Function minimization by conjugate gradients

By R. Fletcher and C. M. Reeves*

A quadratically convergent gradient method for locating an unconstrained local minimum of a function of several variables is described. Particular advantages are its simplicity and its modest demands on storage, space for only three vectors being required. An ALGOL procedure is presented, and the paper includes a discussion of results obtained by its use on various test functions.

The problem of locating an unconstrained local minimum of a function of several variables is recognized as important both in its own right and as a means of solving sets of simultaneous non-linear algebraic equations. Our concern is with functions defined numerically. In particular we consider a function of n variables whose value $f(x)$ and gradient vector $g(x)$ can be calculated at any point x. We assume that in a neighbourhood of the required minimum h, the function may be expanded in the form

$$f(x) = f(h) + \tfrac{1}{2}(x - h)'A(x - h) + \text{higher terms} \quad (1)$$

where A, the matrix of second-order partial derivatives, is symmetric and positive definite.

Particularly attractive are iterative methods having quadratic convergence, meaning that for quadratic functions it is guaranteed that the minimum will be located exactly, apart from rounding errors, within some finite number of iterations, usually n. For general functions, as the iterate approaches the minimum, the function is more closely quadratic and so convergence is more nearly assured. Furthermore, even in regions remote from the minimum, such methods, by taking account of the curvature of the function, are best able to deal with complex situations such as the presence of a long curving valley. The oscillatory behaviour characteristic of methods such as steepest descents is thereby avoided.

Virtually all iterative minimization procedures, whether quadratically convergent or not, locate h as the limit of a sequence $x_0, x_1, x_2, \ldots$ where x_0 is an initial approximation to the position of the minimum, and where for each $i \geqslant 0$, x_{i+1} is the position of the minimum of $f(x)$ with respect to variations along the line through x_i in some specified direction p_i. Thus, for example, the method of steepest descents uses the direction of the negative gradient of $f(x)$ at x_i, and the method of alternating directions uses cyclically the directions of the n coordinate axes. Methods which calculate each new direction of search as part of the iteration cycle are inherently more powerful than those in which the directions are assigned in advance, in that any accumulated knowledge of the local behaviour of the function can be taken into account. Setting $g(x_i) = g_i$ for each i, the step from x_i to x_{i+1} is determined by the relation

$$g_{i+1}'p_i = 0 \quad (2)$$

where
$$x_{i+1} = x_i + \alpha_i p_i \quad (3)$$

for some scalar parameter α_i.

Methods using conjugate directions

Let us consider the minimization by successive linear searches of the quadratic function

$$f(x) = f(h) + \tfrac{1}{2}(x - h)'A(x - h) \quad (4)$$

for which the gradient is

$$g(x) = A(x - h). \quad (5)$$

By repeated use of equation (3), we have

$$x_n = x_{j+1} + \sum_{i=j+1}^{n-1} \alpha_i p_i \quad (6)$$

for any j in $0 \leqslant j \leqslant n - 1$. It then follows from equation (5) that

$$g_n = g_{j+1} + \sum_{i=j+1}^{n-1} \alpha_i A p_i \quad (7)$$

and therefore, using equation (2), that

$$g_n'p_j = \sum_{i=j+1}^{n-1} \alpha_i p_i' A p_j. \quad (8)$$

Now if the vectors $p_0, p_1, p_2, \ldots p_{n-1}$ are A-conjugate, satisfying

$$p_i'Ap_j = 0 \quad \text{for} \quad i \neq j, \quad (9)$$

then
$$g_n'p_j = 0 \quad (10)$$

and therefore, since $p_0, p_1, p_2, \ldots p_{n-1}$ form a basis,

$$g_n = 0 \quad (11)$$

whence, by equation (5)

$$x_n = h. \quad (12)$$

This demonstrates that the method of successive linear searches is quadratically convergent when using any set of A-conjugate directions. The minimum is located at the nth iteration, or earlier if in any particular case the later values of α_i should be zero.

* *Electronic Computing Laboratory, The University, Leeds* 2.

Now we have stipulated that $f(x)$ and its gradient are defined numerically so that A is not explicitly available. This naturally complicates the generation of a set of A-conjugate directions. Various proposals have been made. Shah, Buehler, and Kempthorne (1961) and Powell (1962) make use of the geometric properties of a quadratic surface. Similarly, Smith (1962) has reported a method which only requires function values, and not the gradients, to be calculated. This method and modifications of it seem to be the best of the currently available non-gradient procedures.

