tive functions and general framework on which any desired algorithm can be built. In certain instances the position of the report seems to be at variance with these principles, and it might be worthwhile to re-examine the document with respect to these points.

Appendix

Illegal Character Check. Outside of strings, only the following characters are legal:

0 through 9
$+ - . ,$ ⟨blank⟩ $_{10}$

Count Check Over a Format Item

$+$ or $-$: Not more than two
$.$: Once only
$_{10}$: Once only

Immediate Sequence Check

Previous Symbol	*Current Symbol*					
	0–9	+ or −	.	$_{10}$	*Blank*	,
0–9		E*			E*	
+ or −		E				E
.		E	E	E		E
$_{10}$			E	E		E
Blank						
,		E	E	E		E

* Allowed only if in the last field of segment *and* if required there by format.

Remote Sequence Check (after the appearance of first digit)

Current Symbol	*Action*
0–9	
+	
−	
$_{10}$	
blank	E*
,	

Format Conformance Check

Expected Symbol	*Actual Symbol*					
	0–9	+ or −	.	$_{10}$	*Blank*	,
D or Z		*	W	W	*	W
+ or −	W		W	W		E
.	W	W		W	W	W
$_{10}$	W	W	W		W	W
C	W	W	W	W	W	

W = warning only
E = nonrecoverable error
* = OK if to left of first digit encountered

Algorithms

G. E. FORSYTHE, J. G. HERRIOT, Editors

ALGORITHM 251
FUNCTION MINIMISATION [E4]

M. Wells (Recd. 13 July 1964 and 5 Oct. 1964)
Electronic Computing Lab., U. of Leeds, England

```
procedure FLEPOMIN (n, x, f, est, eps, funct, conv, limit, h,
        loadh);
  value n, est, eps, loadh, limit;
  real f, est, eps;  integer n, limit;  Boolean conv, loadh;
  array x, h;  procedure funct;
comment function minimisation by the method of Fletcher
  and Powell [Comput. J. 6, 163-168 (1963)]. On entry x[1:n] is an
  estimate of the position of the minimum, est an estimate of the
  minimum value, eps a tolerance used in terminating the proce-
  dure when the first derivative of f nearly vanishes, and loadh
  indicates whether or not an approximation to the inverse of the
  matrix of second derivatives of f is available. If loadh is true
  the procedure supplies the unit matrix as this estimate, other-
  wise it is assumed that the upper triangle of a symmetric posi-
  tive definite matrix is stored by rows in h[1:n × (n+1)/2].
  The statement funct (n, x, f, g) assigns to f the function value
  and to g[1:n] the gradient vector.
    A successful exit from FLEPOMIN, with conv true, occurs
  if two successive values of f are equal, or if a new value of f
  is larger than the previous value (due to rounding errors), or
  if after n or more iterations the lengths of the vectors s and
  sigma are less than eps. If the number of iterations exceeds
  limit, then an exit occurs with conv false. In either case, the
  final function value, estimated position of the minimum and
  inverse matrix of second derivatives are in f, x and h;
begin
  real oldf, sg, ghg;
  integer i, j, k, count;
  array g, s, gamma, sigma [1:n];
  real procedure dot (a, b);
    array a, b;
    comment inner product of a and b [In this procedure and
      in up dot greater accuracy would be obtained by accumulat-
      ing the inner products in double precision.—Ref.];
    begin integer i;  real s;  s := 0;
      for i := 1 step 1 until n do s := s + a[i] × b[i];
      dot := s
    end of dot;
  real procedure up dot (a, b, i);
    value i;
    array a, b;  integer i;
    comment multiply b by the ith row of the symmetric
      matrix a, whose upper triangle is stored by rows;
    begin integer j, k;  real s;  k := i;  s := 0;
      for j := 1 step 1 until i − 1 do
        begin s := s + a[k] × b[j];  k := k + n − j end steps
          to diagonal. Now go along row;
      for j := i step 1 until n do s := s + a[k+j−i] × b[j];
      up dot := s
    end of up dot;
set initial h:
  if loadh then
```

```
  begin k := 1;
    for i := 1 step 1 until n do
      begin h[k] := 1;
        for j := 1 step 1 until n − i do h[k+j] := 0;
        k := k + n − i + 1
      end
  end formation of unit matrix in h;
start of minimisation:
  conv := true;
  funct (n, x, f, g);
  for count := 1, count + 1 while oldf > f do
  begin oldf := f;
    for i := 1 step 1 until n do
    begin sigma[i] := x[i];  gamma[i] := g[i];
      s[i] := −up dot(h, g, i)
    end preservation of x, g and formation of s;
```

