5. Summary and Conclusions

A numerical technique for finding an isolated root of an equation has been given. Four families of algorithms which use the technique have been described—respectively, the sectioning algorithms, the secant algorithms, the secant-tangents algorithms, and the parabola algorithms. In each iteration of one of the procedures, several approximations to the root are computed, each of which can be computed independently of any other. Hence, in a multiprocessing environment the several approximations can be computed in approximately the same time as the single approximation obtained in one iteration of a standard procedure.

Measurements of the numbers of iterations required by the procedures to achieve prescribed accuracy have been given along with measurements of the error after a fixed number of iterations. These measurements indicate that in a multiprocessing environment the given families of algorithms outperform comparable standard algorithms when the measure of performance is the time required for the computation. Estimates of the amount of computation required by the algorithms indicate that in this respect they are comparable to standard algorithms.

Acknowledgment. The author expresses his gratitude to Dr. A. L. Rosenberg and Dr. J. L. Ercolano for their suggestions and stimulating criticisms.

APPENDIX

Computation of $x_{n+1} = L(x_{n-2}, x_{n-1}, x_n, f_{n-2}, f_{n-1}, f_n)$

Let $h_n = x_n - x_{n-1}$ and $q_n = h_n/h_{n-1}$. Further let

$$A_n = q_n f_n - q_n(1 + q_n)f_{n-1} + q_n^2 f_{n-2}$$

$$B_n = (2q_n + 1)f_n - (1 + q_n)^2 f_{n-1} + q_n^2 f_{n-2},$$

$$C_n = (1 + q_n)f_n .$$

Now compute

$$q_{n+1} = \frac{-2C_n}{B_n - (B_n^2 - 4A_nC)^{\frac{1}{2}}}$$

Then $x_{n+1} = x_n + h_n q$, where q is that value of q_n which has the larger absolute value in the denominator of the right-hand term of the equation which defines q_{n+1}.

Received July, 1966; revised February, 1967

REFERENCES

1. Ostrowski, A. M. *Solution of Equations and Systems of Equations.* Academic Press, New York, 1960.
2. Henrici, P. *Elements of Numerical Analysis.* John Wiley, New York, 1964.
3. Muller, D. E. A method of solving algebraic equations using an automatic computer. *MTAC 10,* (1956), 208–215.
4. Shedler, G. S., and Lehman, M. M. On parallel algorithms and redundancy in computation. *IBM Sys. J.* to appear.

Algorithms

J. G. HERRIOT, Editor

ALGORITHM 301
AIRY FUNCTION [S20]

