AN ABSTRACT OF THE THESIS OF

PAUL WAYNE CULLOP, Jr for the MASTER OF SCIENCE
(Name) (Degree)

in MATHEMATICS presented on June 5, 1972
(Major) (Date)

Title: AUTOMATED RIGOROUS SOLUTIONS TO NONLINEAR EQUATIONS

Redacted for privacy

Abstract approved: _____
Joel Davis

We will consider the implementation of a computer program to solve a nonlinear algebraic system of N equations and unknowns. The program involves the use of a parameter, Newton's method, and an automatic change of parameter. Also considered are rigorous error bounds for the answer. The program was implemented and some numerical results are given.

Automated Rigorous Solutions to Nonlinear Equations

by

Paul Wayne Cullop, Jr.

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1973

APPROVED:

Redacted for privacy

Associate Professor of Mathematics

in charge of major

Redacted for privacy

Acting Chairman of Department of Mathematics

Redacted for privacy

Dean of Graduate School

Date thesis is presented June 5, 1972

Typed by Clover Redfern for Paul Wayne Cullop, Jr.

TABLE OF CONTENTS

AUTOMATED RIGOROUS SOLUTIONS TO NONLINEAR EQUATIONS

I. INTRODUCTION

The purpose of this paper is to describe the implementation of a package to solve nonlinear equations. This paper is based upon a technical report (No. 25) written by Joel Davis [1]. The report, entitled, "The Solution of Nonlinear Operator Equations with Critical Points," deals with all of the theoretical aspects of this approach to the problem of nonlinear equations.

The approach is to insert a parameter in the system of equations. The parameter may be inserted, such that for some value of the parameter, an initial guess may easily be obtained. For example:

$$x^2 - xz + y - 8 = 0$$

$$z^3 - 2x + \frac{1}{2}y + 4 = 0$$

$$-y^2 + xz + yz - 1 = 0$$

may become

$$x^2 - \alpha xz + y - 8 = 0$$

$$\alpha z^3 - 2x + \frac{1}{2}y + 4 = 0$$

$$-\alpha y^2 + xz + yz - 1 = 0$$

where α is the inserted parameter. Also such a parameter may be inherited from a problem that describes some natural phenomena. The

parameter might represent pressure or resistance. The basic way of proceeding is to solve the system by Newton's method, using some initial guess. Then the parameter is varied and the previous answer is improved for the use as an initial guess for the new value of the parameter. This process is continued until all of the desired results are obtained. It will be shown that the above process fails when a critical point is being approached. A discussion on an automatic change of parameter will be given in the hopes that it may enable the program to continue past the critical point.

Only the basic theory is given here. The reader is referred to [1] for a much more detailed explanation. The theory given includes Newton's method and prediction. A discussion is made on the handling of critical points. Finally there is a discussion on putting some type of an error bound in the answers found.

The main emphasis is the implementation of the theory. There will also be a discussion of some of the problems that arose in the implementation. Finally there will be several chapters dealing with the use of the package and its subroutines.

The examples given are from the equations of a shallow clamped symmetrical cap under uniform pressure problem (see [1]). The answer has six components of which only three are listed in the examples. The program was written in FORTRAN for a CDC 3300 computer.

II. THEORY FOR NEWTON'S METHOD WITH A PARAMETER

Let R be the set of real numbers, R^N be the set of n-tuples $(x_1, x_2, \ldots, x_N)$ where $x_i \in R$, $i = 1, 2, \ldots, N$, and F a mapping from R^N into R^N. Suppose one is asked to solve $F(x) = 0$, a nonlinear algebraic system of N equations and unknowns. Define $F(a, x)$ to be a function of a real parameter a and a vector $x \in R^N$. Suppose further, one knows the answer to $F(a, x) = 0$ for some a, say $a = a_0$. Is there a way of finding a solution to $F(a_k, x) = 0$ if a solution to $F(a_0, x) = 0$ is known? The answer is maybe! By using this parameter and Newton's Method, one might be able to proceed from a solution to $F(a_i, x) = 0$ to a solution to $F(a_{i+1}, x) = 0$, where $0 \leq i \leq k-1$.

The derivation of Newton's Method given here is for a fixed a. Throughout the text, the norms used will be the maximum norms. If $x \in R^N$, then

$$\|x\| = \max_i |x_i| \qquad 1 \leq i \leq N \tag{2.1}$$

If $J: R^N \rightarrow R^N$ is linear then

$$\|J\| = \max_i \sum_{k=1}^{N} |J_{ik}| \qquad 1 \leq i \leq N \tag{2.2}$$

A solution pair, (a, x) or $(a, x(a))$, will be an a and a

corresponding x such that $F(a, x) = 0$. The derivatives used are Fréchet derivatives.

Let F be a real valued function of $a \in R$ and $x \in R^N$ such that $F(a, x) = 0$ for some $x \in R^N$. Assume $F(a, x)$ is continuously differentiable. Taylor's series yields

$$F(a, \hat{x}) = F(a, x) + F_x(a, x)(\hat{x}-x) + \ldots \tag{2.3}$$

Since $F(a, \hat{x}) = 0$, we have the following approximation

$$0 \approx F(a, x) + F_x(a, x)(\hat{x}-x) \tag{2.4}$$

Assuming that $[F_x(a, x)]^{-1}$ exists and treating (2.4) as an equation we find

$$\bar{x} = x - [F_x(a, x)]^{-1}F(a, x) \tag{2.5}$$

It is now hoped that $\bar{x}$ is a better approximation to $\hat{x}$ than x was. The process is repeated until the method apparently converges or until it diverges. There are sufficient conditions, which will be given in Chapter VII, that will guarantee convergence. It should be also noted that in practice $[F_x(a, x)]^{-1}$ is not found. Instead, the following system of linear equations is solved by a method such as Gaussian elimination.

$$F_x(a, x)(\bar{x}-x) = -F(a, x) \tag{2.6}$$

The vector $(\bar{x}-x)$ is then added to x to obtain $\bar{x}$.

After an answer is obtained for one value of a, we wish to change a and obtain a new answer. The old answer could be used as an initial guess to the new problem. But it is possible to do better. The prediction of the initial guess to the new problem also uses Taylor's series considering x as a function of a. Assuming x is continuously differentiable, Taylor's series gives

$$x(a_{i+1}) = x(a_i) + x'(a_i)(a_{i+1}-a_i) + \ldots \tag{2.7}$$

or

$$x(a_{i+1}) \approx x(a_i) + x'(a_i)(a_{i+1}-a_i) \tag{2.8}$$

Differentiating $F(a_i, x) = 0$ with respect to a, we get

$$F_x(a_i, x)x'(a_i) + F_a(a_i, x) = 0 \tag{2.9}$$

Solving (2.9) we obtain $x'(a_i)$ and by adding $x'(a_i)(a_{i+1}-a_i)$ to $x(a_i)$ we attain a possibly more accurate initial guess to the solution of $F(a_{i+1}, x) = 0$ than we get by just using the solution to $F(a_i,x) = 0$ as an initial guess.

For a particular a_i, the initial guess is refined by using Newton's method. After the guess is refined, a new problem is created by changing the parameter, a, slightly. A new initial guess is predicted. The process is repeated.

If the answer to $F(a_0, x) = 0$ is known and the answer to $F(x) = 0$ is desired, one may be able to find a solution to $F(x) = 0$ by starting with $F(a_0, x) = 0$ and advancing a through a sequence of intermediate values to a_k, where $F(a_k, x) = F(x)$.

III. PROGRAM FOR NEWTON'S METHOD WITH A PARAMETER

By varying the parameter a with various stepsizes and, for each, a solving $F(a, x) = 0$, we hope that we can obtain the solution to $F(a_k, x) = 0$, where $F(a_k, x) = F(x)$.

Chapter X gives a more detailed description of the following FORTRAN subroutines necessary for this part of the program. FUNCEVAL evaluates $F(a, x)$ for a particular a and x. FUNCEVAL returns with the value of the system in a vector F. PTL evaluates the partial derivatives of $F(a, x)$ with respect to x, i.e., $F_x(a, x)$, for a particular a and x. PTL returns with $F_x(a, x)$ in a real matrix called J. APRTL evaluates the derivative of $F(a, x)$ with respect to a, i.e., $F_a(a, x)$ for a particular a and x. APRTL returns with the value in a vector AP. These three subroutines must be written by the user. The arrays F, J, and AP are passed as parameters in the call to their respective subroutines. An additional subroutine, GAUSS, is used that solves a linear system of equations. Among the list of parameters of GAUSS, is a flag IGS, which is set to zero if the system is singular or nearly so, otherwise IGS is set to one and the solution is calculated.

In addition to the above subroutines, the following variables and constants, which the user supplies, are necessary for this part of the program. X is the iterate in the Newton process. The variable A is the value of a currently being used

in the program. The variable AINCR is the stepsize or the amount to be added to A to obtain a new A. The constant AMAX is the maximum stepsize allowed. The constants XI and XD are the factors to increase and decrease, respectively, AINCR. The constant EPSIL is used for convergence criterion based upon the relative difference in successive iterations. ALAST is the last value of A wanted. ELMAX is the maximum (in absolute value) component of the answer the user expects. The vector AW contains an optional list of special alphas for which the user would like answers. Chapter IX tells how these variables and constants are initialized.

The Newton method proceeds as follows. First, $F(a_i, x)$ is calculated. Then on the first or fifth iteration the Jacobian matrix, $F_x(a_i, x)$, is calculated. GAUSS is called to solve $F_x(a_i, x)Z = -F(a_i, x)$ where $Z = x^{i+1} - x^i$. If $F_x(a_i, x)$ is singular or nearly so, the Newton method has failed. This situation will be discussed later. If $\|x^{i+1}-x^i\|$ is less than $\|x^i\| \cdot$EPSIL, convergence is assumed. If $\|x^{i+1}\|$ is greater than 100$\cdot$ELMAX, the iterations are considered to have diverged. If the above tests are not met, the iteration loop is traversed again. If at the end of the sixth loop, the iteration process has not converged, the process is considered to have failed. The importance of six iterations will be discussed in Chapter VI.

If the process has converged, the answer is printed according

to the option selected by the user (see Chapter IX). If A equals ALAST or $\|x\|$ is greater than or equal to ELMAX, then the program terminates. If the number of iterations necessary for convergence is six, then there is no change made in the stepsize and an initial guess is calculated for the next A (see below). Otherwise, the stepsize is increased by a factor of XI, which should be greater than 1 and is commonly 2. This incrementing in the stepsize is suppressed if the initial guess was supplied by the user or the stepsize was decreased in order to obtain the last solution. If ABS(AINCR) is greater than ABS(AMAX), then AINCR is set to ABS(AMAX) times the sign of AINCR. Finally an initial guess is calculated for the next A.