Probably the most powerful of the gradient methods is Fletcher and Powell's (1963) reformulation of a procedure originated by Davidon (1959). In this the A-conjugate directions p_i are given by

$$p_i = - H_i g_i \tag{13}$$

where $H_0, H_1, H_2, \ldots$ is a sequence of symmetric positive definite matrices. H_0 is arbitrary but is usually taken to be the unit matrix. Subsequent members of the sequence are generated by

$$H_{i+1} = H_i + \frac{p_i p_i'}{p_i' A p_i} - \frac{H_i \gamma_i \gamma_i' H_i}{\gamma_i' H_i \gamma_i} \tag{14}$$

where $$\gamma_i = g_{i+1} - g_i. \tag{15}$$

It is shown that, as x_i reaches h, so H_i becomes A^{-1}. Thus the method yields full information on the curvature of the function $f(x)$ at its minimum. This information is, however, obtained at the price of providing storage space for the matrix H, and time for its manipulation. In many applications a method which is more economical in operation and which merely locates the minimum may be preferred.

The method of conjugate gradients

The method of conjugate gradients (Hestenes and Stiefel, 1952) is an elegant n-step procedure for solving a set of simultaneous linear equations having a symmetric positive definite matrix of coefficients. The equivalence of that problem and the minimization of a quadrative function $f(x)$ is clear from equations (4) and (5). The condition for the gradient to vanish is seen to be

$$Ax = b \tag{16}$$

where $$b = Ah. \tag{17}$$

In the solution of these equations, directions $p_0, p_1, \ldots$ are generated such that p_{i+1} is a linear combination of $-g_{i+1}$ and $p_0, p_1, \ldots p_i$ such that the A-orthogonality conditions (9) are satisfied. A full and lucid description has been given by Beckman (1960). In the event many of the coefficients are zero and the following simple form results.

$$p_{i+1} = - g_{i+1} + \beta_i p_i \tag{18}$$

where $$\beta_i = \frac{g_{i+1}^2}{g_i^2}. \tag{19}$$

This leads to the following general minimization algorithm.

$$\left.\begin{aligned} x_0 &= \text{arbitrary} \\ g_0 &= g(x_0),\ p_0 = -g_0 \\ x_{i+1} &= \text{position of minimum of } f(x) \text{ on the} \\ &\quad\ \text{line through } x_i \text{ in the direction } p_i \\ g_{i+1} &= g(x_{i+1}) \\ \beta_i &= g_{i+1}^2/g_i^2 \\ p_{i+1} &= -g_{i+1} + \beta_i p_i. \end{aligned}\right\} \tag{20}$$

This process is guaranteed, apart from rounding errors, to locate the minimum of any quadratic function of n arguments in at most n iterations. For functions which are not quadratic the process is iterative rather than n-step, and a test for convergence is required. The directions p_i that are generated are those corresponding to the current local quadratic approximation to the function, and the rate of convergence depends upon the response to changes in the local quadratic approximation from iteration to iteration. Thus in applying the process (20) to general functions, four main points require attention. These are the choice of x_0, the linear search to locate each x_{i+1}, the overall rate of convergence, and the final convergence criterion.

The choice of x_0

For quadratic functions any choice of starting point is in principle equally satisfactory. For general functions the best that can reasonably be expected is that the minimization process will lead as quickly as possible to the bottom of whatever valley it starts in. In some applications it is possible to detect when convergence to an unwanted minimum has occurred, and some form of extrication process is then desirable. Here we merely note the importance of the choice of starting point.

The linear search

In any practical application the time spent evaluating the function and gradient at the various points required may well dominate the time for the whole minimization process. It is therefore desirable to limit the number of such evaluations as much as possible.

Essentially the linear search problem is to determine t_m such that

$$y'(t_m) = 0 \tag{21}$$

where $$y(t) = f(x_i + tp_i) \tag{22}$$

and therefore

$$y'(t) = p_i' g(x_i + tp_i). \tag{23}$$

Thus $y(t)$ and $y'(t)$ are calculable for any t and, in particular, $y(0) = f_i$ and $y'(0) = p_i' g_i \leqslant 0$ are already available.

The method adopted is that proposed by Davidon and used also by Fletcher and Powell. The calculation is in three stages; the first estimates the order of magnitude of t_m, the second establishes bounds on it, and the third interpolates its value.