Gillian Bond and M. L. V. Pitteway
(Recd. 7 Apr. 1966 and 19 Oct. 1966)
Cripps Computing Centre, University of Nottingham, England

```
procedure Airy (Ai, Bi, Aid, Bid, x, xia, control);
  value x, xia, control;  real Ai, Aid, Bi, Bid, x, xia;
  integer control;
comment This procedure evaluates the real Airy functions and
  their derivatives by solution of the differential equation y'' = xy.
  The solutions Ai and Bi satisfy the Wronskian relation Ai Bi' -
  Bi Ai' = 1/π. Ai decreases exponentially for large positive
  values of x. For large negative values of x, Ai and Bi have simi-
  lar amplitudes but differ by π/2 in phase.
    The solution is tabulated in the interval -6.6 < x < 6.6 by
  Taylor integration of the differential equation in the stable
  directions (towards negative x for Ai and away from the origin
  for Bi) with step size 0.1. Alternate values are stored using 268
  locations so that any point is within Taylor range for subse-
  quent interpolation in the table. Asymptotic series are used
  outside this range. The solutions are accurate to eight decimal
  figures.
    For extensive use, computation times can be reduced by ex-
  tending the tabular range to -10 < x < 10 and changing the
  step size to 0.05, using 804 locations. The coefficients A[7] to
  A[10] may then be dropped from the asymptotic series, and
  tor [9] and tor [10] from the Taylor series (J. C. P. Miller, The
  Airy Integral, British Association Mathematical Tables, part-
  volume B, Cambridge, 1946).
    The operation of the procedure is controlled by the integer
  code. A negative value should be assigned to code to set up the
  Airy function tables on the first call for the procedure, or when-
  ever the tables have been disturbed. A subsequent entry with
  code greater than 0 will form:

  Ai = exp(xia) × Ai(x)        Aid = exp(xia) × Ai'(x)
  Bi = exp(-xia) × Bi(x)       Bid = exp(-xia) × Bi'(x)

  If the derivatives are not required, code should be set to zero.
  This will avoid asymptotic series calculations, but Aid and Bid
  are set if |x| < 6.6 even if code = 0;
begin
  real rtmdx, xi, factor, p, q, scale, s, c, xtab, h, pi;
  integer n, r, j;
  array A[0:10];
  own real array Aitab, Bitab, Aidtab, Bidtab[-33:33];
  procedure Taylor(y1, derivy1, x, h, y, derivy);
    value x, h, y, derivy;  real y1, derivy1, x, h, y, derivy;
  comment Calculates y(x+h) from y(x) by series expansion of
    dy²/dx² = xy;
  begin
    real square;
    array tor[0:10];
    integer n;
    if h = 0 then
```

```
    begin
      y1 := y;
      derivy1 := derivy;
      go to zerostep
    end shortcut
    else
    begin
      tor[0] := y;
      tor[1] := h × derivy;
      square := h × h;
      tor[2] := 0.5 × square × x × tor[0];
      y1 := tor[0] + tor[1] + tor[2];
      derivy1 := tor[1] + 2 × tor[2];
      for n := 3 step 1 until 10 do
      begin
        tor[n] := square × (x×tor[n−2]+h×tor[n−3])/
          ((n−1)×n);
        y1 := y1 + tor[n];
        derivy1 := derivy1 + n × tor[n]
      end;
      derivy1 := derivy1/h
    end calculation of coefficients in series expansion;
zerostep:
end Taylor;
  pi := 3.14159 26536;
  if control < 0 then
  begin
    Bitab[0] := 0.61492 66274;
    Bidtab[0] := 0.44828 83574;
    Aitab[33] := 2.15659 99525₁₀ − 6;
    Aidtab[33] := −5.61931 9442₁₀ − 6;
    xtab := 0;
    for n := 0 step 1 until 32 do
    begin
      Taylor(Bi, Bid, xtab, 0.1, Bitab[n], Bidtab[n]);
      Taylor(Bitab[n+1], Bidtab[n+1], xtab+0.1, 0.1, Bi, Bid);
      Taylor(Bi, Bid, −xtab, −0.1, Bitab[−n], Bidtab[−n]);
      Taylor(Bitab[−n−1], Bidtab[−n−1], −xtab−0.1, −0.1, Bi,
        Bid);
      xtab := xtab + 0.2
    end setting up Bi tables;
    for n := 33 step −1 until −32 do
    begin
      Taylor(Ai, Aid, xtab, −0.1, Aitab[n], Aidtab[n]);
      Taylor(Aitab[n−1], Aidtab[n−1], xtab−0.1, −0.1, Ai, Aid);
      xtab := xtab − 0.2
    end setting Ai tables
  end;
  if abs(x) ≦ 6.6 then
  begin
    j := 5 × x;
    xtab := j/5;
    h := x − xtab;
    scale := exp(−xia);
    Taylor(Ai, Aid, xtab, h, Aitab[j], Aidtab[j]);
    Taylor(Bi, Bid, xtab, h, Bitab[j], Bidtab[j]);
    Ai := Ai/scale;
    Aid := Aid/scale;
    Bi := Bi × scale;
    Bid := Bid × scale;
    go to finish
  end interpolation in previously established table;
  rtmdx := sqrt(abs(x));
  xi := rtmdx ↑ 3/1.5;
  factor := 1/(12×xi);
  A[0] := 1/sqrt(pi×rtmdx);
  r := 6;
  for n := 0 step 1 until 9 do
  begin
    A[n+1] := (r−1) × (r−5) × factor × A[n]/r;
    r := r + 6
  end calculation of asymptotic series coefficients;
  if x < 0 then go to neg;
  p := A[0] + A[2] + A[4] + A[6] + A[8] + A[10];
  q := A[1] + A[3] + A[5] + A[7] + A[9];
  scale := exp(xi−xia);
  Ai := (p−q)/(2×scale);
  Bi := (p+q) × scale;
  go to continue;
neg:
  p := A[0] − A[2] + A[4] − A[6] + A[8] − A[10];
  q := A[1] − A[3] + A[5] − A[7] + A[9];
  s := sin (xi+pi/4);
  c := cos(xi+pi/4);
  scale := exp(−xia);
  Ai := (p×s−q×c)/scale;
  Bi := (p×c+q×s) × scale;
continue:
  if control = 0 then go to finish
  else if x < 0 then
  begin
    p := −(rtmdx/xi) ×
      (−2×A[2]+4×A[4]−6×A[6]+8×A[8]−10×A[10]);
    q := −(rtmdx/xi) ×
      (A[1]−3×A[3]+5×A[5]−7×A[7]+9×A[9]);
    Aid := −(rtmdx×Bi)/(scale×scale)−Ai/(4×x)
      − (p×s−q×c)/scale;
    Bid := rtmdx × Ai × scale × scale − Bi/(4×x)
      − (p×c+q×s) × scale;
    go to finish
  end calculation of derivatives;
  p := (rtmdx/xi) ×
      (2×A[2]+4×A[4]+6×A[6]+8×A[8]+10×A[10]);
  q := −(rtmdx/xi) ×
      (A[1]+3×A[3]+5×A[5]+7×A[7]+9×A[9]);
  Aid := (p−q)/(2×scale) − Ai × (rtmdx+1/(4×x));
  Bid := (p+q) × scale + Bi × (rtmdx−1/(4×x));
finish:
end Airy
```

