ALGORITHM 59
ZEROS OF A REAL POLYNOMIAL BY RESULTANT PROCEDURE

E. H. Bareiss and M. A. Fisherkeller
Argonne National Laboratory, Argonne, Ill.

```
procedure   RES (n, c, alpha, mu, re, im, rt, gc)  ; value n,
            c, alpha ; integer n, alpha ; integer array
            mu  ; array c, re, im, rt, gc  ;
comment     RES finds simultaneously all zeros of a polynomial of
    degree n with real coefficients, c_j (j = 0, ...  n), where c_n
    is the constant term. The real part, re_i , and imaginary part,
    im_i , of each zero, with corresponding multiplicity, mu_i , and
    remainder term, rt_i , (i = 1, ... , n), are found and a poly-
    nomial with coefficients gc_j (j = 0, ... , n), is generated from
    these zeros. Alpha provides an option for local or nonlocal
    selection of M, the number of root-squaring iterations, and
    delta and epsilon, acceptance criteria. If alpha = 1, these
    parameters are assigned locally. If alpha = 2, M, delta and
    epsilon are set equal to the global parameters Mp, deltap,
    and epsilonp, respectively. In cases where zeros may be found
    more than once, the superfluous ones are eliminated by fac-
    torization. The method has been described by E. H. Bareiss
    (J. ACM 7, Oct. 1960, pp. 346-386).  ;
    begin integer M  ; real delta, epsilon  ; switch U :=
    U1, U2  ;
    go to U [alpha];
U1:           M := 10  ; delta := 0.2  ; epsilon := 10^-8  ;
              go to START  ;
U2:           M := Mp  ; delta := deltap  ; epsilon :=
              epsilonp  ;
START:        begin integer CT, nu, nuc, beta, m, j, jc, k,
              i, p  ; Boolean ROOT  ;
              real X, Y, GX, rp  ; array a, ac [0:n, 0:M],
              R, Rc, t [0:n],
                s [-1:n], ag [-2:n], rh, q, G, F [1:2×n]  ;
              switch S := S1, S2  ; switch T := T1, T2  ;
              switch V := V1, V2  ;
              real procedure min (u,v)  ; real u,v  ;
                min := if u ≦ v then u else v  ;
              real procedure SYND (W, Q, I, T)  ;
              integer I  ; real W, Q  ;
                array T  ;
SYNTHETIC     begin s [-1] := 0  ; s [0] := T [0]  ; for
 DIV:         m := 1 step 1 until I do
                s [m] := T [m] - W*s [m - 1] - Q×s
              [m - 2]  ;
              if Q = 0 then SYND := abs (s[I]) else
                SYND := abs (W/2×s [I - 1] + s[I])
              end SYND  ;
              CT := beta := 1  ; for j := 0 step 1 until
              n do a [j,0] := c[j]  ;
SQUARING      begin integer e1  ; real h  ; for m :=
 OPERATION:   1 step 1 until M do
              begin for j := 1 step 1 until n do
              begin h := 0  ; for e1 := 1 step 1 until
              min (n - j, j) do
                h := + (-1) ↑ e1 × a [j - e1,m - 1] × a
              (j + e1    - 1]  ;
                a [j,m] := (-1) ↑ j × (a [j, m - 1] ↑
                2 + 2×h) end end end  ;
              for j := 0 step 1 until n do R [j] := (-1) ↑
              j×a [j, M - 1] ↑ 2/a [j,M]  ;
                j := 0  ; nu := 1  ;
RD:           if (1 - delta ≦ R [j]) ∧ (R [j] ≦ 1 + delta)
              then
              begin rp := (a [j,M]/a [j - nu, M]) ↑ (1/(2 ↑
              M×nu))  ;
              go to T [beta] end  ;
1:            nu := nu + 1  ;
2:            j := j + 1  ; if j = n then go to S [beta]
              else go to RD  ;
3:            nu := 1  ; go to 2  ;
T1:           rh [CT] := rp  ; X := rp + epsilon × rp  ;
              Y := X + epsilon × rp  ;
              for k := 0 step 1 until n do t [k] := abs (c[k])  ;
