1, the mode switching capability of the program was used to reduce the area to be searched, rather than to enlarge it. Values of .4 and .1 were used for the pertinent parameters. A different initial point and a different set of limiting values was used for the variables. The original error was 23.69, the mode changed at iteration 43. The final error, reached at iteration 180, was 0.2402. The final point appears to be similar to that attained for run 1. The initialization of the random number generator was the same as that used for run 1.

Run 4 - This run was made on problem 2, using a poorly chosen starting initial point with an error of 4.35E+04. At iteration 104 the error had been reduced to 3.840E+03. One hundred unsuccessful iterations followed, and the program switched to its local mode at iteration 204. Only 5 successful iterations occurred before the program exited on maximum iterations (800), however, these reduced the error to 6.352E+02. Although the total reduction in error accomplished by the 800 iterations was not great, the values of the variables were changed considerably. For example, x_2 and x_4 were changed from unity to values quite close to 0.1, while the other three variables remained in the vicinity of unity. The results of this run were used as a starting point for run 5 which is described below.

Run 5 - This run used the final point of run 4 as a starting point. The upper and lower bounds of the variables were also redefined to more closely surround the initial values. At iteration 121 the error had been reduced to 40.08. No further improvements were recorded in the next 40 iterations, so at iteration 161, the program switched to its local mode. After continuing through the specified 800 iterations, the error had been reduced to .3249, a significant reduction from the original error specified in run 4. A comparison of the resulting values for the variables x_i shows them to be remarkably close to the known minimum specified for problem 2 in Sec. V. If it was desired to reduce the error even further, these values could be used as a starting point, together with redefined upper and lower limits on the ranges of the variables. It should be noted that, although the total number of iterations used in run 4 and run 5 seems high, namely 1600 iterations, each of these iterations requires only a single evaluation of the anlysis subroutine defining the problem. Thus, the total central processor time required for the two runs was only slightly over 7 sec. Such a result compares quite favorably with the time required to make considerably fewer iterations in optimization strategies which require many function evaluations per cycle of operation. In many cases, the random type of search procedure described here will be found to be more efficient than runs requiring larger amounts of computing time.

TABLE 6B - TEST RESULTS FOR OPT3

Run No.	1	2	3	4	5
Problem	1	1	1	2	2
X(1)	.5	.5	1.	1.	.1021
X(2)	.5	.5	1.	1.	1.048
X(3)	.5	.5	1.	1.	.1754
X(4)	.5	.5	1.	1.	.9536
X(5)	.5	.5	1.	1.	1.361
XL(1)	.01	.01	.5	.01	.09
XL(2)	.01	.01	.5	.01	.9
XL(3)	.01	.01	.5	.01	.09
XL(4)	.01	.01	.5	.01	.9
XL(5)	.01	.01	.5	.01	.9
XU(1)	1.5	1.5	2.	1.5	.12
XU(2)	1.5	1.5	2.	1.5	1.2
XU(3)	1.5	1.5	2.	1.5	.19
XU(4)	1.5	1.5	2.	1.5	1.1
XU(5)	1.5	1.5	2.	1.5	1.5
PARAM(3,1)	.20	.20	.4	.4	.25
PARAM(3,2)	.6	.6	.1	.15	.05
PARAM(3,3)	35	35	40	100	40
NOPT(3,1)	0	0	0	0	0
NOPT(3,3)	0	5	0	0	0
ITMAX	200	200	200	800	800
ERMIN	.001	.001	.001	.001	.001
Oringinal error	5172	5172	23.69	4.35E+04	635
Final error	3.523	2.559	.2401	635	.3249
Iterations	200	200	200	800	800
Final values:					
X(1)	1.219	.3935	1.231	.1021	.1008
X(2)	1.440	1.201	.8269	1.048	.9001
X(3)	.9165	1.084	1.509	.1754	.0991
X(4)	1.172	1.478	1.115	.9536	1.100
X(5)	.4829	1.076	.5000	1.361	.9972
CP time	3.49	3.49	3.480	3.534	3.701
Successful iterations	13	11	5	16	30
Last successful iteration	45	31	180	736	772
Mode switched at iteration	80	66	43	204	161

C. PATTERN SEARCH OPTIMIZATION SUBROUTINE OPT4

The theory of pattern search may be explained by assuming that the search algorithm is initiated at a "base" point in n-space specified by a vector $\underline{X}_2$. From this base point a series of exploratory moves are made by perturbing each of the variables x_i in sequence. On the initial cycle of the algorithm, the first perturbation of each variable is made in a positive direction, if this does not produce a point in n-space with a lower error, then a perturbation in the negative direction is tried. If neither perturbation brings an improvement in the error, then the variable is restored to its previous value. On succeeding cycles of the algorithm, each variable is first perturbed in the direction which last brought an improvement in the error. If this is unsuccessful, then a perturbation in the opposite direction is tried. The perturbations of the x_i which have been retained from a new base point which we may call $\underline{X}_1$. The search algorithm now makes a "pattern" move to another point $\underline{X}_3$ using the relation

$$\underline{X}_3 = \underline{X}_2 + 2(\underline{X}_1 - \underline{X}_2) \qquad (6C.1)$$

A new series of exploratory moves is made from the point $\underline{X}_3$, and the process is continued. Because of the accumulation, from move to move, of variable changes which yield reductions in the error function, this algorithm develops a pattern of movement in n-space which has been found in practice to be quite successful in following contour irregularities.

If a pattern move and the following sequence of exploratory moves fails to yield an improvement in the error, then the perturbation size is reduced, and the process is restarted. User experience with this search algorithm has indicated that the computing time tends to increase in proportion to the first power of the number of variables. This is indeed attractive, since most classical optimization methods have computing times which are proportional to the cube of the number of variables.

Discussion of the Program

The logic of the digital computer program for subroutine OPT4 which implements the pattern search described above follows the basic theory quite closely. Parameters are provided to enable the user to vary the initial exploratory step size, to vary the factor used to reduce the step size, and to set a lower limit for the step size. An option is also provided to permit extended printout for diagnostic purpose. The options and the parameters are as follows:

NOPT(4,1) - If this option is set to 0, printout of the variables x_i, the error, the exploratory step size, and the number of restarts of the pattern move algorithm will be made at iterations which are multiples of PARAM(4,1). If this option is set to 1, printout occurs at every step.

PARAM(4,1) - This parameter determines the frequency of the reduced printout which occurs if NOPT(4,1) is set to 0. The parameter is initialized to 10, thus printout will normally occur on iterations numbered 10, 20, 30, etc., unless some other value of the parameter is specified by the user or unless NOPT(4,1) is set to 1.

PARAM(4,2) - This parameter determines the size of the exploratory move which is initially made for each variable. It gives the fraction of the total permitted range of each variable which is used as a perturbation. It is initialized to 0.05 unless some other value is read in by the user.

PARAM(4,3) - This parameter specifies the minimum value which is permitted for the exploratory step size. It is initialized to 0.00001 unless some other value is read in. When the step size is reduced below this value, a return is made to the main program.

PARAM(4,4) - This parameter specifies the factor which is used to reduce the step size after a set of unsuccessful exploratory moves. It is initialized to 0.5 unless some other value is read in.

PARAM(4,5) - This parameter specifies the improvement in error which is necessary for an exploratory move or a pattern move to have to be considered as successful. It is initialized to .9999 in the program. Thus, unless the new error is less than 99.99 percent of the old error, no improvement in error is considered as having taken place. Any other desired value of this parameter may, of course, be read in by the user.

Results

Run 1 - This run was made on problem 1. The initial error was 10.72. After 181 iterations the error had been reduced below the specified .001 criteria.

Run 2 - This run was also made on problem 1, however, a different initial starting point with an error of 7899 was used. After 105 iterations the error had been improved to a value of .2459, and the step size had been reduced below the 1.E-05 criteria normally used for PARAM(4,3).

Run 3 - In an effort to improve the results obtained on run 2, the run was repeated using a value of 0.7 for PARAM(4,4) rather than the value of 0.5 used in run 2.

Thus, smaller decreases were made in the exploratory step size, after a set of unsuccessful moves, than were made in run 2. The results were considerably improved. Starting from the same initial point and the same initial error (7899), the error was reduced to 9.971E-04 in 65 iterations. Thus, in addition to improving the final errror, the number of iterations required was also considerably reduced.

Run 4 - This run was made on problem 2. The original error was 43546. The algorithm proceeded very efficiently to reduce the error to 6.615E-04 in 49 iterations.

TABLE 6C - TEST RESULTS FOR OPT4

Run No.	1	2	3	4
Problem	1	1	1	2
X(1)	1.	.4	.4	1.
X(2)	1.	.4	.4	1.
X(3)	1.	.4	.4	1.
X(4)	1.	.4	.4	1.
X(5)	1.	.4	.4	1.
XL(1)	.01	.01	.01	.01
XL(2)	.01	.01	.01	.01
XL(3)	.01	.01	.01	.01
XL(4)	.01	.01	.01	.01
XL(5)	.01	.01	.01	.01
XU(1)	1.5	2.	2.	1.5
XU(2)	1.5	2.	2.	1.5
XU(3)	1.5	2.	2.	1.5
XU(4)	1.5	2.	2.	1.5
XU(5)	1.5	2.	2.	1.5
PARAM(4,1)	10	10	10	10
PARAM(4,2)	.05	.05	.05	.05
PARAM(4,3)	1.E-05	1.E-07	1.E-05	1.E-05
PARAM(4,4)	.5	.5	.7	.5
PARAM(4,5)	.9999	.9999	.9999	.9999
NOPT(4,1)	0	0	0	0
ITMAX	300	300	300	500
ERMIN	.001	.001	.001	.001
Original error	10.72	7899	7899	43546
Final error	9.849E-04	.2459	9.971E-04	6.615E-04
Iterations	181	105	65	49
Final values				
X(1)	.923	1.282	1.204	.1000
X(2)	1.317	.7843	.8931	1.0999
X(3)	.942	2.057	1.708	.1000
X(4)	1.483	.9495	1.064	.9000
X(5)	.497	.4338	.4329	.9997
Final step size	2.44E-05	9.536E-08	1.163E-04	9.765E-05
CP time	7.807	5.372	3.864	

D. STEEPEST DESCENT OPTIMIZATION SUBROUTINE OPT6

Theory

The steepest descent optimization strategy is a well-known one, and the theoretical discussion of it which is presented here will be brief. Basically, the method involves the determination of the gradient direction in n-space of the error surface determined by the error criteria $y(\underline{X})$. If the gradient vector is called $\underline{D}$ with elements d_i, then these elements are defined by the relation

$$d_i = \partial y(\underline{X})/\partial x_i \qquad (6D.1)$$

Frequently, the elements d_i are normalized so that the length in n-space of the vector $\underline{D}$ is unity. A minimization of the error function $y(\underline{X})$ may then be achieved by computing a vector $\Delta\underline{X}$ by the relation

$$\Delta\underline{X} = -\alpha\underline{D} \qquad (6D.2)$$

and where the quantity α is called the step size. If this step size is correctly chosen, then replacing $\underline{X}$ by $\underline{X} + \Delta\underline{X}$ should produce a new point in n-space which has a lower error. In the implementation of this basic approach there are many variations concerned with the proper selection of step size, the method of computing the elements of the gradient vector $\underline{D}$, and the possibilities of making one-dimensional searches in the negative gradient direction to minimize the error after each determination of the gradient.

The steepest descent optimization strategy has limited usefulness in practice, despite its attractive basic philosophy. One disadvantage is that in the vicinity of a minimum convergence tends to be painfully slow, especially when compared with techniques such as the Newton-Raphson or the Fletcher-Powell. The steepest descent method is, however, well suited to rapidly covering broad stretches of relatively flat terrain in n-space; therefore, it may be appropriately applied to cases where a good initial starting point is not known. In many such situations, the optimization strategies which are very efficient near a minimum such as the Newton-Raphson, may flounder helplessly; thus, the steepest descent method provides a useful addition to the optimizer's "bag-of-tricks".

Discussion of the Program

Several options have been provided for the user of the steepest descent optimization subroutine OPT6. Some of these have to do with the manner in

which the elements of the gradient vector are computed. A first option provides for the normalization of the elements by dividing the change in the variable by the magnitude of the variable. This feature may be useful where the magnitudes of the variables encompass a wide range. A second option permits the perturbation of the variables used to compute the gradient to be made as a fraction of the actual current value of the variable, or as a fraction of the total range permitted for the variable. The following options are also available: acceleration of the step size following successful steps, thus, in effect providing an automatic step size adjustment feature; one-dimensional search in the gradient direction until a minimum is found; different types of printout to document the progress of the strategy. The details of the various options and parameters follow:

NOPT(6,1) - This option is used to determine whether or not normalization of the elements of the gradient vector is desired. If the option is set to 0, then, if we let $k_i = \Delta y/\Delta x$, the elements of the gradient vector are found by the relation

$$d_i = \frac{k_i}{\sqrt{\Sigma k_i^2}} \qquad (6D.3)$$

Thus, the components of d_i are unnormalized, although the magnitude of the $\underline{D}$ vector in n-space is set to unity. If the option is set to 1, then the quantities k_i are determined as $\Delta y/(\Delta x/x)$. Equation (3) is again used to find the components d_i. In this case the components of $\underline{D}$ are normalized with respect to the magnitude of the variables.

NOPT(6,2) - This option determines the manner in which the quantity Δx is computed. If it is set to 0, then the statement

```
DXX = (XU(I) - XL(I)) * PARAM(6,1)
```

is used (where DXX is the variable used for Δx). Thus the perturbation is proportional to the specified range of x_i. If the option is set to 1, then the statement

```
DXX = X(I) * PARAM(6,1)
```

is used. In this case, if the magnitude of X(I) is very small, such a computation would give a perturbation which might be too small.

To avoid this, if the magnitude of x_i is less than 0.01, the perturbation variable DXX is simply set equal to PARAM(6,1).

NOPT(6,3) - This option provides for different types of printed output. If it is set to 0, only successful steps are recorded in the output. If it is set to 1, then the successful steps are printed only if the acceleration step size option is in effect, after which a printout occurs whenever ITER is a multiple of the value of PARAM(6,6). If the option is set to 2, then printout occurs at every step.

NOPT(6,4) - This option provides for acceleration of the step size to reinforce a series of successful steps. If it is set to 0, the step size will be multiplied by PARAM(6,5) after each successful iteration. If it is set to 1, no such acceleration will take place. If NOPT(6,4) is initially set to 0, then, when the first reduction in step size is made by the subroutine, NOPT(6,4) is set to 1, and an "Acceleration Stopped" message is printed.

NOPT(6,5) - If this variable is set to 1, the option attempts to improve the speed of convergence by making additional steps in the gradient direction until the error no longer decreases. If it is set to 0, no such one-dimensional search is made.

PARAM(6,1) - This parameter specifies the perturbation size used to change the variables when computing the elements of the gradient vector. It is set to 0.0001 if not read in.

PARAM(6,2) - This parameter specifies the initial size of the step made in the gradient direction. It is used to multiply the elements of the gradient vector $\underline{D}$. It is set to 0.05 if not read in.

PARAM(6,3) - This parameter specifies the minimum value permitted for the step size. If an attempt is made to reduce the step size below this value, optimization is stopped, and the subroutine returns control to the main program. The parameter is initialized to 1.0E-06 if it is not read in.

PARAM(6,4) - This parameter specifies the factor used in reducing the step size after an unsuccessful move in the negative gradient direction. It is set to 0.8 if not read in.

PARAM(6,5) - This parameter specifies the acceleration factor which is used to increase the step size when NOPT(6,4) is 0. It is initialized to

1.25 unless some other value is read in.

PARAM(6,6) - This parameter determines the iterations at which printout is made if NOPT(6,3) is set to 1. In this case, printout of the variables, the elements of the gradient vector, the step size, and the error is made at any iteration which is an even multiple of PARAM(6,6). The parameter is initialized to 20; thus, printout will normally occur at the following values of ITER: 20, 40, 60, etc. Any other desired value of this parameter may be read in.

A listing and a flow chart for the subroutine OPT6 will be found in the Appendix of this report.

Results

Tests were made to determine the effectiveness of the program in solving test problems 1 and 2. The data for these tests is itemized in Table 6D. Some comments on the different runs follows:

Run 1 - This and the following two runs were made on test problem 1. In 100 iterations, the error was reduced from 23.89 to 3.98. Another 100 iterations only reduced the error to 3.80, illustrating the slow convergence of the steepest descent method as it approaches a minimum.

Run 2 - This run was started at a point considerably further removed from a known optimum than the preceding run. The original error was 67751. After 200 iterations, however, the error had been reduced to .383, considerably below that obtained from the better initial point which was used in Run 1. This type of behavior is frequently observed in the steepest descent method, consequently, it may advantageously be used when a good initial starting point is unknown.

Run 3 - The starting point for this run was even poorer than the one used for the preceding run. The initial error was 275040. After 200 iterations the error had been reduced to .851.

Run 4 - This run was made on problem 2. The initial error was 43546. After 200 iterations, the error had been reduced to 188, but the convergence was very slow.

TABLE 6D - TEST RESULTS FOR OPT6

Run No.	1	2	3	4
Problem	1	1	1	2
X(1)	1.	10.	100.	1.
X(2)	1.	10.	100.	1.
X(3)	1.	10.	100.	1.
X(4)	1.	10.	100.	1.
X(5)	1.	10.	100.	1.
XL(1)	.01	0.	0.	.01
XL(2)	.01	0.	0.	.01
XL(3)	.01	0.	0.	.01
XL(4)	.01	0.	0.	.01
XL(5)	.01	0.	0.	.01
XU(1)	1.5	20.	200.	1.5
XU(2)	1.5	20.	200.	1.5
XU(3)	1.5	20.	200.	1.5
XU(4)	1.5	20.	200.	1.5
XU(5)	1.5	20.	200.	1.5
PARAM(6,1)	.0001	.0001	.0001	.0001
PARAM(6,2)	.05	4.	4.	.05
PARAM(6,3)	1.E-06	1.E-06	1.E-06	1.E-06
PARAM(6,4)	.8	.8	.8	.8
PARAM(6,5)	1.25	1.25	1.25	1.25
PARAM(6,6)	20.	20.	20.	20.
NOPT(6,1)	0	1	1	0
NOPT(6,2)	0	0	0	0
NOPT(6,3)	1	1	1	1
NOPT(6,4)	0	0	0	0
NOPT(6,5)	0	0	0	0
ITMAX	200	200	200	200
ERMIN	.001	.001	.001	.001
Original Error	23.89	6.77E+04	2.75E+05	4.35E+04
Final Error	3.80	.383	.851	188
Iterations	200	200	200	200
X(1)	1.03	.695	.771	.1289
X(2)	.753	1.05	1.07	.9991
X(3)	1.50	1.61	1.29	.1289
X(4)	.753	1.05	1.07	.9991
X(5)	1.03	.695	.771	.9154
CP Time				

E. NEWTON-RAPHSON OPTIMIZATION SUBROUTINE OPT7

Theory

The Newton-Raphson optimization strategy is a well-known method of minimizing a given functional relationship. First let us review the basic (scalar) concept of the Newton-Raphson approach. To do this, suppose that a function $g(x)$ is defined, and that it is desired to find the value of x which satisfies the relation

$$g(x) = 0 \tag{6E.1}$$

It is assumed that the process is started at some x^0 which does not satisfy (1). If x^1 is a point in the vicinity of x^0, we may write the Taylor expansion

$$g(x^1) = g(x^0) + g'(x^0)\ (x^1 - x^0) + \ldots \tag{6E.2}$$

where $g'(x^0)$ is the derivative $g(x)$ evaluated at $x = x^0$. If $g(x)$ is linear, only the first two terms are present in the series given in (2), and we may select x^1 so that $g(x^1) = 0$ by using the relation

$$x^1 = x^0 - g(x^0)/g'(x^0) \tag{6E.3}$$

This is the well known Newton-Raphson formula. In practice, for functions which are not linear, if the process converges, the value x^1 is a better solution to (1) than x^0 is, and the method may be continued iteratively, until the desired degree of accuracy is obtained. The convergence properties of this approach are well documented in the literature.