The Newton process can fail in any one of several ways. If the process fails to converge in six steps, while using an initial guess supplied by the user, the program travels the loop six additional times. This is done for two reasons. One, the guess supplied by the user is usually less accurate than that supplied by the program in later steps. Secondly, there is no way to recover from a failure on the initial step. If the program again fails to converge, a message so stating is printed. The user may restart by supplying a new guess, which must correspond to the starting value of A given earlier.

If the iterations failed to converge in six steps using a guess predicted by the program, the following steps are taken. The

stepsize, AINCR, is reduced by a factor of XD, which should be positive and less than 1, commonly 0.5. The values of A and X are restored to the last known solution pair. The Jacobian, which may have been changed by the previous unsuccessful attempt is also restored. Finally, an initial guess is calculated.

Before an initial guess is calculated, a check is made to see if a specified α, say A_s, lies between A and (A+1.1·AINCR). If so, AINCR is assigned the value A_s - A. To calculate an initial guess, $F_\alpha(\alpha_i, x)$ is calculated and the system $F_x(\alpha_i, x)COR = -F_\alpha(\alpha_i, x)$ is solved where COR is equal to $x(\alpha_{i+1}) - x(\alpha_i)$. Note that $F_x(\alpha_i, x)$ is now known to be nonsingular. The vector AINCR·COR is added to X to obtain an initial guess corresponding to a new A, found by adding AINCR to A. The program now returns to the beginning of the Newton process to solve $F(\alpha_{i+1}, x) = 0$.

	A	X(1)	X(3)	-X(5)
1	.0000	.01811	.03293	.07442
2	.1000	.01905	.03375	.07577
3	.3000	.02143	.03571	.07896
4	.7000	.03155	.04297	.08921
5	.8500	.04442	.05170	.09603
6	.8875	.05225	.05751	.09655
7	.9275	.06109	.06532	.09181
8	1.0000	.06780	.07439	.07610

Example 1

Example from elasticity [1], $\mu^2 = 7$

As seen in Example 1, the scheme sometimes works. The stepsize first increased, then decreased and finally increased again. In this example, a basic assumption made in Chapter II that $F_x(\alpha, x)$ always had an inverse, was always valid. This assumption is not always valid as seen in Example 2.

	A	X(1)	X(3)	-X(5)
1	.00000	.01091	.02299	.06459
2	.50000	.01611	.02682	.07539
3	.62500	.01966	.02858	.08127
4	.68750	.02326	.03003	.08646
5	.71875	.02712	.03138	.09138
6	.72656	.02914	.03204	.09374
7	.73047	.03110	.03268	.09591
8	.73145	.03227	.03305	.09715
9	.73157	.03259	.03315	.09748

Example 2

Example from elasticity [1], $\mu^2 = 9.8$

As seen above, the stepsizes are becoming smaller. Apparently $F_x(\alpha, x)$ is becoming singular. The parameter α is approaching a critical point, α_c, and will never be able to go beyond α_c. If, when using the parameter α, the Jacobian becomes singular, we must find a way of changing the parameter so as to make the matrix nonsingular again.

IV. THEORY FOR A NEW PARAMETER

An operator or matrix is singular when its null space has dimension greater than zero. In practice, a singular point, a solution pair for which the Jacobian is singular, is almost never reached. As a matrix is slowly becoming singular, the norm of its inverse becomes larger. This affects the prediction as well as the Newton process. (See last example in the preceding chapter.) This is recognized by the fact, that increasingly smaller stepsizes are necessary for the algorithm to converge. The purpose of this chapter is to describe a method in which $F_x(a, x)$ can be changed so that it is nonsingular and the program can proceed.

A change of parameter will be defined as follows: $a = \rho + V \cdot x$ where $V : R^N \rightarrow R$. Now a has become a function of both ρ and x. Thus

$$F(a, x) = F((\rho + V \cdot x), x) = \overline{F}(\rho, x) \tag{4.1}$$

and

$$\overline{F}_x(\rho, x) = F_a(a, x) \cdot V + F_x(a, x) \tag{4.2}$$

It is hoped that such a change makes $F_x(a, x)$ nonsingular and thus the results of Chapter II may be applied. With $A = F_x(a, x)$ and $B = F_a(a, x)$ the following theorem shows how a change of parameter works. The theorem is from a slightly more general theorem by Davis [1].

Theorem: Let $A: X \rightarrow X$, $B: Z \rightarrow X$ where X and Z are linear spaces of finite dimension. Let nullity A = corank A = 1 and $A(X) + B(Z) = X$. Then there exists a V such that $V: X \rightarrow Z$ and $A + B \cdot V$ is nonsingular.

Proof: Choose Q_1 and Q_2 such that $X = N(A) \oplus Q_1$ and $Z = N(B) \oplus Q_2$. $N(A)$ is the set of x such $x \in X$ and $A(X) = 0$ and similarly for $N(B)$. The direct sum, $\oplus$, is defined as follows. With R, S, and T being linear spaces, $R = S \oplus T$ if and only if $R = S + T$ and $S \cap T = \{0\}$. B restricted to Q_2 is a one-to-one mapping from Q_2 to X. $R(A) + R(B) = X$ implies that there exists Q_3, a subspace of Q_2, such that $X = R(A) \oplus R(Q_3)$. Thus $\dim(Q_3) = \dim N(A) = 1$. There exists a linear mapping V such that $N(V) = Q_1$ and V maps $N(A)$ onto Q_3, with V being continuous. A, restricted to Q, mapping Q_1 to $R(A)$, and $B \cdot V$, restricted to $N(A)$ mapping $N(A)$ to $B(Q_3)$, are both isomorphisms, i.e., they are onto and one-to-one. Thus $A + BV$, mapping X to X, is an isomorphism and thus nonsingular.

It can be shown that V need only be chosen so that $B \cdot V \neq 0$. The general scheme follows. When the stepsize, AINCR, begins to steadily decrease, perhaps $F_x(a, x)$ is becoming singular. If it is, then a change of parameter may be helpful when the stepsize drops

below a certain threshold. In making a change of parameter, a functional, V, is chosen and from that a new parameter, ρ, is calculated.

It should be pointed out that the change of parameter will not work in all cases when the Newton process fails. If $F_a(a, x)$ belongs to the range of $F_x(a, x)$, that is range $(F_x(a, x) \; F_a(a, x))$ does not span the space R^N a change of parameter will not work. Also if the nullity of the space R^N is greater than 1, the method described here will not work. A higher dimensional parameter and functional could be used in the latter case but this is not done here.

As mentioned above, a has become a function of ρ, i.e., $a = \rho + V \cdot x(\rho)$. Therefore $a' = \frac{da}{d\rho} = 1 + V \cdot x'(\rho)$, where $x'(\rho)$ can be found by solving $\overline{F}_x(\rho, x(\rho))\, x'(\rho) = -\overline{F}_\rho(\rho, x(\rho)) = -F_a(a, x)$. An indication of the stepsize for the new parameter is given by

RIN = AINCR/a '.

V. PROGRAM FOR NEWTON'S METHOD WITH A NEW PARAMETER

Besides the subroutines, flags, and constants mentioned in Chapter III, the following are necessary for this part of the program. The flag IFX is used to determine which functional FX is used for the change of parameter. Initially IFX is equal to one. After a functional is calculated, IFX is set equal to -IFX. The vector FX, which will be the functional V in the preceding chapter, is determined as follows. If IFX is equal to one, then

$$FX_i = 1; \qquad i = 1, 2, \ldots, N.$$

If IFX is equal to -1, then

$$FX_i = (-1)^i; \qquad i = 1, 2, \ldots, N.$$

The constant AMIN is the minimum value of the absolute value of the stepsize allowed, i.e., the threshold mentioned previously. The name for the new parameter is RHO, while RIN is its stepsize.

A change of parameter takes place when ABS(AINCR) becomes less than a certain value AMIN. A further test made on AINCR in the case where the iterations failed to converge, was not described in Chapter III. If ABS(AINCR) is equal to ABS(AMIN), then the program switches to the change of parameter (see below). Otherwise, a reduction in the stepsize is made. If the ABS(AINCR) is now less than

the ABS(AMIN), AINCR is set equal to AMIN with the proper sign. The process then continues as described in Chapter III.

Several things are initialized for the change of parameter. FX is initialized according to IFX. The arrays $F_x(a, x)$ and $F_a(a, x)$ are restored using the latest solution pair. Now FX is scaled by a factor of $\| F_x(a, x) \| / (N \cdot \| F_a(a, x) \|)$. Finally, the new parameter, RHO, is now assigned the value $A - X \cdot FX$, where A and X are the latest solution pair.

The Newton iterations for the new parameter are similar to the iterations for the old parameter A, with the following differences. Instead of solving $F_x(a_i, x)Z = -F(a_i, x)$, $\overline{F}_x(\rho_i, x)Z = -\overline{F}(\rho_i, x)$ is solved where $\overline{F}_x(\rho_i, x) = F_a(a_i, x)\ FX + F_x(a_i, x)$. With each iteration, a new value of X is found and since $A = RHO + FX \cdot X$, a new A must be calculated with each change in X.

The events that follow convergence in the new parameter are like those that follow convergence in the old parameter. The program does make an additional test. After the fifth successful iteration, the A parameter is tried.

If the iterations fail when using the last solution pair for A as an initial guess for the RHO parameter, the program checks to see if the other functional has been tried. If not, then it is used and its corresponding RHO is tried as a new parameter. If it has been tried, a message is printed stating that the program can not proceed further.

If the iteration process fails on a guess predicted while using the RHO parameter, RHO, A and X are restored to the last known solution triple, $(\rho, a, x(a))$. If ABS(RIN) is greater than ABS(AMIN), the normal scheme of decreasing the stepsize introduced in Chapter III is used (including the test mentioned in this chapter). If ABS(RIN) is equal to ABS(AMIN) and the total number of successful iterations is one or two, then the other functional is tried, else, the program returns to the old parameter A.

The prediction of the next answer is again like that in Chapter III with a few exceptions. First, no special values of A are allowed when iterating in the new parameter. If RIN has not been previously calculated, then the system $\overline{F}_x(\rho_i, x)$ COR = $-F_a(a_i, x)$ is solved. This must be done anyway for the prediction of the next answer. RIN is assigned AINCR/(1+COR*X) where COR = $x'(\rho_i)$.

