Two algorithms for the solution of polynomial equations to limiting machine precision

J. A. Grant* and G. D. Hitchins†

Two algorithms are presented for the solution of polynomial equations, one for real coefficients, the other for complex coefficients. The theoretical development of the method used is briefly reviewed, together with a fuller account of its practical implementation.
(Received August 1973)

1. Introduction

In an earlier paper (Grant and Hitchins, 1971) it was shown how a minimisation technique could be used to give an always convergent algorithm for finding the roots of a general polynomial equation. [A similar approach has been suggested in Moore (1967) and Moore (1973).] There it was proposed that the algorithm should be used to find good initial approximations ready for refinement using the standard Newton or Bairstow iterations until limiting machine precision is attained. Here two implementations of the method are given, one for polynomials with real coefficients, the other for polynomials with complex coefficients. For both, the one method is used until the calculated value of the polynomial at the estimate of a root lies within a computed error bound, indicating that no further refinement of the root can be made and that it has been found as accurately as the machine precision permits.

The implementations are given in the form of ALGOL 60 procedures **realpolsolv** and **compolsolv**, written on the assumption that they will be used on a computer using binary arithmetic, one of the parameters of the procedures specifying the accuracy of the floating point arithmetic used.

In Section 2, the main features of the method are briefly reviewed. Section 3 is devoted to the question of terminating the iterations, whilst other particular points in the implementation are described in Section 4. Finally, in Section 5, some account is given of the tests the algorithms have undergone to check their robustness. The algorithms themselves are given in Appendix 1.

2. Basis of the method

Consider the polynomial

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

where the coefficients a_r may be real or complex. Let

$$f(z) = f(x + iy) = R(x, y) + iJ(x, y)$$

where R and J are real functions. Then the problem of finding a zero of (1) is equivalent to finding a solution of the pair of non-linear equations

$$R(x, y) = 0 = J(x, y) \tag{2}$$

which, in turn, is equivalent to finding a minimum of the function

$$\phi = \phi(z) = R^2(x, y) + J^2(x, y). \tag{3}$$

In the earlier paper, it was proved that a minimum could be found using the following algorithm:

(*a*) choose some initial point $\mathbf{z}_0$;

(*b*) at the point $\mathbf{z}_r$ evaluate the correction vector

$$\Delta \mathbf{z}_r = -\frac{1}{R_x^2 + J_x^2}\begin{pmatrix} RR_x + JJ_x \\ JR_x - RJ_x \end{pmatrix} \tag{4}$$

where R, J and the partial derivatives R_x and J_x are all taken at $\mathbf{z}_r$;

(*c*) for $\lambda_k = 1, \frac{1}{2}, \frac{1}{4}, \ldots$, set $\mathbf{z}_r^{(k)} = \mathbf{z}_r + \lambda_k \Delta \mathbf{z}_r$ and choose λ_k as large as possible, say λ_M, so that

$$\phi(\mathbf{z}_r) - \phi(\mathbf{z}_r^{(k)}) \geqslant 2\lambda_k \phi(\mathbf{z}_r)\sigma \tag{5}$$

where σ is some small positive number, in practice $_{10} - 4$;

(*d*) set $\mathbf{z}_{r+1} = \mathbf{z}_r + \lambda_M \Delta \mathbf{z}_r$ and, if sufficient accuracy has not been achieved when taking $\mathbf{z}_{r+1}$ as an estimate of a zero of (1) return to (*b*).

[In the later discussions, the whole minimisation process will be referred to as 'a search', whilst stage (*c*) will be referred to as 'a step'.]

The only difficulty that can arise within a search is that $\mathbf{z}_r$ may be at or near a saddle-point of ϕ, when $\Delta \mathbf{z}_r$ will be either undefined or large.

However, to use the above as the basis of a complete polynomial solver, some deflation process must be associated with it. Thus, there are five major points to be considered in any implementation.

1. *The choice of initial point for a search*

Assuming that forward deflation is to be used; it is desirable to find the roots in increasing order of modulus. Thus a sensible choice for the initial point would appear to be the origin. Since, however, in the case of a real polynomial, a purely real estimate for a root leads to another purely real estimate, it is necessary to start from a point displaced from the origin and generally $\mathbf{z}_0$ is taken to be $0{\cdot}001 + 0{\cdot}1i$. An option, however, is built into the procedures of starting from an alternative point in the search for the first root.

2. *The test for closeness to a saddle point*

At a saddle point, $R_x = 0 = J_x$ and the immediately obvious test would be one on $R_x^2 + J_x^2$. However, $R_x = 0 = J_x$ will also hold at a multiple root, when $R^2 + J^2$ will also be zero, which will not be the case at a saddle point. Thus a useful test must involve both $R^2 + J^2$ and $R_x^2 + J_x^2$ Now

$$\Delta \mathbf{z}_r = -\frac{1}{R_x^2 + J_x^2}\begin{pmatrix} RR_x + JJ_x \\ JR_x - RJ_x \end{pmatrix}$$

whence

$$|\Delta \mathbf{z}_r|^2 = \frac{R^2 + J^2}{R_x^2 + J_x^2}$$

and in the neighbourhood of a saddle point this could be expected to be $O(\rho^{-2})$ where ρ is the maximum relative error in any floating point operation. Thus having calculated R, J, R_x and J_x at $\mathbf{z}_r$, the quantity

$$k(R^2 + J^2) - (R_x^2 + J_x^2)$$

where k is some small number depending on the working precision, is examined. If it is negative, the direction of search

*Department of Mathematics, University of Bradford, Bradford BD7 1DP.
†Centre for Computer Studies, University of Leeds (now at the Computing Centre, University College, Cardiff).

should be meaningful, though the predicted step may still be large; if it is positive, it is possible that the search has led to a saddle point.

