AN ABSTRACT OF THE THESIS OF

HOWARD BASIL NOONCHESTER for the MASTER OF SCIENCE
(Name) (Degree)

in MATHEMATICS presented on August 1966
(Major) (Date)

Title: STUDY OF EFFECTIVE ALGORITHMS FOR SOLVING POLYNOMIAL ALGEBRAIC EQUATIONS IN ONE UNKNOWN

Redacted for privacy

Abstract approved:
Harry E. Goheen

This paper makes available practical algorithms and their associated FORTRAN IV computer programs for finding the roots of polynomial equations.

The purpose of this paper is to examine effective algorithms for solving polynomial algebraic equations in one unknown on a digital computer. The advent of high-speed digital computing systems makes it practical to examine numerical methods which otherwise would be too time consuming if not impossible. Algorithms requiring only the polynomial coefficients are examined since they can be used as subprograms to solve polynomial equations which arise in other computer programs.

The above considerations have lead to the examination of the following algorithms:

(i). Lehmer's algorithm, (used to find rough approximations to

the roots).

a) The Newton-Raphson algorithm, (used to refine the root approximations).

(ii). Muller's algorithm.

(iii). Rutishauser's Quotient-Difference (QD) algorithm, (used to find rough approximations to the roots).

a) Newton-Raphson's algorithm, (used to refine approximations to simple roots).

b) Bairstow's algorithm, (used to refine approximations to two roots i. e. complex conjugates).

(iv). The Steepest Descent algorithm.

Study of Effective Algorithms for Solving Polynomial Algebraic Equations in One Unknown

by

Howard Basil Noonchester

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

June 1969

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 August 22, 1966

Typed by Clover Redfern for Howard Basil Noonchester

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

STUDY OF EFFECTIVE ALGORITHMS FOR SOLVING POLYNOMIAL ALGEBRAIC EQUATIONS IN ONE UNKNOWN

I. INTRODUCTION

In the study of the stability of airplanes it is necessary to solve linear differential equations with constant coefficients. The solution of such differential equations can be obtained by solving the associated characteristic equations i. e. polynomial equations, using the algorithms and computer programs discussed in this paper.

The eigenvalues of arbitrary complex matrices can be thought of as the roots of the determinant of the matrix $(A-\lambda I)$. The eigenvalues can be calculated by determining the coefficients of the associated characteristic polynomial and finding the roots of the resulting polynomial equation using the computer programs in this paper.

Of the many algorithms that have been developed for solving polynomial algebraic equations in one unknown, (hereafter referred to as polynomial equations) not all are general enough to be suitable for use as subprograms on a digital computer. The prime requirement for an algorithm is that it converge for all roots in a finite number of steps without special starting values. In terms of the language of the theory of algorithms, this requirement is that the computer program will always stop when presented with the description of the equation and tolerances on its roots and its output will be the

values of the roots to within these tolerances.

We examine six algorithms which are combined to form four computer programs. These computer programs and algorithms are:

i) Lehmer's algorithm (5, 9), (used to find rough approximations to the roots).

   a) The Newton-Raphson algorithm (3) (used to find closer approximations to the roots).

ii) Muller's algorithm (8).

iii) Rutishauser's Quotient-Difference (Q. D.) algorithm (2), (used to find rough approximations to the roots).

   a) The Newton-Raphson algorithm, (used to find closer approximations to simple roots).

   b) Bairstow's algorithm (3) (used to find closer approximations to two roots, i. e., complex conjugates).

vi) The Steepest Descent algorithm (7).

The algorithms are examined and special tests that are required by the computer program are noted. Next the computer programs for the associated algorithm or algorithms are described in the notation of Lyapunov's "Operator Programming" (see Appendix) to indicate the flow of the computer program logic.

In Chapter IV the results from solving nine polynomial equations from Milne's "Numerical Calculus" (6) are examined and compared. These polynomial equations are:

1) $z^3 - z - 4 = 0$ p. 38, no. 1.

2) $z^4 - 2.0379z^3 - 15.4245z^2 + 15.6696z + 35.4936 = 0$ p. 40, no. 4.

3) $z^4 - 2z^3 - 4z^2 - 4z + 4 = 0$ p. 41, no. 1.

4) $4z^4 - 24z^3 + 44z^2 - 24z + 3 = 0$ p. 41, no. 4.

5) $2z^4 + 16z^3 + z^2 - 74z + 56 = 0$ p. 42, no. 1.

6) $z^3 - 6.0266z^2 + 4.3048z + 15.9533 = 0$ p. 44, no. 1.

7) $z^4 + 12z^3 - 9.5z^2 - 6z + 4.5 = 0$ p. 44, no. 2.

8) $z^4 - 6z^3 - 113z^2 + 504z + 2436 = 0$ p. 44, no. 3.

9) $z^4 + 16z^3 + 11z^2 - 224z + 286 = 0$ p. 44, no. 4.

These particular polynomial equations were given as Exercises by Milne to illustrate difficulties in the Newton-Raphson method. They serve to test for weaknesses in the algorithms. Some of these polynomial equations have very close roots as the results in Chapter IV indicate.

The polynomial equations $x^n + x = 1$, $(n=3, 5, \ldots, 99, 101)$ are also solved and the results from the different methods examined and compared.

II. DESCRIPTION OF ALGORITHMS STUDIED

1. Lehmer's Algorithm (5, 9)

Lehmer's algorithm is also known as the Lehmer-Schur method since D. H. Lehmer uses a theorem of I. Schur to answer the question: "Does a given polynomial have a root inside a given circle?"(5).

Using the notation of (5, 9) we set up a procedure for location, in the complex plane, the roots of a polynomial $f(z)$. First the coefficients of auxiliary polynomials $T^i(f(z))$ for successive natural numbers i are computed, (5). By the method of construction the degree of $T^i(f(z))$ is decreasing. Let k be the smallest value of i for which $T^i(f(z))$ vanishes identically. For each auxiliary polynomial $T^i(f(z))$, the value at the origin is computed. If for every i in the interval $1 \leq i < k$, $T^i(f(0))$ is positive and $T^{k-1}(f(z))$ is a constant then there is no root of $f(z)$ inside the unit circle. The polynomial $f(z)$ is transformed by replacing z by $2z$ and the same process applied to the new unit circle. After recursive use of this replacing, one gets to an arbitrarily large circle. One knows that there is a root of $f(z)$ in an annulus $r < |z| < 2r$ but no root inside the circle $|z| = r$. On the other hand if for some i in the interval $1 \leq i < k$, $T^i(f(0))$ is negative, then there is a root of $f(z)$ inside the unit circle. An annulus can be determined in a fashion similar to the above, such that there is no root inside the circle

$|z| = r$ but there is a root in the annulus $r < |z| < 2r$.

The procedure tells us nothing in the case when $T^k(f(z))$ vanishes but $T^{k-1}(f(z))$ is not a constant. This case is handled in the algorithm by choosing a new radius. If the radius was being doubled then in place of $2r$ we use $1.5r$. If the radius was being halved then in place of $0.5r$ we use $0.75r$. The radius is repeatedly modified in this manner until the original procedure of locating a root can be applied. It is assumed by both Lehmer (5) and Ralston (9) that one may avoid this difficulty in a manner similar to the above, but the authors give no proof of this statement. The polynomial equation $f(z) = 6z^4 - 35z^3 + 62z^2 - 35z + 6$ due to Lehmer is a case in point. For this example $T(f(z))$ vanishes identically but $f(z)$ certainly is not a constant. The above procedure worked for this example as the computer output on the following page indicates.

The annulus can be completely covered by eight overlapping circles each of radius $.8r$ with centers, c_k at $3r\, e^{2\pi ik/8}/2\cos \pi/8$, $k=0, 1, \ldots, 7$; $i=\sqrt{-1}$, (Ref. 9). The polynomial $f(z)$ is transformed by replacing z by $.8z + c_k$ to get $f_k(z)$, $k=0, 1, \ldots, 7$. The above process of checking circles for roots is applied to successive $f_k(z)$ until a circle is found that contains a root. Then an annulus is determined such that there is no root inside the circle $|z| = r^*$ but there is a root in the annulus $r^* < |z| < 2r^*$ where r^* is a new radius determined using the preceeding

procedure.

Case 1, 5 Coefficients

```
 6.000000000000E+00  0.
-3.500000000000E+01  0.
 6.200000000000E+01  0.
-3.500000000000E+01  0.
 6.000000000000E+00  0.
```

4 Roots		Remainders		
3.333333333333E-01	0.	NI= 5	5.68434189E-14	0.
5.000000000000E-01	0.	NI= 5	-8.52651283E-14	0.
3.000000000000E+00	0.	NI= 0	-7.02016223E-12	0.
2.000000000000E+00	0.	NI= 0	-1.33582034E-12	0.

2 Roots Found by Lehmers Method.

3.044228062500E-01	0.	NI= 27
4.058970750000E-01	0.	NI= 18

Execution Time = .040 seconds
Total Execution Time = .044 seconds

It should be noted that part of each of these circles falls outside the annulus. This allows the possibility that the process will converge to a root of $f_k(z)$ outside the current annulus. We know that for some k there is a root of $f_k(z)$ which is inside the current annulus so in either case we will converge on a root of $f_k(z)$. Since any root of $f_k(z)$ corresponds to a root of $f(z)$, when the above process is applied recursively, the root of $f(z)$ is ultimately determined to within an arbitrarily fine tolerance.

Once a root is found to within the tolerance it is refined in the Newton-Raphson method (3). This is done for the sake of economy since Lehmer's method is only linearly convergent whereas Newton-

Raphson's method is quadratically convergent, (see Chapter V). The degree of $f(z)$ is now depressed by synthetic division. This leads to an accumulation of round-off error, but must be done to guarantee convergence by Lehmer's method and the Newton-Raphson method on a new root. If the depressed equation is only used in getting the root approximation by Lehmer's method, and then the approximation is refined in the Newton-Raphson method using the original polynomial we may not converge on a new root but on one previously found.

This is exactly what happened in Case 5, of the nine test cases, in Chapter IV when the original polynomial was used in a test run. Looking at this case on page 102 we see that the polynomial has two very close roots, 1.123105625615 and 1.121320343563. Lehmer's method found 1.623588300 as the approximation to the first root using the original equation and the same approximation for the second root using the reduced equation. Since the roots are so close together we would expect the root approximations to be the same for the approximations are circle centers and the pattern of obtaining them is the same. Since these root approximations are the same we see that Newton-Raphson's method will, (and did) converge on the same root when the original polynomial is used.

We could try to get better approximations from Lehmer's method and then use Newton-Raphson's method with the original polynomial. This will work up to a point even though it is expensive, computer

time wise, since Lehmer's method is linearly convergent. The point is that if the tolerance is too small the accuracy is unattainable by Lehmer's method due to round-off errors that result from the computer operations. This is discussed in detail starting on page 90 . To change the tolerance only when roots are close and avoid excessive iterations when the roots are well separated requires a fore knowledge of the root distribution which is not always available.

Although the roots found using Newton-Raphson's method and the reduced equation were correct (had remainders with magnitude of 10^{-13}) we were not able to maintain this accuracy, (see Chapter III, Table 3). It should be noted that the remainders were computed using the original polynomial equation $f(z)$.

The desirability of a faster method that does not have to resort to depressing the degree of $f(z)$ by synthetic division is apparent. Muller's method, which is discussed later in this paper, fulfills both of these requirements.

Operator Program

On the following pages are described the operator programs which go together to form Lehmer's program. Each of the operators is first defined and then the program is documented as a string of operators. A discussion of operator programming is given in the Appendix. Note that Lehmer's computer program includes Lehmer's

algorithm and the Newton-Raphson algorithm.

Lehmer's Program--Operator Programming Definitions:

Π_0 Input the program and input data into the memory of the computer.

Comment: NC = the degree of the polynomial equation plus one.

KC = 0 if the coefficients are all real.

A(I) = the complex coefficients of the polynomial equation (I = 1, . . . , NC).

A_1 Translate the input data into binary.

Comment: LRTS = the number of roots found by Lehmer's algorithm.

NR = the total number of roots found, i. e., complex conjugates are not iterated for by Lehmer's algorithm.

NRTS = the number of roots that the original polynomial equation has, (the degree).

N = the degree of the reduced polynomial equation. The degree of the polynomial equation is reduced as the roots are found.

β_2 LRTS = 0;

NR = 0;

NRTS = N;

A_3 N = NC - 1;

Comment: AR(I) = the complex coefficients of the reduced polynomial

equation. At this point the degree has not been reduced.

$\mathcal{Y}_4$ AR(I) = A(I), (I = 1, . . . , NC);

Comment: L is a flag which is equal to one if complex conjugates have not been formed and equal to two if they have been formed.

$\mathcal{Z}_5$ L = 1;

Comment: If the degree of the reduced polynomial equation is two, solve the equation using the quadratic formula.

P_6 N = 2? ;

Comment: If the degree of the reduced polynomial equation is one, divide to find the root. Note that the degree of the reduced polynomial can go from three to one when a root and its complex conjugate are found.

P_7 N = 1? ;

Comment: IT = the number of iterations performed to find a root.

E_8(LEHMER) CALL LEHMER (AR, N, IT, Z);

Comment: N is set to zero if Lehmer's algorithm can not find a root to within a specified tolerance. This is possible due to round-off errors made by the computer.

P_9 N ≠ 0? ;

Π_{10} Write: All the roots could not be found.

A_{11} NR = NR + 1;

Comment: NI(NR, 1) is an array in which are stored the number of iterations required by Lehmer's algorithm to find each root.

RTZ(NR, 1) is an array in which are stored the root approximations found by Lehmer's algorithm.

$\mathcal{J}_{12}$ NI(NR, 1) = IT;

RTZ(NR, 1) = Z;

Comment: Test to see if z = 0 is a root. If it is, decrease the debree of the reduced polynomial equation and store the number of iterations, the root and the remainder.

NI(NR, 2) is an array in which are stored the number of iterations required to refine the root.

RTZ(NR, 2) is an array in which are stored the refined root approximations.

REM(NR) is an array in which are stored the remainders from evaluating the polynomial f(z) at the refined root approximation.

P_{13} |Z| ≠ 0;

A_{14} N = N - 1;

$\mathcal{J}_{15}$ NI(NR, 2) = NI(NR, 1);

RTZ(NR, 2) = RTZ(NR, 1);

REM(NR) = 0;

Comment: If the reduced equation is linear then the number of roots

found is increased by one and the root is solved for and stored.

A_{16} NR = NR + 1

Z = -AR(2)/AR(1);

$\mathcal{B}_{17}$ NI(NR, 1) = 0;

RTZ(NR, 1) = Z;

IT = 0;

Comment: If the reduced equation is quadratic then it is solved by the quadratic formula in the form of a subprogram.

E_{18}(QUAD) CALL QUAD(AR(1), AR(2), AR(3), Z(1), Z(2));

Comment: Store the roots and compute and store the remainders.

E_{19}(20, 22) (I = 1, 2);

A_{20} NR = NR + 1;

$\mathcal{B}_{21}$ NI(NR, 1) = 1;

RTZ(NR, 1) = Z(I);

NI(NR, 2) = 0;

RTZ(NR, 2) = Z(I);

E_{22}(POLY) REM(NR) = POLY(A, NRTS, Z(I));

Comment: Use the Newton-Raphson method to refine the root approximations found by Lehmer's method.

E_{23}(NEWTON) CALL NEWTON(AR, N, IT, Z, REM(NR));

$\mathcal{B}_{24}$ NI(NR, 2) = IT;

RTZ(NR, 2) = Z;

Comment: Find the remainder using the coefficients of the original polynomial.

E_{25}(POLY) REM(NR) = POLY(A, NRTS, Z);

Comment: If all the roots have been found go and print out the results.

P_{26} NR $\geq$ NRTS?;

Comment: If the absolute value of the remainder just found is greater than or equal to one stop the procedure and print out the results found. Since the equation must be deflated using the root just found, the root must not have a large error.

P_{27} |REM(NR)| $\geq$ 1?;

Comment: Calculate the coefficients for the reduced equation.

A_{28} AR(I) = AR(I) + Z * AR(I-1), (I=2, . . . , N);

A_{29} N = N - 1;

Comment: Have the complex conjugates been found?

P_{30} L = 2?;

Comment: Are the coefficients all real?

P_{31} KC $\neq$ 0?;

Comment: Is the imaginary part of the root zero?

P_{32} Imaginary Z = 0?;

P_{33} $\left|\frac{\text{real Z}}{\text{imaginary Z}}\right| > 100?$;

Comment: Set L = 2 since we are computing the complex conjugate.

$З_{34}$ L = 2;

IT = 0;

A_{35} Z = $\overline{Z}$;

$П_{36}$ Output the coefficients, roots and remainders.

$Ж_{37}$ Stop the machine.

Combining the above operators, the logical scheme of the program has the form:

$$П_0 A_1 З_2 A_3 Ц_4 \overset{30}{\urcorner} З_5 \overset{15,31,32,33}{\urcorner\!\!\urcorner} P_6 \overset{18}{\lceil} P_7 \overset{16}{\lceil}$$

$$E_8 P_9 \overset{11}{\lceil} П_{10} \overset{36}{\lceil} ; \overset{9,35}{\urcorner} A_{11} З_{12} P_{13} \overset{23}{\lceil} A_{14} З_{15} \overset{6}{\lceil} ;$$

$$\overset{7}{\urcorner} A_{16} З_{17} \overset{24}{\lceil} ; \overset{6}{\urcorner} E_{18} E_{19} A_{20} З_{21} E_{22} \overset{36}{\lceil} ;$$

$$\overset{13}{\urcorner} E_{23} \overset{17}{\urcorner} З_{24} E_{25} P_{26} \overset{36}{\lceil} P_{27} \overset{36}{\lceil} A_{28} A_{29} P_{30} \overset{5}{\lceil} P_{31} \overset{6}{\lceil}$$

$$P_{32} \overset{6}{\lceil} P_{33} \overset{6}{\lceil} \; {}_{34} A_{35} \overset{11}{\lceil} ; \overset{10,22,26,27}{\urcorner\!\!\urcorner} П_{36} Ж_{37} .$$

E(LEHMER) SUBROUTINE LEHMER(A, N, IT, Z)

Comment: A(I) = the complex coefficients of the polynomial equation

(I=1, . . . , N+1).

N = the degree of the polynomial equation.

IT = the number of iterations to find the root.

Z = the root approximation.

IRT = 1 if there is a root inside the circle with radius RDS, otherwise IRT = 0.

IRDS = 0 when starting the iteration procedure.

IRDS = 1 if the radius is being halved.

IRDS = 2 if the radius is being doubled.

K = the number of the circle center being tested.

ITIME = 1 if this is the initial circle being tested, otherwise ITIME = 2.

NC = the number of coefficients.

CENTER = the center of the previous annulus.

RDS = the radius of the circle we are working with.

$\mathcal{Z}_0$ IRT = 0;

IRDS = 0;

IT = 0;

K = 0;

ITIME = 1;

CENTER = 0;

RDS = 1;

A_1 NC = N + 1;

Comment: Test if z = 0 is a root. If it is store it and return to the calling program.

P_2 |A(NC)| ≠ 0? ;

$\mathcal{Z}_3$ Z = (0. , 0.);

Comment: The three T arrays are used to store the coefficients of the original polynomial and the coefficients of the deflated polynomials that result from applying Lehmer's algorithm.

$\mathcal{U}_4$ T(I) = A(I), (I=1, . . . , NC);

T(I, 3) = A(I);

Comment: NCT = the number of coefficients of the deflated polynomial.

$\mathcal{Z}_5$ NCT = NC;

$\mathcal{Z}_6$ T(I, 1) = T(I, 2), (I=1, . . . , NCT);

A_7 NCT = NCT - 1;

IT = IT + 1;

Comment: Construct the T polynomials, see discussion of method.

A_8 NX = NCT + 1 - I, (I=1, . . . , NCT);

$T(I, 2) = \overline{T(NCT+1, 1)} \cdot T(I+1, 1) - T(1, 1) \cdot \overline{T(NX, 1)};$

Comment: The bars above denote complex conjugates. If T(NCT, 2) is not zero we will divide the other coefficients T(I, 2) by it.

P_9 T(NCT, 2) = 0. ? ;

A_{10} $T(I, 2) = T(I, 2)/|T(NCT, 2)|$, (I=1, . . . , NCT);

Comment: If ITIME = 1 this is the initial circle we are testing.

P_{11} ITIME = 2? ;

Comment: We now test T(NCT, 2) to see if there is a root in the unit circle.

If T(NCT, 2) < 0. 0 then there is a root inside the circle.

If T(NCT, 2) > 0. 0 we must continue construction of the reduced T polynomials.

If T(NCT, 2) = 0. 0 then we must test to see if the reduced T polynomial with coefficients T(I, 1), (I=1,...,NCT+1) is a constant. It is a constant if T(I, 1) = 0. 0, (I=1, . . . , NCT).

If we have a constant polynomial then there is no root inside the unit circle.

If we do not have a constant polynomial then we must choose a new radius, one that is not so large (RDS = .75RDS) or not so small (RDS = 1. 5RDS) depending on whether the radius was being doubled or halved.

P_{12} T(NCT, 2) = 0. 0? ;

P_{13} T(NCT, 2) < 0. 0? ;

Comment: IRT = 1 if there is a root inside the circle with radius 2RDS.

If NCT = 1 then our current reduced polynomial T is a constant and the next reduced polynomial T, if it were constructed would be zero. In this case there is no root inside the circle with radius RDS.

P_{14} IRT = 1 and NCT = 1? ;

P_{15} NCT = 1? ;

P_{16} T(I, 1) ≠ 0. 0, (I=1, . . . , NCT)? ;

P_{17} IRT = 1? ;

A_{18} RDS = 2RDS;

Comment: Set the flag to indicate the radius is being doubled.

$\mathcal{Z}_{19}$ IRDS = 2;

Comment: If we reach this point in the program we know that f(z) does have a zero inside the unit circle. We now halve the radius until we find a circle inside which there is no zero.

IRT = 1 if there is a root inside the circle with radius RDS.

IRDS = 2 if the radius was being doubled.

P_{20} IRDS = 2? ;

$\mathcal{Z}_{21}$ IRT = 1;

IRDS = 1;

A_{22} RDS = RDS/ 2. ;

Comment: Transform the coefficients for the new radius.

A_{23} $T(I, 1) = T(I, 3) \cdot RDS(ITIME)^{NC-I}$, (I=1, . . . , N);

Comment: IRDS = 2 if radius was being doubled.

P_{24} IRDS = 2? ;

Comment: When we reach this point in the program we know the

radius was being halved so we want to try a radius not so small.