Example 3, a continuation of Example 2, illustrates that the change of parameter method can be successful. The change of parameter enabled the program to proceed past two singular points. Note that the sign of the stepsize for A became negative while using the first RHO parameter and became positive again after the second successful change of parameter. The second change of parameter was not successful since only two successful iterations, lines 20 and 21, were found. When the program switched to the other functional, it calculated the wrong sign for RIN. The solution pair given in

line 23 lies between the solution pairs given in lines 9 and 18. How to recognize the fact that the program is finding answers already known, i.e., retracing itself, will be discussed in the next chapter along with other modifications made while writing the program.

	A	X(1)	X(3)	-X(5)	RHO
1	.73145	.03227	.03305	.09715	
2	.73157	.03259	.03315	.09748	
3	.73157	.03259	.03315	.09748	1.5457
4	.73163	.03287	.03325	.09777	1.5455
5	.73163	.03327	.03327	.09818	1.5451
6	.73151	.03383	.03355	.09875	1.5443
7	.73107	.03464	.03382	.09955	1.5426
8	.73107	.03464	.03382	.09955	
9	.73054	.03526	.03402	.10015	
	...	...	...	...	
18	.61270	.08035	.06008	.11335	
19	.61243	.08112	.06104	.11241	
20	.61243	.08112	.06104	.11241	.77497
21	.61241	.08153	.06160	.11182	.77487
22	.61241	.08153	.06160	.11182	.69391
23	.61313	.07972	.05928	.11408	.73022
24	.61241	.08153	.06160	.11182	.69391
25	.61338	.08305	.06395	.10906	.65760
26	.62124	.08514	.06875	.10184	.58498
27	.68431	.08374	.07926	.07559	.42973
28	.70723	.08218	.08090	.06902	.42157
29	.70723	.08218	.08090	.06902	
30	.73361	.08026	.08224	.06239	

Example 3

Example from elasticity [1], $\mu^2 = 9.8$

VI. IMPROVEMENTS IN IMPLEMENTATION

Various improvements were made while writing and testing the program. One of the first goals was to minimize the total number of iterations. A further problem that developed was determining when the program should switch back to the old parameter from the new parameter. A major problem was the retracing that was pointed out in the previous example. Another difficulty was finding a suitable functional for the change of parameter.

In Chapter III it was pointed out that if the iteration process had not converged by the end of the sixth loop, it was considered to have failed. This is done in order to help minimize the number of iterations. Originally, three variables were read into the program. IVAR1 was used to determine a lower limit on the iterations. If the number of iterations was less than IVAR1, the stepsize was increased. IVAR2 was used to determine an ideal range for the number of iterations. If the number of iterations was greater than IVAR2, then the stepsize was decreased. If the number of iterations was between IVAR1 and IVAR2, there was no change in the stepsize. IVAR3 was the upper limit for the number of iterations allowed to take place. The test set involved two separate problems. Each problem was a system of three polynomial equations and unknowns. In testing, it was found that the minimum total number of iterations occurred when

all three variables were equal to six. No general conclusion is intended here, as a greater variety of problems should be tested before any type of conclusion can be made. It should be noted that in at least one other published report that convergence usually occurred within six iterations if it occurred at all [7].

The switching from the A parameter to the RHO parameter takes place when the program can no longer continue with the A parameter. But when should the program switch from a RHO parameter back to the A parameter? This is important since control over A can only take place while using the A parameter. By control over A, it is meant, that the desired limits on the stepsize can be made and desired answers, for special alphas, can be found as explained in Chapter IX. The switch from RHO to A is desired when the program has passed the singular point for the A parameter.

This switching takes place in two different parts of the program. First, if the program fails for the RHO parameter, the program may try the A parameter as mentioned in the previous chapter. If the number of successful iterations is equal to 1 or 2, it is assumed that the singular point has not yet been passed. Therefore the other functional is tried. If it too fails, then the program stops and a message is printed to the user. If the number of successful iterations is greater than two, then the A parameter is tried. Secondly, the A parameter will be tried if the number of successful iterations is

five. The program assumes that it has passed the singular point. The program tries the A parameter. If the attempt is successful, the program continues in the A parameter. If the attempt is not successful, the program returns and continues with the RHO it was using previously. It will try the A parameter again after another five successful iterations.

With all the switching between parameters, one might wonder if the program ever started to recalculate previous solution pairs. This retracing comes about when the stepsize is calculated for a change of parameter, and it has the wrong sign. Example 3 has a point at which the program started to retrace the previous solutions. The following steps are taken to prevent the retracing. The program saves the last two solution pairs from the old parameter. The program also uses the second solution pair found after a change of parameter. The first solution pair for the new parameter is the same as the last solution pair for the old parameter. Let S_1, S_2, and S_3 correspond respectively to the solution pairs mentioned above. Let A_1, A_2, and A_3 be the alphas associated respectively with S_1, S_2, and S_3. The sign of $(A_2-A_1)(A_3-A_2)$ is tested. If the sign is positive, the program is not retracing. This is true since A_2 lies between A_1 and A_3. The program goes on with the process, skipping the next test. If the sign is negative, the following test is performed. Let X_1, X_2, and X_3 be the answers associated with S_1, S_2, and

S_3 respectively. Let X_1^1, X_2^1, and X_3^1 be the first component of each vector. Now the sign of $(X_2^1-X_1^1)(X_3^1-X_2^1)$ is tested. If the sign is positive, the program is not recalculating previous solutions and the testing stops and the program continues. If the sign is negative, then the next component of the answers is tested as above. The test is done for each component until one test is successful or for all components, the test has failed. If the test fails for all components, then the solution pairs S_1, S_2, and S_3 are not in correct order, i.e., S_1, S_2, S_3 or S_3, S_2, S_1 and the program is retracing the solution pairs calculated previously. The sign of the stepsize is changed and the program starts over at the point where the change of parameter took place.

When the program was being tested it was noted that $\|F_a(a,x)\cdot FX\|$ might be large or small compared to $\|F_x(a,x)\|$. Obviously, $F_a(a,x)\cdot FX$ is singular. If $F_x(a,x)$ is nearly singular, and if there is a greater difference in the norms, the sum of the two matrices would still be nearly singular. Thus the scaling factor of $\|F_x(a,x)\|/(N\cdot\|F_a(a,x)\|)$ was used on FX to make the two matrices approximately the same size.

The problems mentioned above deal with the program difficulties encountered while implementing the theory given in Chapters II and IV. One problem that faces any program written for a computer is how accurate are the results. This major problem is discussed next.

VII. THEORY FOR RIGOROUS BOUNDS

This chapter shows how rigorous bounds may be found for each answer found by Newton's method as developed in the previous chapters.

In theory the following theorem is all that is necessary to obtain a guaranteed bound for an answer.

Theorem (Kantorovic). Let $F(a,x)$ be twice differentiable with the following conditions:

1. $\| F_x(a,x^{(0)})^{-1} \| \leq a$ $\quad x^0$ is the initial iterate
2. $\|x^{(0)}-x^{(1)}\| \leq b$ $\quad$ where $x^{(1)}=x^{(0)}-F_x(a,x^{(0)})^{-1}F(a,x^{(0)})$
3. $\underset{i}{\text{Max}} \sum_{j,k=1}^{N} |F''_{ijk}(a,x)| \leq c$ for all x such that $\|x-x^{(0)}\| \leq 2b$

Then if abc is less than or equal $1/2$, the iterates are uniquely defined and lie within a distance of $2b$ from $x^{(0)}$.

It also can be shown that the iterates converge to x^* where $F(a,x^*) = 0$ and that x^* is the unique solution to $F(a,x) = 0$, within the $2b$-sphere about $x^{(0)}$.

Proof: See Isaacson [3].

There exists a difficulty in trying to implement the above theorem on a digital computer. The numbers are almost never

exact due to roundoff error. There also exists a problem in finding the inverse of a matrix, particularly if there is some error introduced when the matrix was calculated. Interval arithmetic and the Hotelling method of inverting a matrix solve the above problems.

An interval number is a closed bounded interval of the real numbers. If $\bar{x}$ is an interval number then $\bar{x} = [x_\ell, x_u]$ with $x_\ell \leq x_u$. With each real number y associate an interval $\bar{y} = [y, y]$. Thus intervals include real numbers. Let $\circ$ be any of the four arithmetic operations. Then we define

1. $\bar{x} \circ \bar{y} = \bar{z} = \{x \circ y : x \in \bar{x},\ y \in \bar{y}\}$
2. $\pm\bar{x} = 0 \pm \bar{x}$

if $x \circ y$ is defined for each $x \in \bar{x}$ and $y \in \bar{y}$. It should be noted that when interval arithmetic is implemented, it is usually rounded interval arithmetic. The bounds on an answer are rounded so that the machine answer contains the correct interval answer. Thus the roundoff error is taken care of by the machine. The absolute value of $\bar{x}$, $|\bar{x}|$, is as follows:

$$|\bar{x}| = \bar{x} \quad \text{if} \quad x_\ell > 0$$

$$|\bar{x}| = -\bar{x} \quad \text{if} \quad x_u < 0$$

$$|\bar{x}| = [0,\ \max\{|x_\ell|, |x_u|\}] \quad \text{otherwise.}$$

Let $\bar{v}$ be a vector in which each of its N components being an interval

number. Then we define $\|\bar{v}\| = \max_i v_{u_i}$ where $|\bar{v}_i| = [v_{\ell_i}, v_{u_i}]$, $i = 1, 2, \ldots, N$. Let $\bar{M}$ be an $N \times N$ matrix, with each component being an interval number, then $\|\bar{M}\| = T$, where T is the $\max_i$ of the upper bound of $\sum_{j=1}^{N} |\bar{M}_{ij}|$, $i = 1,2,\ldots,N$. Now that interval numbers, vectors, and matrices and their respective norms have been defined, a method of inverting a matrix is needed.

An algorithm for inverting a real nonsingular matrix, A, given by Hotelling (see [2]), is as follows.

Let B be an approximation to A^{-1} then

$$E = I - AB \qquad (A^{-1} = B(I-E)^{-1}) \tag{7.1}$$

If $\|E\| < 1$ then it can be shown that

$$(I-E)^{-1} = I + E + E^2 + E^3 + \ldots \tag{7.2}$$

$$\|(I-E)^{-1} - (I+E+E^2+E^3+\ldots E^k)\| \leq \|E\|^{k+1}/(1-\|E\|) \tag{7.3}$$

approximate A^{-1} by

$$B(I+E+E^2+E^3+\ldots+E^k) \tag{7.4}$$

The Hotelling method yields an error bound easily from Equation (7.3). A modified version of this method is used when the matrix is an interval matrix.

