AN ABSTRACT OF THE THESIS OF

ROBERT ELWIN DIFFELY (Name) for the MASTER OF SCIENCE (Degree)

in MATHEMATICS (Major) presented on July 20, 1970 (Date)

Title: A FORTRAN PROGRAM FOR THE SIMULTANEOUS SOLUTION OF TWO ALGEBRAIC POLYNOMIAL EQUATIONS IN TWO UNKNOWNS

Redacted for Privacy

Abstract approved: Harry E. Goheen

The author investigates a FORTRAN program designed to find all solutions of two nonlinear polynomial equations in two unknowns.

For the method of Sylvester to be a method adaptable to a general purpose solver in this area, it will be necessary to have multiple precision capability combined with a much larger storage capacity than is presently available. The defects of the method are illustrated by working out a single precision program with provision for storage allocation of arrays of size ten. Applying this package to several examples, we see that the program actually works provided the bounds on storage and precision are not exceeded. However, this does occur in relatively simple examples.

A method of recursive evaluation of Sylvester's Determinant

is compared with the general recursive method for evaluating determinants and shown to be much better both in number of operations and in storage requirements than the latter method.

A Fortran Program for the Simultaneous Solution of Two Algebraic Polynomial Equations in Two Unknowns

by

Robert Elwin Diffely

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1972

APPROVED:

Redacted for Privacy

Professor of Mathematics

in charge of major

Redacted for Privacy

Acting Chairman of Department of Mathematics

Redacted for Privacy

Dean of Graduate School

Date thesis is presented July 20, 1971

Typed by Clover Redfern for Robert Elwin Diffely

TABLE OF CONTENTS

Chapter		Page
I.	INTRODUCTION	1
II.	SYLVESTER'S METHOD	2
III.	CONCLUSIONS	7
IV.	FORTRAN PROGRAM	12
V.	COMMENTS ON THE PROGRAM	47
	BIBLIOGRAPHY	51
	APPENDIX	52

LIST OF FIGURES

Figure		Page
1.	Flow chart--pivot element selection.	4
2.	Flow chart--comparison of roots.	6
3.	Block flow chart--PROGRAM SYLVESTER.	15
4.	Detailed flow chart--PROGRAM SYLVESTER.	16
5.	Block flow chart--SUBROUTINE SYLVES.	18
6.	Detailed flow chart--SUBROUTINE SYLVES.	19
7.	Block flow chart--SUBROUTINE SYL2.	25
8.	Detailed flow chart--SUBROUTINE SYL2.	26
9.	Detailed flow chart--SUBROUTINE PMULT.	30
10.	Detailed flow chart--SUBROUTINE PDIV.	30
11.	Detailed flow chart--SUBROUTINE PSUB.	32
12.	Detailed flow chart--SUBROUTINE PSHIFT.	32
13.	Detailed flow chart--SUBROUTINE WRIMAT.	34
14.	Detailed flow chart--SUBROUTINE MULT.	35
15.	Detailed flow chart--SUBROUTINE SUM.	37
16.	Block flow chart--EUCLID'S ALGORITHM.	39
17.	Detailed flow chart--SUBROUTINE COMFAC.	41
18.	Detailed flow chart--SUBROUTINE DEGREE.	43
19.	Detailed flow chart--SUBROUTINE DIV.	45

A FORTRAN PROGRAM FOR THE SIMULTANEOUS SOLUTION OF TWO ALGEBRAIC POLYNOMIAL EQUATIONS IN TWO UNKNOWNS

I. INTRODUCTION

The FORTRAN program presented in this thesis is designed to determine the solution set of two nonlinear polynomial equations in two unknowns. The equations are of the following form:

$$\begin{aligned} &E1: \quad A_1 x^m + A_2 x^{m-1} + \ldots + A_m x + A_{m+1} = 0 \\ &E2: \quad B_1 x^n + B_2 x^{n-1} + \ldots + B_n x + B_{n+1} = 0 \end{aligned} \tag{1}$$

where $A_1 \neq 0$, $B_1 \neq 0$, $m \geq 1$ and $n \geq 1$. Further, each of the coefficients of x (i.e., A_i or B_i) are of the form:

$$a_1 y^r + a_2 y^{r-1} + \ldots + a_r y + a_{r+1}$$

for some arbitrary value of $r \geq 0$. The a's and r may be different for each coefficient of x.

The algorithm used in the program is basically that algorithm presented by Holroyd (1) who shows that the variables may be separated by using Sylvester's method of elimination (3).

II. SYLVESTER'S METHOD

Sylvester's method of separating variables in (1) consists of entering the polynomials A_i and B_i in the determinant (2) such that:

1. The determinant contains $m+n$ rows and $m+n$ columns.
2. The elements in the first row consist of the polynomials $A_1, A_2, \ldots, A_{m+1}$ followed by $n-1$ zeroes.
3. The elements in row i $(2 \leq i \leq n)$ consist of $i-1$ zeroes followed by the polynomials $A_1, A_2, \ldots, A_{m+1}$, followed by $n-i$ zeroes.
4. The elements in row i $(n+1 \leq i \leq m+n)$ consist of $i-m-2$ zeroes followed by $B_1, B_2, \ldots, B_{n+1}$ followed by $2m+1-i$ zeroes.

The determinant (2) is then evaluated using recursive methods in such a manner that it always remains in the form (2) until only diagonal elements remain.

$$\begin{vmatrix}
A_1 & A_2 & A_3 & \cdots & A_m & A_{m+1} & 0 & 0 & \cdots & 0 \\
0 & A_1 & A_2 & \cdots & A_{m-1} & A_m & A_{m+1} & 0 & \cdots & 0 \\
0 & 0 & A_1 & \cdots & A_{m-2} & A_{m-1} & A_m & A_{m+1} & \cdots & 0 \\
\cdots & & & & & & & & & \cdots \\
0 & 0 & \cdots & A_1 & A_2 & \cdots & A_{m-1} & A_m & \cdots & A_{m+1} \\
B_1 & B_2 & \cdots & & B_n & B_{n+1} & 0 & 0 & \cdots & 0 \\
0 & B_1 & B_2 & \cdots & & B_n & B_{n+1} & 0 & & 0 \\
\cdots & & & & & & & & & \cdots \\
0 & 0 & \cdots & B_1 & B_2 & \cdots & & B_n & & B_{n+1}
\end{vmatrix} \qquad (2)$$

The determinant (2) is expanded by selecting a pivot element (either A_1 or B_1 or A_{m+1} or B_{n+1}) according to the following rules:

1. Use A_1 if $A_1 \neq 0$ and $m \leq n$.
2. Use B_1 if $B_1 \neq 0$ and $A_1 = 0$ and $m \leq n$.
3. Use A_{m+1} if $B_{n+1} = 0$ and $A_{m+1} \neq 0$ and $m > n$.
4. Use B_{n+1} if $B_{n+1} \neq 0$ and $m > n$.
5. If none of the above four rules applies, stop the process.

   The algorithm does not apply.

A flow chart of the pivot selection process is shown in Figure 1. Also shown in Figure 1 is the method of reduction used in the algorithm. R in the figure is used to contain intermediate values of the expansion of the determinant. At the end of the process R

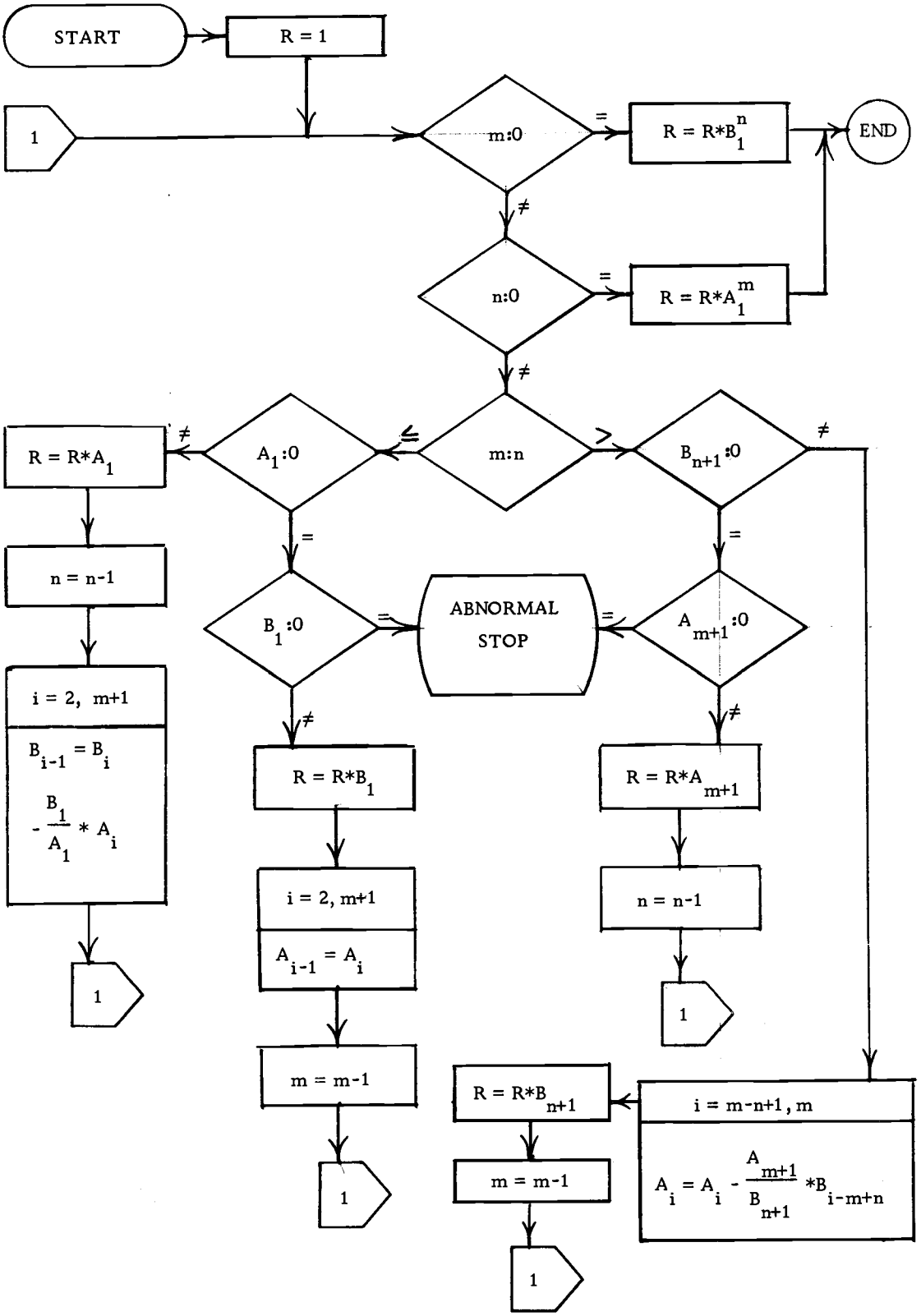

Figure 1. Flow chart--pivot element selection.

contains a polynomial representing the value of the determinant.

It should be noted that R, A_i and B_i are, in general, composed of two polynomials, one in the numerator and one in the denominator. Successful completion of the flow shown in Figure 1 (i.e., arrival at the END symbol) produces a single polynomial in the variable y. (Common factors in the numerator and denominator of R are periodically removed and the terminal value of R should have a denominator whose value is 1).

The next step in the program consists of finding the roots of the polynomial in y. These roots are found from a program developed by Noonchester (2) who uses Muller's method. For each of the roots found for y, an evaluation of the coefficients in x is made for both of the original equations. The roots are then found for each of these equations and compared. If one of the roots in the first equation is equal to (within some tolerance) any root of the second equation, then the program prints a pair consisting of this common root and the value of y. The program is designed to get all members of the solution set, and does this except for abnormal cases and except for round-off errors. The flow chart in Figure 2 describes this portion of the program.

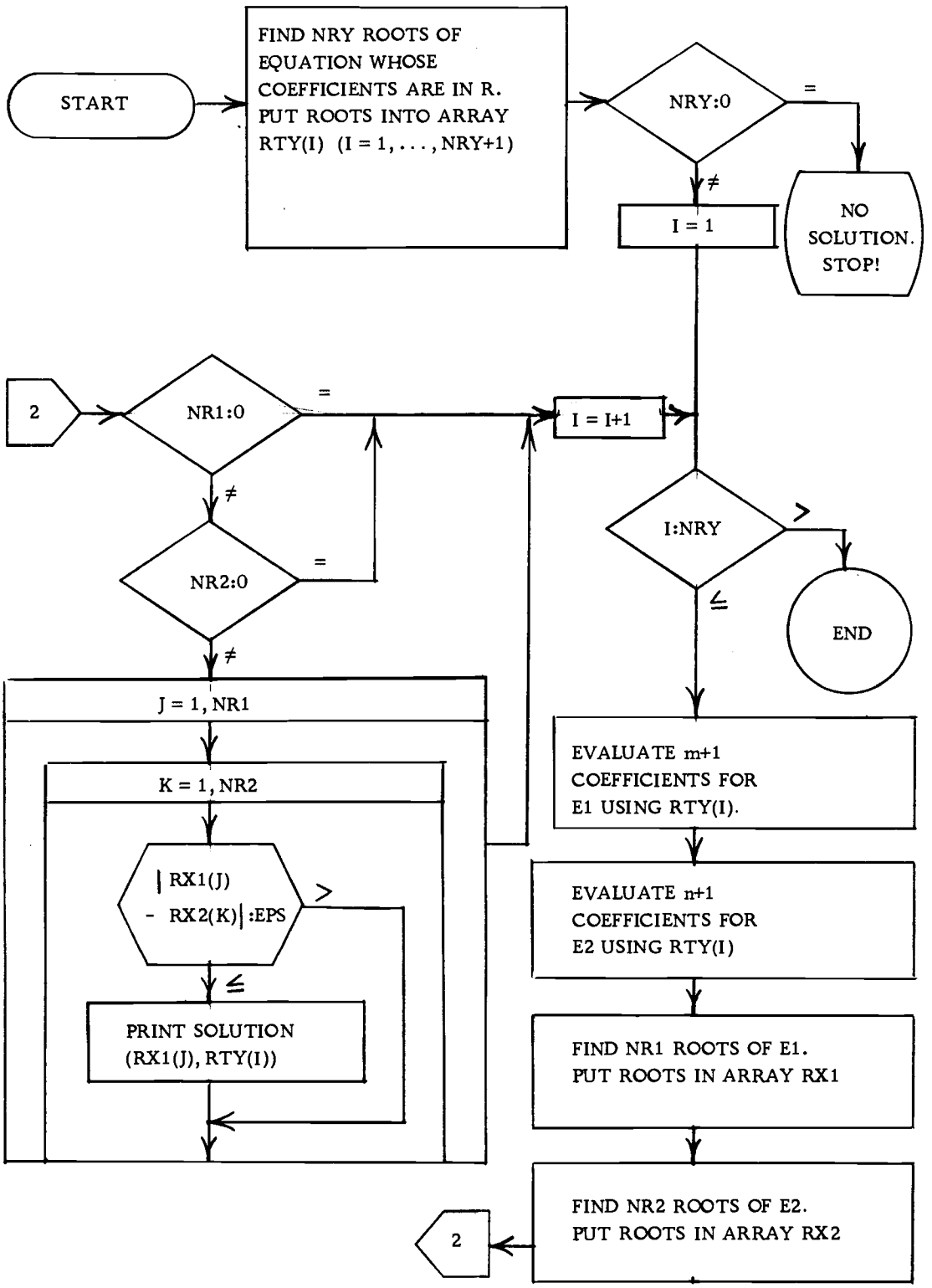

Figure 2. Flow chart--comparison of roots.

III. CONCLUSIONS

The program described in Chapter IV successfully solved the following three sets of equations:

Set 1: $yx^2 + (y+6)x + y = 0$

$(y-2)x^2 + (y+1)x + (y-2) = 0$

Equation in y: $y^2 - 8y + 16 = 0$

Roots Found

-.4999999 + 0i, 3.999999 + 0i

-.5000001 + 0i, 4.000001 + 0i

-1.999999 + 0i, 4.000001 + 0i

-2.000001 + 0i, 3.999999 + 0i

Set 2: $x^3 + (y+1)x + x^2 + (y^2+3y)x + (y+2) = 0$

$x^2 + (y+2)x + (y^2+4y+2) = 0$

Equation in y: $y^4 + 10y^3 + 33y^2 + 40y + 16 = 0$

Roots Found

.618034 + .000111i, -1.00000 + .000047i

.618034 - .000112i, -1.00000 + .000048i

-1.618034 - .000016i, -1.00000 + .000047i

-1.618034 - .000016i, -1.00000 - .000048i

Set 3: $x^2 + (y+4)x + (y^2+6y-1) = 0$

$x^2 + (2y+8)x + (y^2+5y-3) = 0$

Equation in y: $y^4 + 15y^3 + 74y^2 + 124y + 20 = 0$

Roots Found

3.000000 + 0i, -5,000000 + 0i

1.877404 + 0i, -6.279452 + 0i

0.476452 + 0i, -0.179910 + 0i

-3.353856 + 0i, -3.540637 + 0i

The following two sets of equations were not solved by the program for the reasons indicated:

Set 4: $x^3 + (y+3)x^2 + (y^2+4y)x + (y+6) = 0$

$x^3 + (y+1)x^2 + (y^2+2y-4)x - 2 = 0$

Reason for failure: storage overflow. Sufficient space was allowed only for polynomials of degree nine or less. This degree was exceeded during the course of the program.

Set 5: $(y-5)x^2 + (y^2-3y+2)x - (y+101) = 0$

$(-2y^2+13y-9)x^2 + (y^2+2)x + (y-16) = 0$

Reason for failure: program failed to find the common factors in the numerator and denominator of array R. This was caused by round

off error.

As seen by the preceding five sets of equations, it may be difficult, because of round off errors, to find satisfactory solutions to even relatively simple sets of equations (e.g., set 5). However, by using multiple precision methods (10 digits of accuracy was used in the examples) one would expect a significant increase in the complexity of the equations solvable by the method used here. Furthermore a simple increase in the size of the input bound on arrays from ten to say twenty will make the program of wider applicability.

The proposed method of evaluating the resultant is a recursive method, as is Gaussian elimination. However, Gaussian elimination as applied to the evaluation of Sylvester's determinant is not as efficient, as shown below.

One important difference between the two methods is the size of the arrays needed to contain the polynomial during intermediate steps. Gaussian elimination techniques require a single array of four dimensions (two of the dimensions would locate a particular polynomial fraction in the matrix, a third dimension would indicate the polynomial in the numerator or denominator of that fraction, and the fourth dimension would indicate the particular single coefficient wanted). The proposed method uses two arrays, each of three dimensions. An example points out the difference: assuming sufficient storage space for polynomials up to the 19^{th} degree, the space

required for Gaussian elimination is $20 \times 20 \times 20 \times 2 = 16,000$ storage locations. The equivalent space required by the proposed method is $2 \times 20 \times 20 \times 2 = 1600$ storage locations, which favors the proposed method by a factor of ten.

Another important difference between the two methods is the number of operations required to find a solution to a particular system of equations. The proposed method reduces, by one, the dimension of a matrix by operating on the elements (polynomial fractions) of just one row of the matrix. Gaussian elimination must (normally) operate on more than one row to reduce the dimension by one. Two examples follow.

Example 1. Assume a matrix of size $2n \times 2n$, (i.e., two sets of nth degree equations in two unknowns). Gaussian elimination techniques would (to reduce the dimension of the matrix) operate on first one row, then two rows, then three rows, etc. until an $n \times n$ matrix remained, at which time Gaussian techniques would operate on n rows, then $n-1$ rows and so forth until only one element remained in the determinant. Thus a total of $1+2+3+ \ldots +n-1+n+n+n-1 + \ldots +2+1 = n(n+1)$ row operations are required. This is contrasted with $2n-1$ row operations required for the proposed method. The ratio here is, roughly, $n/2$ which, again, is in favor of the proposed method.