In the earlier paper, it was explained how the Lehmer–Schur algorithm could be used to determine a sensible movement away from a saddle-point. In practice, the implementation requires a good deal of coding, which has only rarely been required, and then only for artificial problems. Thus it was decided to adopt an 'ad hoc' approach, an exit from the procedure being forced when a saddle point situation has been detected, re-entry with an alternative starting point for the current search being allowed.

3. *Restriction of step-size*

The desirability of finding the roots in increasing order of modulus and the possibility of predicting a large step in a meaningful direction have both been mentioned. To encourage the former and to nullify the effect of the latter, it was decided to impose the restriction that the maximum step length that could be taken at any stage should be unity. Thus, having found that the predicted direction of search is a valid one

$$s^2 = |\Delta z_r|^2$$

is calculated. For $s < 1$, the iteration proceeds normally. Otherwise Δz_r is replaced by $\Delta z_r/s$, which is equivalent to replacing σ in (5) by σ/s. With this restriction on step-size, it is desirable to transform the polynomial to ensure that it has at least one root within the unit circle. Use is made of the Schur algorithm (Ralston, 1965) to test whether there is a root in the unit circle; if not, the transformation $z \to 2z$ is applied repeatedly until the test is satisfied. For higher degree polynomials, this can lead to large coefficients and a normalisation consisting of division by an integral power of 2 is used, the power being chosen to make the product of the non-zero coefficients approximately equal to unity. If necessary, the transformation and the normalisation are repeated after each deflation.

4. *Termination of the iteration*

The basic iteration can be used to limiting machine precision in the sense that it is terminated when the calculated values of R and J are less than the possible accumulated rounding error. A fuller discussion is given in the next section.

5. *Deflation process*

Although the roots are generally found in increasing order of modulus, composite deflation as advocated in Peters and Wilkinson (1971) is incorporated into the procedures.

3. Evaluation of R and J and the termination of the iterations

In this section, the analysis of Peters and Wilkinson (1971) is followed and similar notation is used. It is assumed that

$$fl(x \pm y) = (x \pm y)/(1 + \varepsilon)$$
$$fl(x \times y) = xy(1 + \varepsilon)$$
$$fl(x/y) = x(1 + \varepsilon)/y$$

where $|\varepsilon| \leqslant \rho = 2^{-t}$, with t the number of binary digits in the mantissa of a floating point number.

Basically, there are two types of evaluation to be considered:
(*a*) a real polynomial taken at a complex point;
(*b*) a complex polynomial taken at a complex point.
For the former, Wilkinson (1965) is followed, leading to the evaluation of the polynomial (1) at the point $x + iy$ by using the recurrence

$$b_0 = a_0,\ b_1 = a_1 - pb_0$$
$$b_k = a_k - pb_{k-1} - qb_{k-2},\ k = 2, 3, \ldots, n-1$$
$$b_n = a_n + xb_{n-1} - qb_{n-2}\ ,$$

where $p = -2x$ and $q = x^2 + y^2$ when

$$f(x + iy) = b_n + iyb_{n-1} = R + iJ\ .$$

Adams (1967) derives an error bound on the calculated function values, which under the assumptions given above and using single length arithmetic only, is given by the recurrence

$$e_0 = 0{\cdot}8|b_0|$$
$$e_k = |x + iy|e_{k-1} + |b_k| = \sqrt{q}\,e_{k-1} + |b_k|,\ k = 1, 2, \ldots, n$$

when

$$|f(x + iy) - (R + iJ)| \leqslant (2|xb_{n-1}| - 8(|b_n| + \sqrt{q}|b_{n-1}|) + 10e_n)\rho = E$$

and $x + iy$ is accepted as a zero if

$$\sqrt{\phi} = |R + iJ| \leqslant E\ .$$

Assuming that e_n dominates the expression for E and that the maximum relative error in taking a square root is ρ, allowance can be made for the rounding errors committed in the evaluation of E by replacing E by $E(1 + \rho)^{4n+3}$.

Wilkinson (1965) also shows that the required values of R_x and J_x can be found by extending the algorithm thus:

$$c_0 = b_0,\quad c_1 = b_1 - pc_0$$
$$c_k = b_k - pc_{k-1} - qc_{k-2},\ k = 2, 3, \ldots, n-3$$
$$c_{n-2} = b_{n-2} - qc_{n-4}$$

when

$$R_x = -2y^2c_{n-3} + b_{n-1}$$

and

$$J_x = 2y(xc_{n-3} + c_{n-2})\ .$$

For the evaluation of a complex polynomial for a complex argument, the complex form of the standard Horner algorithm is used. For

$$f(z) = (a_0 + ib_0)z^n + (a_1 + ib_1)z^{n-1} + \ldots + a_n + ib_n$$

with a_k, b_k real, taken at the point $x + iy$, use is made of the recurrence relations

$$c_0 = a_0,\quad d_0 = b_0$$
$$\left.\begin{aligned} c_k &= xc_{k-1} - yd_{k-1} + a_k \\ d_k &= yc_{k-1} + xd_{k-1} + b_k \end{aligned}\right\}\ k = 1, 2, \ldots, n$$

when $R = c_n$ and $J = d_n$. Analysis of this algorithm (Appendix 2) shows that an error bound can be computed using the recurrence

$$g_0 = h_0 = 1$$
$$\left.\begin{aligned} g_k &= |x|(g_{k-1} + |c_{k-1}|) + |y|(h_{k-1} + |d_{k-1}|) + |a_k| + 2|c_k| \\ h_k &= |y|(g_{k-1} + |c_{k-1}|) + |x|(h_{k-1} + |d_{k-1}|) + |b_k| + 2|d_k| \end{aligned}\right\}\ k = 1, 2, \ldots, n$$