To apply the basic Newton-Raphson method to the problem format used in this report it is first necessary to convert the scalar formulation of the Newton-Raphson method to a matrix format. Let us now assume that we have a scalar functional relationship $g(h,\underline{X})$, where $\underline{X}$ is an n-vector with elements x_i. Let us assume that there are also specified n values h_i of the independent variable h, thus we may define an n-vector $\underline{G}(\underline{X})$ with elements g_i, where $g_i = g(h_i,\underline{X})$. To express a Taylor Series for this n-vector, we define the Jacobian matrix $\underline{J}(\underline{X})$, an n x n matrix with elements

$$j_{ik} = \partial g_i/\partial x_k \tag{6E.4}$$

Our problem is now to find a solution to the vector equation

$$\underline{G}(\underline{X}) = \underline{0} \tag{6E.5}$$

If $\underline{X}^0$ is some point in n-space which does not satisfy (5), and if $\underline{X}^1$ is some point in the vicinity of $\underline{X}^0$, then the matrix equivalent of the Taylor Series given in (2) is

$$\underline{G}(\underline{X}^1) = \underline{G}(\underline{X}^0) + \underline{J}(\underline{X}^0)\ (\underline{X}^1 - \underline{X}^0) + \ldots \qquad (6E.6)$$

If $\underline{G}(\underline{X})$ is a linear function of $\underline{X}$, only the first two terms of the series given in (6) are present. In this case we may select X^1 so that $\underline{G}(\underline{X}^1) = \underline{0}$. This requires that

$$\underline{X}^1 = \underline{X}^0 - [\underline{J}(\underline{X}^0)]^{-1}\ \underline{G}(\underline{X}^0) \qquad (6E.7)$$

Just as was true in the scalar case, if $\underline{G}(\underline{X})$ is not linear, then the iterative application of (7) provides values of $\underline{X}$ which are closer to the vector satisfying (5) within the limits of the convergence of the method. This approach is easily extended to the case where it is desired to find an n-vector $\underline{X}$ such that the quantities g_i approach some specified values r_i which may not be zero. Let us define a requirement n-vector $\underline{R}$ with elements r_i and an error vector $\underline{E}(\underline{X})$ with elements e_i. These are related by the expression

$$\underline{E}(\underline{X}) = \underline{G}(\underline{X}) - \underline{R} \qquad (6E.8)$$

Note that since $\underline{R}$ is not a function of $\underline{X}$, the Jacobian of the vector $\underline{E}(\underline{X})$ is the same as the Jacobian of the vector $\underline{G}(\underline{X})$. Now let us write a Taylor series for $\underline{E}(\underline{X})$ in terms of the points $\underline{X}^0$ and $\underline{X}^1$ in n-space. We obtain

$$E(X^1) = E(X^0) + J(X^0)\ (X^1 - X^0) + \ldots \qquad (6E.9)$$

The same logic as used previously applies, namely, if $\underline{X}^1$ is a value of $\underline{X}$ such that $\underline{E}(\underline{X}^1) = \underline{0}$, giving us zero error, then we find that

$$\underline{X}^1 = \underline{X}^0 - [\underline{J}(\underline{X}^0)]^{-1}\ \underline{E}(\underline{X}^0) \qquad (6E.10)$$

We may now define a change vector $\underline{A}$ by the relation

$$\underline{X}^1 = \underline{X}^0 + \underline{A} \qquad (6E.11)$$

Thus, $\underline{A}$ is the vector which must be added to $\underline{X}^0$ to decrease the magnitude of the elements e_i of the error vector $\underline{E}(\underline{X})$. From (10) we see that

$$\underline{A} = -[\underline{J}(\underline{X}^0)]^{-1}\ \underline{E}(\underline{X}) \qquad (6E.12)$$

Thus, the elements of $\underline{A}$ are found by solving a set of simultaneous equations.

As was true in the scalar case, if $\underline{E}(\underline{X})$ is not a linear function of $\underline{X}$, the approach defined above may be iteratively applied to determine the elements of the vector $\underline{X}$ which produce values of the functional relationship $g(h_i,\underline{X})$ which approach the specified requirements r_i within the limits of the convergence properties of the method. It should be noted that if perturbation techniques are used to determine the Jacobian matrix, then n^2 evaluations of the functional relationship are required for each iteration. In addition, each iteration requires the solution of a set of simultaneous equations of order n. Thus, the Newton-Raphson optimization strategy requires considerable computing effort per cycle of operations. In the region of a minimum, however, the convergence of this method is very rapid, and the strategy is very useful in such cases.

To apply the Newton-Raphson method as described above it is necessary that the number of requirements be equal to the number of variables in the problem. The basic method is easily modified to take account of the case where the number of requirements is greater than the number of variables in the problem. To see this, we may first define a scalar error criteria $y(\underline{X})$ by the relation

$$y(\underline{X}) = \sum_{i=1}^{n_h} w_i \, (e_i)^2 \qquad (6E.13)$$

where the w_i are the weightings given to the various errors e_i. If we assume that the functional relationship $g(h,\underline{X})$ is linear in $\underline{X}$, then a change in n-space from the point $\underline{X}$ to the point $\underline{X} + \underline{A}$ will produce a change in the functional relationship which may be expressed as

$$g(h_i,\underline{X} + \underline{A}) - g(h_i,\underline{X}) = \sum_{k=1}^{n} a_k \, j_{ik} \qquad (6E.14)$$

where the quantities a_i are the elements of the n-vector $\underline{A}$, and the quantities j_{ik} are the elements of the n x n Jacobian matrix $\underline{J}$. We may use (14) to determine an expression for the new error $y(\underline{X}+\underline{A})$ which results at the new point in n-space. Thus we obtain

$$y(\underline{X}+\underline{A}) = \sum_{i=1}^{n_h} w_i \, \left(\sum_{i=1}^{n} a_k j_{ik} + e_i \right)^2 \qquad (6E.15)$$

This new error may be minimized by setting the partial derivatives of $y(\underline{X}+\underline{A})$ with respect to the a_j equal to zero. Thus we may write a set of n equations defined as

$$\frac{\partial y(\underline{X}+\underline{A})}{\partial a_j} = 2 \sum_{i=1}^{n_h} w_i j_{ij} \left(\sum_{k=1}^{n} a_k j_{ik} + e_i \right) = 0 \quad (j=1,2,\ldots,n) \qquad (6E.16)$$

The unknown in these n equations are the elements a_k. Thus, the equations may be solved to find the n-vector $\underline{A}$ which may be used to improve the point in n-space defined by $\underline{X}$ so as to reduce the scalar error criteria $y(\underline{X})$. The relations of (16) are more compactly expressed in matrix notation as

$$\underline{A} = -[\underline{J}(\underline{X})^t \underline{W} \underline{J}(\underline{X})]^{-1} \underline{J}(\underline{X})^t \underline{W} \underline{E}(\underline{X}) \qquad (6E.17)$$

where $\underline{W}$ is a diagonal matrix of order n_h with elements w_i. The observations with respect to the convergence of this modification of the Newton-Raphson approach are similar to those given for the basic method.

Frequently, the convergence of the Newton-Raphson method is improved by multiplying the elements of the change matrix $\underline{A}$ by a scalar α which is less than unity, before computing the new values of the elements of $\underline{X}$. Thus the improved values of $\underline{X}$ are given by the expression $\underline{X} + \alpha\underline{A}$. A provision for this operation has been included in the OPT7 subroutine.

Discussion of the Program

The digital computer program OPT7 which implements the relations presented in the preceding paragraphs has two main paths. One, for the case where $n = n_h$, solves the set of simultaneous equations $\underline{J}(\underline{X})\ \underline{A} = -\underline{E}(\underline{X})$. The other, for the case where n_h is greater than n, solves the set of simultaneous equations $[\underline{J}(\underline{X})^t \underline{W} \underline{J}(\underline{X})]\ \underline{A} = -\underline{J}(\underline{X})^t \underline{W} \underline{E}(\underline{X})$. A subroutine for solving simultaneous equations is included as part of the program. Because of the tendency of the Newton-Raphson algorithm to diverge for large step sizes when it is not near a minimum, a feature has been included in the program which attempts to find a path through difficult terrain by successively halving the elements of the change matrix $\alpha\underline{A}$. Three cycles of this feature are permitted before the program gives up. If success is achieved on any of these reduced step sizes, then the step size is restored to its original value before proceding with the next iteration of the program. This feature has been found to be useful in several of the trial problems.

The parameters and options provided for the user of OPT7 are:

NOPT(7,1) - If this option is set to 1, extended printout is provided at each step of the optimization process. The normal printout consists of the values of the variables x_k and the value of the error at each step. The $\underline{J}$ matrix is also printed when a return is made to the

main program from OPT7.

PARAM(7,1) - The perturbation factor used in calculating the elements of the matrix J. A value of .0001 is selected for this parameter if a value is not provided by RDOPT.

PARAM(7,2) - The factor α used in determining the size of the step to be made, i.e., the initial quantity used to multiply the elements of the matrix A. A value of 0.8 is selected unless some other value is provided from RDOPT.

A listing and a flow chart for subroutine OPT7 may be found in the Appendix of this report.

Results

Tests were made to determine the effectiveness of this optimization strategy in solving test problems 1 and 2. Some comments on the runs are given below. The numerical data for the runs is tabulated in Table 6E.

Run 1 - This and the following three runs were made on problem 1. This run started at a point quite close to a known optimum. In three iterations the error was reduced from .1256 to 1.407E-04, demonstrating the rapid convergence of the Newton-Raphson method. It should be noted that since this problem has only five variables but has seven constraints, the "non-square" feature of the Newton-Raphson program is utilized.

Run 2 - The starting point used for this run was a poorer one than that used for run 1. The initial error was 58.99. After 8 iterations, convergence stopped. The program then used its internal capacity for reducing the step size from an original value of 0.8. A value of 0.4 did not produce convergence, but a further reduction to 0.2 was satisfactory. Following this successful iteration (with step size of 0.2), the step size was restored to its original value of 0.8, and the program converged to a minimum of 6.449E-05 in a total of 10 iterations. A run made leaving the step size at 0.2 required a total of 25 iterations to reach a final error of 8.979E-04, demonstrating the importance of restoring the step size. A different final point was reached than was reached in run 1.

Run 3 - A different starting point from either of the above runs was used for this run. The initial error was 20.17. The step reduction algorithm was required on the first step to achieve convergence. After a successful iteration with a step size of 0.2, the program converged successfully with the 0.8 step size. After 9

TABLE 6E - TEST RESULTS FOR OPT7

Run No.	1	2	3	4	5	6
Problem	1	1	1	1	2	2
X(1)	.71	.8	.6	1.	.11	.05
X(2)	1.61	1.5	1.7	1.	1.15	1.0
X(3)	.89	1.0	1.0	1.	.09	.05
X(4)	1.39	1.5	1.3	1.	.91	1.0
X(5)	.61	.7	.5	1.	1.1	1.0
PARAM(7,1)	.0001	.0001	.0001	.0001	.0001	.0001
PARAM(7,2)	.8	.8	.8	.8	.8	.8
NOPT(7,1)	1	1	0	0	0	0
ITMAX	50	50	50	50	20	20
ERMIN	.001	.001	1.E-08	1.E-08	.001	.001
Original Error	.1256	58.99	19.62	23.89	523.2	12550
Final Error	1.407E-04	6.45E-05	1.748E-06	23.89	9.549E-05	10398
Iterations	3	10	23	5	6	7
X(1)	.7145	1.339	.7124	1.	.1000	-329
X(2)	1.591	1.015	1.5914	1.	1.100	2120
X(3)	.8980	1.231	.8986	1.	.1000	3957
X(4)	1.405	1.187	1.405	1.	.9000	-2334
X(5)	.5902	.4258	.5913	1.	1.000	2487

iterations the error was 1.908E-04 (less than the .001 used as a general mark of success). The rapidity of convergence in the region of a minimum is readily apparent from the next two iterations, (10 and 11) in which the error was reduced to 2.213E-06 (less than .00001). Additional iterations, however, were able to improve the error only slightly. After 23 iterations the error was 1.748E-06, a poor increase for the additional computing expended. The program stopped after a reduction of the step size to 0.1 failed to improve the error. The final point was similar to that found in run 1.

Run 4 - The starting point used for this run illustrates the complexities of hyperspace. The original error was less than that encountered in run 2, but no convergence was obtained by decreasing the initial step size of 0.8 to 0.4, 0.2, or 0.1. Other tries were made to optimize this starting point with initial step sizes of .1, .01, and .001 with no success. Changing the perturbation size to 1.E-06 also had no effect.

Run 5 - This and the following run were made on problem 3. The starting point was fairly close to a known minimum, and the initial error was 523.2. In 6 iterations the error had been reduced to 9.459E-05.

Run 6 - This run was also made on problem 3, but with a starting point somewhat further removed from the known minimum. In this case the initial error was 1.255E+05. In 3 iterations the error had been reduced to 1.040E+05. At this point the method failed to converge despite reduction of the step size to the allowed minimum of 0.1.

F. FLECTCHER-POWELL OPTIMIZATION SUBROUTINE OPT9

Theory

The Fletcher-Powell optimization strategy is one of the most useful and general methods currently available. In the vicinity of a minimum, it demonstrates a rapid convergence which is comparable to that of the Newton-Raphson approach. The Fletcher-Powell method, however, has a considerable advantage over the Newton-Raphson method in that it will also converge from initial points in n-space which are remotely located with respect to a minimum. Thus, it may be applied directly to problems in which the general area of an optimal solution point is unknown. A comparison of the computational effort required by the two methods is of interest. The major computational effort required in the Fletcher-Powell method is the calculation of the gradient of the error function. A second major computational effort is the determination of the minimum of a one-dimensional search which is made at each iteration. The combination of the gradient determination and the one-dimensional search together, however, will in general, require considerably less computational effort than that required to determine the Jacobian in the Newton-Raphson approach.

The general theory of the Fletcher-Powell method begins by assuming that the scalar error criterion $y(\underline{X})$ is quadratic, i.e., has the form

$$y(\underline{X}) = y(\underline{0}) + \underline{C}^t\underline{X} + \frac{1}{2}\underline{X}^t\underline{D}\,\underline{X} \qquad (6F.1)$$

where $\underline{X}$ is the n-vector containing as elements the values of the variables x_i, $\underline{C}$ is a column matrix of first partial derivatives, and $\underline{D}$ is a square matrix of second partial derivatives of $y(\underline{X})$ with respect to the variables x_i. The gradient $P(\underline{X})$ of the function $y(\underline{X})$ is readily found to be

$$\underline{P}(\underline{X}) = \underline{C} + \underline{D}\,\underline{X} \qquad (6F.2)$$

Solving (2) for the point in n-space at which all the elements of the gradient vector $\underline{P}(\underline{X})$ are identically zero, we obtain

$$\underline{X} = -\underline{D}^{-1}\underline{C} \qquad (6F.3)$$

Equation (3) defines the point in n-space at which a minimum is found. Thus, it is necessary that the quantities $\underline{C}$ and $\underline{D}^{-1}$ be known to find the minimum. In the Fletcher-Powell method, the matrix $\underline{D}^{-1}$ is found by an iterative process. This process consists of initializing a matrix $\underline{H}$ to the identity matrix, and then applying an algorithm (which is described below) to iteratively modify the elements of the

$\underline{H}$ matrix so that they approach those of the matrix $\underline{D}^{-1}$. Even though this method only requires computation of the gradient, it may be shown that it has "second-order" convergence, and, if there are n variables x_i, it will converge to a minimum in n cycles.

In practice, the error function is more likely to be of the form

$$y(\underline{X}) = \sum_{i=1}^{n} {}^{h} w_i \, (g_i - r_i)^2 \qquad (6F.4)$$

where the quantities g_i are the values of some functional relationship $g(h,\underline{X})$. Thus, the error criterion is usually of considerably higher degree than the assumed quadratic form given in (1). In such a case, although a minimum will not, in general, be reached in n cycles, the method has been found to yield excellent convergence for a wide range of problems and initial points.

The iterative process used in each cycle of the Fletcher-Powell optimization strategy consists of the following steps:

1. Determine the gradient $\underline{P}(\underline{X})$ at the current point defined by the vector $\underline{X}$ in n-space. This requires the determination of the elements $p_k = \partial y(\underline{X})/\partial x_k$ of the vector $\underline{P}(\underline{X})$.
2. Compute a search direction vector $\underline{S}(\underline{X})$ using the relation

$$\underline{S}(\underline{X}) = - \underline{H} \, \underline{P}(\underline{X}) \qquad (6F.5)$$

3. Make a one-dimensional search in the direction specified by $\underline{S}(\underline{X})$ starting from the current point $\underline{X}$, until a minimum in the error criterion is found. If we let α be the distance to the minimum in the $\underline{S}(\underline{X})$ direction, then we desire to find

$$\min_{\alpha} \; [y(\underline{X} + \alpha \underline{S})] \qquad \alpha > 0 \qquad (6F.6)$$

4. Replace the current values of $\underline{X}$ with the values $\underline{X} + \alpha \underline{S}$.
5. Compute the gradient $\underline{P}'(\underline{X})$ at the new value of $\underline{X}$, and define $\underline{Y}(\underline{X})$, the difference between the new gradient and the old gradient by the relation

$$\underline{Y}(\underline{X}) = \underline{P}'(\underline{X}) - \underline{P}(\underline{X}) \qquad (6F.7)$$

6. Use the following relations to determine the square matrices $\underline{A}$ and $\underline{B}$ (the functional notation has been dropped for compactness of

representation):

$$A = \frac{\alpha}{\underline{S}^t \underline{Y}} \underline{S}\,\underline{S}^t \tag{6F.8}$$

$$B = \frac{-1}{\underline{Y}^t \underline{H}\,\underline{Y}} \underline{H}\,\underline{Y}\,\underline{Y}^t\,\underline{H} \tag{6F.9}$$

7. Determine a new matrix $\underline{H}'$ by the relation

$$\underline{H}' = \underline{H} + \underline{A} + \underline{B} \tag{6F.10}$$

8. Store $\underline{P}'$ as $\underline{P}$ and $\underline{H}'$ as $\underline{H}$ and return to step 2.

The steps described above may be continued until the error has been reduced below some desired minimum value. It is assumed that the matrix $\underline{H}$ has been initialied to the identity matrix before the start of the iterative procedure. The proof of the convergence of this method may be found in the Fletcher and Powell paper and in the other papers listed in the Reference section of this report.

Discussion of the Program

The digital computer program OPT9 which implements the Fletcher-Powell optimization strategy adheres closely to the procedure outlined in the preceding discussion. It differs from the original method, presented by Fletcher and Powell in the details of the manner in which the one-dimensional search is conducted. The method implemented here consists of fitting a quadratic polynomial to the curve, and determining the minimum from this polynomial. To do this the error function $y(\underline{X})$ is evaluated at a series of points taken in a direction determined by the search vector $\underline{S}(\underline{X})$, until three successive points $\underline{X} + a\underline{S}$, $\underline{X} + b\underline{S}$, and $\underline{X} + c\underline{S}$ are found which satisfy the relations

$$a < b < c \tag{6F.11}$$

$$y(\underline{X} + c\underline{S}) \leq y(\underline{X} + b\underline{S}) \tag{6F.12}$$

If we then assume that the error criterion can be expressed in the form

$$y = a_0 + a_1\alpha + a_2\alpha^2 \tag{6F.13}$$

where α is the distance measured in the $\underline{S}(\underline{X})$ direction, then the values of the constants a_0, a_1, and a_2 may be found from the quantities a, b, and c, by first solving the set of equations

$$\begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} y(\underline{X} + a\underline{S}) \\ y(\underline{X} + b\underline{S}) \\ y(\underline{X} + c\underline{S}) \end{bmatrix} \qquad (6F.14)$$

The value of α at which the minimum occurs is then found from the constants a_1 and a_2 by the relation

$$\alpha = -a_1/2a_2 \qquad (6F.15)$$

The procedure described above readily finds the minimum of the error function when the values of a, b, and c have been determined so that (11) and (12) are satisfied. To determine these values, successive steps are made in the $\underline{S}(\underline{X})$ direction, and the error function is evaluated after each step. The procedure for making these steps is started by computing an initial step size. This is taken as the smallest of the reciprocals of the magnitude of the elements of $\underline{S}(\underline{X})$, or unity, whichever is the lesser. If this initial step size does not produce a lower error, it is assumed that it is too large, and it is halved and the error is recomputed. This procedure is continued until a step size is found which does give a lower value of the error (in which case, a quadratic curve is fitted as described above) or until the number of reductions of the initial step size exceeds some specified number. In this latter case it is assumed that the topology is too irregular to permit convergence, and the entire algorithm is restarted by resetting $\underline{H}$ to the identity matrix. This modification of the original Fletcher-Powell algorithm has been found useful in speeding the convergence of problems in which the topology of the error surface is extremely irregular. If a step made using the initial step size does produce a lower error, then the step is doubled on each succeeding evaluation, until no further decrease in the error is obtained, at which time a quadratic curve is fitted to the most recent three points. If no minimum is found after a predetermined number of step size increases, the farthest point in the one-dimensional search is treated as the optimum, and the program procedes to the steps used to iteratively improve the $\underline{H}$ matrix.

The parameters and options provided for the user of OPT9 are:

PARAM(9,1) - The perturbation factor used in computing the gradient $\underline{P}(\underline{X})$. This parameter is set to 1.E-06 if a value is not previously established in the main program.

PARAM(9,2) - The maximum number of iterations allowed in the one-dimensional

search. This parameter is set to 10 if some other value is not read in.

PARAM(9,3) - The maximum number of reset cycles in which the H matrix is reinitialized to the identity matrix. This parameter is set to 3 unless some other value is read in.

NOPT(9,1) - This option provides for the use of a separate subroutine named ANLYD to be used to compute the gradient vector. To use such a subroutine NOPT(9,1) must be set to 1. Otherwise, perturbation techniques are used to compute the gradient.

NOPT(9,2) - This option provides for three levels of printout in the program. If it is set to 0, the values of the variables x_k, the error, the optimum step size, and the number of cycles made in the one-dimensional search will be printed for each iteration. If it is set to 1, then, in addition, the details of each cycle of the one-dimensional search will be printed. Finally, if it is set to 2, the A, B, and H matrices will also be printed for each iteration. When an exit is made from the program, the elements of the H matrix are printed for reference.

A listing and a flow chart of subroutine OPT9 may be found in the Appendix of this report.

Results

Tests were made to determine the effectiveness of the Fletcher-Powell optimization strategy as realized by the subroutine OPT9 in solving test problems 1 and 2. Some comments on some of the different runs which were made follow:

Run 1 - This and the following four runs were made on problem 1. This run used a starting point known to be close to an optimum. The initial error was .1256. After three iterations the error was reduced to 8.551E-05. The number of iterations is the same as was used in run 1 made for the Newton-Raphson OPT7 subprogram; however, this method requires only five evaluations of the analysis subprogram per iteration, plus a total of 24 evaluations for the one-dimensional searches making a total of 39 evaluations. On the other hand, the Newton-Raphson method required 35 evaluations per cycle; thus, it needed a total of 105 evaluations for comparable results.

