

BR-322501

AD-A278 422



①

①

66pp

DTIC  
ELECTE  
APR 25 1994  
S F D

This document has been approved  
for public release and sale; its  
distribution is unlimited

94-12313

INFO QUANTITY INCLUDED 3

94 4 22 002

REFERENCE COPY  
NOT TO BE REMOVED  
FROM THE LIBRARY

United Kingdom Atomic Energy Authority

**HARWELL**



**Fortran subroutines for  
finding polynomial zeros**

K.Madsen and J.K.Reid  
Computer Science and Systems Division  
AERE Harwell, Oxfordshire  
February 1975

FORTRAN SUBROUTINES FOR FINDING POLYNOMIAL ZEROS

by

K. Madsen\* and J.K. Reid

ABSTRACT

In this report we present subroutines for finding all the roots of a polynomial and bounds on their errors. To find the zeros we use the algorithm of Madsen (1973) and to find error bounds we use the work of Peters and Wilkinson (1971) with some significant modifications. Both the real and complex cases are treated.

---

\* Institute for Numerical Analysis, Technical University of Denmark,  
2800 Lyngby, Denmark.

---

Computer Science and Systems Division,  
A.E.R.E. Harwell, Didcot,  
Oxford, England.

February, 1975

HL.75/1172 (C.13)

Accession	
NTIS	
DTIC	
Unannounced	
Justification	
By	
Distribution	
Availability	
Dist	A-1

## CONTENTS

	<u>Page No.</u>
1. Introduction	1
2. Finding the roots in the complex case	2
3. Error estimation	6
4. Changes to the algorithms in the real case	13
5. Description of Fortran subroutines	15
6. Test results and comparisons with other methods	18
7. Appendix. Specification sheets and listings	23
8. References.	65

ISBN-0-70-580215-9

## 1. Introduction

In this report we consider the problem of finding all the zeros of the polynomial

$$f(z) = a_0 + a_1z + \dots + a_nz^n \quad (1.1)$$

and estimating error bounds for them. To find the set of zeros we use the algorithm of Madsen (1973), which we have found to compare favourably with other algorithms. For error estimation we apply Rouché's theorem as recommended by Peters and Wilkinson (1971), but with some difference in detail. Both of these algorithms are a little simpler in the case where the polynomial has complex coefficients, so we describe the algorithms for this case in sections 2 and 3 and the modifications for the real case in section 4.

We believe that our code is in accord with the ANSI standard, having checked it with Bell Telephone Laboratories' Fortran verifier (Ryder, 1973) and run our test programs with array subscript checking. We describe this code in section 5 and in an appendix give specification sheets, and listings (produced by the Bell Laboratories' verifier, so that we include cross references). For the Harwell subroutine library we have made a small number of changes in order to shorten the argument lists, at the expense of a departure from ANSI standard. We have decided against including a single-length version in the library because the IBM 370/16 has a very short single-length word (6 hexadecimal digits, so that numbers just greater than unity are held to about 1 part in  $10^6$ ) but has double-length hardware which executes very little slower than the single-length hardware. Some results obtained with the library subroutines, together with comparisons with other algorithms are given in section 6. Our code (listed in section 7) contains comments suitable for machine processing which detail the changes needed for the single-length and

Harwell library versions.

We would like to acknowledge the help of M.J.D. Powell in carefully checking a draft of this report and making several valuable suggestions.

## 2. Finding the roots in the complex case

We use the algorithm of Madsen (1973) to find the root of minimal or near-minimal modulus and then use forward deflation to construct a polynomial of degree  $n-1$  whose roots are the remaining roots of the original polynomial. The process is repeated until approximations to all the roots have been found. Wilkinson (1963) has shown that forward deflation is stable provided a large root is not accepted before a much smaller one. Our algorithm does not guarantee that the moduli of the roots are strictly increasing but our experience has always been that they are found in roughly increasing order. A version with the composite deflation of Peters and Wilkinson (1971), which should be stable no matter in which order the zeros are found, was tried but it did not give more accurate results. In any case it is not clear how to apply the composite deflation when two complex conjugate roots of a real polynomial have been found so that deflation by a real quadratic factor is wanted.

It remains necessary to describe Madsen's (1973) algorithm in detail and we will consider its application to the original polynomial. The general strategy of the algorithm is that, given an iterate  $z_k$ , a tentative step  $dz_k$  is found and the next iterate  $z_{k+1}$  is taken at the best point (in the sense of  $|f(z)|$ ) encountered in a short search of values on the line through  $z_k$  and  $z_k+dz_k$ . Because the search may sometimes yield no better value than that at  $z_k$  we may sometimes have  $z_{k+1}=z_k$  and in this case ensure that the next tentative step is shorter and in a different direction. The inclusion of searches ensures rapid convergence to multiple roots and reliable convergence when difficulties are

encountered. Such a search is, however, wasteful if we are so near a simple root that Newton's iteration is reliable and fast. We have therefore devised a test (given by inequality (2.7)) which normally ensures this. While the algorithm performs searches we say it is in stage 1; otherwise it is in stage 2, performing straightforward Newton iteration. It begins in stage 1, which we now describe.

The tentative step  $dz_k$  is found with the help of stored values of  $z_k, f(z_k), f'(z_k), z_{k-1}$  and the previous tentative step  $dz_{k-1}$ . If the last iteration was successful ( $z_k \neq z_{k-1}$ ) then the Newton correction

$$n_k = -f(z_k)/f'(z_k) \quad (2.2)$$

is calculated and the next tentative step is taken as

$$dz_k = \begin{cases} n_k & \text{if } |n_k| \leq 3|z_k - z_{k-1}| \\ 3|z_k - z_{k-1}| e^{i\theta} n_k / |n_k| & \text{otherwise} \end{cases} \quad (2.3a)$$

$$(2.3b)$$

where  $\theta$  is chosen (rather arbitrarily) as  $\arctan(3/4)$ . If the last step was unsuccessful ( $z_k = z_{k-1}$ ) then we take the tentative step to be

$$dz_k = -\frac{1}{2} e^{i\theta} dz_{k-1}. \quad (2.3c)$$

After a successful iteration we normally expect to want to take a Newton step (2.3a), but we include the alternative (2.3b) because accidentally coming near to a stationary point of  $f(z)$  is likely to make the Newton step ridiculously large. We include a change of direction in (2.3b) because if a saddle point is being approached the direction  $n_k$  may be a worse search direction than almost any other. After an unsuccessful iteration we want to change the search direction to one likely to be successful and reduce the step size; this leads to formula (2.3c). Its repeated use is sure to yield a descent direction.

Once the tentative step has been found we test the inequality

$$|f(z_k + dz_k)| < |f(z_k)| \quad (2.4)$$

If this is satisfied then we calculate the numbers

$$|f(z_k + p dz_k)|, \quad p=1,2,\dots,n \quad (2.5)$$

continuing for as long as these are strictly decreasing. If inequality (2.4) does not hold then we calculate the numbers

$$|f(z_k + dz_k/2^p)|, \quad p=0,1,2, \quad |f(z_k + \frac{1}{2}e^{i\theta} dz_k)| \quad (2.6)$$

again continuing until the sequence ceases to decrease. In all cases we take  $z_{k+1}$  to be the best point found. Note that if there is a true multiple zero of multiplicity  $m$  or if we are a fair distance from a cluster of  $m$  zeros then  $z_{k+mn}$  will be a very good estimate of the solution and will be found by our search. In fact we get quadratic convergence to a multiple zero. Note also that when an iteration fails following a search to the end of sequence (2.6), the choice (2.3c) leads to a step likely to be in a direction of decreasing  $|f(z)|$ .

To complete our description of stage 1 we need to specify starting iterates. We take these to be

$$\left. \begin{aligned} z_0 &= 0 \\ dz_0 &= \begin{cases} -f(0)/f'(0) & \text{if } f'(0) \neq 0 \\ 1 & \text{otherwise} \end{cases} \\ z_1 &= \frac{1}{2} \min_{k>0} \left( \left| \frac{a_0}{a_k} \right|^{1/k} \right) \frac{dz_0}{|dz_0|} \end{aligned} \right\} \quad (2.7)$$

The iteration really starts from  $z_1$  but we have to include  $z_0$  and  $dz_0$  because they are needed for choosing the tentative step  $dz_1$ . This choice of  $z_1$  is used because its modulus is certainly less than that of any root of  $f(z)$  and it is in the direction of steepest descent of  $|f(z)|$  from the



origin. It is therefore likely that we will converge to a root of near-minimal modulus.

We do not make our main test for switching to stage 2 (straightforward Newton iteration) until a stage 1 search has led to the choice

$z_{k+1} = z_k + dz_k$ . This is obviously sensible and has the added virtue that we should (correctly) avoid the switch when converging to a multiple root.

The test itself is based on the Kantorowicz theorem (see, for example, Ostrowski (1966)) which states that if  $K_0$  is the circle with centre  $z_k + n_k$  and radius  $|n_k|$  where  $n_k$  is the Newton step (2.2) then the conditions

$$\left. \begin{aligned} f(z_k) f'(z_k) &\neq 0 \\ 2|f(z_k)| \max_{z \in K_0} |f''(z)| &\leq |f'(z_k)|^2 \end{aligned} \right\} \quad (2.8)$$

ensure the convergence of Newton's iteration starting from  $z_k$ . This leads us to test the inequality

$$2|f(z_k)| |f'(z_{k-1}) - f'(z_k)| \leq |f'(z_k)|^2 |z_{k-1} - z_k|. \quad (2.9)$$

Of course this is not equivalent to test (2.8) because we have replaced  $\max_{z \in K_0} |f''(z)|$  by a rather crude difference approximation but we

nevertheless expect it usually to predict correctly that straightforward Newton iteration will be satisfactory. We check inequality (2.9) at every step in stage 2 and switch back to stage 1 if it is violated. We also check the inequality (2.4) and if this is violated return to stage 1 beginning by modifying the tentative step with formula (2.3c) as in stage 1.

We complete this section by describing our convergence criterion. We terminate if a stage 1 search or a stage 2 iteration leads to a new iterate  $z_{k+1}$  different from  $z_k$  and yet such that the inequality

$$|z_{k+1} - z_k| < \epsilon |z_{k+1}| \quad (2.10)$$

holds where  $\epsilon$  is the largest number such that to machine accuracy  $1+\epsilon = 1$ .

We also terminate if the condition

$$|f(z_{k+1})| = |f(z_k)| \leq 16n|a_0|\epsilon \quad (2.11)$$

holds. The expression  $16n|a_0|\epsilon$  is a generous overestimate of the final roundoff made in calculating  $f(z)$  at the root of smallest modulus and we expect that such accuracy will be attainable. The normal convergence pattern is that  $|f(z_k)|$  decreases until well below  $16n|a_0|\epsilon$  and then roundoff errors cause a new iterate  $z_{k+1} = z_k$  to be taken so that (2.11) is satisfied. If such accuracy is unattainable then the step will decrease steadily because of the application of (2.3c) until (2.10) is satisfied. This combination of convergence criteria means that we are certain to obtain a good solution and almost certain to obtain the best possible. Furthermore this result is usually obtained with only one more iteration than is necessary to get this good accuracy.

### 3. Error estimation

We seek to estimate error bounds for all the roots produced by the algorithm of the previous section. We base our algorithm on that of Peters and Wilkinson (1971) and will follow their notation. The most significant difference is that they look for non-overlapping discs each of which contains precisely the same number of exact and approximate roots, whereas we allow overlapping discs. This is because it can happen that one root is well determined although it is inside the best disc obtainable for another (ill-determined) root. We therefore look for a separate disc for each root and take its centre to be at the calculated root itself. This also allows a very simple form of output to the user since

all that is required is a radius for each root.

We suppose that we have approximate roots  $\alpha_i, i=1,2,\dots,n$ . Then the polynomial

$$P(z) = a_n \prod_{i=1}^n (z-\alpha_i) \quad (3.1)$$

should agree with the original polynomial (1.1), but because the roots are not exact there will be an error

$$Q(z) = P(z) - f(z). \quad (3.2)$$

Now Rouché's theorem states that if  $P(z)$  and  $Q(z)$  are analytic functions in and on the closed curve  $C$  and the inequality

$$|Q(z)| < |P(z)| \quad (3.3)$$

holds on  $C$  then  $P(z)$  and  $P(z)-Q(z)$  have the same number of zeros inside  $C$ . We apply this here by looking for circles with centres  $\alpha_i, i=1,2,\dots,n$  on which condition (3.3) holds. The following theorem shows that there is no need to worry about the overlapping of some of these discs.