                F [CT] := SYND (Y,0,n,t) - SYND
                (X,0,n,t)  ;
                G [CT] := SYND (rh [CT],0,n,c)  ; if
                F [CT] > G [CT] then
              begin ROOT := true  ; q [CT] := 0  ;
              CT := CT + 1  ; F [CT] := F[CT - 1] end  ;
              rh [CT] := -rp  ; G [CT] := SYND (rh
              [CT],0,n,c)  ;
              if F [CT] > G [CT] then begin ROOT :=
              true  ; q [CT] := 0  ; CT := CT + 1  ;
              F [CT] := F [CT - 1] end  ; if nu = 1 then
              go to 2  ;
              q [CT] := rp ↑ 2  ; nuc := nu  ; jc := j  ;
              for j := 0 step 1 until n do
              begin Rc [j] := R [j]  ; ac [j,M] := a [j,M]
              end  ;
RESULTANT:    begin real h  ; array b [-1:n + 1,
              -1:n + 1], A [1:n],
                r [0:n, 0:n], CB [-1:n + 1]  ;
              b [-1,0] := CB [-1] := CB [n + 1] := 0  ;
              for j := 0 step 1 until n do
              CB [j] := c[j]  ; b [0,0] := 1  ; for k :=
              1 step 1 until n do
              begin b [k,-1] := 0  ; for j := 0 step 1
              until k do
                b [k + 1,j] := b [k,j - 1] - q [CT] × b
                [k - 1,j]  ;
                b [k + 1, k + 1] := h := 0  ; for j :=
                n - k step -1 until 0 do
                h := h + (CB [j]×CB [k + j] - CB [j - 1]
                ×CB[k + j + 1]) × q[CT] ↑ (n - k - j)  ;
                A [k] := (-1 ↑ k×h  ; for j := 0 step
                1 until k - 1 do
              begin r [0,j] := 0  ; r [k,j] := r [k - 1,j] +
              A [k] × b [k,j] end  ,
                r [k,k] := A[k] end  ; beta := 2  ; for
                j := 0 step 1 until n do
                a [j,0] := r [n,j] end  ; go to SQUAR-
                ING OPERATION  ;
T2:           if (rp/2) ↑ 2 ≧ q [CT] then go to 3  ; rh
              [CT] := rp  ;
              G [CT] := SYND (rh [CT], q [CT], n,c)  ;
```

```
            if F [CT] > G [CT] then
            begin CT := CT + 1 ; F [CT] := F
            [CT − 1] ; q [CT] := q [CT − 1] end ;
            rh [CT] := −rp ; G [CT] := SYND [rh [CT],
            q [CT], n,c) ;
            if F [CT] > G [CT] then begin CT := CT
            + 1 ; F [CT] := F [CT − 1] ;
            q [CT] := q [CT − 1] end ; go to 3 ;
S2:         for j := 0 step 1 until n do begin a [j,M] :=
            ac [j,M] ;
            R [j] := Rc [j] end ; j := jc ; beta := 1 ;
            if ROOT then go to 3 else
               nu := nuc ; go to 1 ;
S1:         ag [−2] := ag [−1] := 0 ; ag [0] := 1 ;
            for j := 1 step 1 until n do
            ag [j] := 0 ; k := 1 ; i := n ; m := 1 ;
            for j := 0 step 1 until n do
               t [j] := c [j] ;
MULT:       mu [m] := 0 ; p := if q [k] = 0 then 1
            else 2 ;
IT:         GX := SYND (rh [k], q [k],i,t) ; if F [k]
            > GX then
            begin for j := 1 step 1 until n do
               ag [j] := ag [j] − rh [k] × ag [j − 1] + q
               [k] × ag [j − 2] ;
               mu [m] := mu [m] + p ; i := i − p ;
               for j := 0 step 1 until i do
               t [j] := s [j] ; go to IT end else if
               mu [m] ≠ 0 then begin
               rt [m] := G [k] ; go to V [p] end else
               go to D ;
V1:         re [m] := rh [k] ; im [m] := 0 ; go to E ;
V2:         re [m] := rh [k]/2 ; im [m] := sqrt (q [k] −
            re [m] ↑ 2) ;
E:          m := m + 1 ;
D:          k := k + 1 ; if k ≦ CT ∧ m ≦ n then go to
            MULT ;
            for j := 0 step 1 until n do gc [j] := ag [j] end
            end RES
```