Run 2 - This run used the same starting point as was used for the Newton-Raphson OPT7 subprogram (run 2). Again, the initial error was 58.99. The final error of

7.121E-04 was reached after 8 iterations. Forty evaluations of the analysis function were required in the computation of the gradients, plus 36 more for the one-dimensional searches, making a total of 76. By comparison the Newton-Raphson method required 350.

Run 3 - Another different starting point was tried for this run. In addition, the minimum error desired was set to 1.E-08, in an effort to determine how far the Fletcher-Powell subroutine would reduce the error. The initial error was 19.62. After 21 iterations the error had been reduced to 1.813E-06. At this point, the one-dimensional search was unable to find a point with a lower error after ten reductions of the step size. The H matrix was reset to the identity matrix, and another attempt was made to reduce the error; this also failed. This run is similar to run 3 made for the Newton-Raphson OPT7 subroutine, and the results are comparable.

Run 4 - Another different starting point was tried for this run. The initial error was 7899. After 3 iterations the error had been reduced to 287.7. On the 4th iteration, no improvement in error was found in the S vector direction after 10 reductions (each of 1/2) in step size. Thus, it can be assumed that the topology of the hypersurface was such as to prevent convergence of the method. The modification of the basic Fletcher-Powell algorithm referred to in the description of the program, was called at this point to reset the H matrix to an identiy matrix, in effect restarting the problem. From this point the method converged smoothly, although on iteration 28 no minimum was found in the one-dimensional search after 10 increases (by a factor of 2) of the step size. After 36 iterations the final error was 8.513E-04.

Run 5 - This was the final run made on problem 1. A still different starting point was used than the ones used in the preceding runs. The initial error was 10.72. After 25 iterations the error had been reduced to 4.69E-04. No problems were encountered in the one-dimensional search either in failure to reduce the error, or in failure to find a minimum.

Run 6 - This run was made on problem 3. The initial error was 43550. After 28 iterations OPT9 converged to a known minimum with a final error of 2.235E-04. During the process, no reduction in error was obtained in the one-dimensional search on iteration 5 after 10 evaluations. Therefore the H matrix was reset to the identity matrix at this point. All other one-dimensional searches functioned successfully, most of them using only 2 or 3 evaluations. The largest number of evaluations required by any of the successful one-dimensional searches was 7.

TABLE 6F - TEST RESULTS FOR OPT9

Run No.	1	2	3	4	5	6
Problem	1	1	1	1	1	3
X(1)	.71	.8	.6	.4	1.	1.
X(2)	1.61	1.5	1.7	.4	1.	1.
X(3)	.89	1.0	1.0	.4	1.	1.
X(4)	1.39	1.5	1.3	.4	1.	1.
X(5)	.61	.7	.5	.4	1.	1.
PARAM(9,1)	.001	1.E-06	1.E-06	1.E-06	1.E-06	1.E-06
PARAM(9,2)	10	20	10	10	10	10
PARAM(9,3)	0	0	3	3	3	3
NOPT(9,1)	0	0	0	0	0	0
NOPT(9,2)	1	1	0	0	0	0
ITMAX	20	20	50	100	100	200
ERMIN	.001	.001	1.E-08	.001	.001	.001
Original Error	.1256	58.99	19.62	7899	10.72	43550
Final Error	8.551E-05	7.121E-04	1.812E-06	8.513E-04	4.693E-04	2.236E-04
Iterations	3	8	24	36	25	28
Reset Cycles of H matrix	0	0	3	1	0	1
Final values:						
X(1)	.6987	.7869	.6612	.7233	.5963	0.100
X(2)	1.605	1.539	1.620	1.568	1.382	1.100
X(3)	.9013	.8979	.9051	.8847	.9115	.09998
X(4)	1.396	1.424	1.382	1.435	1.604	.8999
X(5)	.5992	.5464	.6316	.5877	.702	.9999

VII. CONCLUSION

The software package described in the previous sections of this report provides a general capability for the optimal solution of many engineering problems. It should be apparent from the test results itemized in Sec. VI, that each of the optimization strategies has its own set of advantages and disadvantages which depend on several factors such as the nature of the problem being attacked, whether or not a good starting point is known, etc. Thus, each of the optimization subprograms may be considered as a tool whose use may be appropriate for a certain class of problems, or for part of the attack which is to be made on a given problem. The fact that several strategies of widely varying characteristics are included in the software package makes for generality of application. The fact that any one of the optimization subprograms can be applied interchangeably to a given problem, allows considerable flexibility in their use, as well as an opportunity to make a comparison of their relative merits.

It is not fair to generalize exhaustively on the relative merits of the various optimization strategies based on the limited number of tests documented in this report. Some general observations, however, are pertinent by way of a summary. First of all, if the user has no knowledge of a good solution point for the problem he is considering, then the use of random grid search (OPT2) may prove worthwhile as an initial effort. Another possibility would be to use a steepest descent approach (OPT6) with step size acceleration from some assumed point. If a reasonable starting point which nevertheless has a high error is known, then the Fletcher-Powell algorithm (OPT9) or the pattern search (OPT4) might be used. If it is desired to minimize computation time, then random direction and step size search (OPT3) might be used. In the vicinity of a solution, to locate the minimum as accurately as possible, the Newton-Raphson (OPT7) subroutine or the Fletcher-Powell subroutine (OPT9) might be applied. To determine other minima, the random direction and step size search (OPT3) with different starting points and different initializations for the random number generator is sometimes useful. Many other combinations of these various algorithms will suggest themselves to the reader. In general, if several problems of a given type are to be solved, using the same subroutine ANLYZ, but different requirement data, it may be worthwhile to make up a master program to repetitively apply a strategy which has been found to be useful in

previous solutions of the problem. Such a master program is readily implemented using the procedure suggested in Sec. III.

The software developed in connection with this research is written in FORTRAN IV. The programs were run on a CDC 6400 computer, but are readily adapted to any other medium size computer with a FORTRAN IV capability. The application of this software package to the synthesis of distributed-lumped-active networks will be covered in a following report.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the support given to this research project by the National Aeronautics and Space Administration under Grant NGL-03-002-136. He is especially grateful for the interest and encouragement given to the project by Dr. W. J. Kerwin, Chief, Electronics Research Branch, NASA Ames Research Center. The author also wishes to acknowledge the assistance of the graduate students who wrote early versions of some of the subroutines and who contributed to the development of the overall software package, especially Jerry G. Fossum, William J. Steinway, Donald I. Grams, Donald Robinson, and Stephen P. Johnson. The assistance of Daniel E. Nuñez in punching the programs and preparing the flow charts is also gratefully acknowledged. Finally, the author wishes to express his appreciation to Dr. R. H. Mattson, Head of the Department of Electrical Engineering of The University of Arizona, for his enthusiastic support of the project.

REFERENCES

Books

The following book is a complete reference to most of the modern techniques in optimization:

WILDE, D. J., C. S. Beightler. Foundations of Optimizaton, Prentice-Hall, Englewood Cliffs, N. J., 1967.

Another book by Wilde is also very readable:

WILDE, D. J. Optimum Seeking Methods, Prentice-Hall, Englewood Cliffs, N. J., 1964.

Two books containing collections of papers concerning optimization methods and applications are:

LAVI, A., T. P. Vogl (editors). Recent Advances in Optimization Techniques, John Wiley and Sons, New York, 1966.

BALAKRISHNAN, A. J., L. W. Neustadt (editors). Computing Methods in Optimization Problems, Academic Press, New York, 1964.

General Survey Papers

The following papers offer surveys of various optimization techniques and/or tabulations of results obtained from applying different strategies to various problems.

BOAS, A. H. Modern mathematical tools for optimization, Chemical Engineering, Dec. 10, 1962-Apr. 1, 1963.

BOX, M. J. A comparison of several current optimization methods, and the use of transformation in constrained problems, Computer Journal, vol. 9, pp. 67-77, May 1966.

BOX, M. J. A new method of constrained optimization and a comparison with other methods, Computer Journal, vol. 8, p. 42, 1965.

CARNAHAN, B. Optimization methods -- a review and some example applications, Computers in Engineering Design Education, University of Michigan, pp. 56-76, Mar. 1966.

DICKINSON, A. W. Nonlinear optimization: some procedures and examples, Proc. of the 19th National Conference, Association for Computing Machinery, pp. E 1.2-1 to E 1.2-8, 1964.

FIACCO, A. V. Histrocial survey of sequential unconstrained methods for solving constrained minimization problems, Research Analysis Corp., Technical Paper RAC-TP-267, July 1967.

FLEISCHER, P. E. Optimization techniques, Chapter 6 in Kuo, F. F., and J. F. Kaiser, Editors, System Analysis by Digital Computer, John Wiley and Sons, 1966.

SCHEERER, W. G. Optimization with computers, presented at Circuit Design by Computer Tutorial Symposium, New York University, Feb. 1967.

SPANG, H. A., III. A review of minimization techniques for nonlinear functions, SIAM Review, vol. 4, no. 4, pp. 343-365, Oct. 1962.

TEMES, G. C. Iterative optimization techniques for circuit design, Proc. of the Institute on Modern Solid State Circuit Design, Univ. of Santa Clara, pp. 97-114, 1966.

TEMES, G. C., D. A. Calahan. Computer-aided network optimizaton the state-of-the-art, Proc. of the IEEE, vol. 55, no. 11, pp. 1832-1863, Nov. 1967.

WILDE, D. J. Strategies for optimizing macrosystems, Chem. Eng. Progress, vol. 61, no. 3, pp. 86-93, Mar. 1965.

WOOD, C. F. Review of design optimization techniques, IEEE Trans. on Systems Science and Cybernetics, vol. SSC-1, no. 1, pp. 14-20, Nov. 1965.

Applications

The following papers are representative of some of the many papers which have used optimization techniques to solve engineering problems:

CALAHAN, D. A. Computer design of linear frequency selective networks, Proc. of IEEE, vol. 53, pp. 1701-1706, Nov. 1965.

FUJISAWA, T. Optimization of lowpass attenuation characteristics by a digital computer, Proc. of 6th Midwest Symp on Circuit Theory, pp. P1-P13, May 1963.

FUJISAWA, T. Theory and Procedure for optimizaton of low-pass attneuation characteristics, IEEE Trans. on Circuit Theory, vol. CT-11, pp. 449-456, Dec. 1964.

ISHIZAKI, Y., H. Wakazuki, H. Watanabe. Some considerations on min-max approximation problem for network design, Proc. of 3rd Allerton Conf. on Circuit and System Theory, Urbana, Ill., pp. 786-795, Oct. 1965.

ROHRER, R. A. Fully automated network design by digital computer: preliminary considerations, Proc. of the IEEE, vol. 55, no. 11, pp. 1929-1939, Nov. 1967.

SCHEIBE, P. O., E. A. Huber. The application of carroll's optimization technique to network synthesis, Proc. of 3rd Allerton Conf. on Circuit and System Theory, pp. 182-191, Oct. 1965.

SCHOEFFLER, J. D. A solution to the approximation and realization problems for crystal ladder filters, IEEE Internat'l Conv. Rec., pt. I, pp. 282-287, Mar. 1964.

SEMMELMAN, C. L. Experience with a steepest descent computer program for designing delay networks, IRE International Convention Record, Part 2, pp. 206-210, 1962.

SMITH, B. R., G. C. Temes. An iterative approximation procedure for automatic filter synthesis, IEEE Trans. on Circuit Theory, vol. CT-12, pp. 107-112, Mar. 1965.

STOREY, C. Application of a hill climbing method of optimization, Chemical Engineering Science, vol. 17, pp. 45-52, 1962.

TEMES, G. C., J. A. C. Bingham. Iterative chebyshev approximation technique for network synthesis, IEEE Trans. on Circuit Theory, vol. CT-14, pp. 31-37, Mar. 1967.

WAREN, A. D., L. S. Lasdon. Practical filter design using mathematical optimization, Proc. of 3rd Allerton Conf. on Circuit and System Theory, pp. 677-689, Oct. 1965.

Other References

The following list contains some of the more frequently quoted and referenced papers which have appeared in the recent literature:

BECKMAN, F. S. The solution of linear equations by the conjugate gradient method, Mathematical Methods for Digital Computers, Ralson, A., H. S. Wilf, Editors, John Wiley and Sons, New York, 1960.

BROOKS, S. H., M. R. Mickey. Optimum estimation of gradient direction in steepest ascent experiments, Biometrics, vol. 17, pp. 48-56, 1961.

BROYDEN, C. G. A class of methods for solving nonlinear simultaneous equations, Math. Comput., vol. 19, pp. 577-593, 1965.

CORBATO, F. J., V. A. Vyssotsky. Introduction and overview of the multics system, Proc. Fall Joint Computer Conference, pp. 185-196, 1965.

DAVIDON, W. C. Variable metric method for minimization, Argonne National Lab, Rept. No. ANL-5990, Rev. Phys. Math. (TID-4500, 14th ed.).

FIACCO, A. V., G. P. McCormick. Programming under nonlinear constraints by unconstrained minimization: a primal-dual method, Research Analysis Corporation, Technical Paper RAC-TP-96, Sept. 1963.

FLETCHER, R. Function minimization without evaluating derivatives -- a review, Computer Journal, vol. 8, p. 33, 1965.

FLETCHER, R., C. M. Reeves. Function minimization by conjugate gradients, Computer Journal, vol. 7, p. 149-154, July 1964.

FLETCHER, R., M. J. C. Powell, A rapidly convergent descent method for minimization, Computer Journal, vol. 6, no. 4, pp. 163-168, June 1963.

GUILFOYLE, G., I. Johnson, P. Wheatley. One-dimensional search combining golden section and cubic fit techniques, NASA Report N68-18823; Contract NAS 9-4036, Report No. 67-1; Jan. 1967.

HOOKE, R., T. A. Jeeves. Direct search solution of numerical and statistical problems, J. Assoc. Computing Machinery, vol. 8, pp. 212-229, Apr. 1961.

KELLY, J. E., Jr. The cutting plane method for solving convex programs, SIAM Journal, vol. 8, pp. 703-712, 1960.

KROLAK, P., L. Cooper. An extension of fibonaccian search to several variables, Communications of the ACM, vol. 6, pp. 639, -641, 1963.

MARQUARDT, D. W. An algorithm for least squares estimation of nonlinear parameters, SIAM Journal, vol. 11, pp. 431-441, 1963.

MARTIN, D. W., G. J. Tee. Iterative methods for linear equations with symmetric positive definite matrix, Computer Journal, vol. 4, p. 242, 1961.

McCORMICK, G. P., W. C. Mulander, III, and A. V. Fiacco. Computer program implementing the sequential unconstrained minimization technique for nonlinear programming, Research Analysis Corp., Technical Paper RAC-TP-151, Apr. 1965.

McGILL, R., P. Kenneth. Solution of variatonal problems by means of a generalized Newton-Raphson operator, AIAA Journal, vol. 2, no. 10, pp. 1761-1766, 1964.

MURATA, T. The use of adaptive constrained descents in system design, IEEE International Convention Record, vol. 12, pt. 1, pp. 296-306, Mar. 1964.

NELDER, J. A., R. Mead. A simplex method for function minimization, Computer Journal, vol. 7, no. 4, pp. 308-313, Jan. 1965.

POWELL, M. J. D. A method for minimizing a sum of squares of non-linear functions without calculating derivatives, Computer Journal, vol. 7, p. 303, 1964.

POWELL, M. J. D. An efficient method of finding the minimum of a function of several variables without calculating derivatives, Computer Journal, vol. 7, p. 155, 1964.

POWELL, M. J. D. An iterative method for finding stationary values of a function of several variables, Computer Journal, vol. 5, no. 2, pp. 147-151, 1962.

ROSENBROCK, H. H. An automatic method for finding the greatest or least value of a function, Computer Journal, vol. 3, no. 3, pp. 175-184, Oct. 1960.

SHAH, V. B., R. J. Buehler, and O. Kempthorne. The method of parallel tangents (partan) for finding an optimum, Technical Report No. 2, Iowa State University Statistical Laboratory, Ames, Iowa, Apr. 1961 (revised Aug. 1962).

SHAH, B. V., R. J. Buehler, and O. Kempthorn. Some algorithms for minimizing a function of several variables, J. Soc. Indust. Appl. Math., vol. 12, no. 1, pp. 74-92, Mar. 1964.

SPENDLEY, W., G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimization and evolutionary operation, Techometrics, vol. 4, pp. 441-461, 1962.

WILDE, D. J. Optimization by the method of contour tangents, AICHE Journal, vol. 9, no. 2, p. 186, 1963.

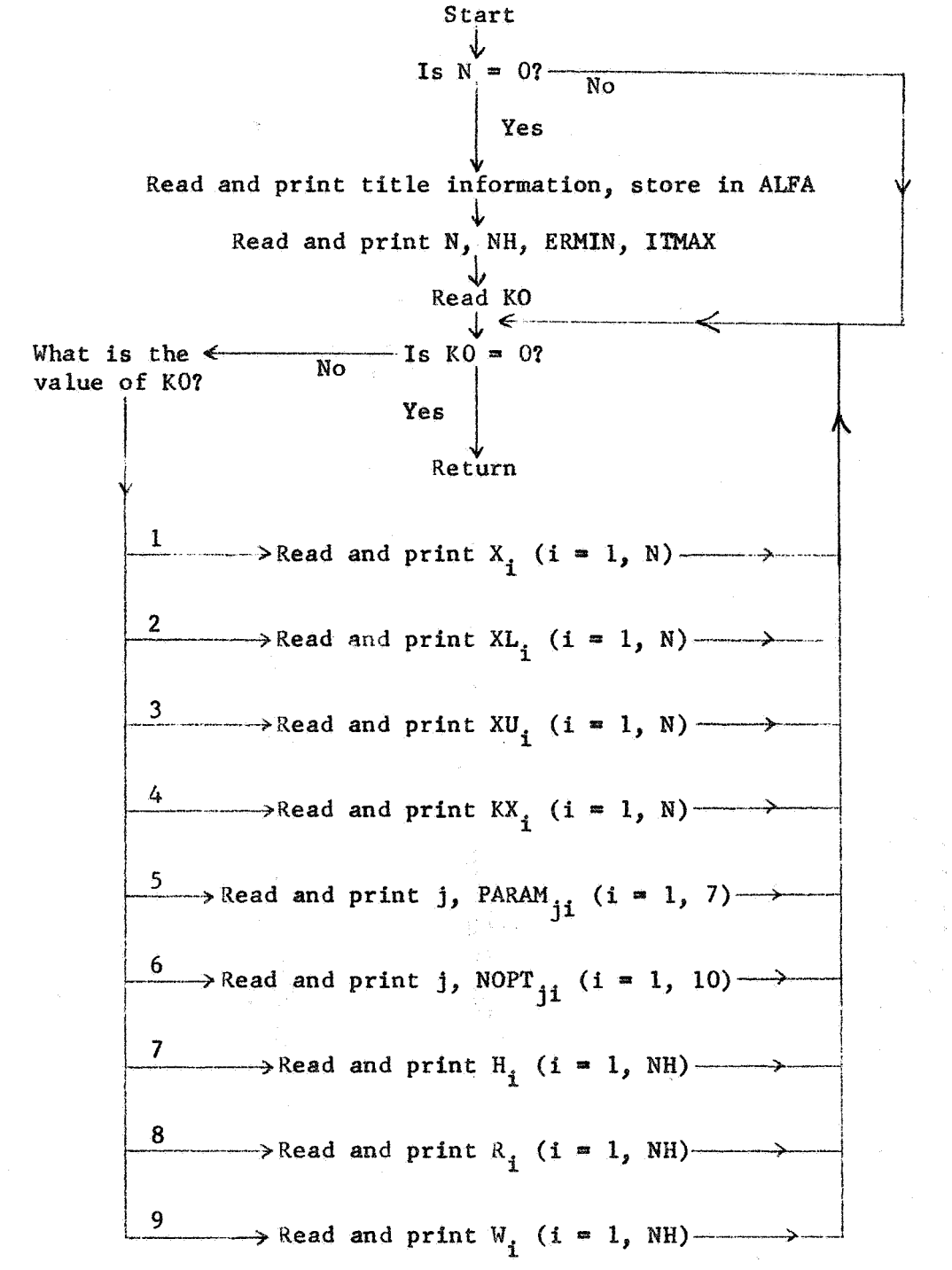
ZANGWILL, W. I. Minimizing a function without calculating derivatives, Computer Journal, vol. 11, pp. 293-296, Feb. 1968.

ZELLNIK, H. E., N. E. Sondak, and R. S. Davis. Gradient search optimization, Chemical Engineering Progress, vol. 53, pp. 35-41, 1962.

APPENDIX

This appendix contains listings and flow charts for the following subroutines: ERR (listing only), RDOPT, PROPT, ANLYZ (for test problem 1), ANLYZ (for test problem 2), OPT2, OPT3, OPT4, OPT6, OPT7, and OPT9.

