Algorithms Supplement

Previously published algorithms

The following algorithms have been published in the *Communications of the Association for Computing Machinery* during the period January–April 1967.

294 UNIFORM RANDOM
Generates the next uniformly distributed pseudorandom number on an interval (A, B) using the multiplicative congruential method.

295 EXPONENTIAL CURVE FIT
Uses a least squares method to determine the parameters a, b, c of a curve $f(x) = a + be^{-cx}$ *which approximates n data points* (x_i, y_i) *with associated weights* p_i.

296 GENERALIZED LEAST SQUARES FIT BY ORTHOGONAL POLYNOMIALS
Obtains the coefficients of the best fitting polynomial of given, or less, degree to a set of observations (x_i, f_i) *each with its associated weight* w_i.

297 EIGENVALUES AND EIGENVECTORS OF THE SYMMETRIC SYSTEM
Solves the equation $(A - \lambda B)X = 0$ *for symmetric A, B provided one of these is either positive or negative definite.*

298 DETERMINATION OF THE SQUARE-ROOT OF A POSITIVE DEFINITE MATRIX

299 CHI-SQUARED INTEGRAL
Finds the probability that X^2, *on f degrees of freedom, exceeds x.*

300 COULOMB WAVE FUNCTIONS
Evaluates the Coulomb wave functions, F_L *and* G_L, *and their derivatives for all L in the range* (0, *lmax*).

The following algorithms have been published in *Numerische Mathematik* during the period July 1966–August 1967.

(*a*) VAN DER MONDE SYSTEMS AND NUMERICAL DIFFERENTIATION
(*b*) THE JACOBI METHOD FOR REAL SYMMETRIC MATRICES
(*c*) RATIONAL CHEBYSHEV APPROXIMATION USING INTERPOLATION
(*d*) SOLUTION OF SYMMETRIC AND UNSYMMETRIC BAND EQUATIONS AND THE CALCULATION OF EIGENVECTORS OF BAND MATRICES
(*e*) NUMERICAL QUADRATURE BY EXTRAPOLATION
(*f*) CALCULATION OF THE EIGENVALUES OF A SYMMETRIC TRIDIAGONAL MATRIX BY THE METHOD OF BISECTION
(*g*) CALCULATION OF THE SINE, COSINE AND FRESNEL INTEGRALS

The following algorithms have been published in *Nordisk Tidskrift for Informationsbehandling* during the period January–December 1966.

Contributions

17 LIST PROCESSING
18 PROCEDURES FOR SIMPLIFYING BOOLEAN EXPRESSIONS
19 EIGENVALUES OF A COMPLEX MATRIX BY THE QR-METHOD

Papers

(*a*) THE SUMMATION OF SOME SERIES WITH VARIABLE COEFFICIENTS BY APPROXIMATE ANALYTICAL EXPRESSIONS
(*b*) ON THE COMPUTATION OF CERTAIN FUNCTIONS OF LARGE ARGUMENT AND PARAMETER
(*c*) A SET OF PROCEDURES MAKING REAL ARITHMETIC OF UNLIMITED ACCURACY POSSIBLE WITHIN ALGOL 60

Algorithms

Algorithm 20.

PERMUTATIONS OF THE ROWS OR COLUMNS OF A MATRIX

J. Boothroyd,
Hydro-University Computing Centre,
University of Tasmania.

procedure *permx* (*a, b, j, k, r, n, p, control*); **value** *n, control*;
real *a, b*; **integer** *j, k, n, p, control*; **integer array** *r*;
comment *A procedure using Jensen's device which may be used to permute the rows or columns of a matrix A*[1 : *n*, 1 : *n*] *in any of the ways specified by either pre-multiplying or post-multiplying A by a permuted identity matrix* I_r *or its inverse* I_r^{-1}. *The permutation matrix is not stored explicitly but is defined by the permuted identity vector r*[1 : *n*] *so that*

$$I_r[i, r[j]] = \delta_{ij},\ I_r^{-1}[r[i], j] = \delta_{ij}$$

where δ_{ij} *is Kronecker's delta.*

For example r = 3, 1, 4, 2 *defines either*

$$I_r: \begin{matrix} 0&0&1&0 \\ 1&0&0&0 \\ 0&0&0&1 \\ 0&1&0&0 \end{matrix} \quad \textit{or} \quad I_r^{-1}: \begin{matrix} 0&1&0&0 \\ 0&0&0&1 \\ 1&0&0&0 \\ 0&0&1&0 \end{matrix}$$