Example 2. Assume a pair of equations such that the first

equation is of degree n-1 and the second equation is of degree 1. An n x n matrix results which has n-1 rows with two polynomial fractions in each row and one row with n polynomial fractions. In this example, both methods of reduction would need to operate on only one row at any time to reduce the dimension of the matrix by one.

As implied by the examples above, the evaluation of Sylvester's determinant should normally require considerably fewer operations by using the proposed method than by using Gaussian elimination. Furthermore, in no case should the proposed method require more operations than its counterpart. In any case, it would appear that the proposed method provides very significant advantages over Gaussian techniques when applied to determinants in the form of Sylvester's Determinant.

A comparison between the proposed method and an iterative method, such as Newton's method, reveals that the latter would require less storage space but may or may not converge. However, the proposed method always stops, having obtained all the roots, if the bounds on precision and storage are not exceeded. A comparison of the relative speeds of the two methods (assuming convergence of the iterative method) was not investigated.

IV. FORTRAN PROGRAM

There are three major FORTRAN programs described in this chapter (PROGRAM SYLVESTER, SUBROUTINE SYLVES and SUBROUTINE SYL2) along with many supporting subroutines. In most of these subroutines, information is represented by arrays of either three dimensions or two dimensions. In general, the three dimensional arrays correspond to actual real values of the coefficients of some polynomial. A related two dimensional array is used to store the degree of these polynomials. The array subscript notation is as follows:

1. Real arrays (three dimensional). If A(I, J, K) is the array element in question then I refers to the Ith coefficient (which is a polynomial in y) of a polynomial in x. J refers to the Jth coefficient of the polynomial in y. K refers to either the numerator (K = 1) or denominator (K = 2).
2. Integer arrays (two dimensional). If DA(I, K) is the array element in question then I refers to the Ith coefficient of x. K refers to the numerator (K = 1) or denominator (K = 2). The value of the element is the degree of the polynomial.

An example should help to clarify these rules. Assume a seventh

degree polynomial in x which has as the coefficient of x^4, the following polynomial fraction:

$$\frac{4y^3 + 2y + 1}{9y + 3}$$

then $I = 7 - 4 + 1 = 4$ and $A(4, 1, 1) = 4$, $A(4, 2, 1) = 0$, $A(4, 3, 1) = 2$, $A(4, 4, 1) = 1$, $A(4, 1, 2) = 9$, $A(4, 2, 2) = 3$, $DA(4, 1) = 3$, $DA(4, 2) = 1$. (DA is the array holding the degrees of the polynomials in array A.) For coefficients which are identically zero, the corresponding integer array is assigned the value minus one.

The remainder of this chapter is a series of detailed outlines describing each of the subroutines used by the author.

PROGRAM SYLVESTER is the main program used in the algorithm. Its basic function is to input EPS (a measure of the accuracy desired), and to input the coefficients of the two equations for which a solution set is sought. It then calls SUBROUTINE SYLVES which returns with a single equation in y, followed by a call to SUBROUTINE SYL2 which determines the solution set. In PROGRAM SYLVESTER, NA and NB are integers specifying the degrees of the first and second equations in x respectively. DAA(I) (I = 1,...,NA+1) and DAB(I) (I = 1,...,NB+1) are integer arrays containing the degree of the polynomial which is the Ith coefficient of x for the first and second equations respectively. AA(I, J)

(I = 1,...,NA+1; J = 1,...,DA(I)+1) and AB(I, J) (I = 1,...,NB+1; J = 1,...,DAB(I)+1) are real arrays which contain the coefficients of the polynomials in y. A block flow chart and a detailed flow chart are shown in Figures 3 and 4.

SUBROUTINE SYLVES constructs a single polynomial in y of degree NT with coefficients in array AT(i) (i = 1,...,NT+1). The input parameters to this subroutine are AA, NA, DAA, AB, NB, DAB and EPS. (These names are the ones defined in PROGRAM SYLVESTER). To determine the single polynomial in y, SYLVES constructs working matrices A, B, DA, DB to hold (initially) the contents of AA, AB, DAA and DAB. In addition, matrices R, DR (to hold the eliminant) C and DC (for intermediate storage) are defined. Arrays A, B, R and C are three dimensional real arrays whose subscript notation was explained at the beginning of this chapter. Arrays DA, DB, DR and DC are two dimensional integer arrays, (also described at the beginning of this chapter) which contain the various degrees of the polynomials whose coefficients are in A, B, R and C respectively. The variables M and P (both integer) are used in the program to hold the number of polynomials in arrays A and B respectively. SUBROUTINE WRIMAT writes the content of the array in its parameter list (i.e., the coefficients of the polynomials in y). SUBROUTINE PSHIFT shifts each polynomial one place to the left (e.g., $A_i \rightarrow A_{i-1}$). Three other

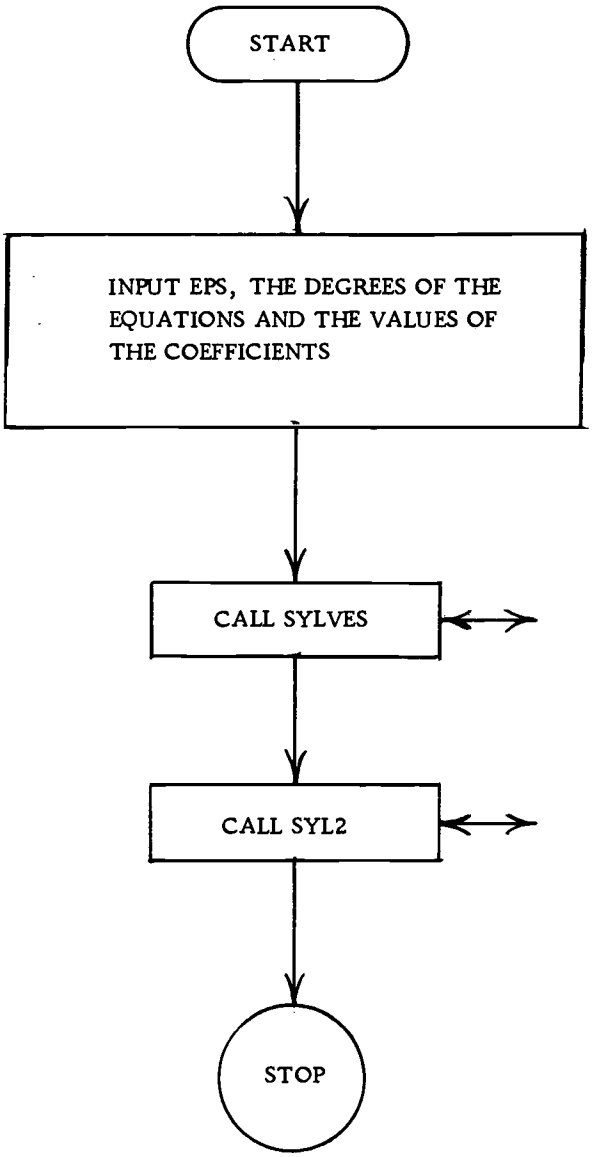

Figure 3. Block flow chart--PROGRAM SYLVESTER.

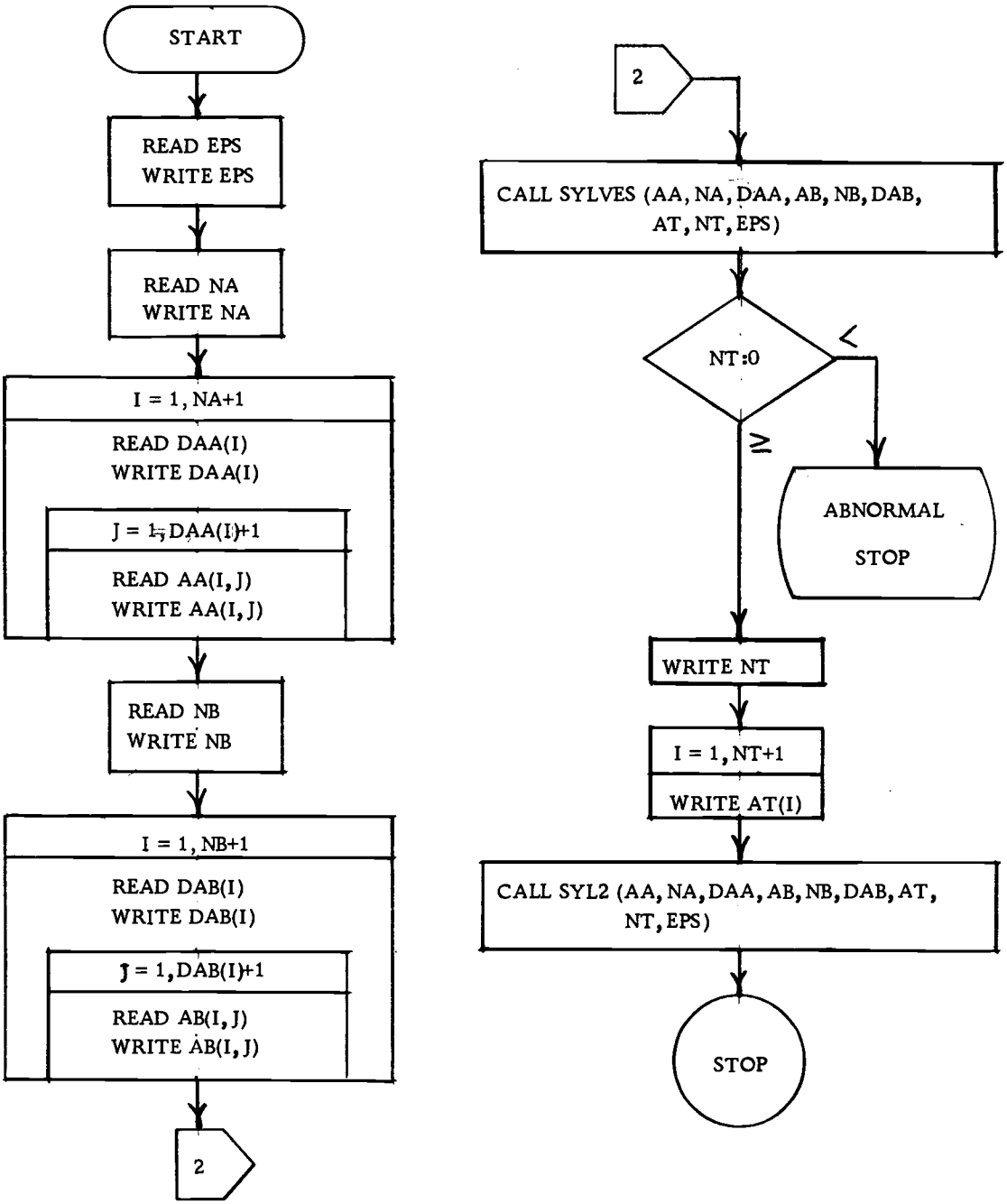

Figure 4. Detailed flow chart--PROGRAM SYLVESTER.

subroutines, PMULT, PDIV and PSUB are responsible for the polynomial arithmetic: multiplication, division and subtraction respectively. It should be remembered that the work performed by these three subroutines is actually concerned with polynomial fractions. For example, PSUB finds the difference

$$\frac{P_{1N}}{P_{1D}} - \frac{P_{2N}}{P_{2D}} = \frac{P_{1N}*P_{2D} - P_{1D}*P_{2N}}{P_{1D}*P_{2D}}$$

P's are single polynomials.

A block flow chart of SUBROUTINE SYLVES is shown in Figure 5 while Figure 6 shows a detailed flow chart.

SUBROUTINE SYL2 finds the roots of the polynomial in y which was output from SYLVES. For each of these roots, SYL2 evaluates the coefficients of x for each of the two input equations. It then finds the roots of each equation and compares each root of the first equation with each root of the second equation. If any of the roots in the comparison differ by an amount EPS or less, a solution set {root of x, root of y} is said to have been found.

Since some of the roots of the polynomials may be complex, most of the arrays used in SYL2 are complex arrays. In addition, the subscript notation is different from the notation explained at the beginning of this chapter. Figure 7 shows a block flow chart for SYL2 while Figure 8 shows a detailed flow chart.

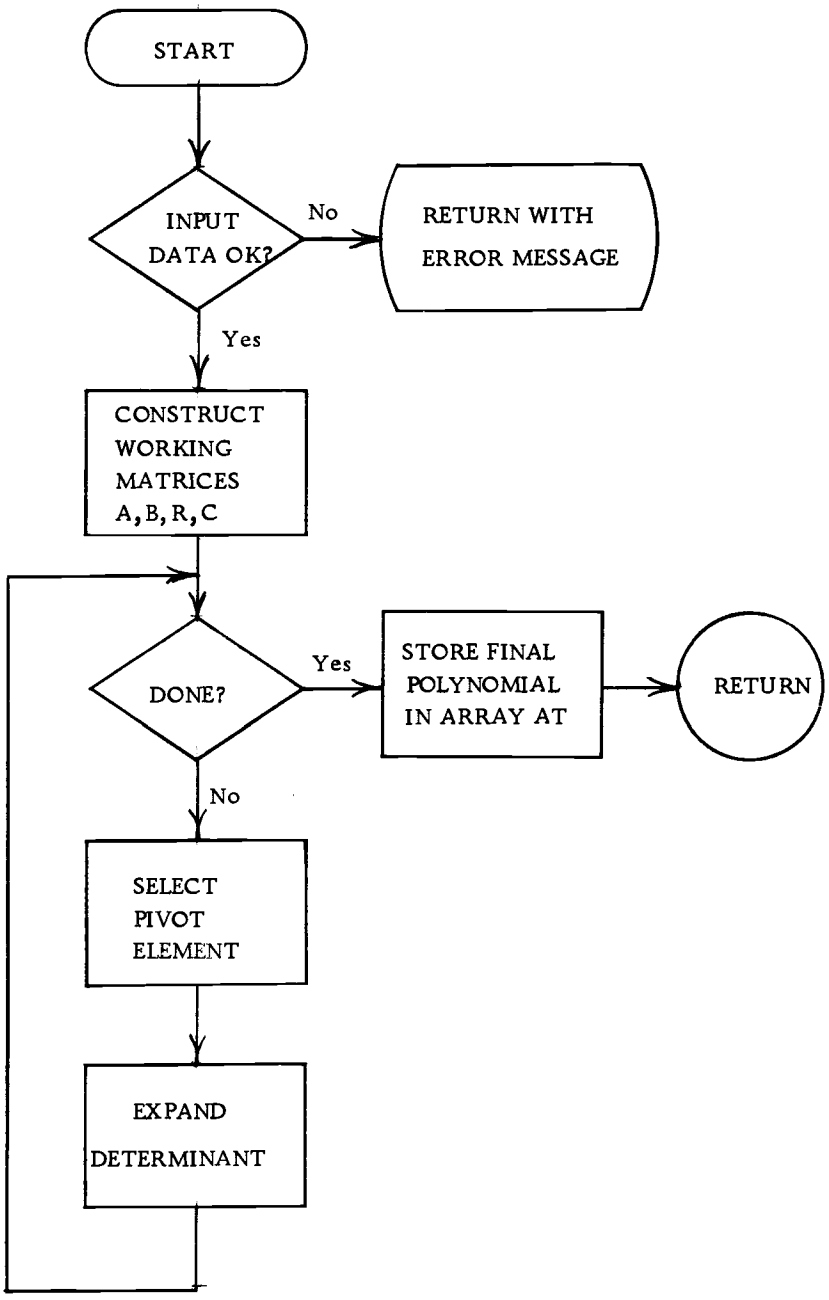

Figure 5. Block flow chart--SUBROUTINE SYLVES.

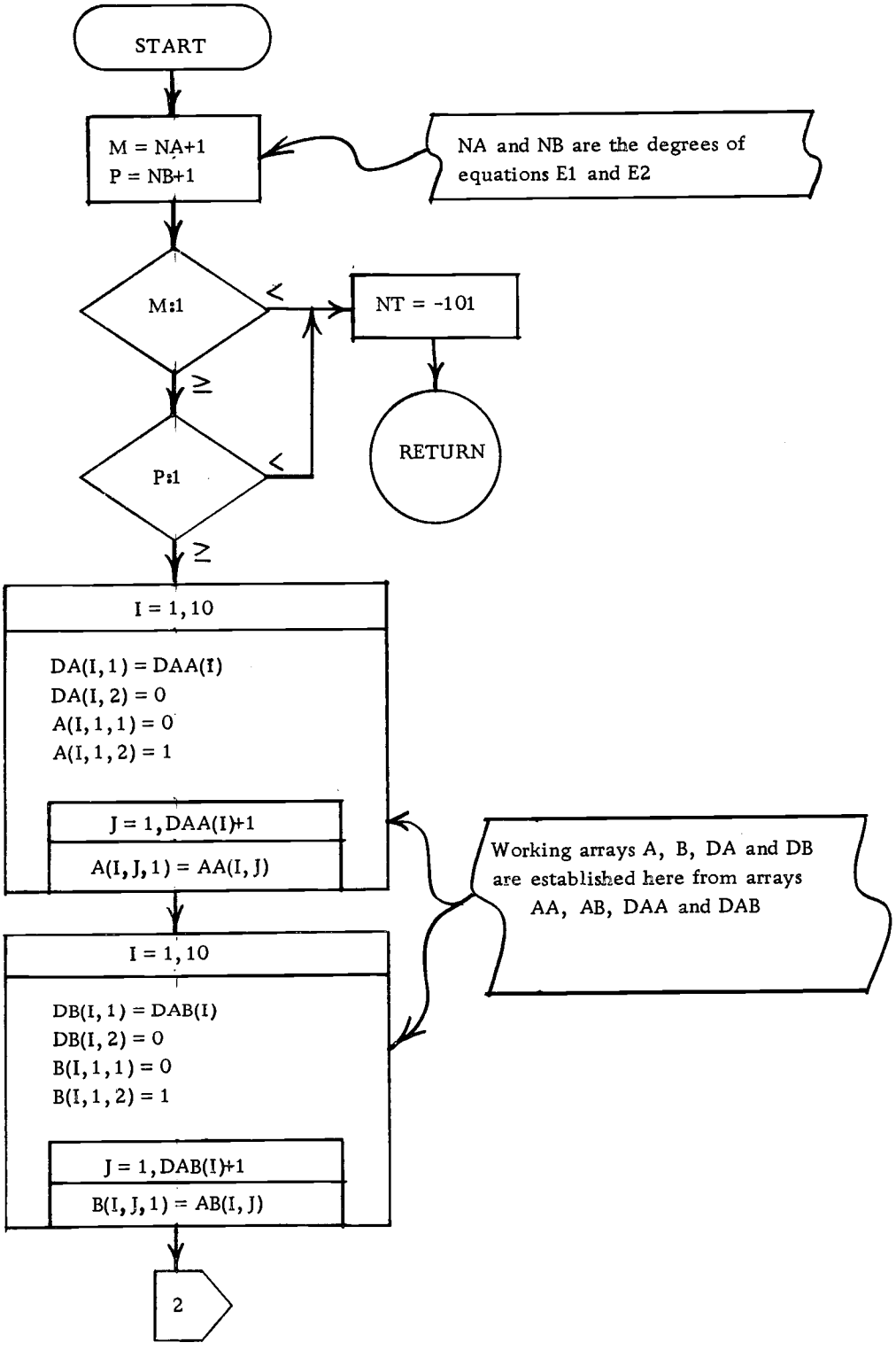

Figure 6. Detailed flow chart--SUBROUTINE SYLVES.