Revised Algorithms Policy • May, 1964

A contribution to the Algorithms department must be in the form of an algorithm, a certification, or a remark. Contributions should be sent in duplicate to the editor, typewritten double-spaced in capital and lower-case letters. Authors should carefully follow the style of this department, with especial attention to indentation and completeness of references. 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 must be written in the ALGOL 60 Reference Language [*Comm. ACM 6* (Jan. 1963), 1-17], and normally consists of a commented procedure declaration. Each algorithm must be accompanied by a complete driver program in ALGOL 60 which generates test data, calls the procedure, and outputs 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 published with the algorithm if it would be of major assistance to a user.

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

```
procedure inreal (channel, destination); value channel; integer channel;
  real destination; comment the number read from channel channel is
  assigned to the variable destination; . . . ;
procedure outreal (channel, source); value channel, source; integer channel;
  real source;  comment the value of expression source is output to channel
  channel; . . . ;
procedure ininteger (channel, destination);
  value channel; integer channel, destination; . . . ;
procedure outinteger (channel, source);
  value channel, source;  integer channel, source; . . . ;
procedure outstring (channel, string); value channel; integer channel;
  string string; . . . ;
```

If only one channel is used by the program, it should be designated by 1. Examples:

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

It is intended that each published algorithm be a well-organized, clearly commented, syntactically correct, and a substantial contribution to the ALGOL literature. All contributions will be refereed both by human beings and by an ALGOL compiler. Authors should give great attention to the correctness of their programs, since referees cannot be expected to debug them. Because ALGOL compilers are often incomplete, authors are encouraged to indicate in comments whether their algorithms are written in a recognized subset of ALGOL 60 [see "Report on SUBSET ALGOL 60 (IFIP)," *Comm. ACM 7* (Oct, 1964), 626-627].

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 the 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.—G.E.F.

```
    search along s:
      begin real ya, yb, va, vb, vc, h, k, w, z, t, ss;
        yb := f;  vb := dot(g, s);  ss := dot(s, s);
        if vb ≧ 0 then go to skip;
        k := 2 × (est-f)/vb;
    scale:  h := if k > 0 and k ↑ 2 × ss < 1 then k else 1/sqrt(ss);
        k := 0;
    extrapolate:  ya := yb;  va := vb;
        for i := 1 step 1 until n do x[i] := x[i] + h × s[i];
        funct(n, x, f, g);
        yb := f;  vb := dot(g, s);
        if vb < 0 and yb < ya then
        begin h := k := h + k;  go to extrapolate end;
        t := 0;
    interpolate:  z := 3 × (ya−yb)/h + va + vb;
        w := sqrt(z ↑ 2−va×vb);
        k := h × (vb+w−z)/(vb−va+2×w);
        for i := 1 step 1 until n do x[i] := x[i] + (t−k) × s[i];
        funct(n, x, f, g);
        if f > ya or f > yb then
        begin vc := dot(g, s);
          if vc < 0 then
          begin ya := f;  va := vc;  t := h := k end
            else
          begin yb := f;  vb := vc;  t := 0;  h := h − k end;
          go to interpolate
        end;
    skip:  end of search along s;
      for i := 1 step 1 until n do
      begin sigma[i] := x[i] − sigma[i];
        gamma[i] := g[i] − gamma[i]
      end;
      sg := dot(sigma, gamma);
      if count ≧ n then
        begin if sqrt(dot(s, s)) < eps and sqrt(dot(sigma, sigma)) <
          eps then go to finish
        end test for vanishing derivative;
      for i := 1 step 1 until n do s[i] := up dot(h, gamma, i);
      ghg := dot(s, gamma);
      k := 1;
      for i := 1 step 1 until n do for j := i step 1 until n do
      begin  h[k] := h[k] + sigma[i] × sigma[j]/sg − s[i] × s[j]/ghg;
        k := k + 1
      end updating of h;
      if count > limit then go to exit;
      end of loop controlled by count;  go to finish;
    exit: conv := false;
    finish: end of FLEPOMIN
```