Theorem 1 If condition (3.3) holds on each of the circles centre  $\alpha_i$ , radius  $r_i, i=1,2,\dots,n$ , then the roots  $\phi_i$  of  $f(z)$  may be ordered so that

$$|\alpha_i - \phi_i| \leq r_i, \quad i=1,2,\dots,n. \quad (3.4)$$

Proof Regard the perimeters of the circles as dividing the plane into a set of non-overlapping regions  $R_i$ , each of which is the intersection of a subset of the set of discs that the circles enclose and their complements. A simple case is illustrated in Figure 1. Let  $R_{k_i}$  be the region containing  $\alpha_i, i=1,2,\dots,n$ . Note that a region may contain more than one  $\alpha_i$  so that there may be coincidences among the  $k_i$  (e.g.  $k_1=k_2=3$  in Figure 1). The set of regions  $R_{k_i}, i=1,2,\dots,n$  together contain all the

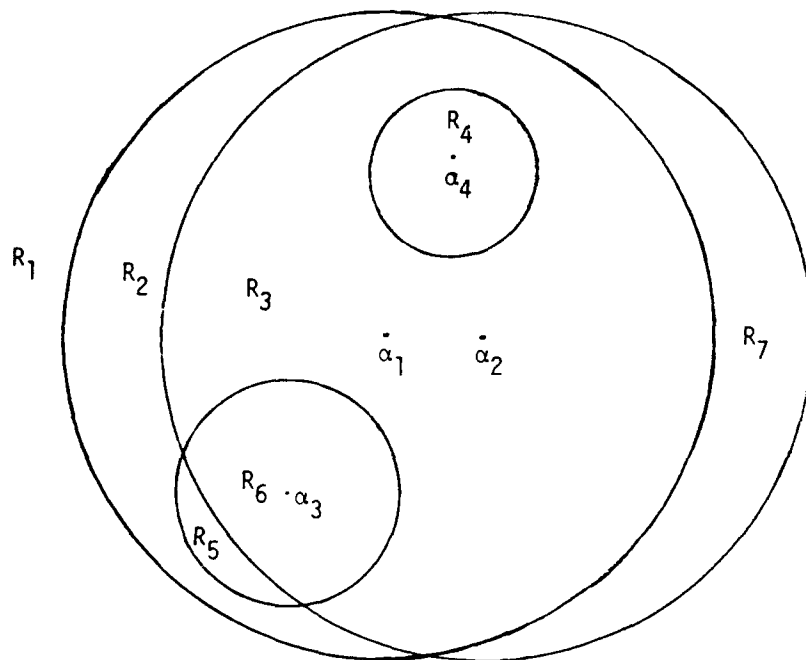


Figure 1 Four circles centre  $\alpha_i$  dividing plane into seven regions  $R_i$

roots  $\alpha_i$  of  $P(z)$  and each  $R_{k_i}$  has a boundary on which condition (3.3) holds so contains exactly the same number of roots of  $f(z)$  as of  $P(z)$ . Therefore the regions  $R_{k_i}$  ( $i=1,2,\dots,n$ ) contain all the roots of  $f(z)$  and these may be ordered so that  $R_{k_i}$  contains  $\phi_i$  ( $i=1,2,\dots,n$ ). The result now follows since each  $R_{k_i}$  is contained in the disc centre  $\alpha_i$  radius  $r_i$ . ■

Being able to use overlapping discs leads to simplifications in coding and sometimes to much better error bounds. This improvement is illustrated by the example shown in Figure 1, where  $\alpha_1$  and  $\alpha_2$  are quite ill-conditioned but  $\alpha_3$  and  $\alpha_4$  are not. The procedure of Peters and Wilkinson would have forced us to regard all four  $\alpha_i$  as a cluster and to enclose them in a single disc. This would have made little difference to the bounds for  $\alpha_1$  and  $\alpha_2$  but would have significantly worsened the bounds

for  $\alpha_3$  and  $\alpha_4$ . Another illustration of this point is given in §6 with an example using our code.

We now suppose that the error polynomial is

$$Q(z) = \sum_{i=0}^n \epsilon_i z^i \quad (3.5)$$

and describe how to find a suitable circle for a typical root. For simplicity of notation let us imagine that the roots are reordered so that the one for which we are seeking a circle is  $\alpha_1$  and  $|\alpha_i - \alpha_1|$  increases monotonically with  $i$ . Rouché's condition (3.3) is satisfied on a circle centre  $\alpha_1$  and radius  $r$  if the inequality

$$\sum_{i=0}^n |\epsilon_i| (r + |\alpha_1|)^i < |a_n| \prod_{i=1}^n (|\alpha_i - \alpha_1| - r) \quad (3.6)$$

holds. If  $m$  is such that

$$|\alpha_m - \alpha_1| < r < |\alpha_{m+1} - \alpha_1| \quad (3.7)$$

(we ignore the case  $r = |\alpha_i - \alpha_1|$  for some  $i$  because clearly inequality (3.6) cannot be satisfied in this case) then we may rewrite inequality (3.6) in the form

$$k(r) < \prod_{i=1}^m (r - |\alpha_i - \alpha_1|) \quad (3.8)$$

where  $k(r)$  is given by the equation

$$k(r) = \frac{\sum_{i=0}^n |\epsilon_i| (r + |\alpha_1|)^i}{|a_n| \prod_{i=m+1}^n (|\alpha_i - \alpha_1| - r)} \quad (3.9)$$

Peters and Wilkinson (1971) solve the equation

$$(1.1) \quad k(0) = \left( r_1 - \max_{1 \leq i \leq m} |\alpha_i - \alpha_1| \right)^m \quad (3.10)$$

and then check whether  $r_1$  satisfies inequalities (3.7) and (3.8). Almost always inequality (3.8) holds because

(i) the right hand side of (3.10) underestimates the corresponding expression in (3.8),

(ii) the safety factor (1.1) has been introduced

and (iii)  $k(r)$  is nearly constant in the usual case where the roots

$\alpha_i, i=1,2,\dots,m$  are well separated from the rest.

They give no recommendation for dealing with the case where  $r_1$  does not satisfy (3.8), but presumably intend that an iteration should be set up with 0 and  $r_1$  in (3.10) being replaced by  $r_j$  and  $r_{j+1}$ . This yields a monotonically increasing sequence, so that we must eventually either satisfy inequality (3.8) or break the right-hand inequality (3.7)

necessitating a new start with a greater value for  $m$ . Actually Peters and Wilkinson take the centre of their circle to be  $\bar{\alpha} = \frac{m}{\sum_{i=1}^m \alpha_i}/m$  rather than  $\alpha_1$  and this means that the right-hand side of (3.10) is quite a good estimate of the right-hand side of (3.8) so that the procedure gives a realistic radius  $r$  without very much computation. Unfortunately this is not the case with  $\alpha_1$  in use, except when  $m=1$  and we have sometimes obtained solutions such that the right-hand side of (3.8) exceeds the left by factors as big as 1,000. We have therefore decided to use an iteration based directly on (3.8). Given an iterate  $r_j$  we seek a new iterate such that

$$(1.05) k(r_j) < \prod_{i=1}^m (r_{j+1} - |\alpha_i - \alpha_1|) < (1.1) k(r_j). \quad (3.11)$$

In view of inequality (3.7) it seems sensible to start with  $r_0 = |\alpha_m - \alpha_1|$  rather than Wilkinson and Peters'  $r_0=0$ . To solve (3.11) we use the method of bisection because  $m$  is usually small so that function evaluations are cheap, no great accuracy is required in view of the slack in (3.11),

and suitable initial upper and lower bounds are available in  $r_j$  and the Peters and Wilkinson overestimate obtained from the analogue of equation (3.10). Typically between four and seven iterates are required.

Peters and Wilkinson suggest that  $r_{j+1}$  should be checked directly in (3.8) but a simple test often avoids any need for this. If the inequality

$$\left( \frac{r_{j+1} + |\alpha_1|}{r_j + |\alpha_1|} \bigg/ \frac{|\alpha_{m+1} - \alpha_1| - r_{j+1}}{|\alpha_{m+1} - \alpha_1| - r_j} \right)^n \leq 1.05 \quad (3.12)$$

holds then it follows from the definition (3.9) that the inequality

$$k(r_{j+1}) \leq 1.05 k(r_j) \quad (3.13)$$

also holds. From this last inequality and the first inequality (3.11) we deduce that  $r_{j+1}$  satisfies inequality (3.8).

Rounding errors occur when the coefficients of  $P(z)$  are calculated, but can be reduced by multiplying out the factors  $(z - \alpha_i)$  in order of increasing  $|\alpha_i|$ . We therefore use the same order as that in which they were calculated. If bounds  $e_i$  on the errors were available we could work with the error polynomial  $\sum_{i=0}^n (|\epsilon_i| + e_i) z^i$  in place of the polynomial (3.5) and obtain strict bounds on the errors in the calculated roots. We hoped to use the running error analysis of Peters and Wilkinson for this purpose, but unfortunately found that it sometimes gave gross overestimates, as explained in the last paragraph of this section. We therefore have ignored this source of error, relying on such errors being reflected in larger coefficients  $\epsilon_i$  in the polynomial  $Q(z)$ . It should be noted that the sequence of constructed polynomials  $\prod_{i=1}^r (z - \alpha_i)$ ,  $r=1,2,\dots,n$  is not identical with the sequence of deflated polynomials used when finding the roots because we perform the multiplication in the same order as that in which the roots were found. Therefore errors produced by the

multiplication are likely to show in enlarged coefficients  $\epsilon_i$ . Not bounding these errors means that our bounds are not strict bounds, but they are more realistic. In none of our tests did the actual errors exceed the estimated bounds.

It is straightforward to allow for the effects of uncertainties in the original coefficients by working with the error polynomial  $\sum_{i=0}^n (|e_i| + b_i)$  where  $b_i, i=0,1,\dots,n$ , are bounds on these errors. This we do in our subroutine.

We complete this section by explaining why the running error analysis of Peters and Wilkinson for the coefficients of  $\prod_{i=1}^n (z - \alpha_i)$  sometimes gives very pessimistic results. If the computed product  $\prod_{i=1}^r (z - \alpha_i)$  is  $\sum_{i=0}^r b_i^{(r)} z^i$  then they generate coefficients  $f_i^{(r)}$  from the recurrences.

$$\left. \begin{aligned} f_0^{(1)} &= f_1^{(1)} = 0 \\ f_0^{(r+1)} &= |\alpha_{r+1}| f_0^{(r)} + |b_0^{(r+1)}| \\ f_i^{(r+1)} &= f_{i-1}^{(r)} + |\alpha_{r+1}| f_i^{(r)} + |\alpha_{r+1}| (|b_i^{(r)}| + |b_i^{(r+1)}|), \\ &\quad i=1,2,\dots,r \\ f_{-1}^{(r+1)} &= f_{r+1}^{(r+1)} = 0 \end{aligned} \right\} r=1,2,\dots,n-1 \quad (3.14)$$

to yield bounds  $2^{-t} f_i^{(n)}$  for the coefficients of  $P(z)$  if floating-point computation with  $t$  binary places is used. This can sometimes lead to very pessimistic bounds since the recurrence (3.14) fails to distinguish properly between  $\prod(z - \alpha_i)$  and  $\prod(z + |\alpha_i|)$ . In fact the recurrence (3.14) is obtained by bounding the corresponding recurrence

$$e_i^{(r+1)} = e_{i-1}^{(r)} - \alpha_{r+1} e_i^{(r)} - \epsilon_1 \alpha_{r+1} b_i^{(r)} - \epsilon_2 b_i^{(r+1)}. \quad (3.15)$$



Such bounds are particularly unrealistic in such a case as  $f(z) = z^n - 1$  since the numbers  $f^{(r)}$  grow with  $r$  in much the same way as the recurrence for generating the coefficients of  $(z+1)^n$  and these coefficients are  ${}^nC_r, r=0,1,\dots,n$ . In fact the recurrence (3.14) has some added terms when compared with the recurrence for the coefficients of  $(z+1)^n$ .

#### 4. Changes to the algorithms in the real case

The only change made to our root-finding algorithm in the case where  $f(z)$  has real coefficients is that once a complex root has been found it is either perturbed into a real root or its conjugate is taken as another root. This ensures that each deflated polynomial is also real and that work is saved for genuinely complex roots. Once a complex root  $\alpha_k = x_k + iy_k$  has been found we evaluate  $f(x_k)$  and use Peters and Wilkinson's (1971) running error analysis to bound the roundoff error made in this evaluation. The recurrence used for evaluating  $f(x_k)$  is given by the equations

$$\left. \begin{aligned} s_n &= a_n \\ s_i &= x_k s_{i+1} + a_i, \quad i=n-1, \dots, 0 \\ f(x_k) &= s_0 \end{aligned} \right\} \quad (4.1)$$

and the corresponding running error analysis is given by the equations

$$\left. \begin{aligned} g_n &= 0 \\ g_i &= |x_k| (g_{i+1} + |s_{i+1}|) + |s_i|, \quad i=n-1, \dots, 0 \end{aligned} \right\} \quad (4.2)$$

to yield the bound  $\epsilon g_0$  for the error in  $f(x_k)$  where  $\epsilon$  is the relative floating-point accuracy. If the inequality

$$|s_0| < 2\epsilon g_0 + |f(\alpha_k)| \quad (4.3)$$

holds (where  $f(\alpha_k)$  is the computed value at  $\alpha_k$ ), we take  $x_k$  to be a real root and deflate with it. Otherwise we deflate with the complex conjugate pair  $x_k \pm iy_k$ . It is very important that the case where we have a simple

real root well separated from the other roots should not be taken as a complex pair because a double deflation in such a case would be disastrous. We therefore consider this case and suppose that  $\phi_k$  is such a real root and  $\alpha_k$  is the (complex) approximation to it found by the algorithm.  $x_k$  (real part of  $\alpha_k$ ) is a better approximation to  $\phi_k$  in the sense that the inequality

$$|\phi_k - x_k| \leq |\phi_k - \alpha_k| \quad (4.4)$$

holds since  $\phi_k$  is real. Further the exact value  $f(z)$  behaves like a constant times  $(z - \phi_k)$  if  $z$  is near  $\phi_k$ . It is therefore reasonable to expect that the inequality

$$|f(x_k)| \leq |f(\alpha_k)| \quad (4.5)$$

holds between the corresponding exact values. We therefore expect  $x_k$  to be at least as good a root as  $\alpha_k$  so that it is virtually certain that inequality (4.3) will hold, since  $\epsilon_{g_0}$  is a rigorous upper bound on the round-off error in computing  $f(x_k)$  and may also be taken (slightly incorrectly) as a bound on the error in computing  $|f(\alpha_k)|$ .

A disadvantage of the test (4.3) is that we may decide we have a real root when a complex conjugate pair is the true position. We feel that this disadvantage is quite slight because it is reasonable to regard any point for which inequality (4.3) holds as a good approximate root.

The only change to the error bounding algorithm is that once a bound for a complex root has been found, this bound is also used for its complex conjugate.