The parameter control should be positive to achieve pre-multiplication by I_r *or post-multiplication by* I_r^{-1}, *negative otherwise. The following examples of procedure calls illustrate how to obtain each of the four possible operations.*

1. *Premultiplication by* I_r : *permx*(*A*[*j*,*p*],*A*[*k*,*p*],*j*,*k*,*r*,*n*,*p*,1)
2. *Postmultiplication by* I_r : *permx*(*A*[*p*,*j*],*A*[*p*,*k*],*j*,*k*,*r n p*,−1)
3. *Premultiplication by* I_r^{-1}: *permx*(*A*[*j*,*p*],*A*[*k*,*p*],*j*,*k*,*r*,*n*,*p*,−1)
4. *Postmultiplication by* I_r^{-1}: *permx*(*A*[*p*,*j*],*A*[*p*,*k*],*j*,*k*,*r*,*n*,*p*,1);

```
begin integer i, ri, t, q;  real w;
  procedure ritoi;
  comment uses the permutation vector r[1 : n] to achieve
    a permutation of the form NEW[i]:= OLD[r[i]],
    i = 1, 2, . . ., n;
    begin for i := n step − 1 until 2 do
    begin ri := r[i];
    L: if i≠ri then
      begin if ri>i then
        begin ri := r[ri];
          goto L
        end;
        j := i; k: = ri;
        for p := 1 step 1 until n do
        begin w= a;
          a:= b;
          b:= w
        end
      end
    end
  end ritoi;
  procedure itori;
  comment uses the permutation vector r[1 : n]
  to achieve the inverse permutation to that of
  procedure ritoi viz NEW[r[i]]:= OLD[i], i = 1, 2, . . ., n;
  begin for t := n step − 1 until 2 do
    begin i:= t; ri:= r[i];
    L: if i≠ri then
      begin for q:= t+1 step 1 until n do
        if i=r[q] then
        begin i:= q;
          goto L
        end;
        j:= i; k:= ri;
        for p:= 1 step 1 until n do
        begin w:=a;
          a:= b;
          b:= w
        end
      end
    end
  end itori;
  if control>0 then
  ritoi
  else
  itori
end permx
```

Algorithm 21.

SOLUTION OF POLYNOMIAL EQUATION USING THE METHOD OF BAIRSTOW

J. M. Watt,
Department of Computational and Statistical Science,
The University,
Liverpool, 3.

Author's Note:

The algorithm uses the method of Bairstow (Wilkinson (1963), Ralston (1965)) to find approximations to the roots of a polynomial equation of degree *n*.

When run using the Whetstone compiler on the KDF9 computer (which uses floating point numbers with a precision of 12½ decimal digits in the range from 10^{-38} to 10^{+38}) with *eps*[1] = *eps*[2] = *eps*[3] = $_{10}-6$ the Bairstow algorithm successfully found all the roots of the 45 polynomial equations given by Henrici and Watkins (1965) as corrected by Thomas (1966) with the following exceptions.

(1) The roots of the 13th polynomial of Table 2, which is of degree 36, were not found, the computer failing due to calculated numbers falling outside the allowed range. (The remaining polynomials had degree 19 or less.)

(2) In the case of polynomials 4, 6, 9, 10, 11 and 12 of Table 1 which have repeated or very close zeros the accuracy of 5 significant figures was not always achieved. When in an attempt to achieve higher accuracy all the polynomials were re-run with *eps*[1] = 0, *esp*[2] = *eps*[3] = $_{10}-6$ the process did not converge in 250 iterations for polynomials 4 and 11. The remaining polynomials never needed more than 84 iterations per quadratic factor, and 5 figure accuracy was achieved in all zeros except those of polynomial 12 of Table 1 which has the zero 2 repeated four times.