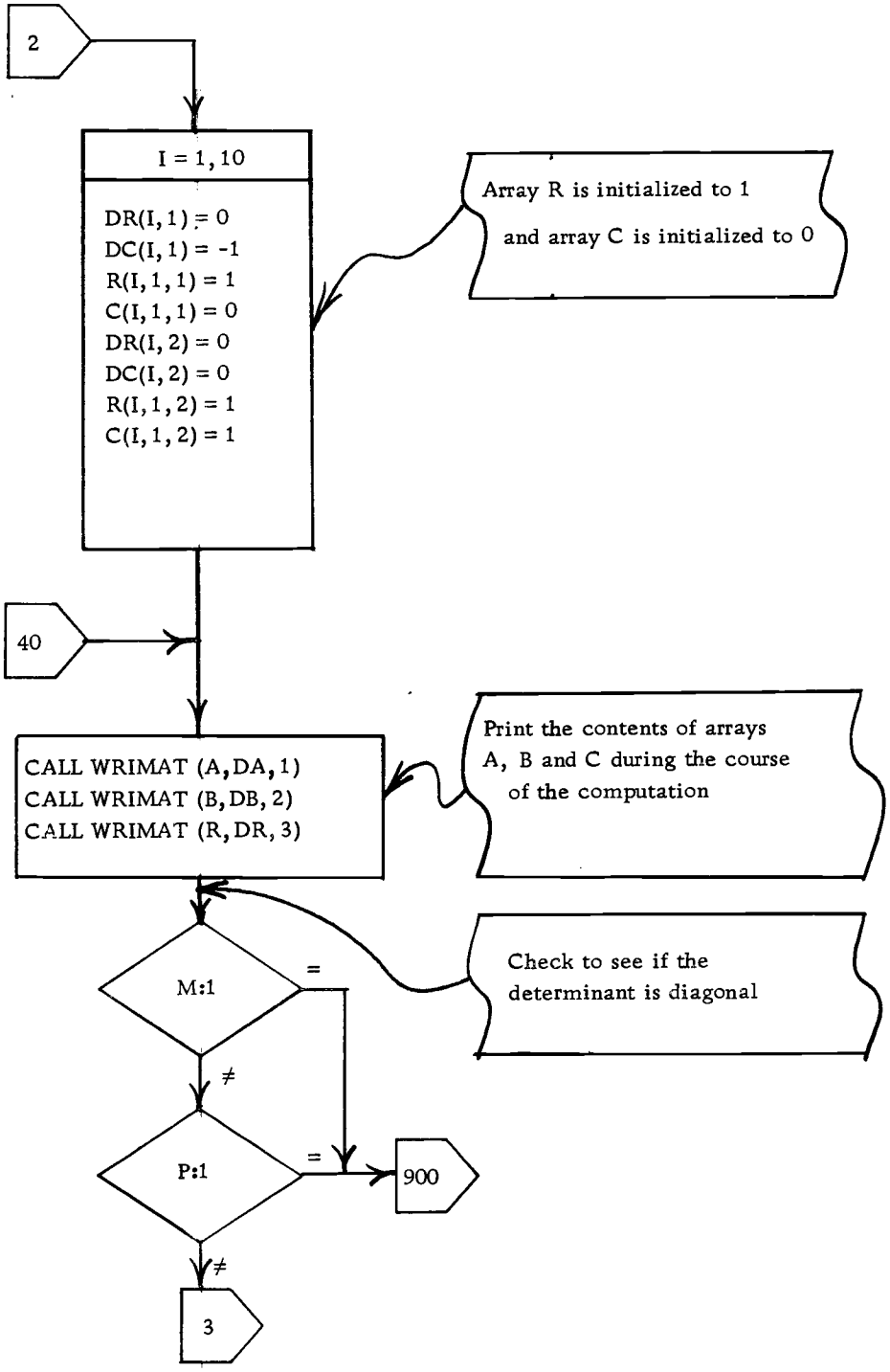

Figure 6. Continued.

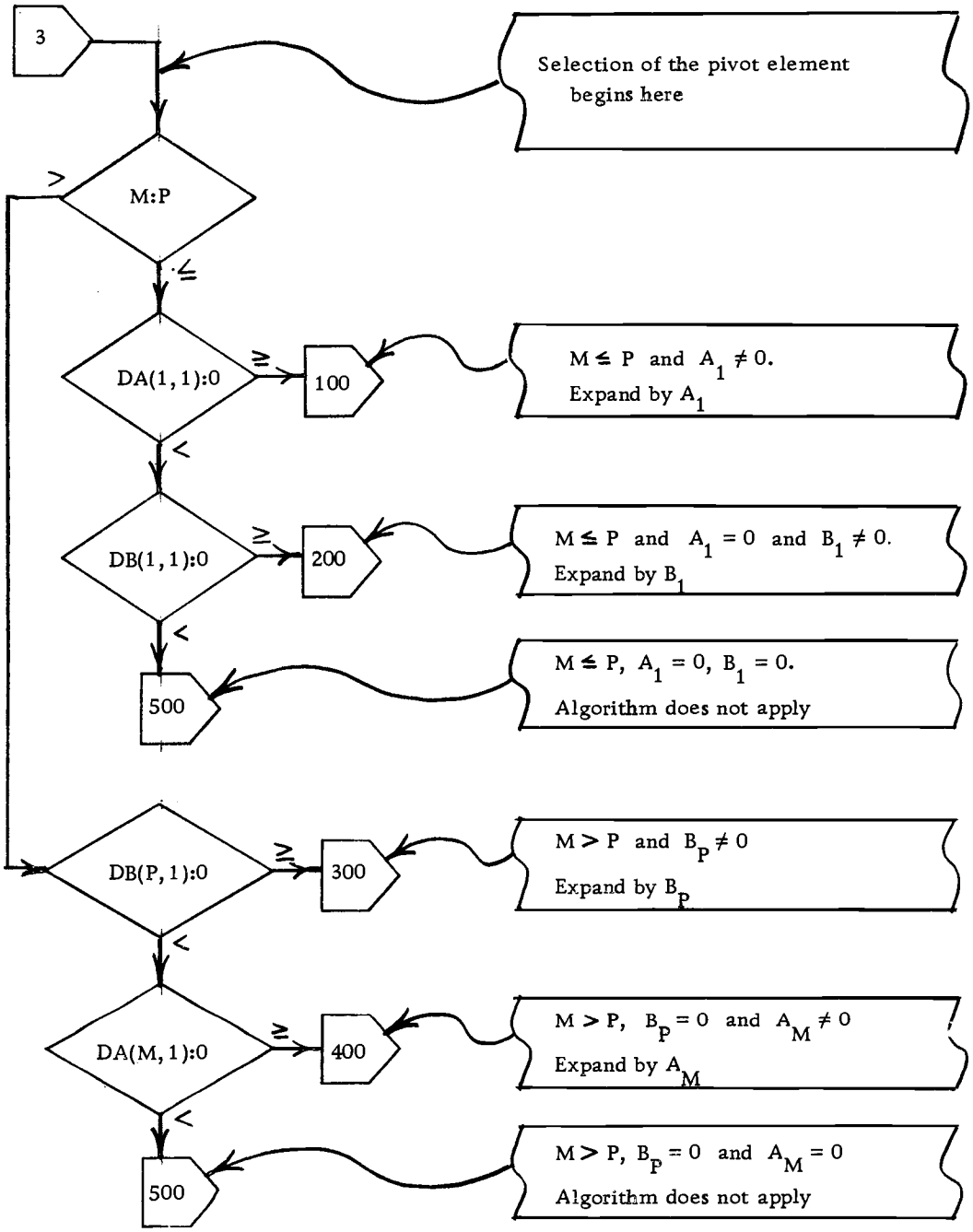

Figure 6. Continued.

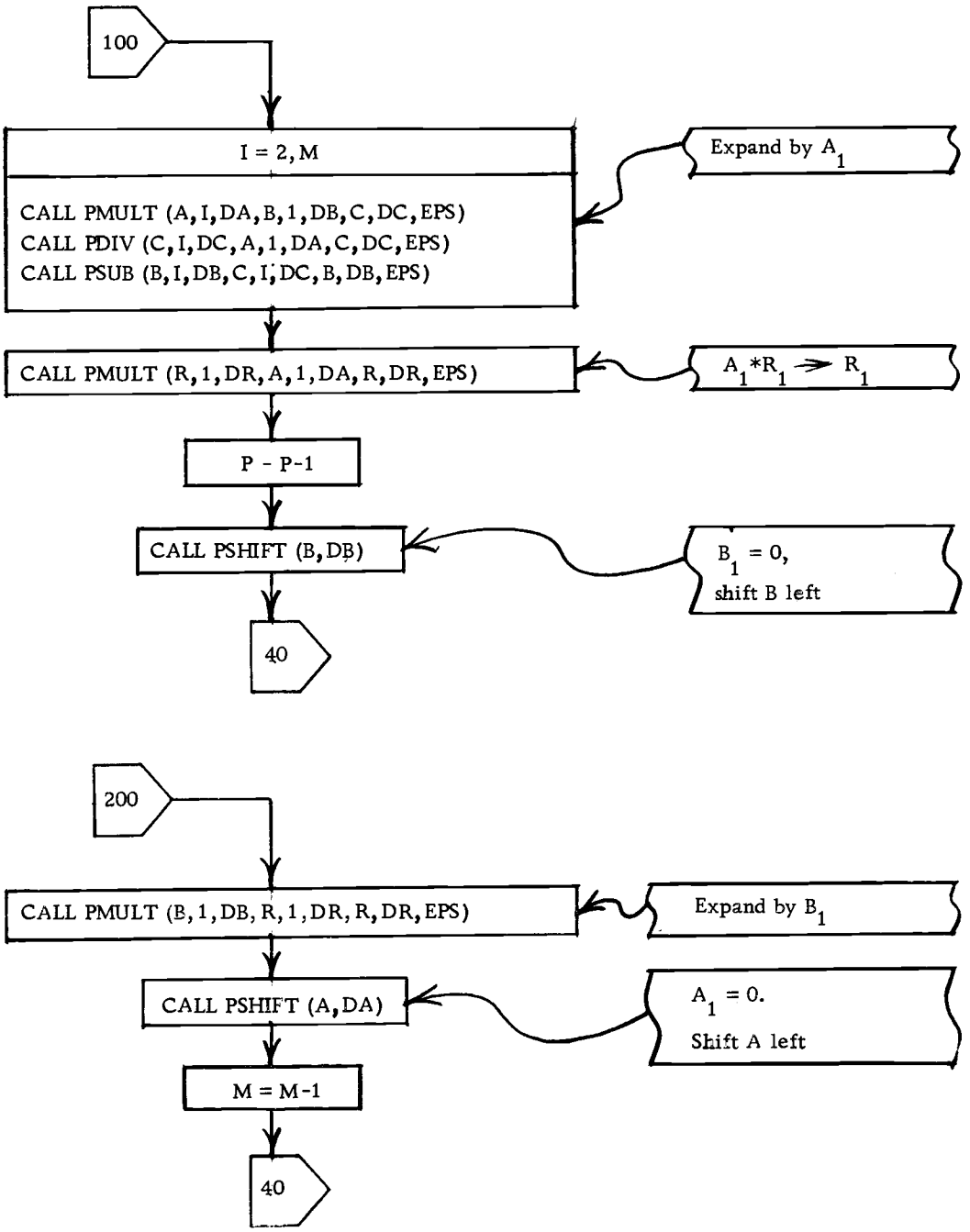

Figure 6. Continued.

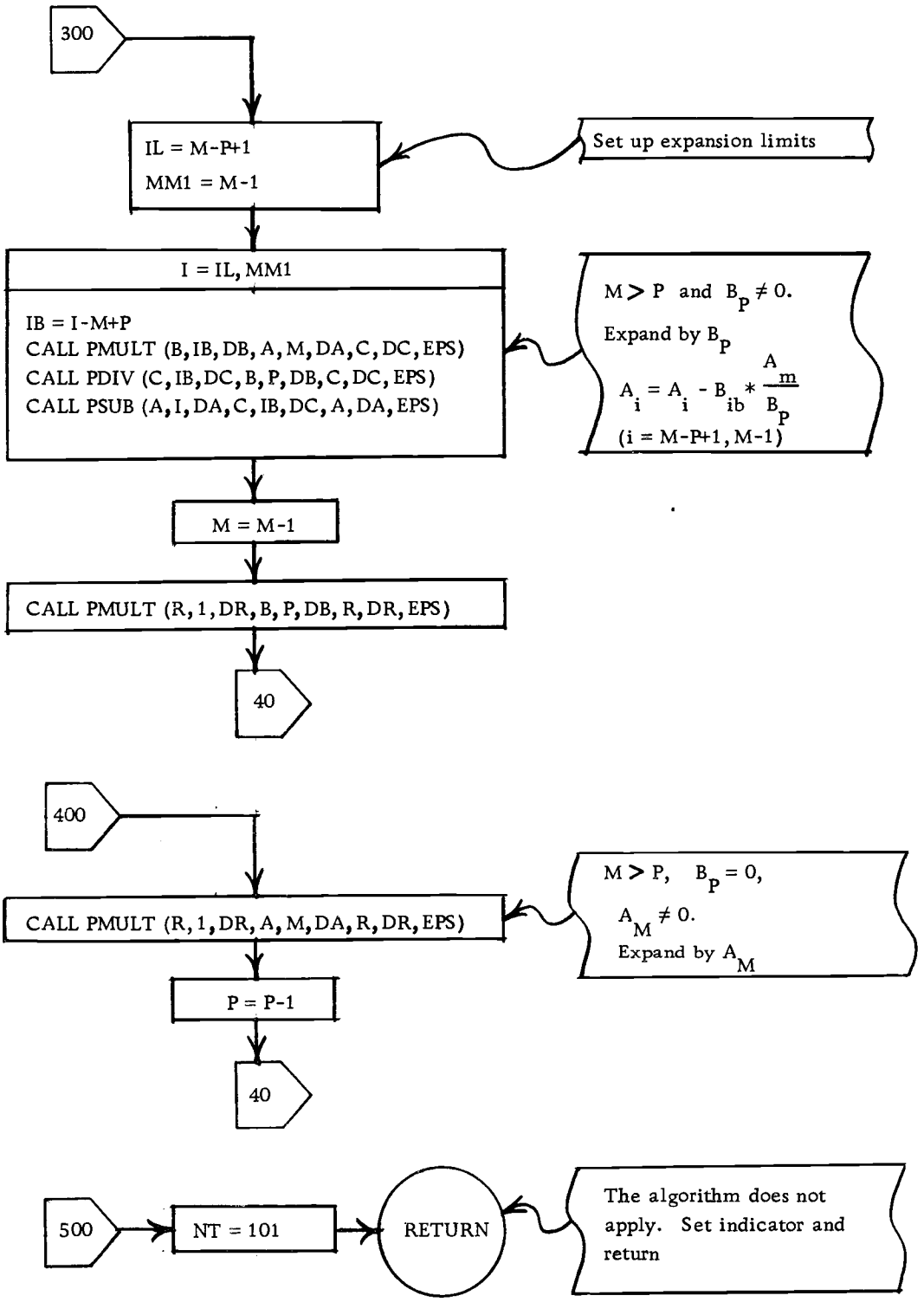

Figure 6. Continued.

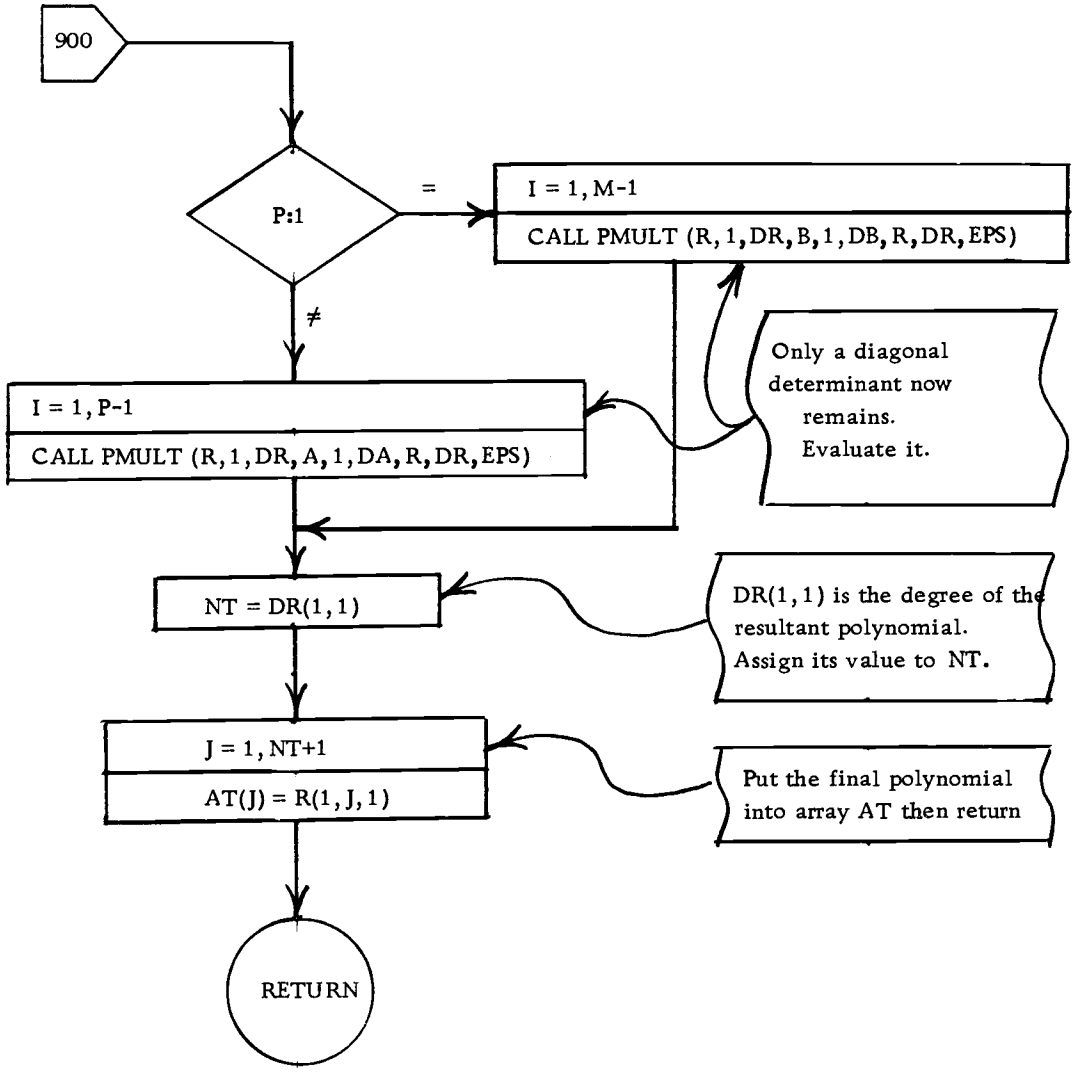

Figure 6. Continued.