ALGORITHM 302
TRANSPOSE VECTOR STORED ARRAY [K2]

J. Boothroyd (Recd. 12 Sept. 1966, 28 Nov. 1966, and 6 Feb. 1967)
U. of Tasmania, Hobart, Tas., Australia

procedure *transpose*(*a*, *m*, *n*); **value** *m*, *n*; **integer** *m*,*n*; **array** *a*, **comment** performs an in-situ transposition of an $m \times n$ array $A[1{:}m, 1{:}n]$ stored by rows in the vector $a[1{:}m \times n]$. The method is essentially that of Windley [1], modified for use with vectors having unit lower subscript bounds.

The algorithm processes only elements $A[1, 2]$ through $A[m, n-1]$ since $A[1, 1]$ and $A[m, n]$ retain their original positions. Elements $A[q, p]$ of the transposed matrix are placed in $a[i]$, in the order $i = 2, 3, \cdots, mn - 2$, by an exchanging process. At the last step two elements are correctly placed which accounts for the value $mn - 2$ as the upper bound on i. Valid subscripts of the vector $a[1{:}m \times n]$ are elements in the 1-origin index set $[1, 2, \cdots, mn]$. Computationally, however, it is more convenient to use the zero-origin set $[0, 1, \cdots, mn-1]$. Denoting by i_0 $(i_0 = i-1)$ the corresponding zero-origin index of $a[i]$, to be occupied by $A[q, p]$, we have $i_0 = m(q-1) + (p-1)$. The corresponding zero-origin index j_0 of the $A[p, q]$ element now in $a[j]$, which must be transferred to $a[i]$, is:

$$j_0 = j - 1 = n(p-1) + (q-1) = n \times i_0 \bmod (mn-1).$$

For each value of $i = 2, 3, \cdots, mn - 2$ (or $i_0 = 1, 2, \cdots, mn-3$) we compute the index j of $a[j]$ and exchange $a[i]$ and $a[j]$ provided $j \geq i$ (i.e., $j_0 \geq i_0$). The case $j < i$ indicates that the element originally in $a[j]$ is now elsewhere following previous exchanges. Its present position is given by the first $j_r \geqq i_0$ in the series of zero-origin indices:

$$j_0, \; j_{r+1} = n \times j_r \bmod (mn-1).$$

The two sequences modulo$(mn-1)$ are generated by different methods. An additive process generates the first, using k to duplicate the function of j, in case this is adjusted in the second recurrence-generated sequence if $j < i$.

Unlike the similar problem [3], transposition does not appear to be completely soluble on wholly group-theoretic lines. A general discussion of transposition and a reference to its formulation as a problem in Abelian-Groups is given in [2].

[1] P. F. Windley, Transposing matrices in a digital computer. *Comp. J. 2* (1959), 47–48. [2] G. A. Heuer, Control Data Technical Report T.R.53, pp. 3–5. [3] Fletcher, W., and Silver, R. Algorithm 284. *Comm. ACM 9* (May 1966), 326;

```
begin integer i, j, k, iless1, mnless1, done, jn, modlessn;
  real t;
  mnless 1 := m × n − 1;  modlessn := mnless1 − n;
  done := mnless1 − 1;  k := 0;  iless1 := 1;
  for i := 2 step 1 until done do
  begin comment computes j = k = n × i0 mod(mn−1);
    j := k := if k ≦ modlessn then k + n else k − modlessn;
  test: if j < iless1 then
    begin comment computes j_{r+1} = n × j_r mod(mn−1);
        jn := j × n;
        j := jn − jn ÷ mnless1 × mnless1;
        go to test
    end;
    comment avoid unnecessary exchanges;
    if j ≠ iless1 then
    begin j := j + 1;
      t := a[i];  a[i] := a[j];  a[j] := t
    end;
    iless1 := i
  end
end transpose
```