when bounds on the errors in R and J are given by ρg_n and ρh_n respectively, or, allowing for possible errors committed during the evaluations of g_n and h_n, more precisely by

$$\rho g_n(1 + \rho)^{5n} \text{ and } \rho h_n(1 + \rho)^{5n}\ .$$

Thus $x + iy$ is accepted as a root of the complex polynomial when

$$|R| < \rho g_n(1 + \rho)^{5n} \text{ and } |J| < \rho h_n(1 + \rho)^{5n}\ .$$

The required values of R_x and J_x are again readily found by extending the basic recurrence:

$$u_0 = c_0,\quad v_0 = d_0$$
$$\left.\begin{aligned} u_k &= xu_{k-1} - yv_{k-1} + c_k \\ v_k &= yu_{k-1} + xv_{k-1} + d_k \end{aligned}\right\}\ k = 1, 2, \ldots, n-1$$

when $R_x = u_{n-1}$ and $J_x = v_{n-1}$.

4. Organisational details

It is hoped that the bodies of the two procedures **realpolsolv** and **compolsolv** given below are largely self-explanatory. There are, however, certain less obvious points.

1. *Function evaluation and the test for convergence*
These are embodied in a parameter-less procedure statement called *function*, the algorithms described in Section 3 being used to evaluate R, J, R_x and J_x at a given point. In **realpolsolv**, the test on convergence is applied on each function evaluation, the **boolean** *sat* being set to **true** when the test shows that convergence has taken place. For **compolsolv**, the convergence test is not applied immediately as it involves more than a doubling in the time of the function evaluation; instead it is activated (using *sat*) when either more than 20 steps have been taken in a search or the last step taken was less than $_{10} - 5$.

2. *Preliminary tests on the polynomials*
Two tests are included to detect extreme cases. One leads to an error exit if it is found that either the leading coefficient is exactly zero, or the degree is less than one. The other detects simple and multiple roots at the origin.

3. *Scaling of the polynomial*
As mentioned above, the coefficients in the polynomial may get large due to the transformations used to bring a zero within the unit circle. To avoid any resulting difficulties, the coefficients are scaled by an integral power of 2, chosen so that the product of the moduli of those coefficients (with modulus greater than $_{10} - 5$) is of the order of unity.

4. *Transformation of the polynomial*
The Schur test is used to see whether or not there is a zero in the unit circle; if not, the transformation $z \rightarrow 2z$ is applied. A record of the number of such transformations is held by means of the variable *fac* which equals 2^r when r transformations have been applied. This form was adopted for the convenience resulting when having to transform calculated zeros to zeros of the original polynomial. The inverse transformation must also be applied to the coefficients of the deflated polynomial before the forced exit brought about by the detection of the possibility of a saddle point.

5. *Acceptance of and deflation with a calculated root*
When a root has been isolated to limiting precision, it is removed from the polynomial using the composite deflation technique of Peters and Wilkinson (1971). In **compolsolv**, there is an immediate investigation to see whether the conjugate of the root just found also satisfies the equation and, to this end, a **boolean** variable *first root* serves to indicate whether the first root of a possible conjugate pair is being calculated.

In **realpolsolv**, care has to be taken to distinguish between the first root of a complex pair, when a second root is immediately available, and a complex root in which the imaginary part is due to the computational process. If the imaginary part is greater than 0·1, it is assumed that the estimate represents a complex root and deflation by a quadratic factor proceeds directly. For smaller imaginary parts, the polynomial is examined to see whether it is satisfied by the real part of the estimate only. If it is, a real root is accepted and a composite linear deflation is performed; otherwise a conjugate pair of complex roots is accepted and composite deflation proceeds with a quadratic factor.

6. *Abnormal entry and exit forced by detection of a saddle point*
When a saddle-point has been detected as described earlier, an error exit from the procedure is forced. To allow for subsequent re-entry with a new starting point for a search, the coefficients of the deflated polynomial must be transformed back to the coefficients of the deflated polynomial in the original variable. Having done this, the **integer** variable *ind* is given the value 2 and the exit is forced.

On normal exit from the procedures, *ind* will have the value 0. The value 1 is reserved for the case of a zero leading coefficient. Detection of *ind* having the value 2, can be followed by re-entry with *ind* unchanged. [Normal entry is with *ind* zero.] On this entry, the starting point for the first search is taken from the first components of the arrays subsequently used for holding the calculated root estimates. [Root estimates are held in these arrays in reverse order of computation.]

5. Results

The two procedures **realpolsolv** and **compolsolv** have been tested extensively and without failure on the ICL KDF9 at the University of Leeds. This machine has a 48-bit word, 39 of which are used in the mantissa of a floating point number. Both polynomial solvers have been used successfully on all the polynomials proposed by Henrici and Watkins (1965). In addition, they have been used on polynomials with coefficients taken from pseudo-random uniform distributions over various ranges and with degrees varying between 5 and 50. In all, both procedures were tested on over 200 polynomials and no failure occurred. To check that the calculated estimates of the roots were meaningful, the first of the *a posteriori* error analyses proposed by Peters and Wilkinson (1971) was applied to several of the test problems. Allowing for the general worsening of the error bounds as the degree of the polynomial increases, the results again appeared satisfactory. For both procedures, the number of steps taken per search was of the order indicated in detail in an earlier paper (Grant and Hitchins, 1971).

