INSTITUTE OF TECHNOLOGY

AIR UNIVERSITY

UNITED STATES AIR FORCE

SCHOOL OF ENGINEERING

THESIS

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

AF-WP-B-SEP 60 5M

AUTOMATIC COMPUTATION OF ROOT LOCUS USING DIGITAL COMPUTER

THESIS

Presented to the Faculty of the School of Engineering of

the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

By

Stuart Brown Herndon, B. S.
Capt USAF

Graduate Electrical Engineering

August 1961

Preface

The objective of this thesis is to relieve the engineer of the laborious and time consuming task of computing the root locus manually. The objective is achieved by developing a program for the digital computer which uses the ability of the computer to perform accurate and rapid calculations to compute the root locus automatically.

I wish to acknowledge my sincere appreciation to Captain C. W. Richard, Jr., for the time and effort he so generously expended to assist me over the mathematical hurdles encountered in this thesis. The confidence which I gained from him concerning the feasibility of the thesis was an immeasurable asset. I am deeply indebted to my Faculty Thesis Advisor, Prof. J. J. D'Azzo, for the exceptionally competent guidence and encouragement he has given me in preparing this thesis.

Without my wife, Holly, I could not have written this thesis. In addition to typing the entire report, she has been a constant source of hope and inspiration.

Stuart B. Herndon

Contents

Contents

List of Figures

List of Tables

Abstract

The problem of computing the root locus is transformed into the format of repeated solutions of the characteristic equation expressed as a polynomial equation of degree N. To resolve the problem of convergence, a simultaneous Newton-Raphson and Bairstow iteration, with substitution of variables when required, is employed to solve the polynomial equation. By means of a systematized iterative process, a program is developed in FORTRAN language for the IBM 1620 which computes the root locus automatically. Several root loci formed by plotting the roots evaluated by the computer, which are accurate to at least four figures, are provided in the Appendices.

AUTOMATIC COMPUTATION OF ROOT LOCUS USING DIGITAL COMPUTER

I. Introduction

Subject

An engineer who designs control systems must resolve the dual problem of analyzing and synthesizing control systems. The problem can be resolved by deriving the differential equations for a control system and solving these equations to obtain an accurate solution of the control system's performance; however, the approach is not feasible for other than simple systems. Among the several methods available, the root-locus method offers distinct advantages provided the design engineer can obtain the root locus without expending excess time and effort. The root-locus method incorporates the desirable features of both the classical method and the frequency-response method by providing an accurate solution of the transient and the steady-state responses of the system.

Although another advantage of the root-locus method is the relative ease and simplicity of applying the method, manual computation of the root locus becomes laborious and time consuming. In manual computation, the engineer uses a trial and error procedure based on geometrical short cuts (Ref 1:144-154) to determine specific points on the root locus. These points are then plotted and connected by the use of a French Curve or other means to form the root

locus. The use of a Spirule (Ref 1:508), which is basically a protractor with a revolving concentric arm, facilitates this procedure. However, the accuracy is limited by the skill and experience of the person performing the calculations and manipulating the Spirule. Therefore, the effectiveness of the root-locus method depends upon the ease and accuracy with which the root locus may be obtained. Digital Computers are capable of performing calculations in a matter of microseconds with an accuracy limited only by the number of figures retained in the computer calculation. The ability of digital computers to perform accurate and rapid computation suggests their use for the automatic computation of the root locus.

Purpose

The scale generally used for plotting the root locus is such that only three significant figures can be plotted. The purpose of this thesis is to develop a program for the digital computer which computes the root loci for control systems with an accuracy consistent with the engineer's needs, thereby relieving him of the burdensome task of manual computation. In conjunction with the objective of minimizing the time and effort required of the engineer, the FORTRAN (FORmula TRANslation) system of programming is used in this thesis. FORTRAN is used because it is a concise, convenient means for stating the steps for problem solution in mathematical formulas, which are translated automatically by the FORTRAN processor (as the name implies) into basic machine language. Therefore, little knowledge of the computer is required of the engineer to

understand and to execute the program.

Scope and Limitations

The program developed in this thesis is written in FORTRAN language for the IBM 1620 Computer, which is located in the Institute's Computer Laboratory. This computer was used because it was the computer most readily available and it is representative of the smaller members of the computer family. The similarity of all digital computers should facilitate the conversion of the program in this thesis to a program for any of the other digital computers.

As in the manual method of computing the root locus, the program is based on the fact that the poles of the closed-loop system are related to the zeros and poles of the open-loop transfer function and to the static loop sensitivity (Ref 1:131). By using this relation, the characteristic equation is expressed as a general polynomial equation of degree N. The program employs a simultaneous Newton and Bairstow iterative process, with a substitution of variable when required to ensure convergence (Ref 2:644), to compute the roots of the polynomial equation. The values of the roots are printed in tabular form for the engineer to plot the root locus. If an automatic plotter is available, the root locus may be plotted automatically. The program is designed so that the static loop sensitivity may be varied by a geometric or an arithmetic progression. Therefore, the engineer may use his engineering judgment and knowledge of root locus to control the speed and order of the computation by choosing between the two methods of varying the sensitivity.

To prevent exceeding the memory capacity of the IBM 1620 Computer, the degree of the polynomial is limited to N=20. Since the degree of the characteristic equation is determined by the number of open-loop poles, the program is limited to the calculation of the root locus for a control system of twenty or less open-loop poles. The program is further limited by the plotting accuracy requirement that the roots computed be accurate to at least three figures. The accuracy of the roots, as evaluated by the iterative process, degenerates as the degree of the polynomial equation is increased and as the multiplicity of roots increases. Results indicate that the program is capable of computing accurately the root loci for control systems with ten or less open-loop poles provided the multiplicity of roots does not exceed four.

The static loop sensitivity cannot be varied from zero to infinity because the largest permissible number in FORTRAN is 10^{50}, however, for each of the root locus thus far computed, it has been possible to vary the sensitivity to a sufficiently large value to obtain the useful part of the root locus.

Plan of Development

Before attempting to develop a program for the digital computer to compute the root locus, a detailed analysis of a typical feedback-control system is performed in Chapter II. The definition and derivation of the root locus is investigated so that the problem of computing the root locus may be transformed into a format consistent with computer solution techniques.

The mathematical theory and procedures used to solve the problem

are examined and explained in Chapter III. The procedures are further systematized so that decisions can be preset in a mechanized digital computer program.

In Chapter IV the mechanized procedures, expressed as a series of formulas, are translated into a set of related commands to form the program. A detailed flow chart and the program as typed by the computer are provided in the appendices, on gate-folds, for the convenience of the reader.

The results and conclusions are contained in Chapter V. Thus, the thesis is divided into five chapters as follows:

I Introduction

II Problem Format

III Mathematical Theory and Procedures

IV Development of Program

V Results and Conclusions

II. Problem Format

Before attempting to write a program for a digital computer, it is essential that all aspects of the problem to be solved are understood and that the problem is expressed in a format which is consistent with computer techniques of solution. To gain a better insight into the problem of computing the root locus, a typical feedback-control system is analyzed to investigate the definition and derivation of the mathematical expressions of the root locus. From this analysis, the problem format is established.

Definition and Derivation of Root Locus

To simplify the analysis of a closed-loop system, a functional block diagram is used, as shown in Fig. 1, in which the control elements have been cascaded to produce the over-all foward and feedback transfer functions G and H.

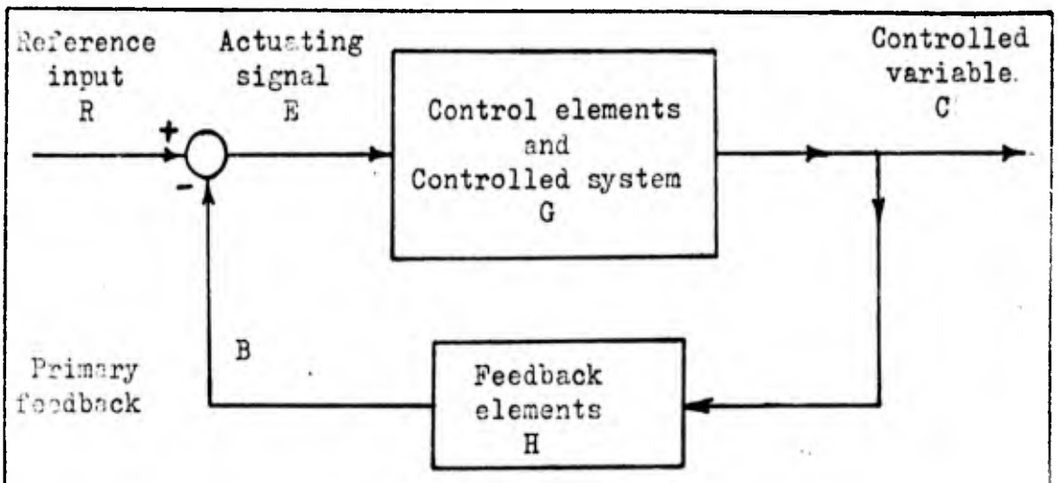


FIG. 1
[illegible] Feedback Control System
(From [illegible])

The equations, describing the control system shown in Fig. 1, may be expressed in terms of the Laplace-transform variable S as (Ref 1: 97-98)

$$C(s) = G(s)\,E(s) \tag{1}$$

$$B(s) = H(s)\,C(s) \tag{2}$$

$$E(s) = R(s) - B(s) \tag{3}$$

where R = reference input

E = actuating signal

C = controlled variable

B = primary feedback

G = forward transfer function

H = feedback transfer function

The combination of Eqs (1), (2), and (3) produces the closed-loop transfer function or control ratio

$$\frac{C(s)}{R(s)} = \frac{G(s)}{1 + G(s)\,H(s)} \tag{4}$$

The open-loop transfer function is defined as the ratio of B(S) to E(S). Combining Eqs (1) and (2) yields

$$\frac{B(s)}{E(s)} = G(s)\,H(s) \tag{5}$$

In securing the open-loop transfer function, it is important to keep the factors in either linear or quadratic form. In linear form the open-loop transfer function may be expressed as

$$G(s)\,H(s) = \frac{K\prod_{w=1}^{M}(s+a_w)}{\prod_{u=1}^{N}(s+b_u)} \tag{6}$$

where the a's and b's may be real, complex, or zero. When the transfer function is expressed with the coefficients of S equal to unity, as in Eq (6), K is defined as the static loop sensitivity (Ref 1:138). For brevity, K is referred to as sensitivity through the remainder of the thesis.

An inspection of Eq (6) shows that the zeros and poles of the open-loop transfer function exist at

$$S = -a_1, -a_2, \cdots, -a_w, \cdots, -a_M \tag{7}$$

$$S = -b_1, -b_2, \cdots, -b_u, \cdots, -b_N \tag{8}$$

where N and M are the number of open-loop poles and zeros respectively.

The characteristic equation of the closed-loop system may be obtained from the denominator of the control ratio and written as

$$1 + G(s)\,H(s) = 0 \tag{9}$$

Substituting Eq (6) into Eq (9) the characteristic equation becomes

$$1 + \frac{K\prod_{w=1}^{M}(s+a_w)}{\prod_{u=1}^{N}(s+b_u)} = 0 \qquad (10)$$

The root locus is defined as a plot of the roots of the characteristic equation as the sensitivity K is varied from zero to infinity. As previously stated and as shown in Eq (10), the underlying principle of the computation of the root locus is based on the fact that the poles of the system's closed-loop transfer function, which are the roots of the characteristic equation, are related to the zeros and poles of the system's open-loop transfer function.

Characteristic equation expressed as polynomial equation of Degree N

Eq (10) may be rewritten as

$$\prod_{u=1}^{N}(s+b_u) + K\prod_{w=1}^{M}(s+a_w) = 0 \qquad (11)$$

where the a's and b's are the negatives of the zeros and poles, respectively, as shown in Eqs (7), (8). Also, since the zeros and poles may be complex and it is known that the complex zeros and poles always occur in conjugate pairs, the complex pairs may be expressed as quadratic factors. Thus, Eq (11) can be written as

$$\prod_{\substack{v=1\\ c=1}}^{\substack{Q\\ P}}(s+b_v)(s^2+X_c s+Y_c) + K\prod_{\substack{z=1\\ d=1}}^{\substack{T\\ R}}(s+a_z)(s^2+X_d s+Y_d) = 0 \qquad (12)$$

where the number of open-loop poles N, and the number of open-loop zeros M are given by

$$N = P + 2Q \tag{13}$$

$$M = R + 2T \tag{14}$$

By performing the multiplication of factors as indicated in Eq (12), the characteristic equation can be expressed as the sum of two sub-polynomials

$$E_1 S^N + E_2 S^{N-1} + \cdots E_N S + E_{N+1} + K\left[D_1 S^M + D_2 S^{M-1} + \cdots D_M S + D_{M+1}\right] = 0 \tag{15}$$

where N and M are as specified in Eqs (13) and (14) and the sub-polynomial formed by the zeros is multiplied by the sensitivity K. Multiplying the D's by K and adding coefficients of equal powers of S, the two sub-polynomials may be consolidated to produce

$$A_1 S^N + A_2 S^{N-1} + \cdots A_N S + A_{N+1} = 0 \tag{16}$$

which is a polynomial equation of degree N, since the number of open-loop poles N is equal to or greater than the number of open-loop zeros M. As seen by a comparison of Eqs (15) and (16), one or more of the A coefficients are functions of K with the exact number being determined by the relative values of N and M as shown in the following example.

Example. Consider the open-loop transfer function

$$G(s)\,H(s) = \frac{K(s+a_1)}{(s+b_1)(s^2+X_1 s+Y_1)} \tag{17}$$

Substituting into the characteristic equation, Eq (9), yields

$$1 + \frac{K(s+a_1)}{(s+b_1)(s^2+X_1 s+Y_1)} = 0 \tag{18}$$

which expressed in the form of Eq (12) is

$$(s+b_1)(s^2+X_1 s+Y_1) + K(s+a_1) = 0 \tag{19}$$

Performing the multiplication and adding coefficients of equal powers, Eq (19) becomes

$$s^3 + s^2(b_1+X_1) + s(Y_1 + b_1 X_1 + K) + (b_1 Y_1 + K a_1) = 0 \tag{20}$$

By inspection, Eq (20) is of the same form as Eq (16), where the degree is N=3 and

$$A_1 = 1$$
$$A_2 = b_1 + X_1$$
$$A_3 = Y_1 + b_1 X_1 + K$$
$$A_4 = A_{N+1} = b_1 Y_1 + K a_1$$

In this example, only the coefficients A_3 and A_4 are functions of K; however, if the number of zeros had been equal to the number

of poles, all the coefficients would have been functions of K.

As the above analysis shows, if the open-loop transfer function of a control system is known, the characteristic equation can be expressed as a general polynomial equation of degree N. From the definition of the root locus, a plot of the roots of the general polynomial equation as the sensitivity K is varied from zero to infinity would produce the root-locus. As a point of interest, an analysis of Eq (11) reveals that, when K is equal to zero, the roots are the open-loop poles and that, as K approaches infinity, the roots approach the open-loop zeros or infinity.

Since the polynomial is formed by a process of multiplication and addition which involves the open-loop poles and zeros, the process can be mechanized and a program can be developed which will perform the computation for any given open-loop transfer function. The general method of finding the roots of a polynomial equation employs numerical techniques which are readily adaptable to digital computer techniques of solution. Therefore, the problem of computing the root locus has been transformed into a format such that a generalized program can be developed to compute the root locus for any control system for which the open-loop transfer function is known.

III. MATHEMATICAL THEORY AND PROCEDURES

Theory of Numerical Techniques

There are numerous numerical methods for computing the roots of a polynomial equation expressed as

$$f(S) = A_1 S^N + A_2 S^{N-1} + \cdots A_N S + A_{N+1} = 0 \tag{21}$$

where the A coefficients are real.

Most of the useful methods involve iterative processes (Ref 3:443) in which an initial approximation Z_0 to a real root S=r is estimated by graphical methods or other means, and a recurrence relation is used to generate a sequence of successive approximations Z_1, Z_2, . . , Zn, . . . which converges to the limit r. One of these methods is the relatively simple yet representative method known as successive substitutions. In this method, Eq (21) is expressed as

$$S = F(S) \tag{22}$$

and the recurrence relation is of the form

$$Z_{k+1} = F(Z_k) \tag{23}$$

Example. As an example of the successive substitutions method, consider the polynomial equation

$$S^3 - 23S^2 + 62S - 40 = 0 \tag{24}$$

This may be expressed in the form of Eq (22) as

$$S = 23 - \frac{62}{S} + \frac{40}{S^2} \tag{25}$$

In this example, the initial estimate Z_0 to the root r is obtained by assuming a large root exists and that it is sufficiently large such that the last two terms of Eq (25) are negligible (an estimate could have been determined by making a plot of the polynomial). Therefore, $Z_0=23$ is the initial estimate and using the recurrence relation in Eq (23), with k=0, yields

$$Z_1 = F(Z_0) = 23 - \frac{62}{23} + \frac{40}{(23)^2} = 20.3756$$

$$Z_2 = 20.3756 - \frac{62}{20.3756} + \frac{40}{(20.3756)^2}$$

$$Z_3 = Z_2 - \frac{62}{Z_2} + \frac{40}{(Z_2)^2}$$

. . . . . .

$$Z_{N+1} = Z_N - \frac{62}{Z_N} + \frac{40}{(Z_N)^2}$$

This iteration is continued until convergence occurs, i.e., $Z_{k+1}-Z_k$ equals zero or is less than a specified tolerance, or it

is determined that the estimated root does not exist and the iteration is diverging. In this example, the iteration (which was performed on the digital computer) converges to twenty; therefore, S=20 is a root and (S-20) is a factor which may be divided into Eq (24) to produce a reduced polynomial equation of

$$S^2 - 3S + 2 = 0 \tag{26}$$

The reduced polynomial equation may be further reduced by the iterative process until all roots of the original polynomial equation have been found; however, in this simple case, the quadratic formula could be used.

There are many convenient ways of expressing the polynomial equation in the form of Eq (22) which leads to the various numerical methods for computing the roots, as mentioned earlier. The convergence or divergence of the sequence, for each method used, depends on the initial estimate to the root and the class of the polynomial (Ref 3:443). Unfortunately, in all these methods, convergence is: (1) uncertain, (2) slow and excessively laborious, or (3) critical concerning the initial estimate of the root.

Problem of Convergence

To gain a better understanding of why convergence is uncertain or why convergence depends upon the class of polynomial and the initial estimate to the root, one of the better known methods for computing the real roots, the Newton-Raphson method, is examined

to illustrate the convergence problem. The Newton-Raphson method is a special case of the successive-substitution method where Eq (22) is expressed as (Ref 3:447)

$$F(S) = S - \frac{f(S)}{f'(S)} \tag{27}$$

and the recurrence formula becomes

$$Z_{k+1} = Z_k - \frac{f(Z_k)}{f'(Z_k)} \tag{28}$$

where Z_k = initial estimate of root

$f(Z_k)$ = polynomial evaluation at Z_k

$f'(Z_k)$ = derivative of polynomial evaluated at Z_k

Z_{k+1} = corrected estimate to root

A plot of the polynomial equation expressed as

$$Y = f(S) = 0 \tag{29}$$

shows graphically the values of S at which Y=0; therefore, these values of S are real roots of the polynomial equation. The plot of the polynomial equation shown in Fig. 2, is abbreviated for sake of clarity to depict only one crossing of the abscissa axis or one real root at r.