For an interval matrix $\overline{A}$ the following version of Hotelling algorithm is used. (A different application of interval arithmetic to Hotelling's method is given by Hansen [2].) Let $\overline{B}$, of zero width, be a supposed approximation to some matrix in $\overline{A}^{-1}$.

$$\overline{E} = I - \overline{A}\,\overline{B} \qquad (\overline{A}^{-1} = \overline{B}(I-\overline{E})^{-1}) \tag{7.5}$$

If this is not the initial iterate, then a check is made to see if $\|E\|$ is less than the norm of the previous E. If not, then the algorithm terminates (see below). Otherwise, the algorithm continues as follows:

$$\overline{D} = \overline{B}(I+\overline{E}) \qquad (\overline{D} \approx \overline{A}^{-1}) \tag{7.6}$$

Now $\overline{B}_{ij}$ is set equal to the midpoint of $\overline{D}_{ij}$, $i, j = 1, 2, \ldots, N$ and the algorithm returns to (7.5).

The algorithm terminates in the following fashion. If the final $\|\overline{E}\|$ is not less than 1, the algorithm has failed and nothing further can be done. Otherwise, set w equal to $\|\overline{E}\| / (1 - \|\overline{E}\|)$. Finally $\overline{W}$ is made where $\overline{W}_{ij} = [-w, w]$ and $\overline{A}^{-1} = \overline{D} + \overline{W}$. Now all of the inverses of A are contained in $\overline{A}^{-1}$.

In order to calculate a rigorous bound, a solution is first found using the method of the previous chapters. Then $\|[F_x(a,x)]^{-1}\|$ and $\|x^{(i+1)} - x^{(i)}\|$ are calculated for Kantorovič's theorem. Then, if possible, error bounds are found. This procedure is implemented in the next chapter.

VIII. PROGRAM FOR RIGOROUS BOUNDS

Now the feature of the program that enables rigorous solutions to be found will be described. The main goal is to verify that the hypothesis of the theorem in the previous chapter is met.

Chapter X gives a more detailed description of the following subroutines necessary for the rigorous bounds option of this program. The subroutines IFUNC, IPTL, and IAPRTL are identical to FUNCEVAL, PTL, APRTL, respectively, as described in Chapter III, except that they use interval arithmetic and return with type interval variables [4, p. 2-6]. Three other subroutines necessary for rigorous bounds are partially described here. INVERT inverts a square matrix. A flag, IIS, is set to zero if the matrix is singular or nearly so. Otherwise, IIS is set to one and an approximation to the inverse of the matrix is found. HOTEL finds the inverse of an interval matrix, using the inverse found by INVERT as an initial guess (see previous chapter). A flag IHS is set to zero if HOTEL is unable to find an inverse. Otherwise IHS is set to one and the inverse is found. IMATVECT multiplies interval matrix by an interval vector. The real variable, XM2, supplied by the user, is a bound over the second derivative for all points where a rigorous bound is desired. XM2 corresponds to the c in the theorem of the previous chapter.

When the program finds that rigorous bounds are desired for the particular point it is at, the following steps are taken. The Jacobian matrix, $F_x(a, x)$, is calculated in interval form, AI. If the point is desired while using the RHO parameter, the interval form of $F_a(a, x)$ is calculated and AI is modified as in Chapter V. A real matrix, AR, is now made by setting AR_{ij} equal to the midpoint of AI_{ij}, $i, j = 1, 2, \ldots, N$. The matrix BR, an approximation to the inverse of AR, is calculated. If AR is singular or nearly so, then a message is printed and the program returns to the Newton process. Otherwise HOTEL is called to find an interval matrix, CI, which contains all of the inverses of the interval matrix AI. Again, if AI is nearly singular, a message is printed and the program returns to the Newton process. Otherwise $\|CI\|$, which equals the a in Kantorovič's theorem, is found. The interval form of $F(a, x)$, FI, is now calculated and the system, $AI(x^{(1)}-x^{(0)}) = -FI$, is solved.

Since the inverse of AI is now known, AI^{-1} is multiplied by -FI. The $\|x^{(1)}-x^{(0)}\|$ is calculated, which equals the b in the theorem of the previous chapter. If $\|CI\| \cdot \|x^{(1)}-x^{(0)}\| \cdot XM2$ is less than or equal to 0.5, then rigorous bounds can be made. Otherwise $\|CI\|$, $\|x^{(1)}-x^{(0)}\|$, and the product are printed and the program returns to the Newton process. Now that the conditions of Kantorovič's theorem are satisfied, it is known that the true solution lies within a sphere, about X, with a radius of $2\|x^{(1)}-x^{(0)}\|$.

The answers with the bounds are now printed (see example below). The program returns to the Newton process until another answer with a rigorous bound is desired.

Implementation of this and other options is discussed in the following chapter.

```
POINT NUMBER 1
ALPHA=              0E00        X (+OR-    1.014905565E-10) IS
 1.105209593E-02    2.044221957E-02    -6.626234502E-02

POINT NUMBER 2
ALPHA=     1.000000000E-01    X (+OR-    1.006814882E-10) IS
 1.168678237E-02    2.088673801E-02    -6.780584868E-02

POINT NUMBER 3
ALPHA=     2.000000000E-01    X (+OR-    1.211095710E-10) IS
 1.245236576E-02    2.137716679E-02    -6.958681251E-02

POINT NUMBER 4
ALPHA=     3.000000000E-01    X (+OR-    5.718166809E-10) IS
 1.340571274E-02    2.192784014E-02    -7.169358648E-02

POINT NUMBER 5
ALPHA=     4.000000000E-01    X (+OR-    4.542888639E-09) IS
 1.464940827E-02    2.256374558E-02    -7.427887002E-02

POINT NUMBER 6
ALPHA=     5.000000000E-01    X (+OR-    6.406836828E-10) IS
 1.639964067E-02    2.333675410E-02    -7.764852985E-02

POINT NUMBER 7
ALPHA=     6.000000000E-01    X (+OR-    7.476404539E-08) IS
 1.926763841E-02    2.439607761E-02    -8.261243797E-02

POINT NUMBER 16
ALPHA=     7.017433984E-01    X (+OR-    1.681666663E-08) IS
 3.122791839E-02    2.801132785E-02    -9.815057854E-02

POINT NUMBER 17
ALPHA=     7.017433980E-01    +OR-    2.379238140E-08
X(+OR-    7.121444856E-10) IS
 3.122791821E-02    2.801132780E-02    -9.815057834E-02
```

Example 4

Example from elasticity [1], $\mu^2 = 9.8$.

IX. INITIALIZATION AND OPTIONS

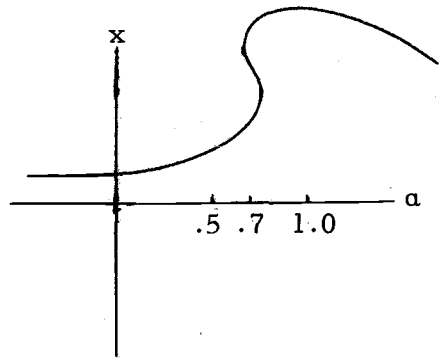
Throughout the previous chapters the main concepts of the program have been discussed, the initialization of the program and various other options will be discussed in this chapter. The user must initialize several variables, constants and select which options are to be exercised. Because the program is designed to be used from a teletype, the initialization is done in a conversational fashion.

The program may be used in a batch job. The user must punch on cards the information asked for in the conversation. The machine's part of the conversation will be printed on the line printer.

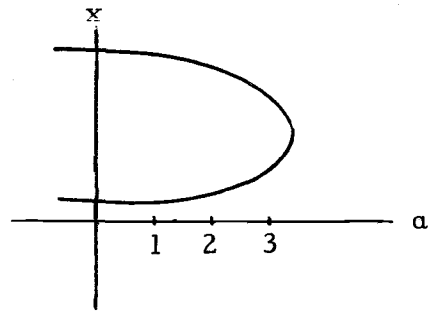
The order in which the initialization takes place is as follows. The program asks first for the starting value and then the last value of α. The initial guess to $F(\alpha, x) = 0$, supplied by the user, should correspond to this starting value of α. The program stops the iteration process if an answer for the last value of α is found. Next a minimum, a maximum, and a starting value for the stepsize, AINCR, is asked for. When the ABS(AINCR) is equal to the absolute value of the minimum stepsize and the program wants to decrease AINCR, a change of parameter takes place. The program will not allow the absolute value of a stepsize to be greater than the absolute value of the maximum stepsize. A constant, typically 10^{-5}, used as part of a convergence criterion is given next. If the norm of the

difference of two successive iterates in Newton's method is less than $\| F_x(a, x) \|$ times the constant, then the process is considered to have converged.

The program then asks if an answer is wanted for any specific a's. The user should respond with a 0 if there are none, and 1 if the answer is yes. The user must list the a's in the order that they will be encountered. For example, if the solution pairs (plotted in 2-dimensions) looked like the following



The user must list 0.7 three times if he wishes the three answers at 0.7. If he lists 0.7 once, he will get the answer for the first 0.7 encountered.



If the graph looks similar to the above example and the user wants all the answers for $\alpha = 1$, $\alpha = 2$, and $\alpha = 3$, he should list the α's in the following order 1 2 3 3 2 1.

After the program asks for the size of the system, it requests the absolute value of the maximum component of the answer expected. The program stops the Newton process when this value is exceeded. Sometimes the system being solved, depends on an auxiliary parameter. The program asks for the value of this parameter, which is not changed during a specific run. The user must supply a value even though there is no such parameter in the problem. Next a request for an increasing factor and a decreasing factor, needed to change the stepsize, is made. The increasing factor must be greater than 1.0 and usually is set 2.0. The decreasing factor must be positive and less than 1.0 and typically is 0.5. The initial guess is given next. The guess should correspond to the starting value of alpha. If $F(\alpha,x)$ is linear for the starting values of alpha, any initial guess will do.

The user now must decide which one of the following four options he would like to use. Option 1 just prints the final answer in FORTRAN E-formats. Option 2 prints each solution pair. If a solution pair is found using the alpha parameter, the first line is as follows:

POINT NUMBER NUMBER OF ITERATIONS
TOTAL ITERATIONS ALPHA AINCR

If the solution pair is found using the rho parameter, the first line is

POINT NUMBER NUMBER OF ITERATIONS
TOTAL ITERATIONS ALPHA RHO RIN

The next line or lines contain the components of the answer. The third option involves rigorous bounds. It will require the point number where the rigorous bounds are wanted. It will also require the user to write the subroutines IFUNC, IPTL, and IAPRTL (see Chapters VIII and X). Finally option 3 requires a bound over the second derivative of the system. Option 3 prints the answers in the following manner.