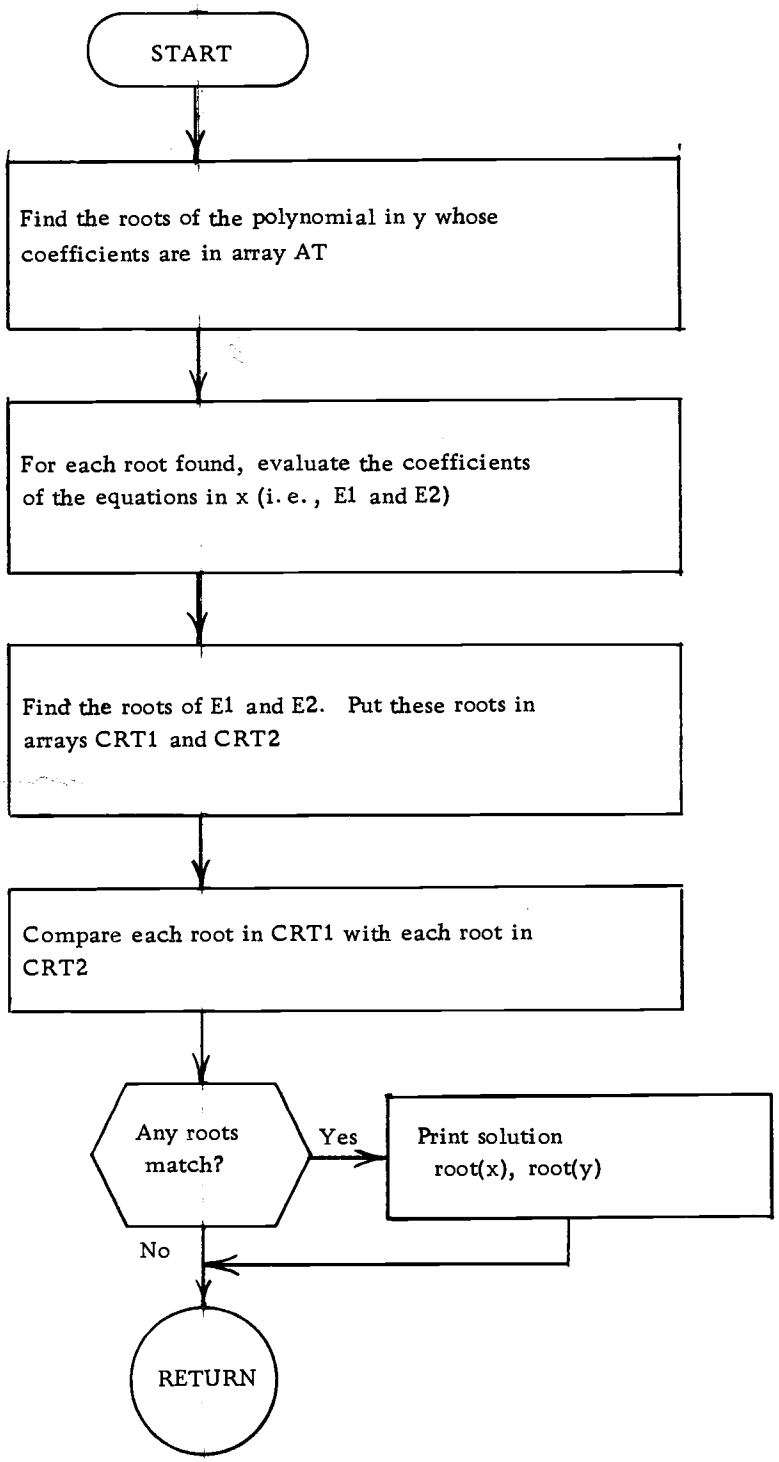

Figure 7. Block flow chart--SUBROUTINE SYL2.

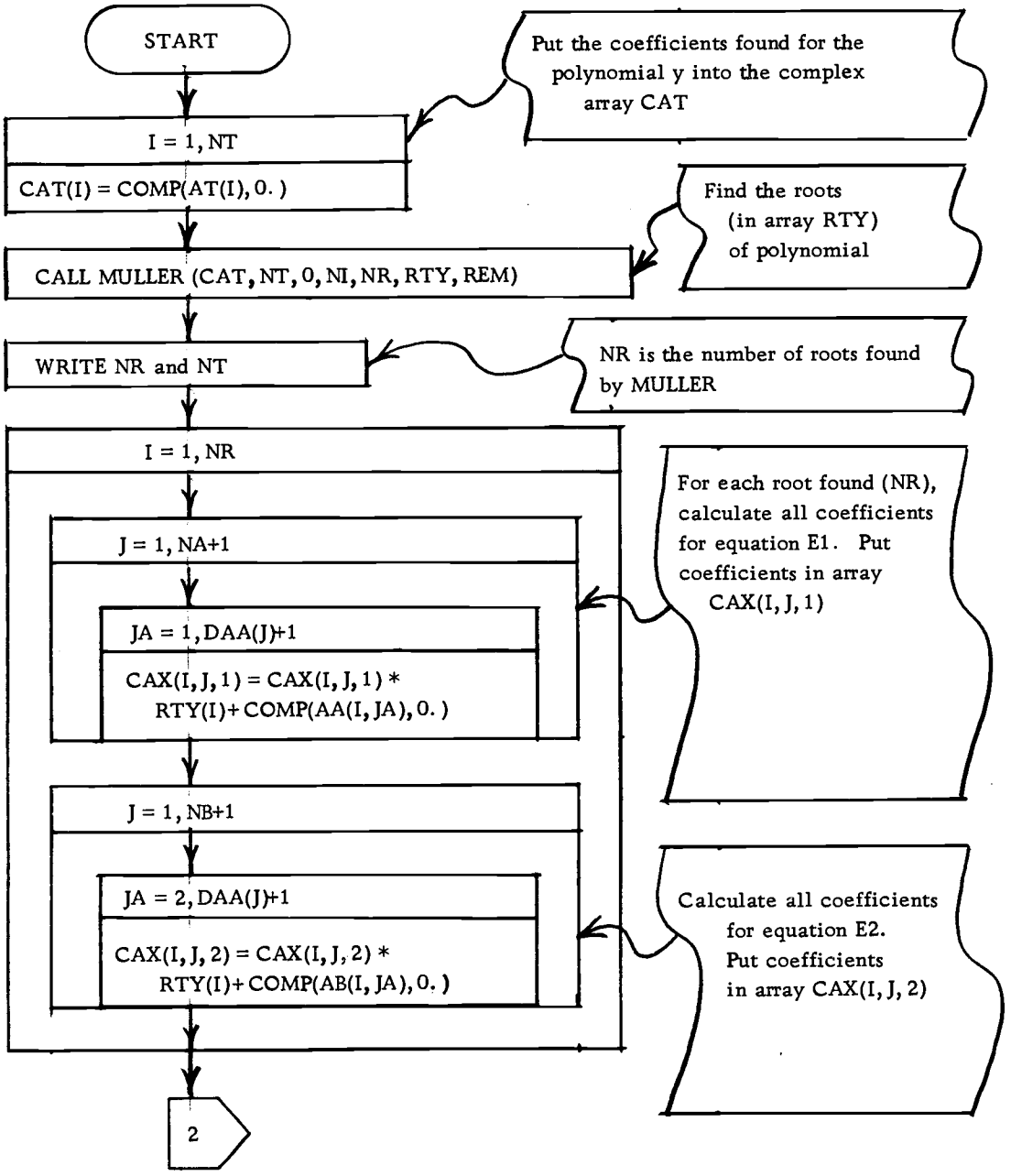

Figure 8. Detailed flow chart--SUBROUTINE SYL2.

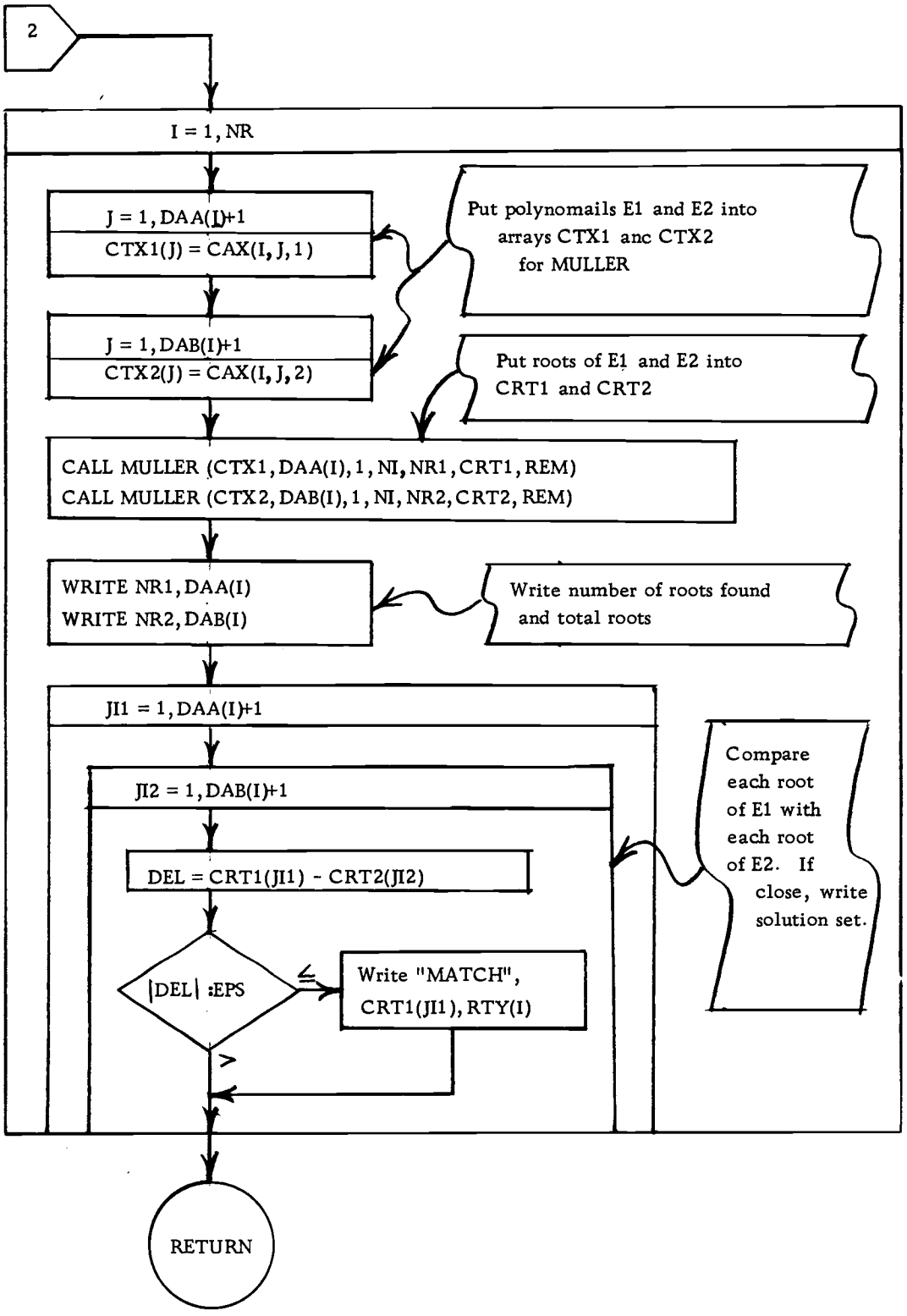

Figure 8. Continued.

SUBROUTINE PMULT determines the product of two polynomial fractions then factors out any common factors contained in the numerator and denominator of the product. The calling sequence is

CALL PMULT(A, I, DA, B, II, DB, P, DP, EPS).

Here A and B are arrays containing the polynomial fractions, DA and DB are arrays containing the degrees of the polynomial fractions and I and II are integers denoting which polynomial fractions are to be multiplied together. The product is put into the Ith position of array P which has polynomials whose degrees are in array DP. EPS is used by the COMFAC subroutine. In summary, PMULT calculates the product

$$P = \frac{A_N}{A_D} * \frac{B_N}{B_D} = \frac{A_N * B_N}{A_D * B_D}$$

where A_N and A_D are polynomials in the numerator and denominator of array A and B_N and B_D are polynomials in the numerator and denominator of array B. The product is placed in array P after common factors are removed. Figure 9 shows the flow chart for SUBROUTINE PMULT.

SUBROUTINE PDIV is used to divide one polynomial fraction by a second polynomial fraction. The calling sequence is

CALL PDIV(N, I, DN, D, II, DD, Q, DQ, EPS)

Here the Ith polynomial fraction in array N is divided by the IIth polynomial fraction in array D. The result (another polynomial fraction) is placed in the Ith position of array Q. Integer arrays DN, DD and DQ contain the degrees of the polynomials in arrays N, D and Q respectively. EPS is a real number used by the COMFAC subroutine. In effect, PDIV performs the following computation:

$$Q = \frac{N_N}{N_D} \Big/ \frac{D_N}{D_D} = \frac{N_N * D_D}{N_D * D_N} .$$

Common factors are removed from the numerator and denominator of Q before the result is entered into array Q. Figure 10 is a flow chart describing the action of SUBROUTINE PDIV.

SUBROUTINE PSUB finds the difference between two polynomial fractions. The calling sequence for PSUB is

CALL PSUB(A, I, DA, B, II, DB, S, DS, EPS).

Here the polynomial fraction in array B which is specified by the value of II is subtracted from the polynomial fraction in array A specified by the content of I. The result is placed in the position of array S specified by I. Integer arrays DA, DB and DS contain the degrees of the polynomials in arrays A, B and S respectively. EPS is used by the COMFAC subroutine.

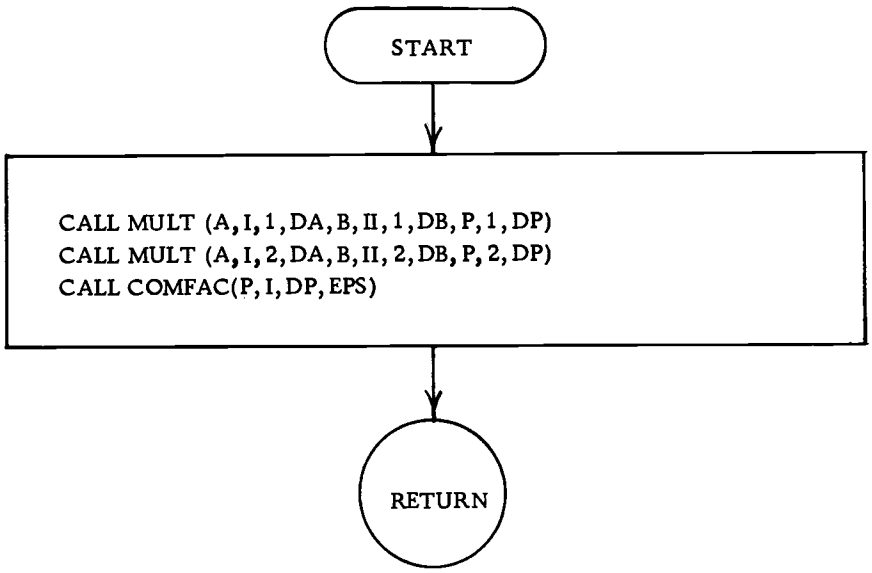

Figure 9. Detailed flow chart--SUBROUTINE PMULT.

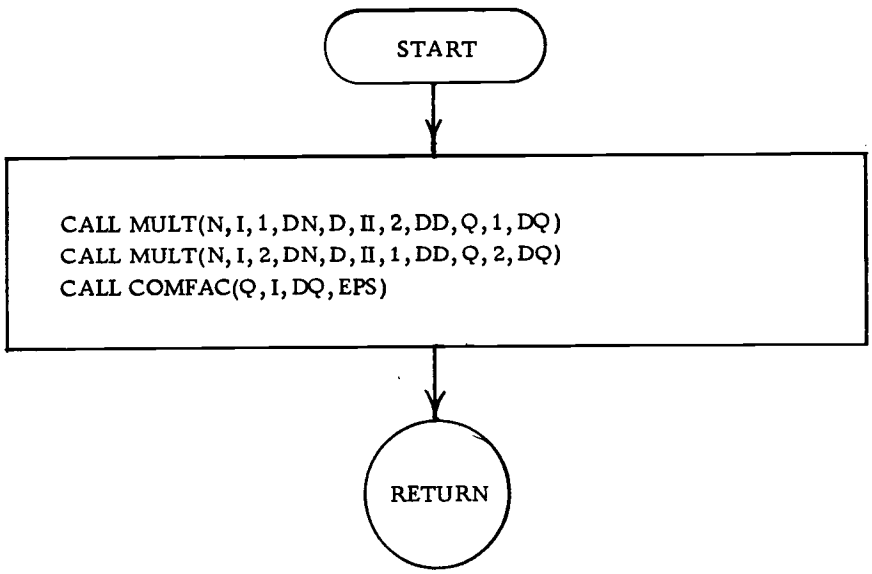

Figure 10. Detailed flow chart--SUBROUTINE PDIV.

PSUB performs the calculation

$$S = \frac{A_N}{A_D} - \frac{B_N}{B_D} = \frac{A_N * B_D - A_D * B_N}{A_D * B_D} .$$

DA, DB and DS contain the degrees of the polynomials in arrays A, B and S respectively. Figure 11 shows the flow of SUBROUTINE PSUB.

SUBROUTINE PSHIFT is used to shift all of the polynomial fractions in an array one place to the left, i.e., $A_i \rightarrow A_{i-1}$ $(i = 2, 3, \ldots)$ where A_i is the ith polynomial fraction in the array to be shifted. The calling sequence is

CALL PSHIFT(A, DA)

where A is the array containing the polynomial fractions to be shifted and integer array DA contains the degrees of the polynomials to be shifted. Figure 12 shows a detailed flow chart for PSHIFT.

SUBROUTINE WRIMAT is an output routine which prints all nonzero elements of a real array which contains polynomials. The calling sequence is

CALL WRIMAT(A, DA, KN).

Here A is the array whose elements are to be printed and DA is

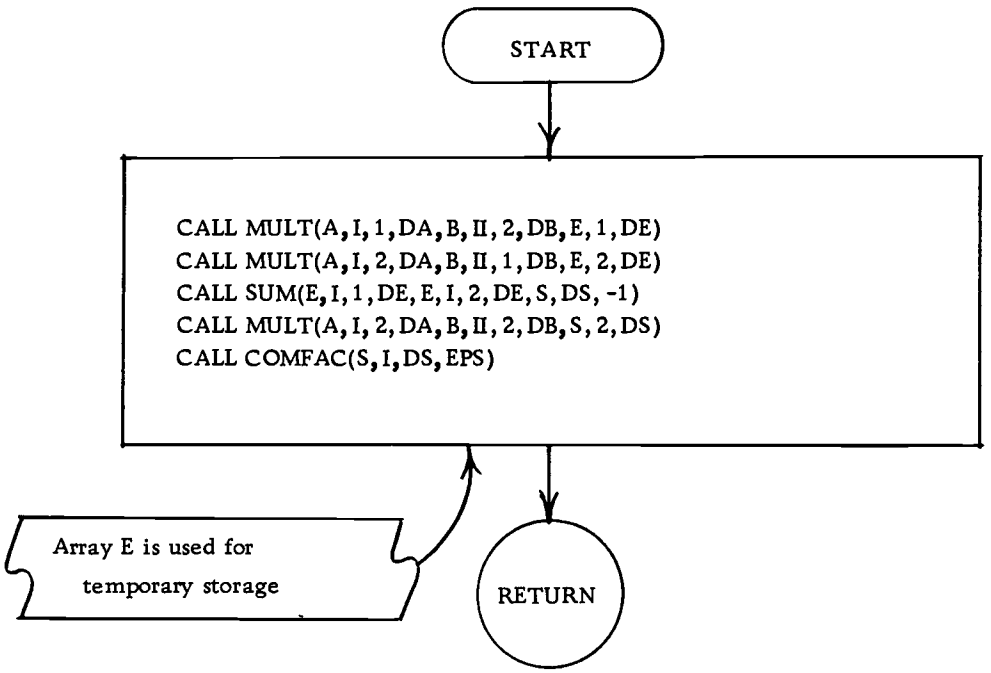

Figure 11. Detailed flow chart--SUBROUTINE PSUB.