From an inspection of the geometric solution of the recurrence formula shown in Fig. 2, it is obvious that Z_i approaches r as the number of iterations k approaches infinity.

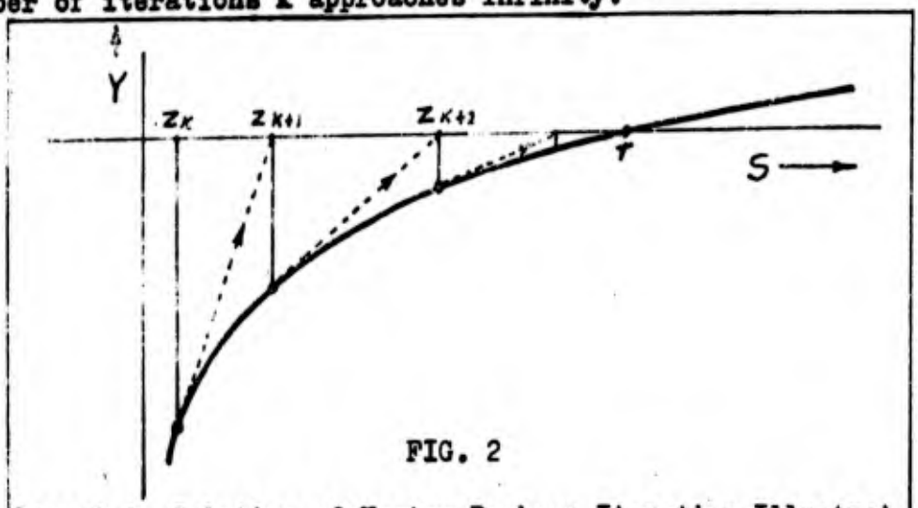


FIG. 2

Geometric Solution of Newton-Raphson Iteration Illustrating Convergence (From Ref 3:448)

The iteration shown in Fig. 2, is an example where convergence is certain; however, other classes of polynomials, such as the one plotted in Fig. 3, may be selected to show geometrically that convergence is uncertain and is critical concerning the initial estimate Z_0 to the root r.

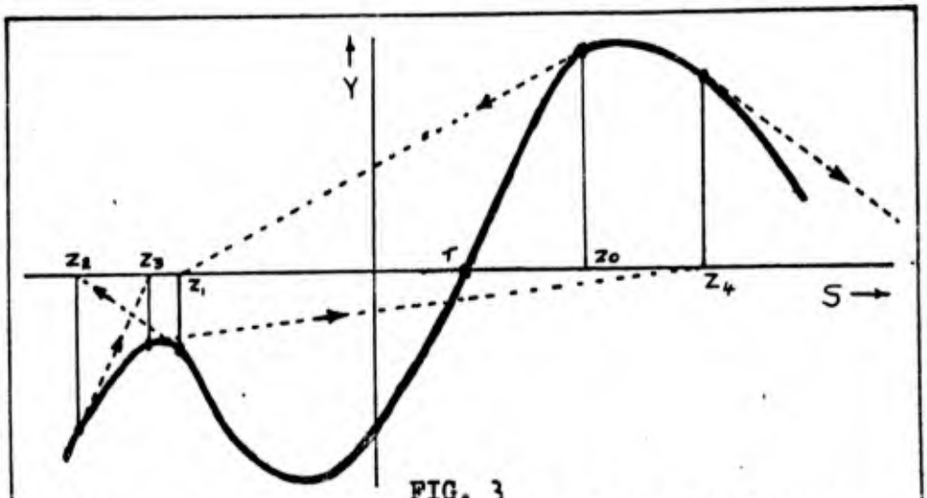


FIG. 3

Geometric Solution of Newton-Raphson Iteration Illustrating Non-Convergence (From Ref 3:449)

As shown in Fig. 3, the geometric solution to the recurrence formula reveals that Z_i is not approaching the root r; therefore, convergence to the root r is not achieved. However, convergence to another root may or may not occur if other roots exist.

Solution to Convergence Problem

The problem of uncertain convergence may be overcome by using two different methods simultaneously (Ref 2:644). The Newton-Raphson method solves for real roots provided the initial estimate Z_0 is real and is sufficiently close to the true value of the root. In principle, the method is valid for the determination of complex linear roots, however, the process must be initiated with an initial estimate Z_0 which is complex and the succeeding operations involve calculations with complex numbers. Since the coefficients of the polynomial equation $f(s)=0$ are real, the complex roots occur in conjugate pairs, and this fact is exploited by solving for quadratic factors which are the product of two linear complex factors. The Bairstow method is a method which solves for these quadratic factors.

It is well known that there is a large class of polynomials for which the Newton-Raphson Method fails to converge. Also, there is a large class for which the Bairstow Method fails. However, the class of polynomials for which the Bairstow Method fails is generally not the same class as that for which the Newton-Raphson Method fails (Ref 2:645).

The simultaneous use of the Bairstow and Newton Method (Ref 2: 644) results in a powerful iterative process for solving polynomial

equations provided the equation is scaled and the coefficients are reversed, when necessary, to prevent loss of significance in the synthetic division. The Newton-Raphson and Bairstow Methods are now discussed in detail to show how these methods of solution involve systematic procedures and decisions which can be preset in the mechanization of a program for the digital computer. Also, the criteria for scaling the polynomial and for reversing the coefficients will be analysed after the theory of the two methods have been examined.

Newton-Raphson Method

As previously stated, the Newton-Raphson Method is used to find real roots of the polynomial, where the corrections to the initial estimate Z_0 to real root r are given by

$$Z_{K+1} = Z_K - \frac{f(Z_K)}{f'(Z_K)} \tag{30}$$

To systematize the calculation of Eq (30) for the digital computer, it can be shown that f(z) is the remainder R when the polynomial f(s) is divided by the factor (S-Z). Dividing the polynomial, as expressed in Eq (21), the quotient may be expressed as a reduced polynomial and a remainder term

$$\frac{f(S)}{(S-Z)} = B_1 S^{N-1} + B_2 S^{N-2} + \cdots B_{N-1} S + B_N + \frac{R}{(S-Z)} \tag{31}$$

Multiplying Eq (31) by the factor (S-Z) produces

$$f(S) = (S-Z)(B_1 S^{N-1} + B_2 S^{N-2} + \cdots B_{N-1} S + B_N) + R \qquad (32)$$

By letting S take on the value Z, Eq (31) may be written as

$$f(Z) = R \qquad (Q.E.D.) \qquad (33)$$

Also, it can be shown that f'(Z) is the remainder R' when the reduced polynomial of Eq (31)

$$B_1 S^{N-1} + B_2 S^{N-2} + \cdots B_{N-1} S + B_N$$

is again divided by the factor (S-Z) to produce

$$\frac{B_1 S^{N-1} + B_2 S^{N-2} + \cdots B_{N-1} S + B_N}{(S-Z)} = C_1 S^{N-2} + C_2 S^{N-3} + \cdots C_N S + C_{N-1} + \frac{R'}{(S-Z)} \qquad (34)$$

Multiplying by (S-Z), Eq (34) may be written as

$$B_1 S^{N-1} + B_2 S^{N-2} + \cdots B_{N-1} S + B_N \qquad (35)$$

$$= (S-Z)(C_1 S^{N-2} + C_2 S^{N-3} + \cdots C_N S + C_{N-1}) + R'$$

For simplification of the expression in Eq (35), let

$$P(S) = C_1 S^{N-2} + C_2 S^{N-3} + \cdots C_{N-1}$$

The substitution of Eq (35) into Eq (31) yields

$$\frac{f(S)}{(S-Z)} = (S-Z)\,P(S) + R' + \frac{R}{(S-Z)} \tag{36}$$

which may be expressed as

$$f(S) = (S-Z)^2\,P(S) + (S-Z)\,R' + R \tag{37}$$

Taking the derivative of Eq (37) with respect to S and denoting the derivative of P(S) as P'(S), produces

$$f'(S) = 2\,(S-Z)\,P(S) + P'(S)\,(S-Z)^2 + R' \tag{38}$$

Again, by letting S take on the value Z, Eq (38) may be written as

$$f'(Z) = R' \qquad \text{(Q.E.D.)} \tag{39}$$

Therefore, substituting the results of Eqs (33) and (39) into Eq (30), the recurrence formula may be written as

$$Z_{k+1} = Z_k - \frac{R}{R'} \tag{40}$$

where R and R' are simply the respective remainders when the polynomial f(s) is divided by (S-Z) and the reduced polynomial resulting from this division is again divided by (S-Z).

By using synthetic division to systematize the repeated calculations of R and R', the tedious labor of evaluating f(Z) and F'(Z) by calculating powers of S and combining terms is avoided. The method of synthetic division is based on the principle of equating coefficients of equal powers in Eqs (21) and (32) to produce the relations

$$\begin{aligned} A_1 &= B_1 \\ A_2 &= B_2 - Z\,B_1 \\ A_3 &= B_3 - Z\,B_2 \\ &\cdots\cdots \\ A_n &= B_n - Z\,B_{n-1} \\ A_{n+1} &= R - Z\,B_n \end{aligned}$$

From these relations, a recurrence formula

$$B_k = A_k + Z\,B_{k-1} \qquad (k = 2, 3, \cdots, N+1) \tag{41}$$

may be used to calculate the coefficients of the reduced polynomial expressed in Eq (32) and the remainder R is given by

$$R = B_{N+1} = A_{N+1} + Z\,B_N \tag{42}$$

In a similar manner, the C coefficients of Eq (35) may be related to the B coefficients as the B coefficients are related to the A coefficients (K=2, 3, . . . N) and the remainder R' is given by

$$R' = C_N = B_N + Z\,C_{N-1} \tag{43}$$

To mechanize the computation for the digital computer, it is helpful to arrange the coefficients in tabular form as in Table I.

Table I
Coefficients Formed by Two Linear Synthetic Divisions

A's	B's	C's
A_1	B_1	C_1
A_2	B_2	C_2
A_3	B_3	C_3
.	.	.
.	.	.
A_{n-2}	B_{n-2}	C_{n-2}
A_{n-1}	B_{n-1}	C_{n-1}
A_n	B_n	C_n
A_{n+1}	B_{n+1}	

By inspecting the synthetic-division recurrence formula expressed in Eq (41), a rule may be determined for the mechanical calculation of the B and C elements of Table I and may be expressed as:

RULE 1: To find an element, add to its left-hand neighbor, Z times its upward neighbor.

As previously shown, $A_1=B_1=C_1$; therefore, when applying RULE 1, the first elements, B_1 and C_1, are obtained by considering the upward neighbors B_0 and C_0 are equal to zero.

As stated earlier, the recurrence formula expressed in Eq (40) involves repeated calculations of R and R' for each succeeding value of Z_i. Thus, if an initial approximation of Z is used, the corrected approximation Z^* is given by

$$Z^* = Z - \frac{R}{R'} \tag{44}$$

The iterative process is repeated with Z replaced by Z^* until convergence is achieved, i.e., $Z^*-Z=0$, which means Z^* is a root and the remainder term $R=B_{n+1}=0$. Of course, the requirement that $Z^*-Z=R=0$ is not practical since perfect convergence would require an infinite number of iterations; therefore, the criterion for convergence used in this thesis is that the remainder R, compared to the equal-power coefficient of the original polynomial, be smaller than a specified tolerance. The convergence criteria may be expressed mathematically as

$$\left|\frac{R}{A_{N+1}}\right| = \left|\frac{B_{N+1}}{A_{N+1}}\right| < TOLERANCE = 10^{-NO} \qquad (45)$$

Where NO is the number of significant figures retained in the calculation.

Bairstow Method

As stated earlier in the chapter, the Bairstow Method is used to find a quadratic factor S^2+PS+Q of the polynomial f(S), where the factor is the product of a pair of linear roots of the polynomial and the roots are real or complex depending on the values of P and Q of the quadratic factor.

The Bairstow Method involves an iterative process similar to the Newton-Raphson Method except that the synthetic division is performed by a quadratic factor rather than by a linear factor.

Again, by expressing the polynomial as

$$f(S) = A_1 S^N + A_2 S^{N-1} + \cdots A_N S + A_{N+1} \qquad (46)$$

and dividing the polynomial by the factor S^2+PS+Q, the quotient may be expressed as a polynomial reduced in degree by two and a remainder term:

$$\frac{f(S)}{S^2+PS+Q} = B_1 S^{N-2} + B_2 S^{N-3} + \cdots B_{N-2} S + B_{N-1} + \frac{RS+U}{S^2+PS+Q} \quad (47)$$

Multiplying Eq (47) by the quadratic factor S^2+PS+Q produces

$$f(S) = (S^2+PS+Q)(B_1 S^{N-2} + B_2 S^{N-3} + \cdots B_{N-2} S + B_{N-1}) + RS + U \quad (48)$$

Equating coefficients of equal powers of S in Eq (46) and (48) produces the relations

$$\begin{aligned}
A_1 &= B_1 \\
A_2 &= B_2 + PB_1 \\
A_3 &= B_3 + PB_2 + QB_1 \\
&\cdots\cdots \\
A_k &= B_k + PB_{k-1} + QB_{k-2} \\
&\cdots\cdots \\
A_n &= R + PB_{n-1} + QB_{n-2} \\
A_{n+1} &= U + QB_{n-1}
\end{aligned}$$

From these relations, the recurrence formula

$$B_k = A_k - PB_{k-1} - QB_{k-2} \qquad (k = 2, 3, \cdots, N+1) \quad (49)$$

with $B_0 = 0$ and $B_1 = A_1$, may be used to calculate the B coefficients

of the reduced polynomial expressed in Eq (47), where the remainder coefficients are

$$R = B_N = A_N - P B_{N-1} - Q B_{N-2} \tag{50}$$

$$U = A_{N+1} - Q B_{N-1} \tag{51}$$

From the recurrence formula in Eq (49) the remainder U may be expressed as

$$U = B_{N+1} + P B_N \tag{52}$$

For the quadratic S^2+PS+Q to be a factor of the polynomial f(s), the remainders $R=B_n$ and $U=B_{n+1}+PB_n$, which are functions of P and Q as shown in Eqs (50) and (52), must equal zero. Therefore, solving the equations

$$R(P, Q) = 0 \quad \text{AND} \quad U(P, Q) = 0$$

by using the Newton-Raphson iteration, which again requires a second synthetic division where the C's are related to the B's as the B's are to the A's, the recurrence formulas for calculating corrections to P_0 and Q_0 of the quadratic factor S^2+PS+Q are (Ref 3:472-475)

$$P_{k+1} = P_k + \frac{B_N C_{N-1} - B_N C_{N-2}}{C_{N-1} C_{N-1} - \bar{C}_N C_{N-2}} \tag{53}$$

$$Q_{K+1} = Q_K + \frac{B_{N+1} C_{N-1} - B_N \bar{C}_N}{C_{N-1} C_{N-1} - \bar{C}_N C_{N-2}} \tag{54}$$

where

$$\bar{C}_N = C_N - B_N = - PC_{N-1} - QC_{N-2} \tag{55}$$

Again, to mechanize the computation of the B and C coefficients, it is helpful to arrange the coefficients in tabular form as in Table II.

Table II
Coefficients Formed by Two Quadratic Synthetic Divisions

A's	B's	C's
A_1	B_1	C_1
A_2	B_2	C_2
A_3	B_3	C_3
.	.	.
.	.	.
A_{n-2}	B_{n-2}	C_{n-2}
A_{n-1}	B_{n-1}	C_{n-1}
A_n	B_n	$\bar{C}_n$
A_{n+1}	B_{n+1}	

By inspecting the synthetic division recurrence formula expressed in Eq (49) and by noting the definition of $\bar{C}_n$ in Eq (55), a rule may be determined for the mechanical calculation of the B and C elements of Table II and may be expressed as:

RULE 2: To find an element, except $\bar{C}_n$, subtract from its left-hand neighbor, P times its first upward neighbor and Q times its second upward neighbor. $\bar{C}_n$ is calculated in the same way except the left-hand neighbor is considered to be zero.

When applying RULE 2, the elements B_1, B_2, and C_2 are obtained by considering that the missing elements B_0, B_{-1}, C_0, and C_{-1} are equal to zero

As in the Newton-Raphson iteration, the use of the recurrence formulas expressed in Eqs (53) and (54) involves repeated calculations of the coefficients for each succeeding value of P_i and Q_i. Thus, if initial estimates of P and Q are used, the corrected estimates P^* and Q^* are calculated by Eq (53) and (54) and the process is repeated with P and Q replaced by P^* and Q^* respectively. The iteration is continued until convergence is achieved, i.e., $P^*-P=0$ and $Q^*-Q=0$. This means that P^* and Q^* are the correct values for S^2+PS+Q to be a factor of the polynomial and the remainders $R=B_N$ and $U=B_{N+1}+PB_N$ are equal to zero. Since R and U are zero, it is obvious from the relation $U=B_{N+1}+PB_N$ that B_{N+1} must equal zero for exact convergence. The practical limitation, discussed earlier concerning the convergence of the Newton-Raphson iteration applies to the Bairstow iteration; therefore, the convergence criteria may be expressed mathematically as

$$\left|\frac{B_N}{A_N}\right| < TOLERANCE = 10^{-NO} \tag{56}$$

$$\left|\frac{B_{N+1}}{A_{N+1}}\right| < TOLERANCE = 10^{-NO} \tag{57}$$

Loss of Significance

Although the classes of polynomials for which the Bairstow and Newton-Raphson Methods fail to converge do not have any appreciable number in common, failure to achieve convergence can occur because of the loss of significance in the synthetic division. Experience has

shown that in order to avoid loss of significance the iteration must converge on the root of the smallest or largest magnitude depending on the polynomial. A substitution of variables can be made in the polynomial to ensure that the iteration will always converge on the root with the smallest magnitude, thereby avoiding loss of significance (Ref 2: 645-646). Assume first that the iteration always converges on the smallest root. Then the initial estimates P_k and Q_k in Eqs (53) and (54) would be zero. If the iteration should converge on the largest root, the polynomial equation f(S)=0 can be transformed with the change of variable

$$X = \frac{1}{S}$$

which when substituted in Eq (46) produces

$$\frac{A_1}{X^N} + \frac{A_2}{X^{N-1}} + \cdots \frac{A_N}{X} + A_{N+1} = 0 \qquad (58)$$

Multiplying both sides of Eq (58) by X^n and reversing the sequence of terms yields

$$g(x) = A_{N+1} X^N + A_N X^{N-1} + \cdots A_2 X + A_1 = 0 \qquad (59)$$

Therefore, it can be seen that reversing the coefficients is equivalent to transforming the roots to their reciprocals.

The iteration will now converge on the smallest root in the transformed polynomial equation g(X)=0 if the iteration converged on the

largest root in the original polynomial f(S)=0; thus, the iteration may always be initiated with the convenient estimates to P and Q of $P_0=Q_0=0$. A similar analysis may be made for the Newton-Raphson iteration; however, since the bulk of computation goes into the Bairstow iteration, the decision to reverse the sequence of coefficients is based on the condition which will make the Bairstow iteration more likely to converge. A detailed analysis of the computation of the coefficients conducted by K. W. Ellenberger (Ref 2:646) shows that the criterion for reversing the sequence of coefficients depends on the quantity

$$\left|\frac{A_2}{A_1}\right| - \left|\frac{A_N}{A_{N+1}}\right| \tag{59}$$

If this quantity is negative, the sequence should be reversed.