In the Davidon method it is shown that as H_i approaches A^{-1}, so t_m approaches unity, thus providing an inherent scale in which to work. There is no comparable feature of the present method, and so we have arbitrarily chosen a unit for t which corresponds to a displacement along p_i of unit length in the x-space. We supplement this with the requirement that there be available an estimate, est, of the value of $f(x)$ at the unconstrained minimum. On the suppositions that this is a correct estimate, that the unconstrained minimum lies on the line $x_i + tp_i$, and that $f(x)$ is quadratic, we would obtain the value k for t_m where

$$k = 2(\text{est} - f_i)/p_i'g_i. \tag{24}$$

In fact the unconstrained minimum will generally not lie on the line, and so equation (24) will tend to overestimate t_m. We have therefore followed Fletcher and Powell in taking as a tentative step length

$$\begin{aligned} h &= k \quad \text{if } 0 < k < (p_i^2)^{-1/2}, \\ &= (p_i^2)^{-1/2} \text{ otherwise.} \end{aligned} \tag{25}$$

In the second stage y' is examined at the points $t = 0, h, 2h, 4h, \ldots, a, b$, where t is doubled each time, and where b is the first of these values at which either y' is non-negative or y has not decreased. It then follows that t_m is bounded in the interval $a < t_m \leqslant b$.

The third stage uses the cubic interpolation given by Davidon. Defining

$$z = 3\frac{y(a) - y(b)}{b - a} + y'(a) + y'(b) \tag{26}$$

and

$$w = (z^2 - y'(a)y'(b))^{1/2} \tag{27}$$

the estimate t_e of t_m is

$$t_e = b - \left(\frac{y'(b) + w - z}{y'(b) - y'(a) + 2w}\right)(b - a). \tag{28}$$

If neither $y(a)$ nor $y(b)$ is less than $y(t_e)$ then t_e is accepted as the estimate of t_m. Otherwise, according as $y'(t_e)$ is positive or negative, the interpolation is repeated over the sub-interval (a, t_e) or (t_e, b), respectively. A single application of the interpolation formula produces the exact value for t_m in the limit as $f(x)$ becomes quadratic in the neighbourhood of a local minimum. Increased accuracy in the general case is obtained only at the cost of further evaluations of $f(x)$ and its gradient. As the only region where the interpolation is likely to be inaccurate is that remote from the minimum, it is uneconomic to require high accuracy in this region. Numerical tests have shown that no significant reduction in the number of iterations can be achieved by using higher accuracy interpolation. From the point of view of stability however, what must be done is to ensure that the values $f(x_i)$ do form a decreasing sequence. Hence the provision for repeating the interpolation over smaller intervals.

The rate of convergence

Early experience in using the process on Rosenbrock's banana-shaped valley function (Table 1) led to successive directions p_i being so nearly parallel that the points x_i were scarcely separated. This made for very slow convergence. It was also noted that the path of x swung wide on the bend, raising fears that, had the calculation been allowed to continue, the rate of convergence might have been reduced further by a spiral approach to the minimum. A modification of the basic process was sought which would overcome such ill-effects with anharmonic functions, and which would nevertheless retain the quadratic convergence of the process when applied to harmonic functions. The solution adopted was to revert periodically to the steepest descent direction $-g$ in place of the customary p. Thus the whole process is restarted from the current x, discarding all previous experience, whether useful or erroneous, that would normally be transmitted in the calculation of p. The process remains quadratically convergent provided that such restarts are not more frequent than every n iterations. In practice we have modified the process in this way every $(n + 1)$ iterations with satisfactory results. This period was suggested by analogy with the use of the conjugate gradient method for solving linear equations, where it is found that an additional iteration is beneficial in compensating for the accumulation of rounding errors in the first n iterations.

The convergence criterion

Clearly if any g_i^2 vanishes the iterations must stop, both to avoid division by zero in the next iteration and because this is the formal requirement for x_i to be at a minimum. This condition is unlikely to be realized in practice because of round-off errors. We therefore continue the iterations until a complete cycle of $(n + 1)$ iterations, starting from a steepest descent search, produces no reduction in the value of the function. The reader is invited to consider as an Awful Warning against the uncritical acceptance of this criterion, the effect of applying it in the minimization of the function $f(x) = \frac{1}{2}(x_1^2 + 4x_2^2)$ using floating-point arithmetic.

In particular cases it may well be that some less stringent criterion would be appropriate. Thus it might be sufficient to continue the iterations until the change in each argument throughout a cycle of $(n + 1)$ iterations was less than some assigned value. Such requirements could be implemented in the ALGOL procedure that follows by suitable definition of the procedure MONITOR referred to therein.