The presence of a saddle-point was not detected for any of the test polynomials. However, this part of the procedures was tested on the polynomials $z^n \pm 1$, $z^n \pm i$, which have saddle-points at the origin. For sufficiently large values of n, the starting point for a search lies within the regions of influence of these saddle-points and an exit is forced. Re-entry with the starting point close to the unit circle led to successful convergence to a root, followed by the complete solution of the resulting deflated polynomial.

Appendix 1 The algorithms

procedure *realpolsolv*(*a*, *nn*, *rez*, *imz*, *tol*, *ind*);
comment *This procedure attempts to solve a real polynomial equation of degree nn using the search algorithm proposed in Grant and Hitchins* (1971) *to limiting machine precision. On entry, the coefficients of the polynomial are held in the array a*[0: *nn*], *with a*[0] *holding the coefficient of the highest power. On normal entry, the parameter ind has value zero and will remain zero on successful exit with the calculated estimates of the roots held as rez*[*k*] + *iimz*[*k*], *k* = 1(1)*nn, in approximate decreasing order of magnitude. The parameter tol gives the precision of the floating binary arithmetic used, normally* 2 ↑ (−*t*), *where t is the number of bits in the mantissa.*

Abnormal exits will be indicated by ind having value 1 *or* 2. *The former implies that either a*[0] = 0 *or nn* < 1. *For ind* = 2, *a possible saddle point has been detected. The degree of the reduced polynomial is stored in nn and its coefficients are held in a*[0] *to a*[*nn*], *the roots obtained thus far being stored in the arrays rez and imz starting with rez*[*nn* + 1] + *iimz*[*nn* + 1]. *An immediate re-entry is possible with ind unchanged and with a new starting point for the search held in rez*[1] + *iimz*[1];
value *tol*; **real** *tol*;
integer *nn*, *ind*; **real array** *a*, *rez*, *imz*;

```
    begin integer i, k, n;
      real r, j, rx, jx, sig, t, scale, g, tol2, s1, s2, s, x, y, fun, nfun,
        fac;
      real array b, c[0: nn]; boolean sat, flag;
      switch ss := finish, linear, quadratic;
    procedure function;
    comment Evaluates r, rx, j, jx at the point x + iy and applies
      the Adams test. The boolean variable sat is given the value
      true if the test is satisfied;
    begin real p, q, a1, a2, a3, b1, b2, b3, c, t;
      integer k, m;
      p := - 2·0 × x; q := x × x + y × y; t := sqrt(q);
      a2 := b2 := 0·0;
      b1 := a1 := a[0]; c := abs(a1) × 0·8; m := n - 2;
      for k := 1 step 1 until m do
      begin a3 := a2; a2 := a1;
        a1 := a[k] - p × a2 - q × a3; c := t × c + abs(a1);
        b3 := b2; b2 := b1; b1 := a1 - p × b2 - q × b3
      end;
      a3 := a2; a2 := a1; a1 := a[n - 1] - p × a2 - q × a3;
      r := a[n] + x × a1 - q × a2; j := a1 × y;
      rx := a1 - 2·0 × b2 × y × y;
      jx := 2·0 × y × (b1 - x × b2);
      c := t × (t × c + abs(a1)) + abs(r);
      sat := sqrt(r × r + j × j) < (2·0 × abs(x × a1) - 8·0 ×
        (abs(r) + abs(a1) × t) + 10·0 × c) × tol × (1 + tol)
        ↑ (4 × n + 3)
    end of procedure function;
      fac := 1·0; flag := ind = 2; ind := 0; tol2 := tol × sqrt
        (tol); n := nn;
      if a[0] = 0·0 or n < 1 then begin ind := 1; goto fail end;
zerotest:
      if a[n] = 0·0 then begin
      rez[n] := imz[n] := 0·0; n := n - 1; goto zerotest end;
normalisation:
      scale := 0·0;
      for i := 0 step 1 until n do
      if abs(a[i]) ≥ 10 - 5 then scale := scale + ln(abs(a[i]));
      k := scale/((n + 1) × ln(2·0)); scale := 2·0 ↑ (-k);
      for i := 0 step 1 until n do b[i] := a[i] := a[i] × scale;
      comment Test for low order polynomial for explicit
        solution;
      if n ⩽ 2 then goto ss[n + 1];
schur test:
      for i := n step -1 until 1 do
      begin
        for k := 1 step 1 until i do c[k - 1] := b[i] × b[k] -
          b[0] × b[i - k];
        if c[i - 1] < -tol then goto search;
        t := if c[i - 1] < 1·0 then 1·0 else 1·0/c[i - 1];
        for k := i - 1 step -1 until 0 do b[k] := c[k] × t
      end of schur test for zero in unit circle;
comment transformation;
      fac := 2·0 × fac; scale := 1·0;
      for i := n - 1 step -1 until 0 do
      begin scale := 2·0 × scale; b[i] := a[i] := a[i] × scale end;
      goto schur test;
search:
      if flag then
      begin x := rez[1]; y := imz[1] + tol; flag := false end
      else begin x := 10 - 3; y := 0·1 end;
      function; fun := r × r + j × j;
again:
      g := rx × rx + jx × jx;
      if g < fun × tol2 then
      begin ind := 2; scale := 1·0;
        for i := n - 1 step -1 until 0 do
        begin scale := scale × fac; a[i] := a[i]/scale end;
        goto fail
      end with possible saddle point detected;
      s1 := -(r × rx + j × jx)/g; s2 := (r × jx - j × rx)/g;
      sig := 2_10 - 4; s := sqrt(s1 × s1 + s2 × s2);
      if s > 1·0 then begin s1 := s1/s; s2 := s2/s; sig := sig/s
        end;