## 5. Description of Fortran subroutines

In this section we describe the Fortran subroutines themselves. It is a short section because we have included rather full comments in the code itself, since we believe that this provides the most convenient form of documentation. Specification sheets and listings of the code are given in section 7. The specification sheets document all three versions (double- and single-length standard Fortran and double-length IBM Fortran). The listings are of the standard Fortran double-length versions and contain specially coded comments detailing changes needed for the other versions. We have been using a simple preprocessor (written in standard Fortran) to convert from one version to another. Where a statement differs between versions we preceded the genuine (double-length standard) statement by the alternative version or versions modified by the insertion of C in column 1, / in column 72, and the letter I or S (for IBM version or single-length version) in column 71.

The listings used are those produced by the Bell Telephone Laboratories' Fortran verifier because they include cross-references which are very helpful when reading the code. They consist of lists of all the identifiers and labels in lexicographical order together with the statement numbers of all references to them and the following coded information for each identifier:

TYPE:	Col 1:	E if explicitly typed
	Col 2:	I for integer
		R for real
		D for double precision
		C for complex
		L for logical
		H for Hollerith
USE:		FA for arithmetic statement function argument
		FN for function
		E for external subroutine or function
		GT for assigned "go to" variable
		IF for intrinsic function
		SN for subroutine
		V for variable

ATTRIBUTES: Col 1: C if in COMMON  
Col 2: E if an EQUIVALENCE  
Col 3: A if a dummy argument  
Col 4: S if value set by program unit  
Col 5: S if scalar  
          A if array  
Col 6: if array then number of dimensions

Because double-length complex facilities are not available in standard Fortran we do not use any complex variables, although the arguments A,R and E may be regarded as COMPLEX\*16 (or COMPLEX for the single-length version) on the IBM 360/370 series. All complex variables are written as arrays of length two and all complex arrays are written as 2-dimensional arrays whose first dimension is two. This leads to the code being almost as easy to read as if the complex facilities were used. Complex division is more complicated than can be written conveniently as in-line code so we have taken this out of line to the subroutines PA06ED/7ED.

We begin by describing the versions for complex polynomials, that is PA06AD/BD/CD/DD/ED.

The main call is to a short routine PA06AD which calls PA06BD to find the roots and PA06CD to find error bounds. PA06DD and PA06ED are short subroutines for polynomial evaluation and complex division, respectively. The time taken to find error bounds is usually about 25% of that taken to find the roots themselves but can rise to as much as 100% when there are many multiple roots. Because of this overhead we felt it was desirable for the user to be allowed to call PA06BD directly and instructions about how to do this are included in the specification sheet. When calling PA06CD (instruction 9 of PA06AD) the work-space provided by the user is divided into the four areas required by PA06CD. Also PA06AD checks for zero leading coefficients and sets dummy error bounds to correspond, since PA06CD assumes that the leading coefficient is non-zero.

The algorithm used by PA06BD to find the roots has already been explained in section 2 with the minor exceptions of the inclusion of tests for zero leading coefficients (roots at infinity), tests for zero trailing coefficients (roots at zero) and the scaling of all deflated polynomials so that the largest coefficient has modulus approximately equal to the reciprocal of the modulus of the smallest non-zero coefficient. We implicitly assumed in section 2 that leading and trailing coefficients were non-zero and the inclusion of scaling minimizes the likelihood of underflow or overflow. To avoid additional roundoff we scale by a power of the floating-point base. For speed of execution (in function calls rather than basic arithmetic) we use single-length working for scaling and for finding the initial iterate. Also we avoid time-wasting evaluations of the moduli of complex numbers by working with the squares of the moduli or the sums of the absolute values of the real and imaginary parts. Subroutines PA06DD and PA06ED, called from several places in PA06BD, should be regarded as part of PA06BD. PA06DD evaluates a complex polynomial at a complex point and finds the square of the modulus of the result. PA06ED is a simple subroutine for complex division. The arguments of PA06DD and PA06ED are explained in comments at the heads of each subroutine. Other coding details of PA06BD are explained in comments.

Subroutine PA06CD, which finds the error bounds using the algorithm of section 3, again makes use of single-length arithmetic whenever possible. In particular we found that CABS executes significantly faster than its double-precision equivalent (which in any case is not standard FORTRAN). Therefore the error polynomial is held in single-length, the function  $k(r)$  given by equation (3.9) is calculated in single-length and all the Rouché tests are performed in single-length. The details of the code are explained in comments.

We have tried to make the code for the case with real coefficients, namely subroutines PA07AD-PA07ED, resemble that for the complex case as much as possible, using the same labels and the same comments wherever this is appropriate. The only significant change in PA07AD lies in the way the work-array W is subdivided when PA07CD is called. The only significant changes in PA07BD lie in the code for deflation (instructions 138-166), which is much more complicated because we need to test for a real root and need to include code for deflation with a pair of complex conjugate roots. The code for polynomial evaluation (in PA07DD) is longer because we make use of the fact that the coefficients are real and treat separately the case where the point of evaluation is real. The main changes in PA07CD are

(i) Recognising complex roots when forming the polynomial  $\Pi(z-\alpha_j)$  and multiplying them in as a complex conjugate pair to preserve real coefficients (instructions 18-28). It is assumed that conjugate pairs are adjacent in array R.

(ii) Much more complicated code (instructions 39-59) for finding distances from the  $I^{\text{th}}$  root to all the rest. This code avoids calling CABS to find the distances between two real roots and so is much faster where most roots are real.

(iii) Using the same error bound for a complex root and its conjugate (instructions 114-115).

#### 6. Test results and comparisons with other methods

Our original reason for providing a new routine for the Harwell library was that the existing routine PA01 sometimes gave incorrect answers. The new routine (PA07) is able to get answers all of which have good accuracy in about half the time. We did not pursue the error in PA01.

We have compared our algorithm with that of Jenkins and Traub (1970) in two ways. First we compared the Algol-W code given by Madsen (1973) with an Algol-W version of the code of Jenkins (1969) and found Madsen's to be 4 to 5 times faster. The required changes from Algol-60 to Algol-W are very minor. Next we compared our Fortran code with a rather free translation into Fortran of Jenkins' code. We did not try to make a literal translation but rather to use his ideas to produce efficient Fortran code. The resulting program executed between 2 and 4 times slower than ours. A further advantage of our algorithm is that it is simpler and so the code is less bulky (the object code being about  $2/3$  as long). The accuracy of the roots produced by the two algorithms was very comparable except in the last test shown in Table 1 (Jenkins' 6th example) where we obtained much smaller errors (all less than  $2 \times 10^{-14}$ ) because the roots of modulus 0.9 were not all found before those of modulus 1 and therefore ill-conditioned deflated polynomials were not generated. To compare the error bounds with those of Jenkins we ran the examples documented in his report. In his 6th example our bounds were much better simply because the roots had been found so much more accurately. In his 5th example (a complex polynomial of degree 21 having roots of multiplicities 1, 2 and 3) several of his bounds were quite unrealistic and our bounds were all better, most of them by factors over 100 and three (or seven if multiplicities are included) by factors over 1,000. In the remaining examples the differences between the bounds were not severe.

Special purpose subroutines exist in the Harwell library for solving real cubic and quadratic equations (PA03A/AD and PA05A/AD, respectively). They use direct methods involving only the extraction of square and cube roots but sometimes they lose accuracy through unnecessary cancellation. We had hoped that a direct call of PA07BD might be nearly as fast so that

we could withdraw the special purpose subroutines, but the speed difference is by a factor of about 7. It is hoped that better versions of PA03A/AD and PA05A/AD will be written for the library in due course. Extra tests for ensuring reliability are likely to slow down the programs a little, but it seems likely that good programs significantly faster than PA07BD can be written for these special cases.

To test our subroutines we read in sets of roots and a relative error level. We used extended precision arithmetic to construct the corresponding polynomial and then made pseudo-random perturbations to its coefficients at the required relative level. The polynomial and its errors were then handed to our subroutines. This enables us to check the actual errors against the computed error discs. We began by using all the polynomials of Jenkins (1969) and about as many others from local sources, but eventually reduced our test set to those shown in Table 1 (and a few trivial ones designed to explore corners in our code) because the remainder showed no useful additional information. We used the same test data for the real case by adding an additional root consisting of the complex conjugate of any complex root.

Our test results are summarised in Table 1. For each example we show the errors and error bounds obtained for the first and last root found and one intermediate one. This gives the reader an indication of our success in finding the roots in order and we have been able to choose the intermediate root displayed so that the three roots together indicate the full range of conditioning. We also show the times (370/168 secs) for a call of PA06BD/7BD (roots only) and PA06AD/7AD (roots and bounds). It can be seen that finding the bounds usually involves quite a small overhead. The last two cases are exceptional because of the large number of roots which required discs containing many other roots. Such cases are relatively slow



because we try to find a disc with only one root, then try with two, and so on.

It was the complex version of the last example which led us to abandon finding disjoint discs, because some of the roots (e.g. (0,0.9)) are so ill-conditioned that the best disc obtainable covers all the roots. Therefore if we insist on distinct discs all that can be obtained is one disc for all the roots. By using overlapping discs, however, we obtained 12 good error bounds of which two are displayed in Table 1.

The real versions of the third and the last examples led us to abandon Peters' and Wilkinsons' running error analysis on the computed polynomial  $P(z) = \prod(z - \alpha_r)$ . In the last example all the roots are well conditioned but the bounds for the errors in computing  $P(z)$  were so pessimistic that each disc contained all the roots. Example 3 was less dramatic but all the error bounds came to about  $10^{-8}$  instead of about  $10^{-14}$ .

TABLE 1

Test	Roots	Pert. Level	Complex version				Real version					
			Order	Time	Root	Bound	Error	Order	Time	Root	Bound	Error
1	(1, 10-6), 2x(-1, 10-6) (2, 10-8), 2x(-2, 10-8) 5, 2x3, -10	0	10	.024 .030	(-1, 10-6) 3 -10	5.4E-8 2.8E-7 4.9E-15	3.0E-10 3.8E-10 1.1E-15	16	.028 .038	(-1, 0) 3 -10	7.6E-4 4.7E-7 9.2E-15	1.2E-6 4.6E-9 2.2E-15
2	±1, ±1.5, ±2, ±2.5, ..., ±4 ±4.1, ±4.2, ±4.3, ..., ±4.7	10-15	28	.11 .14	1.5 -4.2 4.7	1.1E-11 1.2E-3 3.1E-5	1.8E-13 2.2E-4 4.8E-6	28	.12 .13	1.5 4.3 4.7	1.3E-11 1.3E-3 2.4E-5	5.4E-14 1.6E-4 2.8E-6
3	Jenkins 1*: randomly chosen coefficients*	10-13	18	.067 .087	(.158, 1) (-.209, .900) (-.885, .302)	7.0E-8 6.7E-14 1.7E-9	1.4E-8 3.1E-15 1.1E-10	36	.08 .10	(-.587, .482) (-.579, .72) (-.377, -.598)	8.8E-14 7.7E-14 3.1E-14	4.8E-15 4.6E-15 6.0E-15
4	2 <sup>i</sup> , i = -6, -5, -4, ..., 13	0	20	.050 .063	2 <sup>-6</sup> 2 <sup>2</sup> 2 <sup>12</sup>	2.1E-16 4.2E-13 1.0E-10	1.7E-18 4.8E-14 1.3E-11	20	.037 .038	2 <sup>-6</sup> 2 <sup>2</sup> 2 <sup>13</sup>	2.3E-16 4.9E-13 6.1E-11	1.7E-18 4.0E-14 8.2E-12
5	10-7, 4x1, 3x2, 2x3, 4 (an expansion of Jenkins 4)	10-10	11	.028 .044	10-7 2 4	3.1E-17 7.5E-2 1.3E-5	8.8E-18 1.8E-2 2.6E-6	11	.014 .013	10-7 2 4	2.2E-17 8.3E-2 9.0E-6	2.3E-18 2.7E-2 3.2E-6
6	106, 8, 9, 10, ..., 15	0	9	.017 .019	11 14 106	3.3E-7 1.9E-7 5.7E-10	1.3E-9 7.7E-10 4.4E-11	9	.012 .014	11 14 106	3.2E-7 1.8E-7 5.7E-10	5.5E-11 4.3E-10 4.4E-11
7	5x0, 1, 10x1, 2, 3, 4, 5, 6	10-15	20	.08 .14	0, 1 1 3	9.0E-4 2.9E-1 7.9E-9	2.9E-4 8.4E-2 3.1E-10	20	.04 .07	0, 1 1 5	6.3E-4 2.8E-1 1.1E-9	3.9E-5 8.3E-2 4.3E-11
8	Jenkins 6*: equally spaced on semi- circles centre origin radii .9/1 (left/right half planes)	10-14	31	.37 .72	(0, .9) (-.7794, .4500) (1, 0)	2.0E-0 1.8E-3 8.4E-8	2.7E-2 1.7E-4 1.5E-8	60	.20 .28	(-.8803, -1.871) (.3090, -.5511) (-.4500, .7794)	1.7E-13 3.1E-14 1.6E-13	1.1E-15 6.0E-15 5.5E-15

\* In these cases roots were input to 3/4 significant decimals.

THIS

7. Appendix. Specification sheets and listings

Harwell Subroutine Library

PA06AD/BD

1. Purpose

To find all the roots of a complex polynomial

$$a_1 + a_2x + \dots + a_{n+1}x^n$$

and error bounds for these roots.