REMARK ON ALGORITHM 28 [E2] LEAST SQUARES FIT BY ORTHOGONAL POLYNOMIALS [John G. MacKinney, *Comm. ACM 3* (Nov. 1960), 604]

G. J. Makinson (Recd. 30 Sept. 1965, 29 Aug. 1966 and 7 Nov. 1966)
University of Liverpool, Liverpool 3, England

There are three errors in the published procedure.

Line 32 $i := m + 2$; should read $i := m \div 2$;

Line 56 $delsq/(m-i-1)$; should read $delsq/(m-i-2)$;

Line 69 ; is missing from end of statement $cpoly[i+1] := s[i+1]$;

Three improvements can be made to the procedure. In the case of equally spaced points, it is possible to center them about the origin; all alphas are then zero. This is achieved by replacing the statements on lines 32, 33, and 34 by $deltax := 4/(m-1)$; $xone := -2$; All statements involving alphas can then be revised.

Another improvement can be made by deleting the two statements on line 37 and all of lines 38, 39, and 40. These statements are completely redundant.

The third improvement is to rewrite line 71 to read

clastp$[i+1] := 0$; 9: **end** *of main*

instead of

9: *clastp*$[i+1] := 0$ **end** *of main*

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.

CERTIFICATION OF ALGORITHM 279 [D1] CHEBYSHEV QUADRATURE [F. R. A. Hopgood and C. Litherland, *Comm. ACM 9*, 4 (Apr. 1966), 270]

KENNETH HILLSTROM (Recd. 16 Dec. 1966 and 30 Jan. 1967)
Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois

Work performed under the auspices of the US Atomic Energy Commission

The 40th line of the first column on page 270 should read: *badda* := .5 × (*b*+*a*);

So corrected, Chebyshev quadrature was coded in CDC 3600 ALGOL. A modified version of this quadrature scheme was coded in 3600 Compass language. In this modification the cosine values are program constants, with 3600 single-precision accuracy, as opposed to program generated values, which tests show have maximum absolute errors of 2^{-35}. These errors are carried into the integrand argument evaluation, resulting in large relative errors in the integrand evaluation, for functions bounded by unity over the interval of integration, for example, e^{-x^2} over (0, 4.3) and $\sin(x)$ over (0, 2π), which in turn delays convergence.

Since 3600 Compass does not permit dynamic allocation of storage, the dimension of the cosine array must be fixed. The choice of $129 = 2^7 + 1$ terms is based on the recommendation in the comments of Algorithm 279, "A reasonable value for *n*max is probably 7."

The Chebyshev quadrature 3600 ALGOL program, the modified 3600 Compass routine, and 3600 FORTRAN-coded Romberg and Havie integration routines were tested with six integrands. The Romberg and Havie routines are based upon Algorithm 60, Romberg Integration [*Comm. ACM 4*, (June 1961), 225], and Algorithm 257, Havie Integration [*Comm. ACM 8* (June 1965), 381].

The results of these tests are tabulated in Table I. In the table, *A* is the lower limit of the interval of integration, *B* is the upper limit, *EPS* the convergence criterion, *VI* the value of the integral, and *VA* the value of the approximation.

Due to storage requirements, Chebyshev quadrature is restricted to a maximum of 129 function evaluations. For reasons of comparison, this limit is also imposed on Romberg and Havie quadratures. Thus, in some cases the accuracy called for was not obtained.