      comment Valid direction of search has been determined, now
        proceed to determine suitable step;
      x := x + s1; y := y + s2;
loop:
      function; if not sat then
      begin nfun := r × r + j × j;
      if fun - nfun < sig × fun then
      begin s1 := 0·5 × s1; s2 := 0·5 × s2; s := 0·5 × s;
        sig := 0·5 × sig;
      x := x - s1; y := y - s2; goto loop end;
      fun := nfun; goto again end;
comment newroot;
      fun := 1·0/tol2; k := 0; imz[n] := y × fac;
      if abs(y) > 0·1 then goto complex;
      comment Check possibility of real root;
      s1 := y; y := 0·0; function; y := s1;
      if not sat then goto complex;
      comment Real root accepted and both backward and forward
        deflations are performed with linear factor;
      rez[n] := x × fac; imz[n] := 0·0; n := n - 1;
      b[0] := a[0]; c[n] := -a[n + 1]/x;
      for i := 1 step 1 until n do
      begin b[i] := a[i] + x × b[i - 1];
        c[n - i] := (c[n - i + 1] - a[n - i + 1])/x
      end;
      goto join;
      comment Complex root accepted and both backward and
        forward deflations are performed with quadratic factor;
complex:
      rez[n] := rez[n - 1] := x × fac;
      imz[n - 1] := -imz[n]; n := n - 2;
      r := 2·0 × x; j := -(x × x + y × y);
      b[0] := a[0]; b[1] := a[1] + r × b[0];
      c[n] := -a[n + 2]/j;
      c[n - 1] := -(a[n + 1] + r × c[n])/j;
      for i := 2 step 1 until n do
      begin b[i] := a[i] + r × b[i - 1] + j × b[i - 2];
        c[n - i] := -(a[n - i + 2] - c[n - i + 2] + r ×
        c[n - i + 1])/j
      end;
      comment Matching point for composite deflation;
join:
      for i := 0 step 1 until n do
      begin nfun := abs(b[i]) + abs(c[i]);
        if nfun > tol then
         begin nfun := abs(b[i] - c[i])/nfun;
        if nfun < fun then begin fun := nfun; k := i end
      end end;
      for i := k - 1 step -1 until 0 do a[i] := b[i];
      a[k] := 0·5 × (b[k] + c[k]);
      for i := k + 1 step 1 until n do a[i] := c[i];
      goto normalisation;
linear:
      rez[1] := -a[1]/a[0] × fac; imz[1] := 0·0; goto finish;
quadratic:
      r := a[1] × a[1] - 4·0 × a[0] × a[2];
      if r ⩽ 0·0 then
      begin rez[2] := rez[1] := -0·5 × a[1]/a[0] × fac;
      imz[2] := 0·5 × sqrt(-r)/a[0] × fac;
      imz[1] := -imz[2];
      goto finish end;
      imz[1] := imz[2] := 0·0;
```

```
    rez[1] := 0·5 × (−a[1] − sign(a[1]) × sqrt(r))/a[0] ×
      fac;
    rez[2] := a[2]/(rez[1] × a[0]) × fac × fac;
finish:
    n := 0;
fail: nn := n
  end of realpolsolv;

  procedure compolsolv(ar, ac, nn, rez, imz, tol, ind);
  comment This procedure attempts to solve the complex
    polynomial equation
    (ar[0] + iac[0])z ↑ nn + (ar[1] + iac[1])z ↑ (nn − 1) +
      . . . . + ar[nn] + iac[nn] = 0
    using the search algorithm proposed in Grant and Hitchins
    (1971) to limiting machine precision. On normal entry, the
    parameter ind has value zero and will remain zero on success-
    ful exit with the calculated estimates of the roots held in
    rez[k] + iimz[k], k = 1(1)nn, in approximate decreasing
    order of magnitude. The parameter tol gives the precision
    of the floating binary arithmetic used, normally 2 ↑ (−t),
    where t is the number of bits in the mantissa.
      Abnormal exits will be indicated by ind having value 1
    or 2. The former implies that either ar[0] = 0 and
    ac[0] = 0, or nn < 1. For ind = 2, a possible saddle point
    has been detected. The degree of the reduced polynomial is
    stored in nn and its coefficients are held as ar[k] + iac[k],
    k = 0(1)nn, the roots obtained thus far being stored in the
    arrays rez and imz starting with rez[nn + 1] + iimz[nn +
    1]. An immediate re-entry is possible with ind unchanged
    and with a new starting point for the search held in
    rez[1] + iimz[1];
  value tol; real tol;
  integer nn, ind; real array ar, ac, rez, imz;
  begin integer i, k, fc, n; real array br, bc, cr, cc[0: nn];
    real sig, x, y, r, rx, j, jx, fun, nfun, t, tol2, s1, s2, s, scale, g,
      fac;
    boolean sat, first, root, flag;
    procedure function;
    comment Evaluates r, j, rx, jx, at the point x + iy. The
      running complex error bound is applied if the boolean
      variable sat is true on entry. Sat is true on exit if the
      bound is satisfied;
    begin real nc, oc, nd, od, ne, oe, nf, of, ng, og, nh, oh, t, u, v,
        w, bound;
      integer i, m;
      m := n − 1;
      oe := oc := ar[0]; of := od := ac[0]; og := oh := 1·0;
        t := abs(x); u := abs(y);
      for i := 1 step 1 until m do
      begin nc := x × oc − y × od + ar[i];
        nd := y × oc + x × od + ac[i];
        ne := nc + x × oe − y × of;
        nf := nd + y × oe + x × of;
        if sat then
        begin v := og + abs(oc); w := oh + abs(od);
        ng := t × v + u × w + abs(ar[i]) + 2·0 × abs(nc);
        nh := u × v + t × w + abs(ac[i]) + 2·0 × abs(nd);
        og := ng; oh := nh end;