2. Argument List

CALL PA06AD(A,N,R,E,W,S,NP1,LW)      (double-length standard)  
 CALL PA06AD(A,N,R,E,W,NP1,W)      (single-length standard)  
 CALL PA06AD(A,N,R,E,W)      (IBM)

- A is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n+1) which must be set by the user so that the real and imaginary parts of  $a_i$  are held in A(1,i), A(2,i). It is unaltered by the subroutine. For the IBM version A may be a COMPLEX\*16 array of length n+1.
- N is an INTEGER variable containing the degree n of the polynomial. Its value must be positive.
- R is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n) used to return the roots. These are held with real and imaginary parts in R(1,i), R(2,i), i=1,2,...,n. The dummy value (1D70,1D70) is returned for each infinite root (corresponding to a zero leading coefficient). For the IBM version R may be a COMPLEX\*16 array of length n.
- E is a REAL array of dimension at least (n+1) which must be set by the user to error bounds on the coefficients, or to zero if these are accurate to machine precision. On exit the first n locations contain approximate bounds on the moduli of the errors in the roots.
- W is a DOUBLE PRECISION (REAL for the single-length version) work array of dimensions (2,LW).

S (double-length standard version only) is a REAL work array of dimensions (4,LW), which may be equivalenced with W.

NP1 (standard versions only) is an INTEGER which must be set to n+1.

LW (standard versions only) is an INTEGER which must be set to at least  $5n/4+2$  (or  $3n/2+2$  for the single-length version).

### 3. Alternative Entry

The error analysis part of a call to PA06AD takes typically about 20% of its time. If speed is important and error bounds are not wanted then a call of the form

or           CALL PA06BD(A,N,R,W,NP1)   (standard versions)  
              CALL PA05BD(A,N,R,W)        (IBM version)

should be made. The arguments are the same as those of the main call, but W need have length only (2,n+1).

### 4. Method

The roots are found by the algorithm of Madsen (BIT(1973) 13, 71-75), the principal features of which are Newton iteration followed by deflation. The error bounds are found by the application of Rouché's theorem as recommended by Wilkinson (J.Inst.Maths Applics.(1971) 8, 16-35) except that discs are always taken with centres on the approximate roots and errors in multiplying out the polynomial  $\Pi(x-R(I))$  are ignored. The disc for each root is such that it contains exactly the same number of approximate roots  $R(I)$  as exact roots of the true polynomial. Note that in the case of true multiple roots the corresponding approximate roots may be quite well separated but each will lie in the disc of all the others and their mean will be a good estimate of the true multiple root.

PA07AD/BD

1. Purpose

To find all the roots of a real polynomial

$$a_1 + a_2x + \dots + a_{n+1}x^n$$

and error bounds for these roots.

2. Argument List

CALL PA07AD(A,N,R,E,W,S,NP1,LW) (double-length standard)  
 CALL PA07AD(A,N,R,E,W,NP1,LW) (single-length standard)  
 CALL PA07AD(A,N,R,E,W) (IBM)

A is a DOUBLE PRECISION (REAL for the single-length version) array of length at least (n+1) which must be set by the user to contain the coefficients and is unaltered by the subroutine.

N is an INTEGER variable containing the degree n of the polynomial. Its value must be positive.

R is a DOUBLE PRECISION (REAL for the single-length version) array of dimensions (2,n) used to return the roots. These are held with real and imaginary parts in R(1,i),R(2,i),i=1,2,...,n. The dummy value (1D70,0D0) is returned for each infinite root (corresponding to a zero leading coefficient). For the IBM version R may be a COMPLEX\*16 array of length n.

E is a REAL array of dimension at least (n+1) which must be set by the user to error bounds on the coefficients, or to zero if these are accurate to machine precision. On exit the first n locations contain approximate bounds on the moduli of the errors in the roots.

W is a DOUBLE PRECISION (REAL for the single-length version) work array of length LW.

S (double-length standard version only) is a REAL work array of dimensions (2,LW), which may be equivalenced with W.

NP1 (standard versions only) is an INTEGER which must be set to  $n+1$ .

LW (standard versions only) is an INTEGER which must be set to  $3n/2+2$  (or  $3n+2$  for the single-length version).

### 3. Alternative Entry

The error analysis part of a call to PA07AD takes typically about 20% of its time. If speed is important and error bounds are not wanted then a call of the form

```
CALL PA07BD(A,N,R,W,NP1) (standard versions)
CALL PA07BD(A,N,R,W)     (IBM version)
```

should be made. The arguments are the same as those of the main call, but W need have length only  $n+1$ .

### 4. Method

The roots are found by the algorithm of Madsen (BIT(1973) 13,71-75), the principal features of which are Newton iteration followed by deflation. The error bounds are found by the application of Rouché's theorem as recommended by Wilkinson (J.Inst.Maths Applics.(1971) 8, 16-35) except that discs are always taken with centres on the approximate roots and errors in multiplying out the polynomial  $\Pi(x-R(I))$  are ignored. The disc for each root is such that it contains exactly the same number of approximate roots  $R(I)$  as exact roots of the true polynomial. Note that in the case of true multiple roots the corresponding approximate roots may be quite well separated but each will lie in the disc of all the others and their mean will be a good estimate of the true multiple root.

```

C SUBROUTINE PA06AD(A,N,R,E,W, NPI,LW)
C SUBROUTINE PA06AD(A,N,R,E,W)
C SUBROUTINE PA06AD(A,N,R,E,W,S,NPI,LW)
C REAL A(2,NPI),W(2,LW),R(2,N)
C DOUBLE PRECISION A(2,1),W(2,1),R(2,N)
C DOUBLE PRECISION A(2,NPI),W(2,LW),R(2,N)
C REAL E(NPI)
C REAL E(1)
C REAL E(NPI),S(4,LW)
C CALL PA06BD(A,N,R,W)
C CALL PA06RD(A,N,R,W,N+1)
C CHECK FOR ZERO LEADING COEFFICIENTS
C M=N+1
C M=M-1
C IF(ABS(A(1,M+1))+ABS(A(2,M+1))).EQ.0.0 .AND. M.GT.0)GO TO 5
C IF(DABS(A(1,M+1))+DABS(A(2,M+1))).EQ.0.0 .AND. M.GT.0)GO TO 5
C N2=N/2
C N4=N/4
C IF(M.GE.1)CALL PA06CD(A,M,M+1,R,E,W,W(1,N+2),W,W(1,N+2))
C IF(M.GE.1)CALL PA06CD(A,M,M+1,R,E,W,W(1,N+2),W,W(1,N+2))
C IF(M.GE.1)CALL PA06CD(A,M,M+1,R,E,W,S(1,N+2),S,S(1,N+2))
C SET DUMMY DISCS CORRESPONDING TO INFINITE ROOTS
C IF(M.EQ.N)GO TO 20
C M=M+1
C DO 10 I=M,N
C10 E(I)=ABS(R(1,I))+ABS(R(2,I))
C10 E(I)=DABS(R(1,I))+DABS(R(2,I))
C20 CONTINUE
C RETURN
C END

```

S/  
I/

S/  
I/

S/  
I/

I/

S/

S/

S/

S/

PROGRAM UNIT PA06AD

ARGUMENTS A N R E W S NPI LW

NAME TYPE USE ATTRIBUTES REFERENCES

A	ED	V	A A2	1	2								
DABS	ED	IF		7	13			4	7	9			
E	ER	V	ASAI	1	3			9	13				
I	I	V	SS	12	13								
LW	I	V	A S	1	2			3					
M	I	V	SS	5	6			7	9	10	11	12	
N	I	V	A S	1	2			4	5	8	9	10	12
NPI	I	V	A S	1	2			4					
N4	I	V	SS	8	9			3					
PA06AD		SN		4									
PA06CD		SN		9									
R	ED	V	A A2	1	2			4	9	13			
S	ER	V	A A2	1	3			9					
W	ED	V	A A2	1	2			4	9				
10				12	13								
20				10	14								
5				6	7								



```

1 C SURROUTINE PA06BD(A1,M,ROOT,A) I/
2 C SUBROUTINE PA06BD(A1,M,ROOT,A,MPI) S/
3 C REAL Z(2),FOZ(2),Z(2),DZ(2),FIZ(2),FZ(2),W(2), S/
4 C 1FW(2),A(2,MPI),A1(2,MPI),ROOT(2,M),DZK(2) S/
5 C DOUBLE PRECISION ZO(2),FOZ(2),Z(2),DZ(2),FIZ(2),FZ(2),W(2), I/
6 C 1FW(2),A(2,1),A1(2,1),ROOT(2,M),DZK(2) I/
7 C DOUBLE PRECISION ZO(2),FOZ(2),Z(2),DZ(2),FIZ(2),FZ(2),W(2),
8 C 1FW(2),A(2,MPI),A1(2,MPI),ROOT(2,M),DZK(2)
9 C ALL THE ABOVE ARRAYS HOLD COMPLEX VARIABLES AND COMPLEX ARRAYS AND
10 C WILL BE REFERRED TO IN COMMENTS AS SUCH, WITH NO MENTION OF THE FIRST
11 C SUBSCRIPT
12 C
13 C WHEN (M-N) ZEROS ARE FOUND, A(1),... ,A(N+1), WILL HOLD THE
14 C COEFFICIENTS OF THE DEFLATED POLYNOMIAL. ROOT(N+1),... ,ROOT(M)
15 C WILL HOLD THE ZEROS FOUND. A1(1),... ,A1(N) WILL HOLD THE
16 C COEFFICIENTS OF THE DERIVATIVE OF THE DEFLATED POLYNOMIAL.
17 C ROOT(1),... ,ROOT(N),A1(N+1),... ,A1(M+1) HOLD COEFFICIENTS OF
18 C THE ORIGINAL POLYNOMIAL.
19 C REAL FO,FF,F,FA,PA06DD,DZ1 S/
20 C DOUBLE PRECISION FO,FF,F,FA,PA06DD,DZ1
21 C LOGICAL STAGE1,DIV2
22 C STAGE1 IS .TRUE. DURING STAGE1 OF THE ITERATION AND .FALSE.
23 C DURING STAGE2. DIV2 IS .TRUE. IF THE SEARCH WITH STEP-LENGTHS
24 C DZ,DZ/2,... IS IN USE AND .FALSE. IF THE SEARCH WITH STEPS DZ,2*DZ,...
25 C IS IN USE. S/
26 C DATA RIG/1.E70/,SMALL/1.E-70/,PASE/16./,EPS/1.E-6/ S/
27 C DATA RIG/1.E70/,SMALL/1.E-70/,PASE/16./,EPS/2.3E-16/
28 C RIG,SMALL ARE NUMBERS NEAR OVERFLOW/UNDERFLOW LIMITS. PASE IS THE BASE
29 C OF THE FLOATING POINT ARITHMETIC IN USE. EPS IS THE SMALLEST
30 C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
31 C IN USE.
32 C SSMALL=SQRT(SMALL)
33 C ALOGREALOG(PASE)
34 C N=M
35 C STORE ORIGINAL POLYNOMIAL IN A AND IN ROOT.
36 C
37 C J=M+1
38 C A(1,J)=A1(1,1)
39 C A(2,J)=A1(2,1)
40 C DO 10 I=1,M
41 C ROOT(1,I)=A1(1,I)
42 C ROOT(2,I)=A1(2,I)
43 C A(1,I)=A1(1,J)
44 C A(2,I)=A1(2,J)

```

```

16 A(2,I)=A1(Z,J)
17 J=J-1
C
C
C
18 TEST FOR ZEROS AT INFINITY
19 IF(ABS(A(1,I))+ABS(A(2,I)).GT.0.0)GO TO 40
20 IF(DABS(A(1,I))+DABS(A(2,I)).GT.0.0)GO TO 40
21 DO 30 I=1,N
22 A(1,I)=A(1,I+1)
23 A(2,I)=A(2,I+1)
24 ROOT(1,N)=BIG
25 ROOT(2,N)=BIG
26 N=N-1
27 IF (N.GT.0) GO TO 20
28 GO TO 310
40 IF (N.LE.1) GO TO 260
NI=N+1
C
C SCALE THE COEFFICIENTS
29 U1=0.
30 U2=BIG
31 DO 50 K=1,NI
32 U=ABS(A(1,K))+ABS(A(2,K))
33 U=DABS(A(1,K))+DABS(A(2,K))
34 IF(U.LE.0.)GO TO 50
35 IF (U.GT.U1) U1=U
36 IF (U.LT.U2) U2=U
37 50 CONTINUE
38 U=SQRT(U1)*SQRT(U2)
39 I=-ALOG(U)/ALOG8
40 U=BASE**I
41 DO 70 K=1,N
42 A(1,K)=A(1,K)*U
43 A(2,K)=A(2,K)*U
44 A(1,K)=A(1,K)*FLOAT(N1-K)
45 A(2,K)=A(2,K)*FLOAT(N1-K)
46 A(1,N1)=A(1,N1)*U
47 A(2,N1)=A(2,N1)*U
C
C TEST FOR ZEROS AT (0.,0.)
47 Z(1)=0.
48 Z(2)=0.
C
49 IF(ABS(A(1,N1))+ABS(A(2,N1)).LE.SSMALL)GO TO 290
50 IF(DABS(A(1,N1))+DABS(A(2,N1)).LE.SSMALL)GO TO 290
Z(1)=0.
S/
S/
S/