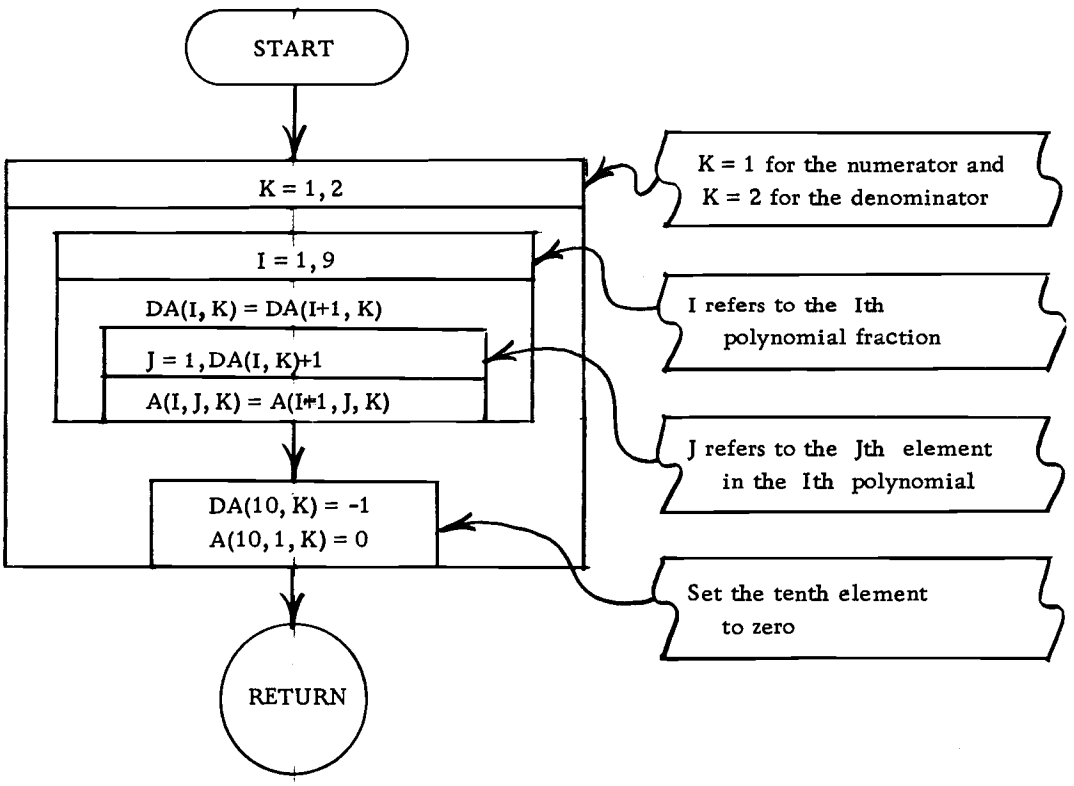

Figure 12. Detailed flow chart--SUBROUTINE PSHIFT.

an integer array containing the degrees of the polynomials in array A. KN is an arbitrary integer which is printed for identification purposes. The flow chart for WRIMAT is shown in Figure 13.

SUBROUTINE MULT calculates the product of two single polynomials. The calling sequence for MULT is

CALL MULT(A, I, K, DA, B, II, KK, DB, P, KKK, DP).

Here, real arrays A and B contain the polynomials to be multiplied while real array P provides the storage for the product. Integer arrays DA, DB and DP contain the degrees of the polynomials in arrays A, B and P respectively. Integers I and II refer to the Ith polynomial in A and the IIth polynomial in B, respectively. (The product of these polynomials is placed in the Ith position of P.) Integers K, KK, and KKK have values of 1 (numerator) or 2 (denominator) and refer to arrays A, B and P respectively. Thus each of the groups (A, I, K), (B, II, KK) and (P, I, KKK) completely specify a particular single polynomial. Figure 14 shows the flow chart for SUBROUTINE MULT.

SUBROUTINE SUM calculates either the sum or difference of two single polynomials. The calling sequence for SUM is

CALL SUM(A, I, K, DA, B, II, KK, DB, S, DS, IOP).

Here IOP is an integer specifying whether the sum ($IOP \geq 0$) or

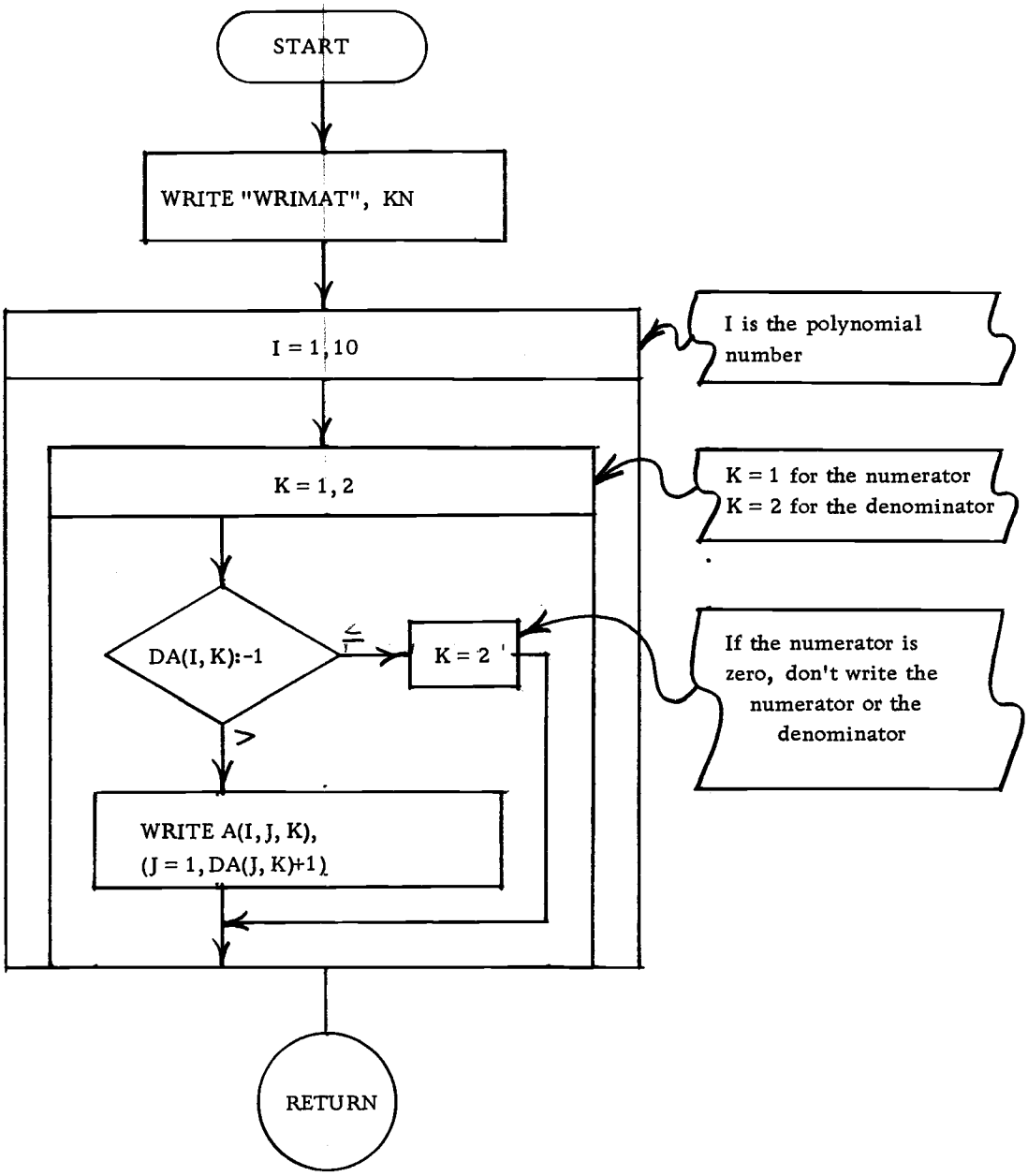

Figure 13. Detailed flow chart-- SUBROUTINE WRIMAT.

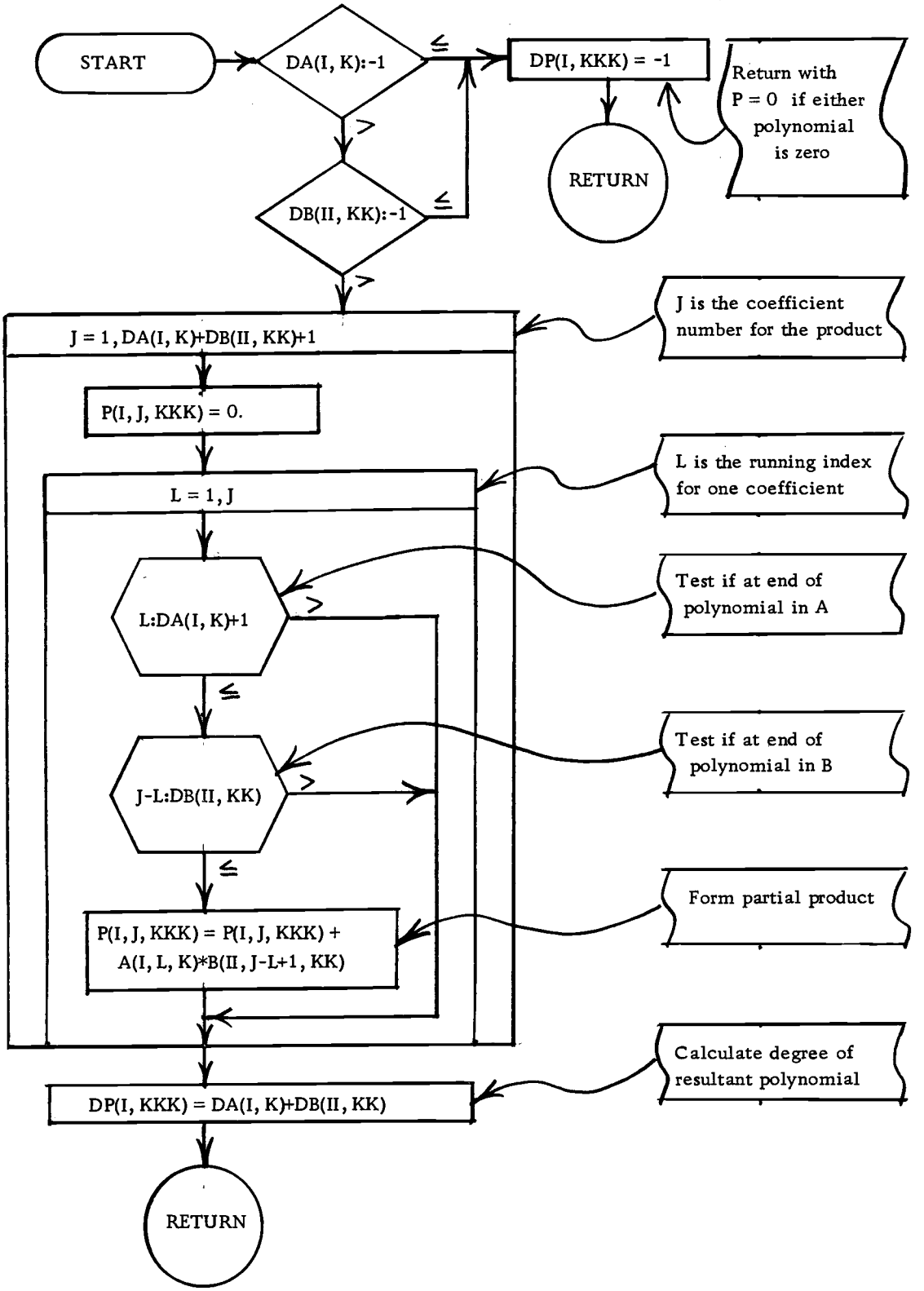

Figure 14. Detailed flow chart--SUBROUTINE MULT.

difference (IOP < 0) is to be taken between two polynomials. The polynomials in question are specified by (A, I, K) and (B, II, KK) where A and B are real arrays containing polynomials. I and II refer to the Ith or IIth polynomial fraction in arrays A and B respectively while K and KK refer to the polynomial in the numerator (K = 1 or KK = 1) or in the denominator (K = 2 or KK = 2). The sum or difference of these polynomials is placed in array S in a position specified by I and K. The flow chart for SUM (Figure 15) shows the use of a separate real array E. This array is used to hold intermediate results during the course of the program and allows the array S to be the same array as either A or B.

SUBROUTINE COMFAC is a routine which removes common factors from a polynomial fraction. The calling sequence for COMFAC is

CALL COMFAC(A, I, DA, EPS)

Here A is a real array containing polynomial fractions, I denotes the Ith polynomial fraction and DA is an integer array which contains the degrees of the polynomials in array A. EPS is a real number used by SUBROUTINE DEGREE.

COMFAC uses Euclid's algorithm for determining common factors. This algorithm is shown in Figure 16 as applied to

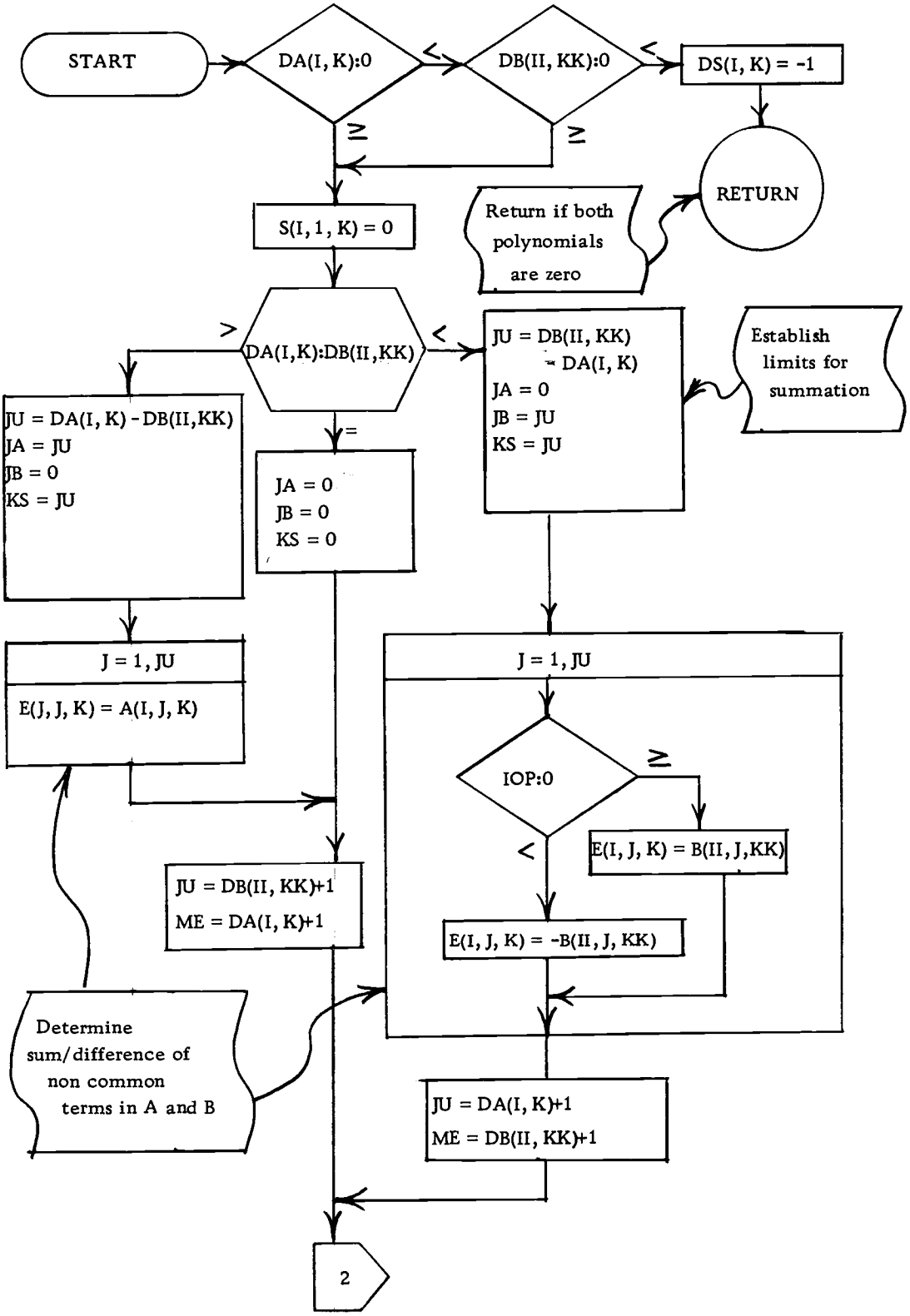

Figure 15. Detailed flow chart--SUBROUTINE SUM.

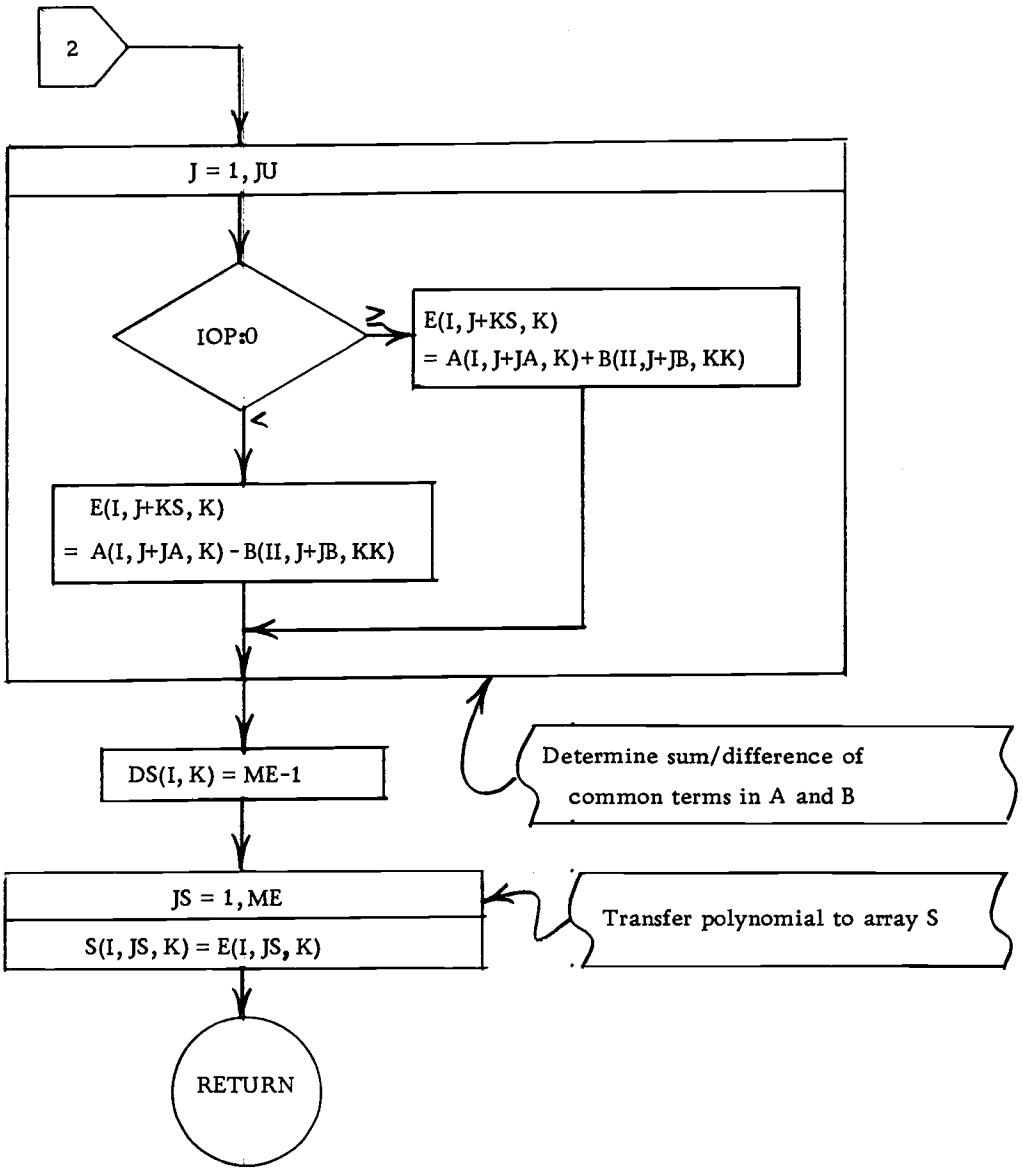

Figure 15. Continued.