(3) A further error was found in Henrici's results for polynomial 18 of Table 1, which has zeros as follows:

$$-0{\cdot}49591 \pm 0{\cdot}90230,\ 1{\cdot}6553 \pm 2{\cdot}2243,\ 3{\cdot}6813$$

References

RALSTON, A. (1965). *A First Course in Numerical Analysis.* New York: McGraw-Hill Book Co.

WILKINSON, J. H. (1963). *Rounding Errors in Algebraic Processes.* London: H.M.S.O.

HENRICI, P. and WATKINS, B. O. (1965). Finding Zeros of a Polynomial by the Q-D Algorithm, *Communications of the Association for Computing Machinery*, Vol. 8, pp. 570–4.

THOMAS, R. F. (1966). Corrections to the Numerical Data on Q-D Algorithm, *Communications of the Association for Computing Machinery*, Vol. 9, pp. 322–3.

procedure *Bairstow* (*n*,*a*,*x*,*y*,*eps*,*N*,*nor*);
value *N*, *eps*, *n*; **integer** *N*,*n*,*nor*; **array** *a*,*x*,*y*,*eps*;
comment. *The procedure uses the method of Bairstow to find approximations to the roots of the nth degree polynomial equation with real coefficients.*

$$a[0]x^n + a[1]x^{n-1} + \ldots + a[n] = 0.$$

(But the quadratic factors of the deflated polynomial are not "purified" in the original polynomial, due to the difficulty of ensuring that convergence of an ill-conditioned factor occurs to the "same" factor in the original polynomial.) On exit the real and imaginary parts of the rth root are given in x[r] and y[r] for r = 1, 2, . . ., (n − nor) respectively. Zero roots are given first and there is a tendency for the remaining roots to be given in order of increasing magnitude. The first root of a complex pair has positive imaginary part, and its exact conjugate follows immediately. The result "nor" is the number of roots that the procedure has been unable to find. The maximum number of iterations used to find a single quadratic factor is "N". (A value of N = 100 is recommended although most factors will be found with about 10 iterations.)

A quadratic factor $x^2 + px + q$ *is accepted as soon as*

1. *The remainder when the original polynomial is divided by the product of this factor and all those accepted so far has coefficients less than eps*[1] *in modulus.*

2. *The calculated corrections to p and q are less than eps*[2] *in modulus.*
3. *The relative corrections to p and q are less than eps*[3] *in modulus.*

Precautions are taken to avoid division by zero, and to avoid large corrections to p and q. Zero leading coefficients are first removed—if all coefficients are zero, nor := $n + 1$. *Then zero roots are removed before the main iterative cycle is entered. If the resultant polynomial is of odd order, the linear factor is found last.*