Flow Chart for Subroutine RDOPT

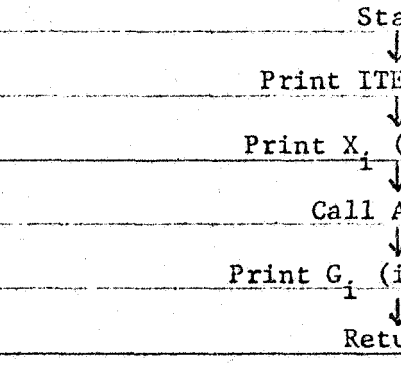


```
      C     SUBROUTINE RDOPT
      C     SUBROUTINE FOR READING DATA
00002       COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
      C     INITALIZE VARIABLE N AND ARRAYS NOPT, PARAM, W
00002       DATA N/0/,NOPT/100*0/,PARAM/70*0./,W/20*1./
00002       IF (N.NE.0) GO TO 104
000003      READ 99,ALFA
000011   99 FORMAT (8A10)
00011       PRINT 98,ALFA
000017   98 FORMAT (1H1,8A10)
000017      READ 100,N,NH,ITMAX,ERMIN
00033   100 FORMAT (2I2,I3,3X,F10.0)
00033       PRINT 103,ERMIN,ITMAX
000043  103 FORMAT (16H0MINIMUM ERROR =,F10.3,5X,20HMAXIMUM ITERATIONS =,I4/)
00043       PRINT 101,N,NH
00053   101 FORMAT ( 1H0,10H1NPUT DATA,I5,10H VARIABLES,I5,12H DATA POINTS/)
000053  104 READ 100, KO
00061       IF (KO) 250,250,110
00063   110 GO TO (115,135,145,155,175,195,215,225,235),KO
000100  115 READ 120, (X(I),I=1,N)
000113  120 FORMAT (8E10.0)
00113       PRINT 125
00117   125 FORMAT ( /1H0,27HINITIAL VALUES OF VARIABLES )
000117      PRINT 130, (X(I),I=1,N)
00132   130 FORMAT (5E1).3)
00132       GO TO 104
000133  135 PRINT 140
00137   140 FORMAT ( /1H0,24HLOWER BOUND OF VARIABLES )
00137       READ 120, (XL(I),I=1,N)
00152       PRINT 130, (XL(I),I=1,N)
000165      GO TO 104
00166   145 PRINT 150
00172   150 FORMAT ( /1H0,24HUPPER BOUND OF VARIABLES )
000172      READ 120, (XU(I),I=1,N)
00205       PRINT 130, (XU(I),I=1,N)
00220       GO TO 104
000221  155 PRINT 160
000225  160 FORMAT ( /1H0,33HNUMBER OF VALUES OF EACH VARIABLE )
00225       READ 165,(KX(I),I=1,N)
00240   165 FORMAT (40I2)
000240      PRINT 170, (KX(I),I=1,N)
00253   170 FORMAT (20I4)
00253       GO TO 104
000254  175 PRINT 180
00260   180 FORMAT ( /1H0,16HINPUT PARAMETERS )
00260       READ 185, KP, (PARAM(KP,I),I=1,7)
000276  185 FORMAT (I2,8X,7E10.0)
000276      PRINT 190,KP
00304   190 FORMAT ( 1H0,24HOPTIMIZATION ROUTINE NO.,I3)
00304       PRINT 130, (PARAM(KP,I),I=1,7)
000320      GO TO 104
00321   195 PRINT 200
00325   200 FORMAT ( /1H0,14HOPTION CHOICES )
000325      READ 205, KP, (NOPT(KP,I),I=1,10)
00343   205 FORMAT (I2,8X,10I1)
00343       PRINT 190,KP
```

```
000351          PRINT 210, (NOPT(KP,I), I=1,10)
000365      210 FORMAT (10I4)
000365          GO TO 104
000366      215 PRINT 220
000372      220 FORMAT ( /1H0,11HDATA POINTS )
000372          READ 120,(H(I),I=1,NH)
000405          PRINT 130,(H(I),I=1,NH)
000420          GO TO 104
000421      225 PRINT 230
000425      230 FORMAT ( /1H0,14HDESIRED VALUES )
000425          READ 120,(R(I),I=1,NH)
000440          PRINT 130, (R(I),I=1,NH)
000453          GO TO 104
000454      235 PRINT 240
000460      240 FORMAT ( /1H0,16HWEIGHTING VALUES )
000460          READ 120,(W(I),I=1,NH)
000473          PRINT 130,(W(I),I=1,NH)
000506          GO TO 104
000507      250 RETURN
000510          END
```

```
000002      SUBROUTINE PROPT
            COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
000002      PRINT 110
000006  110 FORMAT (///1H0,31HSUMMARY OF OPTIMIZATION RESULTS///)
000006      PRINT 115, ITER,YERR
000016  115 FORMAT (15,11H ITERATIONS,   E20.8,12H FINAL ERROR,//)
000016      PRINT 120
000022  120 FORMAT (1H0,23H  FINAL VARIABLE VALUES/)
000022      PRINT 130, (I,X(I),I=1,N)
000037  130 FORMAT (I5,  E20.8)
000037      CALL ANLYZ
000040      PRINT 135
000044  135 FORMAT (1H0,2X,20HFINAL VALUES OF G(I)/)
000044      PRINT 130, (I,G(I),I=1,NH)
000061      RETURN
000062      END
```

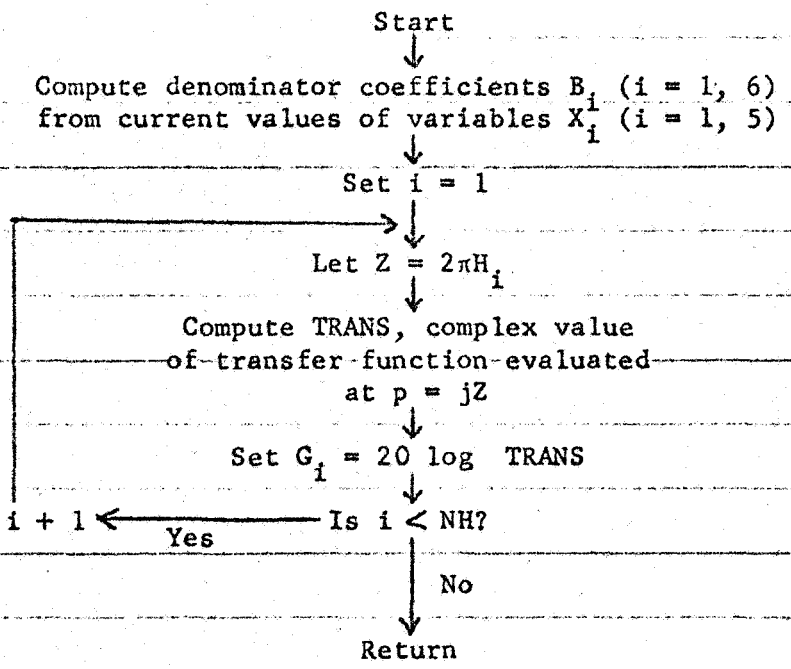
Flow Chart for Subroutine PROPT




```
      SUBROUTINE ERR
C     SUBROUTINE FOR CALCULATING THE ERROR FUNCTION
      COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
     1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
      YERR =0.0
      DO 70 I=1,NH
   70 YERR=YERR+W(I)*(R(I)-G(I))**2
      RETURN
      END
```

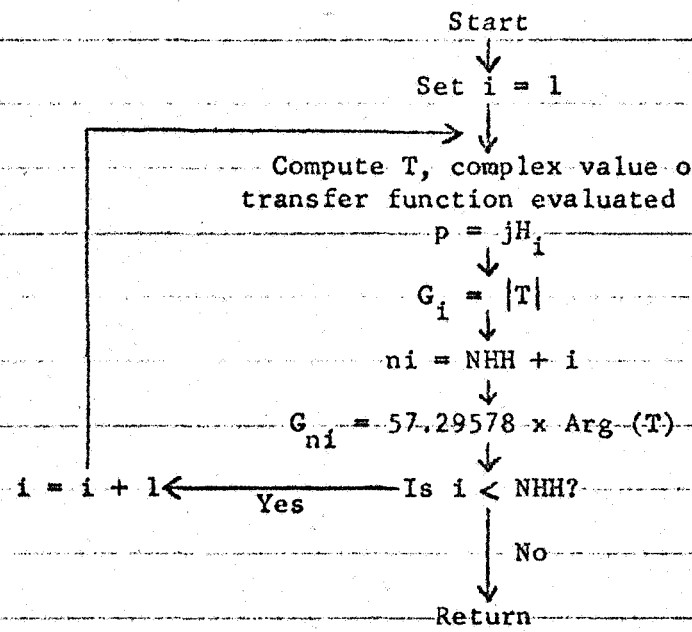
```
      SUBROUTINE ANLYZ
00002 COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),WA(20)
     1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
00002 COMPLEX TRANS
00002 DIMENSION A(10),B(10)
00002 A(1)=1.
00004 B(1)=2.
00005 B(2)=X(1)+X(2)+X(3)+X(4)+X(5)
00012 B(3)=X(1)*X(2) + X(1)*X(4) + X(3)*X(4) + X(2)*X(3) + X(2)*X(5)
     1    + X(4)*X(5)
00021 B(4)=X(1)*X(2)*X(3) + X(1)*X(2)*X(5) + X(1)*X(4)*X(5)
     1    + X(3)*X(4)*X(5) + X(2)*X(3)*X(4)
00031 B(5)=X(1)*X(2)*X(3)*X(4) + X(2)*X(3)*X(4)*X(5)
00036 B(6)=X(1)*X(2)*X(3)*X(4)*X(5)
00040 DO 110 I=1,NH
00042 Z=H(I)*6.28318
00045 TRANS=A(1)/CMPLX( B(5)*Z**4-B(3)*Z**2+B(1) ,
     $    B(6)*Z**5-B(4)*Z**3+B(2)*Z )
00071 G(I)=CABS(TRANS)
00075 110  G(I)= 20.0*ALOG10( G(I) )
00105 RETURN
00105 END
```

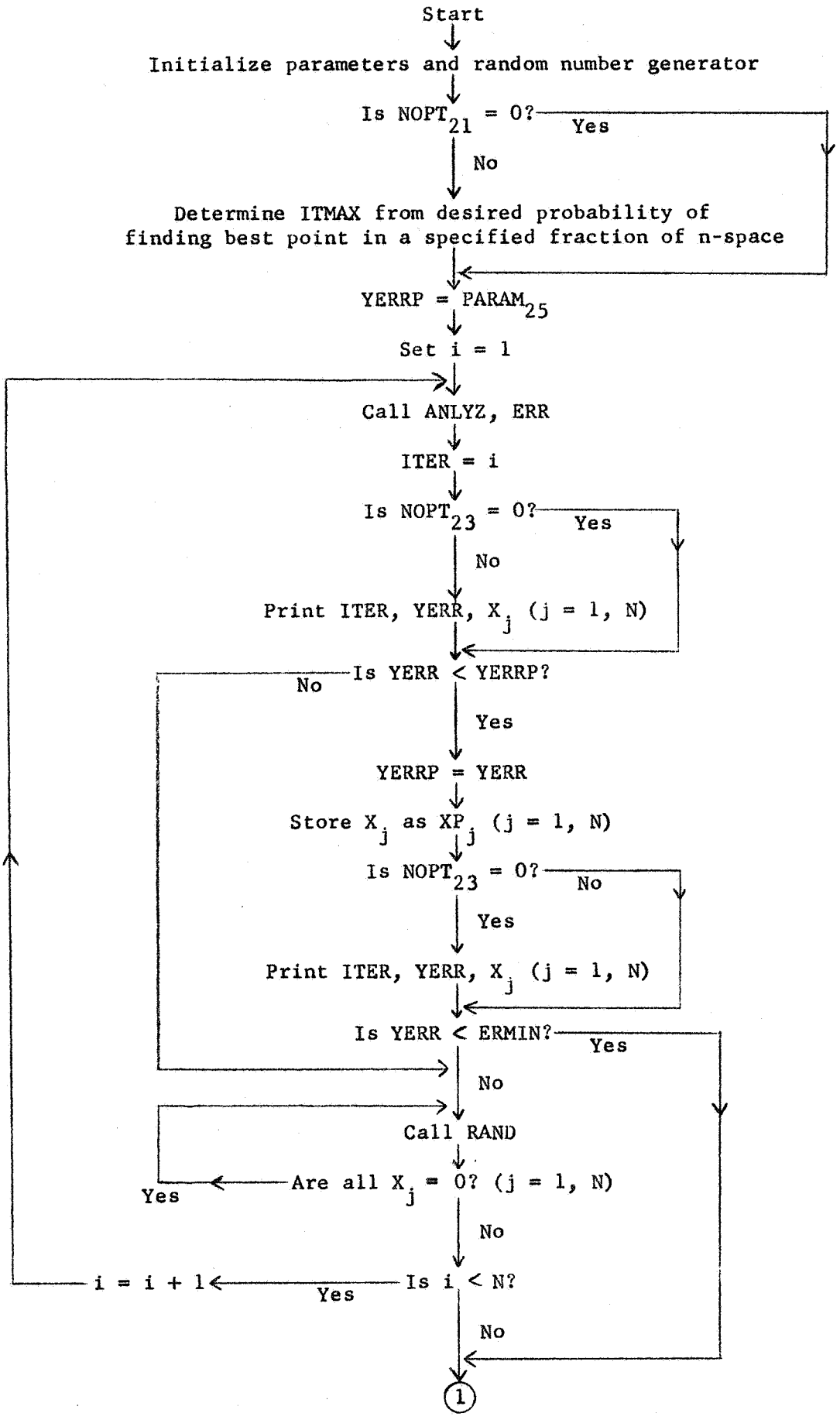
Flow Chart for Test Problem 1

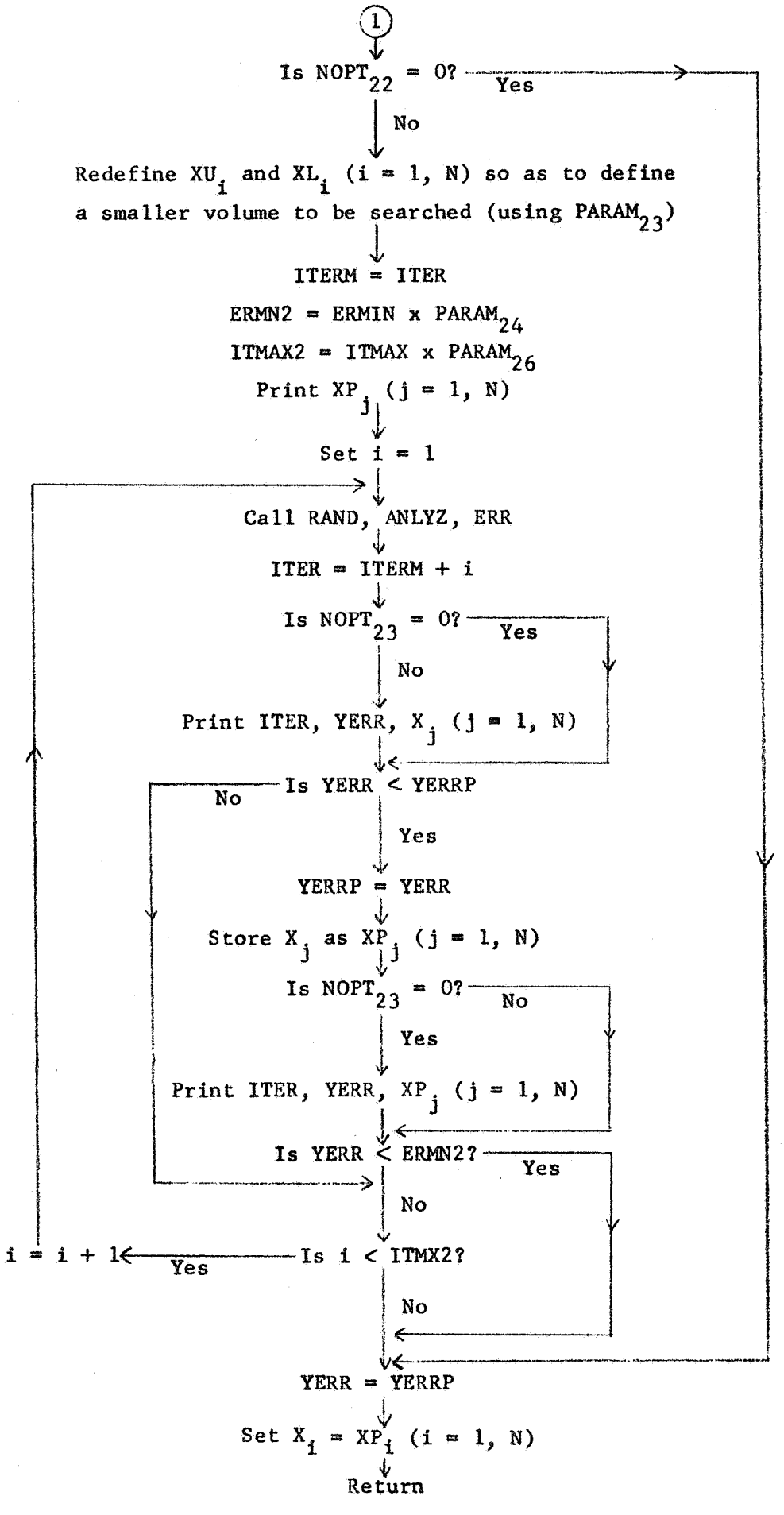


```
      SUBROUTINE ANLYZ                                                    01
C MODIFIED VERSION OF OPT NUMBER THREE FOR 6400                           01
00002 COMMON X,XL,XU,KX,XP,H,R,N,G,PARAM,NOPT,N,NH,YERR,ITER,ERMIN,ITMAX  01
00002 DIMENSION X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),N(20),      01
     1G(20),PARAM(10,7),NOPT(10,10)                                       01
00002 COMPLEX T                                                           01
00002 NHH=NH/2                                                            01
00004 DO 200 I=1,NHH                                                      01
00005 T=-X(5)*H(I)**2/(CMPLX(X(2)-H(I)**2, H(I)*X(1))                     01
     $   *CMPLX(X(4)-H(I)**2 , X(3)*H(I) ))                               01
00036 G(I)=CABS ( T )                                                     01
00042 NI=NHH+I                                                            01
00043 G(NI)=57.2957*ATAN2( AIMAG(T),REAL(T) )                             01
00054 200 CONTINUE                                                        01
00057 RETURN                                                              01
00057 END                                                                 01
```

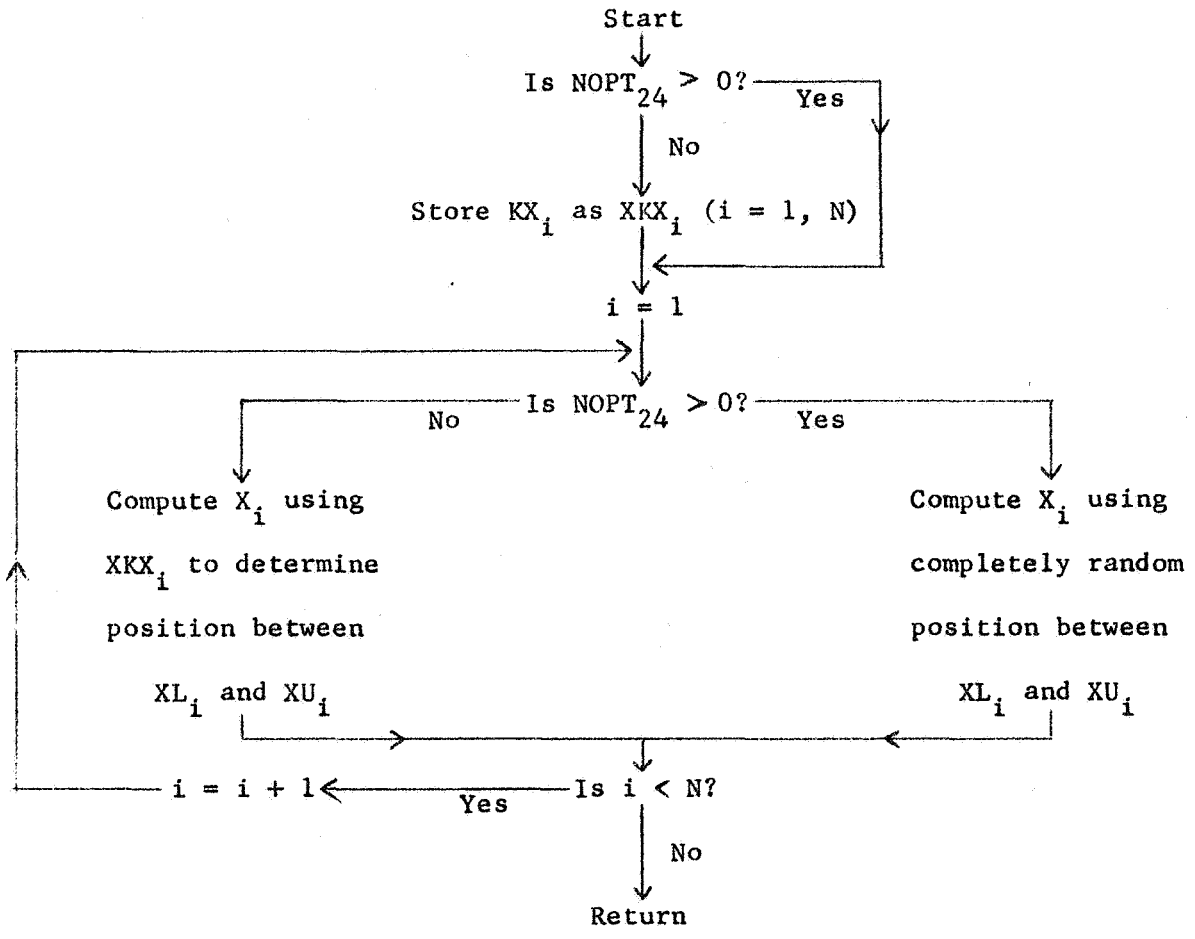
Flow Chart for Test Problem 2




Start
Initialize parameters and random number generator
Is NOPT21 = 0?
Yes
No
Determine ITMAX from desired probability of finding best point in a specified fraction of n-space
YERRP = PARAM25
Set i = 1
Call ANLYZ, ERR
ITER = i
Is NOPT23 = 0?
Yes
No
Print ITER, YERR, Xj (j = 1, N)
Is YERR < YERRP?
No
Yes
YERRP = YERR
Store Xj as XPj (j = 1, N)
Is NOPT23 = 0?
No
Yes
Print ITER, YERR, Xj (j = 1, N)
Is YERR < ERMIN?
Yes
No
Call RAND
Are all Xj = 0? (j = 1, N)
Yes
No
Is i < N?
Yes
i = i + 1
No
1