Scaling Polynomial

To improve the rate of convergence and to help prevent the numbers in the computation from becoming too large for the capacity of the computer (10^{50}), the polynomial is scaled by dividing the polynomial by the geometric mean of the coefficients, which may be expressed as

$$GM = \left(\prod_{i=1}^{N+1} A_i\right)^{\frac{1}{N+1}} \qquad (A_i \neq 0) \tag{60}$$

Where GM is the geometric mean and N is the degree of the polynomial. Computation of the geometric mean GM may be systematized by taking

the natural logarithm of Eq (60) to produce

$$LOG_e\ GM = \frac{1}{N+1}\left[LOG_e \prod_{i=1}^{N+1} A_i \right] \tag{61}$$

which may be rewritten as

$$LOG_e\ GM = \frac{1}{N+1}\left[\sum LOG_e\ |A_i| \right] \tag{62}$$

Letting E equal the quantity on the right side of Eq (62), yields

$$GM = e^{E} \tag{63}$$

IV Development of Program

As shown in Chapter II, the problem of computing the root locus can be transformed into a format of repeated solutions of the roots of the characteristic equation expressed as a polynomial equation of degree N. By using the numerical techniques for the solution of a polynomial equation of degree N, which were systematized for the digital computer in Chapter III, a system of programming may be used to develop a set of related statements known as a program.

System of Programming

The IBM FORTRAN system of programming is used in order that the arithmetic statements in the program may be written as mathematical formulas which closely resemble the formulas as written in Chapter II and III. As the name FORTRAN (FORmula TRANslation) implies, the formulas in the arithmetic statements, along with the control statements and input/output statements are transformed automatically into Basic Machine Language (Ref 4: 1-2). Therefore, a minimum of knowledge of the computer is required of the engineer to understand and to execute the program. Although there are some disadvantages or limitations to the FORTRAN system which affect the program as developed in this thesis (as discussed later), the objective of reducing the time and effort required of the engineer to obtain the root locus led to the choice of the FORTRAN programming system.

Composite Program

The writing of a program for a digital computer is facilitated

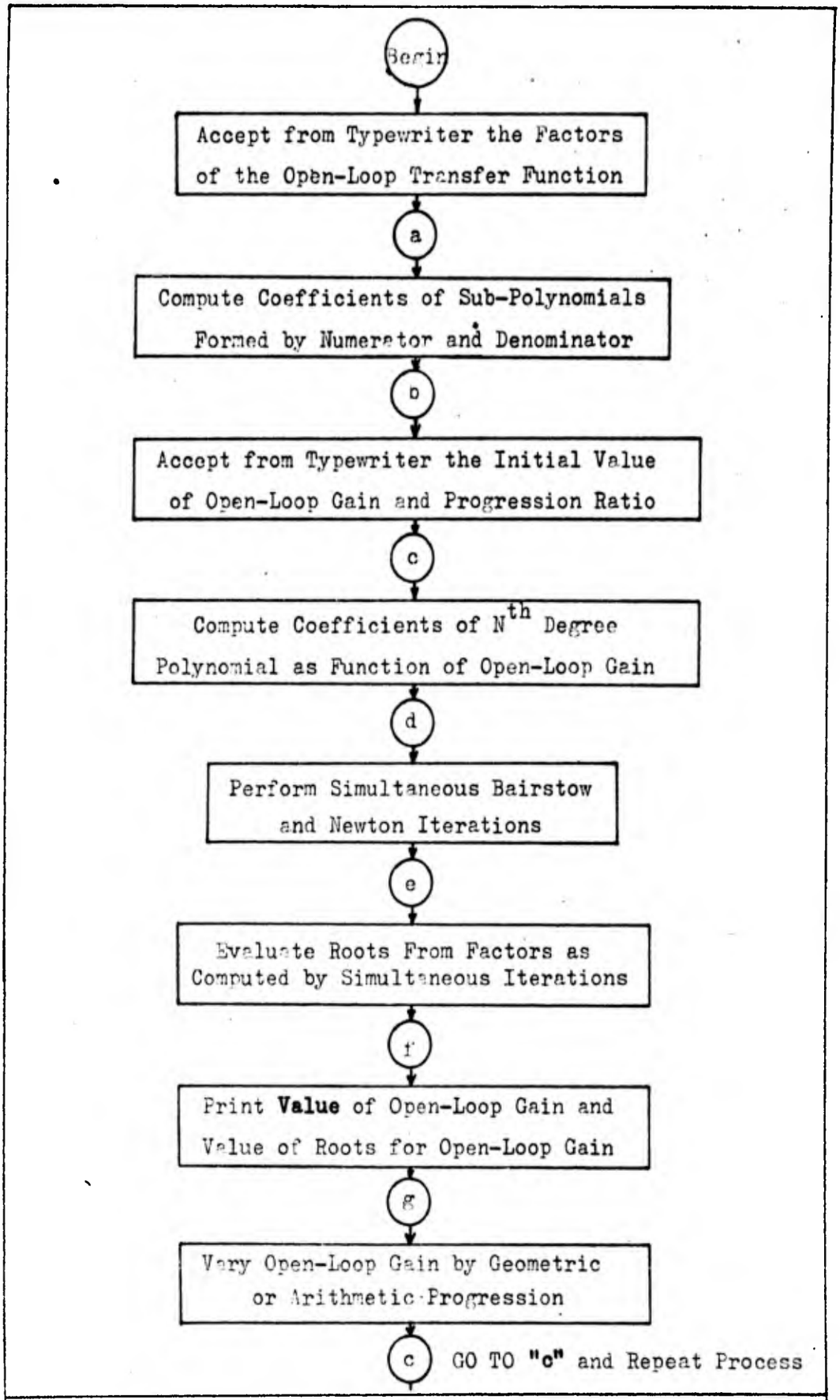


FIG. 4

GENERAL FLOW CHART OF COMPOSITE PROGRAM

by outlining the required computations and the order of execution of the operations to accomplish these computations on a generalized flow-chart, as shown in Fig. 4. A set of related statements may be written so that the computer executes the indicated operations in the order which will produce the desired solutions.

The composite or total program, which was written to perform the operation as outlined in Fig. 4, exceeded the memory capacity of the IBM 1620 computer (20,000 memory cells) by approximately two thousand spaces.

The relatively large requirement of memory space for the composite program is due to the fact that considerably more memory space is required in FORTRAN than in some of the other systems of programming, which are written in more direct machine language. However, in order that the more convenient FORTRAN system may be used, the composite program is divided into two related parts. By referring to Fig. 4, it can be seen that the program may be divided at point "b" of the flow-chart with a minimum of modification required. The first part of the program, which is referred to as the PART 1 program, is designed to compute the number and value of the coefficients of the sub-polynomials [expressed in Eq (15)], which are then used as input data for the PART 2 program. The PART 2 program is designed to read the input data, as computed in the PART 1 program, and to perform the remainder of the operations as shown in Fig. 4. The only required modification to the program, as outlined in Fig. 4, is that a "punch statement" be inserted at the end of the PART 1 program and a "read

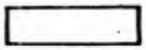
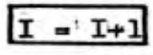
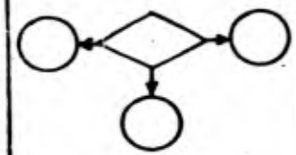
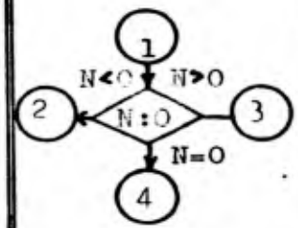
DEFINITION		EXAMPLE	
SYMBOL	MEANING	SYMBOL	MEANING
	SPECIFIC POINT SYMBOL Program is at Point in the Circular Block		Program is at Statement 12 in Program Execution
	OPERATION SYMBOL Computer Performs the Operation Indicated in Rectangular Block		Computer Adds 1 to Quantity in Address I and Places New Quanity Back in Address I
	COMPARISON SYMBOL Computer Compares Quantities in Diamond Shaped Block and Branches to Specific Points as Indicated		Program Moves to Comparison Block From Point 1 where Quantity at Address N is compared to 0 If N < 0 Program Branches to 2 If N > 0 Program Branches to 3 If N = 0 Program Branches to 4

FIG. 5

Definition and Example of Block Symbols Used in Detail Flow Charts of PART 1 Program and PART 2 Program

statement" be inserted at the beginning of the PART 2 program. The only disadvantage (which is slight) to dividing the program into two parts is that the operator must mount two object tapes, rather than one, for program execution. However, a considerable advantage is gained by dividing the program into two parts: The operator is required to enter by typewriter the factors of the open-loop transfer function into the PART 1 program only one time, yet the data punched by PART 1 may be used in PART 2 for repeated solutions of the root locus at any later time convenient to the operator.

Because of the limited memory space in the IBM 1620 computer relative to the larger computers, the program developed in this thesis is limited to the solution of a polynomial of degree twenty. An upper limit must be set, in order that a dimension statement can be used at the beginning of the program to reserve memory space for the subscripted variables.

The PART 1 and PART 2 programs, along with detailed flow-charts, are provided on "gate-folds" in Appendices A and B, respectively, for the convenience of the reader while reading the discussion of the development and execution of the two parts of the program. The symbols used in the flow-charts are defined and examplified in Fig. 5.

PART 1 Program

The function of the PART 1 program is to compute, print, and punch on data tape the coefficients of the two sub-polynomials expressed in Eq (15) as

$$E_1 S^N + E_2 S^{N-1} + \cdots E_N S + E_{N+1} + K\left[D_1 S^M + D_2 S^{M-1} + \cdots D_M S + D_{M+1}\right] = 0 \quad (15)$$

Since the computation of the E and D coefficients of each of the sub-polynomials involves the multiplication of linear and quadratic factors, the PART 1 program is designed to: (1) compute the coefficients formed by the numerator factors of the open-loop transfer function, (2) to transfer these coefficients to the D addresses, and (3) to use the same routine to compute and store in the E addresses the coefficients formed by the denominator factors of the open-loop transfer function. After the coefficients are computed, the value of the coefficients are printed, for the convenience of the operator. Also, the value and number of coefficients are punched on data tape to be used as input data for the PART 2 program.

Referring to Fig. 6, Appendix A, the first statement is a dimension statement which reserves memory spaces for the factors of the open-loop transfer function where the linear factors are expressed as

$$S + R(1),\ S + R(2),\ \cdots,\ S + R(20)$$

and the quadratic factors are expressed as

$$S^2 + X(1)S + Y(1),\ S^2 + X(2)S + Y(2),\ \cdots,\ S^2 + X(10)S + Y(10)$$

Both the numerator and denominator of the open-loop transfer function may be any combination of linear and quadratic factors such that the product of the factors does not generate a polynomial which exceeds the degree of twenty.

In the program, in Appendix A, M is the number of quadratic

factors and N is the number of linear factors. An inspection of Eq (12) shows that there are four possible combinations of factors which are:

1. No quadratic factors and no linear factors (M=0, N=0). This combination applies only to the numerator, in which case the numerator is simply K.

2. No quadratic factors and from one to twenty linear factors (M=0, $0<N\leq 20$).

3. No linear factors and from one to ten quadratic factors (N=0, $0<M\leq 10$).

4. Both linear and quadratic factors ($0<2M+N\leq 20$).

A perusal of the flow-chart and program shows that a combination of IF statements and COMPUTED GO TO statements are used to transfer the order of the program execution based on the values of N and M. For each of the four possible combinations of N and M there are four branches and all of these branches terminate or join together at statement 135. At statement 135, the counter IO is compared to 1 and 2. As shown on the flow-chart, if IO=1 as it would after the computation of the coefficients formed by the numerator factors, the program branches to statement 131 where the coefficients are transferred to the D addresses and the number of coefficients is transferred to the LOW address. At this point, the program branches back to statement 150 where the counter IO is increased to 2 and the process is repeated to compute the coefficients formed by the denominator factors. However, since IO is equal to 2 when the program reaches statement 135, the COMPUTED GO TO statement causes the program to branch to statement 133 where the D and E coefficients are printed at the typewriter and punched on data

tape for input to the PART 2 program. Each of the four branches for the four possible combinations of N and M are discussed as follows:

No Quadratic, No Linear Factors (M=0, N=0). After the operator types in the value of N=0 and M=0, the program branches to statement 102 and then to statement 103 where the only coefficient E(1) and the number of coefficients M are set equal to 1. The program branches to statement 135, which is the termination point of the branch for this combination.

Linear Factors, No Quadratic Factors ($0<N\leq 20$, M=0). After the operator types in the value of N and M, the program branches to statement 104 where the counter NO is increased to 3 and the computer prints "3" to notify the operator to enter the values of the linear factors R(1), R(2), . . . , R(N). SENSE SWITCH #4 affords the operator a method for correcting typing errors when entering the linear factors R(N). After the correct value of the linear factors are entered, M the number of coefficients (M=N+1) and the value of the coefficients are computed. The addresses R(1), E(2), . . ., E(K), . . ., E(M) are initialized to zero and a "nest" of DO LOOPS is used to generate the coefficients by the systematized multiplication of linear factors. Since the factors are normalized (coefficients of the S term equal to unity), the highest order coefficient E(1) is set equal to 1.

The generalized multiplication formula used in the nest of DO LOOPS is

$$E(I+1) = E(I+1) + E(I) * R(J)$$

TABLE III

COEFFICIENTS FORMED BY SYSTEMATIZED MULTIPLICATION OF LINEAR FACTORS

J	K	I	E(1)	E(2)	E(3)	E(4)
			1.0	0.0	0.0	0.0
1	1	1	#	R(1)	#	#
2	1	2	#	#	R(1) * R(2)	#
2	2	1	#	R(1) + R(2)	#	#
3	1	3	#	#	#	R(1)*R(2)*R(3)
3	2	2	#	#	R(1)*R(2)+R(3)[(R(1)+R(2)]	#
3	3	1	#	R(1)+R(2)+R(3)	#	#

NOTE: The Symbol # indicates the Quantity is the Same as Above

The symbol * denotes multiplication in the computer program and is also frequently used for the same purpose in the following discussions throughout the remainder of the thesis. The multiplication formula for forming the coefficients was derived by listing in tabular form, as in Table III, the terms generated by multiplying linear factors and then by observing the general pattern of these terms. For example, consider the multiplication of the linear factors

$$[S + R(1)] \quad [S + R(2)] \quad [S + R(3)]$$

Since the product of these three factors produces a polynomial of four terms, the degree of the polynomial is three and the number of coefficients is four.

Referring to the program and to Table III, the E addresses are initialized to

$$E(2) = E(3) = E(4) = 0$$

and the execution of the nest of DO LOOPS causes the coefficients to be computed and stored in the E addresses as indicated in Table III so that the values of the coefficients are

$$E(1) = 1.0$$

$$E(2) = R(1)+R(2)+R(3)$$

$$E(3) = R(1)*R(2)+R(3)*[R(1)+R(2)]$$

$$E(4) = R(1)*R(2)*R(3)$$

They are the correct coefficients of the polynomial as may be verified by long-hand multiplication.

TABLE IV

COEFFICIENTS FORMED BY SYSTEMATIZED MULTIPLICATION OF QUADRATIC FACTORS

I	J	L	E(1)	E(2)	E(3)	E(4)	E(5)	E(6)
			0.0	1.0	0.0	0.0	0.0	0.0
1	1	2	#	#	X(1)	Y(1)	#	#
2	1	4	#	#	#	#	Y(1)*X(2)+X(1)*Y(2)	Y(1)*Y(2)
2	2	2	#	#	X(1)+X(2)	Y(1)+X(1)*X(2)+Y(2)	#	#

NOTE: The Symbol # Indicates the Quantity is the Same as Above

Quadratic Factors, No Linear Factors ($0<M\leq 10$, $N=0$). After the values of N and M are entered, the program branches to statement 110 and since N=0, the program branches to statement 111. As shown on the flow-chart, the counter MO is set equal to 1 and the program branches to statement 119. Since MO=1, the program branches to statement 120 where the counter NO is increased to 3 and the computer prints "3" to notify the operator to enter the value of the quadratic factors X(1), Y(1), . . . X(1), . . . X(M) Y(M). After the factors are entered, the program branches to statement 123 where a nest of two DO LOOPS is used to generate the coefficients by the systematized multiplication of quadratic factors in a manner similar to the multiplication of linear factors previously described.

Since the factors are quadratic, two generalized formulas for forming the coefficients are required in the nest of DO LOOPS:

$$E(L+2)=E(L+2)+E(L+1)*X(1)+E(L)+Y(I)$$

$$E(L+1)=E(L+1)+E(L)*X(1)+E(L-1)*Y(I)$$

Again, the formulas were derived by multiplying quadratic factors and then observing the pattern of the terms generated as listed in the tabular form in Table IV.

For example, consider the multiplication of the quadratic factors

$$\left[S^2+SX(1)+Y(1)\right] \quad \left[S^2+SX(2)+Y(2)\right]$$

Since the product of two quadratic factors is a polynomial of five terms, the degree of the polynomial is four and the number of coefficients is five.

The process of the nest of DO LOOPS in executing the two generalized formulas requires that the E addresses be initialized to zero

with E(2) set equal to 1.0; therefore, the total number of E addresses is

$$K=2*M+2+N$$

where M is the number of quadratic terms and N is the number of linear terms. Of course, in this case N=0; however, N is included in the formula since the same routine is used when $N\neq 0$ (as explained in the next case).

Referring to the program and to Table IV, the execution of the DO LOOPS causes the coefficients to be stored in the E addresses as indicated in Table IV. However, the number of coefficients $\bar{M}$ is computed as

$$\bar{M}=2*M+1=5$$

where M is the number of quadratic terms; therefore, the coefficients must be transferred back one address so that the coefficients are in the correct addresses as

$$E(1) = 1.0$$
$$E(2) = X(1)+X(2)$$
$$E(3) = Y(1)+X(1)*X(2)+Y(2)$$
$$E(4) = Y(1)*X(2)+X(1)*Y(2)$$
$$E(5) = Y(1)*Y(2)$$

These coefficients may be used in the remainder of the program as defined for the sub-polynomials in Eq (15).

Both Quadratic and Linear Factors ($0<2M+N\leq 20$). As in the preceeding case, the program branches to statement 110 and since N>0, the program branches to statement 121 where the counter NO is again

increased to 3 and the computer prints "3" to notify the operator to type in the linear factors followed by the quadratic factors. After the factors have been entered, the quadratic factors are multiplied as in the preceeding case. However, since N>0, the counter MO is set equal to 2 and, referring to the flow-chart, it is seen that the program branches to statement 128 where the product of the quadratic factors is further multiplied by the linear factors (one at a time) to generate the coefficients of the sub-polynomial.

After the D and E coefficients are computed, printed, and punched on data tape, the execution of the PART 1 program is finished and the data tape is used for the input to the PART 2 program.