        oc := nc; od := nd; oe := ne; of := nf
      end;
    r := x × oc − y × od + ar[n];
    j := y × oc + x × od + ac[n];
    rx := ne; jx := nf;
    if sat then
    begin v := og + abs(oc); w := oh + abs(od);
    ng := t × v + u × w + abs(ar[n]) + 2·0 × abs(r);
    nh := u × v + t × w + abs(ac[n]) + 2·0 × abs(j);
    bound := (1 + tol) ↑ (5 × n) × tol;
    sat := abs(r) ≤ bound × ng and abs(j) ≤ bound × nh
    end end of function;
    fac := 1·0; flag := ind = 2; ind := 0; tol2 := tol × sqrt (tol);
    n := nn;
    if (ar[0] = 0·0 and ac[0] = 0·0) or n < 1 then
    begin ind := 1; goto fail end;
zerotest:
    if n = 1 then goto finish;
    if ar[n] ≠ 0·0 or ac[n] ≠ 0·0 then goto normalisation;
    rez[n] := imz[n] := 0·0; n := n − 1; goto zerotest;
normalisation:
    scale := 0·0;
    for i := 0 step 1 until n do
    if ar[i] ≠ 0·0 or ac[i] ≠ 0·0 then
    begin if abs(ar[i]) > abs(ac[i])
      then fun := abs(ar[i]) × sqrt(1 + (ac[i]/ar[i]) ↑ 2)
      else fun := abs(ac[i]) × sqrt(1 + (ar[i]/ac[i]) ↑ 2);
      if fun ≥ 10 − 5 then scale := scale + ln(fun)
    end;
    k := scale/((n + 1) × ln(2·0)); scale := 2·0 ↑ (−k);
    for i := 0 step 1 until n do
    begin br[i] := ar[i] := ar[i] × scale;
      bc[i] := ac[i] := ac[i] × scale
    end;
schur test:
    for i := n step −1 until 1 do
    begin for k := 1 step 1 until i do
      begin cr[k − 1] := br[i] × br[k] + bc[i] × bc[k] −
        br[0] × br[i − k] − bc[0] × bc[i − k];
        cc[k − 1] := br[i] × bc[k] − bc[i] × br[k] +
        br[0] × bc[i − k] − bc[0] × br[i − k]
      end;
      if cr[i − 1] < −tol then goto search;
      t := if abs (cr[i − 1]) < 1·0 then 1·0 else 1·0/
        abs(cr[i − 1]);
      for k := i − 1 step −1 until 0 do
      begin br[k] := cr[k] × t; bc[k] := cc[k] × t end
    end of schur test for zero in unit circle;
comment transformation;
    fac := 2·0 × fac; scale := 1·0;
    for i := n − 1 step −1 until 0 do
    begin scale := 2·0 × scale;
      br[i] := ar[i] := scale × ar[i];
      bc[i] := ac[i] := scale × ac[i]
    end of transformation;
    goto schur test;
search:
    sat := false; fc := 1;
    if flag then
    begin x := rez[1]; y := imz[1]; flag := false end
    else begin x := 10 − 3; y := 0·1 end;
    first root := true; function; fun := r × r + j × j;
again:
    fc := fc + 1; sat := fc > 20; g := rx × rx + jx × jx;
    if g < fun × tol2 then
    begin ind := 2; scale := 1·0;
      for i := n − 1 step −1 until 0 do
      begin scale := scale × fac; ar[i] := ar[i]/scale;
        br[i] := br[i]/scale
      end; goto fail
    end with possible saddle point detected;
    s1 := −(r × rx + j × jx)/g; s2 := (−j × rx + r × jx)/g;
      sig := 2_10 − 4;
    s := sqrt(s1 × s1 + s2 × s2); if s > 1·0 then
    begin s1 := s1/s; s2 := s2/s; sig := sig/s end;
    comment Valid direction of search has been determined,
      now proceed to determine suitable step;
    x := x + s1; y := y + s2;
loop:
    fc := fc + 1; sat := s < 10 − 5 or fc > 20;
```

```
    function; if sat then goto newroot;
    nfun := r × r + j × j;
    if fun − nfun < sig × fun then
    begin s1 := 0·5 × s1; s2 := 0·5 × s2; s := 0·5 × s;
      sig := 0·5 × sig; x := x − s1; y := y − s2; goto loop
    end;
    fun := nfun; goto again;
new root:
    rez[n] := x × fac; imz[n] := y × fac; n := n − 1;
    comment Forward and backward deflations are performed
      with the complex root;
    s := x × x + y × y; br[0] := ar[0]; bc[0] := ac[0];
    cc[n] := (y × ar[n + 1] − x × ac[n + 1])/s;
    cr[n] := (−x × ar[n + 1] − y × ac[n + 1])/s;
    for i := 1 step 1 until n do
    begin
      br[i] := ar[i] + x × br[i − 1] − y × bc[i − 1];
      bc[i] := ac[i] + x × bc[i − 1] + y × br[i − 1];
      cr[n − i] := (−x × (ar[n − i + 1] − cr[n − i + 1])
        − y × (ac[n − i + 1] − cc[n − i+ 1]))/s;
      cc[n − i] := (y × (ar[n − i + 1] − cr[n − i + 1])
        − x × (ac[n − i + 1] − cc[n − i + 1]))/s
    end;
    fun := 1·0/tol2; k := 0;
    comment Matching point for composite deflation evaluated;
    for i := 0 step 1 until n do
    begin
      nfun := abs(br[i]) + abs(cr[i])
        + abs(bc[i]) + abs(cc[i]);
      if nfun > tol then
      begin nfun := (abs(br[i] − cr[i]) +
        abs(bc[i] − cc[i]))/nfun
      if nfun < fun then begin fun := nfun; k := i end
    end end;
    for i := k − 1 step −1 until 0 do
    begin ar[i] := br[i]; ac[i] := bc[i] end;
    ar[k] := 0·5 × (br[k] + cr[k]); ac[k] := 0·5 ×
      (bc[k] + cc[k]);
    for i := k + 1 step 1 until n do
    begin ar[i] := cr[i]; ac[i] := cc[i] end;
    if n ≠ 1 then
    begin if not first root then goto normalisation;
    comment Check for conjugate complex being a root;
    y := − y; first root := false; function;
    goto if sat then new root else normalisation end;
finish:
    n := 0; nfun := ar[0] ↑ 2 + ac[0] ↑ 2;
    rez[1] := −(ar[0] × ar[1] + ac[0] × ac[1]) × fac/nfun;
    imz[1] := (ar[1] × ac[0] − ar[0] × ac[1]) × fac/nfun;
fail: nn := n
  end of compolsolv;
```