A_{25} RDS = 1.5RDS;

Comment: When we reach this point in the program we know the radius was being doubled so we want to try a radius not so large.

A_{26} RDS = .75RDS;

Comment: Halve the radius since we were doubling.

A_{27} RDS = RDS/2.;

Comment: There is a root in the annulus RDS-2RDS. This annulus can be completely covered by eight overlapping circles of radius 0.8RDS. CT(I), (I=1, . . . , 8) are the eight circle centers. CT1 and CTR are used for temporary storage of computed values.

A_{28} CT1 = 1.6235883 · RDS(ITIME);

CT(1) = CT1 + CENTER;

A_{29} CTR = 0.7853981634 · (I-1), (I=2, . . . , 8);

CT(I) = CT1 (cos(CTR), sin(CTR)) + CENTER;

A_{30} RDS(2) = 0.8 · RDS(ITIME);

3_{31} RDS(1) = RDS(2);

ITIME = 2;

IRT = 0;

IRDS = 0;

K = 0;

$\mathfrak{Z}_{32}$ T(I, 3) = A(I), (I=1, . . . , NC);

Comment: Have all eight circles been tried?

P_{33} K = 8? ;

A_{34} K = K + 1;

NP1 = N + 1;

Comment: NP1 is temporary storage.

Comment: Transform the coefficients of the polynomial equations for the new circle center.

A_{35} T(I, 3) = T(I, 3) + CT(K) · T(I-1, 3), (I=2, . . . , NP1);

A_{36} NP1 = NP1 - 1;

P_{37} NP1 $\geq$ 2? ;

$\mathfrak{Z}_{38}$ T(NC, 1) = T(NC, 3);

Comment: Test for a root inside the circle.

P_{39} T(NCT, 2) < 0? ;

P_{40} T(NCT, 2) = 0? ;

Comment: IRT = 1 if there is a root inside the circle with radius 2RDS. If NCT = 1 then the T polynomial is a constant. In this case there is no root inside the circle with radius RDS and we have found an annulus which contains a root of f(z).

P_{41} IRT = 1 and NCT = 1? ;

Comment: If IRT $\neq$ 1 and NCT = 1 then there is no root inside the

circle. In this case we try the next of the eight circles.

P_{42} NCT = 1? ;

Comment: If NCT ≠ 1 then we perform the transformation on the T polynomial again.

Comment: Test to see if the reduced T polynomial with coefficient T(I, 1), (I=1, . . . , NCT+1) is a constant. It is a constant if T(I, 1) = 0. 0 for (I=1, . . . , NCT).

P_{43} T(I, 1) ≠ 0? , (I=1, . . . , NCT);

Comment: If IRT = 1 there is a root inside the circle with radius RDS.

P_{44} IRT = 1?;

Comment: IRDS = 32 indicates the radius was multiplied by 3/2. Go back to the previous radius since we are going to try a new circle.

P_{45} If IRDS = 32 then set RDS(2) = RDS(1);

Comment: Try a larger radius since we do not get a valid test with this radius, (the reduced T polynomial is not a constant).

$\mathcal{J}_{46}$ IRDS = 32;

A_{47} RDS(2) = 1. 5RDS(2);

Comment: IRT = 1 indicates there is a root inside the circle with radius RDS.

IRDS = 1 indicates the radius is being halved.

$\mathcal{Z}_{48}$ IRT = 1;

IRDS = 1;

Comment: If the radius is less than 10^{-3} accept the center of the circle as a root approximation.

P_{49} RDS(2) < 1.0x10^{-3}?;

A_{50} RDS(2) = RDS(2)/2.;

Comment: When we reach this point we know there is a root in the annulus RDS-2RDS. If the distance between annulus centers is within the tolerance accept the annulus center as a root approximation.

P_{51} $\left|\frac{\text{CT(K)-CENTER}}{\text{CT(K)}}\right| < 20.0?$;

Comment: If the annulus center is not accepted as a root, store the center and start going around the annulus with overlapping circles.

$\mathcal{Z}_{52}$ CENTER = CT(K);

Comment: At this point we have accepted the annulus center as a root approximation. Store it and return to the calling program.

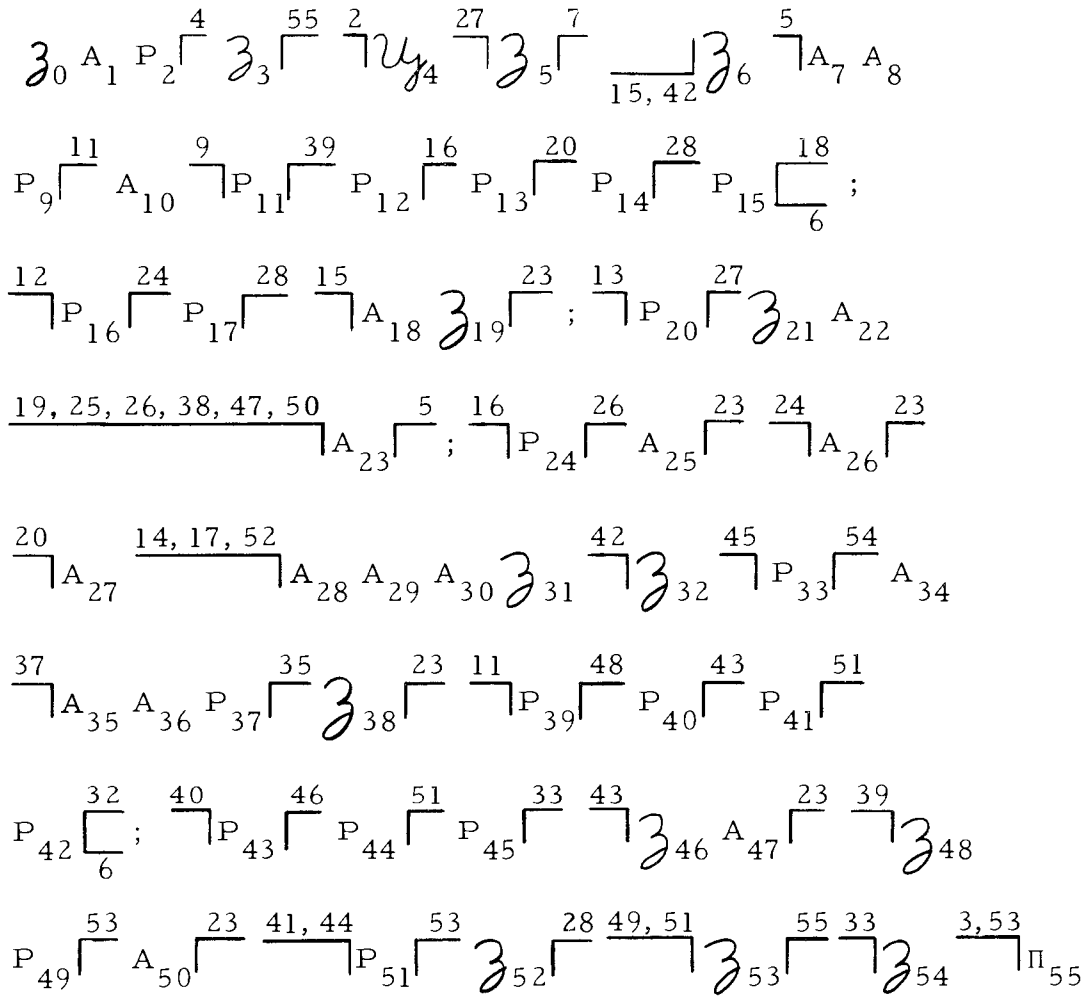
$\mathcal{Z}_{53}$ Z = CT(K);

Comment: If we reach this point in the program no root could be found in any of the eight overlapping circles due to rounding error. Set the indicator N = 0, and return to the calling program.

ζ_{54} N = 0;

Π_{55} Transfer to calling program.

Combining the above operators, the logical scheme of subroutine LEHMER has the form:



E(NEWTON) SUBROUTINE NEWTON(A, N, IT, Z1, PZ)

Comment: A(I) = the complex coefficients of the polynomial equation

(I-1, . . . , N+1).

N = the degree of the polynomial equation.

IT = the number of iterations performed for the root.

Z1 = the independent variable.

PZ = the value of the polynomial equation at Z1.

DPZ = the derivative of the polynomial equation at Z1.

$\mathcal{Z}_0$ IT = 0;

A_1 IT = IT + 1;

$\mathcal{Z}_2$ PZ = A(1);

DPZ = A(1);

Comment: Using iterated synthetic division form PZ and DPZ.

A_3 PZ = A(I) + Z1 • PZ, (I=2, . . . , N);

DPZ = PZ + Z1 • DPZ;

A_4 PZ = A(N+1) + Z1 • PZ;

Z2 = Z1 - PZ/DPZ;

Comment: Test for convergence.

P_5 $|PZ| < 1.0 \times 10^{-20}$? ;

P_6 $|Z1| = 0.$? ;

P_7 $\left|\frac{Z2-Z1}{Z1}\right| < 1.0 \times 10^{-10}$? ;

Comment: Have we exceeded the iteration limit of 300?

P_8 IT = 300? ;

Comment: Store the new root approximation and iterate again.

$\mathcal{Z}_9$ Z1 = Z2;

Comment: When we reach this point we have accepted Z2 as a root so we store it and return to the calling program.

β_{10} Z1 = Z2;

Π_{11} Transfer to the calling program.

Combining the preceding operators, the logical scheme of subroutine NEWTON has the form:

$$\beta_0 \overset{9}{\urcorner} A_1 \beta_2 A_3 A_4 P_5 \overset{10}{\lceil} P_6 P_7 \overset{10}{\lceil} P_8 \overset{10}{\lceil} \overset{6}{\urcorner} \beta_9 \overset{1}{\lceil} \overset{5,7,8}{\urcorner} \beta_{10} \Pi_{11}.$$

E(POLY) FUNCTION POLY(A, N, Z)

Comment: A(I) = the complex coefficients of the polynomial equation (I=1, . . . , N+1).

N = the degree of the polynomial equation.

Z = the value at which the polynomial equation is evaluated.

POLY = the value of the polynomial equation evaluated at Z.

β_0 POLY = A(1);

A_1 POLY = Z * POLY + A(I+1), (I=1, . . . , N);

Π_2 Transfer to the calling program.

Combining the above operators, the logical scheme of subprogram POLY has the form:

$$\beta_0 A_1 \Pi_2.$$

E(QUAD) SUBROUTINE QUAD(A, B, C, Z1, Z2)

Comment: Solution using the quadratic formula.

A, B and C = the coefficients of the quadratic equation.

Z1 and Z2 = the roots of the quadratic equation.

A_0 DISC = $(B^2-4AC)^{1/2}$;

Z1 = (−B+DISC)/2A;

Z2 = (−B−DISC)/2A;

Π_1 Transfer to the calling program.

Combining the above operators, the logical scheme of subrouting QUAD has the form:

$$A_0 \Pi_1 .$$

The FORTRAN subprograms which go together to form Lehmer's computer program are listed in Figure 1.

2. Muller's Algorithm

Muller's algorithm (8), an iterative procedure, uses the Lagrange interpolation formula to fit a quadratic polynomial L through three distinct points (z_1, f_1), (z_2, f_2), and (z_3, f_3), $f_i = f(z_i)$, (i=1, 2, 3) where

(1) $$f(z) = a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots + a_{n-1} z + a_n$$

Figure 1. Lehmer's FORTRAN IV program.

```
      PROGRAM LEHMOUT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION A(16),AR(16),NI(15,2),REM(15),RTZ(15,2)
     1,Z(2)
      COMPLEX A,AR,POLY,REM,RTZ,Z
C
C     KC=0 IF ALL COEFFICIENTS ARE REAL.
      CALL SECOND(TIME1)
      READ(5,1000)NCASES
      DO 1 ICASES=1,NCASES
      CALL SECOND(TSTART)
      WRITE(6,999)
C
      LRTS=0
      NR=0
C
      READ(5,1000)NC,KC,(A(I),I=1,NC)
      N=NC-1
      NRTS=N
C
      DO 12 I=1,NC
   12 AR(I)=A(I)
C
   14 L=1
   15 IF(N.EQ.2)GO TO 18
      IF(N.EQ.1)GO TO 17
      CALL LEHMER(AR,N,IT,Z)
C
C     N=0 IF LEHMERS METHOD COULD NOT FIND THE NEXT ROOT.
      IF(N.NE.0)GO TO 16
      WRITE(6,1100)
      GO TO 50
   16 NR=NR+1
      LRTS=NR
      NI(NR,1)=IT
      RTZ(NR,1)=Z
      IF(CABS(Z).NE.0.)GO TO 20
      N=N-1
      NI(NR,2)=NI(NR,1)
      RTZ(NR,2)=RTZ(NR,1)
      REM(NR)=0.
      GO TO 15
C
   17 NR=NR+1
      Z=-AR(2)/AR(1)
      NI(NR,1)=0
      RTZ(NR,1)=Z
      IT=0
      GO TO 21
C
   18 CALL QUAD(AR(1),AR(2),AR(3),Z(1),Z(2))
      DO 19 I=1,2
      NR=NR+1
      NI(NR,1)=1
      RTZ(NR,1)=Z(I)
      NI(NR,2)=0
      RTZ(NR,2)=Z(I)
   19 REM(NR)=POLY(A,NRTS,Z(I))
      GO TO 50
C
C     USE NEWTONS METHOD TO GET CLOSER ROOTS.
   20 CALL NEWTON(AR,N,IT,Z,REM(NR))
   21 NI(NR,2)=IT
      RTZ(NR,2)=Z
```

```
      REM(NR)=POLY(A,NRTS,Z)
      IF(NR.GE.NRTS)GO TO 50
      IF(CABS(REM(NR)).GE.1.0)GO TO 50

      CALCULATE COEFFICIENTS FOR THE REDUCED EQUATION.
      DO 3 I=2,N
    3 AR(I)=AR(I)+Z*AR(I-1)
      N=N-1
      IF(L.EQ.2)GO TO 14
      IF(KC.NE.0)GO TO 15
      IF(AIMAG(Z).EQ.0.)GO TO 15
      IF(ABS(REAL(Z)/AIMAG(Z)).GT.1.0E2)GO TO 15
      L=2
      Z=CONJG(Z)
      IT=0
      GO TO 16
C
   50 CONTINUE
      WRITE(6,1010)ICASES,NC,(A(I),I=1,NC)
      WRITE(6,1020)NR,(RTZ(I,2),NI(I,2),REM(I),I=1,NR)
      WRITE(6,1015) LRTS,(RTZ(I,1),NI(I,1),I=1,LRTS)
      CALL SECOND(TEND)
      TSEC=TEND-TSTART
      WRITE(6,1300)TSEC
  100 CONTINUE
      CALL SECOND(TIME2)
      TTIME=TIME2-TIME1
      WRITE(6,1310)TTIME
      CALL EXIT
  990 FORMAT(1H1)
 1000 FORMAT(2I5/(8F10.0))
 1010 FORMAT(1H0//////1H0,14X,*CASE*,I3,*,  *,I3,
     1* COEFFICIENTS*//(1H ,14X,2E20.12))
 1015 FORMAT(1H0//1H0,14X,I3,
     1* ROOTS FOUND BY LEHMERS METHOD.*//
     2 (1H ,14X,2E20.12,*  NI= *,I4))
 1020 FORMAT(1H0/1H0,28X,I3,* ROOTS*,39X,
     1*REMAINDERS*//(1H ,14X,2E20.12,*  NI= *,I3,2E17.8))
 1100 FORMAT(1H0,
     1*LEHMERS METHOD COULD NOT FIND ALL THE ROOTS.*)
 1300 FORMAT(1H0,//,15X,*EXECUTION TIME=*,F10.3,
     1* SECONDS*)
 1310 FORMAT(1H0,14X,*TOTAL EXECUTION TIME=*,F10.3,
     1* SECONDS*)
      END



      SUBROUTINE LEHMER(A,N,IT,Z)
      DIMENSION A(16),CT(8),RDS(2),T(16,3)
      COMPLEX A,CENTER,CT,T,Z
C
C     A=COEFFICIENTS OF POLYNOMIAL.
C     IT=NO. OF ITERATIONS.
C     N= DEGREE OF THE EQUATION.
C     NC= THE NUMBER OF COEFFICIENTS.
C     NCT=TEMPORARY NO. OF COEFFICIENTS.
C     IRT=1 IF ROOT INSIDE CIRCLE WITH RADIUS RDS.
C     RDS= RADIUS OF CIRCLE BEING TESTED FOR A ROOT.
C     CENTER= THE CENTER OF THE ANNULUS THAT CONTAINS A ROOT
      IRT=0
      IRDS=0
      IT=0
      K=0
      ITIME=1
      NC=N+1
      CENTER=0.
      RDS=1.
```

```
C
C       TEST FOR Z=0 A ROOT.
  5     IF(CABS(A(NC)).NE.0.)GO TO 20
        Z=(0.,0.)
        GO TO 600
C
 20     DO 25 I=1,NC
        T(I)=A(I)
 25     T(I,3)=A(I)
C
 26     NCT=NC
        GO TO 29
C
 27     DO 28 I=1,NCT
 28     T(I,1)=T(I,2)
 29     NCT=NCT-1
        IT=IT+1
        DO 30 I=1,NCT
        NX=NCT+1-I
 30     T(I,2)=CONJG(T(NCT+1,1))*T(I+1,2)-T(1,1)*CONJG(T(NX,1))
        IF(T(NCT,2).EQ.0.)GO TO 34
C
C       NORMALIZE COEFFICIENTS OF TJ(F(Z))
        DO 32 I=1,NCT
 32     T(I,2)=T(I,2)/CABS(T(NCT,2))
C
C       DOES F(Z) HAVE A ROOT INSIDE THE CIRCLE.
C       TEST FOR ROOT INSIDE CIRCLE.
C       IF T(NCT,J).LT.0.0 THEN ROOT INSIDE CIRCLE.
C       IF T(NCT,J).GT.0.0 CONTINUE ITERATING.
C       IF T(NCT,J).EQ.0.0 THEN TEST T(NCT,J-1).
C       IF T(NCT,J-1).EQ.CONSTANT THEN NO ROOT INSIDE CIRCLE.
C       IF T(NCT,J-1).NE.CONSTANT THEN CHOOSE A RADIUS
C       NOT SO LARGE (RDS=0.75RDS)
C       OR NOT SO SMALL (RDS=1.5RDS),
C       DEPENDING ON THE PREVIOUS RADIUS.
C
 34     IF(ITIME.EQ.2)GO TO 155
        IF(T(NCT,2))60,45,35
C
C       IRT=1 IF ROOT INSIDE PREVIOUS CIRCLE.
C       IF NCT=1, T(F(Z)) IS A CONSTANT AND THE NEXT
C       T(F(Z))=0, IN THIS CASE THERE IS NO ROOT
C       INSIDE THE CIRCLE.
C
 35     IF(IRT.EQ.1.AND.NCT.EQ.1)GO TO 100
        IF(NCT.EQ.1)GO TO 55
        GO TO 27
C
C       IF T(I,J-1)=CONSTANT THEN NO ROOTS INSIDE CIRCLE.
C       TEST IF T(I,J-1).NE.0. (IF TJ-1(Z).NE.CONSTANT).
 45     DO 50 I=1,NCT
 50     IF(T(I,1).NE.0.)GO TO 75
C
C       IRT=1 IF ROOT INSIDE PREVIOUS CIRCLE.
        IF(IRT.EQ.1)GO TO 100
C
C       IRDS=2 IF RADIUS IS BEING DOUBLED.
 55     RDS=2.*RDS
        IRDS=2
        GO TO 64
C
C       F(Z) DOES HAVE A ZERO INSIDE THE UNIT CIRCLE.
C       HALVE THE RADIUS UNTIL WE FIND A CIRCLE INSIDE
C       WHICH THERE IS NO ZERO.
C       IRDS=1 IF RADIUS IS BEING HALVED.
C       IRT=1 IF ROOT INSIDE CIRCLE WITH RADIUS RDS.
 60     IF(IRDS.EQ.2)GO TO 90
```

```
      IRT=1
      IRDS=1
      RDS=RDS/2.

C     TRANSFORM COEFFICIENTS FOR NEW RADIUS.
 64   DO 65 I=1,N
 65   T(I,1)=T(I,3)*RDS(ITIME)**(NC-I)
      GO TO 26
C
C     IF IRDS=2 THE RADIUS WAS BEING DOUBLED.
 75   IF(IRDS.EQ.2) GO TO 80

C     RADIUS WAS BEING HALVED,
C     TRY A RADIUS NOT SO SMALL,
C     MULTIPLY BY 1.5
      RDS=1.5*RDS
      GO TO 64

C     THE RADIUS WAS BEING DOUBLED,
C     TRY A RADIUS NOT SO LARGE,
C     MULTIPLY BY .75
 80   RDS=.75*RDS
      GO TO 64
C
C     HALVE RADIUS SINCE WERE DOUBLING.
 90   RDS=RDS/2.
C
C     THERE IS A ROOT IN ANNULUS RDS-2RDS.
C     THIS ANNULUS CAN BE COMPLETLY COVERED BY
C     EIGHT OVERLAPPING CIRCLES.
C     CT(I)=THE EIGHT CIRCLE CENTERS,
C     EACH CIRCLE IS OF RADIUS .8RDS.
 100  CT1=RDS(ITIME)*1.62358830
      CT(1)=CT1+CENTER
      DO 110 I=2,8
      CTR=(I-1)*0.7853981634
 110  CT(I)=CT1*CMPLX(COS(CTR),SIN(CTR))+CENTER

      RDS(2)=.8*RDS(ITIME)
      RDS(1)=RDS(2)
      ITIME=2

      IRT=0
      IRDS=0
      K=0
 119  DO 120 I=1,NC
 120  T(I,3)=A(I)

C     HAVE ALL 8 CIRCLES BEEN TRIED.
 128  IF(K.EQ.8)GO TO 550
      K=K+1
      NP1=N+1

C     TRANSFORM COEFFICIENTS FOR NEW CENTER.
 129  DO 130 I=2,NP1
 130  T(I,3)= T(I,3)+CT(K)*T(I-1,3)
      NP1=NP1-1
      IF(NP1.GE.2)GO TO 129
      T(NC,1)=T(NC,3)
      GO TO 64

 155  IF(T(NCT,2))180,170,160
C
C     IRT=1 IF ROOT INSIDE CIRCLE WITH RADIUS 2RDS.
C     IF NCT=1, T(F(Z)) IS A CONSTANT AND THE NEXT
C     T(F(0))=0,
C     IN THIS CASE THERE IS NO ROOT INSIDE THE CIRCLE.
 160  IF(IRT.EQ.1.AND.NCT.EQ.1)GO TO 200
```

```
C
C       NO ROOT IN THIS CIRCLE, TRY THE NEXT CT(K).
C       TRANSFORM ORIGINAL COEFFICIENTS TO NEW CENTER.
        IF(NCT.EQ.1)GO TO 119

C       TRY AGAIN, A CONSTANT OR 0 IS NEEDED FOR T(I,J)
C       BEFORE GOING TO THE NEXT CIRCLE.
        GO TO 27
C
C       IF T(I,J-1)=CONSTANT THEN NO ROOTS INSIDE CIRCLE.
  170   DO 171 I=1,NCT
  171   IF(T(I,1).NE.0.)GO TO 175
C
C       T(I,J-1)=CONSTANT.
C       NO ROOT IN THIS CIRCLE,TRY THE NEXT CT(K).
C       IS THERE A ROOT IN THE ANNULUS.
        IF(IRT.EQ.1)GO TO 200
        IF(IRDS.EQ.32)RDS(2)=RDS(1)
        GO TO 128
C
C       TRY A LARGER RADIUS.
C       IRDS=32 INDICATES MULTIPLYING RDS BY 3/2.
  175   IRDS=32
        RDS(2)=1.5*RDS(2)
        GO TO 64
C
C       IRDS=1 IF RADIUS IS BEING HALVED.
C       IRT=1 IF ROOT INSIDE CIRCLE WITH RADIUS RDS.
  180   IRT=1
        IRDS=1
        IF(RDS(2).LT.1.0E-3)GO TO 300
        RDS(2)=RDS(2)/2.

C       USE TRANSFORMED COEFFICIENTS.
        GO TO 64
C
C       THERE IS A ROOT IN THE ANNULUS RDS-2RDS.
C       IF WITHIN TOLERANCE RETURN AND CALL
C       SUBROUTINE NEWTON.
  200   IF(CABS((CT(K)-CENTER)/CT(K)).LT.20.0)GO TO 300
        CENTER=CT(K)
        GO TO 100

C       STORE APPROXIMATION TO ROOT.
  300   Z=CT(K)
        GO TO 600
  550   N=0
  600   RETURN
        END



        SUBROUTINE NEWTON(A,N,IT,Z1,PZ)
        DIMENSION A(16)
        COMPLEX A,DPZ,PZ,Z1,Z2

C       A=COEFFICIENTS.
C       N=DEGREE OF POLYNOMIAL.
C       IT=NO. OF ITERATIONS.
C       Z1=INDEPENDENT VARIABLE.
C       PZ=THE VALUE OF THE POLYNOMIAL AT Z=Z1.
C       DPZ=THE DERIVATIVE OF PZ.

        IT=0
    5   IT=IT+1
        PZ=A(1)
        DPZ=A(1)
        DO 10 I=2,N
        PZ=A(I)+Z1*PZ
   10   DPZ=PZ+Z1*DPZ
```

```
      PZ=A(N+1)+Z1*PZ
      Z2=Z1-PZ/DPZ
      IF(CABS(PZ).LT.1.0E-20)GO TO 100
   15 IF(CABS(Z1).EQ.0.)GO TO 2
      IF(CABS((Z2-Z1)/Z1).LT.1.0E-10)GO TO 100
      IF(IT.EQ.300)GO TO 100
   20 Z1=Z2
      GO TO 5
  100 Z1=Z2
      RETURN
      END


      COMPLEX FUNCTION POLY(A,N,Z)
      DIMENSION A(16)
      COMPLEX A,Z
C
C     A=COEFFICIENTS OF POLYNOMIAL.
C     N=DEGREE OF POLYNOMIAL.
C     Z=VALUE AT WHICH POLYNOMIAL IS EVALUATED.
C     POLY=POLYNOMIAL EVALUATED AT Z.
C

      POLY=A(1)
      DO 20 I=1,N
   20 POLY=Z*POLY+A(I+1)
      RETURN
      END


      SUBROUTINE QUAD(A,B,C,Z1,Z2)
      COMPLEX A,B,C,DISC,Z1,Z2
      DISC=CSQRT(B*B-4.*A*C)
      Z1=(-B+DISC)/(2.*A)
      Z2=(-B-DISC)/(2.*A)
      RETURN
      END
```