POINT NUMBER

ALPHA=	X (+OR-) IS	alpha parameter
ALPHA=	±	rho parameter
X(+OR-	) IS	

Then the components of the answers are printed. The fourth option is just like the second option except only desired components of the answer are printed. After the user selects one of the four options, he may chose to have a linear combination of the components of the answer printed. The linear combination is printed on the line below the lines that contain the answer. Figure 9-1 shows one combination of the options possible.

In order to run the program the user must first write, compile, and save the subroutines necessary for the options he wishes to select.

```
STARTING VALUE OF ALPHA IS
0
LAST VALUE FOR ALPHA IS
1
MINIMUM VALUE OF A-INCREMENT IS
.00001
MAXIMUM VALUE FOR A-INCREMENT IS
1
STARTING VALUE FOR A-INCREMENT IS
.1
CONVERGENCE CRITERION CONSTANT IS
.00001
ARE ANY SPECIFIC ALPHAS WANTED? (0=NO, 1=YES)
1
HOW MANY ALPHAS? (MAXIMUM=100)
12
ENTER THE ALPHAS WANTED.
.1 .2 .3 .4 .5 .6 .7 .7 .7 .8 .9 1
THE SIZE OF THE SYSTEM IS
6
THE LARGEST ELEMENT (IN ABS) IS
10
VALUE OF THE AUXILIARY VARIABLE IS
9.8
INCREASING AND DECREASING FACTOR IS
2 .5
ENTER INITIAL GUESS
1 1 1 1 1 1
WHICH OPTION DO YOU WANT? (1, 2, 3, 4)
3
HOW MANY POINTS DO YOU WANT BOUNDED? (MAXIMUM=100)
13
ENTER THE NUMBER OF EACH POINT WANTED.
1 2 3 4 5 6 7 8 29 52 54 56 57
BOUND ON THE SECOND DERIVATIVE IS
150
DO YOU WISH A LINEAR COMBINATION OF THE ANSWER? (0=NO,
1=YES)
0
```

Figure 9-1.

Assuming the user is using a teletype, the user must equip logical unit (LUN) 20 by the statement

#EQUIP, 20=LUN CR

where LUN, usually 61, is the unit where the answers are to be printed. If LUN is the line printer, user should label it. The next statement will be

#LOAD, *NEWT, L=*NEWLB, SAVEFILE CR

where SAVEFILE has the compiled subroutines mentioned above. The user then types

RUN

The user may replace any subroutine in *NEWLB by including it in SAVEFILE.

The numbers can be typed in free format form. A space terminates a number, which may not be continued from one line to another. Only the first 72 characters of a line are examined. When typing numbers on a teletype, mistakes are bound to happen. If a mistake is made when typing a number, simply place any non-numeric character at the end of the number and the whole number will be disregarded. For example, 7.183E-2 is an acceptable number, while 71.83X is ignored.

X. SUBROUTINES

The subroutines used in the program will be described here. The subroutines will include the ones that the user must write depending on the options he wishes to select. Any of the subroutines in the library may be replaced by the user. On the other hand, the user may use any of them in a different application or program. This may be done by simply loading the library, i.e., LOAD, L=*NEWLB, (OTHER BINARY DECKS). When the user supplies any subroutine, he must not change any of the parameters except those that the subroutine is designed to change. For example the call to the subroutine GAUSS has the following list of parameters XMAT, X, N, IGS, and CONST. The variables X and IGS are designed to be changed with X as the solution and IGS as a flag (see below). The remaining variables must not be changed since they may be used elsewhere in the program.

FUNCEVAL, PTL and APRTL are the subroutines the user must supply. The parameter DUM is the auxiliary variable described in the previous chapter. The following FORTRAN statements are necessary for FUNCEVAL.

```
SUBROUTINE FUNCEVAL (F, X, A, DUM)
COMMON STATEMENT (if needed)
DIMENSION F(20), X(20)
```

FUNCEVAL should return with the value $F(a, x)$ at a particular a and X in F. For PTL the following statements are needed.

```
SUBROUTINE    (XJ, X, A, DUM)
COMMON STATEMENT (if needed)
DIMENSION XJ(20, 21), X(20)
```

PTL should return with $F_x(a, x)$, evaluated at a particular a and x, in the matrix XJ. APRTL should return with $F_a(a, x)$ in AP. The statements necessary for APRTL are:

```
SUBROUTINE APRTL (AP, X, A, DUM)
COMMON STATEMENT (if needed)
DIMENSION AP(20), X(20)
```

The subroutine GAUSS has the statements

```
SUBROUTINE GAUSS (XMAT, X, N, IGS, CONST)
DIMENSION XMAT(20, 21), X(20), A(20, 21)
```

GAUSS solves, by Gaussian elimination, an $N \times N$ linear system of equations of the form $XMAT \cdot X = B$. The vector B is the $N+1$st column of XMAT. GAUSS copies XMAT into A so that XMAT is unchanged. The subroutine uses the largest element in absolute value in the suppressed column as a pivot element. A check is made to see if A is nearly singular. If the pivot element is less than $CONST \cdot \| XMAT \|$, then the subroutine terminates and returns with IGS set to zero. When the matrix is in upper triangular form, a back substitution is employed to find the solution. GAUSS then returns with the answer in X and IGS set to one.

There are several minor functions in *NEWLB. PNOR1 is a norm subroutine.

```
FUNCTION PNOR1 (X, N)
DIMENSION X(20)
```

PNOR1 (X, N) = $\max_{1 \le i \le N} |X_i|$. The norm function for matrices is XINM.

```
FUNCTION XINM (X, N, M)
DIMENSION X(20, 21)
```

XINM (X, N, M) = $\max_{1 \le i \le N} \sum_{j=1}^{M} |X_{ij}|$. In order to have free formats and some error correction the function GET was written.

```
FUNCTION GET(IPT)
```

GET uses SCANIN, see [6], to input numbers. A number is terminated by a blank or column 72. IPT is a pointer telling where the terminating character is. A character, illegal in a number, causes the number immediately preceding it to be discarded (see Chapter IX). The value of the number read is returned. The function DOT calculates the dot product of two vectors of N components.

```
FUNCTION DOT (X, FX, N)
DIMENSION X(20), FX(20)
DOT (X, FX, N) = X·FX
```

The subroutine INVERT is used for matrix inversion.

```
SUBROUTINE INVERT (A, N, B, IIS)
DIMENSION A(20, 21), B(20, 20), C(20, 40)
```

INVERT finds an approximation to the inverse of A, an N x N matrix, by using a modified Gaussian elimination. A is copied into columns 1 to N of C and the identity matrix is put into columns N+1 to 2N of C. Then, using a modified version of the algorithm described for GAUSS, C is operated on until the identity matrix

appears in columns 1 to N. If the pivotal element, which is found in the same manner as in GAUSS, is less than $10^{-8} \cdot \|A\|$, IIS is set to zero and the subroutine returns. After the identity matrix appears on the left, columns N+1 to 2N are copied into B. IIS is set to one and the subroutine returns.

The next six subroutines are used for rigorous bounds. ERROR, described in Chapter VIII, has the following statements:

```
SUBROUTINE ERROR (X, A, RHO, N, XM2, IES, FX, DUM)
DIMENSION X(20), FX(20)
```

X and A are a solution pair. RHO is the value of the new parameter and FX is its associated functional. N is the size of the system and XM2 is the bound over the second derivative, i.e., $F_{xx}(a, x)$. IES is a flag which is set to one if ERROR is successful and set to zero otherwise. ERROR prints out the rigorous bounds.

IMATVECT multiplies an interval matrix C by an interval vector F, using interval arithmetic, and returns with the result in FX:

```
SUBROUTINE IMATVECT (C, F, FX, N)
TYPE INTERVAL (4) C, FX, F
DIMENSION C(20, 20), F(20), FX(20)
```

IMATMULT multiplies two square interval matrices, X and Y, together and returns with the result in XMA:

```
SUBROUTINE IMATMULT (X, Y, N, XMA)
TYPE INTERVAL (4) XMA, X, Y
DIMENSION X(20, 20), Y(20, 20), XMA(20, 20)
```

IMATNORM find the maximum norm of the interval matrix T as described in Chapter VII storing it in XMN.

```
SUBROUTINE IMATNORM (T, N, XMN)
TYPE INTERVAL (4) T
DIMENSION T(20, 20)
```

IMAXNORM finds the maximum norm of the interval vector FX storing it in XM.

```
SUBROUTING IMAXNORM (FX, N, XM)
TYPE INTERVAL (4) FX
DIMENSION FX(20)
```

HOTEL is described in Chapter VII.

```
SUBROUTINE HOTEL (AI, BB, C, N, IHS)
TYPE INTERVAL (4) AI, C
DIMENSION AI(20, 21), BB(20, 20), C(20, 20)
```

The inverse of the interval matrix AI is returned in the interval matrix C. BB is a real matrix approximation to the inverse of a matrix in AI. A flowchart for HOTEL is given in Figure 10-1. The subroutine NEWITER does most of the work of the program.

```
SUBROUTINE NEWITER (X, A, ALAST, AMAX, AMIN, AINCR,
EPSIL, N, INS, XI,XD, ELMAY, INUMB, IOP, XM2, IAW, AW,
DUM, IWISH, XDOT, IWSH)
DIMENSION X(20), AW(101), XDOT(20), IWISH(20), IOP(5),
INUMB(100)
```