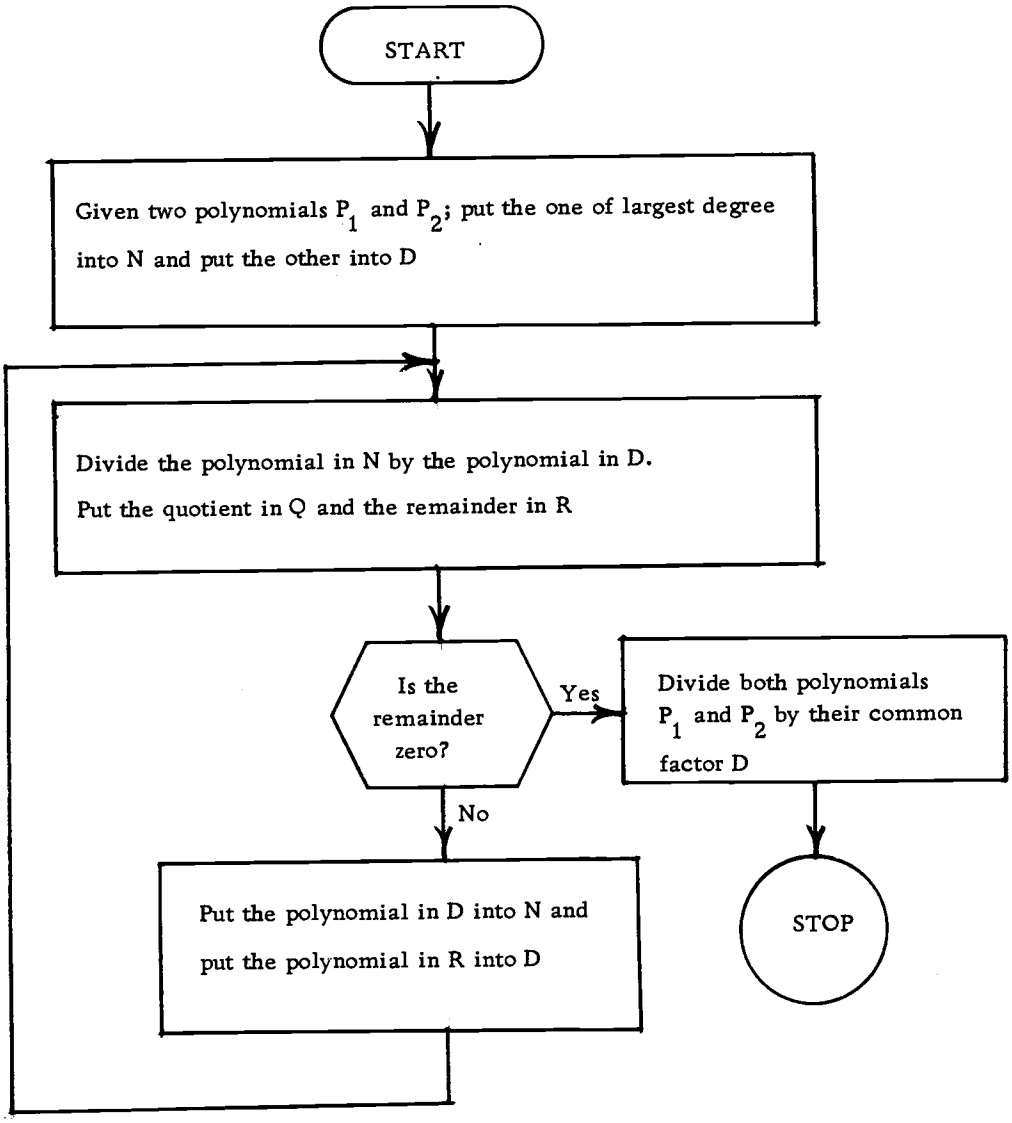

Figure 16. Block flow chart--EUCLID'S ALGORITHM.

polynomials while Figure 17 shows a detailed flow chart of COMFAC. In the flow chart, real arrays N, Q and R are used to hold intermediate results.

SUBROUNTINE DEGREE is used to check if any element in a polynomial is less than a number EPS. If an element is less than EPS then that element is set equal to zero. Further, if that element is the coefficient of the highest degree for that polynomial then the degree is reduced by one and all other coefficients are shifted left one place. The calling sequence for DEGREE is

CALL DEGREE(A, I, DA, EPS)

where A is a real array containing polynomial fractions and I indicates the Ith polynomial fraction. Integer array DA contains the degrees of the polynomials in array A. EPS is a real number representing the smallest allowed value of any coefficient. Figure 18 is a flow chart describing the operation of DEGREE.

SUBROUTINE DIV divides one single polynomial by another which yields a polynomial quotient and a polynomial remainder. The calling sequence for this subroutine is

CALL DIV(N, I, K, DN, D, II, KK, DD, Q, DQ, R, DR)

The parameter list contains real arrays N, D, Q and R which contain polynomials which are the dividend, divisor, quotient and

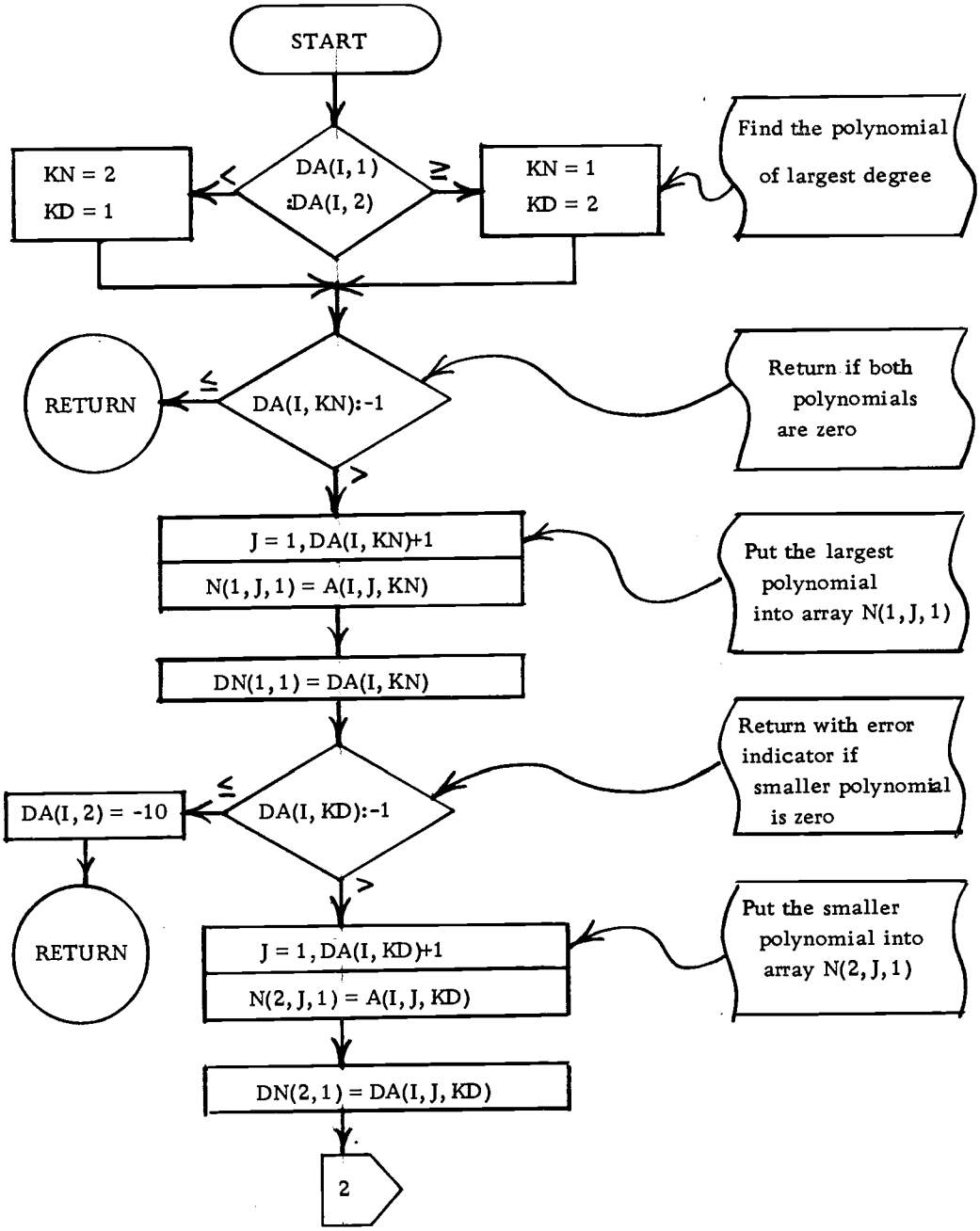

Figure 17. Detailed flow chart--SUBROUTINE COMFAC.

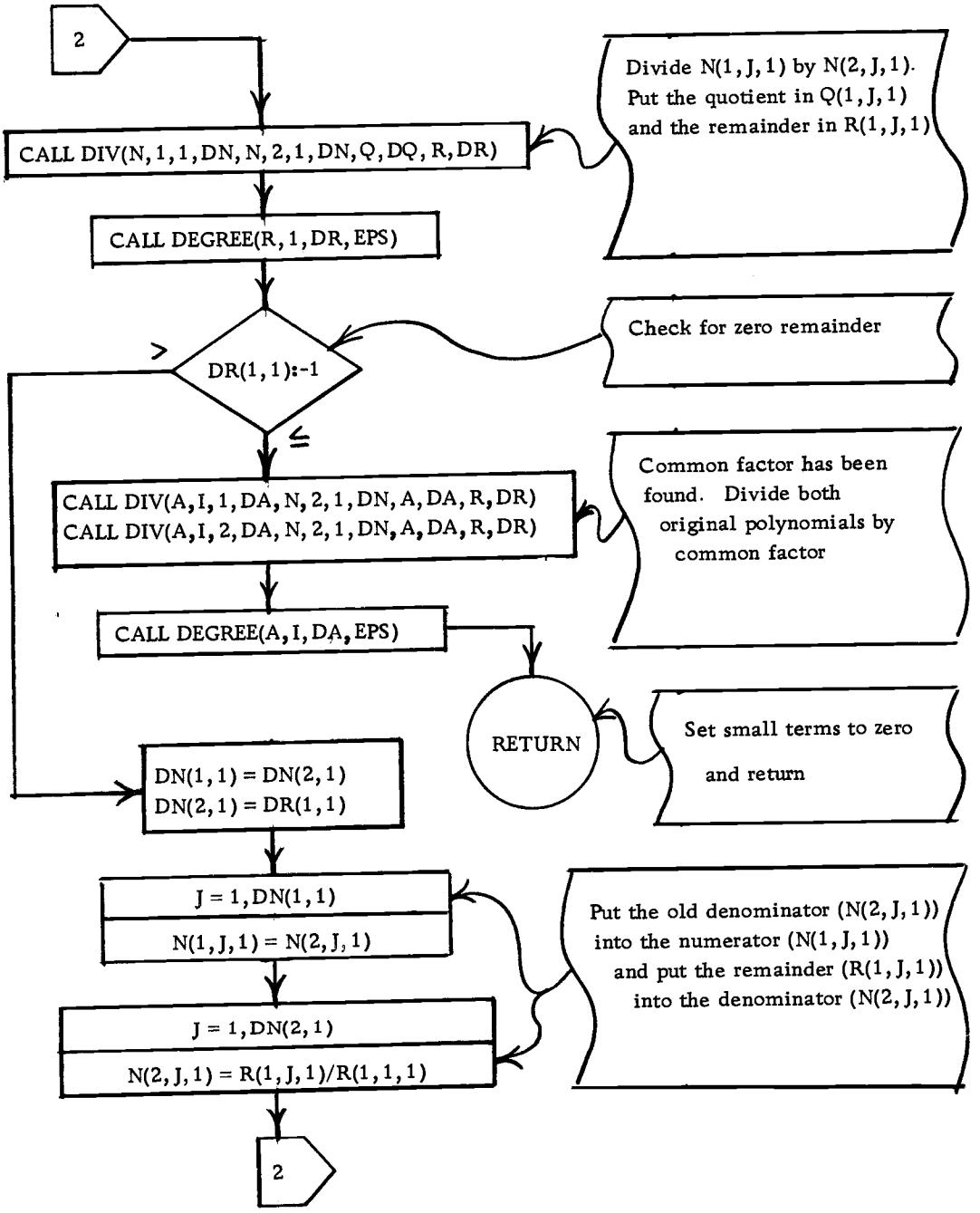

Figure 17. Continued.

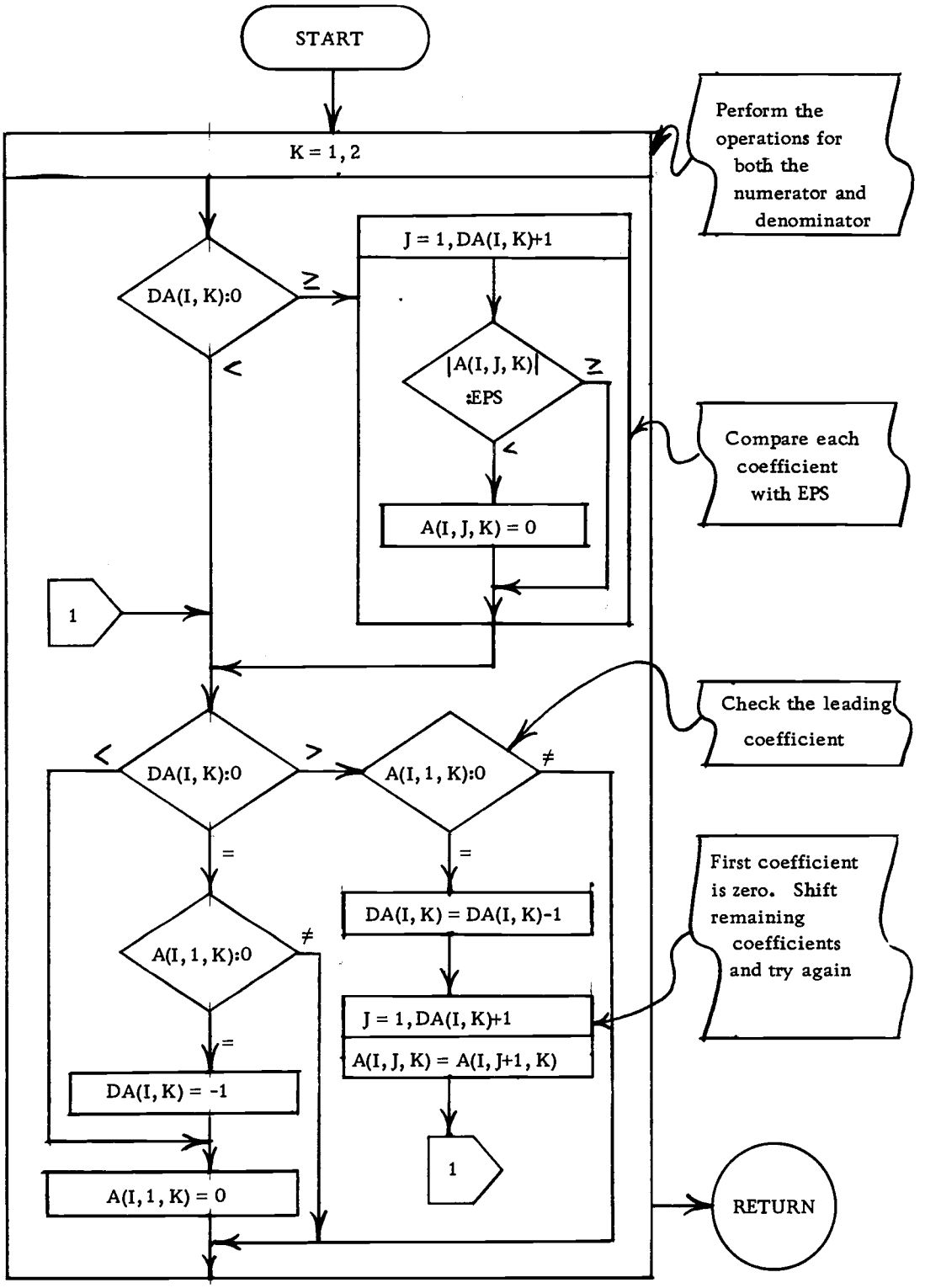

Figure 18. Detailed flow chart--SUBROUTINE DEGREE.

remainder respectively. For each of these arrays there corresponds an integer array (DN, DD, DQ, DR) which contains the degree of the polynomials in N, D, Q and R. I and II refer to the Ith and IIth polynomial fraction in arrays N and D respectively while K and KK refer to the numerator (K = 1 or KK = 1) or to the denominator (K = 2 or KK = 2) of the polynomial fractions. In summary, DIV takes the polynomial specified by (N, I, K), divides it by the polynomial specified by (D, II, KK), puts the quotient into the polynomial (Q, I, K) and the remainder into polynomial (R, I, K). Figure 19 is a flow chart for this process.

SUBROUTINE MULLER is not discussed in this thesis since it is virtually identical to the subroutine of the same name developed by Noonchester. Only one change was made in Noonchester's program: the introduction of a complex value for Z4 at the 20th iteration. See the third statement after the statement labeled 65 in the FORTRAN listing of SUBROUTINE MULLER for details.

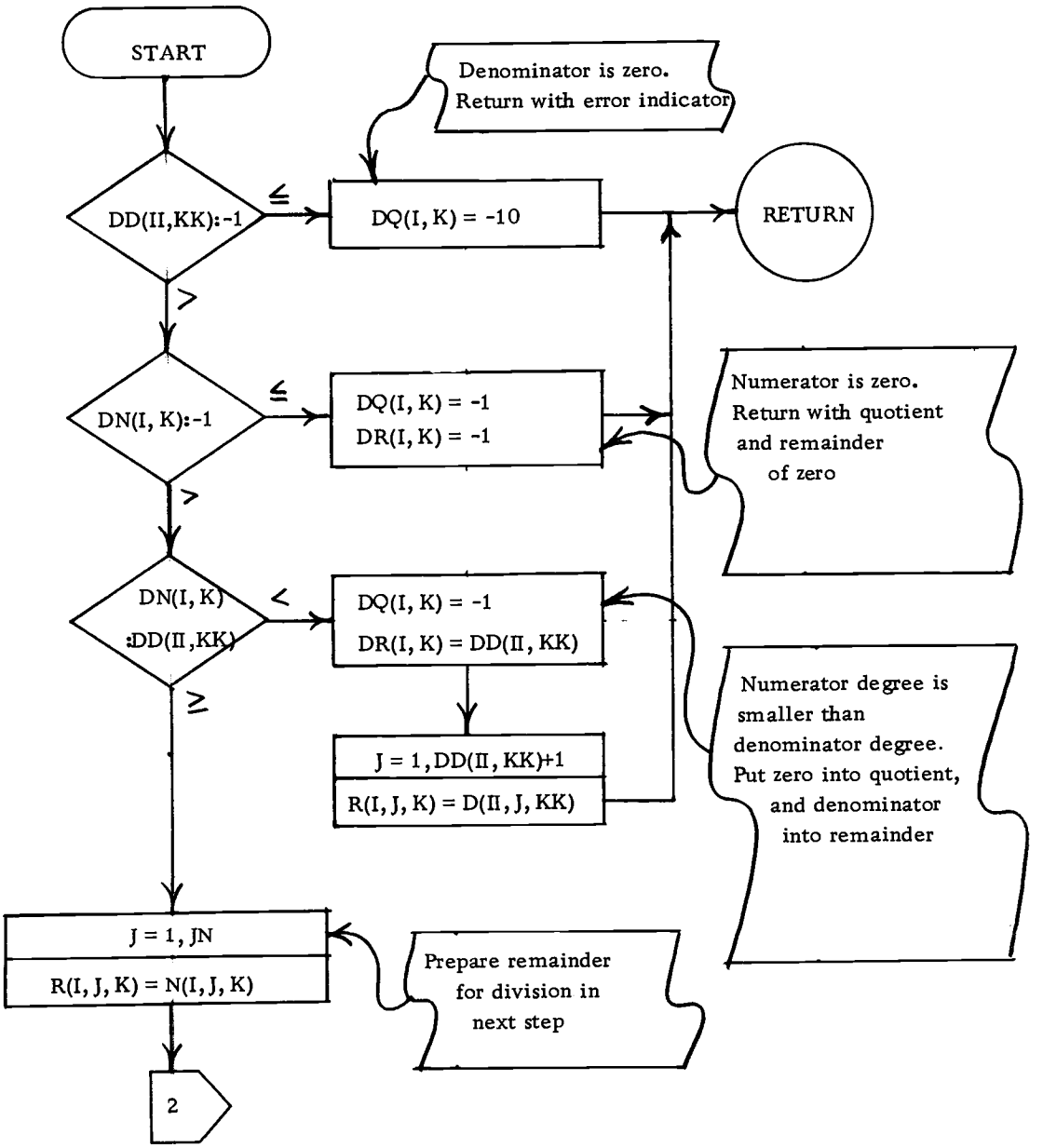

Figure 19. Detailed flow chart--SUBROUTINE DIV.

