ALGORITHM 30
NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION

K. W. Ellenberger
Missile Division, North American Aviation, Downey, California

```
procedure  ROOTPOL (n, a, L, F, u, v, CONV)  ;
           value n, a, L, F  ; integer L, F, n  ;
           array a, u, v, CONV  ;
comment  The Bairstow and Newton correction formulae are
   used for a simultaneous linear and quadratic iterated synthetic
   division. The coefficients of a polynomial of degree n are given as
   a_i (i = 0, i, ... , n) where a_n is the constant term. The coeffi-
   cients are scaled by dividing them by their geometric mean.
   The Bairstow or Newton iteration method will nearly always
   converge to the number of figures carried, F, either to root
   values or to their reciprocals. If the simultaneous Newton and
   Bairstow iteration fails to converge on root values or their
   reciprocals in L iterations, the convergence requirement will be
   successively reduced by one decimal figure. This program antici-
   pates and protects against loss of significance in the quadratic
   synthetic division. (Refer to "On Programming the Numerical
   Solution of Polynomial Equations," by K. W. Ellenberger,
   Commun. ACM 3 (Dec. 1960), 644-647.) The real and imaginary
   part of each root is stated as u[i] and v[i], respectively, together
   with the corresponding constant, CONV_i, used in the con-
   vergence test. This program has been used successfully for over
   a year on the Bendix G15-D (Intercard System) and has recently
   been coded for the IBM 709 (Fortran System);
            begin integer i, j, m  ; array h, b, c, d, e[-2 :n]  ;
               real t, K, ps, qs, pt, qt, s, rev, r  ;
ROOTPOL:    b_-1 := b_-2 := c_-1 := c_-2 := d_-1 := d_-2 := e_-1 :=
               e_-2 := 0  ;
            for j := 0 step 1 until n do h_j := a_j  ;  t := 1  ;
               K := 10^F  ;
ZROTEST:    if h_n = 0 then
            begin u_n := 0  ;  v_n := 0  ;  CONV_n := K  ;
               n := n - 1  ;  go to ZROTEST
            end  ;
INIT:       if n = 0 then go to RETURN  ;
               ps := qs := pt := qt := s := 0  ;
               rev := 1  ;  K := 10^F  ;
            if n = 1 then
            begin r := - h_1/h_0  ;  go to LINEAR
            end  ;
            for j := 0 step 1 until n do
            begin
            if h_j = 0 then s := s else s := s + log(abs(hj))
            end  ;  s := s^10  ;
            for j := 0 step 1 until n do h_j := h_j/s  ;
            if abs (h_1/h_0) < abs (h_n-1/h_n) then
REVERSE:    begin t := -t  ;  m := entier ((n+1)/2)  ;
            for j := 0 step 1 until m do
            begin s := h_j  ;  h_j := h_n-j  ;  j_n-j := s
            end
            end  ;
            if qs ≠ 0 then
            begin p := ps  ;  q := qs  ;  go to ITERATE
            end  ;
            if h_n-2 = 0 then
            begin q := 1  ;  p := -2
            end else
            begin q := h/h_n-2  ;  p := (h_n-1 - q × h_n-3)/h_n-2
            end  ;
            if n = 2 then go to QADRTIC  ;  r := 0  ;
ITERATE:    for i := 1 step 1 until L do
            begin
BAIRSTOW:   for j := 0 step 1 until n do
            begin b_j := h_j - p × b_j-1 - q × b_j-2  ;
               c_j := b_j-p × c_j-1 - q × c_j-2
            end  ;
            if n_n-1 = 0 then go to BNTEST  ;
            if b_n-1 = 0 then go to BNTEST  ;
            if abs (h_n-1/b_n-1)<K then go NEWTON  ;
               b_n := h_n-q × b_n-2  ;
BNTEST:     if b_n = 0 then go to QADRTIC  ;
            if K < abs (h_n/b_n) then go to QADRTIC  ;
NEWTON:     for j := 0 step 1 until n do
            begin d_j := h_j + r × d_j-1  ;  e_j := d_j + r × e_j-1
            end  ;
            if d_n = 0 then go to LINEAR  ;
            if K < abs (h_n/d_n) then go to LINEAR  ;
               c_n-1 := -p × c_n-2-q × c_n-3  ;
               s := c^2_n-2 - c_n-1 × c_n-3  ;
               if s=0 then
            begin p := p - 2  ;  q := q × (q + 1)
            end else
            begin p := p + (b_n-1 × c_n-2 - b_n × c_n-3)/s  ;
               q := q + (-b_n-1 × c_n-1 + b_n × c_n-2)/s
            end  ;
            if e_n-1 = 0 then r := r-1 else r := r - d_n/e_n-1
            end  ;  ps := pt  ;  qs := qt  ;  pt := p  ;
               qt := q  ;
            if rev < 0 then  K := K/10  ;  rev = -rev  ;
               go to REVERSE  ;
LINEAR:     if t < 0 then r := 1/r  ;  u_n := r  ;  v_n := 0  ;
            CONV_n := K  ;  n := n-1  ;
            for j := 0 step 1 until n do h_j := d_j  ;
            if n = 0 then go to RETURN  ;
               go to BAIRSTOW  ;
QADRTIC:    if t < 0 then
            begin p := p/q  ;  q := 1/q
            end  ;
            if 0 < (q - (p/2)^2) then
            begin u_n := u_n-1 := -p/2  ;
               s := sqrt (q - (p/2)^3)  ;  v_n := s  ;
               v_n-1 := -s
            end else
            begin s := sqt ((p/2)^2) - q)  ;
            if p < 0 then u_n := -p/2 + s
               else u_n := -p/2-s  ;  u_n-1 := q/u_n  ;
               v_n := v_n-1 := 0
            end  ;  CONV_n := CONV_n-1 := K  ;
               n := n-2  ;
               for j := 0 step 2 until n do h_j := b_j  ;
               go to INIT  ;
RETURN:     end
```