```

```

51      ZO(2)=0.
52      FO=A(1,N1)**2+A(2,N1)**2
53      FMIN=FO*(FLOAT(N)*16.*EPS)**2

C THE FOLLOWING QUANTITIES ARE HELD AT THE START OF EACH ITERATION
C Z IS THE CURRENT POINT, F=CABS(F(Z))**2
C ZO IS THE LAST POINT, F0Z=F*(ZO), F0=CABS(F(ZO))**2
C ZO=3*CABS(Z-ZO)
C DZ IS THE LAST TENTATIVE STEP IF THE LAST ITERATION WAS SUCCESSFUL OR
C ELSE IS THE REQUIRED NEXT TENTATIVE STEP. ON FIRST ITERATION IT IS
C SET TO Z.
C FF=CABS(F(ZT))**2 WHERE ZT IS THE LAST TENTATIVE POINT.
C
C SET INITIAL ITERATES AND QUANTITY USED IN THE CONVERGENCE TEST.
54      FF=FO
55      UO=FO
56      T=BIG
57      DO 80 K=1,N
58      U=A(1,K)**2+A(2,K)**2
59      IF (U.EQ.UO.) GOTO 80
60      U=ALOG(UO/U)/FLOAT(2*(N1-K))
61      IF (U.LT.T) T=U
62      80 CONTINUE
63      T=EXP(T)
64      FOZ(1)=A(1,N)
65      FOZ(2)=A(2,N)
66      Z(1)=1.
67      Z(2)=0.
68      IF( ABS(FOZ(1))+ ABS(FOZ(2)) .LE. 0.) GO TO 100
69      IF(DABS(FOZ(1))+DABS(FOZ(2)) .LE. 0.) GO TO 100
70      Z=-A(N1)/A(N)
71      CALL PA06ED(A(1,N1),A(1,N),Z)
72      U=0.5*T/( ABS(Z(1))+ ABS(Z(2)))
73      U=0.5*T/(DABS(Z(1))+DABS(Z(2)))
74      Z(1)=Z(1)*U
75      Z(2)=Z(2)*U
76      DZ(1)=Z(1)
77      DZ(2)=Z(2)
78      F=PA06DD(Z,FZ,N*1,A)
79      R0=0.5*T

C CALCULATION OF THE TENTATIVE STEP DZ AND WHETHER IN STAGE1.
C THIS IS WHERE THE ITERATION STARTS IF THE PREVIOUS ONE WAS SUCCESSFUL.
120     U=PA06DD(Z,FIZ,N ,A1)

```

```

78      IF (U<.50<.0.) GOTO 140
79      DZ=-FZ/FIZ
80      CALL PA06ED(FZ,FIZ,DZ)
81      F2=((FOZ(1)-FIZ(1))*2+(FOZ(2)-FIZ(2))*2)/
      ((ZO(1)-Z(1))*2+(ZO(2)-Z(2))*2)
82      STAGE1=F*F2/U.GT.U*0.25.OR.F.NE.FF
83      R=ABS(DZ(1))+ABS(DZ(2))
84      R=DABS(DZ(1))+DABS(DZ(2))
85      IF(R.LE.R0*3.)GO TO 150
86      DZ1=DZ(1)
87      DZ(1)=(DZ1*1.8-DZ(2)*2.4)*R0/R
88      DZ(2)=(DZ1*2.4+DZ(2)*1.8)*R0/R
89      GOTO 150
90      DZ1=DZ(1)
91      DZ(1)=DZ1*1.8-DZ(2)*2.4
92      DZ(2)=DZ1*2.4+DZ(2)*1.8
93      STAGE1=.TRUE.
      FOZ(1)=FIZ(1)
      FOZ(2)=FIZ(2)

94      C
95      C FIND NEXT POINT IN THE ITERATION. THIS IS WHERE THE ITERATION STARTS
96      C IF THE PREVIOUS ONE WAS UNSUCCESSFUL.
97      ZO(1)=Z(1)
98      ZO(2)=Z(2)
99      FC=F
100     DZK(1)=DZ(1)
101     DZK(2)=DZ(2)
102     Z(1)=ZO(1)+DZ(1)
103     Z(2)=ZO(2)+DZ(2)
104     IF (ABS(Z(1)).LT.EPS*ABS(Z(2)))Z(1)=0.
105     IF (DABS(Z(1)).LT.EPS*DABS(Z(2)))Z(1)=0.
106     IF (ABS(Z(2)).LT.EPS*ABS(Z(1)))Z(2)=0.
107     IF (DABS(Z(2)).LT.EPS*DABS(Z(1)))Z(2)=0.
108     W(1)=Z(1)
109     W(2)=Z(2)
110     F=PA06DD(Z,FZ,N+1,A)
111     FF=F
112     IF (.NOT.STAGE1) GOTO 240
113     J=1
114     DIV2=F.GE.F0
115     IRO IF (DIV2) GOTO 190
116     W(1)=W(1)+DZ(1)

```

```

112 W(2)=W(2)+DZ(2)
113 GOTO 200
114 DZ(1)=DZ(1)*0.5
115 DZ(2)=DZ(2)*0.5
116 W(1)=Z0(1)+DZ(1)
117 W(2)=Z0(2)+DZ(2)
118 FA=PA06DD(W,FW,N+1,A)
119 IF (FA.GE.F) GOTO 240
120 F=FA
121 FZ(1)=FW(1)
122 FZ(2)=FW(2)
123 Z(1)=W(1)
124 Z(2)=W(2)
125 J=J+1
126 IF (DIV2.AND.(J.EQ.3)) GOTO 220
127 IF (J.LE.N) GOTO 180
128 GOTO 240
129 DZ1=DZ(1)
130 DZ(1)=DZ1*0.6-DZ(2)*0.8
131 DZ(2)=DZ1*0.8+DZ(2)*0.6
132 Z(1)=Z0(1)+DZ(1)
133 Z(2)=Z0(2)+DZ(2)
134 F=PA06DD(Z,FZ,N+1,A)
135 C END OF STAGE1 SEARCH
136 C RO= ABS(Z0(1)-Z(1))+ ABS(Z0(2)-Z(2))
137 C RO=DARS(Z0(1)-Z(1))+DABS(Z0(2)-Z(2))
138 C CONVERGENCE TEST.
139 IF (F.LT.F0) GO TO 250
140 Z(1)=Z0(1)
141 Z(2)=Z0(2)
142 R1= ABS(Z(1))+ ABS(Z(2))
143 R1=DARS(Z(1))+DABS(Z(2))
144 IF (R0.LT.FPS*R1) GO TO 270
145 IF (F.LT.F0) GOTO 120
146 F=F0
147 IF (F.LE.FMIN) GO TO 270
148 DZ(1)=DZK(1)*(-0.3)-DZK(2)*(-0.4)
149 DZ(2)=DZK(1)*(-0.4)+DZK(2)*(-0.3)
150 STAGE1=.TRUE.
151 GOTO 160
152 C
153 C DEAL WITH N=1 CASE
154 Z=-A(2)/A(1)

```

S/

S/

```

148
149
150
151
152
153
154
155
156
157
158
159
160

260 CALL PA06ED(A(1,2),A(1,1),Z)
    GO TO 290
C
C DEFLATE, STORE ROOT, RESTORE COEFFICIENT OF ORIGINAL POLY AND REDUCE N
270 DO 280 K=2,N
    A(1,K)=A(1,K-1)*Z(1)-A(2,K-1)*Z(2)+A(1,K)
    A(2,K)=A(1,K-1)*Z(2)+A(2,K-1)*Z(1)+A(2,K)
280
290 A(1,N)=ROOT(1,N)
    A(2,N)=ROOT(2,N)
    ROOT(1,N)=Z(1)
    ROOT(2,N)=Z(2)
    N=N-1
    IF(N-1)310,260,40
310 RETURN
    END

```

PROGRAM UNIT PA068D

ARGUMENTS A1 M ROOT A MPI

NAME TYPE USE ATTRIBUTES REFERENCES

A	ED	V	ASA2	1	2	10	11	15	16	18	20
				21	32	41	42	43	44	45	46
				49	52	58	64	65	69	75	105
				118	134	148	151	152			
				7	38	60					
ALOG	ER	FN	SS	7	38						
ALOGB	R	V	ASA2	1	2	10	11	13	14	15	16
AL	ED	V		43	44	77	153	154			
BASE	R	V	SS	5	7	39					
BIG	R	V	SS	5	22	23	30	56			
DABS	ED	IF		18	32	49	68	70	82	101	102
				135	139						
DIV2	EL	V	SS	4	109	110	126				
DZ	ED	V	SAL	2	73	74	79	82	84	85	86
				88	89	90	97	98	99	100	111
				112	114	115	116	117	129	130	131
				132	133	144	145				
DZK	ED	V	SAL	2	97	98	144	145			
DZ1	ED	V	SS	3	84	85	86	88	89	90	129
				130	131						
EPS	R	V	SS	5	53	101	102	140			
EXP	ER	FN		63							
F	ED	V	SS	3	75	81	96	105	106	109	119
				120	134	136	141	142	143		
FA	ED	V	SS	5	118	119	120				
FF	ED	V	SS	3	54	81	106				
FLOAT	R	IF		43	44	53	60				
FMIN	R	V	SS	53	143						
FW	ED	V	AI	2	118	121	122				
FZ	ED	V	SAL	2	75	79	105	121	122	134	136
FO	ED	V	SS	3	52	53	54	55	96	109	
				141	142						
FOZ	ED	V	SAL	2	64	65	68	80	92	93	
FLZ	ED	V	AI	2	77	79	80	92	93		
FZ	R	V	SS	80	81						

**THIS  
PAGE  
IS  
MISSING  
IN  
ORIGINAL  
DOCUMENT**

*36-37*





```

1  SURROUTINE PA06CD(A,N,NP1,R,E,W,F,IG,CR)
2  C IG AND CR MAY BE DYNAMICALLY EQUIVALENCED WITH SEPARATE PARTS OF W
3  C REAL A(2,NP1),R(2,N),W(2,NP1),C(2)
4  C DOUBLE PRECISION A(2,NP1),R(2,N),W(2,NP1),C(2)
5  C REAL E(NP1),F(NP1),CR(N),Z(2),AI(2),IG(N)
6  C W IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
7  C OF THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS.
8  C F IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
9  C OF THE ERROR POLYNOMIAL.
10 C IG IS A WORKSPACE ARRAY OF LENGTH N USED TO LINK TOGETHER THE ROOTS
11 C IN A GROUP. THE FIRST IS IG1, IG(K) FOLLOWS K AND THE LAST IS I.
12 C OTHER ROOTS HAVE IG(K)=0.
13 C CR IS A WORKSPACE ARRAY OF LENGTH N USED TO HOLD DISTANCES FROM ROOT I
14 C TO TH REST.
15 C DATA EPS/1.E-6/,BIG/1.E70/
16 C DATA EPS/2.3E-16/,BIG/1.E70/
17 C BIG IS A NUMBER NEAR THE OVERFLOW LIMIT. EPS IS THE SMALLEST
18 C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
19 C IN USE.
20 C FACT=1.05**(1./FLOAT(N))
21 C NI=N+1
22 C
23 C MULTIPLY OUT THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS
24 C
25 W(1,1)=A(1,N1)
26 W(2,1)=A(2,N1)
27 DO 10 I=1,N
28 W(1,I+1)=0.
29 W(2,I+1)=0.
30 II=N1-I
31 C(1)=-R(1,II)
32 C(2)=-R(2,II)
33 DO 10 JJ=1,I
34 J=I+1-JJ
35 W(1,J+1)=C(1)*W(1,J)-C(2)*W(2,J)+W(1,J+1)
36 W(2,J+1)=C(1)*W(2,J)+C(2)*W(1,J)+W(2,J+1)
37 C
38 C FIND COEFFICIENTS OF ERROR POLYNOMIAL
39 DO 20 I=1,NI
40 II=N+2-I
41 AI(1)=A(1,II)
42 AI(2)=A(2,II)
43 Z(1)=A(1,II)-W(1,I)
44 Z(2)=A(2,II)-W(2,I)

```

```

25 F(I)=CABS(CMPLX(Z(1),Z(2)))+AMAX1(E(I),CABS(CMPLX(AI(1),AI(2)))
26 1*EPS)
27 DO 30 I=1,N
28 IG(I)=0
29 C
30 C MAIN ERROR-BOUNDING LOOP STARTS HERE AND EXTENDS TO THE END
31 C OF THE SUBROUTINE IF FINDS A BOUND FOR ROOT I.
32 DO 130 I=1,N
33 M=1
34 IG(I)=-1
35 IGI=I
36 C(1)=R(1,I)
37 C(2)=R(2,I)
38 D=0.
39 C
40 C FIND DISTANCES TO OTHER ROOTS
41 DO 32 K=1,N
42 Z(1)=R(1,K)-C(1)
43 Z(2)=R(2,K)-C(2)
44 CR(K)=CABS(CMPLX(Z(1),Z(2)))
45 C
46 C INITIALIZATION STATEMENTS WHICH ARE ALSO NEEDED WHEN M IS INCREASED
47 DMIN=BIG
48 Z(1)=C(1)
49 Z(2)=C(2)
50 S=CABS(CMPLX(Z(1),Z(2)))+D
51 RAD=0
52 C TEST ROUCHE CONDITION WITH RADIUS RAD.
53 TOP=F(1)
54 PROD=1.
55 Z(1)=A(1,N1)
56 Z(2)=A(2,N1)
57 BTM=CABS(CMPLX(Z(1),Z(2)))
58 DO 80 K=1,N
59 TOP=S*TOP+F(K+1)
60 DIST=CR(K)
61 IF(IG(K).NE.0)GO TO 70
62 BTM=BTM*(DIST-RAD)
63 IF(DMIN.LE.DIST)GO TO 80
64 OMIN=DIST
65 L=K
66 GO TO 80
67 PROD=PROD*(RAD-DIST)
68 CONTINUE
69
70
80

```

```

60 IF(BTM.EQ.0.0)GO TO 100
61 TR= ABS(TOP/BTM)
62 IF(PROD.GE.TB)GO TO 110
C FIND A NEW TRIAL RADIUS.
63 OLDS=S
64 OLDR=RAD
65 RL=RAC
66 RAD=D+1.1*TR
67 IF (M.EQ.1)GO TO 98
68 RM=D+(1.1*TR)**(1./FLOAT(M))
C BISECTION LOOP. RL AND RM HOLD UPPER AND LOWER BOUNDS.
83 RAD=(RL+RM)/2.
84 PROD=1.
85 K=IGI
90 PROD=PROD*(RAD-CR(K))
86 K=FIX(IG(K))
87 IF(K.GT.0)GO TO 90
88 IF(PROD.GE.1.10*TB)GO TO 94
89 IF(RAD.GE.DMIN)GO TO 100
90 IF(PROD.GT.1.05*TB)GO TO 98
91 RL=RAD
92 GO TO 83
93 RM=RAD
94 GO TO 83
95 S=S+RAD-OLDR
96 IF(RAD.GE.DMIN)GO TO 100
97 IF( S*((DMIN-OLDR)/(DMIN-RAD)).LE.FACT*OLDS)GO TO 110
98 GO TO 45
C ADD ROOT TO GROUP
100 M=M+1
101 IGI(L)=FLOAT(IGI)
102 IGI=L
103 D=DMIN
104 GO TO 40
C STORE ERROR_BOUND AND RESET IG
110 E(I)=RAD
111 K=IGI
112 IGI=FIX(IG(IGI))
113 IG(K)=0.
114 IF(K.NE.1)GO TO 120
115 CONTINUE
116 RETURN
117 END