is the polynomial whose zeros are desired. The coefficients $a_0, a_1, \ldots, a_n$ are complex numbers and $a_0 \neq 0$. The root of the quadratic polynomial equation closest to z_3 is taken as z_4. $f_4 = f(z_4)$ is computed and if

$$|f_4| < \delta_1 \tag{2}$$

or

$$|z_4 - z_3| / |z_3| < \delta_2$$

where δ_1 and δ_2 are tolerance constants, then z_4 is accepted as a root to (1). If the tests fail, z_1 is dropped and the points (z_2, f_2), (z_3, f_3), and (z_4, f_4) become the new points (z_1, f_1), (z_2, f_2), and (z_3, f_3).

A new quadratic polynomial is fitted through the new set of points (z_1, f_1), (z_2, f_2), and (z_3, f_3). The root of the quadratic polynomial equation closest to z_3 is taken as z_4, $f_4 = f(z_4)$ is computed and tested in Equation (2) as described above. The iterations continue until either convergence occurs or a fixed number of iterations are performed.

The iterative procedure is started by letting $z_1 = -1.$, $z_2 = 1.$ and $z_3 = 0.01$ and evaluating $f(z_1)$, $f(z_2)$ and $f(z_3)$. $z_3 = 0.01$ is used instead of $z_3 = 0.0$ since there could be a zero root. This allows the starting values to be changed to values different from the roots that have been found. More is said about this a little later.

Muller (8) suggests using $z_1 = -1.$, $z_2 = 1.$, $z_3 = 0.0$ and

$$a_n - a_{n-1} + a_{n-2} \quad \text{for} \quad f(z_1),$$

$$a_n + a_{n-1} + a_{n-2} \quad \text{for} \quad f(z_2),$$

$$a_n \quad \text{for} \quad f(z_3),$$

to save evaluating the function explicitly. This was tried with the result that more iterations and time were required for convergence on a root than when the functions $f(z_i)$, $(i=1, 2, 3)$ are evaluated explicitly.

The results from solving the nine exercises from Milne (6) and using Muller's suggestion are presented on the following pages. These results should be compared with the results from solving the nine exercises using explicit values for $f(z_i)$, $(i=1, 2, 3)$, (see p. 95).

To avoid re-calculation of zeros already found synthetic division may be used to reduce the degree of the polynomial equation. This can lead to a serious accumulation of rounding errors. Rather than use synthetic division to extract linear factors from the polynomial equation implicit division was performed on the value of the function and a deflated value of the function was obtained. If nr zeros, r_i, $(i=1, \ldots, nr)$ have been found then the deflated values of the function $f_{nr}(z_k)$, $(k=1, 2, 3)$ are formed where

Table 1. Results from using Muller's f(z) approximations for starting values.

Case 1, 4 Coefficients

1.000000000000E+00 -0.
-0. -0.
-1.000000000000E+00 -0.
-4.000000000000E+00 -0.

3 Roots			Remainders	
-8.981609516297E-01	1.191670795605E+00	NI= 9	0.	-2.13162821E-14
-8.981609516297E-01	-1.191670795605E+00	NI= 0	0.	2.13162821E-14
1.796321903259E+00	1.899500362956E-16	NI= 2	-8.52651283E-14	1.64882656E-15

Execution time = .024 seconds

Case 2, 5 Coefficients

1.000000000000E+00 -0.
-2.037900000000E+00 -0.
-1.542450000000E+01 -0.
1.566960000000E+01 -0.
3.549360000000E+01 -0.

4 Roots			Remainders	
-1.201998596673E+00	0.	NI= 6	2.27373675E-13	0.
2.124387030181E+00	0.	NI= 7	2.27373675E-13	0.
-3.211994374397E+00	0.	NI= 4	-9.09494702E-13	0.
4.327505940890E+00	0.	NI= 4	-2.27373675E-13	0.

Execution time = .030 seconds

Case 3, 5 Coefficients

1.000000000000E+00 -0.
-2.000000000000E+00 -0.
-4.000000000000E+00 -0.
-4.000000000000E+00 -0.
4.000000000000E+00 -0.

4 Roots			Remainders	
5.857864376269E-01	0.	NI= 5	2.84217094E-14	0.
-1.000000000000E+00	1.000000000000E+00	NI= 7	0.	0.
-1.000000000000E+00	-1.000000000000E+00	NI= 0	0.	0.
3.414213562373E+00	1.211690350419E-27	NI= 4	6.53699317E-13	7.02067027E-26

Execution time = .026 seconds

Case 4, 5 Coefficients

4.000000000000E+00 -0.
-2.400000000000E+01 -0.
4.400000000000E+01 -0.
-2.400000000000E+01 -0.
3.000000000000E+00 -0.

(Continued)

4 Roots			Remainders	
1.771243444677E-01	0.	NI= 5	0.	0.
6.339745962156E-01	0.	NI= 7	1.42108547E-14	0.
2.366025403785E+00	8.271806125530E-25	NI= 4	-7.38964445E-13	-5.73087539E-24
2.822875655532E+00	2.520315928873E-25	NI= 4	9.23705556E-13	2.66725167E-24

Execution time = .030 seconds

Case 5, 5 Coefficients

2.000000000000E+00 -0.
1.600000000000E+01 -0.
1.000000000000E+00 -0.
-7.400000000000E+01 -0.
5.600000000000E+01 -0.

4 Roots			Remainders	
1.121320343562E+00	0.	NI= 11	-2.27373675E-13	0.
1.123105625617E+00	0.	NI= 6	0.	0.
-3.121320343560E+00	0.	NI= 4	2.04636308E-12	0.
-7.123105625618E+00	0.	NI= 5	2.04636308E-12	0.

Execution time = .032 seconds

Case 6, 4 Coefficients

1.000000000000E+00 -0.
-6.026600000000E+00 -0.
4.304800000000E+00 -0.
1.595330000000E+01 -0.

3 Roots			Remainders	
-1.216399518172E+00	0.	NI= 6	-5.68434189E-14	0.
3.612590155512E+00	5.027707160674E-24	NI= 4	5.68434189E-14	-4.32628004E-25
3.630409362659E+00	6.720842476993E-25	NI= 4	-5.68434189E-14	5.80454240E-26

Execution time = .024 seconds

Case 7, 5 Coefficients

1.000000000000E+00 -0.
1.200000000000E+01 -0.
-9.500000000000E+00 -0.
-6.000000000000E+00 -0.
4.500000000000E+00 -0.

4 Roots			Remainders	
7.071067811872E-01	0.	NI= 11	0.	0.
-7.071067811865E-01	-1.225741041988E-20	NI= 7	0.	-2.94432992E-19
7.082039325014E-01	-3.032780167743E-23	NI= 4	5.68434189E-14	-6.31823110E-25
-1.270820393250E+01	-2.300938973373E-31	NI= 5	-1.18831167E-10	4.97007612E-28

Execution time = .034 seconds

(Continued)

Case 8, 5 Coefficients

1.000000000000E+00 -0.
-6.000000000000E+00 -0.
-1.130000000000E+02 -0.
5.040000000000E+02 -0.
2.436000000000E+03 -0.

4 Roots			Remainders	
-3.164414002969E+00	0.	NI= 6	0.	0.
9.164414003060E+00	0.	NI= 12	1.45519152E-11	0.
-9.165151389912E+00	0.	NI= 4	-2.91038305E-11	0.
9.165151389726E+00	0.	NI= 4	0.	0.

Execution time = .032 seconds

Case 9, 5 Coefficients

1.000000000000E+00 -0.
1.600000000000E+01 -0.
1.100000000000E+01 -0.
-2.240000000000E+02 -0.
2.860000000000E+02 -0.

4 Roots			Remainders	
1.841821538748E+00	-4.311068539088E-01	NI= 8	-1.81898940E-12	1.13686838E-12
1.841821538748E+00	4.311068539088E-01	NI= 0	-1.81898949E-12	-1.13686838E-12
-5.726958892481E+00	1.419949629398E-23	NI= 4	9.09494702E-12	6.71609424E-24
-1.395668427502E+01	-7.754818242685E-25	NI= 4	-3.81987775E-11	1.59408790E-21

Execution time = .030 seconds
Total execution time = .292 seconds

$$f_{nr}(z_k) = f(z_k) / \prod_{i=1}^{nr} (z_k - r_i), \qquad \text{(Ref. 1)}$$

This procedure directs the values of the function away from previously found zeros and avoids the accumulation of rounding errors.

If $z_k = r_i$ for some i, $(1 \leq i \leq nr)$, where z_k, $(k=1, 2, 3)$ are the starting guesses, then z_k must be modified to avoid division by zero. When this occurred in the computer program z_k was replaced by $.8z_k$ and satisfactory results were obtained. If during the iteration to a new root $z_j = r_i$, $(j=1, 2, \ldots)$, for some i, $(1 \leq i \leq nr)$, then the iteration procedure is stopped and the roots that have been found are printed out. This never happened during any of the test cases.

Operator Program

On the following pages are described the operator programs which go together to form Muller's program. Each of the operators is first defined and then the program is documented as a string of operators.

Muller's Program--Operator Programming Definitions:

Π_0 Input the program and input data into the memory of the machine.

Comment: A(I) = the complex coefficients of the polynomial equation (I=1, . . . , NC).

NC = the degree of the polynomial equation plus one.

N = the degree of the polynomial equation.

KC = 0 if the coefficients are all real.

NI(I) = the number of iterations per root, (I=1, . . . , N).

NR = the number of roots found.

RTZ(I) = the complex roots of the polynomial equation, (I=1, . . . , N).

REM(I) = the complex remainders of the polynomial equation, (I=1, . . . , N).

A_1 Translate the input data into binary.

A_2 N = NC - 1;

E_3(MULLER) CALL MULLER(A, N, KC, NI, NR, RTZ, REM);

Π_4 Output the coefficients, roots and remainders.

$\mathscr{L}_5$ Stop the machine.

Combining the above operators, the logical scheme of the program has the form:

$$\Pi_0 A_1 A_2 E_3 \Pi_4 \mathscr{L}_5.$$

E(MULLER) SUBROUTINE MULLER(A, N, KC, NI, NR, RTZ, REM);

Comment: A(I) = the complex coefficients of the polynomial equation,

(I=1, . . . , N+1).

N = the degree of the polynomial equation.

KC = 0 if the coefficients are all real.

NI(I) = the number of iterations per root, (I=1, . . . , N).

NR = the number of roots found.

RTZ(I) = the complex roots of the polynomial equation, (I=1, . . . , N).

REM(I) = the complex remainders of the polynomial equation, (I=1, . . . , N).

$\mathcal{B}_0$ NR = 0;

NS = N;

A_1 I = N + 1;

Comment: Test to see if Z = 0 is a root, which is the case if the constant term of the polynomial equation is zero.

P_2 |A(I)| ≠ 0? ;

A_3 NR = NR + 1;

I = I - 1;

$\mathcal{B}_4$ NI(NR) = 0;

RTZ(NR) = 0. ;

REM(NR) = 0. ;

Comment: Have all the roots been found?

P_5 NR-N = 0? ;

Comment: L = 2 when complex conjugate roots have been found,

L = 1 otherwise.

Store the starting values.

3_6 IT = 0;

L = 1;

Z1 = -1. 0;

Z2 = 1. 0;

Z3 = 0. 01;

Comment: If any roots have been found we must make sure that none of them equal the starting values, Z1, Z2 and Z3.

P_7 NR = 0? ;

$E_8(9, 9)$ (I=1, 2, 3);

P_9 If Z(I) = RTZ(J) then set Z(I) = .8Z(I), (J=1, . . . , NR).

Comment: For Z1, Z2 and Z3 compute the values PZ1, PZ2, and PZ3 of the polynomial and the values PRZ1, PRZ2 and PRZ3 of the reduced polynomial.

E_{10}(POLY) (i=1, 2, 3);

CALL POLY(A, N, RTZ, NR, Zi, PZi, PRZi);

Comment: Compute the new root approximation using Muller's algorithm.

A_{11} L3 = (Z3-Z2)/(Z2-Z1);

B = PRZ1 • $L3^2$ - PRZ2 • $(L3+1)^2$ + PRZ3 • (2. • L3+1.);

SRT = $[B^2$ - 4. • PRZ3 • L3 • (L3+1.)

(PRZ1 • L3 - PRZ2 • (L3+1.)+PRZ3)$]^{1/2}$;

DEN = maximum (|B+SRT|, |B-SRT|);

P_{12} DEN = 0? ;

A_{13} L4 = -2•PRZ3•(L3+1.)/DEN;

Comment: If DEN = 0 we set L4 = 1.;

3_{14} L4 = 1.;

A_{15} Z4 = Z3 + L4•(Z3-Z2);

Comment: For Z4 compute the value PZ4 of the polynomial and the value PRZ4 of the reduced polynomial.

E_{16}(POLY) CALL POLY(A, N, RTZ, NR, Z4, PZ4, PRZ4);

Comment: Subroutine POLY returns N = 0 if the denominator used in dividing out the roots is zero. If this is the case stop iterating and print out the results found so far.

P_{17} N = 0? ;

Comment: If |PZ4/PZ3| > 10 then our increment L4 is too large so try L4/2. .

P_{18} |PZ4/PZ3| < 10? ;

A_{19} L4 = L4/2.;

Comment: Test for convergence.

P_{20} $|PZ4| < 10^{-20}$? ;

Comment: If Z3 = 0 iterate again.

P_{21} Z3 = 0? ;

P_{22} $|(Z4-Z4)/Z3| < 10^{-10}$? ;

Comment: Has the iteration limit been reached?

P_{23} IT $\geq$ 99? ;

Comment: Store values and iterate again.

A_{24} IT = IT + 1;

3_{25} Z1 = Z2;

Z2 = Z3;

Z3 = Z4;

PRZ1 = PRZ2;

PRZ2 = PRZ3;

PRZ3 = PRZ4;

Comment: At this point we have accepted Z4 as a root. Store the root and associated values.

A_{26} NR = NR + 1;

NI(NR) = IT + 1;

RTZ(NR) = Z4;

REM(NR) = PZ4;

Comment: Have all the roots been found?

P_{27} NR $\geq$ N? ;

Comment: If L = 2 the complex conjugate of the root has already been found.

P_{28} L = 2? ;

Comment: If the polynomial coefficients are not all real, iterate again.

P_{29} KC $\neq$ 0? ;

Comment: Is the imaginary part of the root zero? If so iterate for the next root.

P_{30} Imaginary RTZ(NR) = 0? ;

Comment: If the imaginary part of the root is small in comparison to the real part iterate for the next root.

P_{31} $\left|\frac{\text{real RTZ(NR)}}{\text{imaginary RTZ(NR)}}\right| > 100?$;

Comment: We are now ready to form the complex conjugate of the root.

$\mathcal{Z}_{32}$ L = 2;

IT = -1;

Z4 = $\overline{Z4}$;

Comment: Evaluate the polynomial at the new Z4.

E_{33}(POLY) CALL POLY(A, N, RTZ, NR, Z4, PZ4, PRZ4);

Comment: Subroutine POLY returns N = 0 if the denominator used in dividing out the roots is zero.

P_{34} N ≠ 0? ;

$\mathcal{Z}_{35}$ N = NS;

Π_{36} Transfer to the calling program.

If we combine the operators, the logical scheme of subroutine MULLER has the form:

```
ℨ0 A1 ⌋(5) P2 ⌈(6) A3 ℨ4 P5 ⌊(2) (36); 2, 28, 29, 30, 31 ⌉ ℨ6 P7 ⌈(10)
E8 P9 ⌉(7) E10 ⌉(25) A11 P12 ⌈(14) A13 ⌈(15); ⌉(12) ℨ14 ⌉(13, 19) A15
E16 P17 ⌈(35) P18 ⌈(20) A19 ⌈(15); ⌉(18) P20 P21 ⌈(24) P22 ⌈(26) P23 ⌈(26)
⌉(21) A24 ℨ25 ⌈(11) ⌉(20,22,23,33,34) A26 P27 ⌈(36) P28 ⌈(6) P29 ⌈(6) P30 ⌈(6)
P31 ⌈(6) ℨ32 E33 ⌈(26) P34 ⌈(26) ℨ35 ⌉(5, 27) Π36 .
```

E(POLY) SUBROUTINE POLY(A, N, RTZ, NR, Z, PZ, PRZ);

Comment: This subroutine evaluates a polynomial equation at a given complex number Z.

A(I) = the degree of the polynomial equation.

RTZ(I) = the roots of the polynomial equation, (I=1,...,N).

NR = the number of roots that have been found.

Z = the complex number at which the polynomial equation is evaluated.

PZ = the value of the polynomial function at Z.

PRZ = the quotient of the value of the polynomial function at Z divided by (Z-RTZ(I)), (I=1, . . . , NR). The polynomial function is evaluated at Z using iterated synthetic division.

A_0 PZ = Z(1);

PZ = Z * PZ + A(I+1), (I=1, . . . , N);

PRZ = PZ;

Comment: If NR = 0 there are no root values to divide out.

P_1 NR = 0? ;

Comment: Is the denominator zero? If so set N = 0 and return to the calling program.

$E_2(3, 4)$ (I=1, . . . , NR);

P_3 Z - RTZ(I) = 0? ;

A_4 PRZ = PRZ/(Z-RTZ(I));

$\mathfrak{Z}_5$ N = 0;

Π_6 Transfer to the calling program.

Combining the preceding operators, the logical scheme of the subroutine has the form:

$$A_0\, P_1 \overset{6}{\lceil}\; E_2\, P_3 \overset{5}{\lceil}\; A_4 \overset{6}{\lceil}\; \overset{3}{\rceil}\mathfrak{Z}_5 \overset{1,4}{\rceil}\Pi_6 .$$

The FORTRAN subprograms which go together to form Muller's computer program are listed in Figure 2.