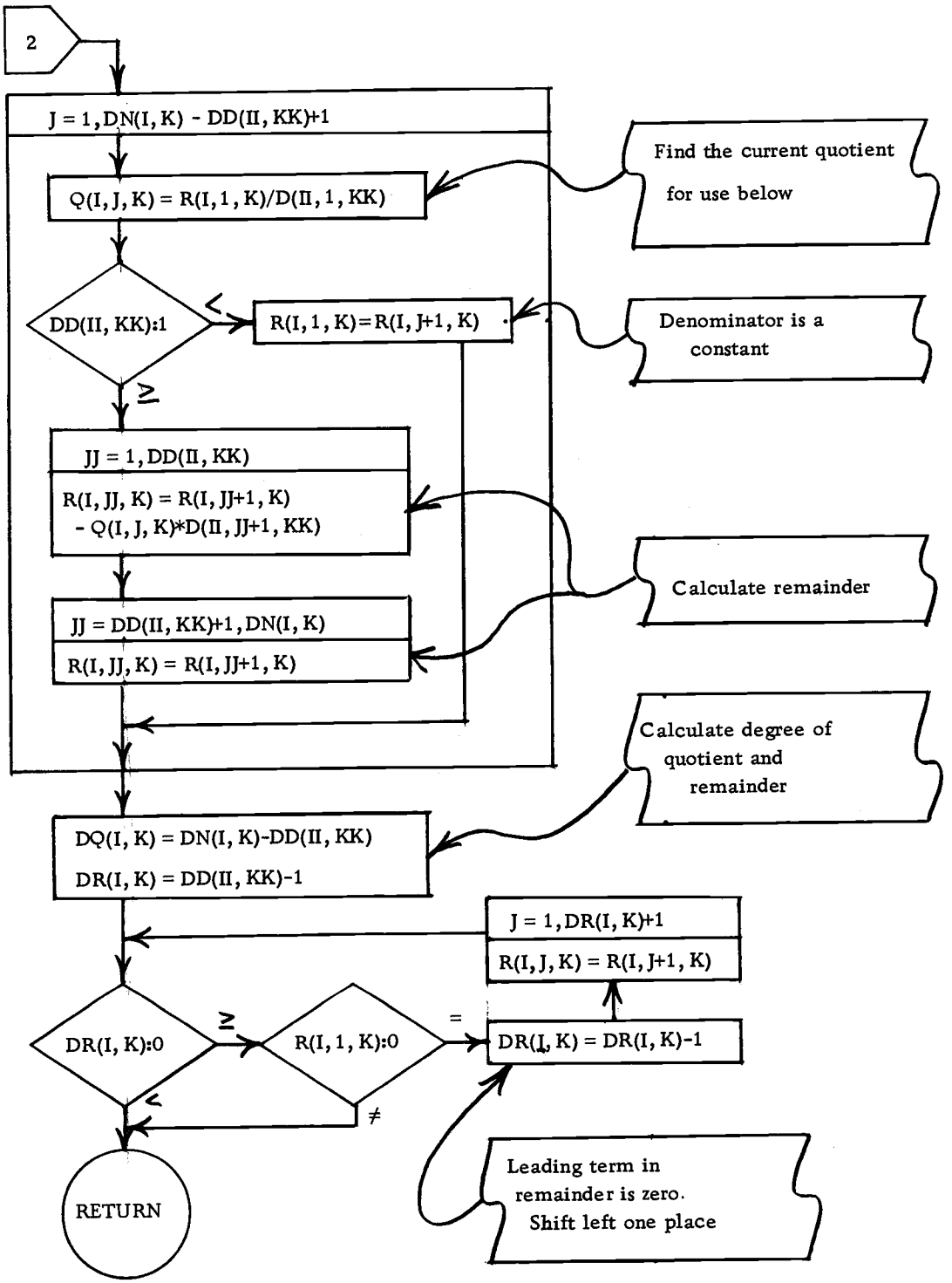

Figure 19. Continued.

V. COMMENTS ON THE PROGRAM

This chapter contains a discussion of the difficulties encountered in the development of three of the subroutines: SUM, COMFAC and DIV. Part of the difficulties arise because of the complexities in seemingly straight-forward algorithms. Other difficulties arise because of certain technicalities within these programs, such as the use of three dimensional arrays to store polynomials plus separate arrays to store the degrees of the polynomials. It is the former difficulties which are described in this chapter.

The sum or difference of two polynomials is a fairly trivial operation for the human computer to perform. However, to completely automate a process to perform the same operation, many decisions must be made. The first decision point must include a check to see if both polynomials are zero. If they are, then the process can store a zero in each coefficient of the "answer" polynomial and stop. If both polynomials are not zero then the limits of summation must be calculated (i.e., which coefficients of the second polynomial are to be added to or subtracted from which coefficients of the first polynomial). This calculation must be based on the relative degrees of the polynomials. If the first polynomial has a degree larger than the degree of the second polynomial, then one programming path is taken. If the degrees are equal, a second path is taken and if

the degree of the first polynomial has a degree smaller than the degree of the second polynomial then a third path must be taken. Once it has been established which coefficients are to take part in the actual operation wanted, then it becomes fairly straight-forward to perform the operation. As seen by the above, several decisions must be made before any computations can begin in order to add two polynomials. This decision making "overhead" is further illustrated in the following discussion about SUBROUTINE COMFAC.

COMFAC is a subroutine designed to find any common factor(s) shared by two polynomials and then to divide each of the polynomials by these common factor(s). However, before COMFAC can really begin to use Euclid's algorithm to find the common factors, several decisions must be made. One of the first decisions is to check to see if either polynomial is zero. If so, the procedure must stop, if not, continue. The next check is one which determines which polynomial has the largest degree. This polynomial, i.e., its coefficients, must be stored in one place, say in an array N, while the other polynomial, i.e., its coefficients, must be stored in another place, say, in array D. Now Euclid's algorithm can be applied to the two stored polynomials. The polynomial in N is divided by the polynomial in D; the quotient and remainder polynomials are put into other storage areas, say Q and R respectively. Next, a check is made on the polynomial in R. If it is zero then a common factor

has been found (which is in N). The algorithm stops after dividing both polynomials by their common factor. If the polynomial in R is not zero, then the polynomial in D is put into N and the polynomial in R is put into D. The program now loops back to where the polynomial in N is divided by the polynomial in D, and continues. Thus, sooner or later a remainder of zero must occur to stop the process.

The third algorithm to be discussed is SUBROUTINE DIV, which, again, requires several decisions to be made before the actual division portion of the program can be performed. First a check must be made to see if the polynomial in the denominator (D) is zero. If it is, an error has occurred and the algorithm must stop. If not, a check is made to see if the polynomial in the numerator (N) is zero. If it is, then the polynomials in the quotient (Q) and remainder (R) are set to zero and the algorithm stops. Otherwise a check is made to see if the degree of the numerator is smaller than the degree of the denominator. If this is true, then the quotient is assigned the value zero and the contents of R are assigned the values of the contents of D and the algorithm stops.

If none of the checks which stop the algorithm have been satisfied, then the division process can begin. However, more decisions must still be made during the course of the program. For example, the number and location of the coefficients to be assigned to the

quotient polynomial must be determined, and similarly for the remainder polynomial; careful attention must be paid to program the exact limits of the arrays used in the division process. Moreover, after the division is completed, a check must be made to see if the remainder is zero. If it isn°t, then a check is made on the leading coefficient of the remainder. If it is zero, then the degree of the remainder must be reduced by one and the remainder must again be checked, for zero, and so on until the format of the results agree with the format one usually expects of a polynomial. In any case, the amount of detail involved in writing some programs, such as DIV, is considerably greater than the amount one would expect of such a "straightforward" procedure. The amount of detail seems to be the result of having to completely specify a unique and valid action to be taken for any values of the input variables.

BIBLIOGRAPHY

1. Holyroyd, John Ries. Algorithms for the solution of two algebraic equations in two unknowns. Master's thesis. Corvallis, Oregon State University, 1963. 47 p.

2. Noonchester, Howard Basil. Study of effective algorithms for solving polynomial algebraic equations in one unknown. Master's thesis. Corvallis, Oregon State University, 1969. 119 p.

3. Sylvester, James Joseph. The collected mathematical papers of James Joseph Sylvester. Cambridge, University Press, 1904. 650 p.