PART 2 Program

As stated previously, the function of the PART 2 program is to perform the remainder of the operation shown in Fig. 4, beginning at point "b". Therefore, as shown in the program and flow-chart in Fig. 7, Appendix B, the number and value of the D and E coefficients, as computed in PART 1, are read from data tape and stored in memory. A counter NO is set equal to 1 and the computer prints "1" to notify the operator to enter the initial value of the sensitivity K and the value of the geometric progression ratio by which the sensitivity K is to be varied. As shown on the flow-chart, the addresses used for the sensitivity K and the geometric progression ratio are G and GRAT, respectively. The standard symbol K can not be used for the sensitivity K since K is reserved for "fixed-point" variables

in FORTRAN and the sensitivity is a "floating-point" variable as used in the program. Therefore, the sensitivity is referred to as G throughout the entire program and also in the discussion of the program.

After the initial value of G is entered, the A coefficients as expressed in Eq (16) are formed and stored in the addresses A(1), A(2), . . . , A(I), . . . , A(M):

$$A_1 S^N + A_2 S^{N-1} + \cdots A_N S + A_{N+1} = 0 \tag{16}$$

The value M=MOW is equal to the number of the E coefficients. It is also equal to the number of A coefficients since the number of E's is always equal to or greater than the number of D's. A counter NON is set equal to 0 and the computer prints "0" followed by the value of gain G. The 0 is printed preceeding the value of G to assist the operator in readily distinguishing between the value of gain and roots of the polynomial as printed by the computer.

To determine if any of the roots are zero, the last coefficient A(M) is compared to zero. If the last coefficient is zero, the polynomial equation is of the form

$$A_1 S^N + A_2 S^{N-1} + \cdots A_N S + 0 = 0 \tag{64}$$

from which an S may be factored to yield

$$S(A_1 S^{N-1} + A_2 S^{N-2} + \cdots A_N) = 0 \tag{65}$$

An inspection of Eq (65) shows that if the last coefficient is zero, one of the roots is zero (S=0) and the degree of the polynomial equation is reduced by one. After the zero roots are found and printed (if any exist), the degree of the polynomial equation N is tested by comparing M=N+1 to 3. As shown on the flow-chart if M>3, the program branches to statement 7 where numerical techniques are used to compute the roots. If M=3, the polynomial is a simple quadratic equation and the program branches to statement 71 where the roots are evaluated using the quadratic formula. If M<3, the program branches to statement 78 where M is compared to 2. If M=2, the polynomial is a linear equation, $A_1S+A_2=0$, and the root is simply $S=-A_2 / A_1$. Since M is never less than 2; the polynomial is solved and the program branches to statement 81 where the G is varied (as explained later) and the program branches back to statement 4 to repeat the process. The case of M=3 or M=2 is relatively simple as shown; however, when M>3, the program branches to statement 7 where the systematized procedures developed in Chapter III are used to compute and to store in address CAT the geometric mean of the A coefficients. A DO LOOP is used to perform the recommended scaling of the polynomial.

Reversal of Coefficients. After the scaling is accomplished, the quantity as expressed in Eq (59)

$$\left|\frac{A_2}{A_1}\right| - \left|\frac{A_N}{A_{N+1}}\right| \qquad (59)$$

is computed to determine if the coefficients should be reversed. As shown on the flow-chart, if the quantity in Eq (59) is less than zero, the program branches to statement 19. where the sequence of coefficients is reversed and the counter NO is set equal to -1. If the quantity in Eq (59) is positive or zero, the program branches to statement 22 where the counter NO is set equal to +1. The NO counter is used later in the program to determine if the roots or the reciprocal of the roots should be evaluated, based on the negative or positive value of the NO counter.

Convergence Criteria. As specified in Chapter III, the criteria for convergence of the iterative processes is that the ratio of the remainder coefficients in the synthetic division to the equal-power coefficients in the polynomial be less than a tolerance which was expressed in Eqs (45), (56), and (57) as

$$\left|\frac{B_{N+1}}{A_{N+1}}\right| < TOLERANCE = 10^{-NO} \tag{45}$$

$$\left|\frac{B_{N}}{A_{N}}\right| < TOLERANCE = 10^{-NO} \tag{56}$$

$$\left|\frac{B_{N+1}}{A_{N+1}}\right| < TOLERANCE = 10^{-NO} \tag{57}$$

Where NO is the number of significant figures retained in the calculations. In the FORTRAN system, only eight figures are retained in the calculations with the other figures being truncated, rather than being rounded-off. Although the method of using the simultaneous Bairstow and Newton iterations has never failed to converge, the slight loss

of accuracy caused by the truncation has resulted in the iteration failing to converge until the tolerance requirement was reduced from 10^{-8} to 10^{-7}. In an experimental program which was developed during the research for this thesis, a counter was inserted to record the tolerance used to achieve convergence. Approximately fifty percent of the iterations converged with 10^{-8} as the tolerance. Except for a few isolated cases, the remaining fifty percent converged with 10^{-7} as the tolerance. For the few isolated cases, the iteration always converged with a tolerance of 10^{-6}. In most cases the roots of a polynomial equation computed with a tolerance of 10^{-8} agreed exactly with the roots of the same polynomial equation computed with a tolerance of 10^{-7}. In those cases where the roots did not agree exactly, the difference was never reflected in more than the three right-most figures, and usually only in the eighth figure. Therefore, to reduce the computation time, the program is designed to use an initial tolerance of 10^{-7} with provisions for automatic reduction of the tolerance requirement to 10^{-6} if the iterative process fails to converge after twenty iterations.

The practical limit of twenty iterations was established by inserting another counter in the experimental program which recorded the number of iterations required for convergence. In almost every trial solution, convergence occured within ten iterations or failed to converge at all with a tolerance of 10^{-8}; however, with the tolerance requirement reduced to 10^{-7}, convergence usually occured within ten iterations and always converged in less than twenty iterations.

The problem of division by zero occurs if the coefficient A_N of Eq (56) is equal to zero. Therefore, a procedure must be established to test for convergence without attempting division by zero. Since the remainder term $R=B_{M-1}$ in Eq (56) must be small compared to A_N, which in this case is equal to zero, the convergence criteria is satisfied if R is reduced to zero. However, because of the truncation mentioned earlier, the best possible convergence criteria is

$$\left|B_{M-1}\right| < 10^{-8}$$

For the reasons given previously for using a tolerance of 10^{-7} instead of 10^{-8}, the convergence criteria used, as shown on the flow-chart at statement 34, is

$$\left|B_{M-1}\right| < 10^{-7}$$

Simultaneous Bairstow and Newton Iteration. Beginning with a tolerance of 10^{-7} and using twenty iterations, the program is designed to perform the Bairstow and Newton iterations simultaneously. "Simultaneous", as used here, means that a Bairstow iteration is performed and a consecutive Newton iteration is performed before attempting another Bairstow iteration with corrected values of P and Q. Thus full advantage is taken of the fact that convergence may be achieved for one method but may fail for the other, depending on the class of polynomial.

As shown on the flow-chart, the initial estimates for both iterations are set to P=Q=R=0. The Bairstow iteration is performed first by computing the coefficients of the mechanized synthetic division as shown in Table II, Chapter III. After the coefficients

are computed, the program progresses from statement 26 where the recurrence formulas

$$P_{K+1} = P_K + \frac{B_N C_{N-1} - B_N C_{N-2}}{C_{N-1} C_{N-1} - \bar{C}_N C_{N-2}} \tag{53}$$

$$Q_{K+1} = Q_K + \frac{B_{N+1} C_{N-1} - B_N \bar{C}_N}{C_{N-1} C_{N+1} - \bar{C}_N C_{N-2}} \tag{54}$$

are used to compute the corrected estimates for P and Q which are referred to as PI and QI, respectively, in the flow-chart. The recurrence formulas as expressed in the program differ slightly in symbolics from Eqs (53) and (54) in that M, the number of coefficients, is used as the reference subscript. The subscript N, the degree of the polynomial, is used in Eqs (53) and (54). Remembering that M=N+1, it can be seen that the coefficients used in the program are identical to those used in Eqs (53) and (54). The problem of division by zero occurs again when the denominator of the correction formulas is equal to zero (This problem will be discussed in detail later). If the denominator is not zero, the program branches to statement 29 where PI and QI are computed. After the computation of PI and QI, the Bairstow iteration is tested for convergence. If the iteration has converged, the program branches to statement 60 where the quadratic factor S^2+PIS+QI is evaluated (as explained later) to yield a pair of roots of the polynomial equation. If the Bairstow iteration fails to converge, the program branches to statement 44 where the recurrence formula

$$Z^* = Z - \frac{R}{R'} \tag{44}$$

is used to compute the corrected estimates to the real root Z which is denoted as R in the program. Also, the corrected value Z^* is expressed as RI in the program and, by remembering that M=N+1, Eq (44) can be written as in the program. As in the Bairstow iteration, the problem of division by zero occurs if the coefficient C_N in Eq (44) is zero. The problem will be discussed later when the problem of division by zero in the Bairstow iteration is discussed. However, if $C_N \neq 0$, the corrected estimate RI is computed and the Newton iteration is tested for convergence. If the iteration converges, the program branches to statement 80 where the linear factor S-RI=0 is evaluated to yield the real root S=RI.

Denominator of Recurrence Formulas Equal Zero. When the denominator of Eqs (44), (53), and (54) are equal to zero a procedure must be established for computing the corrected estimates to R, P, and Q which will circumvent the division by zero. An analysis of the computation of these denominators reveals that the denominators become zero when one or more of the coefficients (other than A_M) are zero.

For example, consider the polynomial equation

$$S^4 - 16 = (S^2 + 4)(S^2 - 4) = 0 \qquad (66)$$

in which the coefficients are

A_1=1.0
A_2=0.0
A_3=0.0
A_4=0.0
A_5=-16.0

For the Bairstow iteration, the results of performing the two quadratic synthetic divisions (applying RULE 2, Chapter III), with P=Q =0, are tabulated in Table V.

TABLE V

COEFFICIENTS RESULTING FROM TWO QUADRATIC SYNTHETIC DIVISIONS

P, Q =		0, 0		0, 4	
Sub-scripts	A's	B's	C's	B's	C's
1	1	1	1	1	1
2	0	0	0	0	0
3	0	0	0	-4	-4
4	0	0	0	0	0
5	-16	-16		0	

The formula for the denominator is written in the program as

$$DENOM = C(M-2) * C(M-2) - C(M-1) * C(M-3) \qquad (67)$$

Substituting the values of the coefficients and M=5 in Eq (67) gives

$$DENOM = C(3) * C(3) - C(4) * C(2) \qquad (68)$$

As shown, the denominator is zero. However, if the initial value of Q is taken as

$$Q = \left|B(M)\right|^{2/N} = 16^{1/2} = 4 \qquad (69)$$

and the synthetic division is performed with starting values of P=0 and Q=4, the B and C coefficients are changed as shown in Table V.

With these coefficients, the value of the denominator becomes

$$DENON = (-4)^2 - 0*0) = 16$$

Using the recurrence formulas as expressed in the program, the computer now computes the corrected values of P and Q as

$$PI = 0 + \frac{0*(-4) - 0*0}{16} = 0 \qquad (70)$$

$$QI = 4 + \frac{0*(-4) - 0*0}{16} = 4 \qquad (71)$$

Since the initial values of P and Q are exactly the same as the corrected values PI and QI and the remainder coefficients $B_4 = B_5 = 0$, the iteration has converged. The resulting quadratic factor is

$$S^2 + PIS + QI = S^2 + 4$$

which is known to be correct, as shown in Eq (66).

Although it is conceivable that there are classes of polynomials other than those with zero coefficients (unknown to the author) for which the denominator becomes zero, the procedure of initiating the starting value of Q by means of Eq (69) produces immediate convergence for many of the classes of polynomials with zero coefficients. For other classes tested in which the denominator was zero, but the value of Q computed in Eq (69) was not the exact value of QI (as in the above example), the iteration still converged to the correct values of PI and QI. In practice, the procedure (statement 30 of program) of restarting the iteration with a value of Q other than zero has always produced convergence.

For the Newton Method, the results of performing the two linear synthetic divisions on Eq (66) (applying RULE 1, Chapter III), with R=0, are as tabulated in Table VI.

TABLE VI

COEFFICIENTS RESULTING FROM TWO LINEAR SYNTHETIC DIVISIONS

R = 0			
Subscripts	A's	B's	C's
1	1	1	1
2	0	0	0
3	0	0	0
4	0	0	0
5	-16	-16	

Since the denominator is simply C_N, it is obvious from Table VI that $C_N=C_4=0$. As shown on the flow-chart, the procedure is to set the counter IO equal to zero and to branch to the next Bairstow iteration. An inspection of the example polynomial equation in Eq (66) shows that the polynomial is composed of quadratic factors which can be found as shown by the Bairstow Method; therefore, only the Bairstow Method is used to compute roots when the denominator C_N of the Newton recurrence formula is zero. As shown on the flow-chart at statement 44, the COMPUTED GO TO statement senses IO=1 and the program completely by-passes the Newton iteration.

Evaluating Roots of Quadratic Factor. As stated earlier, the program branches to statement 60 after a quadratic factor has been computed by the Bairstow iteration. At statement 60, the counter NO is compared to zero to determine if the sequence of coefficients was reversed. If the sequence was not reversed, the counter NO was set equal to +1 (as previously described) and the program branches to statement 62 where the roots are evaluated. However, if the sequence was reversed, the counter NO was set equal to -1 and the program branches to statement 61 where the substitution of variables is reversed by expressing the quadratic factor as

$$S^2 + \frac{PI}{QI} S + \frac{1.0}{QI} = 0 \qquad (72)$$

As shown on the flow-chart, beginning at statement 62, the familiar quadratic formula is used to evaluate the pair of roots if the roots are complex. However, if the roots are real, the pair of roots are evaluated by expressing the quadratic factor as

$$S^2 + PI\,S + QI = (S - RR1)(S - RR2) = 0 \qquad (73)$$

where RR1 and RR2 are the values of the two real roots. By multiplying the two real factors, Eq (75) may be written as

$$S^2 + PI\,S + QI = S^2 + S(-RR1 - RR2) + (RR1 * RR2) = 0 \qquad (74)$$

Equating coefficients of equal powers in Eq (74) yields

$$PI = -(RR1 + RR2) \tag{75}$$

$$QI = RR1 * RR2 \tag{76}$$

Then solving for RR1 from Eqs (75) and (76), with |RR1|>|RR2| , and applying the quadratic formula produces

$$RR1 = -\frac{PI}{2} - \sqrt{\left(\frac{PI}{2}\right)^2 - QI} \qquad (PI > 0) \tag{77}$$

$$RR1 = -\frac{PI}{2} + \sqrt{\left(\frac{PI}{2}\right)^2 - QI} \qquad (PI < 0) \tag{78}$$

and from Eq (76)

$$RR2 = \frac{QI}{RR1} \tag{79}$$

The evaluation of the real roots by this form of the quadratic formula prevents loss of significance when |RR2|<|RR1| (Ref 2:646).

After the roots are evaluated and printed, the degree of the polynomial equation is reduced by two when a pair of roots is found by the Bairstow iteration and by one when a single root is found by the Newton iteration. The number of coefficients M is compared to 1, 2, and 3 consecutively and the program branches according to the value of M as described earlier in the chapter. If M>1, the polynomial equation has not been completely solved; therefore, the B coefficients

of the reduced polynomial equation are transferred to the A addresses, as shown at statements 70 and 79, and the program continues until all the roots have been evaluated and printed for a specific value of G.

Variations of Sensitivity. After computing and printing all the roots for a specific value of G, the program branches to statement 81 where a series of SENSE SWITCH statements are used to afford the operator a combination of methods for controlling the variation of G. As shown on the flow-chart, the execution of the program may be halted by turning SENSE SWITCH #3 on. The program remains halted until the operator presses START at the console. By halting the program execution, the operator has time to set the remaining SENSE SWITCHES to the desired positions. If the operator desired to vary G by an arithmetic progression, SENSE SWITCH #1 is turned on. After the operator presses START, the program branches to statement 5 where a new initial value of G and the common difference GINC are entered such that G is varied as

G, G+GINC, G+2 GINC, G+3 GINC, . . .

If SENSE SWITCH #1 is left in the OFF position, the program progresses to statement 87 where the operator again has the option of continuing the variation by an arithmetic or geometric progression. If SENSE SWITCH #2 is in the OFF position, the program progresses to statement 89 where the previous value of G and the common ratio GRAT (which was entered at the beginning of the program) are used to vary the sensitivity G as

$$G,\ G*GRAT,\ G*GRAT^2,\ G*GRAT^3,\ \ldots$$

Since practice has shown that it is not necessary to change the value of GRAT, the program is written such that the value of GRAT remains as initially entered. As a refinement to the program, a procedure is shown in Appendix H for changing the program to afford the operator the option of changing GRAT during program execution.

Conflicting requirements concerning the variation of G led to the development of the two methods for varying G, as previously described. Small variations of G are needed to compute accurately the points on the root locus when small changes of G produce rapid changes in the root-locus, i.e., at break-in points, break-away points, and points of intersection of branches at other than the real axis. However, if small variations are used for computing roots when the root locus is changing slowly for large variation in G, i.e., at points when the root locus is approaching a finite zero or infinity, the computation is excessively slow. Therefore, the arithmetic progression is used for small variations and the geometric progression is used for large variations.

Operating Instructions for Program Execution

As explained earlier, the division of the composite or total program into two parts gives the operator the added convenience of executing the two parts as two individual operations. However, since the two parts are usually executed consecutively, the detailed operating instructions are given for the execution of the entire program.

To Initiate Program Execution

1. Mount Object tape Part 1
2. Press RESET, INSERT
3. Type 360000000 3 (insure enough leader tape is punched)
4. Press RELEASE, START
5. Computer types, "LOAD DATA", operator presses START
6. Computer types, "1", operator types number of linear factors in numerator, presses RELEASE, START and types number of quadratic factors in numerator, presses RELEASE, START
7. Computer types "3", operator types value of linear factors in numerator, presses RELEASE, START and types value of quadratic factors in numerator, presses RELEASE, START
8. Computer types "2", operator types number of linear factors in denominator, presses RELEASE, START. Types number of Quadratic factors in denominator, presses RELEASE, START
9. Computer types "4", operator types value of linear factors in denominator, presses RELEASE, START, types value of quadratic factors in denominator, presses RELEASE, START
10. Computer prints at typewriter and punches on data tape the value of coefficients formed by numerator followed by value of coefficients formed by denominator
11. Remove object tape Part 1, mount object tape Part 2
12. Press RESET, INSERT
13. Type 3600000003
14. Press RELEASE, START
15. Computer types "LOAD DATA", operator mounts data tape punched by Part 1, presses START
16. Computer reads data tape then types "1"
17. Turn SENSE SWITCHES 1, 2, 3, and 4 to OFF position

18. Operator types initial value of G, presses RELEASE, START, types progression ratio GRAT, presses RELEASE START

19. Computer prints "0" and value of G on one line then computes and prints roots for the value of G

20. Computer automatically varies G by geometric progression and prints roots for each new value of G as long as SENSE SWITCHES 1, 2, 3, and 4 are in OFF position

21. Turn SENSE SWITCH #3 to ON position to HALT computation, Press START to resume computation

To Change from Geometric to Arithmetic Progression.