3. The Quotient-Difference (QD) Algorithm (2)

Given the polynomial equation

Figure 2. Muller's FORTRAN IV program.

```
      PROGRAM INOUT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION A(16),NI(15),REM(15),RTZ(15)
      COMPLEX A,REM,RTZ
C
C     KC=0 FOR REAL COEFFICIENTS
C
      CALL SECOND(TIME1)
      READ(5,1000)NCASES
      DO 100 ICASES=1,NCASES
      CALL SECOND(TSTART)
      READ(5,1000)NC,KC,(A(I),I=1,NC)
      N=NC-1
      CALL MULLER(A,N,KC,NI,NR,RTZ,REM)
      WRITE(6,1010)ICASES,NC,(A(I),I=1,NC)
      WRITE(6,1020)NR,(RTZ(I),NI(I),REM(I),I=1,NR)
      CALL SECOND(TEND)
      TSEC=TEND-TSTART
      WRITE(6,1300)TSEC
  100 CONTINUE
      CALL SECOND(TIME2)
      TTIME=TIME2-TIME1
      WRITE(6,1310)TTIME
 1000 FORMAT(2I5/(8F10.0))
 1010 FORMAT(1H1///////1H0,14X,*CASE*,I3,*,  *,I3,
     1* COEFFICIENTS*//(1H ,14X,2E20.12))
 1020 FORMAT(1H0/1H0,28X,I3,* ROOTS*,39X,*REMAINDERS*//
     1(1H ,14X,2E20.12,*  NI= *,I3,2E17.8))
 1300 FORMAT(1H0,//,15X,*EXECUTION TIME=*,F10.3,
     1* SECONDS*)
 1310 FORMAT(1H0,14X,*TOTAL EXECUTION TIME=*,F10.3,
     1* SECONDS*)
      END


      SUBROUTINE MULLER(A,N,KC,NI,NR,RTZ,REM)
      DIMENSION A(16),NI(15),REM(15),RTZ(15)
      COMPLEX A,B,DEN,L3,L4,PZ1,PZ2,PZ3,PZ4,PRZ1,PRZ2,
     1PRZ3,PRZ4,REM,RTZ,SRT,Z1,Z2,Z3,Z4
      NR=0
      NS=N
      I=N+1
    5 IF(CABS(A(I)).NE.0.)GO TO 20
      NR=NR+1
      NI(NR)=0
      I=I-1
      RTZ(NR)=(0.,0.)
      REM(NR)=(0.,0.)
      IF(NR-N)5,1000,5
   20 IT=0
      L=1
      Z1=(-1.,0.)
      Z2=(1.,0.)
      Z3=(.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
C
```

```
 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)
C
C     COMPUTE B=P1*L3*L3-P2*(L3+1.)*(L3+1.)+P3*(2.*L3+1.)
      B=PRZ1*L3*L3-PRZ2*(L3+1.)*(L3+1.)+PRZ3*(2.*L3+1.)
C
C     SRT=CSQRT(B*B-4.*P3*L3*(L3+1.)*(P1*L3-P2*(L3+1.)+P3))
      SRT=CSQRT(B*B-4.*PRZ3*L3*(L3+1.)*(PRZ1*L3-PRZ2*
     1(L3+1.)+PRZ3))
C
C     DETERMINE MAX(B+SRT,B-SRT).
      DEN=B+SRT
      IF(CABS(DEN).LT.CABS(B-SRT))DEN=B-SRT
      IF(DEN.EQ.0.)GO TO 35

C     L4=-2.*P3(L3+1.)/DEN
      L4=-2.*PRZ3*(L3+1.)/DEN
      GO TO 40

 35   L4=(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(CABS(PZ4/PZ3).LT.10.)GO TO 50
      L4=L4/2.
      GO TO 40

 50   IF(CABS(PZ4).LT.1.0E-20)GO TO 100
      IF(Z3.EQ.0.)GO TO 65
      IF(CABS((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
      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(AIMAG(RTZ(NR)).EQ.0.)GO TO 20
      IF(ABS(REAL(RTZ(NR))/AIMAG(RTZ(NR))).GT.1.0E2)
     1GO TO 20
 110  L=2
      IT=-1
      Z4=CONJG(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)
      DIMENSION A(16),RTZ(15)
      COMPLEX A,PZ,PRZ,RTZ,Z
C     A=COEFFICIENTS OF POLYNOMIAL.
C     N=DEGREE OF POLYNOMIAL.
C     RTZ=ROOTS OF POLYNOMIAL.
C     NR=NO. OF ROOTS THAT HAVE BEEN FOUND.
C     Z=VALUE AT WHICH POLYNOMIAL IS EVALUATED.
C     PZ=POLYNOMIAL EVALUATED AT Z.
C     PRZ=VALUE OF POLYNOMIAL WITH ROOTS DIVIDED OUT.
      PZ=A(1)
C
      DO 20 I=1,N
 20   PZ=Z*PZ+A(I+1)
C
      PRZ=PZ
      IF(NR.EQ.0)GO TO 100
C
 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
```

$$f(z) = a_0 z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n = 0$$

where a_i are complex numbers. If all a_i are not different from zero, transform the equation to eliminate any zero coefficients. Use for a new center $ct = 1.$ and transform the equation. If this does not eliminate all zero coefficients try $ct = 1/2$ then $1/4,\ldots,1/128.$ The zero coefficients in equations $x^n + n = 1$, $n=3$ to 101 were successfully eliminated using this procedure. The required equation

$$f(z_*+ct) = a_0(z_*+ct)^n + a_1 (z_*+ct)^{n-1} + \ldots + a_n = 0,$$

where $z = z_* + ct,$ after expanding by the binomial theorem and collecting terms, reduces to the form

$$F(z_*) = c_0 z_*^n + c_1 z_*^{n-1} + \ldots + c_{n-1} z_* + c_n = 0,$$

where $c_0 = a_0$, $c_1 = nct a_0 + a_1$ etc.

This transformation can be accomplished using synthetic division and by noting that if $z = z_* + ct,$ then $z_* = z - ct$ and

$$f(z) = f(z_*+ct) = \phi(z_*) = \phi(z-ct)$$

so that

$$c_0(z-ct)^n + \ldots + c_{n-1}(z-ct) + c_n \equiv a_0 z^n + \ldots + a_{n-1} z + a_n$$

and using synthetic division to divide $f(z)$ by $z - ct$ we obtain c_n as the remainder. If again the quotients thus obtained are divided by $z - ct$ and so on, the successive remainders will be

$c_{n-1}, c_{n-2}, \ldots, c_1$, and the final quotient will be c_0.

Form the quotients

$$q_0^{(1)} = -\frac{a_1}{a_0}, \quad q_{1-k}^{(k)} = 0, \quad k = 2, 3, \ldots, n$$

$$e_{(1-k)}^{(k)} = \frac{a_{k+1}}{a_k}, \quad k = 1, 2, \ldots, n-1$$

Consider the elements thus generated as the first two rows of a QD scheme, and generate successive rows by

$$q_{i+1}^{(k)} = (e_i^{(k)} - e_{i+1}^{(k-1)}) + q_i^{(k)}$$

$$e_{i+1}^{(k)} = \frac{q_i^{(k+1)}}{q_{i+1}^{(k)}} e_i^{(k)}$$

where

$$e_i^{(0)} = e_i^{(n)} = 0, \quad i = 1, 2, \ldots$$

If the zeros z_k are distinct then

$$\lim_{i \to \infty} q_i^{(k)} = z_k$$

and

$$\lim_{i \to \infty} e_i^{(k)} = 0$$

If some of the zeros, z_k have the same modulus then the coefficients of the polynomial equation with these zeros can be

constructed from the $q_i^{(k)}$. Distinct zeros or groups of zeros with equal moduli will be separated by small $e_i^{(k)}$ ($e_i^{(k)} < .5 \times 10^{-2}$ was used in the computer program).

Suppose the polynomial equation has m zeros $z_{k+1}, a_{k+2}, \ldots, z_{k+m}$ with the same modulus. Consider the polynomials where

$$\lim_{i\to\infty} P_i^{(k+m)} = (z-z_{k+1})(z-z_{k+2})\ldots(z-z_{k+m}),$$

that is, the coefficients of $P_i^{(k+m)}$ tend for $i\to\infty$ to the coefficients of the polynomial equation with zeros $z_{k+1}, z_{k+2}, \ldots, z_{k+m}$ and leading coefficient 1.

The polynomials $P_i^{(k+m)}$ are constructed from the recurrence relations

$$P_i^{(k)}(z) = 1, \quad i = 0, 1, \ldots .$$

$$P_i^{(k+\ell)}(z) = zP_{i+1}^{(k+\ell-1)}(z) - q_i^{(k+\ell)}P_i^{(k+\ell-1)}(z),$$

$$\ell = 1, 2, \ldots, m; \; i = 0, 1, \ldots .$$

So

$$P_i^{(k+1)}(z) = z - q_i^{(k+1)}$$

and for $m = 2$ we have

$$P_i^{(k+2)}(z) = z(z-q_{i+1}^{(k+1)}) - q_i^{(k+2)}(z-q_i^{k+1}),$$

$$= z^2 - (q_{i+1}^{(k+1)}+q_i^{(k+2)})z + q_i^{(k+1)}q_i^{(k+2)} .$$

This gives a convenient means of obtaining approximations to coefficients of quadratic equations $z^2 + bz + c = 0$ with zeros z_{k+1} and z_{k+2},

$$\lim_{i\to\infty} -(q_{i+1}^{(k+1)} + q_i^{(k+2)}) = b$$

$$\lim_{i\to\infty} q_i^{(k+1)} q_i^{(k+2)} = c$$

These coefficients can readily be refined using Bairstow's algorithm (3) and then the roots can be obtained using the quadratic formula.

Most frequently the quadratic equation occurs in connection with complex conjugate zeros of real polynomials.

The coefficients of polynomial equations of higher degree can be obtained in a similar way. These coefficients would be only approximate so the roots from the resulting polynomial equations would be in error. There is no convenient way to solve polynomial equations, whose zeros have equal moduli, for degree greater then two unless we consider a method such as Muller's as presented in this paper. This is not done since more accurate results would be obtained by using Muller's algorithm on the original polynomial equation.

Operator Program

On the following pages are described the operator programs which go together to form the QD program. Each of the operators is

first defined and then the program is documented as a string of operators.

Rutishauser's Quotient-Difference (QD) Program--Operator Programming Definitions:

Π_0 Input the program and input data into the memory of the machine.

Comment: NC = the degree of the polynomial equation plus one.

KC = 0 is all the coefficients are real.

A(I) = the complex coefficients of the polynomial equation, (I, 1, . . . , NC).

A_1 Translate the input data into binary.

Comment: CT is the new origin to be used if any of the coefficients are zero. Note that the QD algorithm requires that all the coefficients be non zero.

ICT is the number of times we have changed origins.

NR is the number of roots that have been found.

NTYPE1 is the number of simple roots.

NTYPE2 is the number of approximations for quadratic equations.

$\mathcal{Z}_2$ CT = 2.;

ICT = 0;

NR = 0;

NTYPE1 = 0;

NTYPE2 = 0;

ONE = (1., 0.);

A_3 N = NC - 1;

Comment: Store the coefficients in a new array since they may be transformed to eliminate any zero terms.

β_4 AT(I) = A(I), (I=1, . . . , NC);

Comment: Test for zero coefficients.

P_5 |AT(I)| = 0.?, (I=1, . . . , NC);

Comment: If we have a zero coefficient and have not transformed the coefficients eight times, transform them again.

P_6 ICT $\leq$ 8?;

Π_7 Write: zero coefficients after eight transformations.

Comment: CT is the new origin.

A_8 CT = CT/2.;

NP1 = N + 1;

Comment: Transform the coefficients to eliminate any zero terms.

A_9 AT(I) = AT(I) + CT * AT(I-1), (I=2, . . . , NP1);

NP1 = NP1 - 1;

P_{10} NP1 $\geq$ 2?;

A_{11} ICT = ICT + 1;

Comment: Solve the polynomial using the QD algorithm.

E_{12}(QDIFF) CALL QDIFF(AT, NC, IT, Z, B, NTYPE1, NTYPE2);

Π_{13} Write: Output from QD algorithm, ... iterations.

Comment: Have any simple roots been found? ;

P_{14} NTYPE1 = 0? ;

Comment: If the coefficients have been transformed, transform the roots back.

P_{15} ICT = 0? ;

A_{16} Z(I) = Z(I) + CT, (I=1, . . . , NTYPE1);

Π_{17} Write: Approximations to simple roots: . . . .

Comment: Have any approximations for quadratic equations been found?

P_{18} NTYPE2 = 0? ;

Comment: Use Bairstow's algorithm to refine the approximations to the quadratic equations.

B(1, I) and B(2, I) are the coefficients of the quadratic equation. The leading coefficient is one.

$E_{19}(20, 35)$ (I=1, . . . , NTYPE2);

$\mathcal{Z}_{20}$ P = B(1, I);

Q = B(2, I);

E_{21}(BAIRST) CALL BAIRST(AT, NC, IT, P, Q);

Π_{22} Write: Quadratic approximations: . . . .

A_{23} NR = NR + 1;

Comment: IT is the number of iterations performed. Store this number in NI(NR) and print out later.

$\mathcal{Z}_{24}$ NI(NR) = IT;

Comment: Solve the quadratic equation using the quadratic formula.

E_{25}(QUAD) CALL QUAD(ONE, P, Q, Z1, Z2);

Comment: If the coefficients have been transformed, transform the roots back.

P_{26} ICT = 0? ;

A_{27} Z1 = Z1 + CT;

Comment: Evaluate the polynomial at Z1, (find the remainder).

E_{28}(POLY) REM(NR) = POLY(A, N, Z1);

Comment: Store the first root from the quadratic formula and go on to the second root.

$\mathcal{Z}_{29}$ RTZ(NR) = Z1;

A_{30} NR = NR + 1;

$\mathcal{Z}_{31}$ NI(NR) = IT;

P_{32} ICT = 0? ;

A_{33} Z2 = Z2 + CT;

E_{34}(POLY) REM(NR) = POLY(A, N, Z2);

$\mathcal{Z}_{35}$ RTZ(NR) = Z2;

Comment: If any simple roots were found use the Newton Raphson algorithm to refine them.

P_{36} NTYPE1 = 0? ;

E_{37}(38, 41) (I=1, . . . , NTYPE1);

A_{38} NR = NR + 1;

E_{39}(NEWTON) CALL NEWTON(A, N, IT, Z(I), REM(NR));

$\mathcal{Z}_{40}$ NI(NR) = IT;

$\mathcal{Z}_{41}$ RTZ(NR) = Z(I);

Π_{42} Output the coefficients, roots and remainders.

$\mathcal{L}_{43}$ Stop the machine.

Combining the preceding operators, the logical scheme of the QD program has the form:

$$\Pi_0\, A_1\, \mathcal{Z}_2\, A_3\, \mathcal{Z}_4\, \overset{11}{\urcorner} P_5 \underset{12}{\llcorner}\, P_6 \overset{8}{\lceil}\, \Pi_7 \overset{43}{\lceil}\, \overset{6}{\urcorner} A_8 \overset{10}{\urcorner} A_9\, P_{10} \overset{9}{\lceil}$$

$$A_{11} \overset{5}{\lceil}\, \underset{5}{\lrcorner} E_{12}\, \Pi_{13}\, P_{14} \overset{18}{\lceil}\, P_{15} \overset{17}{\lceil}\, A_{16} \overset{15}{\urcorner} \Pi_{17} \overset{14}{\urcorner} P_{18} \overset{36}{\lceil}$$

$$E_{19}\, \mathcal{Z}_{20}\, E_{21}\, \Pi_{22}\, A_{23}\, \mathcal{Z}_{24}\, E_{25}\, P_{26} \overset{28}{\lceil}\, A_{27} \overset{26}{\urcorner} E_{28}\, \mathcal{Z}_{29}$$

$$A_{30}\, \mathcal{Z}_{31}\, P_{32} \overset{34}{\lceil}\, A_{33} \overset{32}{\urcorner} E_{34}\, \mathcal{Z}_{35} \overset{18}{\urcorner} P_{36} \overset{42}{\lceil}\, E_{37}\, A_{38}\, E_{39}$$

$$\mathcal{Z}_{40}\, \mathcal{Z}_{41} \overset{36}{\urcorner} \Pi_{42} \overset{7}{\urcorner} \mathcal{L}_{43}\,.$$

E(QDIFF) SUBROUTINE QDIFF(A, NC, IT, Z, B, NTYPE1, NTYPE2);

Comment: Solution of the polynomial equation using the QD algorithm.

A(I) = the complex coefficients of the polynomial equation,

(I=1, . . . , NC).

NC = the degree of the polynomial equation plus one.

IT = the number of iterations for the roots.

Z(I) = approximations to simple roots, (I=1, 2, . . .).

B(I) = approximations to the coefficients of a quadratic equation, the leading coefficient is one, (I=1,2,...).

NTYPE1 = the number of simple root approximations.

NTYPE2 = the number of quadratic equation approximations.

3_0 IT = 0;

Comment: Set up the iteration limit.

A_1 LIMIT = 3 · NC;

P_2 If LIMIT < 20 set LIMIT = 20;

P_3 If LIMIT > 100 set LIMIT = 100;

A_4 N = NC - 1;

Comment: Compute the initial quotient and difference terms.

A_5 Q(1, 2) = - A(2)/A (1);

3_6 Q(1, K+1) = 0., (K=2, . . . , N);

A_7 E(1, K) = A(K+1)/A(K), (K=2, . . . , N);

Comment: Compute the quotient and difference terms until we reach the limit.

$E_8(9, 31)$ (I=2, . . . , 101);

3_9 NTYPE1 = 0;

NTYPE2 = 0;

E(I-1, 1) = (0., 0.);

E(I-1, NC) = (0., 0.);

E(I, NC) = (0., 0.);

A_{10} Q(I, K) = E(I-1, K) - E(I-1, K-1) + Q(I-1, K), (K=2, . . . , NC);

A_{11} E(I, K) = Q(I, K+1)/Q(I, K) • E(I-1, K), (K=2, . . . , N);

A_{12} IT = IT + 1;

Comment: Iterate again if the number of iterations, (IT) is less than the limit.

P_{13} IT < LIMIT? ;

Comment: Test for convergence. Start by testing for simple roots.

$\mathcal{Z}_{14}$ K = 1;

A_{15} K = K + 1;

Comment: Are the roots separated by a small E(I, K)?

P_{16} $|E(I, K)| \geq 5. \times 10^{-2}$? ;

Comment: We can not have division by zero.

P_{17} |Q(I, K)| = 0.? ;

Comment: Are successive root approximations close enough?

P_{18} $|(Q(I, K) - Q(I, K-1))/Q(I, K)| > 1.0 \times 10^{-2}$? ;

A_{19} NTYPE1 = NTYPE1 + 1;

Comment: Store the root approximation.

$\mathcal{Z}_{20}$ Z(NTYPE1) = Q(I, K);

Comment: Have all E(I, K) been tested?

P_{21} K ≤ NC? ;

Comment: Do we have approximations to a quadratic term?

P_{22} $|E(I, K+1)| \geq 5. \times 10^{-2}$? ;

Comment: Have these quadratic approximations converged?

A_{23} P1 = -Q(I-1, K) - Q(I-1, K+1);

P2 = -Q(I, K) - Q(I, K+1);

P_{24} |(P2-P1)/P2| > 1. x 10^{-2}? ;

A_{25} Q1 = Q(I-2, K)•Q(I-1, K+1);

Q2 = Q(I-1, K)•Q(I, K+1);

P_{26} |(Q2-Q1)/Q2| > 1. x 10^{-2}? ;

A_{27} NTYPE2 = NTYPE2 + 1;

Comment: Store the quadratic term approximations.

B_{28} B(1, NTYPE2) = P2;

B(2, NTYPE2) = Q2;

A_{29} K = K + 1;

Comment: Have all E(I, K) been tested? ;

P_{30} K $\leq$ NC? ;

Π_{31} Write: Limit of 100 iterations reached, any roots found will be printed.

Π_{32} Write: Three or more roots with equal moduli, the QD program can not find such roots, any other roots will be found.

A_{33} K = K + 2;

Comment: Have all E(I, K) been tested? ;

P_{34} K > N? ;

Comment: Do we have the roots separated? ;

P_{35} $|E(I,K)| < 5. \times 10^{-2}$?;

A_{36} K = K + 1;

Π_{37} Transfer to the calling program.

Combining the preceding operators, the logical scheme of subroutine QDIFF has the form:

$$\beta_0 A_1 P_2 P_3 A_4 A_5 \beta_6 A_7 E_8 \beta_9 A_{10} A_{11} A_{12} P_{13}^{\overset{31}{\lceil}}$$

$$\beta_{14} \overset{21,30,35}{\lceil} A_{15} P_{16}^{\overset{22}{\lceil}} P_{17}^{\overset{31}{\lceil}} P_{18}^{\overset{31}{\lceil}} A_{19} \beta_{20} P_{21}^{\underset{37}{\lfloor}};$$

$$\overset{16}{\rceil} P_{22}^{\overset{32}{\lceil}} A_{23} P_{24}^{\overset{31}{\lceil}} A_{25} P_{26}^{\overset{31}{\lceil}} A_{27} \beta_{28} A_{29} P_{30}^{\underset{37}{\lfloor}};$$

$$\overset{13,17,18,24,26}{\rceil} \Pi_{31}^{\overset{37}{\lceil}}; \overset{22}{\rceil} \Pi_{32} A_{33} \overset{36}{\rceil} P_{34}^{\overset{37}{\lceil}} P_{35}^{\overset{15}{\lceil}} A_{36}^{\overset{34}{\lceil}} \overset{21,30,31,34}{\rceil} \Pi_{37}.$$

E(BAIRST) SUBROUTINE BAIRST(A, NC, IT, P, Q), (Ref. 3)

Comment: A(I) = the complex coefficients of the polynomial equation,

(I=1, . . . , NC).

NC = the degree of the polynomial equation plus one.

IT = the number of iterations performed.

P and Q = the approximations to the quadratic term

$Z^2 + PZ + Q$.

B(I) and C(I) are arrays used for storing terms generated by the iteration scheme.

β_0 IT = 0;

B(1) = A(1);

A_1 IT = IT + 1;

B(2) = A(2) - P•B(1);

A_2 B(I) = A(I) - P•B(I-1) - Q•B(I-2), (I=3, . . . , NC);

$\mathcal{B}_3$ C(1) = B(1);

A_4 C(2) = B(2) - P•C(1);

N2 = NC - 2;

A_5 C(I) = B(I) - P•C(I-1) - Q•C(I-2), (I-3, . . . , N2);

A_6 C(NC-1) = -P•C(NC-2) - Q•C(NC-3);

$D = (C(NC-2))^2 - C(NC-1) \cdot C(NC-3)$;

$DP = B(NC-1) \cdot C(NC-2) - B(NC) \cdot C(NC-3)$;

$DQ = B(NC) \cdot C(NC-2) - B(NC-1) \cdot C(NC-1)$;

DELP = DP/D;

DELQ = DQ/D;

P = P + DELP;

Q = Q + DELQ;

Comment: Have we reached the iteration limit?

P_7 IT $\geq$ 100?;

Comment: Have the coefficients of the quadratic term $Z^2 + PZ + Q$ comverged?

P_8 $\left|\frac{DELP}{P}\right| > 1.0 \times 10^{-8}$ and $\left|\frac{DELQ}{Q}\right| > 1.0 \times 10^{-8}$?;

Π_9 Transfer to the calling program.