1
Is NOPT22 = 0?
Yes
No
Redefine XUi and XLi (i = 1, N) so as to define a smaller volume to be searched (using PARAM23)
ITERM = ITER
ERMN2 = ERMIN x PARAM24
ITMAX2 = ITMAX x PARAM26
Print XPj (j = 1, N)
Set i = 1
Call RAND, ANLYZ, ERR
ITER = ITERM + i
Is NOPT23 = 0?
Yes
No
Print ITER, YERR, Xj (j = 1, N)
Is YERR < YERRP
No
Yes
YERRP = YERR
Store Xj as XPj (j = 1, N)
Is NOPT23 = 0?
No
Yes
Print ITER, YERR, XPj (j = 1, N)
Is YERR < ERMN2?
Yes
No
i = i + 1
Yes
Is i < ITMX2?
No
YERR = YERRP
Set Xi = XPi (i = 1, N)
Return

Flow Chart for RAND Subroutine used with Random Grid Subroutine OPT2

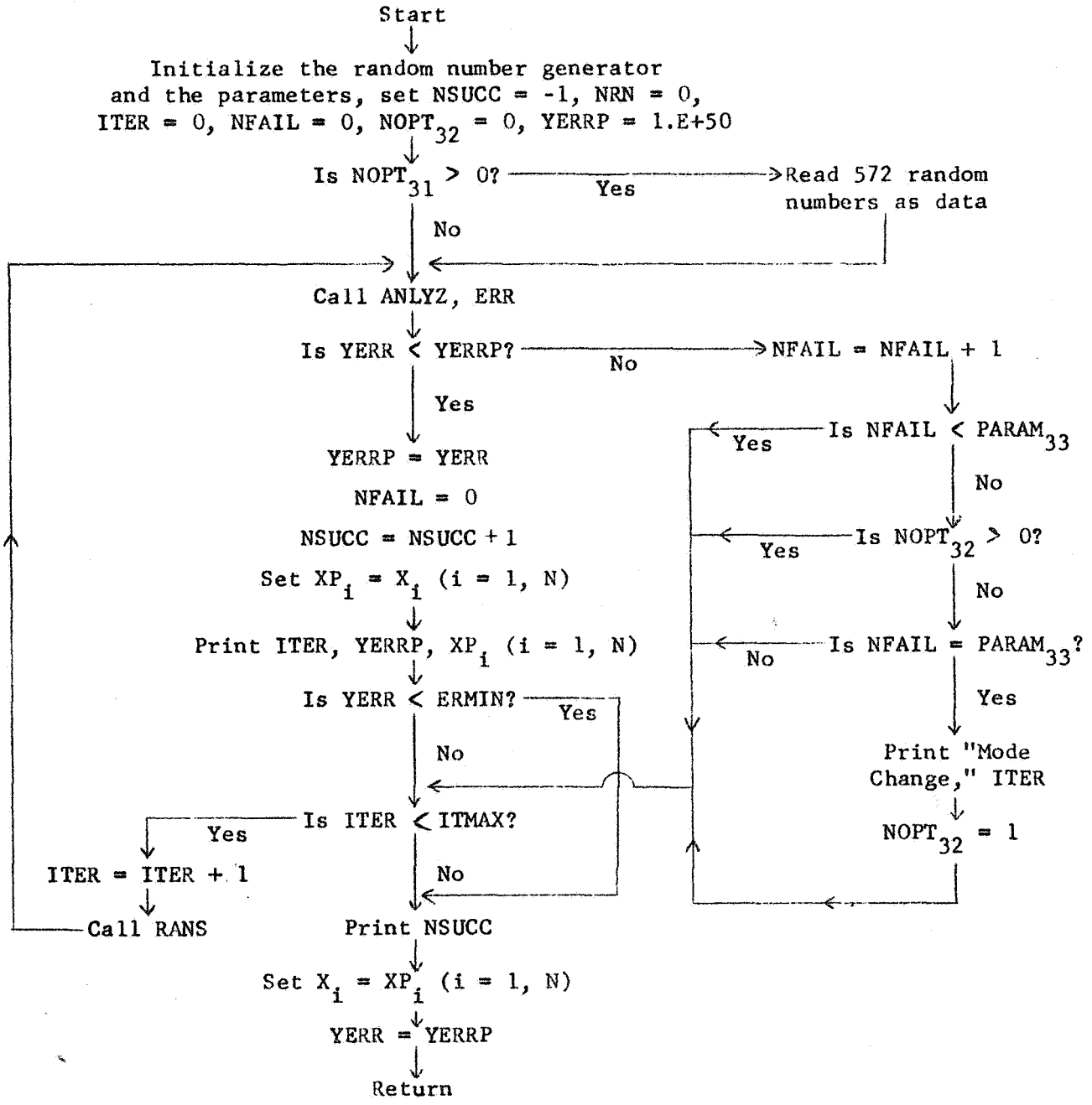


```
      SUBROUTINE OPT2
C-----MAIN SUBROUTINE TO OBTAIN DESIRED OPTIMIZATION
C     SUBROUTINE CALLS ANALYZ,ERR,AND RAND
C      CDC 6400 VERSION APRIL 1968 FOSSUM/HUELSMAN
000002      COMMON /OPT/X(20),XL(20),XU(20),KA(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NM,YERR,ITER,ERMIN,ITMAX,ALFA(8)
C      INITALIZE UNSPECIFIED PARAMETERS
000002      IF (PARAM(2,1).EQ.0.) PARAM(2,1)=.05
000004      IF (PARAM(2,2).EQ.0.) PARAM(2,2)=.95
000006      IF (PARAM(2,3).EQ.0.) PARAM(2,3)=.2
000010      IF (PARAM(2,4).EQ.0.) PARAM(2,4)=.1
000012      IF (PARAM(2,5).EQ.0.) PARAM(2,5)=1.E+20
000014      IF (PARAM(2,6).EQ.0.) PARAM(2,6)=1.
000016      PRINT 610
000022  610 FORMAT (1H0*RANDOM GRID SEARCH ROUTINE OPT2 HAS BEEN CALLED*/)
000022      PRINT 611,(PARAM(2,I),I=1,6),(NOPT(2,I),I=1,5)
000044  611 FORMAT (
           1 1X*PARAM(2,1)-FRACTION OF TOTAL VOLUME.............*E9.2/
           2 1X*PARAM(2,2)-DESIRED PROBABILITY..................*E9.2/
           3 1X*PARAM(2,3)-LOCAL VOLUME TO BE SEARCHED..........*E9.2/
           4 1X*PARAM(2,4)-REDUCTION OF MINIMUM ERROR...........*E9.2/
           5 1X*PARAM(2,5)-ARBITRARY ORIGINAL ERROR.............*E9.2/
           6 1X*PARAM(2,6)-RATIO FOR ITERATIONS FOR LOCAL SEARCH..*E9.2/
           7 1X*NOPT(2,1)-USE PROBABILITY DATA TO COMPUTE ITERATIONS (1)..*I2/
           8 1X*NOPT(2,2)-MAKE LOCAL SEARCH AROUND BEST POINT (1).........*I2/
           9 1X*NOPT(2,3)-PRINT SUCCESSFUL (0) OR ALL (1) ITERATIONS......*I2/
           1 1X*NOPT(2,4)-USE KX (0) OR COMPLETELY RANDOM (1) VALUES......*I2/
           1 1X*NOPT(2,5)-INITIALIZATION FOR INTERNAL RANDOM GENERATOR....*I2/)
000044      Z3=1.
000046      IF (NOPT(2,5).GT.0) Z3=NOPT(2,5)
000051      Z4=RANF(Z3)
000054      Z5=RANF(0.)
000056      IF(NOPT(2,1)) 5, 6, 5
C-----OPTION 1 - DETERMINE ITMAX FROM PROBABILITY OF FINDING A GRID
C                POINT IN A BEST FRACTION OF THE TOTAL VOLUME IN THE
C                N-SPACE
C-----PARAM(2,1) IS THE DESIRED FRACTION OF THE TOTAL VOLUME
C-----PARAM(2,2) IS THE PROBABILITY OF FINDING A POINT IN THE BEST
C     (100*PARAM(2,1)) PER CENT OF THE TOTAL VOLUME
C-----PARAM(2,2) = 1.0 - (1.0 - PARAM(2,1))**ITMAX
000057    5 ITMAX = (ALOG (1.0 - PARAM(2,2)))/(ALOG (1.0 - PARAM(2,1))) + 1.0
000073    6 YERRP = PARAM(2,5)
C-----PARAM(2,5) IS THE ARBITRARILY CHOSEN INITIAL ERROR TO BE
C     IMPROVED
000075      DO 200 I = 1, ITMAX
000076      CALL ANLYZ
000077      ITER = I
000101      CALL ERR
000102      IF(NOPT(2,3)) 500, 501, 500
C-----OPTION 3 - PRINT OUT RESULTS OF EVERY ITERATION
C-----IF OPTION 3 IS NOT USED, THE RESULTS OF ONLY THOSE ITERATIONS
C     WHICH YIELD AN IMPROVEMENT IN THE ERROR WILL BE PRINTED OUT
000103  500 PRINT 999, ITER, YERR, (X(IP), IP = 1, N)
000122  999 FORMAT(/ 15H ITERATION NO. , I5 / 8H YERR = , E12.5, 8H   AT   ,
           16E11.4 / (28X, 8E11.4))
000122  501 IF(YERR - YERRP) 225, 800, 800
000125  225 YERRP = YERR
```

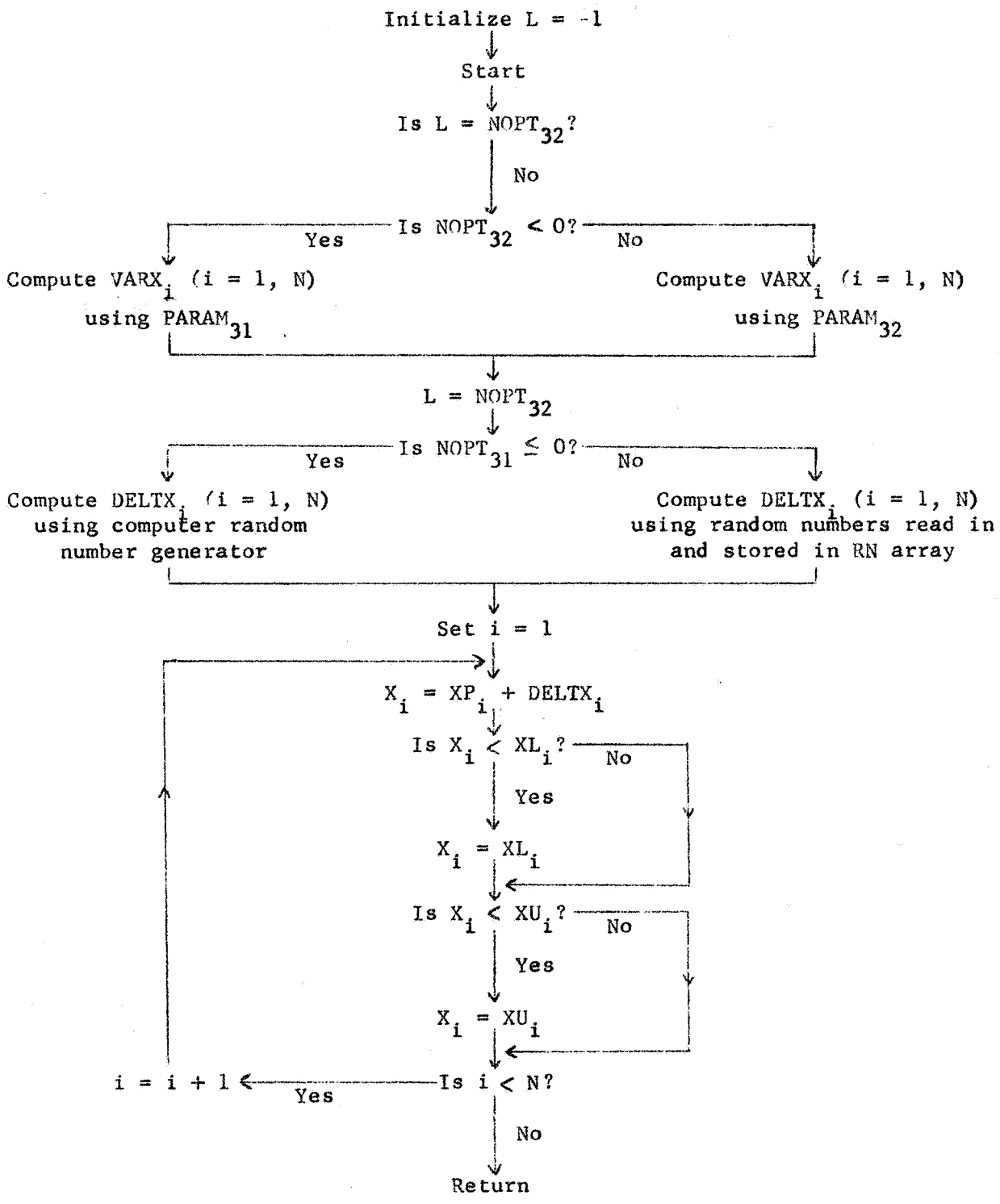
```
000127          DO 50 J = 1, N
000130       50 XP(J) = X(J)
000135          IF(NOPT(2,3)) 504, 503, 504
000136      503 PRINT 502, ITER, YERRP, (IP, XP(IP), IP = 1, N)
000157      502 FORMAT(// 36H ERROR IMPROVEMENT AT ITERATION NO. , I5, / 5X,8HERRO
                1R = , E12.5, 6H   AT    , 2HX(, I2, 4H) = , E11.4, / (33X, 2HX(,
                2I2, 4H) = , E11.4))
000157      504 IF(YERR - ERMIN) 125, 800, 800
000162      800 CALL RAND
000163          DO 90 J0 = 1, N
000165          IF(X(J0)) 200, 90, 200
000167       90 CONTINUE
000172          GO TO 800
000172      200 CONTINUE
000175      125 IF(NOPT(2,2)) 150, 250, 150
            C-----OPTION 2 - SEARCH LOCALLY AROUND THE BEST POINT IN THE PARAMETER
            C                SPACE
000176      150 DO 300 I = 1, N
000200          XLB = XL(I)
000202          XL(I) = XP(I) - PARAM(2,3)*(XU(I) - XLB)
            C-----PARAM(2,3) DEFINES THE LOCAL VOLUME TO BE SEARCHED
000210      300 XU(I) = XP(I) + PARAM(2,3)*(XU(I) - XLB)
000217          ITERM = ITER
000221          ERMN2 = ERMIN*PARAM(2,4)
            C-----PARAM(2,4) SPECIFIES A SMALLER MINIMUM ERROR
000223          ITFAC = PARAM(2,6)
000224          ITMX2 = ITMAX*ITFAC
            C-----PARAM(2,6) DEFINES THE MAXIMUM NUMBER OF ITERATIONS ALLOWED IN
            C       THE LOCAL SEARCH
000227          PRINT 998, (XP(JP), JP = 1, N)
000241      998 FORMAT(/ 21H LOCAL SEARCH AROUND , 9E11.4 / (21X, 9E11.4))
000241          DO 350 I = 1, ITMX2
000243          CALL RAND
000244          CALL ANLYZ
000245          ITER = ITERM + I
000247          CALL ERR
000250          IF(NOPT(2,3)) 550, 551, 550
000251      550 PRINT 999, ITER, YERR, (X(IP), IP = 1, N)
000270      551 IF(YERR - YERRP) 600, 350, 350
000273      600 YERRP = YERR
000275          DO 75 J = 1, N
000276       75 XP(J) = X(J)
000303          IF(NOPT(2,3)) 552, 553, 552
000304      553 PRINT 502, ITER, YERRP, (IP, XP(IP), IP = 1, N)
000325      552 IF(YERR - ERMN2) 250, 350, 350
000330      350 CONTINUE
000333      250 YERR = YERRP
000335          DO 400 J = 1, N
000336      400 X(J) = XP(J)
000343          RETURN
000344          END
```

```
      SUBROUTINE RAND
C     RANDOM PARAMETER GENERATOR,HUELSMAN/FOSSUM 4/68
000002      COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
000002      DIMENSION XKX(20)
000002      IF (NOPT(2,4).GT.0) GO TO 12
000005      DO 10 I=1,N
000006   10 XKX(I)=KX(I)
000014   12 DO 20 I=1,N
000016      IF (NOPT(2,4).GT.0) GO TO 15
000020      KKX=RANF(0.)*XKX(I)-.0001
000026      RNNM=KKX
000027      XA=KX(I)-1
000032      X(I)=XL(I)+RNNM*(XU(I)-XL(I))/XA
000041      GO TO 20
000042   15 X(I)=XL(I)+RANF(0.)*(XU(I)-XL(I))
000055   20 CONTINUE
000060      RETURN
000060      END
```

Flow Chart for Random Pattern Search Subroutine OPT3



Flow Chart for Subroutine RANS used with Subroutine OPT3

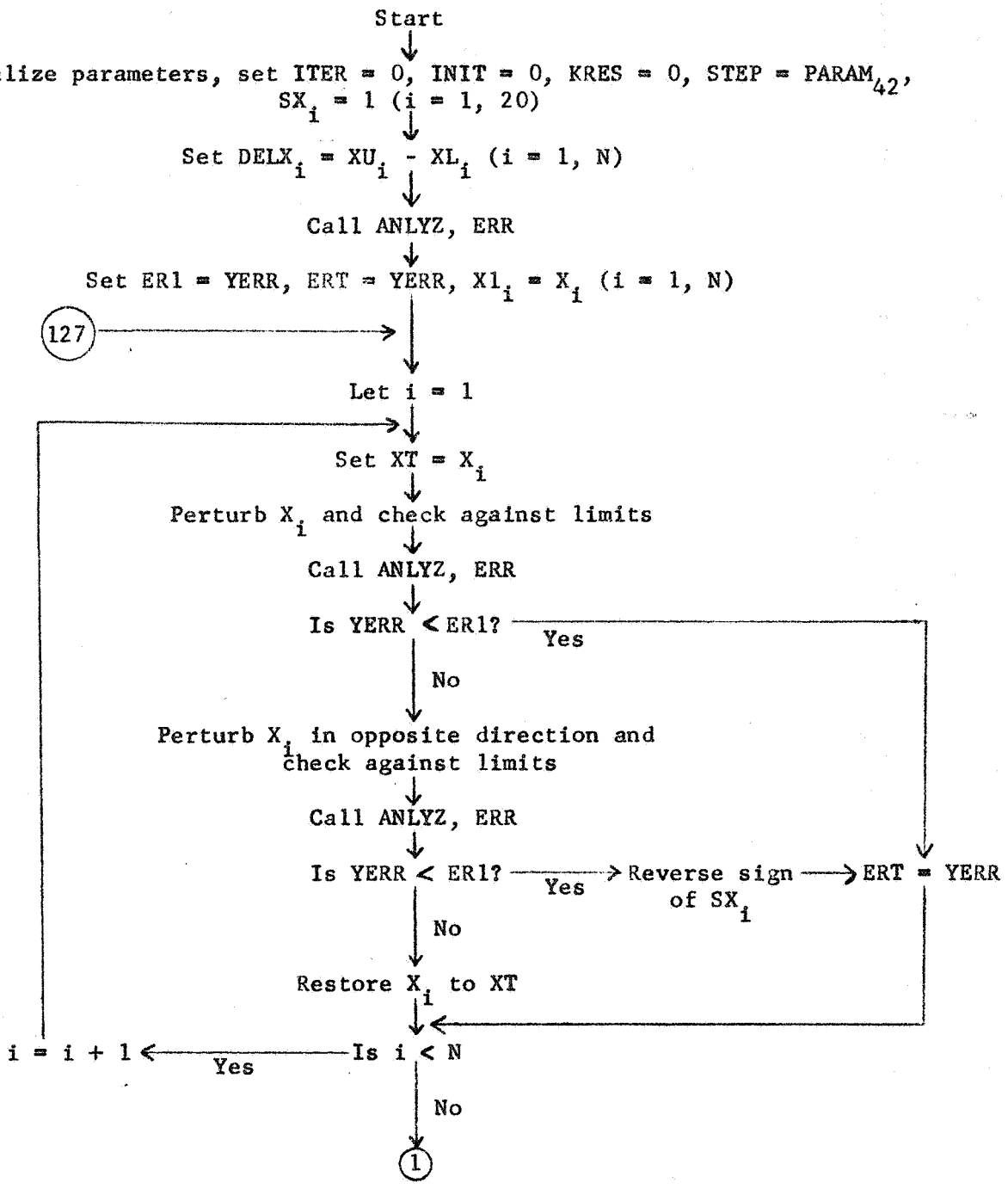


```
      SUBROUTINE OPT3
C     SUBROUTINE, RANDOM DIRECTION AND STEP SIZE SEARCH
C     FORTRAN 4, CDC 6400,4/68,UA
000002      COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
00002       DIMENSION RN(600)
C     INITIALIZE THE PARAMETER VALUES AND THE RANDOM
C     NUMBER GENERATOR
00002       Z3=1.
00004       IF (NOPT(3,3).GT.0) Z3=NOPT(3,3)
000007      Z4=RANF(Z3)
00012       Z5=RANF(0.)
00014       NRNMX=572
000015      NSUCC=-1
000016      NRN=0
00017       ITER=0
00020       YERRP=1.E+50
00021       NFAIL=0
00022       NOPT(3,2)=0
00023       IF (PARAM(3,1).EQ.0.) PARAM (3,1)=.25
000025      IF (PARAM(3,2).EQ.0.) PARAM (3,2)=.05
00027       IF (PARAM (3,3).EQ.0.) PARAM (3,3)=40.
00031       PRINT 900
000035    900 FORMAT(//1X,
           1*OPTIMIZATION PROGRAM 3 HAS BEEN CALLED, PARAMETERS FOLLOW*/)
00035       PRINT 901, (PARAM(3,I),I=1,3),NOPT(3,1),NOPT(3,3)
00054     901 FORMAT(
           11X*PARAM(3,1)-INITIAL PER UNIT MAX VARIABLE CHANGE=*E9.2/
           21X*PARAM(3,2)-FINAL   PER UNIT MAX VARIABLE CHANGE=*E9.2/
           31X*PARAM(3,3)-NUMBER OF FAILURES BEFORE MODE CHANGE=*F4.0/
           41X*NOPT(3,1)-RANDOM NUMBER SOURCE (0 IS INTERNAL) =*I2/
           51X*NOPT(3,3)-INITIALIZATION FOR INTERNAL RANDOM SOURCE =*I2/)
C     TEST THE FIRST OPTION FOR THE RANDOM NUMBER SELECTION *
000054      IF(NOPT(3,1))2,2,53
000056     53 READ 54,(RN(I),I=1,NRNMX)
00071      54 FORMAT(13F6.3)
00071       2 CALL ANLYZ
000072      3 CALL ERR
C     TEST FOR AN IMPROVED ERROR  ****************************
00073       4 IF(YERRP-YERR)17,17,10
C     IF SUCCESS, UPDATE THE ERROR AND PARAMETER VALUES  ***
000076     10 YERRP=YERR
00100       NFAIL=0
000101      NSUCC=NSUCC+1
000102      DO 12 I=1,N
C     PRINT THE ERROR AND PARAMETER VALUES  ****************
00104      12 XP(I)=X(I)
000111      PRINT 106,ITER,YERRP
00121     106 FORMAT(/28H SUCCESS AT ITERATION NUMBER,I5,10X,8H ERROR =,  E17.8)
00121       PRINT 105
000125    105 FORMAT(/,17H PARAMETER VALUES)
000125      DO 229 I=1,N
00127     229 PRINT 22,I,XP(I)
00142      22 FORMAT(4X,3H X(,I2,4H) = ,  E17.8)
C     TEST FOR ACCEPTABLE ERROR  *****************************
42          IF(YERR-ERMIN)13,13,14
C     IF FAILURE, INCREMENT FAILURE PARAMETER  **************
```