Appendix 2

Analysis of Horner's Algorithm for a complex polynomial taken at a complex point.

$$R + iJ = f(x + iy) = f(z) = \sum_{k=0}^{n} (a_k + ib_k)\, z^{n-k}$$

Exact Algorithm

$$\left.\begin{aligned} \gamma_0 &= a_0,\ \gamma_k = x\gamma_{k-1} - y\delta_{k-1} + a_k \\ \delta_0 &= b_0,\ \delta_k = y\gamma_{k-1} + x\delta_{k-1} + b_k \end{aligned}\right\} \quad k = 1, 2, \ldots, n$$

when

$$R = \gamma_n$$

and

$$J = \delta_n \ .$$

Computational

$$\left.\begin{aligned} c_0 &= a_0,\ c_k = fl(xc_{k-1} - yd_{k-1} + a_k) \\ d_0 &= b_0,\ d_k = fl(yc_{k-1} + xd_{k-1} + b_k) \end{aligned}\right\} \quad k = 1, 2, \ldots, n$$

when

$$\bar{R} = c_n$$

and

$$\bar{J} = d_n \ .$$

Using standard analysis assuming evaluation proceeds from left to right:

$$c_k = [(xc_{k-1}(1 + \varepsilon_{1,k}) - yd_{k-1}(1 + \varepsilon_{2,k}))/(1 + \eta_{1,k}) + a_r]/(1 + \eta_{2,k}) \ .$$

Multiplying through and neglecting second order terms

$$c_k(1 + \eta_{1,k} + \eta_{2,k}) = xc_{k-1}(1 + \varepsilon_{1,k}) - yd_{k-1}(1 + \varepsilon_{2,k}) + a_k(1 + \eta_{1,k})$$

or

$$c_k = xc_{k-1}(1 + \varepsilon_{1,k}) - yd_{k-1}(1 + \varepsilon_{2,k}) + a_k(1 + \eta_{1,k}) - c_k(\eta_{1,k} + \eta_{2,k}) \ .$$

Similarly

$$d_k = yc_{k-1}(1 + \varepsilon'_{1,k}) + xd_{k-1}(1 + \varepsilon'_{2,k}) + b_k(1 + \eta'_{1,k}) - d_k(\eta'_{1,k} + \eta'_{2,k}) \ .$$

Set

$$c_k = \gamma_k + e_k$$

and

$$d_k = \delta_k + f_k \ ,$$

then

$$\gamma_k + e_k = x\gamma_{k-1} + xe_{k-1} + xc_{k-1}\varepsilon_{1,k} - y\delta_{k-1} - yf_{k-1} - yd_{k-1}\varepsilon_{2,k} + a_k(1 + \eta_{1,k}) - c_k(\eta_{1,k} + \eta_{2,k})$$

or

$$e_k = xe_{k-1} + xc_{k-1}\varepsilon_{1,k} - yf_{k-1} - yd_{k-1}\varepsilon_{2,k} + a_k\eta_{1,k} - c_k(\eta_{1,k} + \eta_{2,k}) \ .$$

Similarly

$$f_k = ye_{k-1} + yc_{k-1}\varepsilon'_{1,k} + xf_{k-1} + xd_{k-1}\varepsilon'_{2,k} + b_k\eta'_{1,k} - d_k(\eta_{1,k} + \eta'_{2,k}) \ .$$

Define

$$\left.\begin{aligned} g_0 &= h_0 = 1 \\ g_k &= |x|(g_{k-1} + |c_{k-1}|) + |y|(h_{k-1} + |d_{k-1}|) + |a_k| + 2|c_k| \\ h_k &= |y|(g_{k-1} + |c_{k-1}|) + |x|(h_{k-1} + |d_{k-1}|) + |b_k| + 2|d_k| \end{aligned}\right\} \quad k = 1, 2, \ldots, n$$

then

$$|\gamma_n - c_n| = |R - \bar{R}| \leqslant \rho g_n$$

and

$$|\delta_n - d_n| = |J - \bar{J}| \leqslant \rho h_n \ .$$

Note that the recurrence is started with $g_0 = h_0 = 1$ instead of, as might have been expected, $g_0 = h_0 = 0$ to allow for certain extreme cases. Consider, for example, a polynomial in which all the coefficients are of the form $O(1) + iO(\rho)$ and suppose that it is to be evaluated in the neighbourhood of a zero at the point $x + iy = O(1) + iO(\rho)$. From the algorithm above it is clear that $\bar{J}$ is $O(\rho)$.