Combining the above operators, the logical scheme of subroutine BAIRST has the form:

$$\mathcal{Z}_0 \overset{8}{\urcorner} A_1\ A_2\ \mathcal{Z}_3\ A_4\ A_5\ A_6\ P_7 \overset{9}{\llcorner}\ P_8 \overset{1}{\llcorner} \overset{7}{\urcorner} \Pi_9 .$$

E(NEWTON) SUBROUTINE NEWTON(A, N, IT, Z1, PZ):

See operator programming definition for subroutine Newton of Lehmer's program, page 23.

E(POLY) FUNCTION POLY(A, N, Z):

See operator programming definitions for function Poly of Lehmer's program, page 25.

E(QUAD) SUBROUTINE QUAD(A, B, C, Z1, Z2):

See operator programming definitions for subroutine Quad of Lehmer's program, page 26.

The FORTRAN subprograms which go together to form the QD computer program are listed in Figure 3. Note that this program includes the QD algorithm, Bairstow's algorithm and the Newton-Raphson algorithm.

4. The Steepest Descent Algorithm

The Steepest Descent algorithm with Šiljak functions (7) is used to minimize a nonnegative function, the minimal values of which are zero and correspond to the zeros of the polynomial equation under

Figure 3. The QD FORTRAN IV program.

```
      PROGRAM INOUT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION A(16),AT(16),B(2,7),NI(15),REM(15),
     1RTZ(15),Z(16)
      COMPLEX A,AT,B,ONE,P,POLY,PZ,Q,REM,RTZ,Z,Z1,Z2
C
C     NC=NO. OF COEFFICIENTS.
C     KC=0 IF ALL COEFFICIENTS ARE REAL.
C
      CALL SECOND(TIME1)
      READ(5,1000)NCASES
      DO 100 ICASES=1,NCASES
      CALL SECOND(TSTART)
      WRITE(6,1050)ICASES
      READ(5,1000)NC,KC,(A(I),I=1,NC)
C
      CT=2.
      ICT=0
      N=NC-1
      NR=0
      NTYPE1=0
      NTYPE2=0
      ONE=(1.,0.)
C
      DO 5 I=1,NC
 5    AT(I)=A(I)
C
C     TEST FOR ZERO COEFFICIENTS.
 8    DO 10 I=1,NC
 10   IF(CABS(AT(I)).EQ.0.)GO TO 12
      GO TO 25
C
C     IF THE ZERO COEFFICIENTS ARE NOT ELIMINATED AFTER
C     EIGHT TRANSFORMATIONS, GO TO THE NEXT CASE.
C
 12   IF(ICT.LE.8) GO TO 15
      WRITE(6,1005) ICASES
      GO TO 100
C
C     NEW CENTER.
 15   CT=CT/2.
      NP1=N+1
C
C     TRANSFORM EQUATION TO ELIMINATE ZERO COEFFICIENTS.
 18   DO 20 I=2,NP1
 20   AT(I)=AT(I)+CT*AT(I-1)
      NP1=NP1-1
      IF(NP1.GE.2)GO TO 18
      ICT=ICT+1
      GO TO 8
C
C     AT=COEFFICIENTS OF TRANSFORMED EQUATION.
C     IT=NO. OF ITERATIONS.
C     NC=NO. OF COEFFICIENTS.
C     Z=APPROXIMATIONS FOR SIMPLE ROOTS.
C     B=APPROXIMATIONS FOR QUADRATIC TERM.
C     NTYPE1=NO. OF SIMPLE ROOTS.
C     NTYPE2=NO. OF APPROXINATIONS FOR QUADRATIC TERMS.
 25   CALL QDIFF(AT,NC,IT,Z,B,NTYPE1,NTYPE2)
      WRITE(6,1020)IT
      IF(NTYPE1.EQ.0)GO TO 35
C
C     IF THE EQUATION HAS BEEN TRANSFORMED,
C     TRANSFORM THE ROOTS BACK.
 28   IF(ICT.EQ.0)GO TO 31
      DO 30 I=1,NTYPE1
      Z(I)=Z(I)+CT
```

```
   30   CONTINUE
   31   WRITE(6,1025) (Z(I),I=1,NTYPE1)
C
   35   IF(NTYPE2.EQ.0)GO TO 45
        DO 40 I=1,NTYPE2
        P=B(1,I)
        Q=B(2,I)
C
C       USE BAIRSTOWS ALGORITHM TO GET CLOSER
C       APPROXIMATIONS TO THE ROOTS.
C       USE COEFFICIENTS OF TRANSFORMED EQUATION SINCE
C       QUADRATIC APPROXIMATIONS ARE FOR THIS EQUATION.
        CALL BAIRST(AT,NC,IT,P,Q)
        WRITE(6,1030) P,Q
        NR=NR+1
        NI(NR)=IT
        CALL QUAD(ONE,P,Q,Z1,Z2)
        IF(ICT.EQ.0)GO TO 37
        Z1=Z1+CT
   37   REM(NR)=POLY(A,N,Z1)
        RTZ(NR)=Z1
        NR=NR+1
        NI(NR)=IT
        IF(ICT.EQ.0)GO TO 39
        Z2=Z2+CT
   39   REM(NR)=POLY(A,N,Z2)
   40   RTZ(NR)=Z2
   45   IF(NTYPE1.EQ.0)GO TO 50
C
C       USE NEWTONS METHOD TO GET CLOSER ROOTS.
C       USE ORIGINAL COEFFICIENTS SINCE ROOTS WERE
C       TRANSFORMED BACK.
        DO 49 I=1,NTYPE1
        NR=NR+1
        CALL NEWTON(A,N,IT,Z(I),REM(NR))
        NI(NR)=IT
        RTZ(NR)=Z(I)
   49   CONTINUE
C
   50   WRITE(6,1010)ICASES,NC,(A(I),I=1,NC)
        WRITE(6,1015)NR,(RTZ(I),NI(I),REM(I),I=1,NR)
        CALL SECOND(TEND)
        TSEC=TEND-TSTART
        WRITE(6,1300)TSEC
  100   CONTINUE
        CALL SECOND(TIME2)
        TTIME=TIME2-TIME1
        WRITE(6,1310)TTIME
 1000   FORMAT(2I5/(8E10.0))
 1005   FORMAT(6H0CASE I3,
       1*, ZERO COEFFICIENTS AFTER 8 TRANSFORMATIONS.*)
 1010   FORMAT(1H1///////1H0,14X,*CASE*,I3,*,  *,I3,
       1* COEFFICIENTS*//(1H ,14X,2E20.12))
 1015   FORMAT(1H0/1H0,28X,I3,* ROOTS*,39X,*REMAINDERS*//
      1(1H ,14X,2E20.12,*  NI= *,I3,2E17.8))
 1020   FORMAT(1H0,14X,*OUTPUT FROM QD ALGORITHM,*,I4,
       1* ITERATIONS.*)
 1025   FORMAT(1H0,14X,*APPROXIMATIONS TO SIMPLE ROOTS*//
      1 (1H ,14X,2E20.12))
 1030   FORMAT(1H0,14X,*QUADRATIC APPROXIMATIONS*//
      1(1H ,14X,2E20.12))
 1050   FORMAT(1H1///////1H0,14X,*CASE*,I3)
 1300   FORMAT(1H0,//,15X,*EXECUTION TIME=*,F10.3,
       1* SECONDS*)
 1310   FORMAT(1H0,14X,*TOTAL EXECUTION TIME=*,F10.3,
       1* SECONDS*)
        END
```

```
      SUBROUTINE QDIFF(A,NC,IT,Z,B,NTYPE1,NTYPE2)
      DIMENSION A(16),B(2,7),E(100,16),Q(100,16),Z(16)
      COMPLEX A,B,E,P1,P2,Q,Q1,Q2,Z
C
C     A=COEFFICIENTS.
C     NC=NO. OF COEFFICIENTS.
C     IT=NO. OF ITERATIONS.
C     Z=APPROXIMATIONS FOR SIMPLE ROOTS.
C     B=APPROXIMATIONS FOR BAIRSTOWS QUADRATIC TERM.
C     NTYPE1=NO. OF SIMPLE ROOTS.
C     NTYPE2=NO. OF APPROXINATIONS FOR QUADRATIC TERMS.
C
      IT=0
      N=NC-1
      LIMIT=3*NC
      IF(LIMIT.LT.20)LIMIT=20
      IF(LIMIT.GT.90)LIMIT=90
      Q(1,2)=-A(2)/A(1)
      DO 5 K=2,N
      Q(1,K+1)=(0.,0.)
 5    E(1,K)=A(K+1)/A(K)
C
 12   DO 40 I=2,101
      NTYPE1=0
      NTYPE2=0
      E(I-1,1)=(0.,0.)
      E(I-1,NC)=(0.,0.)
      E(I,NC)=(0.,0.)
C
      DO 15 K=2,NC
 15   Q(I,K)=E(I-1,K)-E(I-1,K-1)+Q(I-1,K)
      DO 16 K=2,N
 16   E(I,K)=Q(I,K+1)/Q(I,K)*E(I-1,K)
      IT=IT+1
C
C     ITERATE AGAIN IF LESS THAN LIMIT.
      IF(IT.LT.LIMIT)GO TO 40
      K=1
 20   K=K+1
      IF(CABS(E(I,K)).GE.5.0E-2)GO TO 30
 22   IF(CABS(Q(I,K)).EQ.0.)GO TO 40
      IF(CABS((Q(I,K)-Q(I-1,K))/Q(I,K)).GT.1.0E-2)GO TO 40
      NTYPE1=NTYPE1+1
      Z(NTYPE1)=Q(I,K)
      IF(K.LT.NC)GO TO 20
      GO TO 400
C
 30   IF(CABS(E(I,K+1)).GE.5.0E-2)GO TO 350
      P1=-Q(I-1,K)-Q(I-1,K+1)
      P2=-Q(I,K)-Q(I,K+1)
      IF(CABS((P2-P1)/P2).GT.1.0E-2)GO TO 40
      Q1=Q(I-2,K)*Q(I-1,K+1)
      Q2=Q(I-1,K)*Q(I,K+1)
      IF(CABS((Q2-Q1)/Q2).GT.1.0E-2)GO TO 40
      NTYPE2=NTYPE2+1
      B(1,NTYPE2)=P2
      B(2,NTYPE2)=Q2
      K=K+1
      IF(K.LT.NC)GO TO 20
      GO TO 400
 40   CONTINUE
C
C     LIMIT OF 100 ITERATIONS REACHED,
C     ANY ROOTS FOUND WILL BE PRINTED.
      WRITE(6,1015)
      GO TO 400
```

```
C      THREE OR MORE ROOTS WITH EQUAL MODULI,
C      THE QD ALGORITHM CAN NOT FIND SUCH ROOTS,
C      ANY OTHER ROOTS WILL BE FOUND.
 350   WRITE(6,1030)
       K=K+2
 360   IF(K.GT.N)GO TO 400
       IF(CABS(E(I,K)).LT.5.0E-2)GO TO 20
       K=K+1
       GO TO 360

 400   RETURN
 1015  FORMAT(1H0,14X,*LIMIT OF 100 ITERATIONS REACHED*/
      115X,*ANY ROOTS FOUND WILL BE PRINTED.*)
 1030  FORMAT(1H0,14X,
      1*THREE OR MORE ROOTS WITH EQUAL MODULI.*/15X,
      2*THE QD PROGRAM CAN NOT FIND SUCH ROOTS,*/15X,
      3*ANY OTHER ROOTS WILL BE FOUND.*)
       END



       SUBROUTINE BAIRST(A,NC,IT,P,Q)
       DIMENSION A(16),B(16),C(15)
       COMPLEX A,B,C,P,Q,D,DP,DQ,DELP,DELQ
C
C      SUBROUTINE BAIRST REFINES THE APPROXIMATIONS TO A
C      QUADRATIC POLYNOMIAL, SEE HILDEBRAND, P.472-475.
C
       IT=0
       B(1)=A(1)
  5    IT=IT+1
       B(2)=A(2)-P*B(1)
       DO 10 I=3,NC
  10   B(I)=A(I)-P*B(I-1)-Q*B(I-2)
       C(1)=B(1)
       C(2)=B(2)-P*C(1)
       N2=NC-2
       DO 20 I=3,N2
  20   C(I)=B(I)-P*C(I-1)-Q*C(I-2)

C      NOTE  C(NC-1) HAS B(NC-1) SUBTRACTED FROM IT.
       C(NC-1)=-P*C(NC-2)-Q*C(NC-3)
       D=C(NC-2)*C(NC-2)-C(NC-1)*C(NC-3)
       DP=B(NC-1)*C(NC-2)-B(NC)*C(NC-3)
       DQ=B(NC)*C(NC-2)-B(NC-1)*C(NC-1)
       DELP=DP/D
       DELQ=DQ/D
       P=P+DELP
       Q=Q+DELQ
       IF(IT.GE.100)GO TO 400
       IF(CABS(DELP/P).GT.1.0E-8
      1.OR.CABS(DELQ/Q).GT.1.0E-8)GO TO 5
  400  RETURN
       END


       SUBROUTINE NEWTON(A,N,IT,Z1,PZ)
       DIMENSION A(16)
       COMPLEX A,DPZ,PZ,Z1,Z2

C      A=COEFFICIENTS.
C      N=DEGREE OF POLYNOMIAL.
C      IT=NO. OF ITERATIONS.
C      Z1=INDEPENDENT VARIABLE.
C      PZ=THE VALUE OF THE POLYNOMIAL AT Z=Z1.
C      DPZ=THE DERIVATIVE OF PZ.
```

```
      IT=0
 5    IT=IT+1
      PZ=A(1)
      DPZ=A(1)
      DO 10 I=2,N
      PZ=A(I)+Z1*PZ
 10   DPZ=PZ+Z1*DPZ
      PZ=A(N+1)+Z1*PZ

      IF(DPZ.EQ.0.)GO TO 20
      Z2=Z1-PZ/DPZ
      IF(CABS(PZ).LT.1.0E-20)GO TO 100

C     TEST FOR 0. DENOMINATOR.
      IF(Z1.EQ.0.)GO TO 20

C     TEST FOR CONVERGENCE.
      IF(CABS((Z2-Z1)/Z1).LT.1.0E-10)GO TO 100
      IF(IT.EQ.100)GO TO 100
 20   Z1=Z2
      GO TO 5
 100  Z1=Z2
      RETURN
      END


      COMPLEX FUNCTION POLY(A,N,Z)
      DIMENSION A(16)
      COMPLEX A,Z
C
C     A=COEFFICIENTS OF POLYNOMIAL.
C     N=DEGREE OF POLYNOMIAL.
C     Z=VALUE AT WHICH POLYNOMIAL IS EVALUATED.
C     POLY=POLYNOMIAL EVALUATED AT Z.
C
      POLY=A(1)
      DO 20 I=1,N
 20   POLY=Z*POLY+A(I+1)
      RETURN
      END


      SUBROUTINE QUAD(A,B,C,Z1,Z2)
      COMPLEX A,B,C,DISC,Z1,Z2
      DISC=CSQRT(B*B-4.*A*C)
      Z1=(-B+DISC)/(2.*A)
      Z2=(-B-DISC)/(2.*A)
      RETURN
      END
```

investigation.

We are concerned with the equation

(1) $$f(z) = u + iv = 0,$$

where $f(z)$ is an entire function of the complex variable z, where

(2) $$z = x + iy.$$

Now we form the function

(3) $$F(x,y) = u^2 + v^2 .$$

$F = F(x, y)$ is a function having the property that the zeros of F are the zeros of $f(z)$. In fact these zeros are the only minima of F, (10). Also note that $F \geq 0$ for all z and that $\partial F/\partial x$ and $\partial F/\partial y$ exist.

The level lines of the function $F(x, y)$ are the intersections of the surface $w = F(x, y)$ with the planes parallel to the x, y-plane (w is the altitude above the x, y-plane). At an arbitrary point $P(x, y)$ the gradient of F has the components

(4) $$F_x = \frac{\partial F}{\partial x}, \quad F_y = \frac{\partial F}{\partial y}.$$

The gradient vector ∇F is orthogonal to the level line through P and points in the direction of increasing values of F. We must

move in the opposite direction, the direction of steepest descent. The differential equations of the orthogonal trajectory O to the level lines of $F(x, y)$ are

$$\frac{dx}{dt} = -p(t)\frac{\partial F}{\partial x}, \quad \frac{dy}{dt} = -p(t)\frac{\partial F}{\partial y}. \tag{5}$$

Here p is a proportionality factor and t is a parameter along the curve O. The function $p(t)$ must be positive and is chosen to minimize the function F along the direction of steepest descent. We will now consider the calculation of $p(t)$.

In the neighborhood of a zero of $f(z)$, the higher order terms of a series expansion for u and v are negligible so we have:

$$-u \simeq \frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y, \quad -v \simeq \frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y, \tag{6}$$

where Δx and Δy are the distances in the x and y directions from a point in the neighborhood of the zero of $f(z)$ to the zero of $f(z)$. Using the Cauchy-Riemann equations and solving Equation (6) for Δx and Δy yield

$$\Delta x \simeq \frac{-u\frac{\partial u}{\partial x} - v\frac{\partial v}{\partial x}}{(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial x})^2}, \quad \Delta y \simeq \frac{u\frac{\partial v}{\partial x} - v\frac{\partial u}{\partial x}}{(\frac{\partial u}{\partial x})^2 + (\frac{\partial v}{\partial x})^2} \tag{7}$$

From these equations we see that

$$(8) \qquad p(t) = \frac{0.5}{(\frac{\partial u}{\partial x})^2+(\frac{\partial v}{\partial x})^2} > 0.$$

If $F = u^2 + v^2$ in Equation (3) is greater than the value previously calculated, replace the current Δx and Δy values by $.75\Delta x$ and $.75\Delta y$ until a value of F is found that is smaller than for the preceding iteration.

Operator Program

On the following pages are described the operator programs which go together to form the Steepest Descent program. Each of the operators is first defined and then the program is documented as a string of operators.

The Steepest Descent Program--Operator Programming Definitions:

Π_0 Input the program and data into the memory of the machine.

Comment: A(I) = the complex coefficients of the polynomial equation, (I=1, . . . , N+1).

N = the degree of the polynomial equation.

NC = the degree of the polynomial equation plus one.

KC = 0 if the coefficients are all real.

NI(I) = the number of iterations per root, (I=1, . . . , N).

NR = the number of roots found.

RTZ(I) = the roots of the polynomial equation,

(I=1, . . . , NR).

REM(I) = the remainders from evaluating the polynomial equation at the roots, (I=1, . . . , NR).

A_1 Translate the input data into binary.

A_2 N = NC - 1;

E_3(STEDES) CALL STEDES(A, N, KC, NI, NR, RTZ, REM);

Π_4 Output the coefficients, roots and remainders.

$\mathscr{L}_5$ Stop the machine.

Combining the above operators, the logical scheme of the program has the form:

$$\Pi_0 A_1 A_2 E_3 \Pi_4 \mathscr{L}_5 .$$

E(STEDES) SUBROUTINE STEDES(A, N, KC, NI, NR, RTZ, REM);

Comment: A(I) = the complex coefficients of the polynomial equation,

(I=1, . . . , N+1).

N = the degree of the polynomial equation.

KC = 0 if the coefficients are all real.

NI(I) = the number of iterations per root, (I=1, . . . , NR).

NR = the number of roots found.

RTZ(I) = the roots of the polynomial equation,

(I=1, . . . , NR).

REM(I) = the remainders from evaluating the polynomial equations at the roots, (I=1, . . . , NR).

DZ is the correction to the root.

We now initialize some of the terms.

$\mathcal{Z}_0$ DZ = 0.;

NR = 0;

A_1 NC = N + 1;

Comment: B(I) is used for storing the coefficients of the reduced polynomial (polynomial with the roots found removed).

$\mathcal{Z}_2$ B(I) = A(I), (I=1, . . . , NC);

Comment: Test if z = 0 is a root.

P_3 |A(NC)| ≠ 0.?;

Comment: z = 0 is a root, store it and test again.

A_4 NR = NR + 1;

$\mathcal{Z}_5$ NI(NR) = 0;

A_6 NC = NC - 1;

$\mathcal{Z}_7$ RTZ(NR) = 0.;

REM(NR) = 0.;

Comment: Have all the roots been found?

P_8 NR - N = 0?;

Comment: Initialize starting values. IT is the number of iterations for the root. L = 1 if the complex conjugate has not been found. L = 2 if the complex conjugate has been

found.

3_9 IT = 0;

L = 1;

Z = (1., 1.);

ZN = Z;

ZR = 1.;

ZI = 1.;

Comment: This is the start of the iteration loop.

A_{10} IT = IT + 1;

3_{11} X(1) = 1.;

Z(2) = ZR;

Y(1) = 0.;

Y(2) = ZI;

Comment: Split the coefficients into their real and imaginary parts.

3_{12} AR(I) = Real B(I), (I=1, . . . , NC);

AI(I) = Imaginary B(I);

Comment: Calculate the real and imaginary parts of f(z) and their partial derivatives in terms of the Šiljak functions X(I) and Y(I), (see Reference 7).

A_{13} U = AR(NC) + AR(NC-1)·ZR - AI(NC-1)·ZI;

V = AI(NC) + AR(NC-1)·ZI + AI(NC-1)·ZR;

3_{14} PU = AR(NC-1);

PV = AI(NC-1);

A_{15} ZR2 = ZR + ZR;

ZS = $ZR^2 + ZI^2$;

$E_{16}(17, 18)$ (I=3, . . . , NC);

A_{17} X(I) = ZR2 • X(I-1) - ZS • X(I-2);

Y(I) = ZR2 • Y(I-1) - ZS • Y(I-2);

J = NC + 1 - I;

U = U + AR(J) • X(I) - AI(J) • Y(I);

V = V + AR(J) • Y(I) + AI(J) • X(I);

PU = PU + (I-1) • (AR(J) • X(I-1)-AI(J) • Y(I-1));

PV = PV + (I-1) • (AR(J) • Y(I-1)+AI(J) • X(I-1));

P_{18} I < NC? ;

Comment: Store the value of the polynomial f(z) evaluated at the new root approximation, z.

3_{19} PZ = (U, V);

A_{20} PS = $PU^2 + PV^2$;

FN = $U^2 + V^2$;

Comment: Is the square of the value of the polynomial zero?

P_{21} FN = 0? ;

Comment: Is this the first iteration performed?

P_{22} IT = 1? ;

Comment: F is the square of the value of the polynomial at the previous root approximation.