```

PROGRAM UNIT PA06CD

ARGUMENTS	A	N	NPI	R	E	W	F	IG	CR
A	ED	V	A	A2	1	2	47		
ABS	R	IF			46				
AI	ER	V	SAI		61				
AMAX1	R	IF			3	21			
BIG	R	V	SS		25				
BTM	R	V	SS		4	39			
C	ED	V	SAI		48	13			
					2	13			36
					31	40			
CARS	ER	FN			41				
CMPLX	EC	IF			42	48			
CR	ER	V	ASAI		42	46			
D	R	V	SS		38	3			
DIST	R	V	SS		38	3			
DMIN	R	V	SS		42	89			
E	ER	V	ASAI		51	68			
EPS	R	V	SS		54	58			89
F	ER	V	ASAI		39	54			
FACT	R	V	SS		1	3			
FLOAT	R	IF			4	25			
I	I	V	SS		1	3			
					25	44			
IFIX	I	IF			25	50			
IG	ER	V	ASAI		87				
IG1	I	V	SS		11	15			20
II	I	V	SS		22	24			31
					32	33			
J	I	V	SS		72	93			
JJ	I	V	SS		1	3			
K	I	V	SS		94				
L	I	V	SS		31	71			
M	I	V	SS		12	13			
					25				
					16	17			
					15	16			
					35	36			
					56	71			
					56	87			
					29	67			
					37	38			52
					72	73			95
					88	88			
					14	20			
					18				
					27	30			
					27	73			93
					87				
					11	16			20
					25	28			31
					91	95			
					30	52			
					27	87			
					87	93			
					14	22			24
					37	49			
					72	74			
					88	92			
					68	86			

N	I	V	A S	1	2	3	5	6	9	20	26
NP1	I	V	A S	28	3E	49					
N1	I	V	A S	1	2	3					
OLDR	I	V	SS	6	7	8					
OLDS	R	V	SS	64	82	84	12	19	46	47	
PROD	R	V	SS	63	84						
R	R	V	SS	45	58	62	70	72	75	77	
RAD	ED	V	A A2	1	2	13	14	32	33	36	37
	R	V	SS	43	53	58	64	65	66	69	72
	R	V	SS	76	78	80	82	83	84	91	
RL	R	V	SS	65	69	78					
RM	R	V	SS	68	69	80					
S	R	V	SS	42	50	63	82	84			
TB	R	V	SS	61	62	66	68	75	77		
TOP	R	V	SS	44	50	61					
W	ED	V	ASA2	1	2	7	8	10	11	17	18
Z	ER	V	SA1	23	24	24	25	36	37	38	40
10				3	23	24	25	36	37	38	
100				41	42	46	47	48			
110				9	15	18					
120				60	76	83	86				
130				62	84	91					
20				92	95						
30				28	96						
32				19	25						
40				26	27						
45				35	38						
70				39	90						
80				44	85						
83				52	58		59				
90				49	54	57					
94				69	79	81					
98				72	74						
				75	80						
				67	77	82					

```

1      REAL FUNCTION PA06DD(Z,FZ,N1,A)
2      DOUBLE PRECISION FUNCTION PA06DD(Z,FZ,N1,A)
3      REAL Z(2),FZ(2),A(2,N1),P,Q,R,X,Y
4      DOUBLE PRECISION Z(2),FZ(2),A(2,N1),P,Q,R,X,Y
5      Z AND FZ CONTAIN COMPLEX NUMBERS.
6      C A CONTAINS COMPLEX POLYNOMIAL COEFFICIENTS.
7      C POLYNOMIAL VALUE AT Z IS SET IN FZ AND THE SQUARE OF ITS MODULUS
8      C IS RETURNED AS FUNCTION VALUE.
9      N=N1-1
10     X=Z(1)
11     Y=Z(2)
12     P=A(1,1)
13     Q=A(2,1)
14     DO 10 I=1,N
15     R=P*X-Q*Y+A(1,I+1)
16     Q=Q*X+P*Y+A(2,I+1)
17     P=R
18     FZ(1)=P
19     FZ(2)=Q
20     PA06DD=P*P+Q*Q
21     RETURN
22     END

```

NAME TYPE USE ATTRIBUTES REFERENCES

NAME	TYPE	USE	ATTRIBUTES	REFERENCES
A	V		A A2	1 2
FZ	V		ASAI	1 2
I	V		SS	8 9 10
N	V		SS	3 8
N1	V		A S	1 2
P	V		SS	2 6 9
PA06DD	FN		S	1 14
Q	V		SS	2 7 9 11 12 14
R	V		SS	2 9 10 13 14
X	V		SS	2 4 9 10
Y	V		SS	2 5 9 10
Z	V		A A1	1 2 4 5
10				8 11

```

1      SUBROUTINE PA06ED(A,B,Q)
2      REAL A(2),B(2),Q(2),U,A1,C,D,E
3      DOUBLE PRECISION A(2),B(2),Q(2),U,A1,C,D,E
4      C IMPLEMENTS THE COMPLEX DIVISION Q=-A/B
5      U=ABS(B(1))+ABS(B(2))
6      U=DARS(B(1))+DARS(B(2))
7      A1=A(1)
8      C=B(1)/U
9      D=B(2)/U
10     E=-(C*C+D*D)*U
11     Q(1)=(A1*C+A(2)*D)/E
12     Q(2)=(A(2)*C-A1*D)/E
13     RETURN
14     END
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

PROGRAM UNIT PA06ED

ARGUMENTS A R Q

NAME TYPE USE ATTRIBUTES REFERENCES

NAME	TYPE	USE	ATTRIBUTES	REFERENCES
A	ED	V	A A1	1
A1	ED	V	SS	2
B	ED	V	A A1	1
C	ED	V	SS	2
D	ED	V	SS	2
DARS	ED	IF		3
E	ED	V	SS	2
Q	ED	V	AS A1	1
U	ED	V	SS	2

```

1 C SUBROUTINE PAOTAD(A,N,R,E,W,NP1,LW)
2 C SUBROUTINE PAOTAD(A,N,R,E,W)
3 C SUBROUTINE PAOTAD(A,N,R,E,W,S,NP1,LW)
4 C REAL A(NP1),W(LW),R(2,N)
5 C DOUBLE PRECISION A(1),W(1),R(2,N)
6 C DOUBLE PRECISION A(NP1),W(LW),R(2,N)
7 C REAL E(NP1)
8 C REAL E(1)
9 C REAL E(NP1),S(2,LW)
10 C CALL PAOTBD(A,N,R,W)
11 C CALL PAOTBD(A,N,R,W,N+1)
12 C CHECK FOR ZERO LEADING COEFFICIENTS
13 C M=N+1
14 C M=M-1
15 C IF( ABS(A(M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5
16 C IF( DARS(A(M+1)).EQ.0.0 .AND. M.GT.0)GO TO 5
17 C N2=N+2+1
18 C N2=N/2
19 C IF(M.GE.1)CALL PAOTCD(A,M,M+1,R,E,W,W(N+2),W,W(N2+2))
20 C IF(M.GE.1)CALL PAOTCD(A,M,M+1,R,E,W,W(N+2),W,W(N2+2))
21 C IF(M.GE.1)CALL PAOTCD(A,M,M+1,R,E,W,S(1,N+2),S,S(1,N2+2))
22 C SET DUMMY DISCS CORRESPONDING TO INFINITE ROOTS
23 C IF(M.EQ.N)GO TO 20
24 C M=M+1
25 C DO 10 I=M,N
26 C E(I)=ABS(R(1,I))+ABS(R(2,I))
27 C E(I)=DARS(R(1,I))+DARS(R(2,I))
28 C CONTINUE
29 C RETURN
30 C END

```



PROGRAM UNIT PA07AD

ARGUMENTS A N R E W S NP1 LW

NAME TYPE USE ATTRIBUTES REFERENCES

A	ED	V	A A1	1	2	4	7	9		
DABS	ED	IF		7	13					
E	ER	V	ASAI	1	3	9	13			
I	I	V	SS	12	13					
LW	I	V	A S	1	2	3				
M	I	V	SS	5	6	7	9	10	11	12
N	I	V	A S	1	2	4	5	8	9	10
NP1	I	V	A S	1	2	3				
N2	I	V	SS	8	9					
PA07BD		SN		4						
PA07CD		SN		9						
R	ED	V	A A2	1	2	4	9	13		
S	ER	V	A A2	1	3	5				
W	ED	V	A A1	1	2	4	9			
10				12	13					
20				10	14					
5				6	7					

```

1 /
C SUBROUTINE PA07BD(A1,M,ROOT,A)
C SUBROUTINE PA07RD(A1,M,ROOT,A,MPI)
C REAL Z0(2),F0Z(2),Z(2),DZ(2),FIZ(2),FZ(2),W(2),
C 1 FW(2),ROOT(2,M),DZK(2)
C DOUBLE PRECISION Z0(2),F0Z(2),Z(2),DZ(2),FIZ(2),FZ(2),W(2),
C 1 FW(2),ROOT(2,M),DZK(2)
C ALL THE ABOVE ARRAYS HOLD COMPLEX VARIABLES AND COMPLEX ARRAYS AND
C WILL BE REFERRED TO IN COMMENTS AS SUCH, WITH NO MENTION OF THE FIRST
C SUBSCRIPT
C
C WHEN (M-N) ZEROS ARE FOUND, A(I), ..., A(N+1), WILL HOLD THE
C COEFFICIENTS OF THE DEFLATED POLYNOMIAL. ROOT(N+1), ..., ROOT(M)
C WILL HOLD THE ZEROS FOUND. A(I), ..., A(N) WILL HOLD THE
C WILL HOLD THE ZEROS FOUND. A(I), ..., A(N) WILL HOLD THE
C COEFFICIENTS OF THE DERIVATIVE OF THE DEFLATED POLYNOMIAL.
C ROOT(1), ..., ROOT(N), A(N+1), ..., A(M+1) HOLD COEFFICIENTS OF
C THE ORIGINAL POLYNOMIAL.
C REAL FO,FF,F,FA,PA07DD,DZ1,A1(MP1),A(MP1),FC,FD,G,AC
C 1,PP,QQ,RR,SS,TT
C DOUBLE PRECISION FO,FF,F,FA,PA07DD,DZ1,A1( 1),A( 1),FC,FD,G
C 1,AC,PP,QQ,RR,SS,TT
C DOUBLE PRECISION FO,FF,F,FA,PA07DD,DZ1,A1(MP1),A(MP1),FC,FD,G
C 1,AC,PP,QQ,RR,SS,TT
C LOGICAL STAGE1, DIV2
C STAGE1 IS .TRUE. DURING STAGE1 OF THE ITERATION AND .FALSE.
C DURING STAGE2. DIV2 IS .TRUE. IF THE SEARCH WITH STEP-LENGTHS
C DZ,DZ/2, ... IS IN USE AND .FALSE. IF THE SEARCH WITH STEPS DZ,2*DZ, ...
C IS IN USE.
C DATA BIG/1.E70/,SMALL/1.E-70/,BASE/16./,EPS/1.0E-6/
C DATA BIG/1.0E70/,SMALL/1.0E-70/,BASE/16./,EPS/2.3E-16/
C BIG,SMALL ARE NUMBERS NEAR OVERFLOW/UNDERFLOW LIMITS. BASE IS THE BASE
C OF THE FLOATING POINT ARITHMETIC IN USE. EPS IS THE SMALLEST
C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
C IN USE.
C
C SSMALL=SQRT(SMALL)
C ALOGB=ALOG(BASE)
C N=M
C
C STORE ORIGINAL POLYNOMIAL IN A AND IN ROOT.
C
C J=M+1
C A(J)=A(1)
C DO 10 I=1,M
C ROOT(I,I)=A(I)

```

```

13 A(I)=A(J)
14 J=J-1
C
C TEST FOR ZEROS AT INFINITY
C20 IF (ABS(A(1)).GT.0.)GO TO 40
20 IF (DABS(A(1)).GT.0.)GO TO 40
DO 30 I=1,N
30 A(I)=A(I+1)
ROOT(1,N)=RIG
ROOT(2,N)=0.
N=N-1
IF (N.GT.0) GOTO 20
GOTO 310
40 IF (N.LE.1) GOTO 260
N1=N+1
C
C SCALE THE COEFFICIENTS
U1=0.
U2=RIG
DO 50 K=1,N1
U=ARS(A(K))
U=DABS(A(K))
IF(U.LE.0.)GO TO 50
IF (U.GT.U1) U1=U
IF (U.LT.U2) U2=U
50 CONTINUE
U=SQRT(U1)*SQRT(U2)
I=-ALOG(U)/ALOGB
U=BASE**I
DO 70 K=1,N
A(K)=A(K)*U
70 A(N1)=A(N1)*U
C
C TEST FOR ZEROS AT (0.,0.)
Z(1)=0.
Z(2)=0.
IF (ABS(A(N1)).LE.SSMALL)GO TO 290
IF (DABS(A(N1)).LE.SSMALL)GO TO 290
Z(1)=0.
Z(2)=0.
FO=A(N1)**2
FMIN=FO*(FLOAT(N)*16.*EPS)**2
C

```