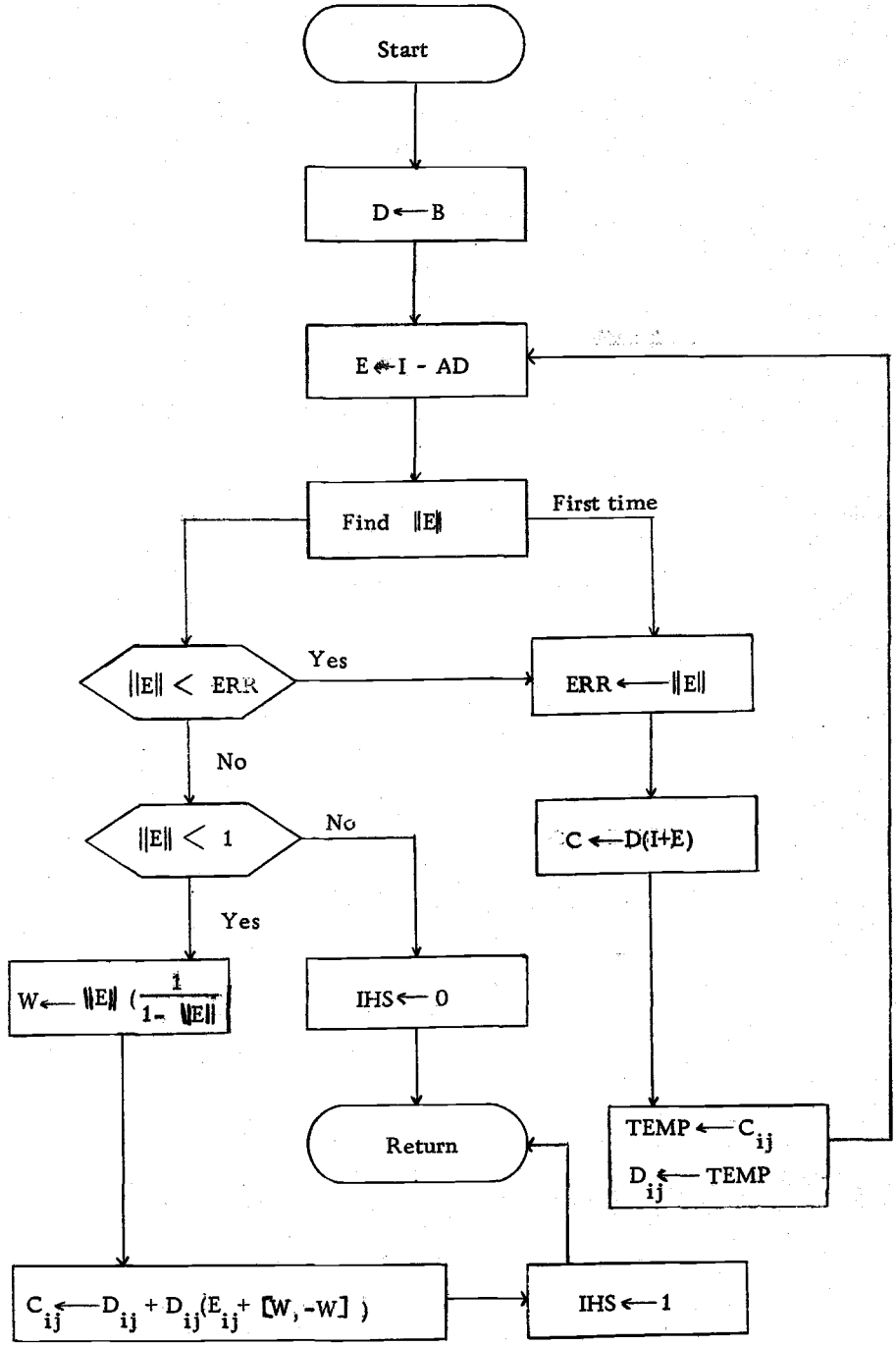


Figure 10-1. Flow chart for HOTEL.

Parameter Name	Input	Normal Output
X	Initial guess	Last answer
A	Starting a	Last a
ALAST	Last a	
AMAX	Maximum stepsize	
AMIN	Minimum stepsize	
AINCR	Starting stepsize	Last stepsize
EPSIL	Convergence constant	
N	Size of system	
INS		Code for return
XI	Increasing factor	
XD	Decreasing factor	
ELMAX	Norm of largest answer	
INUMB	Points for rigorous bounds	
IOP	List of options	
XM2	Bound on $F_{xx}(a, x)$	
IAW	Flag for A's wanted (0 or 1)	
AW	List of A's wanted	
DUM	Auxiliary parameter	
IWISH	List of components wanted	
XDOT	Vector for linear combination	
IWSH	Number components wanted	

The following subroutines are provided in *NEWLB but the user may wish to override some of them with his own. The versions in *NEWLB do nothing. PREP must have the following statements:

SUBROUTINE PREP
COMMON STATEMENT

This subroutine is to initialize constants and pass them to other subroutines written by the user by putting them in COMMON. None of the subroutines in *NEWLB use COMMON. PREP is called at the beginning of each run. IFUNC, IPTL, and IAPRTL are necessary if the user wishes to have rigorous bounds placed on the answers. They

are like their corresponding subroutines in real arithmetic, FUNCEVAL, PTL, and APRTL respectively, except all the arithmetic is performed in interval arithmetic, see [5]. The statements necessary for IPTL are

```
SUBROUTINE IPTL (XJ, X, A, DUM)
TYPE INTERVAL (4) XJ, X
COMMON STATEMENT (if needed)
DIMENSION XJ(20, 20), X(20)
```

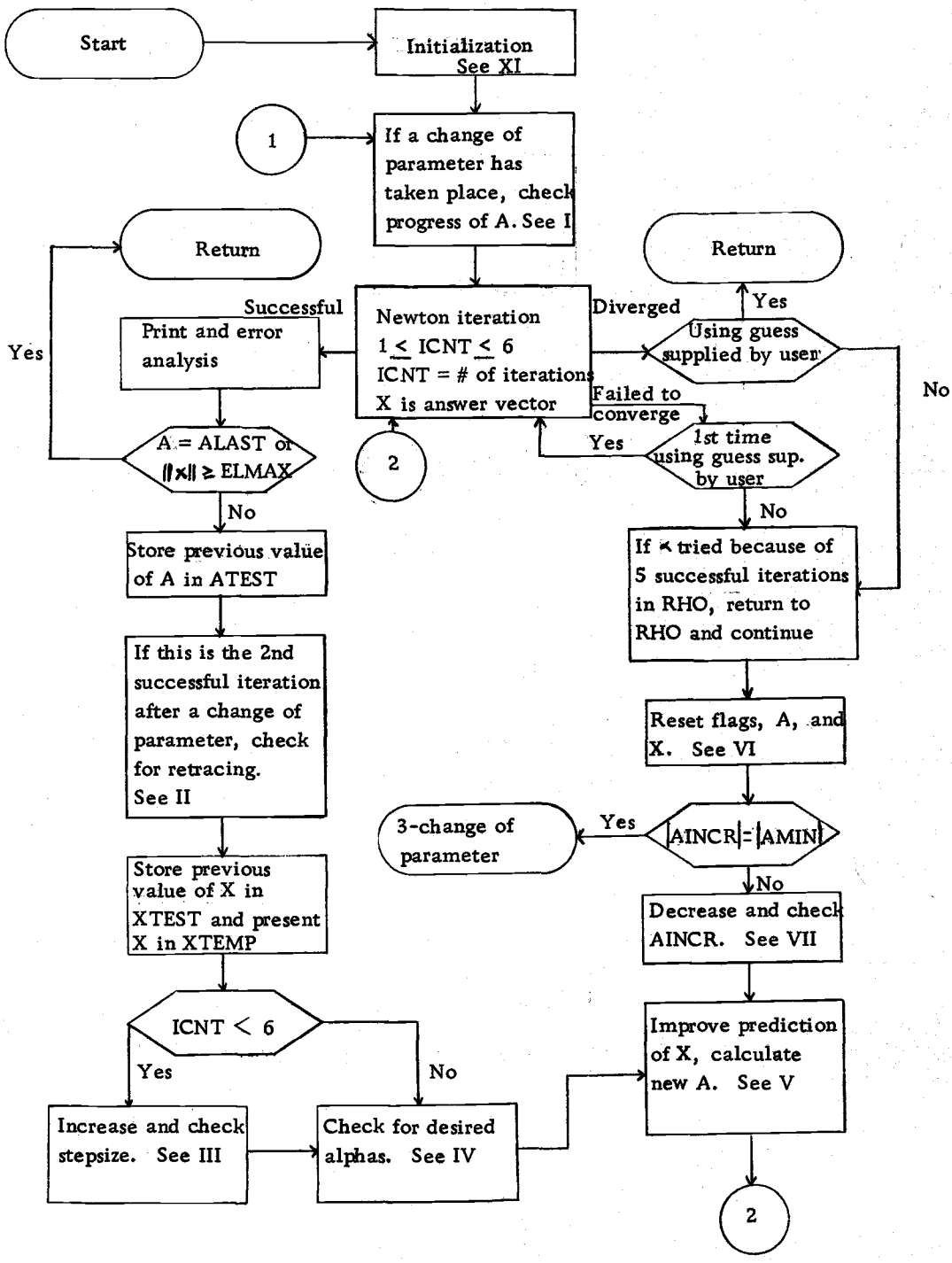
IFUNC has the following statements:

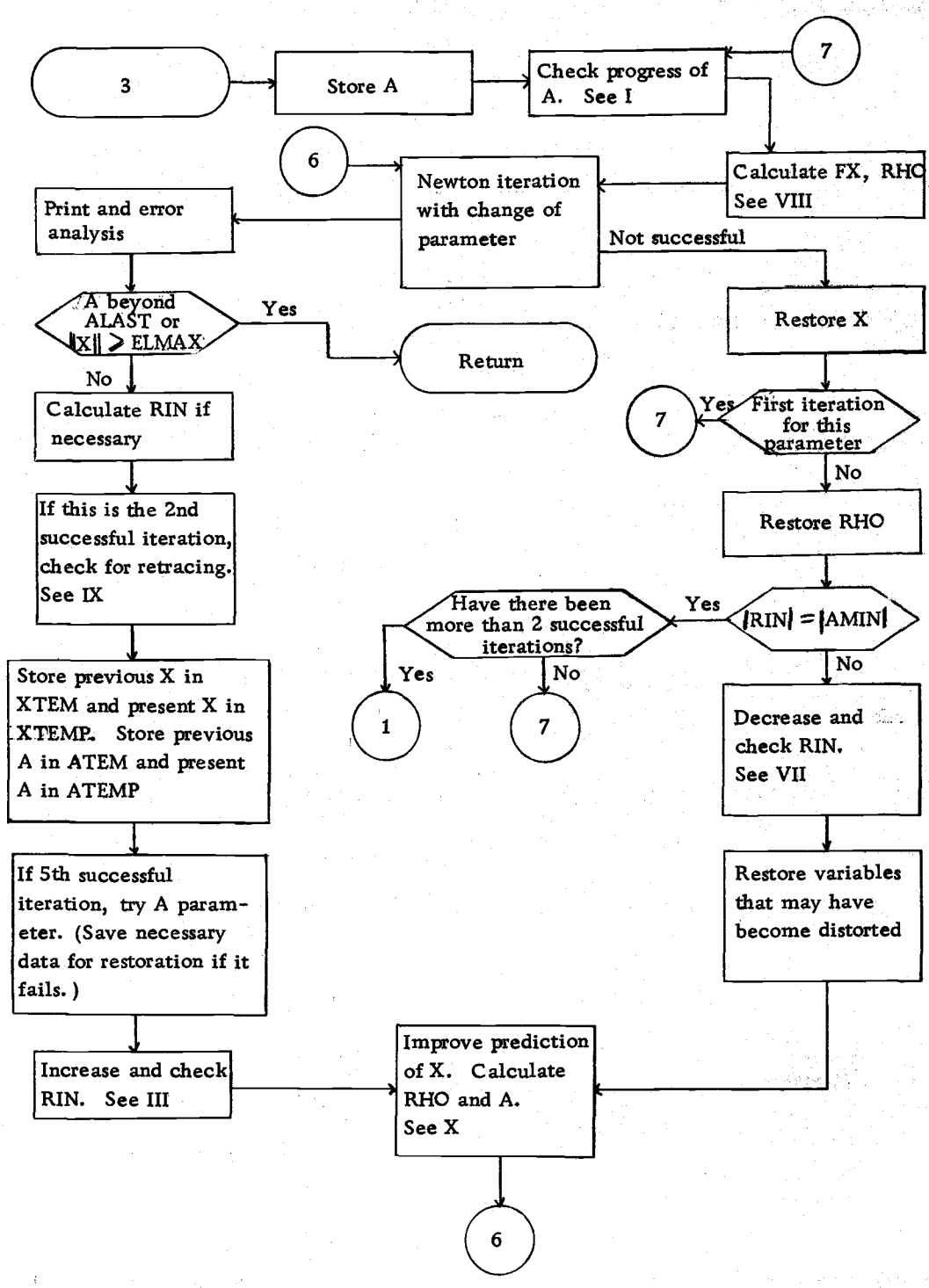
```
SUBROUTINE IFUNC (F, X, A, DUM)
TYPE INTERVAL (4) F, X
COMMON STATEMENT (if needed)
DIMENSION F(20), X(20)
```