Parameters
n: *The order of the polynomial,* $n \geqslant 1$
a: *The polynomial is a* $[0]x^n + a[1]x^{n-1} + \ldots + a[n]$
x: *y*: *The roots are* $x[r] + iy[r]$ *for* $r = 1, 2, \ldots, (n - nor)$
eps: *eps*[1], *eps*[2], *eps*[3] *are the precisions described above. A zero value of eps*[1] *can be used but eps*[2] *and eps*[3] *should be non-zero.*
N: *The maximum number of iterations allowed to find a quadratic factor.*
nor: *The number of roots that the procedure has been unable to find;*

```
begin array b,c[−1 :n]; integer nl,m,i,j,k;
  real p,q,T,M,det0,det1,det2,dp,dq,T0,T1,r0,r1;
  b[0]:= c[0]:= 1; p:= q:= b[−1]:= c[−1]:= 0;
  comment Remove zero leading coefficients;
  for i:= 0 step 1 until n do if a[i]≠0 then goto A;
  nor:= n+1; goto Exit;
A: T:= 1/a[i]; nor:= i; n:= n−i;
  for j:= 1 step 1 until n do b[j]:= a[i+j]×T;
  comment Remove zero roots;
  for m:= 0 step 1 until n do
  if b[n−m]≠0 then
  goto B
  else
  x[m+1]:= y[m+1]:= 0;
B: for nl:= n−m step −2 until 2 do
  begin comment This loop is obeyed once for each quadratic
    factor;
    m:= n−nl;
    comment Find quadratic factor;
    if nl>2 then
    begin p:= p+eps[2]; q:= q+eps[2];
      comment Set up initial values of p,q;
      for k:= 1 step 1 until N do
      begin comment Iterate for quadratic factors;
      step: for i:= 1 step 1 until nl do
        c[i]:= b[i]−p×c[i−1]−q×c[i−2];
        for i:= 1 step 1 until nl−2 do
        c[i]:= c[i]−p×c[i−1]−q×c[i−2];
        r0:= c[nl]; r1:= c[nl−1];
        T0:= c[nl−2]; T1:= c[nl−3];
        M:= q×T1+p×T0; det0:= T0×T0+M×T1;
        det2:= T0×r0+M×r1; det1:= T0×r1−T1×r0;
        comment Check for convergence;
        if abs(r0)<eps[1] ∧ abs(r1)<eps[1] then
        goto reduce;
        M:= abs (det1)+abs(det2);
        if M=0 then
        begin comment Take action if the equations for dp
          and dq are singular;
          p:= 1·01×p+eps[2];
          q:= 1·01×q+eps[2]
        end
        else
        begin comment Find dp and dq making sure that they
          are not too large. Check for convergence;
          M:= M/(10×(abs(p)+abs(q)+100×eps[2]));
          M:= 1/(if M<abs(det0) then
          det0
          else
          if det0⩾0 then
          M
          else
          −M);
          dp:= M×det1; p:= p+dp;
          dq:= M×det2; q:= q+dq;
          if abs(dp)<eps[2] ∧ abs(dq)<eps[2]
           ∨ abs(dp)<abs(p)×eps[3] ∧ abs(dq)
          <abs(q)×eps[3] then
          goto reduce
        end
      end k: Exit is to the Label reduce if convergence is
      achieved in less than N iterations;
      nor:= nor+nl; goto Exit;
      comment Divide out by quadratic factor;
      reduce: for i:= 1 step 1 until nl−2 do
      b[i]:= b[i]−p×b[i−1]−q×b[i−2]
    end
    else
    begin comment if the polynomial has been
      reduced to a quadratic;
      p:= b[1]; q:= b[2]
    end;
    comment Find zeros of quadratic factors;
    M:= −0·5×p; T:= M×M−q;
    if T>0 then
    begin x[m+2]:= M+(if M>0 then
      sqrt(T)
      else
      −sqrt(T));
      x[m+1]:= q/x[m+2];
      y[m+1]:= y[m+2]:= 0
    end
    else
    begin x[m+1]:= x[m+2]:= M;
      y[m+1]:= sqrt(−T);
      y[m+2]:= −y[m+1]
    end
  end nl: All pairs of zeros have now been found;
  comment Treat possible remaining linear factor;
  if ((n−m)÷2)×2≠n−m then
  begin x[n]:= −b[1]; y[n]:= 0
  end;
Exit:
end Bairstow
```

Note on Algorithm 19. *COMPLEX*

B. H. Rudall,
Computing Laboratory,
University College of North Wales.

An alternative and more concise form of the procedure *arg* has been suggested to me by I. D. Hill, Medical Research Council. This version has been amended to include a jump to a non-local label *indeterminate* when the form $a[0] = a[1] = 0$ occurs on entry to the procedure.

```
real procedure arg(a); value a; array a;
comment assigns the argument of a to the identifier arg
where -pi < arg ≤ pi;
begin real pi; pi := 3·141592654;
  if a[0] = 0 then
  begin  if a[1] = 0 then
    goto indeterminate
    else
    arg := 0·5 × pi × sign(a[1])
  end
  else
  begin  real w;
    w := arctan(a[1]/a[0]);
    if a[0] < 0 then
    w := if w > 0 then
    w - pi
    else
    w + pi;
    arg := w
  end
end arg
```

Correspondence

The Editor,
The Algorithms Supplement,
The Computer Journal.