TABLE I

Integrand	*A*	*B*	*EPS*	*VI*	*Routine*	*VA*	*Number of function evaluations*
e^{-x^2}	0	4.3	10^{-6}	0.886226924	Havie	0.886226924	17
					Romberg	0.886226925	65
					Chebyshev	0.886095576	129
					Chebyshev (Rev.)	0.886226926	17
$\sin(x) + 1$	0	2π	10^{-6}	6.283185308	Havie	6.268233308	129
					Romberg	6.268233309	129
					Chebyshev	6.282993876	129
					Chebyshev (Rev.)	6.283185309	5
$(x)^{-(1/2)}\ln\left(\frac{e}{x}\right)$	0	1	10^{-6}	6.0	Havie	5.034254231	129
					Romberg	5.034254231	129
					Chebyshev	5.829597734	129
					Chebyshev (Rev.)	5.701177427	129
$\ln(x)$	1	10	10^{-6}	14.02585088	Havie	14.02585084	65
					Romberg	14.02585085	65
					Chebyshev	14.02585096	17
					Chebyshev (Rev.)	14.02585097	17
$\ln\left(\frac{e}{x}\right)$	0	1	10^{-6}	2.0	Havie	1.979745104	129
					Romberg	1.979745104	129
					Chebyshev	1.999599461	129
					Chebyshev (Rev.)	1.997983436	129
$\frac{1}{(x^4+x^2+0.9)}$	−1	1	10^{-6}	1.5822329[a]	Havie	1.582238946	17
					Romberg	1.582238946	17
					Chebyshev	1.582232967	17
					Chebyshev (Rev.)	1.582232967	17

[a] The value $\int_{-1}^{+1} \frac{dx}{(x^4 + x^2 + 0.9)} = 1.5822329$ is obtained from C. W. Clenshaw and A. R. Curtis, "A method for numerical integration on an automatic computer," *Numer. Math. 2* (1960), 203.

Algorithms Policy • Revised August, 1966

A contribution to the Algorithms Department should be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double spaced. Authors should carefully follow the style of this department with especial attention to indentation and completeness of references.

An algorithm must normally be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1-17] or in ASA Standard FORTRAN or Basic FORTRAN [*Comm.* ACM 7 (Oct. 1964), 590-625]. Consideration will be given to algorithms written in other languages provided the language has been fully documented in the open literature and provided the author presents convincing arguments that his algorithm is best described in the chosen language and cannot be adequately described in either ALGOL 60 or FORTRAN.

An algorithm written in ALGOL 60 normally consists of a commented procedure declaration. It should be typewritten double spaced in capital and lower-case letters. Material to appear in **boldface** type should be underlined in black. Blue underlining may be used to indicate italic type, but this is usually best left to the Editor. An algorithm written in FORTRAN normally consists of a commented subprogram. It should be typewritten double spaced in the form normally used for FORTRAN or it should be in the form of a listing of a FORTRAN card deck together with a copy of the card deck. Each algorithm must be accompanied by a complete driver program in its language which generates test data, calls the procedure, and produces test answers. Moreover, selected previously obtained test answers should be given in comments in either the driver program or the algorithm. The driver program may be publishedwith the algorithm if it would be of major assistance to a user.

For ALGOL 60 programs, input and output should be achieved by procedure statements, using any of the following eleven procedures (whose body is not specified in ALGOL) [See "Report on Input-Output Procedures for ALGOL 60," *Comm. ACM 7* (Oct. 1964), 628-629]:

insymbol	*inreal*	*outarray*	*ininteger*
outsymbol	*outreal*	*outboolean*	*outinteger*
length	*inarray*	*outstring*	

If only one channel is used by the program for output, it should be designated by 1 and similarly a single input channel should be designated by 2. Examples:

outstring (1, '*x*='); *outreal* (1,*x*);
for *i* := 1 **step** 1 **until** *n* **do** *outreal* (1,*A*[*i*]);
ininteger (2, *digit* [17]):

For FORTRAN programs, input and output should be achieved as described in the ASA preliminary report on FORTRAN and Basic FORTRAN.

It is intended that each published algorithm be well organized, clearly commented, syntactically correct, and a substantial contribution to the literature of Algorithms. It is necessary but not sufficient that a published algorithm operate on some machine and give correct answers. It must also communicate a method to the reader in a clear and unambiguous manner. All contributions will be refereed both by human beings and by an appropriate compiler. Authors should pay considerable attention to the correctness of their programs, since referees cannot be expected to debug them.

Certifications and remarks should add new information to that already published. Readers are especially encouraged to test and certify previously uncertified algorithms. Rewritten versions of previously published algorithms will be refereed as new contributions and should not be imbedded in certifications or remarks.

Galley proofs will be sent to authors; obviously rapid and careful proofreading is of paramount importance.

Although each algorithm has been tested by its author, no liability is assumed by the contributor, the editor, or the Association for Computing Machinery in connection therewith.

The reproduction of algorithms appearing in this department is explicitly permitted without any charge. When reproduction is for publication purposes, reference must be made to the algorithm author and to the *Communications* issue bearing the algorithm.—J.G.Herriot