Algorithms Supplement

As already announced, the B.C.S. Algorithms Supplement, formerly published in *The Computer Bulletin*, will in future be published instead in this *Journal*. All correspondence concerning the Supplement should be sent to the Algorithms Editor:

P. Hammersley
Northampton College of Advanced Technology
St. John Street
London EC1.

Published algorithms

The following algorithms have been published in the *Communications of the Association for Computing Machinery* between September 1965 and December 1965.

263* GOMORY 1

An improved version of Algorithm 153 *GOMORY which determines the integer solution of a linear programming problem with integer coefficients only.*

264* INTERPOLATION IN A TABLE

Evaluation of a function by polynomial interpolation in a table of values.

265 FIND PRECEDENCE FUNCTIONS

266 PSEUDO-RANDOM NUMBERS

Generates a pseudo-random number in the open interval (*a*, *b*)

267 RANDOM NORMAL DEVIATE

Produces two independent random variables x1 *and* x2 *each from the normal distribution with mean* 0 *and variance* 1. *This procedure uses* 266.

268 ALGOL 60 REFERENCE LANGUAGE EDITOR

269 DETERMINANT EVALUATION

Evaluates a determinant by triangularization with searching for pivot in row and with scaling of the rows of the matrix before the triangularization.

270 FINDING EIGENVECTORS BY GAUSSIAN ELIMINATION

271 QUICKERSORT

Sorts the elements of an array into ascending order by continually splitting the array into parts such that all elements of one part are less than all elements of the other, with a third part in the middle consisting of a single element.

272 PROCEDURE FOR THE NORMAL DISTRIBUTION FUNCTIONS

* The numbers 263 and 264 were each inadvertently assigned to two algorithms. When giving references to these algorithms, please be careful to do so unambiguously.

Algorithms

Author's note on Algorithms 10 and 11 (below)

Neville's method of evaluating the Lagrange interpolating polynomial has been so widely used that there would appear to be general agreement with Lance (1) that it is the most

(1) LANCE, G. N. *Numerical Methods for High-Speed Computers*, p. 142.

satisfactory method for automatic computation. An alternative scheme is that of Aitken and a comparison of both methods, which are the same in principle and different only in detail, shows that Aitken's method results in a more efficient ALGOL procedure than Neville's method.

For the set of $n + 1$ sample points $\{(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)\}$ the two methods of evaluating the interpolating polynomial for some specified argument u may be described essentially as follows.

NEVILLE

```
for i:= n − 1 step − 1 until 0 do
for j:= 0 step 1 until i do
    f[j]:=f[j] + (u − x[j]) × (f[j + 1]
      − f[j]) × (x[n − i + j] − x[j]);
    Neville:= f[0];
```

AITKEN

```
for i:= 0 step 1 until n − 1 do
for j:= i + 1 step 1 until n do
    f[j] :=f[i] + (u − x[i]) × (f[j] −f[i])/(x[j] − x[i]);
    Aitken :=f[n];
```

After taking all possible steps to improve the run-time efficiency of both processes by following the principle of never causing a computer to perform, more than once, any operation for which once is sufficient, the respective times for 5000 entries to each procedure with $n = 5$ were *AITKEN* 80 seconds *NEVILLE* 90 seconds.

Algorithm 10. *AITKEN* J. Boothroyd,
Computing Centre,
University of Tasmania.

real procedure *aitken* (*x*, *y*, *arg*, *n*, *m*); **value** *arg*, *m*, *n*; **array** *x*, *y*; **real** *arg*; **integer** *m*, *n*; **comment** *array y* [0: *n*] *contains sample values of a function at corresponding values of the argument contained in x*[0: *n*] *which is assumed to have been sorted in ascending order. The procedure yields an approximation to the function at the specified value arg by evaluating an mth order*

polynomial ($m < n$). The subset of $m + 1$ points used in the evaluation are suitably chosen to be evenly distributed about the value of arg. If the requested value of m exceeds n, the assignment $m := n$ occurs.
For arg $<$ x[0] the procedure extrapolates using x[0], x[1], . . ., x[m]
For arg $>$ x[n] the procedure extrapolates using x[n − m], x[n − m + 1], . . . ,x[n];

```
begin   integer i, j, mlessl; real fi, zi; real array z, f[0: m];
        integer procedure setmin(L); label L;
        begin integer i;
        for i:= 0 step 1 until n do if arg < x[i] then
        goto found;
        i:= n;
found:  if arg = x[i] then begin f[m]:= y[i]; goto L
        end;
        i:= i − m ÷ 2 − 1;
        setmin:= if i < 0 then 0 else if i + m > n
        then n − m else i
        end setmin;
        if m > n then m:= n; j:= setmin (out);
        for i:= 0 step 1 until m do begin z[i]:= arg − x[j];
        f[i]:= u[j]; j:= j + 1 [end;]
        end;

        mlessl:= m − 1;
        for i:= 0 step 1 until mlessl do
        begin fi:= f[i]; zi:= z[i];
        for j:= i + 1 step 1 until m do
        f[j]:= fi + zi × (f[j] − fi)/(zi − z[j])
        end;
out:    aitken:= [m]
end
```

Algorithm 11. *EQUIPOL* J. Boothroyd, Computing Centre, University of Tasmania.

```
real procedure equipol (xbase,y,arg,n,m,h);
    value xbase,arg,m,n,h;
real xbase,arg,h; array y; integer m,n;
    begin integer i,j,mlessl; real jh,fi; array f[0:m];
        if m>n then m:= n; i:= entier((arg-xbase)/h)−m—2;
        j:= if i<0 then 0 else if i+m>n then n−m else i;
        for i:= 0 step 1 until m do f[i]:= y[i+j];
        arg:= arg−j × h;
        mlessl:= m−1;
        for i:= 0 step 1 until mlessl do
            begin fi:= f[i]; jh:= h;
                for j:= i+1 step 1 until m do
                    begin f[j]:= fi+arg × (f[j]−fi)/jh;
                        jh: =jh+h end ;
                arg:= arg−h
            end;
        equipol:= f[m]
    end equipol
```

Note on Algorithm 4. *TWOBYTWO* I. D. Hill and M. C. Pike, Medical Research Council, Statistical Research Unit.

Two errors in this algorithm (published in *The Computer Bulletin*, Vol. 9, p. 56 (Sept. 1965)) have been discovered.

The section of program immediately before the label *CHECK* 1

"**if** *method* = 1 **then begin** *E*:= *d*1/*d*2;"

should read

"*E*:= *d*1/*d*2; **if** *method* = 1 **then begin**"

The original version can lead to a considerable waste of computing time but not to an incorrect result.

Immediately after "**end** *sumterms*;" the statement "*count*:= 1" should be added. The original version fails if $a = 0$ and *method* = 2 since in this case the test for $N \leqslant$ *count* is made while *count* has not been given a value. This error was discovered by D. T. Muxworthy of the Norwegian Computing Centre.

Editor's Note

In recent editions of the *Communications of the Association for Computing Machinery* the "Revised Algorithms Policy—May, 1964" has been published, outlining certain requirements to which all submitted algorithms must conform.

No similar policy has yet been defined for the Algorithms Supplement. However, all algorithms submitted for publication must be submitted in duplicate and be accompanied by a driver program, a set of test results and a detailed description of the origin and purpose of the algorithm.

All algorithms which are accepted for publication are published in the reference language. Hence, features which cannot be directly translated into the reference language must not be included in an algorithm. For example

"*m* := *n* **div** 2"

may be included but

"**print** *newline*,*x*,*y*"

may not. However, such features may be included in the driver program.