The ALGOL procedure

procedure *FUMICOG* (*n*, *x*, *f*, *est*, *FUNCT*, *MONITOR*,
    CONVERGED);
value *n*, *est*;
real *f*, *est*; **integer** *n*; **Boolean** *CONVERGED*;
real array *x*; **procedure** *FUNCT*, *MONITOR*;

```
comment FUMICOG (FUnction MInimization by
  COnjugate Gradients) is a quadratically convergent pro-
  cess for locating an unconstrained local minimum of a
  function of n arguments. At entry x[1 : n] is an initial
  approximation to the position of the required minimum
  and est is an estimate, preferably but not essentially low
  rather than high, of the corresponding function value. At
  exit x and f are the location and value of the minimum.
  The procedure statement

              FUNCT (n, x, f, g)

  calculates the value and gradient vector at x[1 : n] of the
  function to be minimized, and assigns them to f and
  g[1 : n]. The procedure MONITOR is activated once in
  every iteration by means of the procedure statement

       MONITOR (n, x, f, g, p, gg, count, i, EXIT)

  where gg is the square of the length of g, count = 1, 2, . . .
  is the index of the current cycle of (n + 1) iterations, and
  i = 0, 1, 2, . . ., n is the iteration index within the current
  cycle. On leaving FUMICOG normally, CONVERGED
  is true, but if required MONITOR can break off the
  iterations by sending control to the label EXIT and then
  CONVERGED will be set false. A discussion of the uses
  of procedures such as MONITOR has been given by
  Rutishauser (1961). ;

begin real gg, old f, old gg, beta; integer count, i, r;
      real array g, p[1 : n];

      real procedure dot (a, b);
      real array a, b;
      comment dot is the scalar product of the vectors
        a and b;
      begin real s; integer j; s := 0;
            for j := 1 step 1 until n do s := s+a[j]×b[j];
            dot := s
      end of dot;

Start: CONVERGED := true;
       FUNCT (n, x, f, g);
       for count := 1, count + 1 while f < old f do
       begin old f := f;
       for i := 0 step 1 until n do
       begin gg := dot (g, g); if gg = 0 then go to finish;
       if i = 0 then begin for r := 1 step 1 until n do
                           p[r] := −g[r] end
                 else begin beta := gg / old gg;
                      for r := 1 step 1 until n do
                          p[r] := −g[r] + beta × p[r]
                      end;
       MONITOR (n, x, f, g, p, gg, count, i, EXIT);

linear search:
       begin real ya, va, yb, vb, vc, pp, h, k, w, z, t;
             yb := f; vb := dot (g, p); pp := dot (p, p);
             if vb ⩾ 0 then go to skip;
             k := 2 × (est − f) / vb;
     scale: h := if k > 0 and k ↑ 2 × pp < 1 then k
                 else 1/sqrt(pp);
             k := 0;
 extrapolate: ya := yb; va := vb;
             for r := 1 step 1 until n do
                 x[r] := x[r] + h × p[r];
             FUNCT (n, x, f, g);
             yb := f; vb := dot (g, p) ;
             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 r := 1 step 1 until n do
                          x[r] := x[r] + (t − k) × p[r];
             FUNCT (n, x, f, g);
             if f > ya or f > yb then
               begin vc := dot (g, p);
                     if vc < 0 then begin ya := f;
                                          va := vc;
                                          h := k;
                                          t := h end
                               else begin yb := f;
                                          vb := vc;
                                          h := h − k;
                                          t := 0 end;
                     go to interpolate
               end;
      skip: end of linear search;

             old gg := gg;
             end of inner loop controlled by i;
             end of outer loop controlled by count;
             go to finish;
EXIT: CONVERGED := false;
finish: end of FUMICOG;
```

Numerical results and conclusions

Limited trials of the procedure have been carried out by means of the ALGOL compilers for the Pegasus and KDF9 computers, the latter through the generosity of I.C.I. Ltd.

Table 1 gives results for Rosenbrock's banana-shaped valley (see Rosenbrock, 1960) in two dimensions,

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

starting from the point $(x_1, x_2) = (-1\cdot 2, 1\cdot 0)$. Column A corresponds to the basic iteration without the restart procedure. Column B shows the effect of continuing the linear search iterations to higher accuracy, namely until