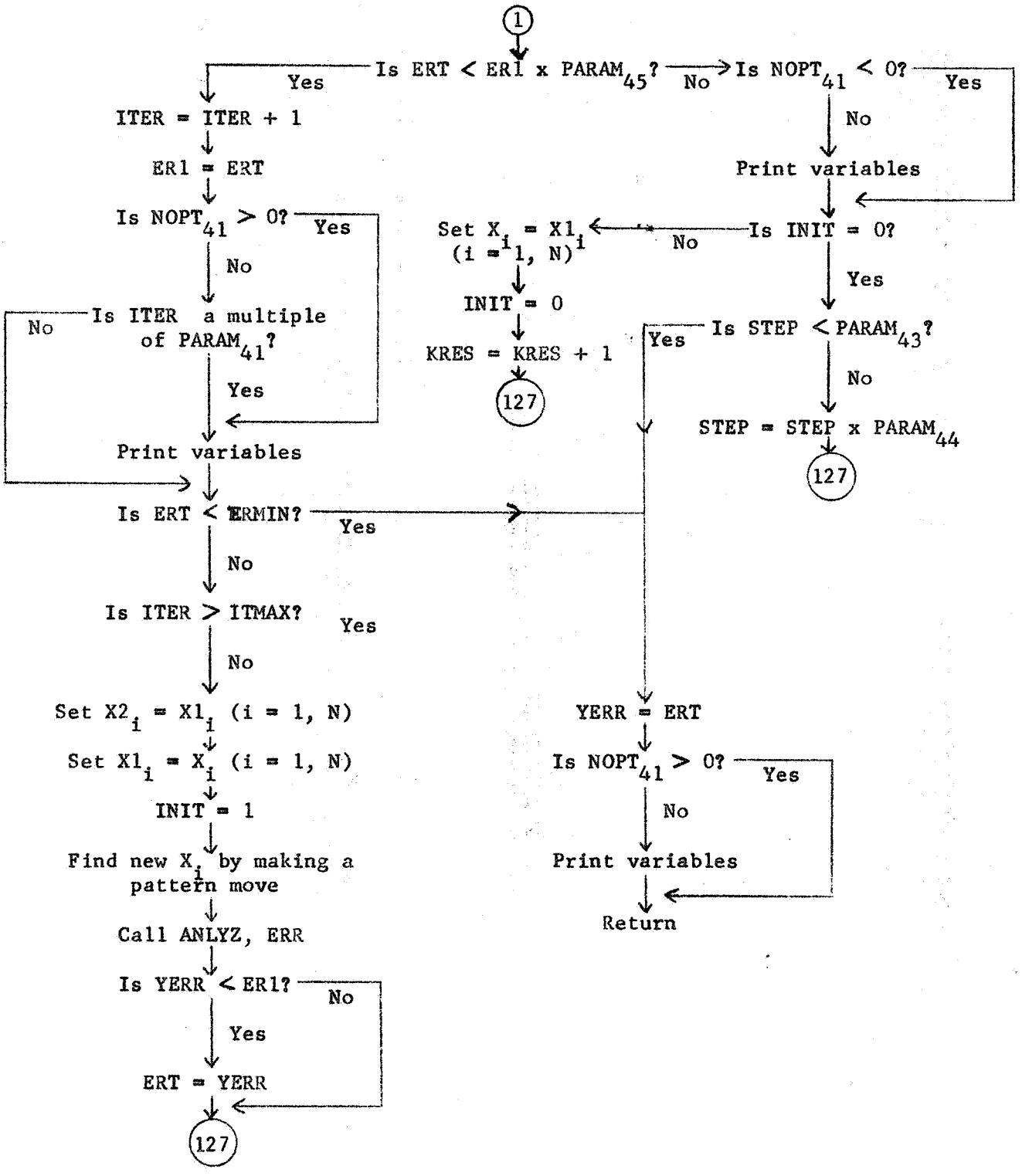
```
000145        17 NFAIL=NFAIL+1
          C      AFTER PARAM(3,3) FAILURES CHANGE THE MODE  ************
000147           NPARA =PARAM(3,3)
000150           IF(NFAIL-NPARA )14,18,18
000152        18 IF(NOPT(3,2).GT.0) GO TO 14
000155           IF(NFAIL-NPARA )14,24,14
000157        24 PRINT 19,ITER
000165        19 FORMAT(/,30H MODE HAS CHANGED AT ITERATION,I5)
000165           NOPT(3,2)=1
          C      TEST FOR MAXIMUM ITERATIONS  ***************************
000166        14 IF(ITER-ITMAX)15,16,16
000171        15 ITER = ITER+1
000173           CALL RANS (NRN,NRNMX,RN)
000175           GO TO 2
          C      EXIT CONDITIONS  ***************************************
000176        13 PRINT 101
000202       101 FORMAT(/,24H EXIT ON ACCEPTABLE YERR)
000202           GO TO 103
000203        16 PRINT 102
000207       102 FORMAT(/,23H EXIT ON MAX ITERATIONS)
000207       103 PRINT 109,NSUCC
000215       109 FORMAT(/,35H NUMBER OF SUCCESSFUL ITERATIONS = ,I5)
000215           DO 23 I=1,N
000217        23 X(I)=XP(I)
000224           YERR=YERRP
000226           RETURN
000226           END
```

```
                SUBROUTINE RANS (NRN,NRNMX,RN)
          C     SUBROUTINE FOR CALCULATING TRIAL PARAMETERS
000005          COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
               1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
000005          DIMENSION RN(600),VARX(20),DELTX(20)
          C     TEST THE SECOND OPTION FOR THE STEP SIZE MODE
000005          DATA L/-1/
000005          IF (L.EQ.NOPT(3,2)) GO TO 91
000007          IF(NOPT(3,2))220,220,221
          C     MODE TWO
000010      221 DO 222 I=1,N
000012      222 VARX(I)=PARAM(3,2)*ABS (XU(I)-XL(I))/3.0
000030          GO TO 91
          C     MODE ONE
000031      220 DO 92 I=1,N
000033       92 VARX(I)=PARAM(3,1)*ABS (XU(I)-XL(I))/3.0
000051       91 L=NOPT(3,2)
          C     TEST THE FIRST OPTION FOR THE RANDOM NUMBER SELECTION
000053          IF(NOPT(3,1))55,55,59
          C     RANDOM NUMBER SELECTION FROM COMPUTER
000054       55 DO 110 I=1,N
000056          ARN= RANF(0.)*100.
000061          IF(ARN-49.0)109,109,118
000065      109 ARN=-ARN
000066          GO TO 110
000067      118 ARN=ARN-49.0
000071      110 DELTX(I)=VARX(I)*ARN/17.0
000100          GO TO 111
          C     RANDOM NUMBER SELECTION, FROM VALUES READ IN
000100       59 DO 102 I=1,N
000102          NRN=NRN+1
000104          IF (NRN-NRNMX)102,102,103
000105      103 NRN=1
000106      102 DELTX(I)=VARX(I)*RN(NRN)
          C     CALCULATE TRIAL PARAMETERS
000115      111 DO 104 I=1,N
000117          X(I)=XP(I)+DELTX(I)
000123          IF(X(I)-XL(I))105,106,106
000127      105 X(I)=XL(I)
000132      106 IF(XU(I)-X(I))107,104,104
000136      107 X(I)=XU(I)
000141      104 CONTINUE
000144          RETURN
000144          END
```

Flow Chart for Pattern Search Subroutine OPT4



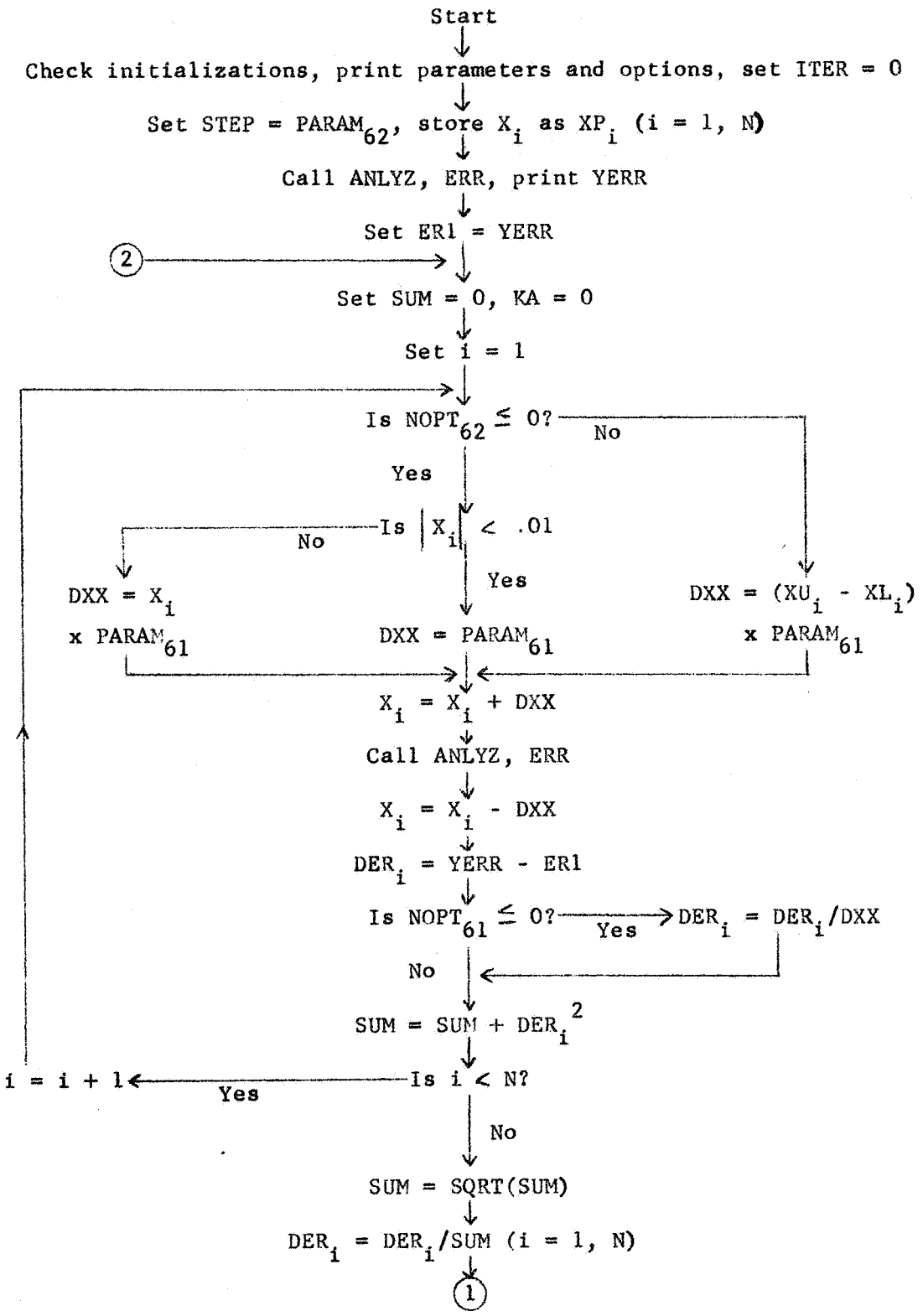
Flow Chart for Pattern Search Subroutine OPT4 (page 2)



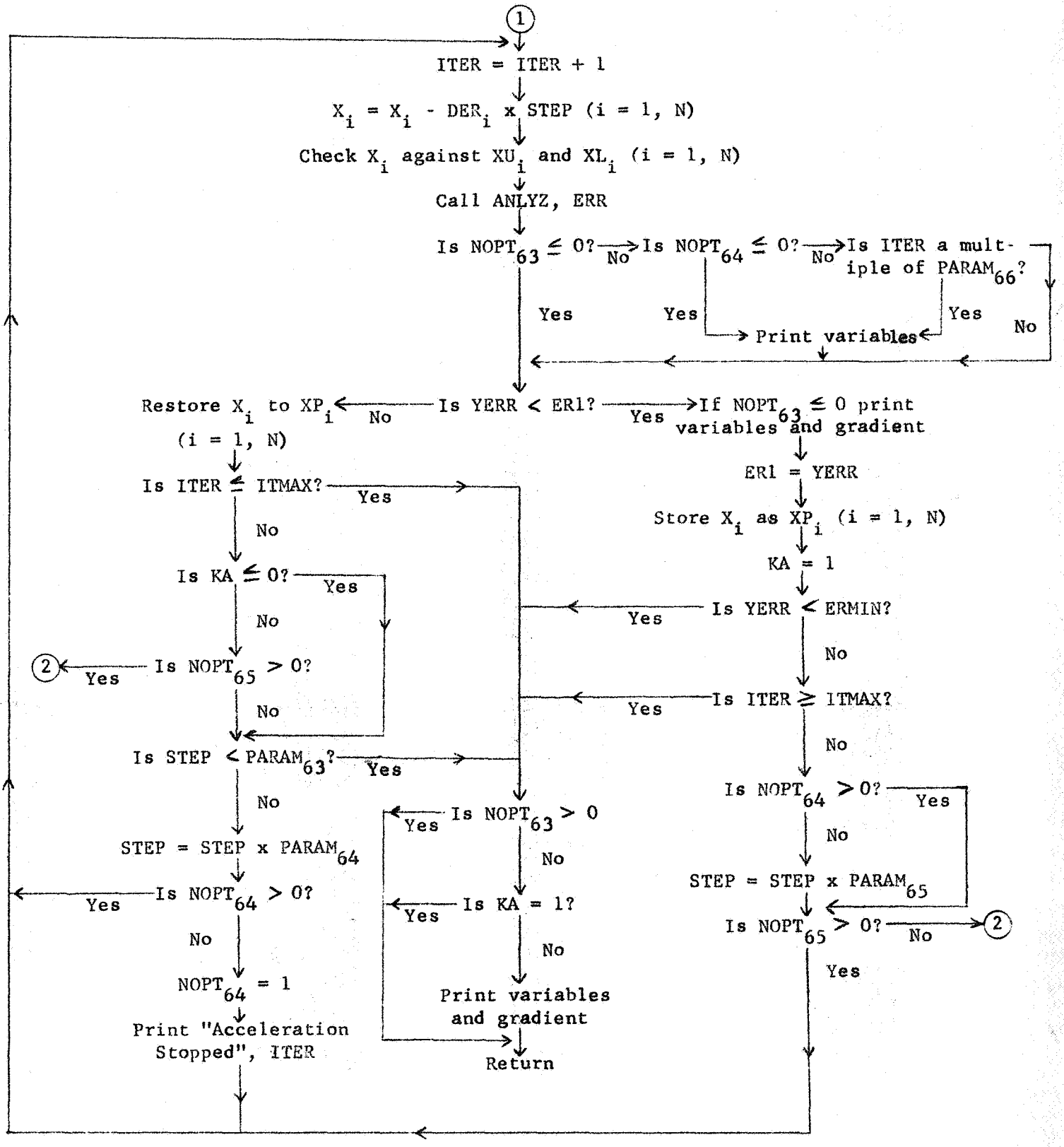
```
      SUBROUTINE OPT4
C     PATTERN SEARCH SUBROUTINE, REVISION # 3-05
000002      COMMON /OPT/X(20),XL(20),XU(20),NA(20),XP(20),G(20),F(20),G(20)
     1,G(20),PARAM(10,7),NOPT(10,10),N,NM,YERR,ITEM,ERMIN,ITMAX,ALFA(3)
000007      DIMENSION SX(20),DELX(20),X1(20),X2(20)
000012      DATA SX/20*1./
000022      IF (PARAM(4,1).LE.0.) PARAM(4,1)=10.
000025      IF (PARAM(4,2).LE.0.) PARAM(4,2)=.05
000030      IF (PARAM(4,3).LE.0.) PARAM(4,3)=.0001
000033      IF (PARAM(4,4).LE.0.) PARAM(4,4)=.5
000036      IF (PARAM(4,5).LE.0) PARAM(4,5)=.999
000041      PRINT 105,(PARAM(4,I),I=1,5),NOPT(4,1)
000056    105 FORMAT (//,1X,*PATTERN SEARCH OPTIMIZATION SUBROUTINE OPT4 HAS BE
     1 CALLED*//
     21X,*PARAM(4,1)=PRINTOUT EVERY NTH STEP, VALUE OF N.....*E9.2/
     41X,*PARAM(4,2)=INITIAL VALUE FOR STEP SIZE.............*E9.2/
     51X,*PARAM(4,3)=MINIMUM VALUE FOR STEP SIZE.............*E9.2/
     61X,*PARAM(4,4)=REDUCTION FACTOR FOR STEP SIZE..........*E9.2/
     71X,*PARAM(4,5) = FACTOR FOR REDUCTION OF ERROR.........*E9.2/
     81X,*NOPT(4,1)=PRINTOUT EVERY STEP (1=YES)..............*I2/)
000056      ITER=0
000057      INIT=0
000060      KRES=0
000061      STEP=PARAM(4,2)
000062      DO 106 I=1,N
000064    106 DELX(I)=XU(I)-XL(I)
000103      IP=PARAM(4,1)
000104      CALL ANLYZ
000105      CALL ERR
000106      PRINT 110,YERR
000114    110 FORMAT (1H0,*INITIAL VALUE OF ERROR=*E11.4)
000114      PRINT 115,(X(I),I=1,N)
000127    115 FORMAT (1X,4HX(I),2X,5E13.5)
000127    120 ER1=YERR
000131      ERT=YERR
000132    122 DO 125 I=1,N
000134    125 X1(I)=X(I)
000141    127 DO 135 I=1,N
000143      XT=X(I)
000145      X(I)=X(I)+SX(I)*STEP*DELX(I)
000153      IF (X(I).LT.XL(I)) X(I)=XL(I)
000161      IF (X(I).GT.XU(I)) X(I)=XU(I)
000170      CALL ANLYZ
000171      CALL ERR
000172      IF (YERR.LT.ERT) GO TO 130
000175      X(I)=X(I)-2.*SX(I)*STEP*DELX(I)
000203      IF (X(I).LT.XL(I)) X(I)=XL(I)
000212      IF (X(I).GT.XU(I)) X(I)=XU(I)
000221      CALL ANLYZ
000222      CALL ERR
000223      IF (YERR.LT.ERT) GO TO 129
000226      X(I)=XT
000230      GO TO 135
000231    129 SX(I)=-SX(I)
000234    130 ERT=YERR
000236    135 CONTINUE
000241      IF (ERT.LT.ER1*PARAM(4,5))GO TO 210
```

```
000224          IF (NOPT(4,1).LE.0) GO TO 130
000410          PRINT 145,ITER,ERT,STEP,KRES
000431          PRINT 115, (X(I),I=1,N)
000244      130 IF (INIT.EQ.0) GO TO 150
000245          DO 141 I=1,N
000247      141 X(I)=X1(I)
000254          INIT=0
000255          KRES=KRES+1
000256          GO TO 127
000257      150 IF (STEP.LT.PARAM(4,3)) GO TO 171
000262          STEP=STEP*PARAM(4,4)
000263          GO TO 127
000263      210 ITER=ITER+1
000265          ER1=ERT
000265          IF (NOPT(4,1).GT.0) GO TO 140
000271          IF (ITER/IP-(ITER-1)/IP) 215,215,140
000277      140 PRINT 145,ITER,ERT,STEP,KRES
000313      145 FORMAT (1H0*ITERATION=*I4,3X*ERROR=*E11.4,3X*STEP SIZE=*E11.4
               1,3X*NO. OF RESETS=*I4)
000313          PRINT 115,(X(I),I=1,N)
000326      215 IF (ERT.LT.ERMIN) GO TO 171
000331          IF (ITER.GT.ITMAX) GO TO 171
000334          DO 220 I=1,N
000335          X2(I)=X1(I)
000340          X1(I)=X(I)
000342      220 X(I)=2.*X1(I)-X2(I)
000351          INIT=1
000352          CALL ANLYZ
000353          CALL ERR
000354          IF (YERR.LT.ER1) ERT=YERR
000360          GO TO 127
000361      171 YERR=ERT
000363          IF (NOPT(4,1).GT.0) RETURN
000365      172 PRINT 145,ITER,YERR,STEP,KRES
000401          PRINT 115,(X(I),I=1,N)
000414          RETURN
000415          END
```