CERTIFICATION OF ALGORITHM 30 NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION (K. W. Ellenberger, *Comm. ACM*, Dec. 1960)

WILLIAM J. ALEXANDER
Argonne National Laboratory,* Argonne, Ill.

ROOTPOL was coded by hand for the LGP-30 using the ACT-III Compiler with 24 bits of significance. The following corrections were found necessary.

(a) $b_{-1} := b_{-2} := c_{-1} := c_{-2} := d_{-1} := d_{-2} := e_{-1} := e_{-2} := 0$ *should be*
$b_{-1} := b_{-2} := c_{-1} := c_{-2} := d_{-1} := e_{-1} := h_{-1} := 0$

(b) $m :=$ entier $((n + 1)/2)$ *should be*
$m :=$ entier $((n - 1)/2)$

(c) $j_{n-j} := s$ *should be* $h_{n-j} := s$

(d) $q := h/h_{n-2}$ *should be* h_n/h_{n-2}

(e) $cj := b_j - p \times c_j - 1 - q \times c_{j-2}$ *should be*
$c_j := b_j - p \times c_{j-1} - q \times c_{j-2}$

(f) **if** $n_{n-1} = 0$ **then go to** BNTEST *should be*
if $h_{n-1} = 0$ **then go to** BNTEST

(g) $s := \text{sqrt}\ (q - (p/2)^3)$ *should be*
$s := \text{sqrt}\ (q - (p/2)^2)$

(h) **for** $j := 0$ **step** 2 **until** n **do** $h_j := b_j$ *should be*
for $j := 0$ **step** 1 **until** n **do** $h_j := b_j$

(i) **go to** BAIRSTOW *should be* **go to** ITERATE

The following correction was found necessary in the given example (Refer to "On Programming the Numerical Solution of Polynomial Equations," by K. W. Ellenberger, *Comm. ACM 3*, Dec., 1960):

$f(x) = (.10098).\ 10^8\ x^4 - (.98913)\ 10^6\ x^2 + (.10000)\ 10^6\ x + (.10000)\ 10^1 = 0$ *should be*

$f(x) = (.10098)\ 10^8\ x^4 - (.98913)\ 10^6\ x^3 - (.10990)\ 10^6\ x^2 + (.10000)\ 10^6\ x + (.10000)\ 10^1 = 0$

With these corrections the results obtained agree with those given in the example.

For equations of higher order it was found necessary to avoid repeated scaling of the reduced equation in order to prevent floating point overflow. The range on the exponent in the ACT III system is $-32 \leqq e \leqq 31$.

Further floating point overflow difficulties were experienced when certain coefficients in the reduced equation became small but not zero. The following additions were made to avoid this fault:

(a) **for** $j := 0$ **step** 1 **until** n **do** $h_j := d_j$ *was replaced by*
for $j := 0$ **step** 1 **until** n **do begin if** abs $(h_j/d_j) < K$ **then** $h_j := d_j$ **else** $h_j := 0$ **end**

(b) **for** $j := 0$ **step** 1 **until** n **do** $h_j := b_j$ *was replaced by*
for $j := 0$ **step** 1 **until** n **do** begin **if** abs $(h_j/b_j) < K$ **then** $h_j := b_j$ **else** $h_j := 0$ **end**