```

C THE FOLLOWING QUANTITIES ARE HELD AT THE START OF EACH ITERATION
C Z IS THE CURRENT POINT, F=CABS(F(Z))**2
C ZO IS THE LAST POINT, FOZ=F(ZO), FO=CABS(F(ZO))**2
C RO=3*CABS(Z-ZO)
C DZ IS THE LAST TENTATIVE STEP IF THE LAST ITERATION WAS SUCCESSFUL OR
C ELSE IS THE REQUIRED NEXT TENTATIVE STEP. ON FIRST ITERATION IT IS
C SET TO Z.
C FF=CABS(F(ZT))**2 WHERE ZT IS THE LAST TENTATIVE POINT.
C
C SET INITIAL ITERATES AND QUANTITY USED IN THE CONVERGENCE TEST.
C
47 C      FF=FO
48 C      UO= ABS(A(N1))
49 C      UO=DARS(A(N1))
50 C      T=BIG
51 C      DC 80 K=1,N
52 C      U= ARS(A(K))
53 C      U=DARS(A(K))
54 C      IF (U.EQ.0.) GOTO 80
55 C      U=ALOG(UO/U)/FLOAT(N1-K)
56 C      IF (U.LT.T) T=U
57 C      80 CONTINUE
58 C      T=EXP(T)
59 C      FOZ(1)=A(N)
60 C      FOZ(2)=0.
61 C      Z(1)=0.5*T
62 C      IF(A(N).LT.0.)Z(1)=-Z(1)
63 C      DZ(1)=Z(1)
64 C      DZ(2)=0.
65 C      F=PA07DD(Z,FZ,N+1,A)
66 C      RO=0.5*T
67
68 C      CALCULATION OF THE TENTATIVE STEP DZ AND WHETHER IN STAGE1.
69 C THIS IS WHERE THE ITERATION STARTS IF THE PREVIOUS ONE WAS SUCCESSFUL.
70 C
71 C      U=PA07DD(Z,FIZ,N ,A1)
72 C      IF (U.EQ.0.) GOTO 140
73 C      DZ=-FZ/FIZ
74 C      CALL PA07ED(FZ,FIZ,DZ)
75 C      F2=((FOZ(1)-FIZ(1))**2+(FOZ(2)-FIZ(2))**2)/
76 C      1 ((ZO(1)-Z(1))**2+(ZO(2)-Z(2))**2)
77 C      STAGE1=F*F2/U.GT.U*0.25 .OR. F.NE.FF
78 C      R= ABS(DZ(1))+ ABS(DZ(2))
79 C      R=DABS(DZ(1))+DABS(DZ(2))
80 C      IF(R.LE.RO*3.)GO TO 150
81 C      DZ1=DZ(1)
82

```

```

73      DZ(1)=(DZ1*1.8-DZ(2)*2.4)*R0/R
74      DZ(2)=(DZ1*2.4+DZ(2)*1.8)*R0/R
75      GOTO 150
76      DZ1=DZ(1)
77      DZ(1)= DZ1*1.8-DZ(2)*2.4
78      DZ(2)= DZ1*2.4+DZ(2)*1.8
79      STAGE1=.TRUE.
80      F0Z(1)=F1Z(1)
81      F0Z(2)=F1Z(2)

      C
82      C FIND NEXT POINT IN THE ITERATION. THIS IS WHERE THE ITERATION STARTS
83      C IF THE PREVIOUS ONE WAS UNSUCCESSFUL.
84      Z0(1)=Z(1)
85      Z0(2)=Z(2)
86      F0=F
87      DZK(1)=DZ(1)
88      DZK(2)=DZ(2)
89      Z(1)=Z0(1)+DZ(1)
90      Z(2)=Z0(2)+DZ(2)
91      C IF EITHER PART OF Z IS SMALL REPLACE BY ZERO TO AVOID UNDERFLOWS S/
92      C IF( ABS(Z(1)).LT.EPS*ABS(Z(2)))Z(1)=0.
93      C IF( DABS(Z(1)).LT.EPS*DABS(Z(2)))Z(1)=0.
94      C IF( ABS(Z(2)).LT.EPS*ABS(Z(1)))Z(2)=0.
95      C IF( DABS(Z(2)).LT.EPS*DABS(Z(1)))Z(2)=0.
96      W(1)=Z(1)
97      W(2)=Z(2)
98      F=PA07DD(Z,FZ,N+1,A)
99      FF=F
100     IF (.NOT.STAGE1) GOTO 240
101     J=1
102     DIV2=F.GE.F0
103     180 IF (DIV2) GOTO 190
104     W(1)=W(1)+DZ(1)
105     W(2)=W(2)+DZ(2)
106     GOTO 200
107     190 DZ(1)=DZ(1)*0.5
108     DZ(2)=DZ(2)*0.5
109     W(1)=Z0(1)+DZ(1)
110     W(2)=Z0(2)+DZ(2)
111     FA=PA07DD(W,FW,N+1,A)
112     IF (FA.GE.F) GOTO 240
113     F=FA
114     FZ(1)=FW(1)

```

```

110 FZ(2)=FW(2)
111 Z(1)=W(1)
112 Z(2)=W(2)
113 J=J+1
114 IF (DIV2.AND.(J.EQ.3)) GOTO 220
115 IF (J.LE.N) GOTO 180
116 GOTO 240
117 DZ1=DZ(1)
118 DZ(1)=DZ1*0.6-DZ(2)*0.8
119 DZ(2)=DZ1*0.8+DZ(2)*0.6
120 Z(1)=Z(1)+DZ(1)
121 Z(2)=Z(2)+DZ(2)
122 F=PA0?DD(Z,FZ,N+1,A)
123 C END OF STAGE1 SEARCH
124 C240 R0= ABS(Z(1))-Z(1))+ ARS(Z(2))-Z(2))
125 R0=DABS(Z(1))-Z(1))+DABS(Z(2))-Z(2))
126 C
127 C CONVERGENCE TEST.
128 IF(F.LT.F0)GO TO 250
129 Z(1)=Z(1)
130 Z(2)=Z(2)
131 R1= ABS(Z(1))+ ABS(Z(2))
132 R1=DABS(Z(1))+DABS(Z(2))
133 IF(R0.LT.EPS*R1)GO TO 270
134 IF (F.LT.F0) GOTO 120
135 F=FO
136 IF(F.LE.FMIN)GO TO 270
137 DZ(1)=DZK(1)*(-0.3)-DZK(2)*(-0.4)
138 DZ(2)=DZK(1)*(-0.4)+DZK(2)*(-0.3)
139 STAGE1=.TRUE.
140 GOTO 160
141 C
142 C DEAL WITH N=1 CASE
143 Z(1)=-A(2)/A(1)
144 Z(2)=0.
145 C
146 C DEFLATE, STORE ROOT, RESTORE COEFFICIENT OF ORIGINAL POLY AND REDUCE N
147 A1(N)=ROOT(1,N)
148 ROOT(1,N)=Z(1)
149 ROOT(2,N)=0.
150 IF(Z(2).EQ.0.)GO TO 277
151 AC= ARS(Z(1))
152 AC=DABS(Z(1))
153 G=0.

```

S/

S/

S/

```

144 FC=A(1)
145 DO 275 K=2,N1
146 FD=FC*Z(1)+A(K)
147 G=AC*(G+ABS(FC))+ABS(FD)
148 G=AC*(G+DABS(FC))+DABS(FD)
149 IF(ABS(FC).GT.2.*EPS*G+SQRT(AMINI(F,FO)))GO TO 283
150 IF(DABS(FC).GT.2.*EPS*G+SQRT(SNGL(DMINI(F,FO)))GO TO 283
151 C DEFLATION WITH A REAL ROOT
152 G=0.
153 DO 280 K=1,N
154 G=G*Z(1)+A(K)
155 A(K)=G
156 GO TO 300
157 C DEFLATION WITH A PAIR OF COMPLEX CONJUGATE ROOTS
158 ROOT(2,N)=Z(2)
159 RR=0.
160 SS=0.
161 PP=-Z(1)-Z(1)
162 QQ=Z(1)**2+Z(2)**2
163 N=N-1
164 DO 287 K=1,N
165 TT=A(K)-PP*RR-QQ*SS
166 A(K)=TT
167 SS=RR
168 RR=TT
169 A1(N)=ROOT(1,N)
170 ROOT(1,N)=Z(1)
171 ROOT(2,N)=-Z(2)
172 N=N-1
173 IF(N-1)310,260,40
174 RETURN
175 END

```

S/

S/

PROGRAM UNIT PA07BD

ARGUMENTS AI M ROOT A MPI

NAME TYPE USE ATTRIBUTES REFERENCES

A	ED	V	ASAI	1	3	10	15	17	28	37
				38	39	42	45	51	57	60
				63	93	106	122	144	146	152
AC	ED	V	SS	153	162	163				
ALOG	ER	FN		3	142	147				
ALOGR	R	V	SS	7	34	53				
AI	ED	V	ASAI	7	34					
				1	3	10	12	13	38	65
				166						138
BASE	R	V	SS	5	7	35				
BIG	R	V	SS	5	18	26	49			
DARS	ED	IF		15	28	42	48	70	89	90
				123	127	142	147	149		
DIV2	EL	V	SS	4	97	98	114			
DMIN1	ED	IF		145						
DZ	ED	V	SAL	2	61	62	67	72	73	74
				76	77	78	85	86	87	88
				100	102	103	104	105	117	118
				120	121	132	133			119
DZK	ED	V	SAL	2	85	86	132	133		
DZ1	ED	V	SS	3	72	73	74	76	77	78
				118	119					117
EPS	R	V	SS	3	46	89	90	128	149	
EXP	ER	FN		56						
F	ED	V	SS	3	63	65	84	93	94	107
				108	122	124	129	130	131	149
FA	ED	V	SS	3	106	107	108			
FC	ED	V	SS	3	144	146	147	148	149	
FD	ED	V	SS	3	146	147	148			
FF	ED	V	SS	3	47	69	94			
FLOAT	R	IF		38	46	53				
FMIN	R	V	SS	46	131					
FW	ED	V	AI	1	106	109	110			
FZ	ED	V	SAL	2	63	67	93	109	110	122
FO	ED	V	SS	3	45	46	47	84	97	124
										129



F0Z	ED	V	SAI	130	149	58	68	80	81
F1Z	ED	V	A1	2	57	67	68	80	81
F2	R	V	SS	2	65				
G	ED	V	SS	68	69				
I	I	V	SS	3	143	147	149	150	152
J	I	V	SS	11	12	12	16	17	34
K	I	V	SS	5	10	13	14	96	113
M	I	V	SS	27	28	36	37	38	50
MP1	I	V	A S	145	146	151	152	153	161
N	I	V	A S	1	2	8	9	11	162
	I	V	SS	1	3				
	I	V	SS	8	16	18	19	20	21
				36	46	50	57	60	63
				106	115	122	138	139	140
				160	161	166	167	168	169
N1	I	V	SS	24	27	38	39	42	45
				145					48
PA07DD	ED	FN		3	63	65	93	106	122
PA07ED	ED	SN		67					
PP	ED	V	SS	3	158	162			
QZ	ED	V	SS	3	159	162			
R	R	V	SS	70	71	73	74		
ROOT	ED	V	ASA2	1	2	12	18	19	138
				155	166	167	168	165	140
RR	ED	V	SS	3	156	162	164	123	128
RO	R	V	SS	64	71	73	74		
R1	R	V	SS	127	128				
SMALL	R	V	SS	5	6				
SNGL	R	IF		149		149			
SORT	ER	FN		6	33	162	164		
SS	ED	V	SS	3	157				
SSMALL	R	V	SS	6	42				
STAGE1	EL	V	SS	4	69	79	95	134	
T	R	V	SS	49	54	56	59	64	
TT	ED	V	SS	3	162	163	165		
U	R	V	SS	28	29	30	31	33	37
				39	51	52	53	54	66
U0	R	V	SS	48	53				
U1	R	V	SS	25	30	33			
U2	R	V	SS	26	31	33			
W	ED	V	SAI	2	91	92	99	100	104
				111	112			105	106
Z	ED	V	SAI	2	40	41	59	60	61
				68	82	83	87	88	89
									90
									91

ZO	ED	V	SAL	92	93	111	112	120	121	122	123
10				125	126	127	136	137	139	141	142
120				146	152	155	158	159	167	168	
140				2	43	44	68	82	83	87	88
150			SAL	104	105	120	121	123	125	126	
160				11	14						
180				65	129						
190				66	76						
20				71	75	80					
200				82	135						
220				98	115						
240				98	102						
250				15	21						
260				101	106						
270				114	117						
275				95	107	116	123				
280				124	127						
283				23	136	170					
287				128	131	138					
290				145	148						
30				141	150						
300				151	153						
310				149	155						
40				161	165						
50				42	166						
70				16	17						
80				154	169						
				22	170	171					
				15	23	170					
				27	29	32					
				36	38						
				50	52	55					