Sir,

I must protest, in a friendly way, at the sentence in your Editor's comment (1967) which reads *Efficiency of computer use should not be a consideration.*

I hope that what you meant was that an algorithm that takes a lot of computer time should not be ruled out solely for this reason. Some jobs are necessarily lengthy, and the use of *necessary* computer time should not be regarded as a sin. Your remark appears, however, to mean that the use of *unnecessary* computer time is not a sin either, and this is quite unacceptable.

What do people want from published algorithms? I suggest that the main requirement is that when one wishes to perform an operation on a computer, for which a published algorithm exists, one should be able to take that algorithm and incorporate it into one's program with a high degree of confidence that it will produce the right answer, and that it will do so efficiently.

I have heard it argued that if one is interested in efficiency, then one should not be using ALGOL. This reminds me of the argument that because a lot of money has been spent on Project A, therefore a lot is available for Project B. Exactly the reverse is true. If the programming language is inefficient, it is even more necessary that the program should be as efficient as possible, within the limitations of that language.

ALGOL is an accepted language for publications because of its international status, its availability on a wide range of machines, its clear and unambiguous definition in the ALGOL Report (notwithstanding a few minor errors and difficulties), and its easy intelligibility to the human reader. But one does not necessarily have to retain the ALGOL language to use a published algorithm; often translations are made. If, for the sake of increased speed, one translates an ALGOL procedure into machine code, one does not wish to be translating an inefficient method.

Those of us who believe in ALGOL, as the language in which we wish to do the great majority of our computer work, are interested in the efficiency of the entire operation—not only the use of computer time, but the use of human time as well. The efficiency of being able to write things down and expect them to work correctly first time; the efficiency of being able to take separately written procedures and throw them together to make a program, knowing that they will not be incompatible; the efficiency of being able to look at a program a year after it was written and see at a glance what it is doing—these factors are not to be despised.

I. D. Hill,
Medical Research Council,
115 Gower Street,
London W.C.1.

Reference

Algorithms Supplement, *The Computer Journal*, Vol. 9, p. 418.

Editor's comment

The statement about *Efficiency of computer use* which has been mentioned above was intended to be interpreted as follows.

Certain classes of algorithm, for example those connected with compiling techniques and other branches of non-numerical analysis, become very inefficient when written in ALGOL 60. Nevertheless such algorithms will not be rejected because they are inefficient. They are of value as an aid to human understanding.

Clearly all algorithms, particularly those which are intended for direct machine use, should be as efficient as it is possible to be within the limitations of the language used. However, an algorithm must not be penalized because the languages available are unsuitable.

This difficulty can be eased by extending the range of languages in which algorithms may be written. This consideration, together with the ones above and those of Mr. Hill, have been included in the *Statement of Policy* given below.

I shall be pleased to hear from anyone who feels that some point made in the *Statement of Policy* is not clear or that the statement is inadequate.

Statement of Policy

A contribution to the Supplement may consist of an algorithm, a note on a previous algorithm or a letter to the Algorithms Editor.

A contribution which is a new algorithm must consist of a program or a section of program written in any fully documented and widely used advanced programming language. In this context ALGOL 60 (1963), FORTRAN (1966) and COBOL (1967) are acceptable. Other similar languages are acceptable, but only if the submitted algorithm exhibits features which cannot adequately be described in any of the above languages. Symbolic assembly languages are not acceptable. The decision as to whether a particular language is acceptable for a given algorithm rests with the editor.