CERTIFICATION OF ALGORITHM 139 [A1] SOLUTIONS OF THE DIOPHANTINE EQUATION

[J.E.L. Peck, *Comm. ACM 5* (Nov. 1962), 556]

HENRY J. BOWLDEN (Recd. 30 Sept. 1964 and 5 Nov. 1964)
Westinghouse Electric Corp., R&D Ctr., Pittsburgh, Pa.

Algorithm 139 was transcribed into Burroughs Extended ALGOL after the following typographical error was corrected: On the line following "**if** $d \neq 1$ **then**" replace "$a := a/d$," by "$a := a/d$;".

The cases shown in the table were tried, with the results shown in columns 4 and 5. These solutions are correct, but perhaps not too useful. Of course, a definition of "useful" in this context would be rather subjective; in any case, the user can always obtain an arbitrary solution "useful" for his purpose. We have chosen to regard a small value of x as a criterion for usefulness, and obtain this by inserting, just before "*print* (x0, y0)", the statements

```
c := x0 ÷ b;  x0 := x0 − c × b;  y0 := y0 + c × a;
```

The following remarks have to do with matters of programming taste rather than accuracy.

(a) A **value** part of form **value** a, b, c; should be inserted to avoid side effects.

(b) The results should be passed back to the calling program for use by the caller. This requires the addition of two call-by-name parameters ($x0$, $y0$), and the removal of the declaration **integer** $x0$, $y0$;. The provisions for printing the results should be omitted.

(c) The procedure contains a deliberate possibility of an infinite loop. This is unacceptable on most operating systems and should be omitted.

(d) The provision of an array (q) "as large as storage will allow" is rather indefinite. The algorithm as given provides no test to prevent exceeding this arbitrary size. The number of partial quotients in the Euclidean algorithm may be shown to be no more than five times the number of decimal digits in the (largest of the) coefficients a, b, c, so a size of five times the number of digits in the largest integer to be considered is sufficient.

The algorithm, modified as suggested above, gives the results in columns 6 and 7 of the table below. The execution time on the B-5000 was approximately 40 milliseconds.

			original		*modified*	
a	b	c	$x0$	$y0$	$x0$	$y0$
1000	23	1046	−2092	91002	−22	1002
0	0	0	indeterminate			
57	−103	47009	2209423	1222234	73	−416
10	12	578	−289	289	−1	49
10	12	97	no solution			

REMARK ON ALGORITHM 145 [D1]
ADAPTIVE NUMERICAL INTEGRATION BY SIMPSON'S RULE [William Marshall McKeeman, *Comm. ACM 6*, (Dec. 1962), 604]

M. C. Pike (Recd. 5 Oct. 1964 and 23 Nov. 1964)
Statistical Research Unit of the British Medical Research Council, University College Hospital Medical School, London, United Kingdom

This procedure was tested on the ICT Atlas computer and found satisfactory after the following three modifications were made:

(1) add "**real** *absarea*;" on the line following "**integer** *level*;",
(2) add "*absarea* := 1.0;" on the line following "*level* := 1;",
(3) substitute

"*Integral* := *Simpson* (F, a, $b-a$, $F(a)$, $4.0 \times F((a+b)/2.0)$, $F(b)$, *absarea*, 1.0, *eps*)"

for

"*Integral* := *Simpson* (F, $b-a$, $F(a)$, $4.0 \times F((a+b)/2.0)$, $F(b)$, 1.0, 1.0, *eps*)".