1. Turn SENSE SWITCH #3 to ON, computer halts after printing last computed roots

2. Turn SENSE SWITCH #1 to ON, press START

3. Operator types new initial value of G, presses RELEASE, START

4. Operator types incremental value of sensitivity GINC, positions SENSE SWITCHES as #3 OFF, #1 OFF, #2 ON, presses RELEASE, START

5. To change GINC or to begin with new initial value of G, repeat 1 through 4

To Change from Arithmetic to Geometric Progression.

1. Turn SENSE SWITCH #3 to ON, computer halts

2. Turn SENSE SWITCH #1 to ON, press START

3. Operator types value of G, presses RELEASE, START

4. Operator types "1.0", positions SENSE SWITCHES as: #3 OFF, #1 OFF, #2 OFF, presses RELEASE, START (The dummy value of "1.0" is entered to satisfy statement 5; however, the progression ratio used to vary G is the GRAT entered at the beginning of the program)

To Correct Typing Error when Entering Factors of Open-loop Transfer Function.

1. If error is made typing R(N) or X(M), Y(M), turn

SENSE SWITCH #4 to ON, press RELEASE, START

2. Turn SENSE SWITCH #4 to OFF, retype R(N) or X(M), Y(M) correctly, press RELEASE, START (If X(M) is entered correctly, but Y(M) is in error, both values must be retyped correctly)

Example

To illustrate the execution of the program and the results of the computation of a root locus as computed by the execution of the program, a typical open-loop transfer function is taken as an example. The tabulation of the sensitivity and the roots printed by the computer, are shown in Fig. 8. The root locus formed by a plot of the computed roots is shown in Fig. 9.

Consider the open-loop transfer function

$$G(s)\,H(s) = \frac{K(S+19.63)}{S(S+9-j4)(S+9+j4)} \qquad (80)$$

as shown in Eq (80) there are three open-loop poles and one zero located at

$$Z_1 = -19.63$$

$$P_1 = 0.0$$

$$P_2 = -9+j4$$

$$P_3 = -9-j4$$

and these points are plotted as shown in Fig. 9.

By multiplying the linear poles to form a quadratic factor and by recognizing the pole at zero is simply a linear factor (S-0), Eq (80) may be written as

$$G(s)\,H(s) = \frac{K(S+19.63)}{(S+0.0)(S^2+18S+97)} \qquad (81)$$

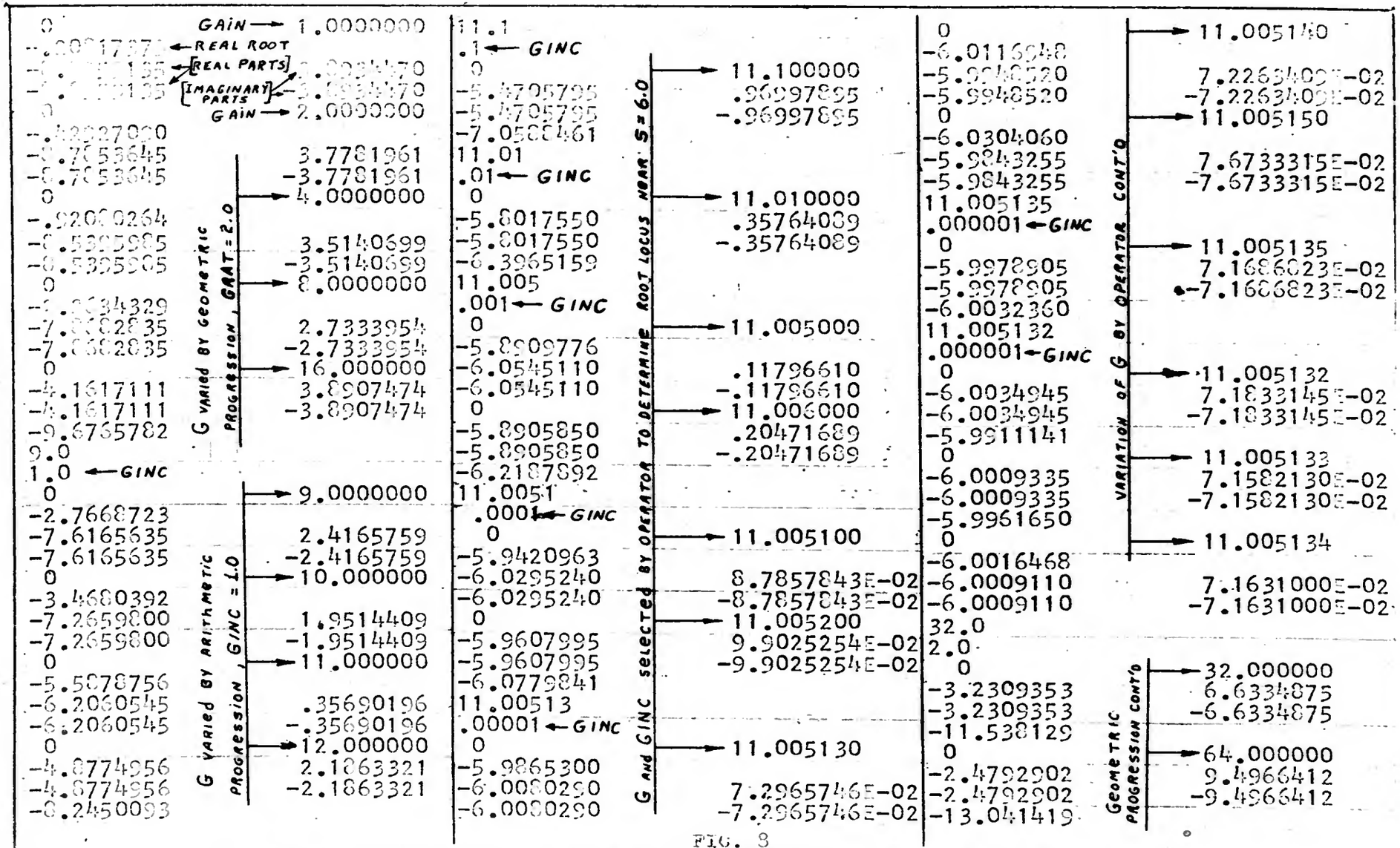


FIG. 8

OUTPUT OF COMPUTED DATA AS PRINTED BY COMPUTER SHOWING GAIN AND ROOTS

$$G(s)\,H(s) = \frac{K\,(S+19.63)}{S\,(S^2+18S+97)}$$

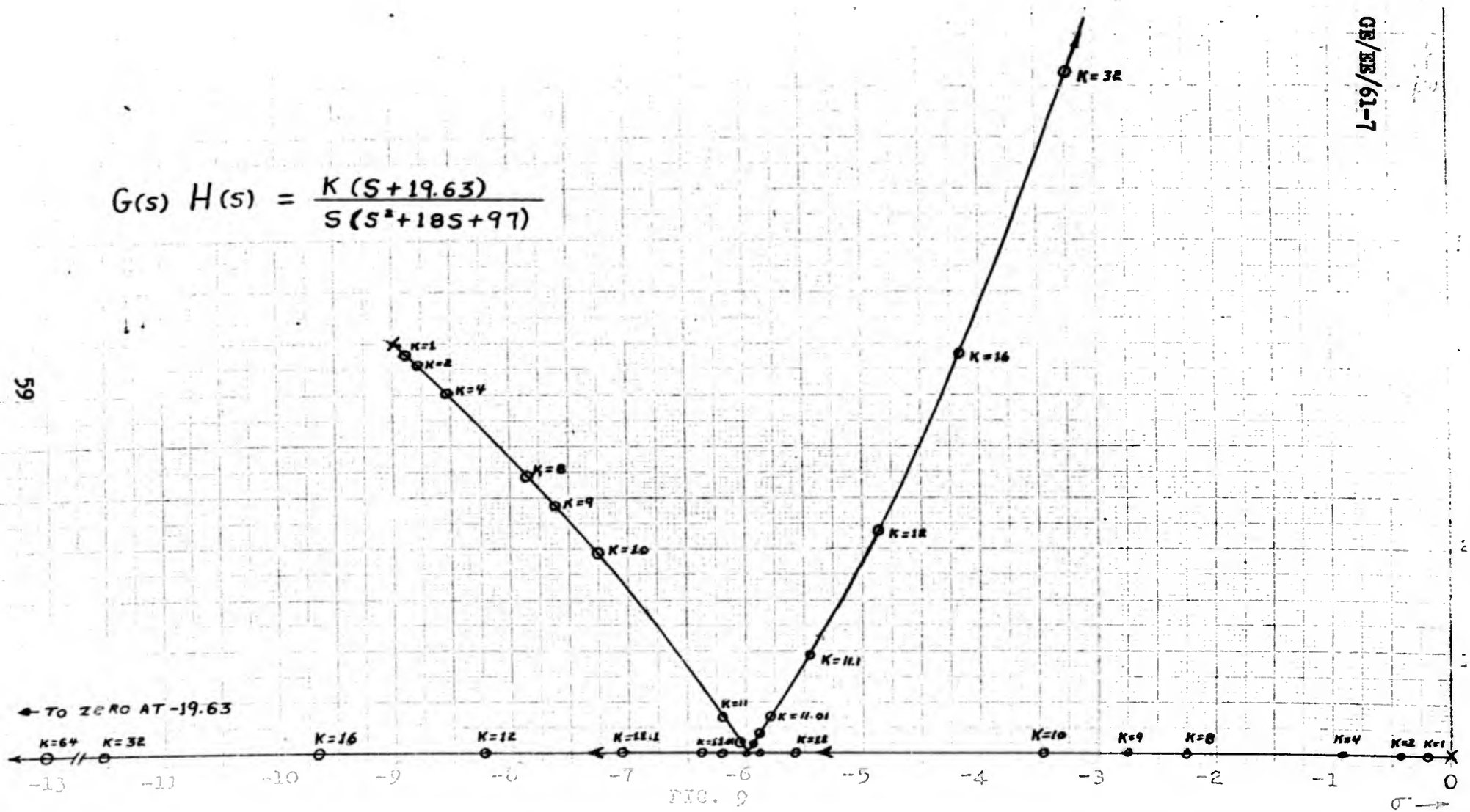


FIG. 9

TABLE VII

INPUT/OUTPUT INFORMATION TO INITIATE EXECUTION OF PROGRAM

INPUT / OUTPUT INFORMATION	EXPLANATORY NOTES		OPERATING INSTRUCTIONS
	TYPED BY	MEANING	
LOAD DATA	Computer	Notify Operator to Load Data	5
1	Computer	Notify Operator to Enter N,M	6
1	Operator	No. of Linear Factors in Numerator	6
0	Operator	No. of Quadratic Factors in Numerator	6
3	Computer	Notify Operator to Enter Factors	7
19.63	Operator	Value of Linear Factors in Numerator	7
2	Computer	Notify Operator to Enter N, M	8
1	Operator	No. of Linear Factors in Denominator	8
1	Operator	No. of Quadratic Factors in Denominator	8
4	Computer	Notify Operator to Enter Factors	9
0.0	Operator	Value of Linear Factors in Denominator	9
18.0	Operator	Value of X_1 of Quadratic	9
97.0	Operator	Value of Y_1 of Quadratic	9
1.0000000	Computer	D1	10
18.0000000	"	D2	"
1.0000000	"	E1	"
18.000000	"	E2	"
97.000000	"	E3	"
.00000000	Computer	E4	10
	End of PART 1		
3(00000003	Operator	Command to Read in PART 2 Object Tape	13
LOAD DATA	Computer	Notify Operator to Load Data	15
1	Computer	Notify Operator to Enter G and GRAT	18
1.0	Operator	Initial Value of G	18
2.0	Operator	Value of GRAT	18

to express the transfer function in the correct format for entering the required information for the execution of the Part 1 program. The input and output information as printed at the typewriter to initiate execution of the program for this example is shown in Table VII. Brief explanatory notes and a reference to the applicable operating instructions are entered opposite the INPUT/OUTPUT information as typed at the console typewriter.

After the last entry in Table VII, the operator presses RELEASE and START to initiate the computation with the initial value of the sensitivity as G=1.0. The value of G=1.0, preceeded by a zero, and the value of the roots for G=1.0 are printed as shown in Fig. 8. The form in which the computer prints the roots depends on the iteration by which the roots were computed. If a quadratic factor is computed by the Bairstow iteration, the value of the pair of roots are printed in complex form with the real part and complex part of the first root printed on one line and the real part and complex part of the second root printed on the next line. However, if a linear factor is computed by the Newton Method, only the value of the real root is printed.

The procedure which is recommended and used in this example is to initiate the computation with an initial sensitivity of G=1.0 and to vary G by a geometric progression until the general shape of the root locus is determined. In this example, a progression ratio of GRAT=2.0 was used and the computation was continued until G=16.0 and the general shape of the root was known. As shown in Fig. 8 the computation was halted after G=16.0. The computation was re-

started with a new value of G=9.0 and the sensitivity was varied by an arithmetic progression in increments of GINC=1.0 to compute the root locus in greater detail between G=8.0 and G=16.0. By inspecting Fig. 8, it can be seen how the operator controlled the computation by varying G to compute in detail the root locus at the critical points such as the "near" break-in/break-away point at S=-6.0. Although the branches of the root locus from the two complex poles approach very closely to the real axis (as shown by the computed roots in Fig. 8) they do not break in and out. However, an analytic solution of the polynomial equation reveals that the branches would have broken in and out at s=-6 if the value of the real zero at -19.63 had been -19.636363.... If the real zero had been of such value that the branches did break in and out, the G could have been varied so as to "pin-point" the point of break-in and point of break-out.

Although the root locus is relatively simple for the open-loop transfer function used in this example, the same procedures may be used to compute the root locus for any control system of twenty or less open-loop poles. However, for higher degree polynomials, the accuracy does degenerate as discussed in the following chapter.

V RESULTS AND CONCLUSIONS

Summary

From an analysis of the definition and derivation of the mathematical expression for the root locus it was determined that a plot of the roots of the characteristic equation, as the sensitivity K was varied from zero to infinity, would produce the root locus. From the derived mathematical expression, it was shown that the characteristic equation could be expressed as a general polynomial equation of degree N where: (1) the degree is determined by the number of open-loop poles, (2) the coefficients of the N^{th} degree polynomial are formed by adding coefficients of equal powers of: (a) the sub-polynomial produced by the product of the open-loop zeros and (b) the sub-polynomial produced by the product of the open-loop poles, and (3) the coefficients of the N^{th} degree polynomial are functions of the sensitivity K, since the sub-polynomial produced by zeros is multiplied by the sensitivity K. Therefore, the problem of computing the root locus was transformed into a format such that repeated solutions for the roots of the polynomial equation as the sensitivity K is varied, would produce the root locus. By using a general procedure for forming the coefficients of the N^{th} degree polynomial and by employing numerical techniques for the solution of a general polynomial, a program was developed which may be used, in theory, to compute automatically the root locus of any control system. A practical limit that the open-loop transfer function contain twenty or less poles was imposed due to the relatively small

memory capacity of the IBM 1620. For a computer with a larger memory capacity, the program can be readily adapted to compute the root locus for control systems with more than twenty poles (provided the accuracy does not degenerate as discussed below).

The numerical technique used to evaluate the roots of the general polynomial equation employs the simultaneous Newton and Bairstow iterative methods. To ensure convergence by one of the two methods, a procedure was established to scale the polynomial and to reverse the sequence of coefficients, when required. In practice, the iterative process has never failed to converge.

The IBM FORTRAN system of programming was used in order that the engineer could readily understand and execute the program with a minimum of knowledge of the computer. With the program written in FORTRAN language, the computer automatically translates the program into basic machine language and punches the program on object tape. The participation required of the engineer is that he mount the object tape, and enter the factors of the open-loop transfer function, the initial value of sensitivity K, and the geometric progression ratio.

Results

Although the iterative process used in the program to evaluate the roots of the polynomial characteristic equation has never failed to converge, the accuracy of the evaluated roots does not always satisfy the plotting accuracy requirement for higher degree polynomial equations with multiple roots.

The accuracy of the roots as computed by the program are affected by: (1) the fact that only eight figures are retained in the computer calculation with the remainder of the figures being truncated, therefore, the eighth figure is not always significant, (2) the iterative process becomes sensitive to the number of significant figures carried when a multiplicity of roots or roots close together in magnitude are present, and (3) the degree of the polynomial, since any error in a root is reflected in the reduced polynomial from which the succeeding roots are evaluated. For the root loci which have been computed by the program (approximately twenty) the iterative process has always converged with a ratio of less than 10^{-6} for the remainder terms of the reduced polynomial and the equal power coefficients of the original polynomial. As another accuracy check, the roots evaluated for zero sensitivity during the computation of the twenty example root loci were compared to the values of the open-loop poles, which are the exact roots of the polynomial equation for zero sensitivity. In all the comparisons, the values of the roots agreed to at least five figures and usually to seven figures with the values of the open-loop poles. As may be seen from an inspection of the plot of the root loci in the example in Chapter IV and in the Appendices (the scale is such that third place accuracy can be plotted), the plot of the roots produced smooth curves. This indicates accuracy well beyond that needed for plotting purposes. However, in all the example root loci computed, the highest degree polynomial equation solved was seven and the multiplicity of roots did not exceed three. To investigate the accuracy for higher degree polynomials with

multiple roots, factors were entered as poles of the open-loop transfer function so that polynomials generated by known factors could be evaluated (with sensitivity equal zero). With no multiplicity of roots, the roots evaluated by the computer agree in magnitude with the known factors to at least three figures for polynomial equations up to a degree of ten. Several polynomial equations were generated with three and four equal factors. For three factors the multiple roots agreed in magnitude with the known factors to at least three figures and usually to five figures. For four factors, the multiple roots and factors agreed to at least two figures (insignificant difference in third figure) and usually to three or more figures. For polynomial equations over the degree of ten or with more than four multiple roots, the accuracy usually degenerated below that needed for plotting accuracy.

It is difficult to determine specifically the accuracy which can be expected of the roots as affected by the combination of the three factors mentioned earlier. However, for the root loci thus far computed, the results indicate that the roots are accurate to at least three figures and usually to five for polynomials of degree ten or less provided the multiplicity of roots does not exceed four.

Conclusions

By expending a minimum of time and effort, the engineer can use the program developed in this thesis to compute accurate root loci automatically for control systems of ten or less open-loop poles. The program can be used to compute root loci for control systems of higher order open-loop poles. However, as the above

results indicate, the accuracy of the roots often degenerate below that generally needed for plotting (three figures). The accuracy could be sufficiently improved by increasing the number of figures (eight in FORTRAN) retained in the computer calculations. Through further study, the program developed in this thesis could be re-written in basic machine language or symbolic programming language so that the number of figures retained in the calculations could be increased (any desired number) to insure sufficient accuracy for a control system of twenty or more open-loop poles.

By initiating the computation with a sensitivity of zero and by using small incremental variations of sensitivity, the root loci can be computed automatically (within the limitations discussed earlier). If an automatic plotter is available, the root locus can be plotted as well as computed automatically (plotter was not available for this study).