An algorithm written in ALGOL 60 must be a self-contained procedure. An algorithm written in FORTRAN must be a self-contained sub-program. An algorithm written

in COBOL must be a complete program. In this context, self-contained means that the algorithm must not use any non-local identifiers (other than standard function names) or any COMMON areas and that all input–output must be through formal parameters. An algorithm written in ALGOL 60 or FORTRAN may consist of a set of related procedures or sub-programs. In this case the algorithm must include a calling sequence showing how the procedures are to be used (e.g. see Rudall, 1967). These conditions may be relaxed but only if the author can provide sufficient reason for so doing.

An algorithm should include comment and may be accompanied by an author's note where this would be helpful. Information describing the organization of the computation, the standard constants used, etc. should be given in the comment. Information describing the method used, the environment in which the algorithm has been written and tested, the computing times achieved, etc. should be given in the author's note. References must be given in accordance with *Notes on the submission of papers*, a revised version of which was first published in Vol. 10, No. 1, p. 120.

Algorithms which are published may be of two kinds, those which are for computer use and those which are for human understanding. Algorithms for computer use must satisfy the rigid conditions laid down below but need not be fully commented. Algorithms which are published in order to indicate a method, which is intended to be understood and adapted before it is used on a computer, need full commenting but need not necessarily satisfy the conditions on machine efficiency. If an algorithm is of the latter type then this should be clearly stated in the author's note.

An algorithm must be written to conform with the appropriate reference document (see references below), it must be submitted in duplicate and be typewritten double-spaced. Where material is to appear in bold face it should be underlined in black. Where the correct character does not exist on a typewriter it should be inserted neatly by hand in black and not replaced by a similar typewritten character (e.g. $\geq$ is not an acceptable substitute for $\geqslant$). Where the language itself, or easy reading of the language, demands it, correct indentation is essential. The algorithms published in this issue show what is required in the case of ALGOL 60.

An algorithm must be accompanied by a driver program incorporating it, test data and test results. These must also be in duplicate but may be in a modified version of the language applicable to a particular computer and may have been prepared on the off-line editing equipment of that computer. Authors should be prepared, if asked, to supply one copy of the paper tape or cards used. For algorithms written in COBOL the driver program and algorithm will be the same.

For an algorithm to be accepted for publication it must either be original or be an improvement on or an extension of an existing algorithm. It must also make a substantial contribution to the existing stock of algorithms. It must be syntactically correct, produce the results claimed, not lead to unnecessary demands on computer time and/or space and use the language specified in as neat a way as possible. It must not use constructions whose results are not uniquely defined (e.g. $y := x + f(x)$ where $f(b)$ is a function which alters the value of b). Where cases of failure may occur they must be explicitly mentioned in the comment. Where the specified language allows sections of program written in machine code, these may be referred to by a suitable calling sequence, provided that the effects of the sections are trivial and are fully explained in the comment. Approximate numerical constants must be written as constants and the constants must be defined in the comment.

Every effort is made to see that all published algorithms are completely reliable. In particular, all algorithms are submitted to independent referees and extensively checked. Certifications must add something to what is known about the algorithm and must take the form of a note. Notes which point out defects in or suggest improvements to previously published algorithms are particularly welcome. To help to prevent printing mistakes, galley proofs will be sent to the authors wherever possible. Whilst every effort is made to publish worthwhile algorithms, no liability is assumed by any contributor, the editor or *The Computer Journal* in connection therewith.

 In the event of the formation of a National Library of Algorithms, all algorithms which have appeared in *The Computer Journal* will be made available to this Library.

A small number of reprints will be available to authors.

References

(1963) Revised Report on the Algorithmic Language ALGOL 60, *The Computer Journal*, Vol. 5, pp. 349–67.

(1966) A.S.A. FORTRAN.

(1967) A.S.A. COBOL. (In draft)

RUDALL, B. H. (1967). *Complex*, Algorithm 19, *The Computer Journal*, Vol. 10, pp. 112–13.

Contributions to the Algorithms Supplement should be sent to

P. Hammersley
University Mathematical Laboratory
Corn Exchange Street
Cambridge