These corrections are necessary since *absarea* appears on the left-hand side of an assignment statement, namely, in line 10 of the **real procedure** *Simpson*, and yet when *Simpson* is called in the third to last line of the **real procedure** *Integral* the actual parameter for *absarea* is given as 1.0.

The author wishes to thank the referee for helpful suggestions.

CERTIFICATION OF ALGORITHM 203 [E4]
STEEP1 [E. J. Wasscher, *Comm. ACM 6* (Sept. 1963), 517; *Comm. ACM 7* (Oct. 1964), 585]

J. M. Varah (Recd. 30 July 1964)
Computation Center, Stanford University, Stanford, Calif.

Algorithm 203 was run on the B5000 at Stanford with the necessary modifications for Burroughs' Extended Algol. After some testing, the following errors were found.

1. There is an extra **begin** in procedure *ATIVE*. The first statement after the comment in this procedure should be changed from

begin *ATIVE*: *lambda* := 0;

to

lambda := 0;

[It was the author's original intention that this **begin** be not in bold-face but that it should be part of the label *begin ATIVE* inserted to clarify the program.—Ed.]

Also, there is a missing semicolon in procedure *ATIVE* at the end of the line preceding *comp*: and procedure *STEP* has an unnecessary **begin-end** block.

2. Because the domain of definition of the function *FUNK* is bounded by the rectangular hyperbox $lb[j] \leqq x[j] \leqq ub[j]$, $j = 1, 2, \cdots, n$, before giving a new direction in which to proceed, the value of *xmin* is checked (in *ATIVE*, under *large*:). If, for any j, *xmin*[j] is within *dx*[j] of the boundary, *xmin*[j] is changed so that it is exactly *dx*[j] from the boundary. However, if the minimum value of *FUNK* occurs at just such a place (say right at the boundary), then a step will be made from this new position back to the boundary. Then the new *xmin*[j] will again be within *dx*[j] of the boundary, so it is moved away, and so on forming a loop. To correct this, the old value of *xmin*[j] should be saved (in *xstep*[j], for example) and below, when A is tested, the function value set equal to the minimum of values at *xmin* and *xstep*. The author, when A was true (i.e. when such a shift had been made), merely set the function equal to the value at *xmin*.

Specifically, this means changing the lines following *large*: to

```
A := B := false; if xmin[j] + dx[j] > ub[j] then
begin
  xstep[j] := xmin[j];
  xmin[j] := ub[j] - dx[j];  A := true
end
else if xmin[j] - dx[j] < lb[j] then
begin
  xstep[j] := xmin[j];
  xmin[j] := lb[j] + dx[j];  A := true
end;
```

and the conditional statement involving A (3rd line after *small*:) to

```
if A then
begin
  gamma := FUNK(xmin);
  if fmin ≦ gamma then xmin[j] := xstep[j]
  else fmin := gamma
end;
```

3. Also in *ATIVE*, under *comp*:, the derivative approximations are all normalized after the **for** loop by division by *lambda*. However, *lambda* will be zero if all *dfdx*[j] are zero to working accuracy. So we should only divide by *lambda* when it is not zero.

Specifically, this means inserting the line

if *lambda* $\neq 0$ **then**

before the third line from the end of procedure *ATIVE*.

With these corrections, the algorithm did run successfully. It should also be mentioned that procedures *ATIVE* and *STEP* could just as well be blocks with labels *ATIVE* and *STEP* rather than procedures, with the calls on them changed to **go to** *ATIVE* and **go to** *STEP*.

REMARK ON ALGORITHM 205 [E4]
ATIVE [J. G. A. Haubrich, *Comm. ACM 6* (Sept. 1963), 519]

E. J. Wasscher (Recd. 23 Nov. 1964)
Philips Computer Center, N. V. Philips' Gloeilampenfabrieken, Eindhoven, Netherlands

There is a misprint in this Algorithm. The first statement in the fifth line from the end of the procedure *ATIVE* should read:

$dx[j] := 3 \times dx[j]$;