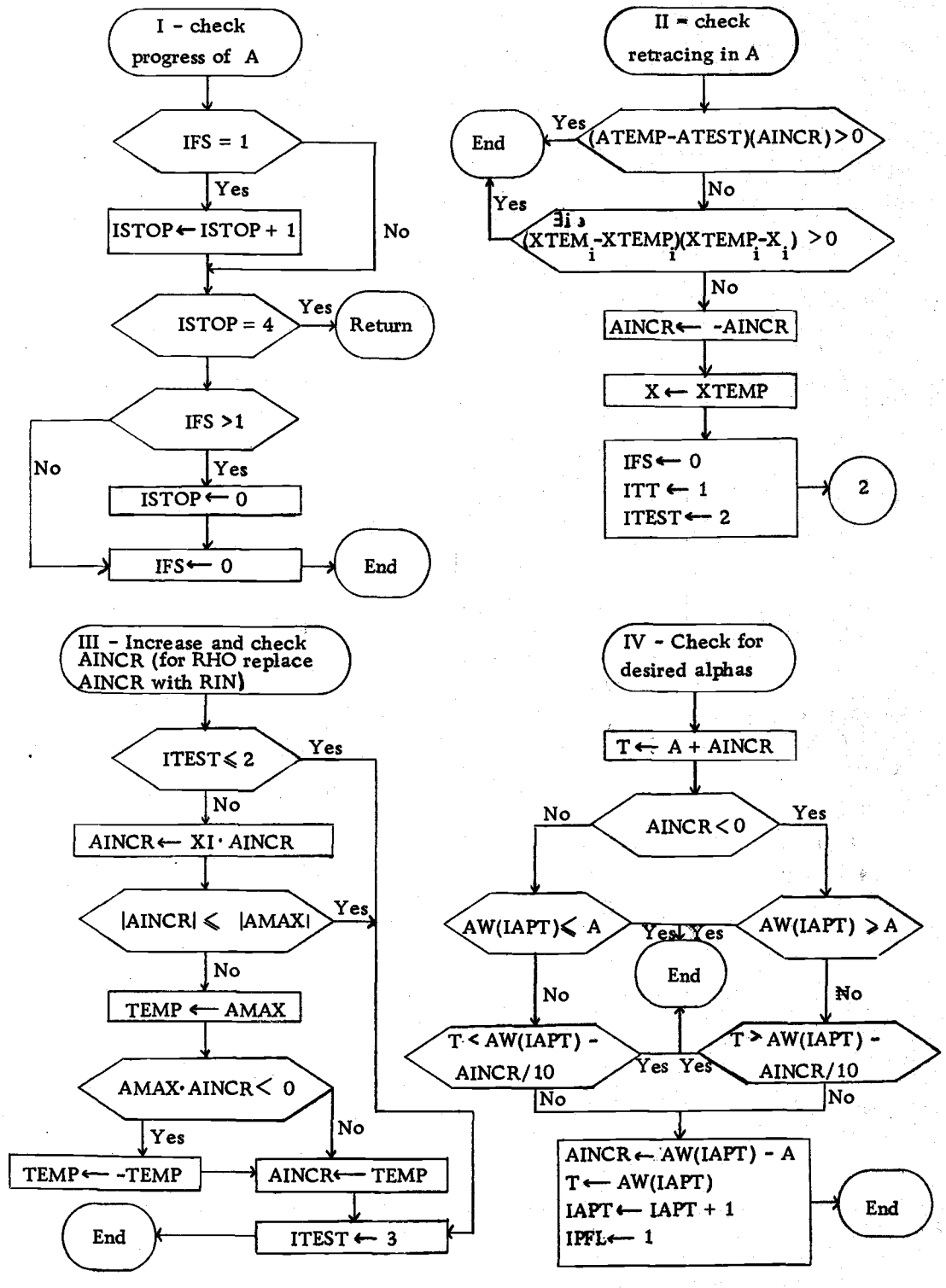
IAPRTL has the following statements:

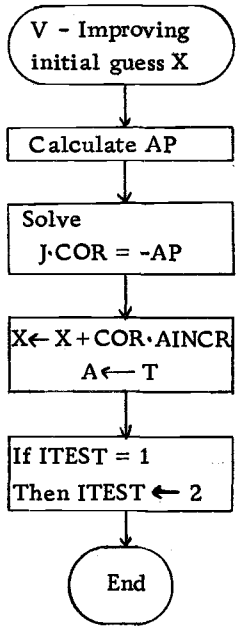
```
SUBROUTINE IAPRTL (AP, X, A, DUM)
TYPE INTERVAL (4) AP, X
COMMON STATEMENT (if needed)
DIMENSION AP(20), X(20)
```

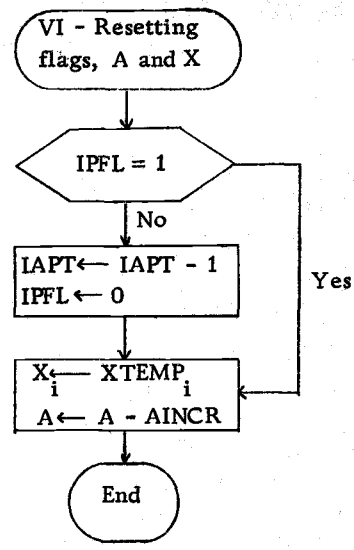
*NEWT contains only the main program and does the initialization in a conversational fashion. We conclude this chapter with a flowchart of the NEWITER subroutine.

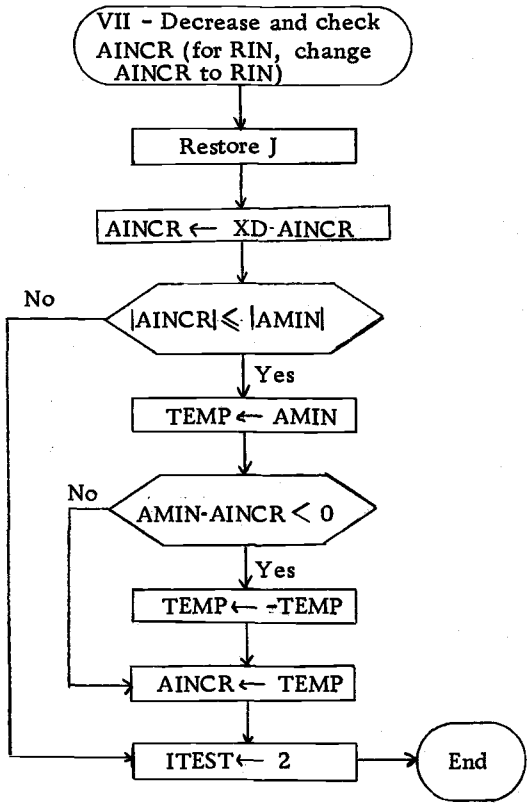

Start
Initialization
See XI
1
If a change of parameter has taken place, check progress of A. See I
Return
Return
Successful
Diverged
Yes
Newton iteration
1 ≤ ICNT ≤ 6
ICNT = # of iterations
X is answer vector
Print and error analysis
Using guess supplied by user
Yes
No
Failed to converge
Yes
1st time using guess sup. by user
2
A = ALAST or ||x|| ≥ ELMAX
No
No
Store previous value of A in ATEST
If tried because of 5 successful iterations in RHO, return to RHO and continue
If this is the 2nd successful iteration after a change of parameter, check for retracing. See II
Reset flags, A, and X. See VI
3-change of parameter
Yes
|AINCR| = |AMIN|
No
Store previous value of X in XTEST and present X in XTEMP
Decrease and check AINCR. See VII
ICNT < 6
Improve prediction of X, calculate new A. See V
Yes
No
Increase and check stepsize. See III
Check for desired alphas. See IV
2