P_{23} FN < F? ;

```
Comment: At this point in the program we have FN > F which means
         the delta z increment for z was too large.  Try a
         smaller delta z.
A24      DZR = .75 DZR;
         DZI = .75 DZI;
Comment: Set IJ = 1 to indicate that delta z has been altered.
3 25     IJ = 1;
Comment: Set IJ = 0 to indicate that delta z has not been altered.
3 26     IJ = 0;
Comment: Compute the delta z values.
A27      DZR = -(U•PU+V•PV)/PS;
         DZI = (U•PV-V•PU)/PS;
Comment: Save the previous iteration's values.
3 28     ZRS = ZR;
         ZIS = ZI;
         F = FN;
Comment: Compute the new z terms.  ZN is the new root approxi-
         mation.
A29      ZR = ZRS + DZR;
         ZI = ZIS + DZI;
3 30     DZ = (DZR, DZI);
         ZN = (ZR, ZI);
Comment: If this is the first iteration do not test for convergence.
```

P_{31} IT = 1? ;

Comment: If delta z has been altered do not test for convergence yet, iterate once.

P_{32} IJ = 1? ;

Comment: Test for convergence.

P_{33} $|PZ| < 1.0 \times 10^{-20}$? ;

P_{34} Z = 0? ;

P_{35} $|DZ/Z| < 1.0 \times 10^{-10}$? ;

Comment: Have we reached the maximum number of iterations? ;

P_{36} IT $\geq$ 200? ;

Comment: Store the new root approximation and iterate again.

$\mathcal{J}_{37}$ Z = ZN;

Comment: We have a new root, store it and compute the value of the polynomial at the new root.

A_{38} NR = NR + 1;

E_{39}(POLY) REM(NR) = POLY(A, N, ZN);

$\mathcal{J}_{40}$ NI(NR) = IT;

RTZ(NR) = ZN;

Comment: If the remainder when evaluating the polynomial at this new root is greater than or equal to one we print out the results found and stop the program. The program is stopped since succeeding roots are found using the coefficients of the reduced polynomial (the root is

divided out using synthetic division). These coefficients would have too much error to yield reasonably correct roots.

P_{41} $|REM(NR)| \geq 1.?$;

Comment: Have we found all the roots?

P_{42} $NR \geq N?$;

Comment: Calculate the coefficients of the reduced polynomial using synthetic division.

A_{43} NC = NC - 1;

A_{44} B(I) = B(I) + ZN • B(I-1), (I=2, . . . , NC);

Comment: If we have a linear equation transfer and solve directly for the root.

P_{45} NC = 2? ;

Comment: If the complex conjugate has been found transfer and iterate for the next root.

P_{46} L = 2? ;

Comment: If the coefficients are not all real iterate for the next root.

P_{47} KC ≠ 0? ;

Comment: If the imaginary part of the root is zero transfer and iterate for the next root.

P_{48} Imaginary RTZ(NR) = 0? ;

Comment: If the imaginary part of the root is small compared to the real part transfer and iterate for the next root.

P_{49} $\left| \frac{\text{real RTZ(NR)}}{\text{imagainary RTZ(NR)}} \right| > 1.0 \times 10^{2}$?;

Comment: Compute the complex conjugate.

β_{50} L = 2;

IT = 0;

ZN = $\overline{ZN}$;

Comment: Solve the linear equation.

β_{51} IT = 0;

A_{52} ZN = -B(2)/B(1);

Π_{53} Transfer to the calling program.

Combining the above operators, the logical scheme for subroutine STEDES has the form:

$$\beta_0 A_1 \beta_2 \underset{8}{\lfloor} P_3 \lceil^{9} A_4 \beta_5 A_6 \beta_7 P_8 \underset{3}{\lfloor}^{53} ; \overset{3,46,47,48,49}{\lceil} \beta_9$$

$$\overset{37}{A_{10}} \beta_{11} \beta_{12} A_{13} \beta_{14} A_{15} \overset{18}{\rceil} E_{16} A_{17} P_{18} \lceil^{16} \beta_{19} A_{20}$$

$$P_{21} \lceil^{38} P_{22} \lceil^{26} P_{23} \lceil^{26} A_{24} \beta_{25} \lceil^{29} \overset{22,23}{\lceil} \beta_{26} A_{27} \beta_{28}$$

$$\overset{25}{\rceil} A_{29} \beta_{30} P_{31} \lceil^{36} P_{32} \lceil^{36} P_{33} \lceil^{38} P_{34} \lceil^{36} P_{35} \lceil^{38} \overset{31,32,34}{\lceil} P_{36} \lceil^{38}$$

$$\beta_{37} \lceil^{10} \overset{21,33,35,36,51,52}{\lceil} A_{38} E_{39} \beta_{40} P_{41} \lceil^{53} P_{42} \lceil^{53}$$

$$A_{43} A_{44} P_{45} \lceil^{50} P_{46} \lceil^{9} P_{47} \lceil^{9} P_{48} \lceil^{9} P_{49} \lceil^{9} \beta_{50} \lceil^{38} ;$$

$$\mathcal{Z}_{51}\, A_{52}\left\lceil \frac{38}{\quad} ;\ \frac{8,41,42}{\quad}\right\rceil \Pi_{53} .$$

E(POLY) FUNCTION POLY(A, N, Z);

See operator programming definitions for function Poly of Lehmer's program, page 25.

The FORTRAN subprograms which go together to form the Steepest Descent computer program are listed in Figure 4.

Figure 4. The Steepest Descent FORTRAN IV program.

```
      PROGRAM INOUT(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      DIMENSION A(16),NI(15),REM(15),RTZ(15)
      COMPLEX A,REM,RTZ

C     KC=0 IF THE COEFFICIENTS ARE ALL REAL.
C
      CALL SECOND(TIME1)
      READ(5,1000)NCASES
      DO 100 ICASES=1,NCASES
      CALL SECOND(TSTART)
      READ(5,1000)NC,KC,(A(I),I=1,NC)
      N=NC-1
      CALL STEDES(A,N,KC,NI,NR,RTZ,REM)
      WRITE(6,1010)ICASES,NC,(A(I),I=1,NC)
      WRITE(6,1020)NR,(RTZ(I),NI(I),REM(I),I=1,NR)
      CALL SECOND(TEND)
      TSEC=TEND-TSTART
      WRITE(6,1300)TSEC
  100 CONTINUE
      CALL SECOND(TIME2)
      TTIME=TIME2-TIME1
      WRITE(6,1310)TTIME
 1000 FORMAT(2I5/(8F10.0))
 1010 FORMAT(1H1///////1H0,14X,*CASE*,I3,*,  *,I3,
     1* COEFFICIENTS*//(1H ,14X,2E20.12))
 1020 FORMAT(1H0/1H0,28X,I3,* ROOTS*,39X,*REMAINDERS*//
     1(1H ,14X,2E20.12,*  NI= *,I3,2E17.8))
 1300 FORMAT(1H0,//,15X,*EXECUTION TIME=*,F10.3,* SECONDS*)
 1310 FORMAT(1H0,14X,*TOTAL EXECUTION TIME=*,F10.3,
     1* SECONDS*)
      END


      SUBROUTINE STEDES(A,N,KC,NI,NR,RTZ,REM)
      DIMENSION A(16),NI(15),REM(15),RTZ(15)
     1,AR(15),AI(15),B(16),X(16),Y(16)
      COMPLEX A,B,DZ,PZ,RTZ,REM,Z,ZN
     1,POLY
C
      DZ=0.
      NC=N+1
      NR=0
C
      DO 2 I=1,NC
    2 B(I)=A(I)
    5 IF(CABS(A(NC)).NE.0.)GO TO 10
      NR=NR+1
      NI(NR)=0
      NC=NC-1
      RTZ(NR)=(0.,0.)
      REM(NR)=(0.,0.)
      IF(NR-N)5,1000,5
   10 IT=0
      L=1
      Z=(1.,1.)
      ZN=Z
      ZR=1.
      ZI=1.
   20 IT=IT+1
      X(1)=1.
      X(2)=ZR
      Y(1)=0.
      Y(2)=ZI
```

```
      DO 28 I=1,NC
      AR(I)=REAL(B(I))
 28   AI(I)=AIMAG(B(I))
      NCM1=NC-1
      U=AR(NC)+AR(NCM1)*ZR-AI(NCM1)*ZI
      V=AI(NC)+AR(NCM1)*ZI+AI(NCM1)*ZR
      PU=AR(NCM1)
      PV=AI(NCM1)
      ZR2=ZR+ZR
      ZS=ZR*ZR+ZI*ZI
C
 30   DO 35 I=3,NC
      X(I)=ZR2*X(I-1)-ZS*X(I-2)
      Y(I)=ZR2*Y(I-1)-ZS*Y(I-2)
      J=NC+1-I
      U=U+AR(J)*X(I)-AI(J)*Y(I)
      V=V+AR(J)*Y(I)+AI(J)*X(I)
      PU=PU+(I-1)*(AR(J)*X(I-1)-AI(J)*Y(I-1))
 35   PV=PV+(I-1)*(AR(J)*Y(I-1)+AI(J)*X(I-1))
      PZ=CMPLX(U,V)
C
      PS=PU*PU+PV*PV
      FN=U*U+V*V
      IF(FN.EQ.0.)GO TO 100
      IF(IT.EQ.1)GO TO 40
      IF(FN.LT.F)GO TO 40
      DZR=.75*DZR
      DZI=.75*DZI
      IJ=1
      GO TO 45
 40   IJ=0
      DZR=-(U*PU+V*PV)/PS
      DZI= (U*PV-V*PU)/PS
      ZRS=ZR
      ZIS=ZI
      F=FN
 45   ZR=ZRS+DZR
      ZI=ZIS+DZI
      DZ=CMPLX(DZR,DZI)
      ZN=CMPLX(ZR,ZI)
      IF(IT.EQ.1)GO TO 70
      IF(IJ.EQ.1)GO TO 70
 50   IF(CABS(PZ).LT.1.0E-20)GO TO 100
      IF(Z .EQ.0.)GO TO 70
      IF(CABS(DZ/Z).LT.1.0E-10)GO TO 100
 70   IF(IT.GE.200)GO TO 100
      Z =ZN
      GO TO 20
C
 100  NR=NR+1
      REM(NR)=POLY(A,N,ZN)
      NI(NR)=IT
      RTZ(NR)=ZN
      IF(CABS(REM(NR)).GE.1.)GO TO 1000
      IF(NR.GE.N)GO TO 1000
      NC=NC-1
C
C     CALCULATE COEFFICIENTS OF REDUCED EQUATION.
      DO 105 I=2,NC
 105  B(I)=B(I)+ZN*B(I-1)
      IF(NC.EQ.2)GO TO 110
      IF(L.EQ.2)GO TO 10
      IF(KC.NE.0)GO TO 10
      IF(AIMAG(RTZ(NR)).EQ.0.)GO TO 10
      IF(ABS(REAL(RTZ(NR))/AIMAG(RTZ(NR))).GT.1.0E-2
     1GO TO 10
      L=2
      IT=0
```

```
      ZN=CONJG(ZN)
      GO TO 100
  110 IT=0
      ZN=-B(2)/B(1)
      GO TO 100
C
 1000 RETURN
      END


      COMPLEX FUNCTION POLY(A,N,Z)
      DIMENSION A(16)
      COMPLEX A,Z
C
C     A=COEFFICIENTS OF POLYNOMIAL.
C     N=DEGREE OF POLYNOMIAL.
C     Z=VALUE AT WHICH POLYNOMIAL IS EVALUATED.
C     POLY=POLYNOMIAL EVALUATED AT Z.
C
      POLY=A(1)
      DO 20 I=1,N
   20 POLY=Z*POLY+A(I+1)
      RETURN
      END
```

III. DISCUSSION OF PROGRAM RESULTS

The four computer programs, (Lehmer's, Muller's, Rutishauser's QD and the Steepest Descent) were used to solve numerous polynomial equations. The results from solving the nine test cases from Milne's "Numerical Calculus" (6) which are listed in the Introduction and again in Chapter IV are discussed first. Chapter IV also contains results (computer output) from solving these nine polynomial equations using the four computer programs. Results from solving the polynomial equation $x^n + x = 1$ $(n=3, 5, \ldots, 101)$ will be discussed second. When discussing $x^n + x = 1$, the case number is $n + 1$, i.e. $x^3 + x = 1$ is referred to as Case 4.

The execution times were significantly different as well as the number of roots that could be found by the different methods. The methods will be discussed in order of results, the most satisfactory first.

1. Solution of Nine Test Cases

Muller's Program

Muller's program converged in all nine cases, finding the 34 roots in 0.275 seconds. Of the 34 roots found three were complex conjugates and were accepted as roots without iterating, so 31 distinct roots were found. This gives an average time of 0.00887

seconds for each root. The roots found by Muller's program had remainders whose absolute values varied from 10^{-10} to 0. These roots were used as a check on the roots found by the other programs since their remainders are as small as or smaller than those found by the other methods.

Muller's program required 101 FORTRAN IV statements.

The Steepest Descent Program

The Steepest Descent program converged in all nine cases, finding the 34 roots in 0.308 seconds. When a root was found, the degree of the equation was reduced using synthetic division, linear equations that resulted were solved explicitly. The occurrence of linear equations and complex conjugate roots made it necessary to iterate for only 23 roots. This gives an average time of 0.01339 seconds for a root. These roots agreed with those from Muller's program in all 13 places in some cases and in only 8 places in other cases. The absolute values of the remainders varied from 10^{-10} to 0.

The Steepest Descent program required 128 FORTRAN IV statements.

Lehmer's Program

Lehmer's program is discussed next since it found more roots than the QD program although it is slower than the QD program and

the preceding programs.

Lehmer's program converged in all nine cases, finding the 34 roots in 0.361 seconds. Sixteen of these were distinct roots which gives an average time of 0.02256 seconds for each root. These roots agreed with those from Muller's program in all 13 places in some cases and in 10 places in other cases. The absolute values of the remainders varied from 10^{-10} to 0.

The 16 roots mentioned in the preceding paragraph refer to the number of roots iterated on by Lehmer's scheme. When a root was found, the degree of the equation was reduced using synthetic division. Whenever the reduced equation was of degree two or one, Lehmer's iteration scheme was stopped. The remaining roots were obtained by using the quadratic formula or by solving the linear equation for the root. The linear equation occurred whenever a root and its complex conjugate were removed from a third degree equation.

Lehmer's program required 223 FORTRAN IV statements.

The QD Program

Rutishauser's QD algorithm is restricted in that there is no simple method for solving the equation that results when three or more roots have equal moduli (see p. 53). When only two roots have equal moduli, the quadratic factor that is found by the QD program is conveniently refined in Bairstow's method and solved by the quadratic

formula. Simple roots are refined by Newton's algorithm with the derivative, f'(z), computed by iterated synthetic division.

Rutishauser's QD program found all the roots in seven of the nine cases. In the remaining two cases the algorithm was only able to separate out one root in each case. In Case 7 three roots vary by only one unit in the third digit so could not be separated out. In Case 8 three roots were the same to three digits and could not be separated out. The remaining 28 roots were found in 0.314 seconds which gives an average time of 0.01121 seconds for each root. These roots agreed with those from Muller's program in all 13 places in some cases and in 11 places in other cases. The absolute values of the remainders varied from 10^{-9} to 0.

The QD program requires 208 FORTRAN IV statements.

Table 2. Results from solution of the nine test cases.

Program	Number of roots found	Average time seconds/root	No. of places of agreement with Muller's program max. of 13	Absolute values of the remainders
Muller's	34	.00887	---	10^{-10} to 0
Steepest Descent	34	.01339	8 to 13	10^{-10} to 0
Lehmer's	34	.02256	10 to 13	10^{-10} to 0
QD	28	.01121	11 to 13	10^{-9} to 0

2. Solution of $x^n + x = 1$

Muller's Program

Muller's program converged in all cases, finding the 2,600 roots in 60.368 seconds. This gives an average time of .02321 seconds for each root. Complex conjugates were accepted as roots without iterating since the coefficients are all real. Since about half of the roots are complex conjugates, the above time should be doubled for arbitrary equations with no complex conjugate roots. The roots from Muller's program had the smallest remainders, (the absolute values varied from 10^{-12} to 0) so were used as a check on the roots found by the other programs.

The Steepest Descent Program

The Steepest Descent program also converged in all cases, finding the 2,600 roots in 159.144 seconds, an average of .06121 seconds for each root. As in the discussion of the results from Muller's algorithm, this time should be doubled for arbitrary equations with no complex conjugate roots. These roots agreed with those from Muller's algorithm in all 13 places for some roots and in only 9 places for other roots. The absolute values of the remainders varied from 10^{-8} to 0. As would be expected the error increases as the degree of the polynomial equation increases.

Lehmer's Program

Lehmer's program converged on the roots in Cases 4 to 52 in 64.589 seconds. There are 675 roots and hence an average time for each root is .09569 seconds. As in the discussion of the previous programs, about half of the roots are complex conjugates so the above time should be doubled for arbitrary equations with no complex conjugate roots. These roots agreed with those from Muller's method in all 13 places for some roots and in only 6 places for other roots. The absolute values of the remainders varied from 10^{-6} to 0.

Since the time required to find these roots was relatively high, the next cases investigated were Cases 72 and 102. In Case 72 all 71 roots were found and in Case 102 only 78 roots were found. The solution time was 133.402 seconds for an average of .89531 seconds for each root, about 40 times longer than required by Muller's program. This time should be doubled for the general case since about half the roots are complex conjugates. These roots agreed with those from Muller's program in all 13 places for some roots and in only one place for other roots. The absolute values of the remainders varied from 1.67 to 10^{-15}.

A root that is found approximately by Lehmer's algorithm is refined in Newton's method since Newton's method converges more rapidly. If after this refinement a root still had a remainder greater

than or equal to one the solution of the problem was stopped. A root not found to a high degree of accuracy caused all subsequent roots to be in error since they were found from the reduced equation. The reduced equation was used to avoid converging on the same root when the roots were close.

In Case 102 the root approximations found by Lehmer's algorithm were not accurate enough for Newton's method to converge so the solution of the problem was stopped. The number of iterations allowed in Newton's method was increased from 100 to 300 but convergence still could not be obtained.

The convergence criterion in Lehmer's algorithm is the distance between annulus centers, (the centers being the root approximations).

$$\left| \frac{CT-CT_0}{CT} \right| < \epsilon$$

where CT is the center of the current annulus that contains a root and CT_0 is the center of the previous annulus that contained a root. For $\epsilon = 1.2$ all the roots were found in Cases 4 through 12 but only two roots were found in Cases 14 through 52. $\epsilon = 1.0$ was small enough to force convergence of all roots in Cases 4 through 52. ϵ was varied from 0.17 to 0.05 for Cases 72 and 102. With $\epsilon = 0.17$ only four roots were found in Cases 72 and 102. With $\epsilon = 0.15$, 71 roots

were found in Case 72 and 78 roots were found in Case 102. With $\epsilon = 0.12$, 36 roots were found in Case 72 and 30 roots were found in Case 102. With $\epsilon = 0.05$, 46 roots were found in Case 72 and 24 roots were found in Case 102.

These results indicate that if ϵ is not small enough then the roots will not be accurate and the succeeding roots will be in error and if ϵ is too small the accuracy is unatttainable due to rounding errors that result in the computer operations.

The QD Program

The QD program is restricted in that there is no simple method for solving the equation that results when three or more roots have equal moduli, (see p. 53).

The QD program found all the roots in Cases 4 through 14. Three or more roots with equal moduli were detected in Cases 16, 18, 24, 26, 28, 30, 32 and 34 so the particular roots could not be found. As a result only 257 roots were found out of a possible 288. The execution time was 7.886 seconds, an average time of .03068 seconds for each root. The finding of complex conjugates is included in this time.

These roots agreed with those from Muller's program in all 13 places for some roots. In a few instances there was no agreement since Bairstow's subprogram did not converge. This can be

attributed to the initial approximations for the QD subprogram not being good enough. More iterations by the QD subprogram helped in some cases. This lack of agreement was apparent in the large absolute values of the remainders, 10^{5} to 10 as compared to 10^{-9} to 10^{-15} for good roots.

An infinite operand (operand > 10^{322}) was generated in subroutine BAIRST during the solution of Case 36 and stopped the program. Scaling was added to subroutine BAIRST which increased the number of cases solved from 32 to the present value of 34 before the infinite operand was generated. The infinite operand can be attributed to the many multiplications that are necessary to perform during the iterated synthetic division.

Table 3. Results from solving $x^{n} + x = 1$, (n=3,5,...,101).

Program	Number of roots found	Average time seconds/root	No. of places of agreement with Muller's program max. of 13	Absolute values of the remainders
Muller's	2,600	.04642	---	10^{-12} to 0
Steepest Descent	2,600	.12242	9 to 13	10^{-8} to 0
Lehmer's	675	.19137	6 to 13	10^{-6} to 0
QD	257	.03068*	0 to 13	10^{5} to 0

* This small time does not mean that the method is the fastest. It occurred because of the low degree of the equations that were solved as compared to the other methods.

IV. COMPUTER PROGRAM TEST RESULTS

1. Solution of Nine Test Cases

NI on the output refers to the number of iterations. For Muller's program NI is the number of iterations performed by the program for the particular root. This is also the case for the Steepest Descent program. The iterations performed by Lehmer's subprogram are listed separately. For Lehmer's program NI refers to the number of iterations performed by Newton's subprogram in refining the roots found by Lehmer's subprogram. The QD subprogram performed a minimum of 20 iterations before turning the root approximations over to Newton's or Baistow's subprogram for final refinement.

2. Solution of $x^{21} + x = 1$

NI on the output refers to the number of iterations. For Muller's program NI is the number of iterations performed by the program for the particular root. This is also the case for the Steepest Descent program. The iterations performed by Lehmer's subprogram are listed separately. For Lehmer's program NI refers to the number of iterations performed by Newton's subprogram in refining the roots found by Lehmer's subprogram. The QD subprogram performed 78 iterations before turning the root approximations over to Newton's or Bairstow's subprogram for final refinement.

Table 4. Muller's program, solution of nine test cases.

Case 1, 4 Coefficients

1.000000000000E+00 -0.
-0. -0.
-1.000000000000E+00 -0.
-4.000000000000E+00 -0.

3 Roots				Remainders	
1.796321903259E+00	0.	NI=	7	-2.84217094E-14	0.
-8.981609516297E-01	-1.191670795605E+00	NI=	2	0.	2.13162821E-14
-8.981609516297E-01	1.191670795605E+00	NI=	0	0.	-2.13162821E-14

Execution time = .022 seconds

Case 2, 5 Coefficients

1.000000000000E+00 -0.
-2.037900000000E+00 -0.
-1.542450000000E+01 -0.
1.566960000000E+01 -0.
3.549360000000E+01 -0.