APPENDIX

```
      PROGRAM SYLVESTER
C     10 OCT 70
C     INPUT DATA DATA (EPS) (NA(DAA(I) (AA(I,J), J=1,DAA(I)+1),I=1,NA+1)
      REAL AA(10,10), AB(10,10), AT(16)
      REAL R(10), P(10,10,2)
      INTEGER DAA(10), DAB(10)
      INTEGER DP(10,2)
      READ (60,102) EPS
      WRITE(61,102) EPS
      READ (60,100) NA
      WRITE(61,100) NA
      NX=NA+1
      DO 10 I=1,NX
      READ (60,100) DAA(I)
      WRITE(61,100) DAA(I)
      NY=DAA(I)+1
      READ (60,101) (AA(I,J), J=1,NY)
      WRITE(61,101) (AA(I,J), J=1,NY)
10    CONTINUE
      NX=NX+1
      IF(NX.GT.10) GO TO 18
      DO 15 I=NX,10
      DAA(I)=-1
15    AA(I,1)=0.
18    CONTINUE
      READ (60,100) NB
      WRITE(61,100) NB
      NX=NB+1
      DO 20 I=1,NX
      READ (60,100) DAB(I)
      WRITE(61,100) DAB(I)
      NY=DAB(I)+1
      READ (60,101) (AB(I,J), J=1,NY)
      WRITE(61,101) (AB(I,J), J=1,NY)
20    CONTINUE
      NX=NB+2
      IF(NX.GT.10) GO TO 30
      DO 25 I=NX,10
      DAB(I)=-1
25    AB(I,1)=0.
30    CONTINUE
      CALL SYLVES(AA,NA,DAA,AB,NB,DAB,AT,NT,EPS)
      IF(NT.LT.0) GO TO 9999
      IN=NT+1
      WRITE(61,100) NT
      WRITE(61,101) (AT(I), I=1,IN)
      CALL SYL2(AA,NA,DAA,AB,NB,DAB,AT,NT,EPS)
9999  STOP
100   FORMAT(I10)
101   FORMAT(10F13.6)
102   FORMAT(F13.6)
      END
```

```
      SUBROUTINE SYLVES(AA,NA,DAA,AB,NB,DAB,AT,NT,EPS)
      INTEGER DAA(10), DAB(10), DA(10,2) , DB(10,2)
      INTEGER DR(10,2), DC(10,2)
      INTEGER P, PP
      REAL AA(10,10), AB(10,10), AT(10), A(10,10,2)
      REAL B(10,10,2), R(10,10,2)
      REAL  C(10,10,2)
C     ARRAYS AA AND AB CONTAIN THE COEFFICIENTS OF THE TWO EQUATIONS
C     AA(I,J), (I=1,NA+1, J=1,DA(I)+1) IS THE ITH COEFFICIENT IN X
C        AND THE JTH COEFFICIENT IN Y
C     THE INTEGER ARRAYS DAA AND DAB CONTAIN THE DEGREE OF THE
C     ITH COEFFICIENT OF X FOR AA AND AB RESPECTIVELY
C     ARRAY C IS USED ONLY FOR STORAGE OF INTERMEDIATE RESULTS
C     AT IS THE OUTPUT POLYNOMIAL IN Y WITH DEGREE NT
C     EPS IS THE SMALLEST ALLOWED COEFFICIENT GREATER THAN ZERO
C
C     DETERMINE IF INPUT DATA IS SATISFACTORY
1     M=NA+1
      P=NB+1
      IF(M.GE.1.AND.P.GE.1) GO TO 10
      NT=-101
      RETURN
C     CONSTRUCT WORKING MATRICES A AND B FROM AA AND AB
10    DO 20 I=1,10
      JA=DAA(I)
      DA(I,1)=JA
      DA(I,2)=0
      A(I,1,1)=0.
      A(I,1,2)=1.
      IF(JA.LT.0) GO TO 20
      JA=JA+1
      DO 15 J=1,JA
15    A(I,J,1)=AA(I,J)
20    CONTINUE
      DO 30 I=1,10
      JB=DAB(I)
      DB(I,1)=JB
      DB(I,2)=0
      B(I,1,1)=0.
      B(I,1,2)=1.
      IF(JB.LT.0) GO TO 30
      JB=JB+1
      DO 25 J=1,JB
25    B(I,J,1)=AB(I,J)
30    CONTINUE
C     INITIALIZE ARRAYS R AND C
      DO 35 I=1,10
      DR(I,1)=-1
      DC(I,1)=-1
      R(I,1,1)=0.
      C(I,1,1)=0.
      DR(I,2)=0
      DC(I,2)=0
      R(I,1,2)=1.
      C(I,1,2)=1.
35    CONTINUE
      DR(1,1)=0
      R(1,1,1)=1.
C     BEGIN BRANCHING DECISIONS
40    CONTINUE
      CALL WRIMAT(A,DA,1)
```

```
      CALL WRIMAT(B,DB,2)
      CALL WRIMAT(R,DR,3)
      IF(M.EQ.1 .OR. P.EQ.1) GO TO 900
      IF(M.GT.P) GO TO 50
      IF(DA(1,1)) 46,44,100
44    IF(ABS(A(1,1,1)).GE.EPS) GO TO 100
46    DA(1,1)=-1
      IF(DB(1,1)) 500,48,200
48    IF(ABS(B(1,1,1)).GE.EPS) GO TO 200
      DB(1,1)=-1
      GO TO 500
50    IF(DB(P,1)) 54,52,300
52    IF(ABS(B(P,1,1)).GE.EPS) GO TO 300
      DB(P,1)=-1
54    IF(DA(M,1)) 500,56,400
56    IF(ABS(A(M,1,1)).GE.EPS) GO TO 400
      DA(M,1)=-1
      GO TO 500
C     THE FOLLOWING STATEMENTS EFFECT AN EXPANSION BY A(1,J,K)
100   DO 110 I=2,M
      CALL PMULT(A,I,DA,R,1,DB,C,DC,EPS)
      CALL WRIMAT(C,DC,4)
      CALL POIV  (C,I,DC,A,1,DA,C,DC,EPS)
      CALL WRIMAT(C,DC,5)
      CALL PSUB(B,I,DB,C,I,DC,B,DB,EPS)
      CALL WRIMAT(B,DB,6)
110   CONTINUE
140   CALL PMULT(R,1,DR,A,1,DA,R,DR,EPS)
      P=P-1
      CALL PSHIFT(B,DB)
      GO TO 40
C     THE FOLLOWING STATEMENTS EFFECT AN EXPANSION BY B(1,J,K)
200   CONTINUE
      CALL PMULT(B,1,DB,R,1,DR,R,DR,EPS)
      CALL PSHIFT(B,DB)
      M=M-1
      GO TO 40
C     THE FOLLOWING STATEMENTS EFFECT AN EXPANSION BY B(P,J,K)
300   CONTINUE
      IL=M-P+1
      MM1=M-1
      DO 310 I=IL,MM1
      IB=I-M+P
      CALL PMULT(B,IB,DB,A,M,DA,C,DC,EPS)
      CALL WRIMAT(C,DC,14)
      CALL POIV(C,IB,DC,B,P,DB,C,DC,EPS)
      CALL WRIMAT(C,DC,15)
      CALL PSUB(A,I,DA,C,IB,DC,A,DA,EPS)
      CALL WRIMAT(A,DA,16)
310   CONTINUE
      DA(M,1)=-1
      A(M,1,1)=0.
      M=M-1
      CALL PMULT(R,1,DR,B,P,DB,R,DR,EPS)
      GO TO 40
C     THE FOLLOWING STATEMENTS EFFECT AN EXPANSION BY A(M,J,K)
400   CALL PMULT (R,1,DR,A,M,DA,R,DR,EPS)
      P=P-1
      GO TO 40
500   NT=-100
      RETURN
C     THE FOLLOWING STATEMENTS CONCLUDE THE WORK OF SYLVES
C        AND EFFECT A RETURN TO THE CALLING PROGRAM
900   CONTINUE
      IF(P.EQ.1) GO TO 940
      P=P-1
      DO 930 I=1,P
930   CALL PMULT(R,1,DR,A,1,DA,R,DR,EPS)
      GO TO 960
940   CONTINUE
      M=M-1
      DO 950 I=1,M
950   CALL PMULT(R,1,DR,B,1,DB,R,DR,EPS)
C     TRANSFER R INTO AT
960   CONTINUE
      CALL WRIMAT(R,DR,90)
      NT=DR(1,1)
      JT=NT+1
      DO 980 J=1,JT
980   AT(J)=R(1,J,1)
      RETURN
      END
```

```
      SUBROUTINE SYL2(AA, NA, DAA, AB, NB, DAB, AT, NT, EPS)
C     10 FEB 71
      INTEGER DAA(10), DAB(10), P
      REAL AA(10,10), AB(10,10), AT(10), NI(10), RY(20), R1(20),
     1 R2(20), RD(20), NORM, RC1(20), RC2(20)
      TYPE COMPLEX(4) CAT(10), RTY(10), REM(10), T1, CAX(10,10,2),
     1 CTX1(10), CTX2(10), COMP, DEL, CRT1(10), CRT2(10)
      EQUIVALENCE (RY,RTY), (R1,CRT1), (R2,CRT2), (RD,DEL)
      EQUIVALENCE (RC1,CTX1), (RC2,CTX2)
C     TRANSFER ARRAY AT INTO COMPLEX ARRAY CAT
      NP1=NT+1
      DO 1000 I=1,NP1
 1000 CAT(I)=COMP(AT(I),0.)
C     FIND ROOTS OF SINGLE EQUATION IN Y
      CALL MULLER(CAT, NT, 0, NI, NR, RTY, REM)
C     NR IS THE NUMBER OF ROOTS FOUND BY MULLER
C     CHECK IF ALL ROOTS FOUND
      WRITE(61,2000) NR, NT
C     PRINT ROOTS FOUND FOR Y
      NR1=2*NR-1
      DO 1005 I=1,NR1,2
 1005 WRITE(61,2005) RY(I), RY(I+1)
 2005 FORMAT(/,4E20.10,///)
 2000 FORMAT(///,I3,2H /,I3,18H ROOTS FOR Y FOUND)
C     BEGIN EVALUATION OF NR POLYNOMIALS
 1010 M=NA+1
      P=NB+1
      DO 1070 I=1,NR
C     EVALUATE A SINGLE POLYNOMIAL FOR A COEFFICIENT OF E1
      DO 1040 J=1,M
      JU=DAA(J)+1
      T1=COMP(AA(J,1),0.)
      IF(JU .LE. 1) GO TO 1030
      DO 1020 JA=2,JU
 1020 T1=T1*RTY(I)+COMP(AA(J,JA),0.)
 1030 CAX(I,J,1)=T1
 1040 CONTINUE
C     EVALUATE A SINGLE POLYNOMIAL FOR A COEFFICIENT OF E2
      DO 1070 J=1,P
      JU=DAB(J)+1
      T1=COMP(AB(J,1),0.)
      IF(JU .LE. 1) GO TO 1060
      DO 1050 JA=2,JU
 1050 T1=T1*RTY(I)+COMP(AB(J,JA),0.)
 1060 CAX(I,J,2)=T1
 1070 CONTINUE
C     THE COMPLEX ARRAY CAX(I,J,K) CONTAINS NR POLYNOMIALS FOR EACH
C     OF E1 AND E2
C     I IS THE POLYNOMIAL NUMBER (CORRESPONDING TO THE ITH POLYNOMIAL)
C     J IS THE JTH COEFFICIENT OF THE ITH POLYNOMIAL
C     K IS 1 FOR E1 OR 2 FOR E2
C     FOR THE ITH ROOT OF Y, FIND THE ROOTS OF E1 AND E2, AND COMPARE
C     IF THE ROOTS ARE EQUAL THEN A SOLUTION SET IS (RTY(I), CRT1(J))
      DO 1150 I=1,NR
C     MAKE COMPLEX TEMPORARY ARRAYS CTX1 AND CTX2 FROM CAX
      DO 1100 J=1,M
 1100 CTX1(J)=CAX(I,J,1)
      DO 1110 J=1,P
 1110 CTX2(J)=CAX(I,J,2)
C     PRINT ARRAYS CTX1 AND CTX2
      KJ1=2*M-1
      DO 1112 KJ=1,KJ1,2
 1112 WRITE(61,2112) RC1(KJ), RC1(KJ+1)
      KJ2=2*P-1
      DO 1114 KJ=1,KJ2,2
 1114 WRITE(61,2114) RC2(KJ), RC2(KJ+1)
 2112 FORMAT(/,5H E1  ,2E15.6,/)
 2114 FORMAT(/,5H E2  ,2E15.6,/)
C     FIND THE ROOTS OF E1 AND E2
      CALL MULLER(CTX1,  NA,    1, NI, NR1, CRT1, REM)
      CALL MULLER(CTX2, NB,     1, NI, NR2, CRT2, REM)
C     THE ROOTS OF E1 AND E2 ARE IN CRT1 AND CRT2
C     CHECK IF ALL ROOTS FOUND
      WRITE(61,2110) NR1, NA
C     PRINT ROOTS FOUND FOR E1
      NRR1=2*NR1-1
      IF(NRR1 .LE. 0) GO TO 1117
      DO 1115 IR=1,NRR1,2
 1115 WRITE(61,2005) R1(IR), R1(IR+1)
 2110 FORMAT(///,I3,2H /,I3,18H ROOTS OF E1 FOUND,///)
 1117 WRITE(61,2120) NR2, NB
C     PRINT ROOTS FOUND FOR E2
      NRR1=2*NR2-1
      IF(NRR1 .LE. 0) GO TO 1130
      DO 1120 IR=1,NRR1,2
 1120 WRITE(61,2005) R2(IR), R2(IR+1)
 2120 FORMAT(///,I3,2H /,I3,18H ROOTS OF E2 FOUND,///)
C     CHECK ROOTS FOR MATCH
 1130 DO 1150 JI1=1,NR1
      DO 1150 JI2=1,NR2
      DEL=CRT1(JI1)-CRT2(JI2)
      I1=2*I-1
      I2=2*JI1-1
      I3=2*JI2-1
      IF(NORM(DEL) .LE. EPS) GO TO 1140
      WRITE(61,2130) RY(I1), RY(I1+1), R1(I2), R1(I2+1), R2(I3),
     1 R2(I3+1), RD(1), RD(2)
 2130 FORMAT(/, 9H NO MATCH,8E15.6,/)
      GO TO 1150
 1140 WRITE(61,2140) RY(I1), RY(I1+1), R1(I2), R1(I2+1), RD(1), RD(2)
 2140 FORMAT(//,9H    MATCH,6E15.6,/)
 1150 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PMULT(A,I,DA,B,II,DB,P,DP,EPS)
C     10 OCT 70
C     FORMS PRODUCT OF ELEMENTS IN A(I) AND B(I)
C     P=A1B1/A2B2
      INTEGER DA(10,2), DB(10,2), DP(10,2)
      REAL A(10,10,2), B(10,10,2), P(10,10,2)
      CALL MULT(A,I,1,DA,B,II,1,DB,P,1,DP)
      CALL MULT(A,I,2,DA,B,II,2,DB,P,2,DP)
      CALL COMFAC(P,I,DP,EPS)
      RETURN
      END
```

```
      SUBROUTINE PDIV(N,I,DN,D,II,DD,Q,DQ,EPS)
C     10 OCT 70
C     FORMS DIVISION Q=N(I)/D(II)
      INTEGER DN(10,2), DD(10,2), DQ(10,2)
      REAL N(10,10,2), D(10,10,2), Q(10,10,2)
      CALL MULT(N,I,1,DN,D,II,2,DD,Q,1,DQ)
      CALL MULT(N,I,2,DN,D,II,1,DD,Q,2,DQ)
      CALL COMFAC(Q,I,DQ,EPS)
      RETURN
      END
```

```
      SUBROUTINE PSUB(A,I,DA,B,II,DB,S,DS,EPS)
C     11 DEC 70
C     FORMS DIFFERENCE S=A1/A2-B1/B2=(A1B2-A2B1)/A2B2
      INTEGER DA(10,2), DB(10,2), DS(10,2)
      INTEGER DE(10,2)
      REAL A(10,10,2), B(10,10,2), S(10,10,2)
      REAL E(10,10,2)
      CALL MULT(A,I,1,DA,B,II,2,DB,E,1,DE)
      CALL MULT(A,I,2,DA,B,II,1,DB,E,2,DE)
      CALL SUM(E,I,1,DE,E,I,2,DE,S,DS,-1)
      CALL MULT(A,I,2,DA,B,II,2,DB,S,2,DS)
      CALL COMFAC(S,I,DS,EPS)
      RETURN
      END
```

```
      SUBROUTINE PSHIFT(A,DA)
C     9 DEC 70
C     THIS SUBROUTINE SHIFTS ALL ELEMENTS IN A ONE PLACE
C     A(I,J,K)=A(I+1,J,K)
      REAL A(10,10,2)
      INTEGER DA(10,2)
      DO 30 K=1,2
      DO 20 I=1,9
      JU=DA(I+1,K)+1
      DA(I,K)=JU-1
      DO 20 J=1,JU
20    A(I,J,K)=A(I+1,J,K)
C     ENTER ZERO FOR TENTH ELEMENT
      DA(10,K)=-1
30    A(10,1,K)=0.
      RETURN
      END
```

```
      SUBROUTINE WRIMAT(A,DA,KN)
      INTEGER DA(10,2)
      REAL A(10,10,2)
      WRITE(61,101) KN
      DO 10 I=1,10
      DO 10 K=1,2
      J1=DA(I,K)+1
      IF(J1.GT.0) GO TO 5
      K=2
      GO TO 10
5     WRITE(61,100) I,K,(A(I,J,K), J=1,J1)
10    CONTINUE
      WRITE(61,102)
      RETURN
100   FORMAT(3H A(,I2,3H,J,I2,3H)= ,10E11.4)
101   FORMAT(7H WRIMAT,I4)
102   FORMAT(///)
      END
```

```
      SUBROUTINE MULT(A,I,K,DA,B,II,KK,DB,P,KKK,DP)
C     9 DEC 70
C     ARRAY E IS FOR INTERMEDIATE STORAGE
      INTEGER DA(10,2), DB(10,2), DP(10,2)
      REAL A(10,10,2), B(10,10,2), P(10,10,2), E(10,10,2)
      JA=DA(I,K)+1
      JB=DB(II,KK)+1
      IF(JA.GT.0 .AND. JB.GT.0) GO TO 10
      DP(I,KKK)=-1
      RETURN
10    JU=JA+JB-1
      DO 20 J=1,JU
      E(I,J,KKK)=0.
      DO 20 L=1,J
      LJ=J-L+1
      IF(L.GT.JA .OR. LJ.GT.JB) GO TO 20
      E(I,J,KKK)=E(I,J,KKK)+A(I,L,K)*B(II,LJ,KK)
20    CONTINUE
      DP(I,KKK)=JU-1
C     TRANSFER E INTO P
      DO 30 J=1,JU
30    P(I,J,KKK)=E(I,J,KKK)
      RETURN
      END
```

```
      SUBROUTINE SUM(A,I,K,DA,B,II,KK,DB,S,DS,IOP)
C     10 DEC 70
C     IOP GE 0  FOR ADDITION ELSE SUBTRACTION
      INTEGER DA(10,2),DB(10,2), DS(10,2)
      REAL A(10,10,2), B(10,10,2), S(10,10,2)
      REAL E(10,10,2)
      N=DA(I,K)
      M=DB(II,KK)
      IF(N.GE.0.OR.M.GE.0) GO TO 10
      DS(I,K)=-1
      RETURN
      S(I,1,K)=0.
10    IF(N-M)  20,40,30
20    JU=M-N
      JA=0
      JB=JU
      KS=JU
      DO 28 J=1,JU
      IF(IOP) 22,24,24
22    E(I,J,K)=-B(II,J,KK)
      GO TO 28
24    E(I,J,K)=B(II,J,KK)
28    CONTINUE
      ME=M+1
      JU=N+1
      GO TO 50
30    JU=N-M
      JA=JU
      JB=0
      KS=JU
      DO 38 J=1,JU
38    E(I,J,K)=A(I,J,K)
      GO TO 45
40    JA=0
      JB=0
      KS=0
45    JU=M+1
      ME=N+1
50    DO 58 J=1,JU
      KA=J+JA
      KB=J+JB
      JS=J+KS
      IF(IOP) 54,56,56
54    E(I,JS,K)=A(I,KA,K)-B(II,KB,KK)
      GO TO 58
56    E(I,JS,K)=A(I,KA,K)+B(II,KB,KK)
58    CONTINUE
C     TRANSFER E TO S
      DS(I,K)=ME-1
      DO 70 JS=1,ME
70    S(I,JS,K)=E(I,JS,K)
      RETURN
      END
```

```
      SUBROUTINE COMFAC(A,I,DA,EPS)
C     15 DEC 70
      INTEGER DA(10,2),DN(10,2),DQ(10,2),DR(10,2)
      REAL A(10,10,2),N(10,1),2),Q(10,10,2),R(10,10,2)
10    CALL DEGREE(A,I,DA,EPS)
      IF(DA(I,1).LT.DA(I,2)) GO TO 20
      KN=1
      KD=2
      GO TO 30
20    KN=2
      KD=1
30    JU=DA(I,KN)+1
      IF(JU) 55,55,40
40    DO 45 J=1,JU
45    N(1,J,1)=A(I,J,KN)
      DN(1,1)=JU-1
      JU=DA(I,KD)+1
      IF(JU) 50,50,60
50    DA(I,2)=-10
55    RETURN
60    DO 65 J=1,JU
65    N(2,J,1)=A(I,J,KD)
      DN(2,1)=JU-1
70    CALL DIV(N,1,1,DN,N,2,1,DN,Q,DQ,R,DR)
      CALL DEGREE(R,1,DR,EPS)
      IF(DR(1,1)) 90,100,100
90    CA LL DIV(A,I,1,DA,N,2,1,DN,A,DA,R,DR)
      CA LL DIV(A,I,2,DA,N,2,1,DN,A,DA,R,DR)
      CALL DEGREE(A,I,DA,EPS)
      RETURN
100   CONTINUE
      JU=DN(2,1)+1
      DN(1,1)=JU-1
      DO 105 J=1,JU
105   N(1,J,1)=N(2,J,1)
      JU=DR(1,1)+1
      DN(2,1)=JU-1
      DO 115 J=1,JU
115   N(2,J,1)=R(1,J,1)/R(1,1,1)
      GO TO 70
      END
```

```
      SUBROUTINE DEGREE(A,I,DA,EPS)
C     10 SEP 70
      INTEGER DA(10,2)
      REAL A(10,10,2)
      DO 100 K=1,2
      JU=DA(I,K)+1
      IF(JU) 30,30,10
10    DO 20 J=1,JU
      IF(ABS(A(I,J,K)).GE.EPS) GO TO 20
      A(I,J,K)=0.
20    CONTINUE
30    JU=DA(I,K)
      IF(JU) 90,70,40
40    IF(A(I,1,K)) 100,50,100
50    DA(I,K)=JU-1
      DO 60 J=1,JU
60    A(I,J,K)=A(I,J+1,K)
      GO TO 30
70    IF(A(I,1,K)) 100,80,100
80    DA(I,K)=-1
90    A(I,1,K)=0.
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE DIV(N,I,K,DN,D,II,KK,DD,Q,DR,R,DR)
C     12 AUG 70
      INTEGER DN(10,2),DD(10,2),DQ(10,2),DR(10,2)
      REAL N(10,10,2),D(10,10,2),Q(10,10,2),R(10,10,2)
      JN=DN(I,K)+1
      JD=DD(II,KK)+1
      IF(JD.GT.0) GO TO 10
      DQ(I,K)=-10
      RETURN
10    IF(JN.GT.0) GO TO 20
      DQ(I,K)=-1
      DR(I,K)=-1
      RETURN
20    IF(JN.GE.JD) GO TO 30
      DQ(I,K)=-1
      DR(I,K)=DD(II,KK)
      DO 25 J=1,JD
25    R(I,J,K)=D(II,J,KK)
      RETURN
30    DO 35 J=1,JN
35    R(I,J,K)=N(I,J,K)
      JU=JN-JD+1
      DO 40 J=1,JU
      Q(I,J,K)=R(I,1,K)/D(II,1,KK)
      IF(JD.GE.2) GO TO 38
      R(I,1,K)=R(I,J+1,K)
      GO TO 40
38    JUI=JD-1
      DO 39 JJ=1,JUI
39    R(I,JJ,K)=R(I,JJ+1,K)-Q(I,J,K)*D(II,JJ+1,KK)
      JUI=JN(I,K)
      DO 40 JJ=JD,JUI
      R(I,JJ,K)=R(I,JJ+1,K)
40    CONTINUE
      DQ(I,K)=JN-JD
      DR(I,K)=DD(II,KK)-1
45    IF(DR(I,K)) 70,50,50
50    IF(R(I,1,K)) 70,55,70
55    DR(I,K)=DR(I,K)-1
      JU=DR(I,K)+1
      DO 60 J=1,JU
60    R(I,J,K)=R(I,J+1,K)
      GO TO 45
70    RETURN
      END
```

```
      SUBROUTINE MULLER(A,N,KC,NI,NR,RTZ,REM)
C     1 FEB 71
      REAL NORM, IM
      TYPE COMPLEX(4) A, B, DEN, L3, L4, PZ1, PZ2, PZ3, PZ4,
     1 PRZ1, PRZ2, PRZ3, PRZ4, REM, RTZ, SRT, Z1, Z2, Z3, Z4,
     1 COMP, CONJ
      DIMENSION A(16), NI(15), REM(15), RTZ(15)
      NR=0
      NS=N
      I=N+1
5     IF( NORM(A(I)) .NE. 0.) GO TO 20
      NR=NR+1
      NI(NR)=0
      I=I-1
      RTZ(NR)=COMP(0.,0.)
      REM(NR)=COMP(0.,0.)
      IF(NR-N) 5, 1000, 5
20    IT=0
      L=1
      Z1=COMP(-1.,0.)
      Z2=COMP(1.,0.)
      Z3=COMP(.01,0.)
      IF(NR .EQ. 0) GO TO 27
      DO 25 I=1,NR
      IF(Z1 .EQ. RTZ(I)) Z1=.8*Z1
      IF(Z2 .EQ. RTZ(I)) Z2=.8*Z2
      IF(Z3 .EQ. RTZ(I)) Z3=.8*Z3
25    CONTINUE
27    CALL POLY(A,N,RTZ,NR,Z1,PZ1,PRZ1)
      CALL POLY(A,N,RTZ,NR,Z2,PZ2,PRZ2)
      CALL POLY(A,N,RTZ,NR,Z3,PZ3,PRZ3)
30    L3=(Z3-Z2)/(Z2-Z1)
      B=PRZ1*L3*L3 - PRZ2*(L3+1.)*(L3+1.) + PRZ3*(2.*L3+1.)
      SRT=CSQRT(B*B - 4.*PRZ3*L3*(L3+1.)*(PRZ1*L3-PRZ2*(L3+1.)+PRZ3))
      DEN=B+SRT
      IF(NORM(DEN) .LT. NORM(B-SRT)) DEN=B-SRT
      IF(DEN .EQ. 0.) GO TO 35
      L4=-2.*PRZ3*(L3+1.)/DEN
      GO TO 40
35    L4=COMP(1.,0.)
40    Z4=Z3+L4*(Z3-Z2)
      CALL POLY(A,N,RTZ,NR,Z4,PZ4,PRZ4)
      IF(N .EQ. 0) GO TO 900
      IF(NORM(PZ4/PZ3) .LT. 10.) GO TO 50
      L4=L4/2.
      GO TO 40
50    IF(NORM(PZ4) .LT. 1.0E-20) GO TO 100
      IF(Z3 .EQ. 0.) GO TO 65
      IF(NORM((Z4-Z3)/Z3) .LT. 1.0E-10) GO TO 100
      IF(IT .GE. 99) GO TO 100
65    IT=IT+1
      Z1=Z2
      Z2=Z3
      IF(IT .EQ. 20) Z4=Z4+COMP(0.,.01)
      Z3=Z4
      PRZ1=PRZ2
      PRZ2=PRZ3
      PRZ3=PRZ4
      GO TO 30
100   NR=NR+1
      NI(NR)=IT+1
```

```
      RTZ(NR)=Z4
      REM(NR)=PZ4
      IF(NR .GE. N) GO TO 1000
      IF(L .EQ. 2) GO TO 20
      IF(KC .NE. 0) GO TO 20
      IF(IM(RTZ(NR)) .EQ. 0.) GO TO 20
      IF(ABS(REAL(RTZ(NR))/IM(RTZ(NR))) .GT. 1.0E2 ) GO TO 20
      L=2
      IT=-1
      Z4=CONJ(Z4)
      CALL POLY(A,N,RTZ,NR,Z4,PZ4,PRZ4)
      IF(N .NE. 0) GO TO 100
900   N=NS
1000  RETURN
      END
```

```
      SUBROUTINE POLY(A, N, RTZ, NR, Z, PZ, PRZ)
      TYPE COMPLEX(4) A, PZ, RTZ, Z, PRZ
      DIMENSION A(16), RTZ(15)
C     1 FEB 71
      COMPLEX A, PZ, PRZ, RTZ, Z
      PZ=A(1)
      DO 20 I=1,N
20    PZ=Z*PZ + A(I+1)
      PRZ=PZ
      IF(NR .EQ. 0) GO TO 100
40    DO 50 I=1,NR
      IF(Z-RTZ(I) .EQ. 0.) GO TO 60
50    PRZ=PRZ/(Z-RTZ(I))
      GO TO 100
60    N=0
100   RETURN
      END
```