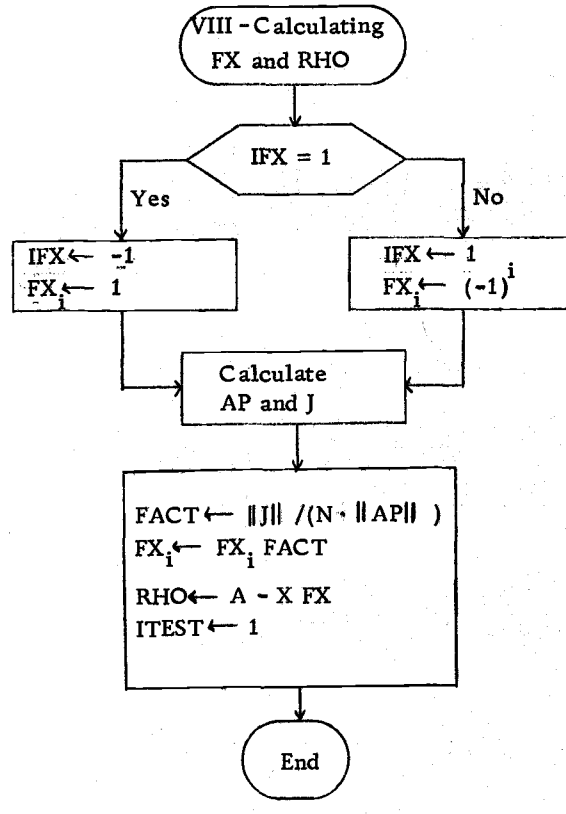

3
Store A
Check progress of A. See I
7
6
Newton iteration with change of parameter
Calculate FX, RHO See VIII
Print and error analysis
Not successful
A beyond ALAST or ||X|| > ELMAX
Yes
Return
Restore X
No
Calculate RIN if necessary
7
Yes
First iteration for this parameter
No
If this is the 2nd successful iteration, check for retracing. See IX
Restore RHO
Have there been more than 2 successful iterations?
Yes
|RIN| = |AMIN|
Yes
No
Yes
No
1
7
Store previous X in XTEM and present X in XTEMP. Store previous A in ATEM and present A in ATEMP
Decrease and check RIN. See VII
If 5th successful iteration, try A parameter. (Save necessary data for restoration if it fails.)
Restore variables that may have become distorted
Increase and check RIN. See III
Improve prediction of X. Calculate RHO and A. See X
6

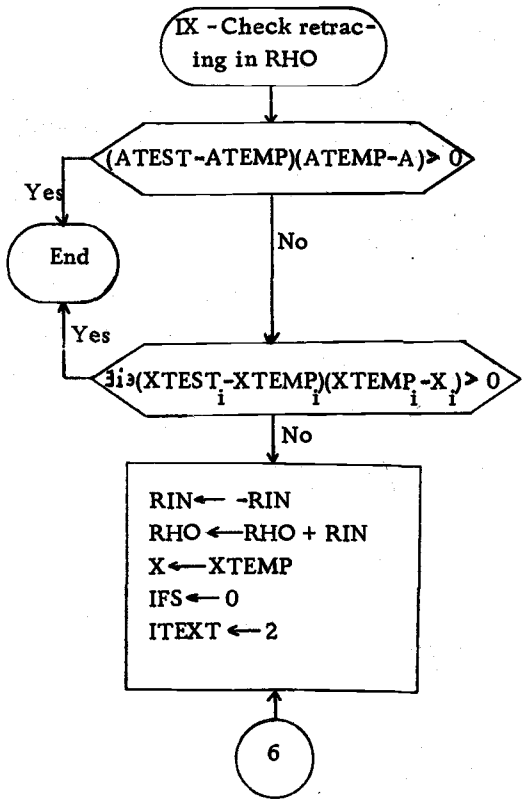

I - check progress of A
IFS = 1
Yes
No
ISTOP ← ISTOP + 1
ISTOP = 4
Yes
Return
IFS >1
No
Yes
ISTOP ← 0
IFS ← 0
End
II - check retracing in A
End
Yes
(ATEMP-ATEST)(AINCR) > 0
No
Yes
∃i ∋
(XTEM_i - XTEMP_i)(XTEMP_i - X_i) > 0
No
AINCR ← -AINCR
X ← XTEMP
IFS ← 0
ITT ← 1
ITEST ← 2
2
III - Increase and check AINCR (for RHO replace AINCR with RIN)
ITEST ≤ 2
Yes
No
AINCR ← XI · AINCR
|AINCR| ≤ |AMAX|
Yes
No
TEMP ← AMAX
AMAX · AINCR < 0
Yes
No
TEMP ← -TEMP
AINCR ← TEMP
End
ITEST ← 3
IV - Check for desired alphas
T ← A + AINCR
No
AINCR < 0
Yes
AW(IAPT) ≤ A
Yes
Yes
AW(IAPT) ≥ A
End
No
No
T < AW(IAPT) - AINCR/10
Yes
Yes
T > AW(IAPT) - AINCR/10
No
No
AINCR ← AW(IAPT) - A
T ← AW(IAPT)
IAPT ← IAPT + 1
IPFL ← 1
End

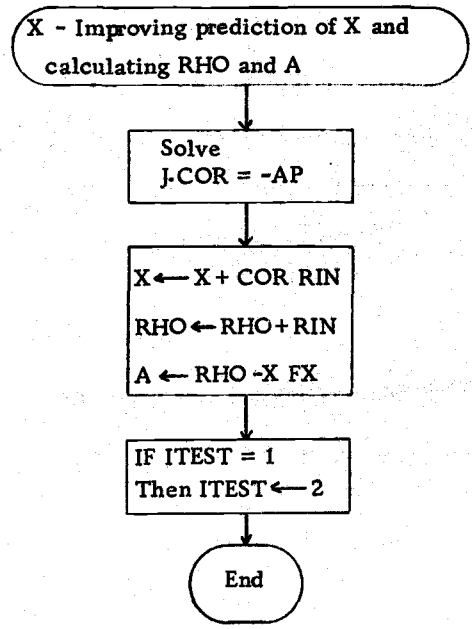

V - Improving initial guess X
Calculate AP
Solve J·COR = -AP
X← X + COR·AINCR
A← T
If ITEST = 1 Then ITEST ← 2
End

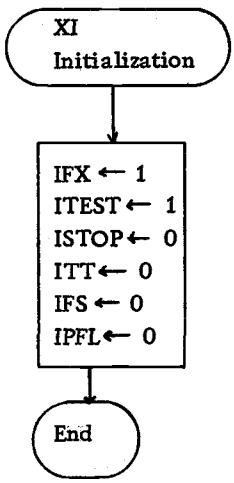

VI - Resetting flags, A and X
IPFL = 1
No
Yes
IAPT← IAPT - 1
IPFL ← 0
X_i ← XTEMP_i
A← A - AINCR
End


VII - Decrease and check AINCR (for RIN, change AINCR to RIN)
Restore J
AINCR ← XD·AINCR
No
|AINCR| ≤ |AMIN|
Yes
TEMP ← AMIN
No
AMIN·AINCR < 0
Yes
TEMP ← -TEMP
AINCR ← TEMP
ITEST← 2
End


VIII - Calculating FX and RHO
IFX = 1
Yes
No
IFX← -1
FX_i ← 1
IFX ← 1
FX_i ← (-1)^i
Calculate AP and J
FACT ← ||J|| /(N · ||AP||)
FX_i ← FX_i FACT
RHO← A - X FX
ITEST ← 1
End


IX - Check retrac-
ing in RHO
(ATEST-ATEMP)(ATEMP-A)> 0
Yes
End
No
Yes
(XTEST-XTEMP)(XTEMP-X)> 0
No
RIN← -RIN
RHO ←RHO + RIN
X←XTEMP
IFS← 0
ITEXT ← 2
6


X - Improving prediction of X and
calculating RHO and A
Solve
J.COR = -AP
X← X + COR RIN
RHO ← RHO + RIN
A ← RHO -X FX
IF ITEST = 1
Then ITEST ← 2
End


XI
Initialization
IFX ← 1
ITEST ← 1
ISTOP ← 0
ITT ← 0
IFS ← 0
IPFL ← 0
End

XI. CONCLUSION

The purpose of this paper was to describe a program which gave rigorous solutions to nonlinear algebraic equations. It should be pointed out that the case of linear equations has already been investigated by Yungen [6, p. 18-28].

The program described involves the use of a parameter, α, that may appear in one of several ways, and Newton's Method. The program will solve $F(\alpha, x) = 0$ for a sequence of α's as long as $F_x(\alpha, x)$ is not singular or nearly so for α_c in the sequence of α's. Example 1 was given to show that the program would work if the above condition was true. If $F_x(\alpha, x)$ were singular for some α_c, then the program would not work for α near α_c. Example 2 was given to show that increasingly smaller stepsizes were necessary for Newton's method to be successful as the α's approached α_c.

When the stepsize became sufficiently small, an automatic change of parameter was made. The change of parameter, it was hoped, made $F_x(\alpha, x)$ not nearly singular for α near α_c. If the new parameter is successful, then Newton's method may be applied to the new parameter. Example 3 was given where Newton's method was not successful in the old parameter α but was successful in the new parameter.

There was also described a feature of the program that enabled

rigorous solutions to be made at selected points. The implementation of rigorous bounds involves a theorem by Kantorovič. The program simply tries to verify that the hypothesis of the theorem is satisfied. If so, it is able to put a bound on the answer.

There are several changes and additions that could be made to the program. It is felt that an automatic differentiating program that would find the various derivatives necessary for the program would be a great help to the user. The user would supply the program with the functions and the program would return with the desired derivatives. With the output of the program printed at a card punch and possibly several cards punched by the user, the subroutines would be written. The subroutine NEWITER may be written so that a single sequence of steps can be used for both parameters. This could be done by considering the functional V, in Chapter IV, as a zero vector. Then the α parameter would be the same as the ρ parameter. When a change of parameter was necessary, one could change the functional and making any other necessary changes, continue in the same sequence of steps.

A different approach to the rigorous bounds and Newton's method is to actually use interval arithmetic in the calculations of each iterate in Newton's process. Instead of one initial guess, use an interval, that is known to contain the solution, as an initial guess. The basic idea is to evaluate the derivative using an interval

containing the solution and evaluate the system at some point in the interval, i.e., solve $F_x(\bar{x})(x^{i+1}-x^i) = -F(x^i)$. For a further discussion see Moore [4, p. 66-69]. This approach should be examined to see if the program with this change in it would be more efficient.

BIBLIOGRAPHY

1. Davis, Joel. The Solution of Nonlinear Operator Equations with Critical Points. Technical Report No. 25. Corvallis, Oregon State University. 1966. 69 numb. leaves.

2. Hansen, Eldon. Interval Arithmetic in Matrix Computations. Journal of the SIAM, ser. B. 2:308-320. 1965.

3. Issacson, Eugene and Herbert Bishop Keller. Analysis of Numerical Methods. New York, Wiley and Sons, 1966. 541 p.

4. Moore, Ruman E. Interval Analysis. Englewood Cliffs, New Jersey, Prentice-Hall Inc., 1966. 145 p.

5. Oregon State University Computer Center. Interval Arithmetic. Corvallis, Oregon, 1967. 8 numb. leaves.

6. Oregon State University Computer Center. Scanin. Corvallis, Oregon, 1968.

7. Reiss, Edward L., Herbert J. Greenwood and Herbert B. Keller. Nonlinear Deflections of Shallow Spherical Shells. Journal of the Aeronautical Sciences 24:533-543. July, 1967.

8. Yungen, Walter Arthur. Rigorous Computer Inversion of Some Linear Operators. Master's Thesis. Corvallis, Oregon State University, 1967. 52 numb. leaves.