Operator Participation. Although the root locus can be computed automatically (with no operator participation) by using very small variations of sensitivity, an unnecessary number of roots would be computed when the root locus is changing slowly for relatively large variations of sensitivity. However, the detail produced by the small variations is required when the root locus is changing rapidly for small variations in sensitivity. In order to eliminate unnecessary computation and yet to obtain detail when required, the conflicting requirements for sensitivity variation were resolved by designing

the program so that the engineer could exercise his knowledge of root locus. Provisions are made in the program so that the engineer can monitor and stop the computation at any time in order to change the value of sensitivity and to choose between an arithmetic or geometric progression for varying the sensitivity. This procedure of utilizing operator participation has proved to be very successful in reducing the amount of computation without sacrificing the necessary detail to plot accurate root loci. Further study could possibly reveal techniques for further automation so that only the essential points on the root locus are computed with little or no participation by the operator.

Recommended Further Study. It is recommended that further study be conducted to convert the program, which is written in FORTRAN language, to a program written in a more basic machine language. Although the convenience of FORTRAN would be lost, the number of figures retained in the calculations could be increased to reduce the degeneration of accuracy for control systems of higher order open-loop poles with multiple roots. The conversion of the program would be facilitated by using a program developed by Lieutenant Pratt (program tape is located in the Institute Computer Laboratory) which automatically translates and types a complete listing of a FORTRAN program in both symbolic programming and basic machine language.

To automate further the program as developed or to write a new program, an analysis should be made of the root locus to determine

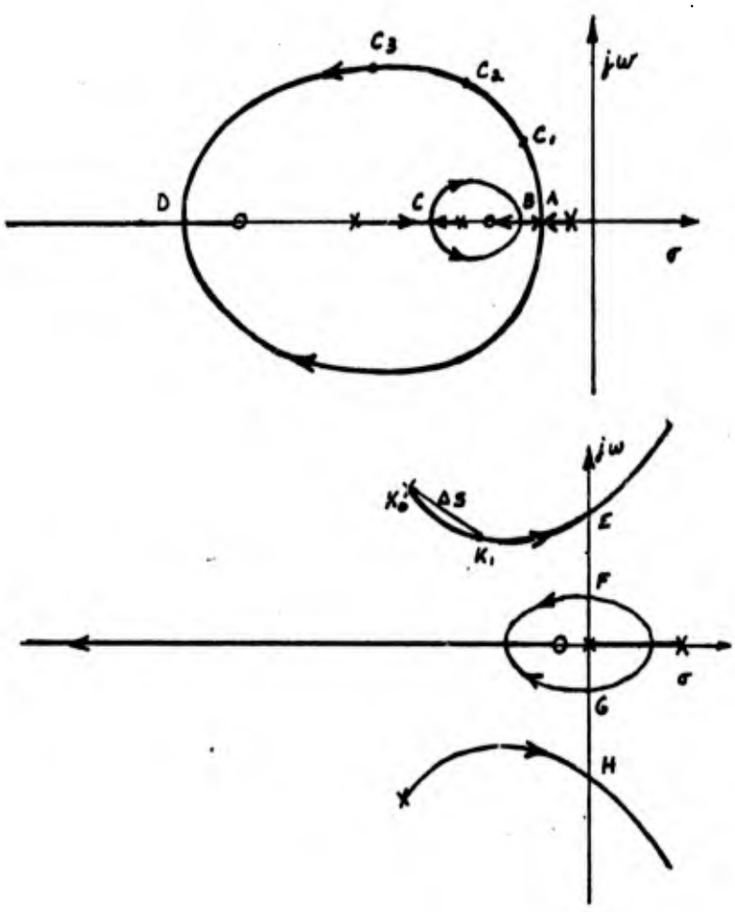


FIG. 10
VARIOUS ROOT-LOCUS CONFIGURATIONS

(From Ref 1:136)

specifically what points are essential for plotting an accurate root locus.

From an inspection of various root loci configurations, such as those shown in Fig. 10, it is obvious that some of the essential points are as follows:

1. Break-in and break-away points, i.e., points A, B, C, and D
2. Points at the $j\omega$ axis crossing, i.e., points E, F, G, and H
3. Sufficient points to determine the outline of the curves. As shown by Fig. 10, when the curvature is greater between C_1 and C_2 than between C_2 and C_3, more points are required to identify the path of the root locus.
4. Points of intersection of branches at points other than real axis (not shown in Fig. 10)

An analysis and dicussion of the feasibility of automating the determination of each of the essential points is performed in the order in which the points are identified above, as follows:

1. Mathematical procedures could be developed and systematized (Ref 1:149-152) such that a program could be written which would compute the break-in and break-away points independent of the sensitivity K. However, it appears feasible that the program as developed in this thesis, could be modified to automate the determination of these points. The imaginary parts of all the roots could be stored temporarily and compared to the value of the respective imaginary parts of the roots for the succeeding value of sensitivity. If the imaginary parts changed from zero to a value greater than zero or vice versa, then the root locus passed through a break-in or break-away point. The increment by which the sensitivity is varied could be repeatedly reduced until

the critical points are determined.

2. Mathematical procedures already developed (Ref 1:153) could be programmed so that the points of crossing at the $j\omega$ axis and the value of the sensitivity for these points could be computed automatically. As another suggested approach, the procedure of varying the sensitivity, as used in the program developed in this thesis could be further automated to determine these points by testing the real parts of the roots for a change in sign. If the signs had changed, the increment by which the sensitivity is varied could be repeatedly reduced to determine the specific value of the root when the real part is equal to zero.

3. The number of points needed to identify the outline of the root locus is proportional to the curvature of the curve. If a simple procedure could be developed for sensing the curvature of the branches of the root locus, a method could be developed for automating the variation of sensitivity based on the curvature. Another suggested approach for computing only the essential points to determine the outline of the curves is that the sensitivity be varied according to the distance moved by the branches between two consecutive values of sensitivity. Since the branches move at different rates for the same change in sensitivity, the automatic change in sensitivity should be dictated by the branch which moves the largest distance. A simple procedure would be needed for computing the arc length. However, as shown in Fig. 10, the distance moved by a branch could be approximated by computing the cord Δs of the arc between K_0 and K_1. Based on the difference between the cord Δs and a desired fixed

length of arc, the amount by which K is incremented could be increased or decreased automatically to make the length of Δs approach the desired fixed length.

Another way of automatically varying K to obtain a desired fixed length of cord could be based on differential considerations.

As shown in Chapter II, the N^{th} degree polynomial composed of the A coefficients is a function of S and K and is formed by the addition of two sub-polynomials

$$A(S,K) = E(S) + K\,D(S) \tag{82}$$

Since the polynomial A is a function of S and K, the differential of A may be written as

$$dA(S,K) = \left(\frac{\partial A}{\partial S}\right) dS + \left(\frac{\partial A}{\partial K}\right) dK \tag{83}$$

Denoting A'(S) as the derivative of A with respect to S and observing that only the D coefficients are multiplied by K, Eq (83) may be written as

$$dA(S,K) = A'(S)\,dS + D(S)\,dK \tag{84}$$

To compute the rate of change of S with respect to K along a locus of constant A, in particular A=0, the differential of A is equal to zero (dA=0) and Eq (84) becomes

$$\frac{dS}{dK} = -\frac{D(S)}{A'(S)} \tag{85}$$

For small changes of K, approximations to the differentials may be used and Eq (85) gives

$$\Delta S = \frac{D(S)}{A'(S)} \Delta K \tag{86}$$

$$\Delta K = \frac{A'(S)}{D(S)} \Delta S \tag{87}$$

Eq (86) shows mathematically why the branches of the root locus move rapidly for small changes in K near multiple roots, i.e., large change in S for small change in K (as shown graphically in Fig. 9). For example, consider a double root at S=-a. The general polynomial can be expressed as

$$A(S) = (S+a)^2 B(S) \tag{88}$$

Taking the derivative with respect to S gives

$$A'(S) = 2(S+a) B(S) + (S+a)^2 B'(S) \tag{89}$$

Letting S take on the value -a yields

$$A'(-a) = 2(0) B(-a) + B'(-a) = 0 \tag{90}$$

Therefore, it can be seen by inspecting Eq (86) that as A'(S) ap-

proaches zero, Δs approaches infinity.

Eq (87) could be used to automatically increment K for a desired fixed Δs, based on the fact that A'(S) becomes small (approaching zero at multiple roots) or large in proportion to the distance moved by the branches. The automation would not require extensive modification since the value of A'(S) has been calculated already as the remainder in the second synthetic division of the Newton iteration (see Eq (39) and the polynomial D(S) could be evaluated by another synthetic division (see Eq (33).

4. The determination of the intersection of branches at points other than the real axis appears to be considerable more difficult to mechanize for the computer than the determination of the other three essential points discussed above. However, it is felt that a detailed analysis of the problem would produce mathematical procedures which could be programmed to compute these points independent of the sensitivity K. Also, from an inspection of the differentials discussed in 3 above, it appears that the relations could be used to modify the program, as developed in this thesis, to further automate the determination of these points.

Based on the analysis of the essential points listed above, it appears feasible that the program as developed could be further automated. However, if the IBM 1620 Computer is used, the relatively limited memory capacity should be considered before additional statements are added to the program. Since the program is divided into two parts, there should be adequate memory space to further automate the program as outlined above.

Bibliography

1. D'Azzo, J. J., and C. H. Houpis Feedback Control System Analysis and Synthesis, New York: McGraw-Hill Book Co. Inc., 1960

2. Ellenberger, K. W. "On Programming the Numerical Solution of Polynomial Equations". Communications of the Association for Computing Machinery, 3 : 644 - 647 (December 1960)

3. Hildebrand, F. B. Introduction to Numerical Analysis. New York: McGraw-Hill Book Co. Inc., 1956.

4. IBM 1620 Data Processing System Bulletin No. J29-4200-2. IBM 1620 FORTRAN: PRELIMINARY SPECIFICATIONS. U.S.A., 1960

Bibliography

1. D'Azzo, J. J., and C. H. Houpis Feedback Control System Analysis and Synthesis, New York: McGraw-Hill Book Co. Inc., 1960

2. Ellenberger, K. W. "On Programming the Numerical Solution of Polynomial Equations". Communications of the Association for Computing Machinery, 3 : 644 - 647 (December 1960)

3. Hildebrand, F. B. Introduction to Numerical Analysis. New York: McGraw-Hill Book Co. Inc., 1956.

4. IBM 1620 Data Processing System Bulletin No. J29-4200-2. IBM 1620 FORTRAN: PRELIMINARY SPECIFICATIONS. U.S.A., 1960

Appendix A

Part 1 Program and Detailed Flow Chart

```
C  ROOT LOCUS PART 1 HEF
DIMENSION R(20),X(10),Y(
10=0
NO=2
150  10=10+1
PRINT,10
 ACCEPT,N,M
IF(M) 102,102,110
102  IF(N)103,103,104
103  E(1)=1.0
M=1
GO TO 135
104  NO=NO+1
PRINT,NO
DO 105 I=1,N
118  ACCEPT,R(I)
IF(SENSE SWITCH 4)118,10
105  CONTINUE
M=N+1
E(1)=1.0
DO 106 K=2,M
106  E(K)=0.
DO  107 J=1,N
DO 108 K=1,J
I=J+1-K
108  E(I+1)=E(I+1)+E(I)*
107  CONTINUE
GO TO 135
110  IF(N)111,111,117
111  MO=1
GO TO 119
117  MO=2
119  GO TO (120,121),MO
120  NO=NO+1
PRINT,NO
DO 112 I=1,M
122  ACCEPT,X(I),Y(I)
IF(SENSE SWITCH 4)122,11
112  CONTINUE
GO TO 123
121  NO=NO+1
PRINT,NO
DO 125 I=1,N
124  ACCEPT,R(I)
IF(SENSE SWITCH 4)124,12
125  CONTINUE
DO 127 I=1,M
126  ACCEPT,X(I),Y(I)
IF(SENSE SWITCH 4)126,12
127  CONTINUE
123  K=(2*M)+2+N
DO 113 I=1,K
113  E(I)=0.
E(2)=1.0
DO 114 I=1,M
DO 115 J=1,I
L=(2*I)-(2*J)+2
E(L+2)=E(L+2)+E(L+1)*X(I
115  E(L+1)=E(L+1)+E(L)*
114  CONTINUE
M=(2*M)+1
DO 116 I=1,M
116  E(I)=E(I+1)
GO TO (135,128),MO
128  E(M+1)=0.
DO 129 J=1,N
L=J+M-1
DO 130 K=1,L
I=L+1-K
130  E(I+1)=E(I+1)+E(I)*
129  CONTINUE
M=M+N
135  GO TO (131,133),10
131  DO 132 I=1,M
132  D(I)=E(I)
LOW=M
GO TO 150
133  PUNCH,LOW,M
DO 134 I=1,LOW
PUNCH,D(I)
134  PRINT,D(I)
DO 136  I=1,M
PUNCH,E(I)
136  PRINT,E(I)
END
```

ERNDON GE 61
Y(10),D(21),E(22)

105

)*R(J)

0

112

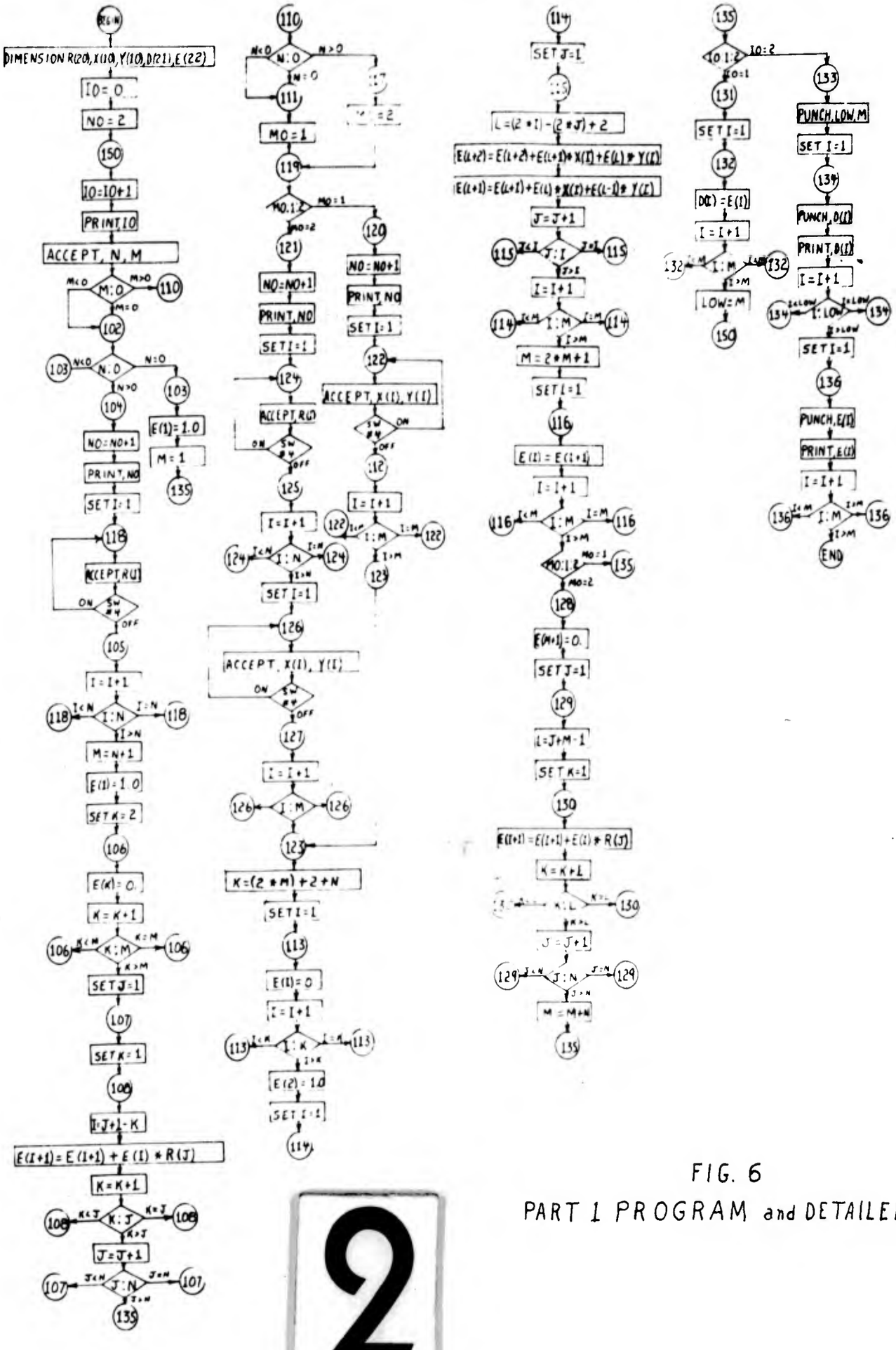
125

127

(I)+E(L)*Y(I)
)*X(I)+E(L-1)*Y(I)

)*R(J)

