Algorithms

J. G. HERRIOT, Editor

ALGORITHM 256
MODIFIED GRAEFFE METHOD [C2]

A. A. GRAU (Recd. 29 July 1964, 23 Oct. 1964 and 18 Jan. 1965)
Northwestern University, Evanston, Illinois

The algorithm given here mechanizes a modified form of the Graeffe process designed to avoid an expanding number range. This was discussed in [1]; the notation used below is the same as in that article.

Let the given polynomial be

$$a_0x^n + \cdots + a_n ;$$

the degree n and the array of coefficients a are input parameters of the procedure. An additional input parameter w is used to determine the number of stages needed to obtain a desired order of resolution; this may be considered to be roughly the number of significant decimal places expected in the zeros of the polynomial.

The algorithm finds the moduli, d_s $(s = 1, \cdots, n)$, of the zeros of the polynomial and the number of stages used for this, p. If the algorithm succeeds, the output parameter q is set equal to 0; otherwise, the value of q serves as the indicator for the reason of failure: $q = 1$ if the polynomial has a zero-valued coefficient, and $q = 2$ if a zero-valued divisor is encountered somewhere in the process. In either case, the moduli of the zeros are not found. Apart from these two cases, the algorithm applies generally; this includes the cases where some zeros have equal moduli or are imaginary.

The algorithm has been tested with polynomials of degree up to 10, including ill-conditioned cases such as polynomials with one or more sets of multiple or imaginary zeros. The algorithm has been compiled as it stands using both the Oak Ridge ALGOL Translator for the Control Data 1604 and the SHARE ALGOL Translator for the IBM 709/7090. In the case of the latter, one change as noted in a comment had to be made; this is presumably no longer necessary in a revision of the translator.

Garwick's device [2] is used as convergence criterion in both root extraction and the basic process. From w and the number of stages determined from it, it is possible to conclude whether some zeros may be considered to be of equal moduli; in such cases an adjustment of their values is possible and is made.

The quantities used in the modified Graeffe process are related to those occurring in the ordinary root-squaring process. This implies that in general the limitations of the Graeffe process (see, for example [3, pp. 67–69]) hold also in the modified process; the most serious of these is that initially the condition of successive polynomials may deteriorate.

An expanding number range is avoided by introducing at each step arithmetic divisions. It follows that if c_i is near zero, over- and underflow can occur in computing subsequent quantities. In the usual machine system, such a condition results in the automatic termination of computation; in this case this is not serious. In an ALGOL system where this is not true, a very unsatisfactory arrangement generally, machine-dependent facilities must be added to the algorithm to obtain the same effect; the ALGOL language contains no way of doing this. Theoretically a bridging mechanism is possible to work around near-zero divisors, but this has not been attempted here.

The modified process can be expected to perform somewhat better than the standard process in the case of equal moduli.

```
procedure Modified Graeffe (w, n, a, d, p, q);
  value w, n;  integer w, n, p, q;  array a, d;
begin
  real aa, eps, eps2, h, h1, h2, hh2, m, nh2;
  integer i, k, k0, k00, s, s3;
  array c[0:n], d1, hh[1:n], e[1:n, 1:n/2];  comment Using the
    SHARE processor, the last subscript bound n/2 was replaced by
    entier(n/2);
  eps := eps2 := 10-5;
  k00 := 40;  comment This is the maximum number of stages
    needed on the CDC 1604 where about 10 significant decimal
    figures may be obtained. On the IBM machines it is less, but
    the figure was not changed for such use;
  for s := 0 step 1 until n do
  begin if a[s] = 0 then begin q := 1;  go to out end end;
Determine the number of stages:
  k0 := entier (3.56 × w + 3.21);
  if k0 > k00 then k0 := k00;
Initialization:
  for s := 1 step 1 until n do
  begin
    if s + s > n then s3 := n-s else s3 := s;
    for i := 1 step 1 until s3 do
      e[s, i] := a[s+i] × a[s-i]/(a[s+i-1] × a[s-i+1]);
    d1[s] := abs (a[s]/a[s-1])
  end;
  c[0] := c[n] := 1;
  m := 1;
Main loop:
  for k := 1 step 1 until k0 do
  begin
    m := m/2;
    for s := 1 step 1 until n-1 do
    begin
      if s + s > n then s3 := n-s else s3 := s;
      h := 0;
      for i := s3 step -1 until 1 do
        h := (1-h) × e[s, i];
      c[s] := 1 - 2 × h;
      if c[s] = 0 then
      begin q := 2;  go to out end
    end;
    for s := 1 step 1 until n do
    begin
      if s + s > n then s3 := n-s else s3 := s;
      for i := 1 step 1 until s3 do
      begin
        h := (c[s+i]/c[s+i-1]) × e[s, i];
        e[s, i] := (c[s-i]/c[s-i+1]) × e[s, i] × h;
      end;
      comment In the paper [1] on which the algorithm is based,
        there is an error in equation (13) and results derived from
        it. The equation should be
```