```

1  SUBROUTINE PA07CD(A,N,NP1,R,E,W,F,IG,CR)
2  C IG AND CR MAY BE DYNAMICALLY EQUIVALENCED WITH SEPARATE PARTS OF W
3  C /
4  REAL A(NP1),R(2,N),W(NP1),C,RS,RP
5  DOUBLE PRECISION A(NP1),R(2,N),W(NP1),C,RS,RP
6  REAL E(NP1),F(NP1),CR(N),IG(N)
7  C W IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
8  C OF THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS.
9  C F IS A WORKSPACE ARRAY OF LENGTH N+1 USED TO HOLD THE COEFFICIENTS
10 C OF THE ERROR POLYNOMIAL.
11 C IG IS A WORKSPACE ARRAY OF LENGTH N USED TO LINK TOGETHER THE ROOTS
12 C IN A GROUP. THE FIRST IS IG1, IG(K) FOLLOWS K AND THE LAST IS I.
13 C OTHER ROOTS HAVE IG(K)=0.
14 C CR IS A WORKSPACE ARRAY OF LENGTH N USED TO HOLD DISTANCES FROM ROOT I
15 C TO TH REST.
16 C /
17 DATA EPS/1.0E-6 /,BIG/1.0E70/
18 DATA EPS/2.3E-16/,BIG/1.0E70/
19 C BIG IS A NUMBER NEAR THE OVERFLOW LIMIT. EPS IS THE SMALLEST
20 C NUMBER WHICH LEAVES UNITY UNCHANGED IN THE FLOATING-POINT ARITHMETIC
21 C IN USE.
22 FACT=1.05**(1./FLOAT(N))
23 N1=N+1
24 C
25 C MULTIPLY OUT THE POLYNOMIAL FORMED FROM THE CALCULATED ROOTS
26 DO 5 I=1,N1
27 W(I)=0.
28 W(1)=A(N1)
29 I=1
30 II=N1-I
31 IF(R(2,II).NE.0.)GO TO 12
32 C=R(1,II)
33 DO 10 JJ=1,I
34 J=I+1-JJ
35 W(J+1)=W(J+1)-C*W(J)
36 GO TO 18
37 RS=R(1,II)+R(1,II)
38 RP=R(1,II)**2+R(2,II)**2
39 I=I+1
40 DO 16 JJ=1,I
41 J=I-JJ
42 W(J+2)=W(J+2)-RS*W(J+1)
43 IF(J.EQ.0)GO TO 16
44 W(J+2)=W(J+2)+RP*W(J)
45 CONTINUE

```

```

27 I=I+1
28 IF(I.LE.N)GO TO 8
C
29 C FIND COEFFICIENTS OF ERROR POLYNOMIAL
30 DO 20 I=1,N1
31 II=N+2-I
32 F(I)=ABS(A(II))-W(I))+AMAX1(E(II),ARS(A(II)))*EPS)
33 F(I)=DABS(A(II))-W(I))+AMAX1(E(II),ABS(SNGL(A(II)))*EPS)
S/
C
34 C MAIN ERROR-ROUNDING LOOP STARTS HERE AND EXTENDS TO THE END
35 C OF THE SUBROUTINE IF FINDS A BOUND FOR ROOT I.
36 I=1
37 M=1
38 IG(I)=-1
39 IG1=1
40 D=0.
C.
41 C FIND DISTANCES TO OTHER ROOTS
42 R1=R(1,I)
43 R2=R(2,I)
44 IF(R2.EQ.0.)GO TO 34
45 CR1=CABS(CMPLX(R1,R2))
46 DO 32 K=1,N
47 R1=R(1,K)-R(1,I)
48 R2=R(2,K)-R(2,I)
49 CR(K)=CABS(CMPLX(R1,R2))
50 GO TO 40
51 K=1
52 CR1=ABS(R1)
53 R2=R(2,K)
54 R1=R(1,K)-R(1,I)
55 IF(R2.EQ.0.)GO TO 37
56 CR(K)=CABS(CMPLX(R1,R2))
57 CR(K+1)=CR(K)
58 K=K+1
59 GO TO 38
60 CR(K)=ABS(R1)
61 K=K+1
62 IF(K.LE.N)GO TO 36
C
63 C INITIALIZATION STATEMENTS WHICH ARE ALSO NEEDED WHEN M IS INCREASED
64 DMIN=RIC

```

```

61 S=CR1+D
62 RAD=D
63 C TEST ROUCHE CONDITION WITH RADIUS RAD.
64 45 TOP=F(1)
C
65 PROD=1.
66 BTM=ABS(A(N1))
67 BTM=DARS(A(N1))
68 DO 80 K=1,N
69 TOP=S*TOP+F(K+1)
70 DIST=CR(K)
71 IF(IG(K).NE.0.)GO TO 70
72 BTM=BTM*(DIST-RAD)
73 IF(DMIN.LE.DIST)GO TO 80
74 DMIN=DIST
75 L=K
76 GO TO 80
77 PROD=PROD*(RAD-DIST)
78 CONTINUE
79 IF(BTM.EQ.0.)GO TO 100
80 TB=ARS(TOP/RTM)
81 IF(PROD.GE.TB)GO TO 110
82 C FIND A NEW TRIAL RADIUS.
83 OLDS=S
84 OLDR=RAD
85 RL=RAD
86 RAD=D+1.1*TR
87 IF (M.EQ.1)GO TO 98
88 RM=D+(1.1*TR)*(1./FLOAT(M))
89 C RISECTION LOOP. RL AND RM HOLD UPPER AND LOWER BOUNDS.
90 RAD=(RL+RM)/2.
91 PROD=1.
92 K=IG1
93 PROD=PROD*(RAD-CR(K))
94 K=IFIX(IG(K))
95 IF(K.GT.0)GO TO 90
96 IF(PROD.GE.1.10*TB)GO TO 94
97 IF(RAD.GE.DMIN)GO TO 100
98 IF(PROD.GT.1.05*TB)GO TO 98
99 RL=RAD
100 GO TO 83
101 RM=RAD
102 GO TO 85
103 S=S+RAD-OLDR
104 IF(RAD.GE.DMIN)GO TO 100

```

S/

```

101 IF( S*((DMIN-OLDR)/(DMIN-RAD)).LE.FACT*OLDS)GO TO 110
102 GO TO 45

103 C ADD ROOT TO GROUP
104 M=M+1
105 IG(L)=FLOAT(IG1)
106 IG1=L
107 D=DMIN
GO TO 40

108 C STORE ERROR BOUND AND RESET IG
109 E(I)=RAD
110 K=IG1
111 IG1=IFIX(IG(IG1))
112 IG(K)=0.
113 IF(K.NE.1)GO TO 120
114 IF(R(2,I).EQ.0.)GO TO 130
115 E(I+1)=E(I)
116 I=I+1
117 IF(I.LE.N)GO TO 31
118 RETURN
119 END

```

PROGRAM UNIT PA07CD

ARGUMENTS A N NPI R E W F IG CR

NAME	TYPE	USE	REFERENCES	A	N	NPI	R	E	W	F	IG	CR
A	ED	V	A A1	1	2	31	65	9	31	65		
ABS	R	IF		31	49	57	78	57	78			
AMAXI	R	IF		31								
BIG	R	V	SS	4	60							
BTM	R	V	SS	65	70	77	78	77	78			
C	ED	V	SS	2	13	16		16				
CABS	ER	FN		42	46	53		53				
CMPLX	EC	IF		42	46	53		53				
CR	ER	V	ASAI	1	3	46	54	46	54	57	68	89
CRI	R	V	SS	42	49	61	65	61	65	106		
D	R	V	SS	38	61	62	83	62	83	106		
DABS	ED	IF		31	65							
DIST	R	V	SS	68	70	71	72	71	72	75		
DMIN	R	V	SS	60	71	72	92	72	92	100	106	
E	ER	V	ASAI	1	3	31	108	31	108	114		
EPS	R	V	SS	4	31							
F	ER	V	ASAI	1	3	31	63	31	63	67		
FACT	R	V	SS	5	101							
FLOAT	R	IF		5	85	104		104				
I	I	V	SS	7	8	10	11	10	11	14	20	21
				22	27	28	29	28	29	30	31	33
				34	36	37	39	37	39	40	44	51
IFIX	I	IF		108	112	113	114	113	114	115	117	
IG	ER	V	ASAI	90	110	33	36	33	36	69	104	110
				111	3							
IG1	I	V	SS	37	88	104	109	104	109	110		
II	I	V	SS	11	12	13	18	13	18	19	31	
J	I	V	SS	15	16	22	23	22	23	24	25	
JJ	I	V	SS	14	15	21	22	21	22			
K	I	V	SS	43	44	45	46	45	46	48	51	53
				54	55	57	58	57	58	59	66	68
				60	73	88	89	88	89	90	91	109
				112								111
L	I	V	SS	73	104	105						

M	I	V	SS	35	84	85	103	6	28	30	32
N	I	V	AS	1	2	3	5				
	I	V	AS	43	59	66	117				
NPI	I	V	AS	1	4	3					
NI	I	V	SS	6	7	9	11	29	65		
OLDR	R	V	SS	81	99	101					
OLDS	R	V	SS	80	101						
PROD	R	V	SS	64	75	79	87	89	92	94	
R	ED	V	A A2	1	2	12	13	18	19	39	40
				44	45	50	51	113			
				62	70	75	81	82	83	86	89
RAJ	R	V	SS	93	95	97	99	100	101	108	
				82	86	95					
RL	R	V	SS	85	86	97					
RM	R	V	SS	2	19	25					
RP	ED	V	ST	2	18	23					
RS	ED	V	SS	3	42	44	46	49	51	53	57
R1	R	V	SS	39	42	44	46	49	51	53	57
R2	R	V	SS	40	41	42	45	46	50	52	53
S	R	V	SS	61	67	80	99	101			
SNGL	R	IF		31							
TB	R	V	SS	78	79	83	85	92	94		
TDP	R	V	SS	63	67	78					
W	ED	V	ASAI	1	2	8	9	16	23	25	31
10				14	16						
100				77	93	100	103				
110				79	101	108					
12				12	18						
120				109	112						
130				113	116						
16				21	24	26					
18				17	27						
20				29	31						
30				32	33						
31				35	117						
32				43	46						
34				41	48						
36				50	59						
37				52	57						
39				56	58						
40				47	60						
				63	102	107					
5				7	8						
70				60	75						
8				11	28						



80  
83  
90  
94  
98

66 71 74 76  
86 96 98  
89 91  
92 97  
84 94 99

S/  
S/

```
1 C REAL FUNCTION PA07DD(Z,FZ,N1,A)
2 C DOUBLE PRECISION FUNCTION PA07DD(Z,FZ,N1,A)
3 C REAL Z(2),FZ(2),A(N1),P,Q,R,S,T
4 C DOUBLE PRECISION Z(2),FZ(2),A(N1),P,Q,R,S,T
5 C Z AND FZ CONTAIN COMPLEX NUMBERS.
6 C A CONTAINS REAL POLYNOMIAL COEFFICIENTS.
7 C POLYNOMIAL VALUE AT Z IS SET IN FZ AND THE SQUARE OF ITS MODULUS
8 C IS RETURNED AS FUNCTION VALUE.
9 N=N1-1
10 T=A(1)
11 IF(Z(2).EQ.0.)GO TO 30
12 P=Z(1)+Z(1)
13 Q=-Z(1)**2+Z(2)**2
14 R=0.
15 IF(N.EQ.1)GO TO 20
16 DO 10 K=2,N
17 S=R
18 R=T
19 T=P*R+Q*S+A(K)
20 FZ(1)=Z(1)*T+Q*R+A(N+1)
21 FZ(2)=Z(2)*T
22 GO TO 50
23 P=Z(1)
24 DO 40 I=1,N
25 T=T*P+A(I+1)
26 FZ(1)=T
27 FZ(2)=0.
28 PA07DD=FZ(1)**2+FZ(2)**2
29 RETURN
30 END
```

PROGRAM UNIT PA07DD

ARGUMENTS Z FZ NI A

NAME TYPE USE ATTRIBUTES REFERENCES

A	ED	V	A AI	1	2	13	14	19	22
FZ	ED	V	ASAI	1	2	14	20	21	
I	I	V	SS	18	19				
K	I	V	SS	10	13				
N	I	V	SS	3	9	10	14	18	
NI	I	V	AS	1	2	3			
P	ED	V	SS	2	6	13	17	19	
PA07DD	ED	FN	S	1	22				
Q	ED	V	SS	2	7	13	14		
R	ED	V	SS	2	8	11	12	13	14
S	ED	V	SS	2	11	13			
T	ED	V	SS	1	4	12	13	14	15
Z	ED	V	A AI	1	2	5	6	7	14
10				10	13				19
20				9	14				15
30				5	17				17
40				18	19				20
50				16	22				17

```

1      SUBROUTINE PAOTED(A,B,Q)
2      REAL A(2),B(2),Q(2),U,A1,C,D,E
3      DOUBLE PRECISION A(2),B(2),Q(2),U,A1,C,D,E
4      C IMPLEMENTS THE COMPLEX DIVISION Q=-A/B
5      U= ARS(R(1))+ ARS(B(2))
6      U=DABS(B(1))+DABS(B(2))
7      A1=A(1)
8      C=B(1)/U
9      D=B(2)/U
10     E=- (C*C+D*D)*U
11     Q(1)=(A1*C+A(2)*D)/E
12     Q(2)=(A(2)*C-A1*D)/E
13     RETURN
14     C
15     END
16     //

```

PROGRAM UNIT PAOTED

ARGUMENTS A B Q

NAME	TYPE	USE	ATTRIBUTES	REFERENCES
A	V		A A1	1
A1	V		SS	2
B	V		A A1	1
C	V		SS	2
D	V		SS	2
DABS	IF			3
E	V		SS	2
Q	V		ASAI	1
U	V		SS	2

## References

- M.A. Jenkins (1969). Three-stage variable-shift iterations for the solution of polynomial equations with a posteriori error bounds for the zeros. Stanford Report CS 138.
- M.A. Jenkins and J.F. Traub (1970). A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. Numer. Math. 14, 252-263.
- K. Madsen (1973). A root-finding algorithm based on Newton's method. BIT 13, 71-75.
- A.M. Ostrowski (1966). Solution of equations and systems of equations. Academic Press.
- G. Peters and J.H. Wilkinson (1971). Practical problems arising in the solution of polynomial equations. J. Inst. Maths. Applics., 8, 16-35.
- B.G. Ryder (1973). The FORTRAN verifier: user's guide. Computing science technical report # 12, Bell Telephone Laboratories.
- J.H. Wilkinson (1963). Rounding errors in algebraic processes. HMSO.