Flow Chart for Steepest Descent Subroutine OPT6



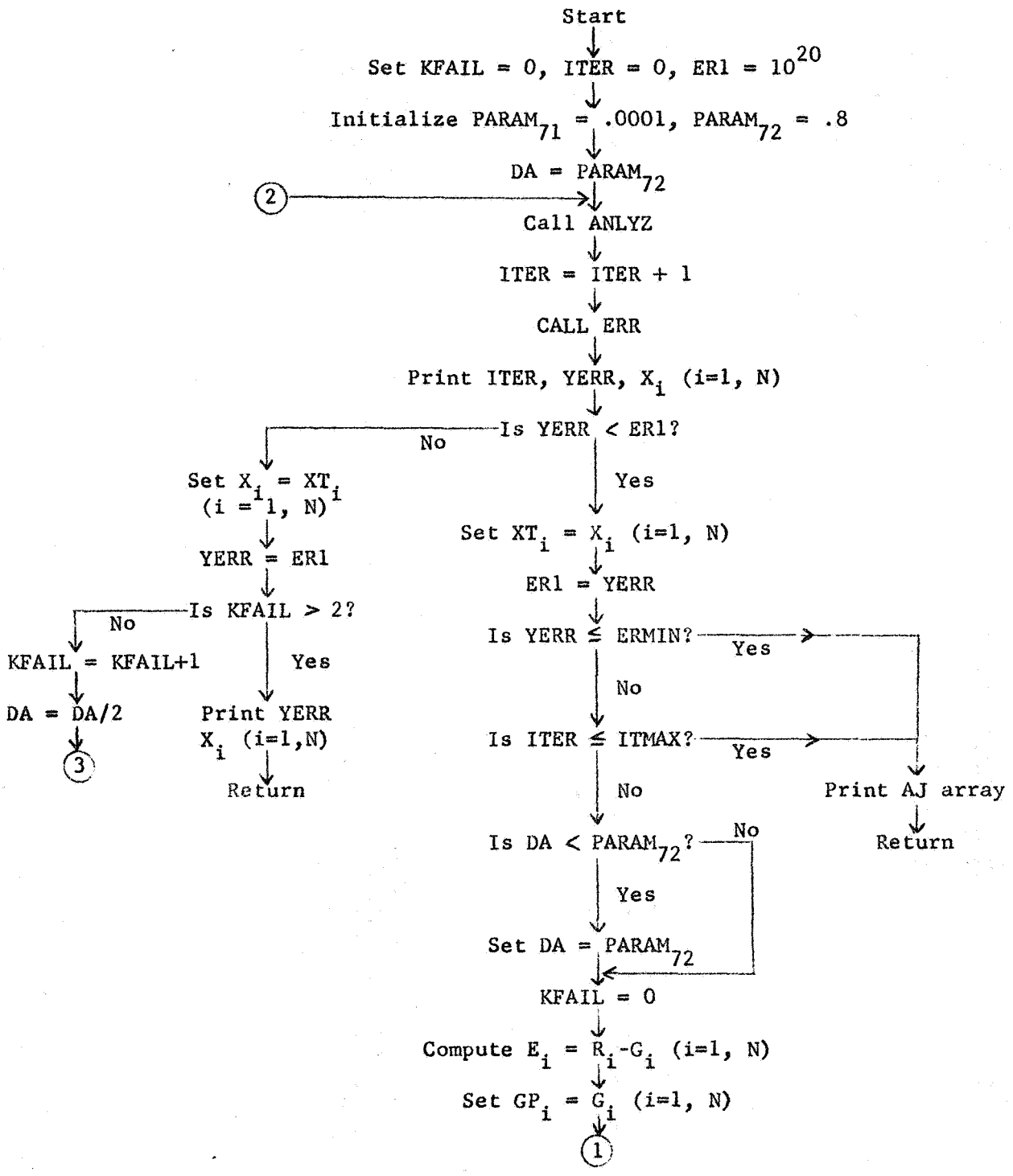
Flow Chart for OPT6 (page 2)

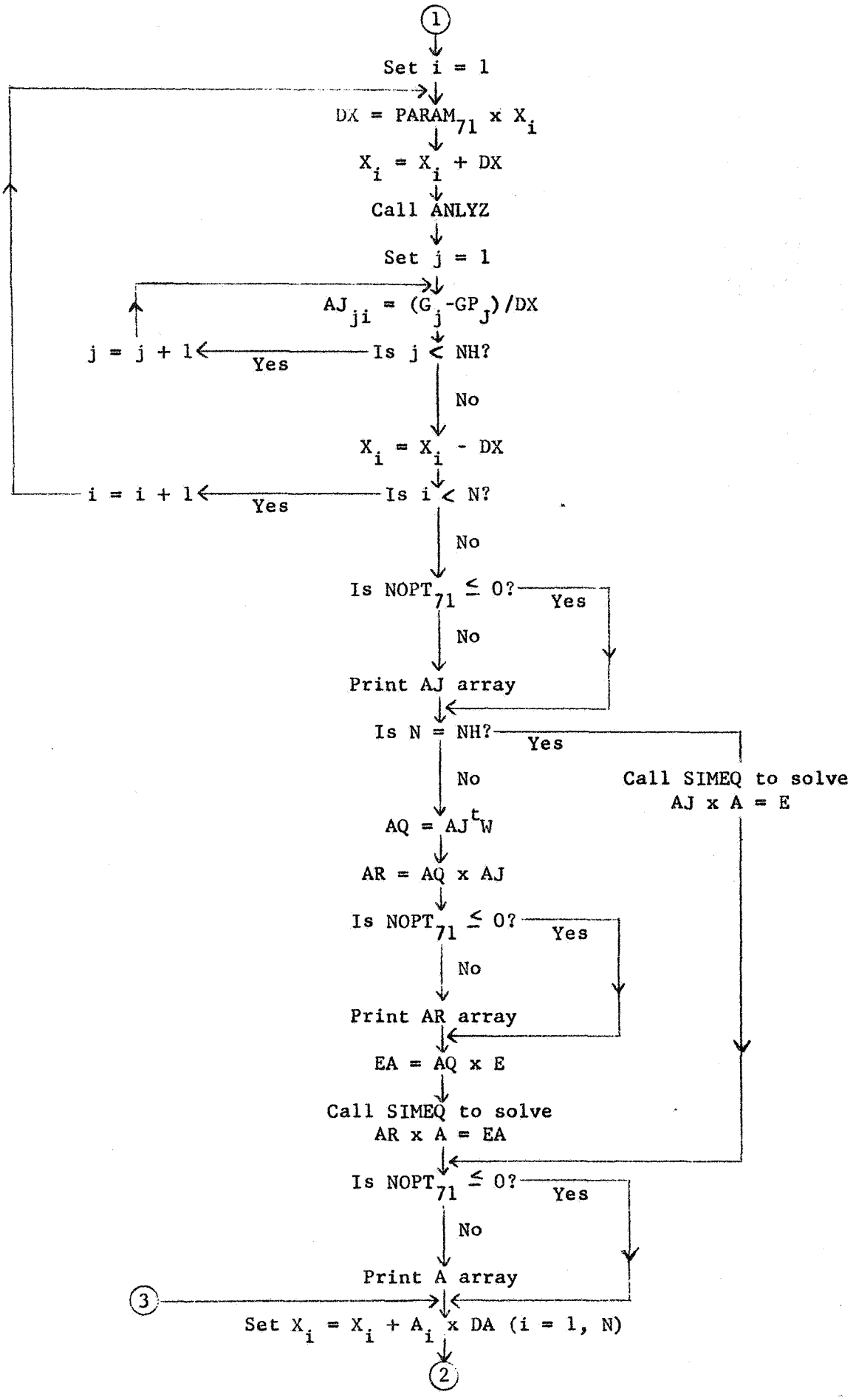


```
      SUBROUTINE OPT6
C     STEEPEST DESCENT OPTIMIZATION SUBROUTINE
C     FOR CDC 6400, NUCLSMAG, JULY 1968
      COMMON /OPT/X(20),XL(20),XU(20),DX(20),XM(20),H(20),R(20),G(20)
     1,G(20),PARAM(10,7),NOPT(10,10),N,MM,YERR,IER,ERR,LL,ITMAX,ALFA(6)
      DIMENSION DER(20)
      IF(PARAM(6,1).EQ.0.) PARAM(6,1)=1.E-04
      IF(PARAM(6,2).EQ.0.) PARAM(6,2)=.1
      IF(PARAM(6,3).EQ.0.) PARAM(6,3)=1.E-04
      IF(PARAM(6,4).EQ.0.) PARAM(6,4)=.8
      IF(PARAM(6,5).EQ.0.) PARAM(6,5)=1.25
      IF (PARAM(6,6).EQ.0) PARAM(6,6)=20.
      PRINT 180
  180 FORMAT (//,1X,*STEEPEST DESCENT SUBROUTINE OPT6 HAS BEEN CALLED*/)
      PRINT 182, (PARAM(6,I),I=1,6),(NOPT(6,I),I=1,5)
  182 FORMAT(
     11X,*PARAM(6,1)=PERTURBATION SIZE =*E9.2/
     21X,*PARAM(6,2)=INITIAL STEP SIZE =*E9.2/
     31X,*PARAM(6,3)=MINIMUM STEP SIZE =*E9.2/
     41X,*PARAM(6,4)=FACTOR FOR DECREASING STEP SIZE =*F9.2/
     51X,*PARAM(6,5)=FACTOR FOR INCREASING STEP SIZE =*E9.2/
     51X,*PARAM(6,6)=PRINTOUT EVERY NTH STEP,N=*E9.2/
     61X,*NOPT(6,1)=NORMALIZE DERIVATIVES (1=YES, 0=NO).............*I2/
     /1X,*NOPT(6,2)=COMPUTE DERIVATIVES FROM VALUE (0), RANGE (1)....*I2/
     61X,*NOPT(6,3)=PRINT EVERY NTH STEP (1), SUCCESS (0), ALL (2)...*I2/
     91X,*NOPT(6,4)=STEP SIZE ACCELERATION OPTION (0=YES, 1=NO)......*I2/
     11X,*NOPT(6,5)=ONE DIMENSIONAL SEARCH OPTION (1=YES, 0=NO)......*I2/)
      STEP=PARAM(6,2)
      DO 183 I=1,N
      XP(I)=X(I)
  183 CONTINUE
      IIER=0
      KK=PARAM(6,6)
      CALL ANLYZ
      CALL ERR
      PRINT 100,YERR
      ER1=YERR
  100 FORMAT (1X,*INITIAL ERROR =*E11.4/)
  102 KA=0
      SUM=0.
      DO 120 I=1,N
      IF (NOPT(6,2).LE.0) GO TO 103
      DXX=(XU(I)-XL(I))*PARAM(6,1)
      GO TO 105
  103 IF (ABS(X(I)).LT.0.01) GO TO 104
      DXX=X(I)*PARAM(6,1)
      GO TO 105
  104 DXX=PARAM(6,1)
  105 X(I)=X(I)+DXX
      CALL ANLYZ
      CALL ERR
      X(I)=X(I)-DXX
      DER(I)=YERR-ER1
      IF (NOPT(6,1).LE.0) DER(I)=DER(I)/DXX
      SUM=SUM+DER(I)**2
  120 CONTINUE
      SUM=SQRT(SUM)
```

```
000140          DO 121 I=1,N
000145          DER(I)=DER(I)/SUM
000150      121 CONTINUE
000152      125 ITER=ITER+1
                DO 150 I=1,N
000155          X(I)=X(I)-DER(I)*STEP
000161          IF(X(I).LT.XL(I)) X(I)=XL(I)
000167          IF(X(I).GT.XU(I)) X(I)=XU(I)
000174      150 CONTINUE
000201          CALL ARLY2
000202          CALL ERR
000203          IF (NOPT(6,3).LE.0) GO TO 146
000205          IF (NOPT(6,4).LE.0) GO TO 134
000206          IF (NOPT(6,4).EQ.2) GO TO 134
000210          IF (ITER/KK-(ITER-1)/KK) 146,134,134
000217      134 PRINT 135,ITER,YERR,STEP
000231      135 FORMAT (1H0,*ITERATION=*I4,5X,*ERROR=*E9.2,5X*STEP SIZE=*E9.2)
000231          PRINT 140,(X(I),I=1,N)
000244      140 FORMAT (5X,1HX,2X,5E10.2)
000244          PRINT 145,(DER(I),I=1,N)
000257      145 FORMAT (5X,5HDER,5E10.2)
000257      146 IF (YERR.LT.ER1*.999) GO TO 155
000263          DO 150 I=1,N
000264          X(I)=XP(I)
000267      150 CONTINUE
000271          IF (ITER.GE.ITMAX) GO TO 170
000273          IF (KA.LE.0) GO TO 152
000275          IF (NOPT(6,5).GT.0) GO TO 102
000277      152 IF (STEP.LT.PARAM(6,3)) GO TO 170
000302          STEP=STEP*PARAM(6,4)
000303          IF (NOPT(6,4).GT.0) GO TO 125
000305          NOPT(6,4)=1
000305          PRINT 151,ITER
000313      151 FORMAT (1H0*ITERATION=I4,5X*ACCELERATION STOPPED*/)
000313          GO TO 125
000314      155 ER1=YERR
000316          DO 156 I=1,N
000317          XP(I)=X(I)
000322      156 CONTINUE
000324          IF(NOPT(6,3).GT.0) GO TO 159
000326          PRINT 135,ITER,YERR,STEP
000337          PRINT 140,(X(I),I=1,N)
000352          PRINT 145,(DER(I),I=1,N)
000365      159 KA=1
000366          IF (YERR.LT.ERMIN) GO TO 170
000371      160 IF (ITER.GE.ITMAX) GO TO 170
000374          IF (NOPT(6,4).LE.0) STEP=STEP*PARAM(6,5)
000377          IF (NOPT(6,5).GT.0) GO TO 125
000402          GO TO 102
000402      170 IF (NOPT(6,3).GT.0) GO TO 174
000405          IF (KA.EQ.1) GO TO 175
000407      173 PRINT 135,ITER,YERR,STEP
000421          PRINT 140,(X(I),I=1,N)
000434          PRINT 145,(DER(I),I=1,N)
000447      175 YERR=ER1
000451          RETURN
000451      174 IF (ITER/KK-(ITER-1)/KK) 173,173,175
000461          END
```

Flow Chart for Subroutine OPT7 - Newton Raphson

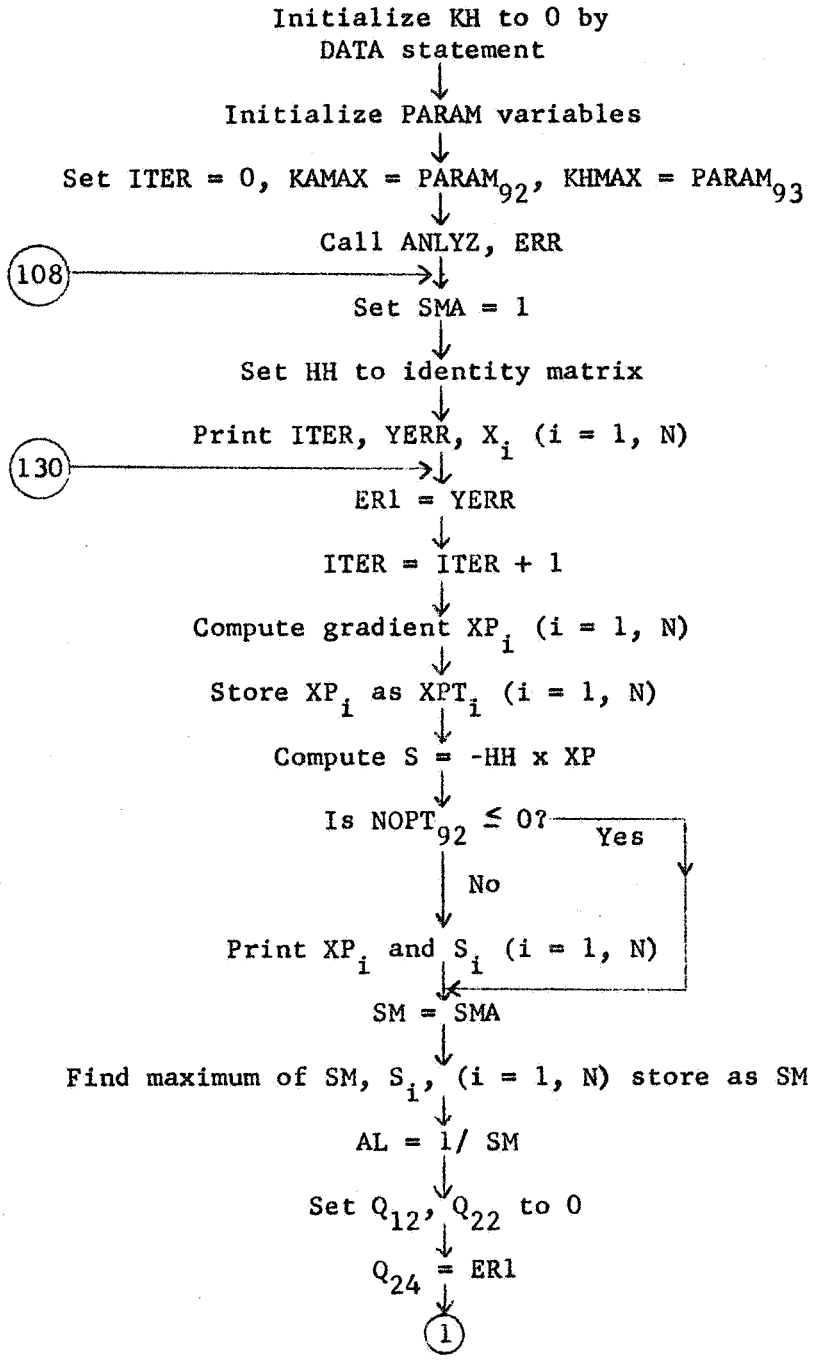



1
Set i = 1
DX = PARAM71 x Xi
Xi = Xi + DX
Call ANLYZ
Set j = 1
AJji = (Gj-GPJ)/DX
Is j < NH?
Yes
j = j + 1
No
Xi = Xi - DX
Is i < N?
Yes
i = i + 1
No
Is NOPT71 ≤ 0?
Yes
No
Print AJ array
Is N = NH?
Yes
Call SIMEQ to solve
AJ x A = E
No
AQ = AJtW
AR = AQ x AJ
Is NOPT71 ≤ 0?
Yes
No
Print AR array
EA = AQ x E
Call SIMEQ to solve
AR x A = EA
Is NOPT71 ≤ 0?
Yes
No
Print A array
3
Set Xi = Xi + Ai x DA (i = 1, N)
2

