```
      begin for i := 1 step 1 until n do x[i] := x[i] + lambda ×
        s[i]; f := fb
      end else
      begin gb := dot(g,s);
        if gb < 0 ∧ count > n ∧ step <10-6 then go to exit;
        fb := f; ita := ita - lambda;
        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 ∧ sqrt(dot(sigma,sigma)) <eps
      then go to finish
    end;
    for i := 1 step 1 until n do s[i] := up dot (h,gamma,i);
    ghg := dot(s, gamma);
    k := 1;
    if sg = 0 ∨ ghg = 0 then go to test;
    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;
test: if count > limit then go to exit;
  end of loop controlled by count; go to finish;
exit:conv := false;
finish:
end of FLEPOMIN
```

With these changes the procedure was run successfully on a KDF 9 computer on the first of the test functions used by Fletcher and Powell, and the appropriate rate of convergence was achieved. (The corresponding values in [1, Table 1, col. 4] being 24.200, 3.507, 2.466, 1.223, 0.043, 0.008, 4×10^{-5}). It could well be, however, that these changes may still not prove satisfactory on some functions. In such cases it will most likely be the search for the linear minimum along s which will be at fault, and not the method of generating s. It should not be necessary to evaluate the function and gradient more than 5 or 6 times per iteration in order to estimate the minimum along s, except possibly at the first few iterations.

I am indebted to William N. Nawatani of Dynalectron Corporation, Calif., for pointing out the discrepancies in the rates of convergence, and to the referee for his calculations and comments with regard to the Hilbert Matrix function.

REFERENCE

1. Fletcher, R., and Powell M. J. D. A rapidly convergent descent method for minimization. *Comput. J. 6* (July 1963), 163.

REMARK ON ALGORITHM 256 [C2]
MODIFIED GRAEFFE METHOD [A. A. Grau, *Comm. ACM 8* (June 1965), 379]

G. Stern (Recd. 8 Mar. 1965 and 24 Mar. 1965)
University of Bristol Computer Unit, Bristol 8, England

This procedure was tested on an Elliott 503 using the two simplifications noted in the comments on page 380. When the 16th line from the bottom of page 380, first column, was changed to read

$$h1 := aa \uparrow (1/(k-s+1));$$

(as suggested in a private communication from the author) correct results were obtained.

REMARK ON ALGORITHM 266 [G5]
PSEUDO-RANDOM NUMBERS [M. C. Pike and I. D. Hill, *Comm. ACM 8* (Oct. 1965), 605]

L. Hansson (Recd. 25 Jan. 1966)
DAEC, Riso, Denmark

As stated in Algorithm 266, that algorithm assumes that integer arithmetic up to $3125 \times 67108863 = 209715196875$ is available. Since this is frequently not the case, the same algorithm with the constants 125 and 2796203 may be useful. In this case the procedure should read

```
real procedure random (a, b, y);
  real a, b; integer y;
begin
  y = 125 × y; y := y - (y÷2796203) × 2796203;
  random := y/2796203 × (b-a) + a
end
```

The necessary available integer arithmetic is $125 \times 2796203 = 348525375 < 2 \uparrow 29$. With this procedure body, any start value within the limits 1 to 2796202 inclusive will do.

Seven typical runs of the poker-test gave the results:

start value	*all different*	*1 pair*	*2 pairs*	*3*	*3 + pair*	*4*	*5*
100001	129	199	39	31	2	0	0
1082857	115	206	45	31	2	1	0
724768	120	195	49	32	3	1	0
78363	130	198	36	31	5	0	0
1074985	127	189	44	34	4	2	0
2567517	124	193	50	28	3	2	0
2245723	119	202	49	24	4	1	1
Totals for 7 runs:							
	864	1382	312	211	23	7	1
Totals for 100 consecutive runs with first start value 100001:							
	12023	20297	4301	2837	358	181	3

REMARK ON ALGORITHM 266 [G5]
PSEUDO-RANDOM NUMBERS [M. C. Pike and I. D. Hill, *Comm. ACM 8* (Oct. 1965), 605]

M. C. Pike and I. D. Hill (Recd. 9 Sept. 1965)
Medical Research Council, London, England

Algorithm 266 assumes that integer arithmetic up to $3125 \times 67108863 = 209715196875$ is available, which is not so on many computers. The difficulty arises in the statements

```
y := 3125 × y;  y := y - (y÷67108864) × 67108864;
```

They may be replaced by

```
integer k;
for k := ⟨for list⟩ do
begin
  y := k × y;
  y := y - (y÷67108864) × 67108864
end;
```

where the ⟨for list⟩ may be

125, 25 (requiring integer arithmetic up to less than 2^{33})
25, 25, 5 (requiring integer arithmetic up to less than 2^{31})
or
5, 5, 5, 5, 5 (requiring integer arithmetic up to less than 2^{29})

according to the maximum integer allowable. The first is appropriate for the ICT Atlas. [And also for the IBM 7090, the second for the IBM System/360 . . . Ref.]

Note. There are frequently machine-dependent instructions available which will give the same values as the above statements much more quickly, if speed is of much importance.