Consider now the recurrences for g_k and h_k. Starting with $g_0 = h_0 = 0$, it is seen that $h_n = O(\rho)$ and it is unlikely that $|\bar{J}| \leqslant \rho h_n$ will be true. On the other hand, with $g_0 = h_0 = 1$, $h_n = O(1)$ and the convergence test has some chance of being satisfied. This change in starting value is equivalent to assuming that the coefficient $a_0 + ib_0$ may be rounded. The need for the change is due to the fact that in the standard analysis, terms of second order in ρ are assumed negligible, whilst in the extreme case considered above, the significant terms of f_k are all of second order, but are not ignored.

References

ADAMS, D. A. (1967). A Stopping Criterion for Polynomial Root Finding, *CACM*, Vol. 10, pp. 655-658.

GRANT, J. A., and HITCHINS, G. D. (1971). An Always Convergent Minimization Technique for the Solution of Polynomial Equations, *JIMA*, Vol. 8, pp. 122-129.

HENRICI, P., and WATKINS, B. O. (1965). Finding Zeros of a Polynomial by the Q-D Algorithm, *CACM*, Vol. 8, pp. 570-574.

MOORE, J. B. (1967). A Convergent Algorithm for Solving Polynomial Equations, *JACM*, Vol. 14, pp. 311-315.

MOORE, J. B. (1973). *A Consistently Rapid Algorithm for Solving Polynomial Equations*, Technical Report EE 7301, Dept. of Electrical Engineering, University of Newcastle, NSW, Australia.

PETERS, G., and WILKINSON, J. H. (1971). Practical Problems Arising in the Solution of Polynomial Equations, *JIMA*, Vol. 8, pp. 16-35.

RALSTON, A. (1965). *A First Course in Numerical Analysis*, McGraw-Hill.

WILKINSON, J. H. (1965). *The Algebraic Eigenvalue Problem*, Oxford University Press.

Book review

Finite Automata, by B. A. Trakhtenbrot and Ya. M. Barzdin, 1973; translated from the Russian by D. Louvish, 362 pages. (*North-Holland*, Dfl. 60.00.)

This book describes the theoretical aspects of finite automata and their relationship to languages and *w*-languages (i.e. languages containing infinitely long words.) Starting with elementary definitions and theorems the authors go on to describe various properties of finite state languages. There is a comparison of meta-languages for finite automata and a description of current work on the identification of automata from input/output information.

Chapter 1 contains a description of outputless automata and the properties of their corresponding languages under various operations such as concatenation, iteration and the less well-known operations of projection and cylindrification. Transition graphs are introduced and there are short sections on probabilistic automata and the grammars of finite state languages.

In the second chapter automata with outputs are discussed including equivalence and decision problems. McNaughton's infinite game analogy is used to show the correspondence between operators and finite automata. In particular, it is shown that certain operators are not realisable as automata.

An account of languages for specifying automata is given in Chapter 3. This includes a description of the meta-language I which is based on propositional logic. A comparison of meta-languages demonstrates that I is the most powerful yet devised. Synthesis of automata corresponding to given descriptions is also discussed.

The last two chapters, written by Barzdin, deal with the problem of automata identification given only input/output specifications. This part of the book contains several previously unpublished results involving statistical estimation of automata parameters. These results have applications in other fields such as syntactic pattern recognition.

Although the book contains a good introduction in which the main concepts are explained carefully it is not really suitable as a first course in finite automata theory since the mathematical approach may be unfamiliar to many readers. However, it will be interesting to those who have a basic knowledge of automata theory or a mathematical background. The presentation is concise and each chapter contains supplementary problems for the reader to consider.

English readers may have difficulties with the terminology (for instance, what is usually called a finite state acceptor or recogniser is referred to as an anchored automaton) but the translator's notes are helpful as are the notes and references at the end of each chapter.

R. H. KEMP (Leicester)

Discrete Models, by D. Greenspan, 1973; 165 pages. (*Addison-Wesley Applied Mathematics and Computation Monographs*, US$16.00 hard cover, US$8.50 paperback.)

This is a disappointing book. The subject of discrete modelling or, more specifically, computer simulation of physical processes is a fascinating one and there are a number of important principles that can now be enunciated. But in his simplistic approach the author has gone too far in denying the usefulness of the concepts of limit, derivative, etc. and hence of continuous models. As a result he has all sorts of problems in trying to analyse the behaviour of his discrete models and has to present each as an inspired guess rather than the result of rational design.

For example, on page 9 stability is defined for an initial-value problem yielding $y_i = 0, 1, 2, \ldots, n$, as each $|y_i|$ being less than the largest number that can be held by one's computer! Such a vague, empty concept can hardly lead to much understanding or discrimination. On pages 11, 12 a particular and non-obvious choice of difference replacements for velocity and acceleration leads to equations for the one-dimensional motion of a particle which obey discrete momentum and energy conservation laws. Not until page 103 is it conceded that this can be done in other ways and a more symmetric scheme is given which is 'inherently more stable, and hence more economical'. A 'stability' analysis for a non-linear oscillator is given on pages 25-28 based on the behaviour as $n \to \infty$, but this fails to distinguish the vital difference between this limit and that in which $n \to \infty$ as $\triangle t \to 0$ so that $n \triangle t \to t$, nor that between linear and nonlinear problems: moreover, it is incorrectly claimed that such analyses, based on energy methods, do not appear in standard texts.

It is indeed unfortunate that such absorbing topics as Nonlinear String Vibrations, Planetary Motion and Discrete Newtonian Gravitation, the *n*-Body Problem and Discrete Fluid Models are so superficially treated in this book.

K. W. MORTON (Reading)