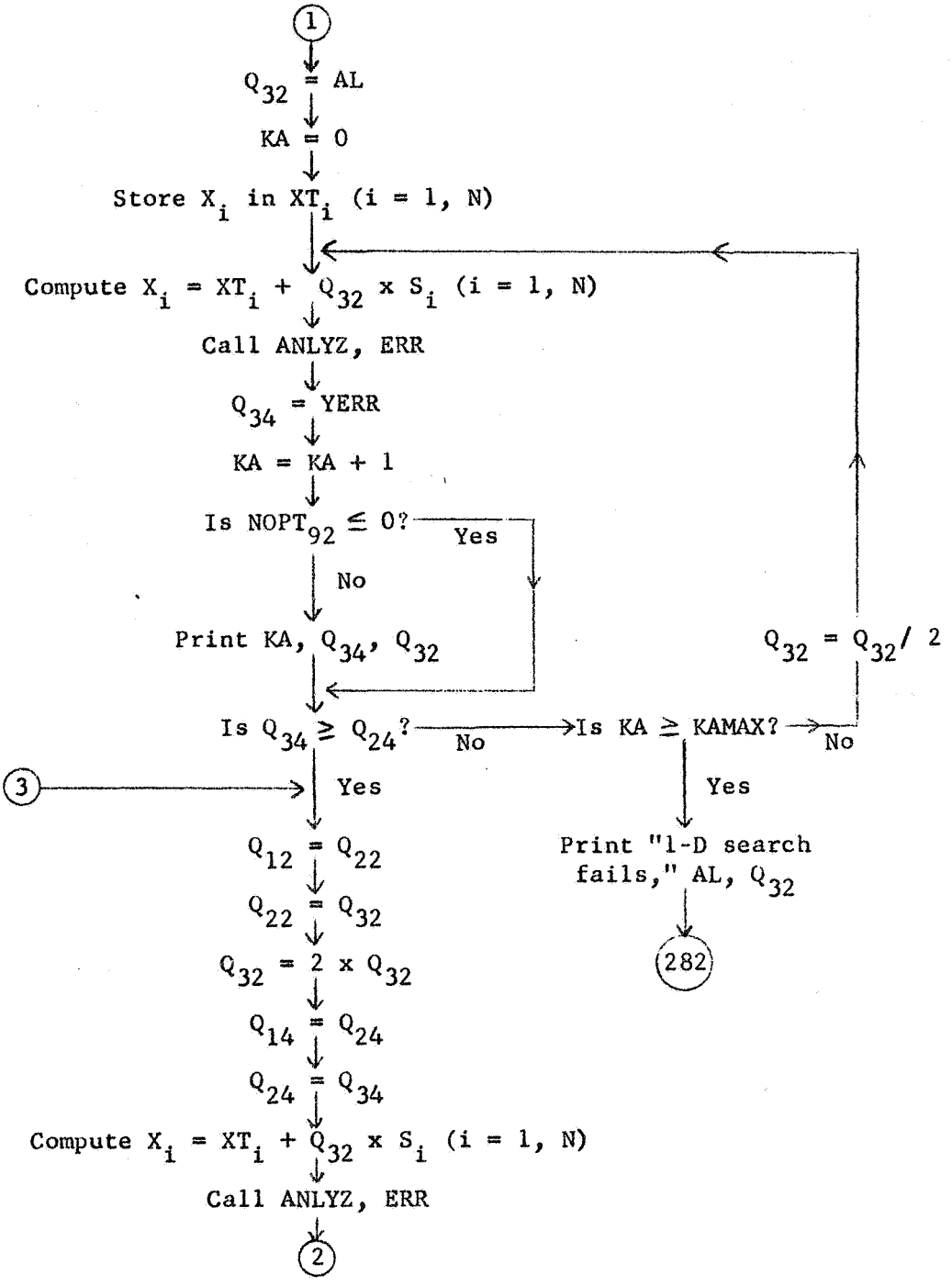
```
      SUBROUTINE OPT7
C     REQUIRES SUBROUTINES MXLMN,MXNNI,STMEQ
C     NUMBER OF REQUIREMENTS,NH,MUST BE GREATER THAN ONE
000002      COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
           1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
000002      DIMENSION E(10),A(10),GP(10),AJ(10,10),AQ(10,10)
000002      DIMENSION AR(10,10),EA(10),XT(20)
000002      ITER=0
000003      KFAIL=0
000004      ER1=1.E+20
000005      IF (PARAM(7,1).EQ.0.) PARAM(7,1)=.0001
000007      IF (PARAM(7,2).EQ.0.) PARAM(7,2)=0.8
000011      PRINT 170
000015  170 FORMAT (//1X,*NEWTON RAPHSON OPT7 CALLED*/)
000015      PRINT 172,PARAM(7,1),PARAM(7,2),NOPT(7,1)
000027  172 FORMAT (
           11X*PARAM(7,1)-PERTURBATION FACTOR FOR DERIVATIVES =*E9.2/
           21X*PARAM(7,2)-FACTOR USED IN CHANGING VARIABLES =*E9.2/
           31X*NOPT(7,1)-NORMAL PRINTOUT (0), EXTENDED PRINTOUT (1)....*T2/)
000027      DA=PARAM(7,2)
000031  112 CALL ANLYZ
000032      ITER=ITER+1
000034      CALL ERR
000035      PRINT 115,ITER,YERR
000045  115 FORMAT (1H0*ITERATION*I4,5X*ERROR=*E10.3)
000045      PRINT 122,(X(I),I=1,N)
000060      IF (YERR.LT.ER1) GO TO 118
000063      DO 116 I=1,N
000064  116 X(I)=XT(I)
000071      YERR=ER1
000073      PRINT 117
000076  117 FORMAT (1H0*OPT7 NEWTON-RAPHSON DOES NOT CONVERGE* )
000076      IF (KFAIL.GT.2) GO TO 123
000102      KFAIL=KFAIL+1
000103      DA=DA/2.
000104      PRINT 124,DA
000112  124 FORMAT (1X*PARAM(7,2) REDUCED TO *E11.3/)
000112      GO TO 141
000113  123 PRINT 121, YERR,ITER
000123  121 FORMAT (1H0*FINAL ERROR=*E10.3,5X,*ITERATION NUMBER*I4)
000123      PRINT 122, (X(I),I=1,N)
000136  122 FORMAT (1X,4HX(I),3X,5E11.3)
000136      RETURN
000137  118 DO 119 I=1,N
000141  119 XT(I)=X(I)
000146      ER1=YERR
000150      IF (YERR.LE.ERMIN) GO TO 150
000152      IF (ITER.GE.ITMAX) GO TO 150
000154      IF (DA.LT.PARAM(7,2)) DA=PARAM(7,2)
000160      KFAIL=0
000161      DO 120 I=1,NH
000163      E(I)=R(I)-G(I)
000167  120 GP(I)=G(I)
000174      DO 130 I=1,N
000175      DX=X(I)*PARAM(7,1)
000200      X(I)=X(I)+DX
000203      CALL ANLYZ
```

```
000204          DO 125 J=1,NH
000206      125 AJ(J,I)=(G(J)-GP(J))/DX
000222      130 X(I)=X(I)-DX
000227          IF (NOPT(7,1).LE.0) GO TO 133
000231          PRINT 155
000234          DO 131 I=1,NH
000236          PRINT 165, (AJ(I,J),J=1,N)
000252      131 CONTINUE
000255      133 IF(N.EQ.NH) GO TO 143
000257          DO 135 I=1,NH
000261          DO 135 J=1,N
000262      135 AQ(J,I)=AJ(I,J)*W(I)
000300          CALL MXLMN(AQ,AJ,N,NH,N,AR)
000304          IF (NOPT(7,1).LE.0) GO TO 138
000306          PRINT 136
000311      136 FORMAT (1X ,*MATRIX AJ(T) X W X AJ* )
000311          DO 137 I=1,N
000313          PRINT 165, (AR(I,J),J=1,N)
000327      137 CONTINUE
000332      138 CALL MXMNI (AQ,F,N,NH,EA)
000336          CALL SIMEQ (AR,N,EA,A)
000341          GO TO 144
000342      143 CALL SIMEQ (AJ,N,F,A)
000345      144 IF (NOPT(7,1).LE.0) GO TO 141
000347          PRINT 142
000352      142 FORMAT (1X,*CHANGE MATRIX A*)
000352          PRINT 165,(A(I),I=1,N)
000365          PRINT 139
000371      139 FORMAT ()
000371      141 DO 140 I=1,N
000373      140 X(I)=X(I)+A(I)*DA
000401          GO TO 112
000402      150 PRINT 155
000406      155 FORMAT (  1H0,*JACOBIAN MATRIX* )
000406          DO 160 I=1,NH
000410          PRINT 165,(AJ(I,J),J=1,N)
000424      160 CONTINUE
000427      165 FORMAT (7E10.2)
000427          PRINT 175
000432      175 FORMAT (1H0,*EXIT FROM OPT7*///)
000432          RETURN
000433          END
```

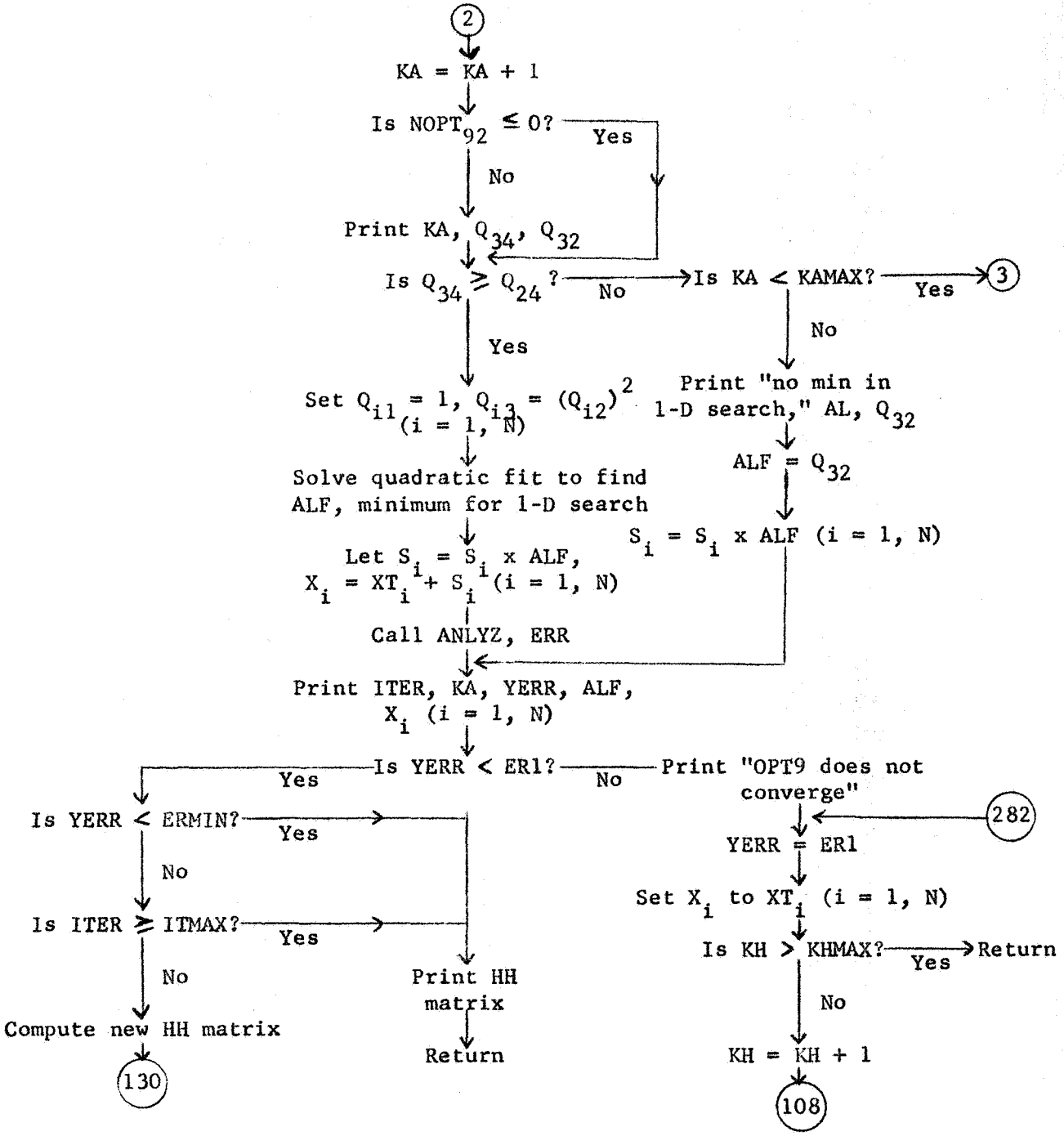
Flow Chart for OPT9 Fletcher Powell Subroutine OPT9



Flow Chart for Subroutine OPT 9 (page 2)



Flow Chart for Subroutine OPT9 (page 3)



```
                SUBROUTINE OPT9
000002          COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
               1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
00002           DIMENSION HH(10,10),XPT(10),S(10),YP(10),AFP(10,10),YPH(10),
               1BFP(10,10),Q(3,4),XT(10)
000002          DATA KH/0/
00002           PRINT 110
00006      110  FORMAT (///1X*OPT9 FLETCHER POWELL HAS BEEN CALLED*/)
000006          ITER=0
000007          IF (PARAM(9,1).LE.0.) PARAM(9,1)=1.E-06
00012           IF (PARAM(9,2).LE.0.) PARAM(9,2)=10.
000015          IF (PARAM(9,3).LE.0.) PARAM(9,3)=3.
000020          PRINT 120,(PARAM(9,I),I=1,3),(NOPT(9,I),I=1,2)
00042      120  FORMAT (
               11X*PARAM(9,1)-PERTURBATION SIZE FOR FINDING GRADIENT =*E9.2/
               21X*PARAM(9,2)-MAX ITERATIONS IN ONE DIMENSIONAL SEARCH =*E9.2/
               21X*PARAM(9,3)-NUMBER OF RESET CYCLES PERMITTED=*E9.2/
               31X*NOPT(9,1)-FIND GRADIENT BY (0) PERTURBATION, (1) ANLYD....*I2/
               41X*NOPT(9,2)-PRINTOUT OPTION, (0) REDUCED, (1) 1-D, (2) ABH..*I2/)
000042          KAMAX=PARAM(9,2)
00044           KHMAX=PARAM(9,3)
000045          CALL ANLYZ
000046          CALL ERR
00047      108  SMA=1.
00051           DO 117 I=1,N
000052          DO 115 J=1,N
00053      115  HH(I,J)=0.
00062      117  HH(I,I)=1.
00067           PRINT 125, ITER,YERR
000077     125  FORMAT (1H0*ITERATION*I4,12X*ERROR=*E10.3)
00077           PRINT 126,(X(I),I=1,N)
000112     126  FORMAT (1X*X(I)*2X,5E11.3)
000112     130  ER1=YERR
00114           ITER=ITER+1
00115           IF (NOPT(9,1)) 135,135,140
000117     135  CALL GRAD
00120           GO TO 145
00121      140  CALL ANLYD
000122     145  DO 170 I=1,N
000124     170  XPT(I)=XP(I)
00131           CALL MXMN1 (HH,XP,N,N,S)
00135           DO 175 I=1,N
000137     175  S(I)=-S(I)
00144           IF (NOPT(9,2).LE.0) GO TO 185
00145           PRINT 160,(XP(I),I=1,N)
000160     160  FORMAT (5H GRAD,2X,5E11.3)
00160           PRINT 180, (S(I),I=1,N)
00173      180  FORMAT (2H S,5X,5E11.3)
000173     185  SM=SMA
000175          DO 190 I=1,N
00176           ASM=ABS(S(I))
00201           IF (ASM.GT.SM) SM=ASM
000204     190  CONTINUE
00207           AL=1./SM
000211          Q(1,2)=0.
000212          Q(2,2)=0.
000213          Q(2,4)=ER1
```

```
000214          Q(3,2)=AL
000216          KA=0
000217          DO 210 I=1,N
000220      210 XT(I)=X(I)
000225      212 DO 215 I=1,N
000227      215 X(I)=XT(I)+Q(3,2)*S(I)
000240          CALL ANLYZ
000241          CALL ERR
000242          Q(3,4)=YERR
000244          KA=KA+1
000245          IF (NOPT(9,2).LE.0) GO TO 245
000247          PRINT 240,KA,Q(3,4),Q(3,2)
000260      240 FORMAT (1X,*1-D SEARCH, CYCLE*I3,5X*ERROR=*E10.3,5X*ALFA=*E10.3)
000260      245 IF(Q(3,4).LT.Q(2,4)) GO TO 248
000263          IF(KA.GE.KAMAX) GO TO 246
000265          Q(3,2)=Q(3,2)/2.
000267          GO TO 212
000267      246 PRINT 247,AL,Q(3,2)
000277      247 FORMAT (1H0*1-D SEARCH FAILS, STEP CHANGED FROM*E10.3* TO*E10.3/)
000277          GO TO 282
000300      248 Q(1,2)=Q(2,2)
000302          Q(2,2)=Q(3,2)
000303          Q(3,2)=Q(3,2)*2.
000304          Q(1,4)=Q(2,4)
000306          Q(2,4)=Q(3,4)
000307          DO 250  I=1,N
000310      250 X(I)=XT(I)+Q(3,2)*S(I)
000321          CALL ANLYZ
000322          CALL ERR
000323          Q(3,4)=YERR
000325          KA=KA+1
000326          IF (NOPT(9,2).LE.0) GO TO 255
000330          PRINT 240,KA,Q(3,4),Q(3,2)
000341      255 IF (Q(3,4).GE.Q(2,4))GO TO 265
000344          IF (KA.LT.KAMAX) GO TO 248
000346          PRINT 260,AL,Q(3,2)
000356      260 FORMAT (1H0*NO MIN FROM 1-D SEARCH, STEP CHANGED FROM*E10.3* TO*
                1E10.3/)
000356          ALF=Q(3,2)
000360          DO 262 I=1,N
000361      262 S(I)=S(I)*ALF
000366          GO TO 275
000366      265 DO 270 I=1,3
000370          Q(I,1)=1.
000372      270 Q(I,3)=Q(I,2)**2
000375          DO 271 J=2,4
000377          DO 271 K=2,3
000400      271 Q(K,J)=Q(K,J)-Q(1,J)
000413          DO 272 I=3,4
000414      272 Q(3,I)=Q(3,I)-Q(2,I)*Q(3,2)/Q(2,2)
000426          ALF=(-Q(3,3)*Q(2,4)+Q(2,3)*Q(3,4))/(2.*Q(2,2)*Q(3,4))
000435          DO 273 I=1,N
000436          S(I)=S(I)*ALF
000441      273 X(I)=XT(I)+S(I)
000447          CALL ANLYZ
000450          CALL ERR
000451      275 PRINT 276,ITER,KA,YERR,ALF
000465      276 FORMAT (1H0*ITERATION*I4,4X,3HKA=,I2,3X*ERROR=*E10.3,5X*ALFA=*
```

```
     1(10,3)
000465       PRINT 126,(X(I),I=1,N)
000500       IF (YERR.LT.ER1) GO TO 285
000503       PRINT 281
000506   281 FORMAT (1H0*OPT9 DOES NOT CONVERGE*/)
000506   282 YERR=ER1
000510       DO 280 I=1,N
000511   280 X(I)=XT(I)
000516       IF (KH.GE.KHMAX) RETURN
000522       KH=KH+1
000524       PRINT 292,KH
000531   292 FORMAT (1X*RESET H TO IDENTITY MATRIX, CYCLE*I3/)
000531       GO TO 10B
000532   285 IF (YERR.LT.ERMIN) GO TO 370
000535       IF (ITER.GE.ITMAX) GO TO 370
000537       IF (NOPT(9,1)) 290,290,295
000541   290 CALL GRAD
000542       GO TO 300
000543   295 CALL ANLYD
000544   300 DO 305 I=1,N
000546   305 YP(I)=XP(I)-XP1(I)
000555       APD=0.
000556       DO 310 I=1,N
000557   310 APD=APD+S(I)*YP(I)
000566       DO 315 I=1,N
000567       DO 315 J=1,N
000570   315 AFP(I,J)=S(I)*S(J)/APD
000605       CALL MXMN1 (HH,YP,N,N,YPH)
000611       BPD=0.
000612       DO 320 I=1,N
000614   320 BPD=BPD+YP(I)*YPH(I)
000623       DO 325 I=1,N
000624       DO 325 J=1,N
000625       BFP(I,J)=-YPH(J)*YPH(I)/BPD
000634   325 HH(I,J)=HH(I,J)+AFP(I,J)+BFP(I,J)
000647       IF (NOPT(9,2).LE.1) GO TO 130
000651       PRINT 326,APD
000657   326 FORMAT (1H0*DENOMINATOR FACTOR FOR A MATRIX=*E9.2)
000657       PRINT 330
000663   330 FORMAT (1X*A MATRIX*)
000663       DO 335 I=1,N
000665       PRINT 340, (AFP(I,J),J=1,N)
000701   340 FORMAT (7E10.2)
000701   335 CONTINUE
000704       PRINT 336,BPD
000711   336 FORMAT (1H0*DENOMINATOR FACTOR FOR B MATRIX=*E9.2)
000711       PRINT 345
000715   345 FORMAT (1X*B MATRIX*)
000715       DO 350 I=1,N
000717       PRINT 340,(BFP(I,J),J=1,N)
000733   350 CONTINUE
000736       PRINT 355
000741   355 FORMAT (1H0*H MATRIX*)
000741       DO 360 I=1,N
000743       PRINT 340,(HH(I,J),J=1,N)
000757   360 CONTINUE
000762       GO TO 130
000762   370 PRINT 355
```

```
000760          DO 372 I=1,N
000770          PRINT 340, (HH(I,J),J=1,N)
001004      372 CONTINUE
001007          RETURN
001007          END
```

```
                SUBROUTINE GRAD
      C         COMPUTES GRADIENT, MUST BE PRECEDED BY A CALL FOR
      C         SUBROUTINES ANALYZ AND ERR
000002            COMMON /OPT/X(20),XL(20),XU(20),KX(20),XP(20),H(20),R(20),W(20)
                 1,G(20),PARAM(10,7),NOPT(10,10),N,NH,YERR,ITER,ERMIN,ITMAX,ALFA(8)
000002            ER1=YERR
000004            DO 150 I=1,N
000005            DX=X(I)*PARAM(9,1)
000010            X(I)=X(I)+DX
000013            CALL ANLYZ
000014            CALL ERR
000015            XP(I)=(YERR-ER1)/DX
000021        150 X(I)=X(I)-DX
000025            YERR=ER1
000026            RETURN
000026            END
```