4 Roots				Remainders	
-1.201998596673E+00	0.	NI=	6	2.27373675E-13	0.
2.124387030181E+00	0.	NI=	6	2.27373675E-13	0.
-3.211994374397E+00	0.	NI=	2	-9.09494702E-13	0.
4.327505940890E+00	0.	NI=	2	-2.50111043E-12	0.

Execution time = .028 seconds

Case 3, 5 Coefficients

1.000000000000E+00 -0.
-2.000000000000E+00 -0.
-4.000000000000E+00 -0.
-4.000000000000E+00 -0.
4.000000000000E+00 -0.

4 Roots				Remainders	
5.857864376269E-01	0.	NI=	5	2.84217094E-14	0.
-1.000000000000E+00	1.000000000000E+00	NI=	7	0.	0.
-1.000000000000E+00	-1.000000000000E+00	NI=	0	0.	0.
3.414213562373E+00	-4.038967834732E-28	NI=	2	-3.97903932E-13	-2.34022342E-26

Execution time = .028 seconds

Case 4, 5 Coefficients

4.000000000000E+00 -0.
-2.400000000000E+01 -0.
4.400000000000E+01 -0.
-2.400000000000E+01 -0.
3.000000000000E+00 -0.

(Continued)

4 Roots			Remainders	
1.771243444677E-01	0.	NI= 6	0.	0.
6.339745962156E-01	0.	NI= 6	-1.42108547E-14	0.
2.366025403785E+00	0.	NI= 2	2.13162821E-13	0.
2.822875655532E+00	0.	NI= 2	1.44950718E-12	0.

Execution time = .028 seconds

Case 5, 5 Coefficients

2.000000000000E+00 -0.
1.600000000000E+01 -0.
1.000000000000E+00 -0.
-7.400000000000E+01 -0.
5.600000000000E+01 -0.

4 Roots			Remainders	
1.121320343558E+00	0.	NI= 9	2.27373675E-13	0.
1.123105625620E+00	0.	NI= 5	4.54747351E-13	0.
-3.121320343560E+00	0.	NI= 2	-1.13686838E-12	0.
-7.123105625618E+00	0.	NI= 3	2.04636308E-12	0.

Execution time = .032 seconds

Case 6, 4 Coefficients

1.000000000000E+00 -0.
-6.026600000000E+00 -0.
4.304800000000E+00 -0.
1.595330000000E+01 -0.

3 Roots			Remainders	
-1.216399518172E+00	0.	NI= 5	-5.68434189E-14	0.
3.612590155512E+00	0.	NI= 2	-1.70530257E-13	0.
3.630409362660E+00	0.	NI= 2	-5.68434189E-14	0.

Execution time = .022 seconds

Case 7, 5 Coefficients

1.000000000000E+00 -0.
1.200000000000E+01 -0.
-9.500000000000E+00 -0.
-6.000000000000E+00 -0.
4.500000000000E+00 -0.

4 Roots			Remainders	
-7.071067811865E-01	0.	NI= 7	0.	0.
7.082039325011E-01	4.90945670725E-19	NI= 8	2.84217094E-14	1.02279363E-20
7.071067811861E-01	-1.218611372268E-27	NI= 2	0.	2.53657372E-29
-1.270820393250E+01	1.596373671581E-27	NI= 3	-1.18831167E-10	-3.44820039E-24

Executiom time = .030 seconds

(Continued)

Case 8, 5 Coefficients

1.000000000000E+00 -0.
-6.000000000000E+00 -0.
-1.130000000000E+02 -0.
5.040000000000E+02 -0.
2.436000000000E+03 -0.

4 Roots			Remainders	
-3.164414002969E+00	0.	NI= 6	0.	0.
9.164414003108E+00	0.	NI= 12	2.91038305E-11	0.
9.165151389772E+00	0.	NI= 1	0.	0.
-9.165151389912E+00	0.	NI= 2	-2.91038305E-11	0.

Execution time = .030 seconds

Case 9, 5 Coefficients

1.000000000000E+00 -0.
1.600000000000E+01 -0.
1.100000000000E+01 -0.
-2.240000000000E+02 -0.
2.860000000000E+02 -0.

4 Roots			Remainders	
1.841821538748E+00	-4.311068539088E-01	NI= 8	-1.81898940E-12	0.
1.841821538748E+00	4.311068539088E-01	NI= 0	-1.81898940E-12	0.
-5.726958802481E+00	2.584939414228E-26	NI= 2	9.09494702E-12	1.22262764E-23
-1.395668427502E+01	7.367077330550E-25	NI= 2	-7.63975549E-11	-1.51438351E-21

Execution time = .028 seconds
Total execution time = .275 seconds

Table 5. The Steepest Descent program, solutions of nine test cases.

Case 1, 4 Coefficients

1.000000000000E+00 -0.
-0. -0.
-1.000000000000E+00 -0.
-4.000000000000E+00 -0.

3 Roots			Remainders	
1.796321903259E+00	4.827632850013E-22	NI= 27	0.	4.19053842E-21
-8.981609516297E-01	1.191670795605E+00	NI= 7	-2.84217094E-14	-6.39488462E-14
-8.981609516297E-01	-1.191670795605E+00	NI= 0	2.84217094E-14	-7.10542736E-14

Execution time = .027 seconds

(Continued

Case 2, 5 Coefficients

1.000000000000E+00 -0.
-2.037900000000E+00 -0.
-1.542450000000E+01 -0.
1.566960000000E+01 -0.
3.549360000000E+01 -0.

4 Roots			Remainders	
2.124387030.81E+00	1.615587133893E-27	NI= 6	1.13686838E-12	-6.31811871E-26
-1.201998596673E+00	-4.316848821761E-24	NI= 6	1.59161573E-12	-1.59595558E-22
4.327505940890E+00	3.801722095294E-21	NI=12	-2.50111043E-12	3.49178173E-19
-3.211994374397E+00	-3.797406862059E-21	NI= 0	-2.04636308E-12	3.07094262E-19

Execution time = .028 seconds

Case 3, 5 Coefficients

1.000000000000E+00 -0.
-2.000000000000E+00 -0.
-4.000000000000E+00 -0.
-4.000000000000E+00 -0.
4.000000000000E+00 -0.

4 Roots			Remainders	
5.857864376269E-01	-6.018531076210E-36	NI= 7	2.84217094E-14	5.98309727E-35
-1.000000000000E+00	-1.000000000000E+00	NI=10	1.98951966E-13	0.
-1.000000000000E+00	1.000000000000E+00	NI= 0	1.98951966E-13	0.
3.414213562373E+00	0.	NI= 0	-2.10320650E-12	0.

Execution time = .027 seconds

Case 4, 5 Coefficients

4.000000000000E+00 -0.
-2.400000000000E+01 -0.
4.400000000000E+01 -0.
-2.400000000000E+01 -0.
3.000000000000E+00 -0.

4 Roots			Remainders	
6.339745962156E-01	0.	NI= 7	9.94759830E-14	0.
2.366025403784E+00	2.603240987229E-29	NI= 9	1.57740487E-12	-1.80357826E-28
1.771243444677E-01	-1.292469707114E-26	NI= 7	-2.27373675E-13	1.36782137E-25
2.822875655532E+00	1.289866466127E-26	NI= 0	7.53175300E-13	1.36506636E-25

Execution time = .028 seconds

Case 5, 5 Coefficients

2.000000000000E+00 -0.
1.600000000000E+01 -0.
1.000000000000E+00 -0.
-7.400000000000E+01 -0.
5.600000000000E+01 -0.

(Continued)

4 Roots			Remainders	
1.121320343561E+00	6.705418668574E-24	NI= 16	-2.27373675E-13	-8.37451008E-25
1.123105625616E+00	-6.700849586211E-24	NI= 6	0.	-8.37413820E-25
-3.121320343560E+00	-6.423719003118E-27	NI= 7	-1.13686838E-12	-9.25817806E-25
-7.123105625618E+00	1.854636640078E-27	NI= 0	-3.31965566E-11	-1.00915411E-24

Execution time = .033 seconds

Case 6, 4 Coefficients

1.000000000000E+00 -0.
-6.026600000000E+00 -0.
4.304800000000E+00 -0.
1.595330000000E+01 -0.

3 Roots			Remainders	
3.612590155512E+00	-4.930380657631E-29	NI= 13	1.70530257E-13	4.24253178E-30
-1.216399518172E+00	-1.912855166529E-24	NI= 12	3.41060513E-13	-4.47707387E-23
3.630409362660E+00	1.912904470335E-24	NI= 0	1.70530257E-13	1.65210465E-25

Executiom time = .025 seconds

Case 7, 5 Coefficients

1.000000000000E+00 -0.
1.200000000000E+01 -0.
-9.500000000000E+00 -0.
-6.000000000000E+00 -0.
4.500000000000E+00 -0.

4 Roots			Remainders	
7.071067811863E-01	1.860283898083E-25	NI= 19	2.84217094E-14	-3.87223306E-27
7.082039324996E-01	-1.858714064882E-25	NI= 7	0.	-3.87228362E-27
-7.071067811865E-01	1.615587133893E-26	NI= 5	5.68434189E-14	3.88077201E-25
-1.270820393250E+01	-1.631285465907E-26	NI= 0	-1.18831167E-10	3.52361059E-23

Execution time = .032 seconds

Case 8, 5 Coefficients

1.000000000000E+00 -0.
-6.000000000000E+00 -0.
-1.130000000000E+02 -0.
5.040000000000E+02 -0.
2.436000000000E+03 -0.

4 Roots			Remainders	
-3.164414002969E+00	3.944304526105E-31	NI= 8	1.45519152E-11	3.59786299E-28
9.164414002341E+00	-4.803393640885E-21	NI= 18	1.45519152E-10	8.00420721E-22
9.165151390540E+00	4.803200325787E-21	NI= 12	1.16415322E-10	8.00468580E-22
-9.165151389912E+00	1.933147035594E-25	NI= 0	-1.89174898E-10	-3.89754659E-22

Execution time = .058 seconds

(Continued)

Case 9, 5 Coefficients

1.000000000000E+00 -0.
1.600000000000E+01 -0.
1.100000000000E+01 -0.
-2.240000000000E+02 -0.
2.860000000000E+02 -0.

4 Roots			Remainders	
1.841821538748E+00	4.311068539088E-01	NI= 7	3.63797881E-12	-9.09494702E-13
1.841821538748E+00	-4.311068539088E-01	NI= 0	3.63797881E-12	0.09494702E-13
-5.726958802481E+00	-7.951273638166E-22	NI= 6	4.36557457E-11	-3.76080263E-19
-1.395668427502E+01	7.951273638166E-22	NI= 0	-7.63975549E-11	-1.63447146E-18

Execution time = .026
Total execution time = .308 seconds

Table 6. Lehmer's program, solution of nine test cases.

Case 1, 4 Coefficients

1.000000000000E+00 -0.
-0. -0.
-1.000000000000E+00 -0.
-4.000000000000E+00 -0.

3 Roots			Remainders	
1.796321903259E+00	0.	NI= 5	0.	0.
-8.981609516297E-01	1.191670795605E+00	NI= 0	0.	0.
-8.981609516297E-01	-1.191670795605E+00	NI= 0	0.	0.

1 Roots found by Lehmers method.

1.623588300000E+00 0. NI= 15

Execution time = .024 seconds

Case 2, 5 Coefficients

1.000000000000E+00 -0.
-2.037900000000E+00 -0.
-1.542450000000E+01 -0.
1.566960000000E+01 -0.
3.549360000000E+01 -0.

4 Roots			Remainders	
2.124387030181E+00	0.	NI= 4	2.27373675E-13	0.
-1.201998596673E+00	2.648153673532E-33	NI= 5	4.54747351E-13	9.79032577E-32
4.327505940890E+00	-7.059854738346E-34	NI= 0	-5.22959454E-12	-6.48429084E-32
-3.211994374397E+00	-1.942168199698E-33	NI= 0	-2.04636308E-12	1.57062104E-31

(Continued)

2 Roots found by Lehmers method.

1.623588300000E+00	0.	NI= 14
-1.623588300000E+00	-1.657167992024E-11	NI= 23

Execution time = .037 seconds

Case 3, 5 Coefficients

1.000000000000E+00 -0.
-2.000000000000E+00 -0.
-4.000000000000E+00 -0.
-4.000000000000E+00 -0.
4.000000000000E+00 -0.

4 Roots			Remainders	
5.857864376269E-01	0.	NI= 5	0.	0.
-1.000000000000E+00	1.000000000000E+00	NI= 5	1.13686838E-13	7.10542736E-14
-1.000000000000E+00	-1.000000000000E+00	NI= 1	0.	2.70006240E-13
3.414213562373E+00	-7.105427357601E-15	NI= 0	-3.97903932E-13	-4.11696458E-13

3 Roots found by Lehmers method.

8.117941500000E-01	0.	NI= 15
-1.148050296794E+00	1.148050296776E+00	NI= 22
-1.000000000000E+00	-1.000000000000E+00	NI= 0

Execution time = .037 seconds

Case 4, 5 Coefficients

4.000000000000E+00 -0.
-2.400000000000E+01 -0.
4.400000000000E+01 -0.
-2.400000000000E+01 -0.
3.000000000000E+00 -0.

4 Roots			Remainders	
1.771243444677E-01	0.	NI= 5	0.	0.
6.339745962156E-01	0.	NI= 5	1.42108547E-14	0.
2.822875655532E+00	0.	NI= 0	2.27373675E-13	0.
2.366025403785E+00	0.	NI= 0	9.94759830E-14	0.

2 Roots found by Lehmers method.

2.029485375000E-01	0.	NI= 23
8.117941500000E-01	0.	NI= 14

Execution time = .036 seconds

Case 5, 5 Coefficients

2.000000000000E+00 -0.
1.600000000000E+01 -0.
1.000000000000E+00 -0.
-7.400000000000E+01 -0.
5.600000000000E+01 -0.

(Continued)

4 Roots			Remainders	
1.123105625615E+00	0.	NI= 14	-2.27373675E-13	0.
1.121320343563E+00	0.	NI= 5	-2.27373675E-13	0.
-3.121320343560E+00	0.	NI= 0	-1.13686838E-12	0.
-7.123105625618E+00	0.	NI= 0	-2.63753464E-11	0.

2 Roots found by Lehmers method.

1.623588300000E+00	0.	NI= 13
1.623588300000E+00	0.	NI= 11

Execution time = .035 seconds

Case 6, 4 Coefficients

1.000000000000E+00 -0.
-6.026600000000E+00 -0.
4.304800000000E+00 -0.
1.595330000000E+01 -0.

3 Roots			Remainders	
-1.216399518172E+00	-1.053242938337E-35	NI= 5	1.70530257E-13	-2.46513511E-34
3.630409362660E+00	-2.854279222898E-33	NI= 0	-1.13686838E-13	-2.46513511E-34
3.612590155512E+00	2.864811652282E-33	NI= 0	1.13686838E-13	-2.46513511E-34

1 Roots found by Lehmers method.

-1.623588300000E+00	-1.657167992024E-11	NI= 23

Execution time = .027 seconds

Case 7, 5 Coefficients

1.000000000000E+00 -0.
1.200000000000E+01 -0.
-9.500000000000E+00 -0.
-6.000000000000E+00 -0.
4.500000000000E+00 -0.

4 Roots			Remainders	
7.082039325014E-01	0.	NI= 12	0.	0.
7.071067811846E-01	0.	NI= 5	2.84217094E-14	0.
-7.071067811866E-01	0.	NI= 0	-5.68434189E-14	0.
-1.270820393250E+01	0.	NI= 0	-1.18831167E-10	0.

2 Roots found by Lehmers method.

8.117941500000E-01	0.	NI= 15
8.117941500000E-01	0.	NI= 13

Execution time = .036 seconds

(Continued)

Case 8, 5 Coefficients

1.000000000000E+00 -0.
-6.000000000000E+00 -0.
-1.130000000000E+02 -0.
5.040000000000E+02 -0.
2.436000000000E+03 -0.

4 Roots			Remainders	
-3.164414002969E+00	3.081487911020E-33	NI= 3	0.	2.81083046E-30
9.165151389567E+00	1.686639269562E-29	NI= 18	-4.36557457E-11	2.81083046E-30
9.164414003314E+00	-1.686808003764E-29	NI= 0	-2.91038305E-11	2.81083046E-30
-9.165151389912E+00	-1.394145894822E-33	NI= 0	-1.89174898E-10	2.81083046E-30

2 Roots found by Lehmers method.

-3.247176600000E+00	-3.314335984047E-11	NI= 51
1.298870640000E+01	0.	NI= 18

Execution time = .048 seconds

Case 9, 5 Coefficients

1.000000000000E+00 -0.
1.600000000000E+01 -0.
1.100000000000E+02 -0.
-2.240000000000E+02 -0.
2.860000000000E+02 -0.

4 Roots			Remainders	
-5.726958802481E+00	0.	NI= 21	9.09494702E-12	0.
-1.395668427502E+01	0.	NI= 42	-3.81987775E-11	0.
1.841821538748E+00	4.311068539081E-01	NI= 0	6.54836185E-11	2.95585778E-12
1.841821538748E+00	-4.311068539081E-01	NI= 0	6.54836185E-11	-2.95585778E-12

2 Roots found by Lehmers method.

1.623588300000E+00	0.	NI= 14
1.623588300000E+00	0.	NI= 10

Execution time = .045 seconds
Total execution time = .361 seconds

Table 7. The QD program, solution of nine test cases.

Case 1, 4 Coefficients

1.000000000000E+00 -0.
-0. -0.
-1.000000000000E+00 -0.
-4.000000000000E+00 -0.

(Continued)

3 Roots			Remainders	
-8.981609516297E-01	1.191670795605E+00	NI= 1	-1.13686838E-13	1.42108547E-14
-8.981609516297E-01	-1.191670795605E+00	NI= 1	-1.13686838E-13	-1.42108547E-14
1.796321903259E+00	0.	NI= 2	-2.84217094E-14	0.

Execution time = .028 seconds

Case 1

Output from QD algorithm, 20 iterations.

Approximations to simple roots

1.796321905156E+00 0.

Quadratic approximations

3.796321903259E+00 0.
5.023094283389E+00 0.

Case 2, 5 Coefficients

1.000000000000E+00 -0.
-2.037900000000E+00 -0.
-1.542450000000E+01 -0.
1.566960000000E+01 -0.
3.549360000000E+01 -0.

4 Roots			Remainders	
4.327505940890E+00	0.	NI= 4	-2.50111043E-12	0.
-3.211994374397E+00	0.	NI= 4	-9.09494702E-13	0.
2.124387030181E+00	0.	NI= 3	9.09494702E-13	0.
-1.201998596673E+00	0.	NI= 2	2.39879228E-10	0.

Execution time = .033 seconds

Case 2

Output from QD algorithm, 20 iterations.

Approximations to simple roots

4.318501853609E+00 0.
-3.202884132530E+00 0.
2.124260753894E+00 0.
-1.201978474973E+00 0.

Case 3, 5 Coefficients

1.000000000000E+00 -0.
-2.000000000000E+00 -0.
-4.000000000000E+00 -0.
-4.000000000000E+00 -0.
4.000000000000E+00 -0.

(Continued)

4 Roots			Remainders	
-1.000000000000E+00	1.000000000000E+00	NI= 2	0.	0.
-1.000000000000E+00	-1.000000000000E+00	NI= 2	0.	0.
3.414213562373E+00	0.	NI= 2	-3.97903932E-13	0.
5.857864376269E-01	0.	NI= 2	0.	0.

Execution time = .034 seconds

Case 3

Output from QD algorithm, 20 iterations.

Approximations to simple roots

3.414213609724E+00 0.
5.857864546377E-01 0.

Quadratic approximations

2.000000000000E+00 0.
2.000000000000E+00 0.

Case 4, 5 Coefficients

4.000000000000E+00 -0.
-2.400000000000E+01 -0.
4.400000000000E+01 -0.
-2.400000000000E+01 -0.
3.000000000000E+00 -0.

4 Roots		Remainders	
2.822875655532E+00	0.	NI= 4	8.89599505E-12	0.
2.366025403785E+00	0.	NI= 4	2.84217094E-13	0.
6.339745962156E-01	0.	NI= 1	-5.92592642E-11	0.
1.771243444677E-01	0.	NI= 1	1.74082970E-11	0.

Execution time = .033 seconds

Case 4

Output from QD algorithm, 20 iterations.

Approximations to simple roots

2.835231957845E+00 0.
2.353669101481E+00 0.
6.339745962070E-01 0.
1.771243444661E-01 0.

Case 5, 5 Coefficients

2.000000000000E+00 -0.
1.600000000000E+01 -0.
1.000000000000E+00 -0.
-7.400000000000E+01 -0.
5.600000000000E+01 -0.

(Continued)

4 Roots			Remainders	
-7.123105625618E+00	0.	NI= 2	-3.31965566E-11	0.
-3.121320343560E+00	0.	NI= 2	2.27373675E-13	0.
1.123105625616E+00	0.	NI= 10	8.41282599E-12	0.
1.121320343562E+00	0.	NI= 10	4.54747351E-12	0.

Execution time = .035 seconds

Case 5

Output from QD algorithm, 20 iterations.

Approximations to simple roots

-7.123105712232E+00 0.
-3.121320133747E+00 0.
1.174656413337E+00 0.
1.069769432641E+00 0.

Case 6, 4 Coefficients

1.000000000000E+00 -0.
-6.026600000000E+00 -0.
4.304800000000E+00 -0.
1.595330000000E+01 -0.

3 Roots			Remainders	
3.630409362660E+00	0.	NI= 9	0.	0.
3.612590155513E+00	0.	NI= 9	1.70530257E-13	0.
-1.216399518172E+00	0.	NI= 2	-5.68434189E-14	0.

Execution time = .029 seconds

Case 6

Output from QD algorithm, 20 iterations.

Approximations to simple roots

3.792066694287E+00 0.
3.450932808651E+00 0.
-1.216399502938E+00 0.

Case 7, 5 Coefficients

1.000000000000E+00 -0.
1.200000000000E+01 -0.
-9.500000000000E+00 -0.
-6.000000000000E+00 -0.
4.500000000000E+00 -0.

1 Roots			Remainders	
-1.270820393250E+01	0.	NI= 1	-1.22605570E-09	0.

Execution time = .025 seconds

(Continued)

Case 7

Three or more roots with equal modulus.
The QD program can not find such roots, any other roots will be found.

Output from QD algorithm, 20 iterations.

Approximations to simple roots

-1.270820393250E+01 0.

Case 8, 5 Coefficients

1.000000000000E+00 -0.
-6.000000000000E+00 -0.
-1.130000000000E+02 -0.
5.040000000000E+02 -0.
2.436000000000E+03 -0.