With the above changes the following results were obtained:

$x^4 - 3\ x^3 + 20\ x^2 + 44x + 54 = 0$
$x = -.9706390 \pm 1.005808i$
$x = 2.470639 \pm 4.640533i$

$x^6 - 2\ x^5 + 2\ x^4 + x^3 + 6x^2 - 6x + 8 = 0$
$x = -.9999999 \pm .9999999i$
$x = 1.500000 \pm 1.322876i$
$x = .5000002 \pm .8660251i$

$x^5 + x^4 - 8x^3 - 16x^2 + 7x + 15 = 0$
$x = 3.000001$
$x = -2.000000 \pm 1.000001i$
$x = -.9999997$
$x = .9999998$

* Work supported by the U. S. Atomic Energy Commission

CERTIFICATION OF ALGORITHM 30 NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION [K. W. Ellenberger, *Comm. ACM 3* (Dec. 1960), as corrected in the previous Certification by William J. Alexander, *Comm. ACM 4* (May 1961)]

KALMAN J. COHEN
Graduate School of Industrial Administration, Carnegie Institute of Technology, Pittsburgh, Pa.

The ROOTPOL procedure originally published by Ellenberger as corrected and modified by Alexander was coded for the Bendix G20 in 20-GATE. Some serious errors were found in the third and fourth lines above the statement labelled "REVERSE" in Ellenberger's Algorithm which were not mentioned in Alexander's Certification. First, the function "log" is not a standard function in ALGOL 60; it is clear from the context, however, that Ellenberger intends this to be the logarithm function to the base 10. Second, Ellenberger's Algorithm failed to divide the accumulated sum of the logarithms by n+1 before taking the antilogarithm.

The correct, and slightly simplified, manner in which the third and fourth lines above the statement labelled "REVERSE" should read is:

if $h_j \neq 0$ **then** $s := \ln(\text{abs}(h_j))$
end; $s := s/(n+1)$; $s := \exp(s)$;

With these corrections, the numerical results obtained essentially agree with those reported by Alexander.

CERTIFICATION OF ALGORITHM 30 [C2] NUMERICAL SOLUTION OF THE POLYNOMIAL EQUATION [K. W. ELLENBERGER, *Comm. ACM 3* (Dec. 1960), 643]

JOHN J. KOHFELD (Recd. 31 Aug. 1964, 18 Nov. 1964 and 10 Nov. 1966)
Computing Center, United Technology Center, Sunnyvale, Calif. 94088

The *ROOTPOL* procedure was found to use the identifiers p, q, without declaring them. They should be declared **real**.

The first ALGOL statement in Cohen's Certification [*Comm. ACM 5* (Jan. 1962), 50] which reads:

if $h_j \neq 0$ **then** $s := ln\ (abs(h_j))$

should read:

if $h_j \neq 0$ **then** $s := ln\ (abs(h_j)) + s$.

The next line could be simplified to read:

end; $s := exp(s/(n+1))$;

The above corrections, as well as Algorithm 30 itself, are in publication language ALGOL. In order to translate the algorithm to reference language ALGOL, which is now used in CACM, 10^F would need to be replaced by $10 \uparrow F$, and h_j would need to be replaced by $h\ [j]$.

With these corrections and those contained in Alexander's Certification [*Comm. ACM 4* (May 1961), 238], Ellenberger's Algorithm was adapted to B-5000 ALGOL and successfully executed on the Burroughs B-5000 computer at United Technology Center. The results from the four examples used by Alexander are given below.

Example 1

$(1.0098)10^7x^4 - (9.8913)10^5x^3 - (1.0990)10^5x^2 + 10^5x + 1 = 0.$

The roots are:

$x = -0.201080185406$

$x = 0.149521622653 \pm 0.163989609283i$

$x = (-9.99989011230)10^{-6}.$

Example 2

$x^4 - 3x^3 + 20x^2 + 44x + 54 = 0$

$x = 2.47063897001 \pm 4.64053316164i$

$x = -0.970638970010 \pm 1.00580758903i$

Example 3

$x^6 - 2x^5 + 2x^4 + x^3 + 6x^2 - 6x + 8 = 0$

$x = -0.999999999990 \pm 1.000000000000i$

$x = 1.500000000000 \pm 1.32287565553i$

$x = 0.500000000000 \pm 0.866025403780i$

Example 4

$x^5 + x^4 - 8x^3 - 16x^2 + 7x + 15 = 0$

$x = 3.00000000000$

$x = -2.00000000000 \pm 1.00000000003i$

$x = -0.999999999990$

$x = 1.000000000000$

These results agree substantially with those given in Alexander's Certification.