$$e_{si}^{(k+1)} = [e_{si}^{(k)}]^2 \frac{c_{s+i}^{(k+1)} c_{s-i}^{(k+1)}}{c_{s+i-1}^{(k+1)} c_{s-i+1}^{(k+1)}} ;$$

```
Root extraction:
      aa := abs (c[s]/c[s-1]);
      comment  If the ↑ operation is suitably implemented for
        fractional exponent, the following 12 lines may be replaced
        by
                    hh[s] := h1 := aa ↑ (1/2↑k);
      h1 := h := 1 + (aa-1) × 10-2 × m;
      nh2 := 1;
AB:   for i := 1 step 1 until k do h := h × h;
      h2 := (aa/h-1) × m;
      h := h1 := h1 + h1 × h2;
      hh2 := abs (h2);
      if hh2 > eps then go to AB;
      if hh2 < nh2 ∧ hh2 ≠ 0 then
      begin
        nh2 := hh2;  go to AB
      end;
      hh[s] := h1;
      d1[s] := d1[s] × h1
    end;
    h := 0;
    for s := 1 step 1 until n do
    begin
      h1 := abs (hh[s]-1);
      if h1 > eps then go to AC;
      if h1 > h then h := h1
    end;
    if h < eps2 ∧ h ≠ 0 then
    begin eps2 := h;  go to AC end;
    go to Root determination;
AC:   end Main loop;
  k := k0;
Root determination:
  q := 0;  p := k;  s := 1;
BA:   for i := s step 1 until n do
  begin
    if abs (c[i]-1) < eps2 then
    begin k := i;  go to AE end
  end;
  k := n;
AE:   if k = s then
  begin
    d[s] := d1[s];  go to AG
  end
  else
  begin
    aa := 1;
    for i := s step 1 until k do
      aa := aa × d1[i];
    comment  If the ↑ operation is suitably implemented for
      fractional exponents, the following 13 lines may be replaced
      by
                    h := aa ↑ (1/(k-s+1));
    h1 := d1[s];
    nh2 := 1;
AF: h := 1;
    for i := s step 1 until k do
    h := h × h1;
    h2 := (aa/h-1)/(k-s+1);
    h1 := h1 + h1 × h2;
    hh2 := abs (h2);
    if hh2 > eps then go to AF;
    if hh2 < nh2 ∧ hh2 ≠ 0 then
    begin
      nh2 := hh2;  go to AF
    end;
    for i := s step 1 until k do d[i] := h1
  end;
```

TABLE 1

n	Coefficients (a_s)	Actual Zeros	p	Computed Moduli (d_s)
		Results from CDC 1604		
4	1 −1 −5 −1 −6	$\pm i$ −2 3	29	3.000000000 2.000000000 1.000000000 1.000000000
4	1 2 3 2 2	$\pm i$ $-1\pm i$	35	1.414213563 1.414213563 1.000001068 .9999989324
4	1 4 6 4 1	−1 (four-fold)	35	1.002108373 .9999999404 .9999999400 .9978961853
5	1 −5 −15 125 −226 120	1 2 3 4 −5	9	5.000000000 4.000000001 2.999999999 2.000000000 1.000000000
5	1 5 10 10 5 1	−1 (five-fold)	21	1.003023179 1.000737942 1.000737942 .9977555433 .9977555433
6	1 1 −45 35 524 −1236 720	1 2 3 4 −5 −6	9	5.999999999 5.000000004 3.999999998 3.000000001 2.000000000 1.000000000
6	1 6 15 20 15 6 1	−1 (six-fold)	22	1.009739721 1.009739721 1.000072156 1.000072156 .9902827716 .9902827715
		Results from IBM 709		
6	1 6 15 20 15 6 1	−1 (six-fold)	22	1.0442011 1.0219216 1.0219216 .97855264 .97855264 .95767000
10	1 10 45 120 210 252 210 120 45 10 1	−1 (ten-fold)	23	1.1896983 1.1896983 1.0977241 1.0977241 1.0001204 1.0001204 .91099190 .91099190 .84044056 .84044056
10	1 −55 1320 −18150 157773 −902055 3416930 −8409500 12753576 −10628640 3628800	1 2 3 4 5 6 7 8 9 10	10	10.001153 8.9947183 8.0090868 6.9926022 6.0027695 4.9996995 3.9999811 2.9999883 2.0000007 1.0000000

```
AG: if k = n then go to out;
  s := k + 1;
  go to BA;
out:
end Modified Graeffe
```

Tests. Some of the tests (Table 1) were run on the CDC 1604 using an earlier version of the algorithm; minor improvements incorporated afterwards should not affect the results substantially. The results obtained using the SHARE ALGOL translator and the IBM 709 suffer in comparison to those obtained on the 1604 for two main reasons: (1) significance of floating-point numbers is 27 bits vs. 35, and (2) input conversion routines introduce greater perturbations into input numbers. The last cases given are very poorly conditioned, so that the rather poor results should not be especially surprising.

Thanks and acknowledgements are due to several members of the Mathematics Division of Oak Ridge National Laboratory for running tests on the Control Data 1604, and to Mrs. Virginia Klema for running tests on the IBM 709 computer at Northwestern University.

REFERENCES:

1. GRAU, A. A. On the reduction of number range in this use of the Graeffe process. *J. ACM 10* (1963), 538–544.
2. GARWICK, J. V. The limit of a converging sequence. *Nord. Tidskr. Informationsbehandlung (BIT) 1* (1961), 64.
3. WILKINSON, J. H. *Rounding Errors in Algebraic Processes.* Prentice-Hall, New York, 1964.