1 Roots				Remainders	
-3.164414002969E+00	0.	NI=	2	0.	0.

Execution time = .027 seconds

Case 8

Three of more roots with equal modulus.
The QD program can not find such roots, any other roots will be found.

Output from QD algorithm, 20 iterations.

Approximations to simple roots

-3.164413979047E+00 0.

Case 9, 5 Coefficients

1.000000000000E+00 -0.
1.600000000000E+01 -0.
1.100000000000E+01 -0.
-2.240000000000E+02 -0.
2.860000000000E+02 -0.

4 Roots				Remainders	
1.841821538748E+00	4.311068539088E-01	NI=	2	1.81898940E-12	-6.82121026E-13
1.841821538748E+00	-4.311068539088E-01	NI=	2	1.81898940E-12	6.82121026E-13
-1.395668427502E+01	0.	NI=	2	-7.63975549E-11	0.
-5.726958802481E+00	0.	NI=	2	-3.63797881E-12	0.

Execution time = .034 seconds
Total execution time = .314 seconds

Case 9

Output from QD algorithm, 20 iterations.

Approximations to simple roots		Quadratic approximations	
-1.395668432028E+01	0.	-3.683643077497E+00	0.
-5.726958763430E+00	0.	3.578159700084E+00	0.

Table 8. Muller's program, solution of $x^{21} + x = 1$.

21 Roots			Remainders	
8.972916221835E-01	0.	NI= 11	-7.10542736E-15	0.
-1.940345152168E-01	-1.002778092979E+00	NI= 12	-1.20792265E-13	-1.11910481E-13
-1.940345152168E-01	1.002778092979E+00	NI= 0	-1.20792265E-13	1.11910481E-13
-4.901687318868E-01	9.022294492945E-01	NI= 8	-2.84217094E-14	3.19744231E-14
-4.901687318868E-01	-9.022294492945E-01	NI= 0	-2.84217094E-14	-3.19744231E-14
8.322780849658E-01	4.978437339585E-01	NI= 12	0.	8.88178420E-15
8.322780849658E-01	-4.978437339585E-01	NI= 0	0.	-8.88178420E-15
-9.257004498721E-01	4.585575083312E-01	NI= 9	-7.81597009E-14	7.81597009E-14
-9.257004498721E-01	-4.585575083312E-01	NI= 0	-7.81597009E-14	-7.81597009E-14
6.584719287059E-01	7.396523993201E-01	NI= 12	-3.55271368E-14	7.10542736E-14
6.584719287059E-01	-7.396523993201E-01	NI= 0	-3.55271368E-14	-7.10542736E-14
-1.022118072328E+00	1.579418431937E-01	NI= 10	0.	-1.77635684E-14
-1.022118072328E+00	-1.579418431937E-01	NI= 0	0.	1.77635684E-14
9.074327090550E-01	2.249415044457E-01	NI= 10	-7.10542736E-15	7.99360578E-15
9.074327090550E-01	-2.249415044457E-01	NI= 0	-7.10542736E-15	-7.99360578E-15
1.166759111031E-01	1.007289866987E+00	NI= 9	-1.20792265E-13	1.09245946E-13
1.166759111031E-01	-1.007289866987E+00	NI= 0	-1.20792265E-13	-1.09245946E-13
-7.423600687337E-01	7.148739183073E-01	NI= 9	-4.97379915E-14	9.76996262E-14
-7.423600687337E-01	-7.148739183073E-01	NI= 0	-4.97379915E-14	-9.76996262E-14
4.108773931162E-01	9.161620708855E-01	NI= 2	7.10542736E-15	2.84217094E-14
4.108773931162E-01	-9.161620708855E-01	NI= 0	7.10542736E-15	-2.84217094E-14

Execution time = .217 seconds

Table 9. The Steepest Descent program, solution of $x^{21} + x = 1$.

21 Roots			Remainders	
6.584719287059E-01	7.396523993201E-01	NI= 13	-4.97379915E-14	1.42108547E-14
6.584719287059E-01	-7.396523993201E-01	NI= 0	-4.97379915E-14	-1.42108547E-14
8.972916221835E-01	-5.422016552177E-15	NI= 16	-6.39488462E-14	-1.84552565E-14
-1.022118072328E+00	1.579418431937E-01	NI= 17	-3.12638804E-13	-8.26005930E-14
-1.022118072328E+00	-1.579418431937E-01	NI= 0	-3.12638804E-13	8.26005930E-14
8.322780849658E-01	4.978437339585E-01	NI= 23	-9.23705556E-14	2.13162821E-14
8.322780849658E-01	-4.978437339585E-01	NI= 0	-9.23705556E-14	-2.13162821E-14
-1.940345152168E-01	1.002778092979E+00	NI= 16	-9.09494702E-13	6.19948537E-13
-1.940345152168E-01	-1.002778092979E+00	NI= 0	-9.09494702E-13	-6.19948537E-13
1.166759111030E-01	1.007289866987E+00	NI= 18	5.68434189E-14	6.27053964E-13
1.166759111030E-01	-1.007289866987E+00	NI= 0	5.68434189E-14	-6.27053964E-13
-9.257004498720E-01	4.585575083312E-01	NI= 33	-2.27373675E-13	8.34887715E-14
-9.257004498720E-01	-4.585575083312E-01	NI= 0	-2.27373675E-13	-8.34887715E-14
9.074327090550E-01	-2.249415044456E-01	NI= 17	-1.13686838E-13	-1.03028697E-13
9.074327090550E-01	2.249415044456E-01	NI= 0	-1.13686838E-13	1.03028697E-13
4.108773931162E-01	9.161620708855E-01	NI= 8	-9.23705556E-14	6.46593890E-13
4.108773931162E-01	-9.161620708855E-01	NI= 0	-9.23705556E-14	-6.46593890E-13
-7.423600687337E-01	7.148739183074E-01	NI= 25	2.66453526E-12	-3.38040707E-12
-7.423600687337E-01	-7.148739183074E-01	NI= 0	2.66453526E-12	3.38040707E-12
-4.901687318867E-01	9.022294492944E-01	NI= 7	-4.96669372E-12	5.38236122E-13
-4.901687318867E-01	-9.022294492944E-01	NI= 0	-4.81037432E-12	-9.00612918E-13

Execution time = .428 seconds

Table 10. Lehmer's program, solution of $x^{21} + x = 1$.

21 Roots			Remainders	
8.972916221835E-01	-3.549874073495E-30	NI= 11	-7.10542736E-15	-1.20829282E-29
9.074327090550E-01	2.249415044457E-01	NI= 8	-7.10542736E-15	7.99360578E-15
9.074327090550E-01	-2.249415044457E-01	NI= 1	0.	-1.15463195E-14
6.584719287059E-01	7.396523993201E-01	NI= 6	-1.27897692E-13	-1.95399252E-14
6.584719287059E-01	-7.396523993201E-01	NI= 1	-1.84741111E-13	-3.55271368E-15
8.322780849658E-01	4.978437339585E-01	NI= 7	-3.55271368E-14	4.79616347E-14
8.322780849658E-01	-4.978437339585E-01	NI= 1	2.84217094E-14	-9.05941988E-14
1.166759111027E-01	1.007289866987E+00	NI= 18	7.73070497E-12	4.86544138E-12
1.166759111027E-01	-1.007289866987E+00	NI= 1	3.15480975E-12	-1.07558407E-11
4.108773931163E-01	9.161620708856E-01	NI= 9	1.84741111E-13	-1.05160325E-12
4.108773931162E-01	-9.161620708855E-01	NI= 1	2.70006240E-13	-1.39976919E-12
-4.901687318857E-01	9.022294492952E-01	NI= 8	-1.94120275E-11	-4.31228386E-11
-4.901687318869E-01	-9.022294492941E-01	NI= 1	-2.55795385E-12	-1.20916610E-11
-7.423600687358E-01	7.148739182994E-01	NI= 12	-3.88240551E-11	3.01380254E-10
-7.423600687357E-01	-7.148739183032E-01	NI= 1	1.31166189E-11	-1.71548109E-10
-9.257004498632E-01	4.585575083446E-01	NI= 14	-2.16992646E-10	-5.90384630E-10
-9.257004498569E-01	-4.585575083433E-01	NI= 1	-4.71601425E-10	5.98845418E-10
-1.940345152172E-01	1.002778092979E+00	NI= 7	4.97379915E-14	2.00612860E-11
-1.940345152159E-01	-1.002778092979E+00	NI= 1	-2.25242047E-11	1.65050196E-11
-1.022118072328E+00	1.579418431893E-01	NI= 0	-1.78346227E-11	1.78505211E-10
-1.022118072349E+00	-1.579418431878E-01	NI= 0	8.07510503E-10	-3.01083602E-10
19 Roots found by Lehmers method				
8.732761359618E-01	3.444150890329E-01	NI= 163		
1.041404209356E+00	2.296100593576E-01	NI= 114		
9.074327090550E-01	-2.249415044457E-01	NI= 0		
6.888301780693E-01	6.888301780728E-01	NI= 136		
6.584719287059E-01	-7.396523993201E-01	NI= 0		
8.987428083911E-01	5.740251483940E-01	NI= 93		
8.322780849658E-01	-4.978437339585E-01	NI= 0		
4.986149767822E-01	1.148050296781E+00	NI= 140		
1.166759111027E-01	-1.007289866987E+00	NI= 0		
4.986149767822E-01	1.148050296781E+00	NI= 121		
4.108773931163E-01	-9.161620708856E-01	NI= 0		
-4.592201187165E-01	1.164368181280E+00	NI= 118		
-4.901687318857E-01	-9.022294492952E-01	NI= 0		
-4.592201187165E-01	1.164368181280E+00	NI= 97		
-7.423600687358E-01	-7.148739182994E-01	NI= 0		
-4.592201187165E-01	1.164368181280E+00	NI= 73		
-9.257004498632E-01	-4.585575083446E-01	NI= 0		
-4.592201187165E-01	1.164368181280E+00	NI= 49		
-1.940345152172E-01	-1.002778092979E+00	NI= 0		

Execution time = .851 seconds

Table 11. The QD program, solution of $x^{21} + x = 1$.

21 Roots			Remainders	
-1.022118190300E+00	1.579420813749E-01	NI= 9	5.46523736E-06	-9.17729537E-06
-1.022118190300E+00	-1.579420813749E-01	NI= 9	5.46523736E-06	9.17729537E-06
-9.257004809081E-01	4.585574955028E-01	NI=17	1.07030931E-06	7.70297616E-07
-9.257004809081E-01	-4.585574955028E-01	NI=17	1.07030931E-06	-7.70297616E-07
-7.423601076346E-01	7.148739698403E-01	NI= 6	2.07735489E-06	-1.23721082E-06
-7.423601076346E-01	-7.148739698403E-01	NI= 6	2.07735489E-06	1.23721082E-06
-4.901687147918E-01	9.022294481375E-01	NI= 7	-5.29597926E-07	-2.72783470E-07
-4.901687147918E-01	-9.022294481375E-01	NI= 7	-5.29597926E-07	2.72783470E-07
-1.940345158357E-01	1.002778096734E+00	NI= 4	9.05780624E-08	-7.72544890E-08
-1.940345158357E-01	-1.002778096734E+00	NI= 4	9.05780624E-08	7.72544890E-08
1.166759086381E-01	1.007289866501E+00	NI= 2	3.34333023E-08	5.92667941E-08
1.166759086381E-01	-1.007289866501E+00	NI= 2	3.34333023E-08	-5.92667941E-08
4.108773929941E-01	9.161620709461E-01	NI= 2	2.55249688E-09	1.63799108E-09
4.108773929941E-01	-9.161620709461E-01	NI= 2	2.55249688E-09	-1.63799108E-09
6.584719287074E-01	7.396523993143E-01	NI= 1	-1.01437081E-10	9.25304278E-12
6.584719287074E-01	-7.396523993143E-01	NI= 1	-1.01437081E-10	-9.25304278E-12
8.322780849658E-01	4.978437339589E-01	NI= 1	3.86535248E-12	-1.28430599E-12
8.322780849658E-01	-4.978437339589E-01	NI= 1	3.86535248E-12	1.28430599E-12
9.074327090550E-01	2.249415044457E-01	NI= 1	-2.84217094E-14	8.61533067E-14
9.074327090550E-01	-2.249415044457E-01	NI= 1	-2.84217094E-14	-8.61533067E-15
8.972916221835E-01	0.	NI= 0	2.13162821E-14	0.

Execution time = .244 seconds

V. CONCLUSION

The main reason for the big difference in times for locating roots by the various methods is the difference in their rates of convergence. The different number of computer operations required to construct the various computer programs will also affect the times.

Examination and comparison of the number of computer operations per iteration, the number of iterations and the times required by the various programs in finding the roots of a given polynomial equation shows that the convergence rate is the overriding factor. In other words the number of computer operations per iteration, required by the various computer programs is relatively the same when compared to the large difference in the number of iterations which is associated with a large difference in times. Before discussing this further we will state some definitions.

An algorithm is said to be linearly convergent if the errors in two successive steps tend to be in a constant ratio.

An algorithm is said to be quadratically convergent if the error in the current step is proportional to the square of the error in the previous step.

Muller's method is slightly less than quadratically convergent when the roots are simple. In this case the error in the current step is proportional to the error in the previous step raised to the power

1.84. When there is a double root the error in the current step is proportional to the error in the previous step raised to the power 1.23, (8).

The solution of $x^{21} + x = 1$ (see Chapter IV) by the Steepest Descent program required about twice as many iterations as Muller's program. Also the execution time for the Steepest Descent program is about double that of Muller's program. In Table 3 of Chapter III we see that the time per root for the Steepest Descent program is more than double the time for Muller's program. From this information we can say that for equations of degree 21 or greater, the rate of convergence of the Steepest Descent program is about half that of Muller's program.

The nine test cases and Table 2 of Chapter III indicate that the Steepest Descent program converges on a root, for equations of degree four or less, at approximately the same rate as Muller's program.

The convergence rate of the QD algorithm is linear, the constant ratio between two successive steps depending on the separation of the roots (2,3). The QD method is used to find rough approximations to the roots. Newton's method or Bairstow's method, both of which are quadratically convergent, are used to refine the root approximations.

The convergence rate of Lehmer's method is linear with the

ratio being 2/5 (9). This information and the examples and tables in Chapter III indicate that Lehmer's program is the slowest of the four programs. It should be noted that Lehmer's method was only used to find rough approximations to the roots. The approximations were then refined in Newton's method.

Muller's method is recommended as the optimal method since it converged, at the highest rate, in all cases tried. It also yielded the greatest accuracy. Besides having a high rate of convergence, it requires the lowest number of FORTRAN IV computer statements which makes it the fastest and easiest of the four methods to code.

Muller's method can also be used to find the roots of transcendental equations and to find eigenvalues of arbitrary matrices without the computation of the coefficients of the associated characteristic equation, (Ref. 1).

BIBLIOGRAPHY

1. Frank, Werner L. Finding zeros of arbitrary functions. Journal of the Association for Computing Machinery 5:154-160. 1958.

2. Henrici, Peter. Elements of numerical analysis. New York, Wiley, 1964. 328 p.

3. Hildebrand, F. B. Introduction to numerical analysis. New York, McGraw-Hill, 1956. 511 p.

4. Keetov, A. N. and N. A. Kreeneetskee. Operator programming (Programmeerovaneeye): Section 32 of Elyektronnye tseefrovye masheeny, by A. N. Keetov and N. A. Kreeneetskee, trans. by Harry E. Goheen. Corvallis, Oregon State University, Dept. of Mathematics, 1962-63. 11 numb. leaves. (Typescript)

5. Lehmer, D. H. A machine method for solving polynomial equations. Journal of the Association for Computing Machinery 2: 151-162. 1961.

6. Milne, William Edmund. Numerical calculus. Princeton, N.J., Princeton University Press, 1949. 393 p.

7. Moore, J. B. A convergent algorithm for solving polynomial equations. Journal of the Association for Computing Machinery 14:311-315. 1967.

8. Muller, David E. A method for solving algebraic equations using an automatic computer. Mathematical Tables and Other Aids to Computation 10:208-215. 1956.

9. Ralston, Anthony. A first course in numerical analysis. New York, McGraw-Hill, 1965. 578 p.

10. Ward, James A. The down-hill method of solving $f(z) = 0$. Journal of the Association for Computing Machinery 4:148-150. 1957.

APPENDIX

Operator Programming (4)

We call a group of commands of a program possessing the following properties an elementary operator.

1. The property of effectiveness. An elementary operator carries out certain operation with numbers necessary for the solution of a problem on the machine.

2. Operations with numbers we understand to be the obtaining of some number or a value of the signal ω with the aid of one or several numbers. In particular, the transfer of numbers from one memory apparatus to another or from one part of the memory to another is an operation with numbers.

3. The property of being ordered. The control from outside (from another operator) may be obtained by only one command of an elementary operator, the first one. Transfer of control from command to command in an elementary operator comes about in only one definite order, as a rule in the order of the numbering of the commands (the numbering of cells holding the commands). Direction to the outside (to another operator) only one command of an elementary operator can give, the last one.

4. The property of connectedness. If the first command of an elementary operator received control, then each command of the elementary operator in order receives control.

5. The property of autonomy. Conditional transfer of control by an elementary operator may arise only depending on the value of the signal ω worked out by the elementary operator itself (and not any other).

6. The property of simplicity. An elementary operator must fulfill the smallest possible set of kinds of work. We always try to have an operator fulfill only an arithmetic calculation, or a verification of the fulfillment of a logical condition, or an address modification etc. But sometimes in order that an elementary operator fulfill only one definite kind of work, there must be introduced into the program supplementary commands useless for getting the

solution of the problem. Hence in place of the requirement that the elementary operator fulfill only one definite kind of work, we put the requirement that it fulfill the smallest possible set of kinds of work.

Of the commands which do not satisfy the above conditions we agree to count as elementary operators the stop command (operator $\mathcal{L}$) and the command to supply a zone of magnetic tape (designated by the symbol $\mathcal{Y}$).

The division of a program into operators is not unique. To aid a programs ability to be surveyed, it is necessary to strive to unite in each operator the largest set of commands.

For the most often encountered elementary operators standard designations are taken as provided in Table 12 on the following page.

Rules for Writing the Logical Schemes of Programs. For convenience in describing logical schemes, the operators depicting a scheme are written in one line. In doing this the following rules are taken:

1. The ordinal number of an operator in a given scheme is expressed by a subscript of the operator. The numbering of operators is the obvious one.

2. If an operator depends on a parameter, then this parameter is expressed by an upper index of the operator (for example A^i_2, P^i_{10}, etc.).

3. If the signs for two operators stand in order in the scheme, then the operator written on the left transfers control to the operator written on the right.

4. If a semicolon stands between two signs for operators in

Table 12. Elementary operators.

No.	Name	Work done	Designation
1	Arithmetic operator	Arithmetic calculation	A
2	Logical operator	Verification of the fulfillment of a logical condition	P
3	Transfer operator	Transfer of numbers from one memory device to another or from one part of the memory to another	Π
4	Address modification (with parameter i)	Address modification	F(i)
5	Reestablishment operator (with parameter i)	Reestablishment of commands by reduction of them to the form corresponding to the initial value of the parameter i	O(i)
6	Dispatch	Introduction of quantities into standard cells	З
7	Inverse dispatch	Transfer of a series of values of quantities from a standard cell into a succession of cells	З*
8	Dispatch of commands	Entering of new commands before repeating the work of an operator in place of certain of its commands	K
9	Appeal	Appeal to a group of operators with numbers m, m=1,...,n or to the subprogram named	E(m, n)
10	Circulation operator	Circulation of numbers in standard cells	Ц
11	Forming operator	Formation of new commands	ϕ
12	Tape feed	Supplying of a zone of the magnetic tape	Ч
13	Stop	Stop the machine	Т
14	Non-standard operator	Any operator distinct from the above enumerated ones	H

order, then from the operator written on the left there is no transfer of control to the operator written on the right.

5. Transfer of control to an operator not directly to the right is designated by the signs "⌜, ⌞". Their use will be explained in the following paragraphs.

6. Since in a machine on the branching of a program, transfer of control not to the next command in order is accomplished by one of the operations of a conditional jump on the basis of the value of a signal ω, it is convenient that logical conditions whose verification is fulfilled by logical operators be formulated so that the value of the signal ω be the truth value of the statement "Condition is fulfilled." For example, if for the choice of the direction of the calculating process comparison of numbers is made then it must be considered that the logical operator tests the truth of the statement "The compared numbers do not coincide." In the tracing of the schemes of programs it is accepted that the symbols answering the signal $\omega = 1$ are "⌜, ⌝" and those answering the signal $\omega = 0$ are "⌞, ⌟"; either symbol pair may be used when not connected with the value of the signal ω.

After the operator transferring control is placed the sign "$\overset{k}{\ulcorner}$" (where k is the number of the operator to which control is transferred). The sign "$\overset{\ell}{\urcorner}$" must be placed before the operator receiving control, (ℓ is the number of the operator transferring control). Similarly there is placed after the operator transferring control the sign "$\llcorner_{k}$" (in which k is the number of the operator receiving control). Before the operator receiving control is placed the sign "$\lrcorner_{\ell}$" (in which ℓ is the number of the operator transferring control).

In the logical scheme of a program the combination of symbols $\overset{k}{\ulcorner}\llcorner_{i}$ may be encountered. This combination is usually replaced by

the symbol $\overset{k}{\underset{i}{\square}}$. In place of the combination of symbols $\overset{k}{\urcorner}\ \underset{i}{\llcorner}$ or $\underset{i}{\llcorner}\ \overset{k}{\urcorner}$ there often is written the single symbol $\overset{k}{\underset{i}{\square}}$.

Before an operator there may occur the combination of symbols $\overset{i}{\urcorner}\ \overset{k}{\urcorner}\ \overset{\ell}{\urcorner}$ or $\underset{i}{\llcorner}\ \underset{k}{\llcorner}\ \underset{\ell}{\llcorner}\ \underset{m}{\llcorner}$ etc., which are usually replaced by the symbols $\overset{i,k,\ell}{\urcorner}$ and $\underset{i,k,\ell,m}{\llcorner}$ respectively, and so forth.

In many cases it is convenient for the simplification of the logical scheme of a program to unite elementary operators of a single type into a group and designate it by a single letter. A group of elementary operators may be designated in the logical scheme of a program by one letter only on condition that only one elementary operator of it may receive control from the outside (from operators not belonging to the group). Such a group of elementary operators we call a generalized operator. A generalized operator may contain elementary operators of differing functional significance but of a single type, i. e., logical.