10



2

FIG. 6

PART 1 PROGRAM and DETAILED FLOW CHA

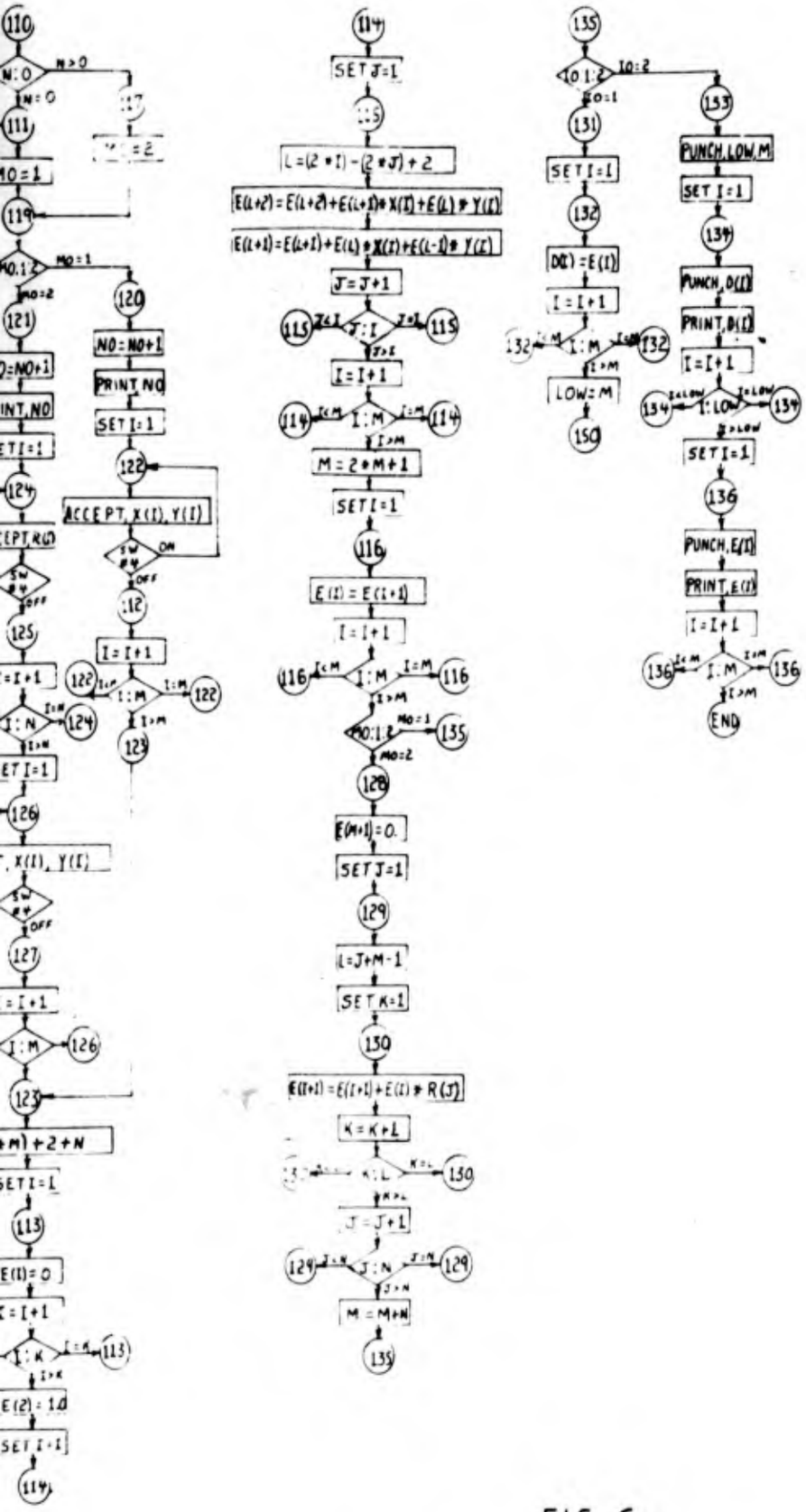


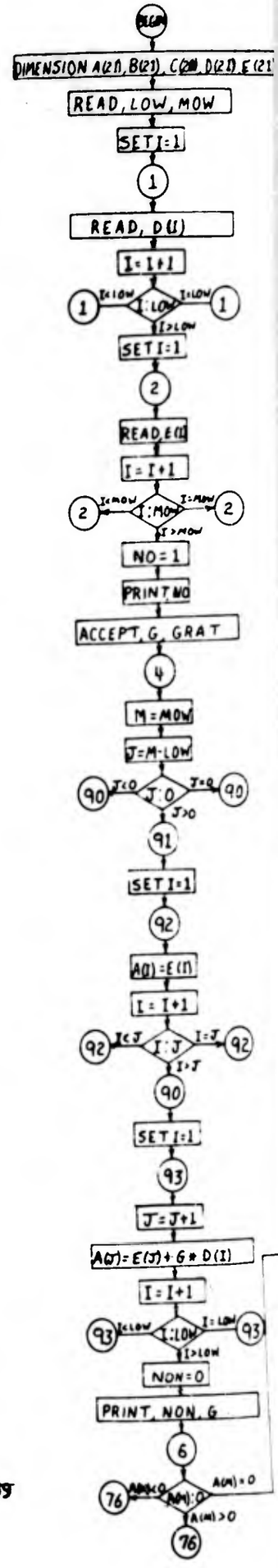
FIG. 6

PART 1 PROGRAM and DETAILED FLOW CHART

Appendix B

Part 2 Program and Detailed Flow Chart

```
C  ROOT LOCUS PART 2 HERNDON GE 61
DIMENSION A(21),B(21),C(20),D(21),E(21)
READ,LOW,MOW
DO 1 I=1,LOW
1  READ,D(I)
DO 2 I=1,MOW
2  READ,E(I)
NO=1
PRINT,NO
ACCEPT,G,GRAT
GO TO 4
5  ACCEPT,G,GINC
4  M=MOW
J=M-LOW
IF(J)90,90,91
91  DO 92 I=1,J
92   A(I)=E(I)
90  DO 93 I=1,LOW
J=J+1
93  A(J)=E(J)+G*D(I)
NON=0
PRINT,NON,G
6  IF(A(M))76,8,76
8  R1=0.
PRINT,R1
M=M-1
GO TO 6
7  SUM=0.
DO 12 I=1,M
X=A(I)
IF(X)9,12,10
9  X=0.-X
10  SUM=SUM+LOG(X)
12  CONTINUE
DOG=M
DOG=1./DOG
 SUM=DOG*SUM
CAT=EXP(SUM)
DO  14 I=1,M
14  A(I)=A(I)/CAT
X=A(2)/A(1)
Y=A(M-1)/A(M)
IF(X)15,16,16
15  X=0.-X
16  IF(Y)17,18,18
17  Y=0.-Y
18  IF(X-Y)19,22,22
19  DO 20 I=1,M
20  B(I)=A(I)
J=M
 DO  21 I=1,M
A(I)=B(J)
21  J=J-1
NO=0-NO
GO TO 23
 22  NO=NO
23  TOL=1.0E-07
58  P=0.
Q=0.
 R=0.
10=2
B(1)=A(1)
C(1)=A(1)
DO 100 K=1,20
B(2)=A(2)-(P*B(1))
DO 25 I=3,M
25  B(I)=A(I)-((P*B(I-1))+(Q*B(I-2)))
C(2)=B(2)-(P*C(1))
J=M-2
IF(J-3)26,27,27
27  DO 28 I=3,J
28  C(I)=B(I)-((P*C(I-1))+(Q*C(I-2)))
26  C(M-1)=0.-((P*C(M-2))+(Q*C(M-3)))
DENOM=(C(M-2)*C(M-2))-(C(M-1)*C(M-3))
IF(DENOM)29,30,29
30  IF(B(M))13,40,40
13  B(M)=0.-B(M)
40  POW=M-1
POW=2./POW
Q=B(M)**POW
GO TO 44
29  P1=P+(((B(M-1)*C(M-2))-(B(M)*C(M-3)))/DENOM)
Q1=Q+(((B(M)*C(M-2))-(B(M-1)*C(M-1)))/DENOM)
IF(A(M-1))32,31,32
31  X=B(M-1)
IF(X)33,34,34
33  X=0.-X
34  IF(X-TOL)37,37,38
32  RAT1=B(M-1)/A(M-1)
IF(RAT1)39,41,41
39  RAT1=0.-RAT1
41  IF(RAT1-TOL)37,37,38
37  RAT2=B(M)/A(M)
IF(RAT2)42,43,43
42  RAT2=0.-RAT2
43  IF(RAT2-TOL)60,60,38
38  P=P1
Q=Q1
44  GO TO (100,50),10
50  DO 51 I=2,M
51  B(I)=A(I)+R*B(I-1)
M=M-1
DO 52 I=2,N
52  C(I)=B(I)+R*C(I-1)
IF(C(N))54,53,54
53     10=1
GO TO 100
54   10=2
R1=R-(B(M)/C(N))
RAT3=B(M)/A(M)
IF(RAT3)55,56,56
55  RAT3=0.-RAT3
56  IF(RAT3-TOL)80,80,57
57  R=R1
100  CONTINUE
TOL=TOL*10.
IF(TOL-1.0E-03)58,59,59
59  J0=2
PRINT,J0,J0
GO TO 81
60  IF(NO)61,61,62
61  P1=P1/Q1
Q1=1.0/Q1
62  RR1=0.-P1/2.
RR2=RR1
RAD=((P1*P1)/4.)-Q1
IF(RAD)63,64,65
63  CR1=SQR(0.-RAD)
CR2=0.-CR1
GO TO 69
65  IF(P1)66,66,67
66  RR1=RR1+SQR(RAD)
GO TO 68
67  RR1=RR1-SQR(RAD)
68  RR2=Q1/RR1
64  CR1=0.
CR2=0.
69  PRINT,RR1,CR1
PRINT,RR2,CR2
M=M-2
IF(M-1)81,81,82
82  DO 70 I=2,M
70  A(I)=B(I)
76  IF(M-3)78,71,7
71  P1=A(2)/A(1)
Q1=A(3)/A(1)
GO TO 60
80  IF(NO)73,73,74
73  R1=1./R1
74  PRINT,R1
M=M-1
IF(M-1)81,81,83
83  DO 79 I=2,M
79  A(I)=B(I)
78  IF(M-2)81,75,76
75  R1=0.-A(2)/A(1)
GO TO 80
81  NO=1
 IF(SENSE SWITCH 3)85,86
85  PAUSE
86  IF(SENSE SWITCH 1)5,87
87   IF(SENSE SWITCH 2)88,89
88  G=G+GINC
GO TO 4
89  G=G*GRAT
GO TO 4
END
```



2

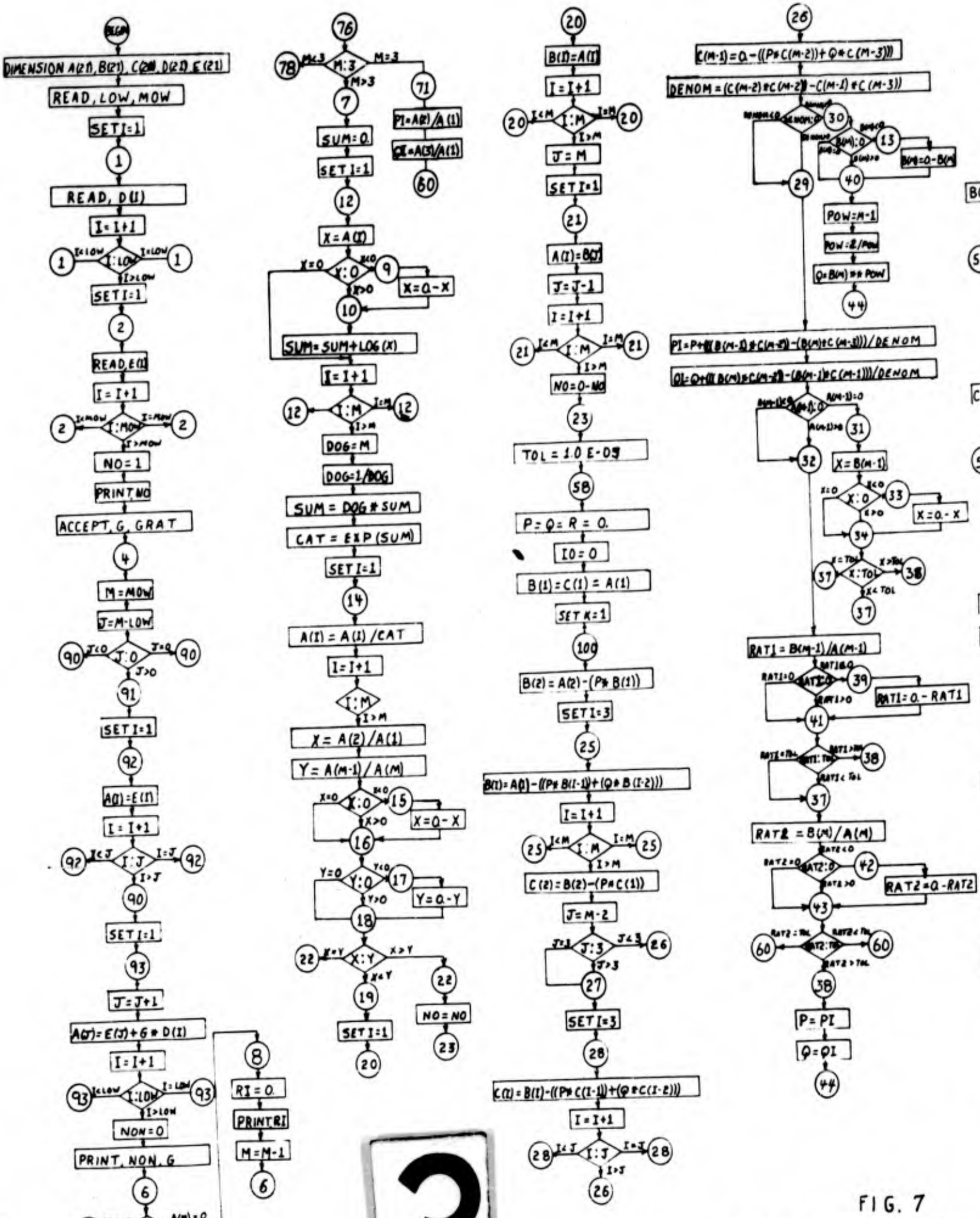
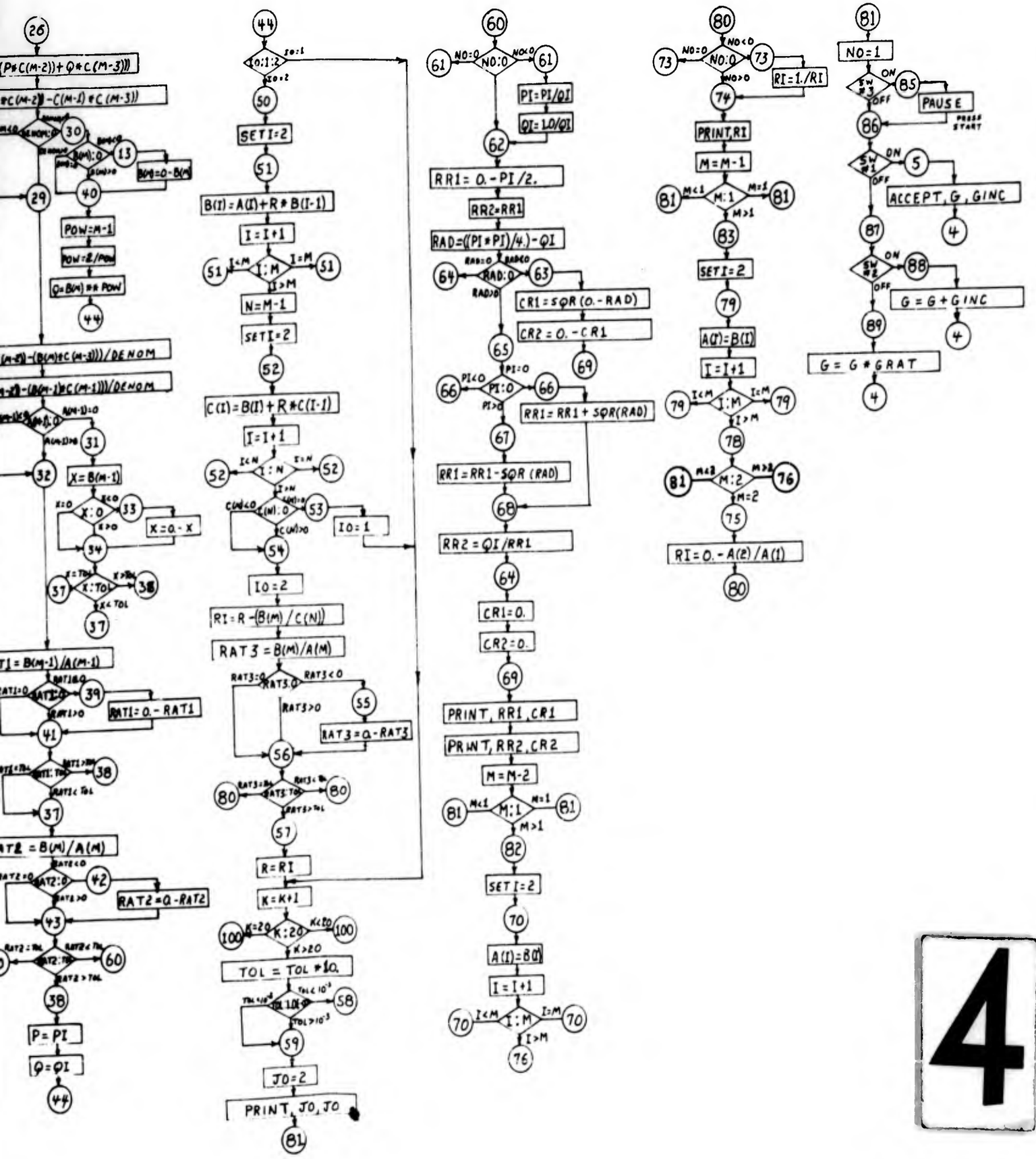


FIG. 7

PART 2 PROGRAM and D

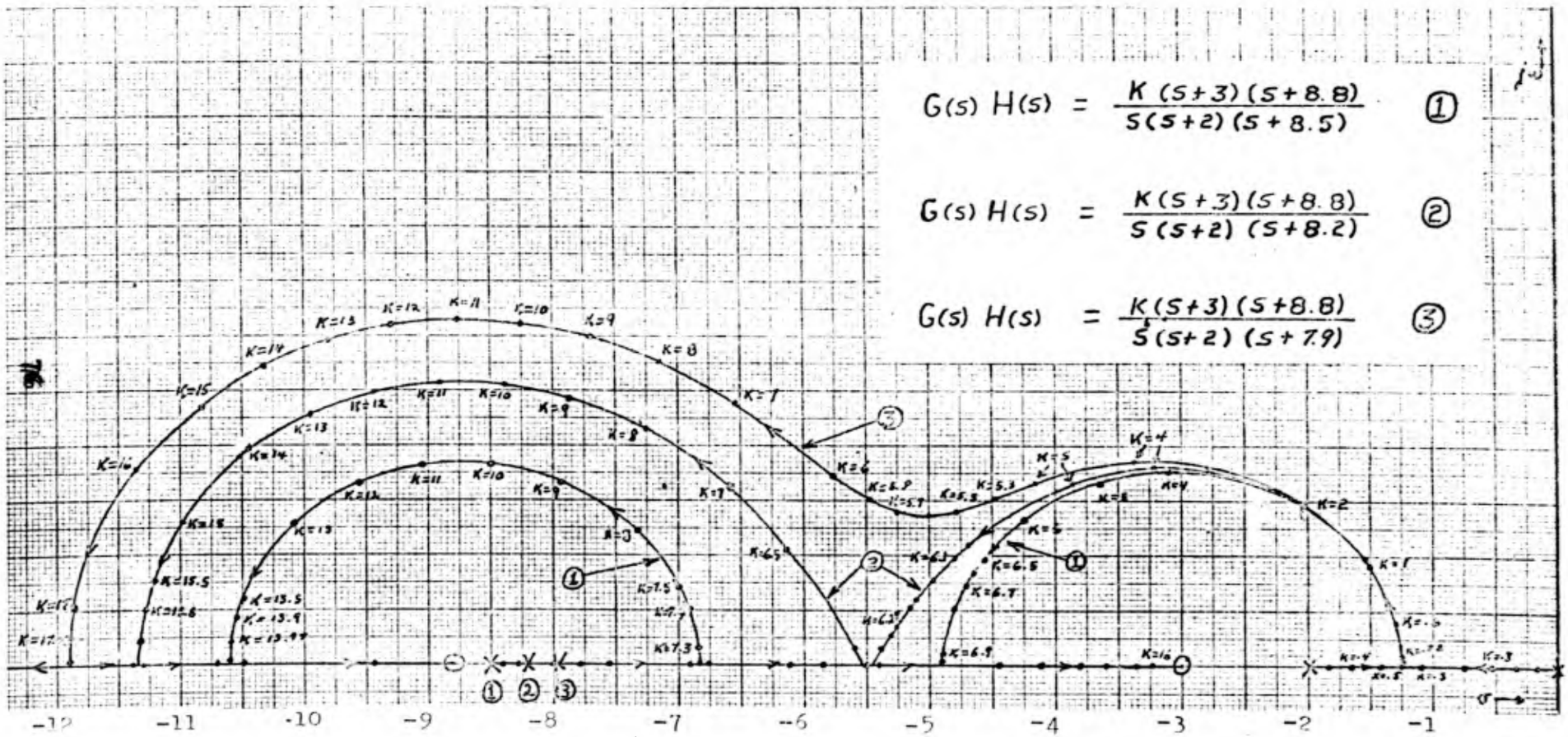
3



4

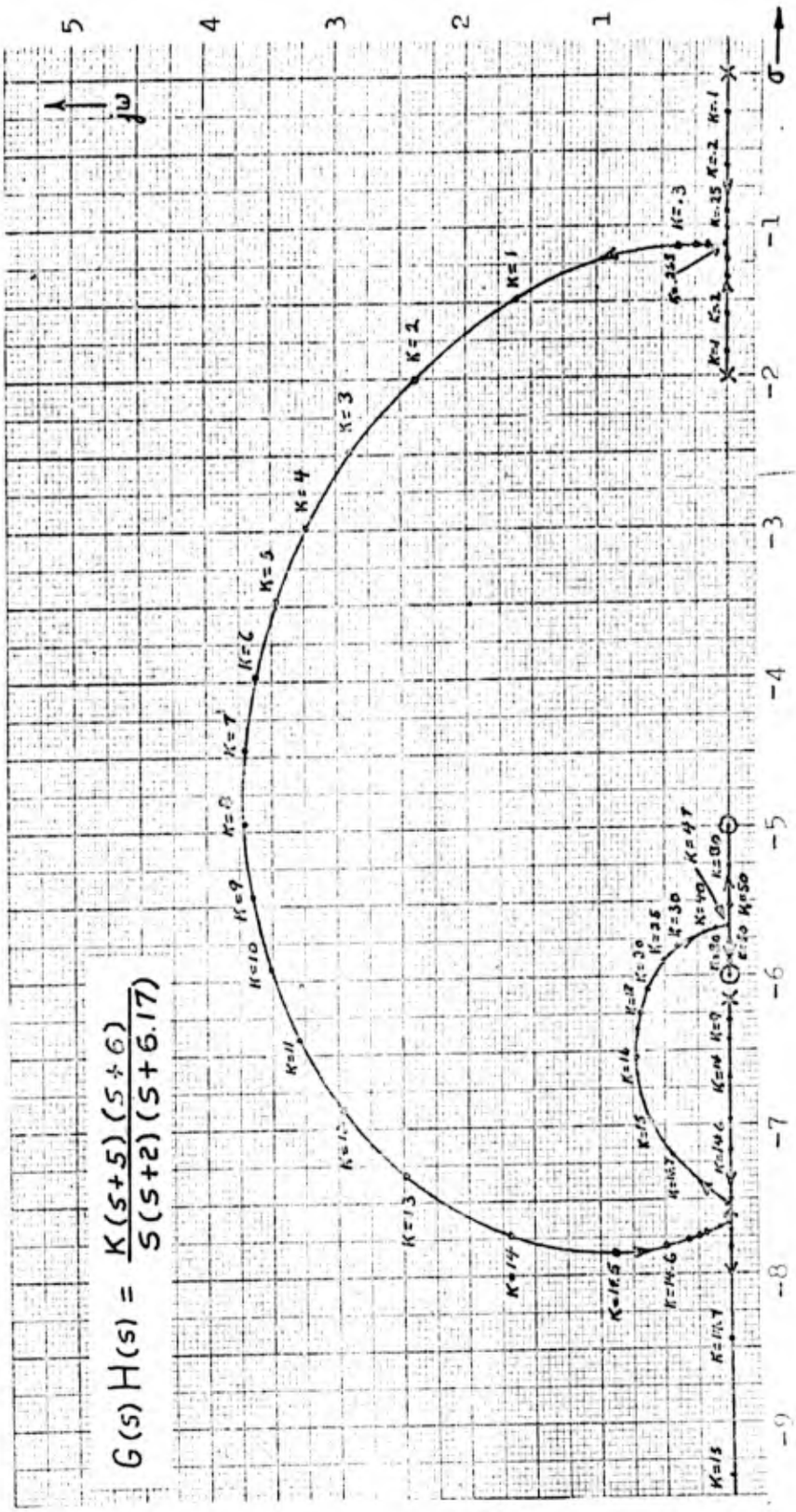
FIG. 7

RT 2 PROGRAM and DETAILED FLOW CHART



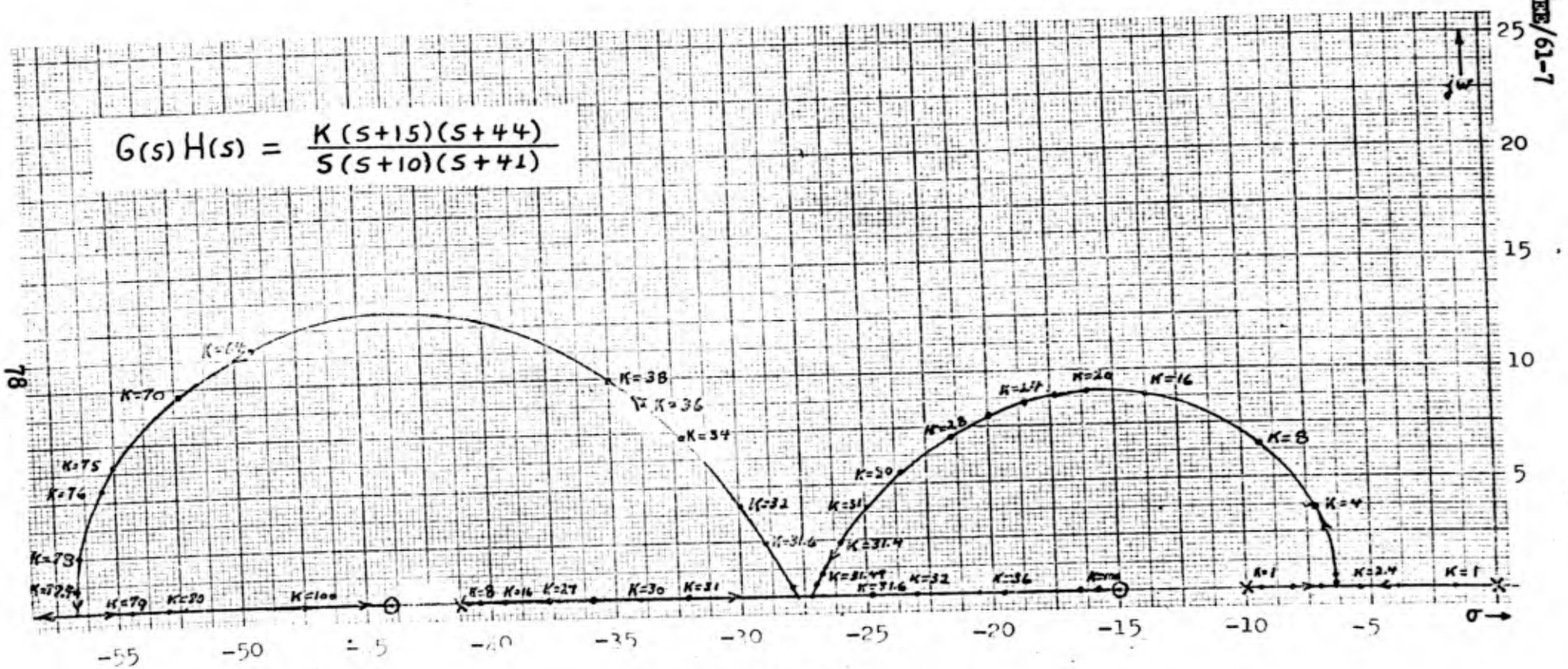
Appendix C

Family of Root Loci for Two Zeros and Three Poles



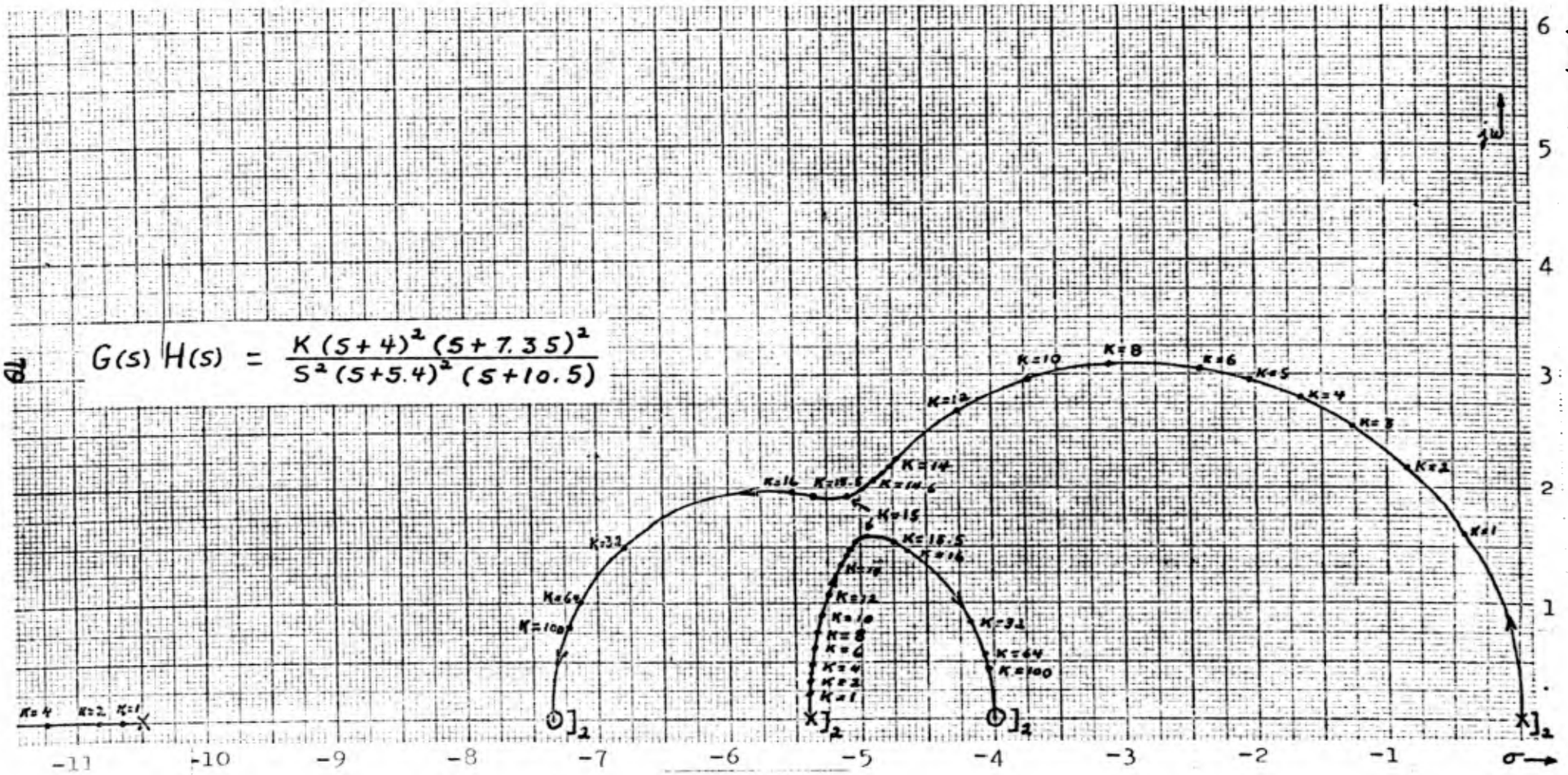
Appendix D

Root Locus for Two Zeros and Three Poles



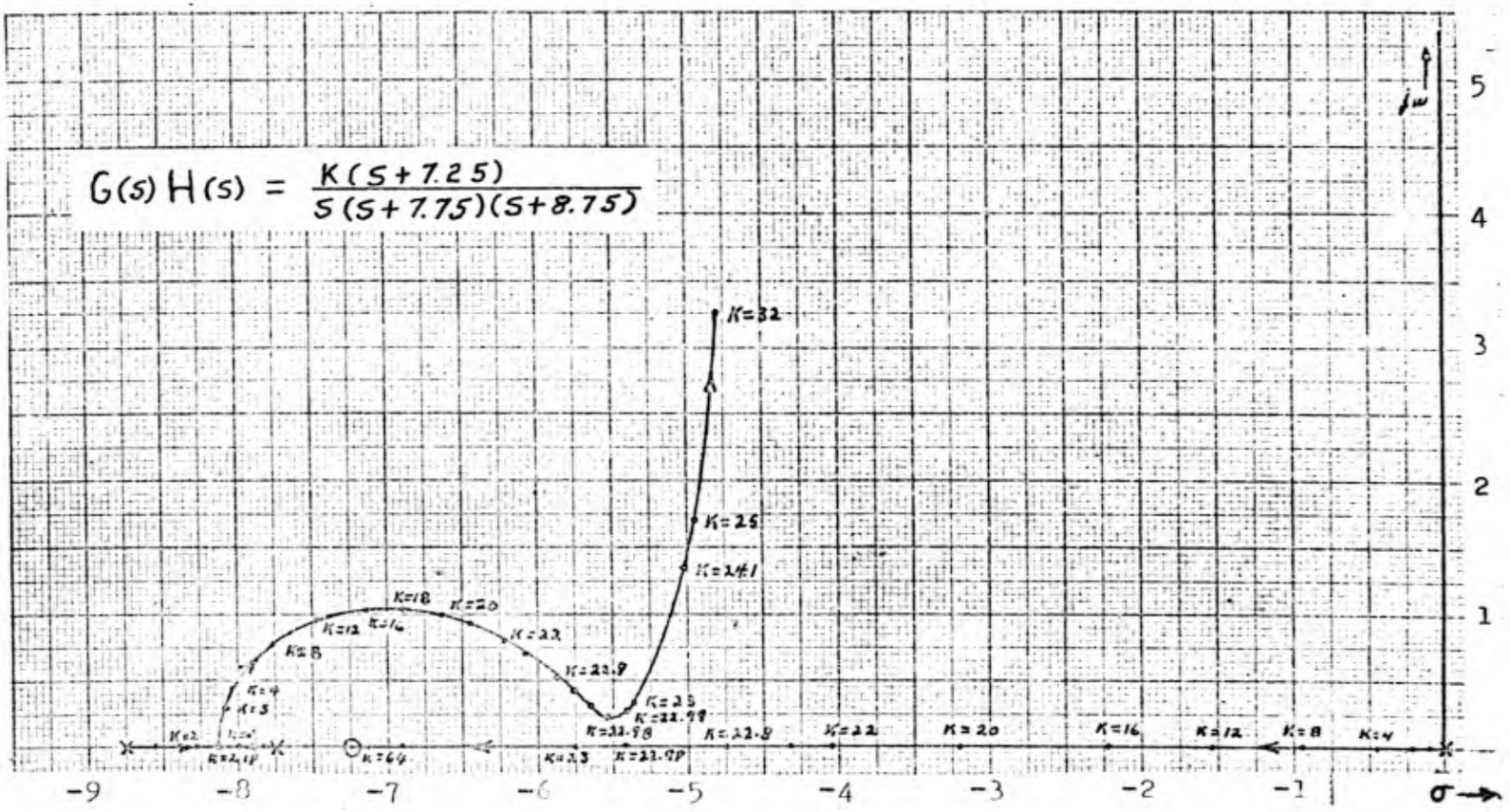
Appendix E

Root Locus for Two Zeros and Three Poles



Appendix F

Root Locus for Four Zeros and Five Poles



Appendix G

Root Locus for One Zero and Three Poles

Appendix H

Refinement to Program for Changing GRAT

To afford the operator the option of changing GRAT during program execution, the program may be changed as follows:

1. Change statement 86 to read

   86 (IF SENSE SWITCH 1) 90, 87

2. Add statement 90 to read

   90 (IF SENSE SWITCH 2) S, 3

3. Change statement at beginning of program which reads

   ACCEPT, G, GRAT

   to read

   3 ACCEPT, G, GRAT

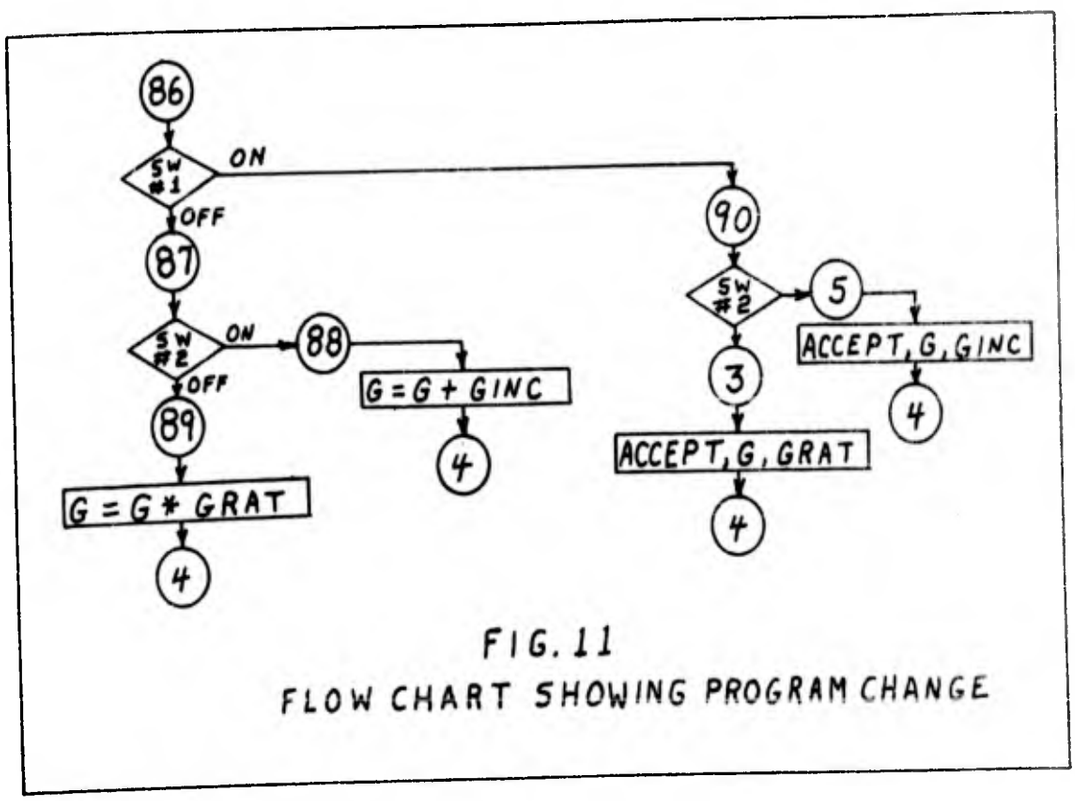


FIG. 11
FLOW CHART SHOWING PROGRAM CHANGE

Vita

Stuart Brown Herndon was born on [redacted]

[redacted]

Florida, he enlisted in the U. S. Navy and served as an enlisted man form 1947 to 1949. Upon his discharge, he enrolled in the U. S. Naval Academy, and in June 1953 was graduated with the degree of Bachelor of Science and was commissioned a Lieutenant in the U. S. Air Force. His military assignments prior to his coming to the Air Force Institute of Technology were as Navigator and Pilot in the Military Air Transport Service.

Permanent Address: [redacted]

This thesis was typed by Mrs. Stuart B. Herndon

UNCLASSIFIED