$$\frac{(g'p)^2}{g^2p^2} < 10^{-6}.$$

This criterion requires the angle between g_{i+1} and p_i to

Table 1

Results for banana-shaped valley

ITERATION	A		B		C		
	x_1	x_2	x_1	x_2	x_1	x_2	f
0	−1·200	1·000	−1·200	1·000	−1·200	1·000	24·200
3	−0·631	0·324	−0·784	0·556	−0·631	0·324	3·199
6	−0·460	0·119	−0·520	0·182	−0·425	0·124	2·353
9	−0·299	−0·017	−0·348	0·018	−0·171	−0·045	1·921
12	−0·172	−0·086	−0·215	−0·067	0·139	−0·023	0·920
15	−0·063	−0·119	−0·102	−0·109	0·510	0·214	0·453
18	0·035	−0·127	−0·000	−0·124	0·681	0·433	0·193
21	0·130	−0·116	0·096	−0·119	0·846	0·698	0·053
24	0·225	−0·085	0·193	−0·094	0·989	0·980	8×10^{-4}
27	0·325	−0·033	0·294	−0·047	1·000	1·000	1×10^{-8}
30	0·434	0·048	0·403	0·028			

differ from the theoretical 90° by not more than $\sin^{-1}(10^{-3}) \approx 0{\cdot}06°$. It is seen that the slow convergence persists. Column C corresponds to the form of the procedure given in the previous Section, incorporating restarts. The function value was reduced to 1×10^{-8} in 27 iterations. This compares with the 18 iterations quoted by Fletcher and Powell for the Davidon method.

Table 2 gives results for Fletcher and Powell's helical valley in three dimensions,

$$f(x_1, x_2, x_3) = 100[(x_3 - 10\theta)^2 + (r - 1)^2] + x_3^2$$

where $x_1 = r\cos 2\pi\theta$ and $x_2 = r\sin 2\pi\theta$. The starting point was $(x_1, x_2, x_3) = (-1, 0, 0)$ and only the region $-\frac{1}{4} < \theta < \frac{3}{4}$ was considered. The function value was reduced to 6×10^{-9} in 36 iterations, which compares with Fletcher and Powell's value of 7×10^{-8} after 18 iterations.

The Davidon method is evidently superior in terms of the number of iterations for convergence. However, the Davidon iteration is much more complicated, and a comparison of running times will depend critically both upon n, the number of arguments, and upon the time required for each evaluation of the function and its gradient. If this time is comparable with that required for calculating a new direction of search, the simplicity of the present method gives it an advantage. Furthermore, the present method requires storage for only three vectors, and so in problems where n is large may be preferred to the Davidon method which requires space for the matrix H.

Table 2

Results for helical valley

ITERATION	x_1	x_2	x_3	f
0	−1·000	0·000	0·000	$2{\cdot}500\ 10^3$
4	−0·091	0·932	2·696	7·831
8	0·497	0·942	1·736	3·444
12	0·577	0·861	1·595	2·797
16	0·833	0·664	1·040	1·600
20	0·962	0·428	0·696	0·858
24	1·005	0·218	0·311	0·261
28	1·008	0·075	0·112	0·030
32	1·001	0·004	0·008	3×10^{-4}
36	1·000	-10^{-5}	-10^{-5}	6×10^{-9}

References

Beckman, F. S. (1960). "The solution of linear equations by the conjugate gradient method" in *Mathematical Methods for Digital Computers*, Ralston, A., and Wilf, H. S. (Eds.), Wiley.

Davidon, W. C. (1959). "Variable metric method for minimisation," A.E.C. Research and Development Report ANL-5990 (Rev.).

Fletcher, R., and Powell, M. J. D. (1963). "A rapidly convergent descent method for minimization," *The Computer Journal*, Vol. 6, p. 163.

Hestenes, M. R., and Stiefel, E. (1952). "Methods of conjugate gradients for solving linear systems," *J. Res. N.B.S.*, Vol. 49, p. 409.

Powell, M. J. D. (1962). "An iterative method for finding stationary values of a function of several variables," *The Computer Journal*, Vol. 5, p. 147.

Rosenbrock, H. H. (1960). "An automatic method for finding the greatest or the least value of a function," *The Computer Journal*, Vol. 3, p. 175.

RUTISHAUSER, H. (1961). "Interference with an ALGOL procedure" in *Annual Review in Automatic Programming*, Vol. 2, Goodman, R. (Ed.), Pergamon Press.

SHAH, B. V., BUEHLER, R. J., and KEMPTHORNE, O. (1961). "The method of parallel tangents (Partan) for finding an optimum," Office of Naval Research Report, NR-042-207 (No. 2).

SMITH, C. S. (1962). "The automatic computation of maximum likelihood estimates," N.C.B. Scientific Department Report, S.C. 846/MR/40.

Book review: ALGOL on the KDF9

ALGOL 60 *Implementation*, by B. RANDELL and L. J. RUSSELL, 1964; 418 pages. (London: *Academic Press Inc*., 84s.)

The authors' intention in writing this book is to present a full description of their implementation of ALGOL 60 on the English Electric KDF9 computer. This aim has been most admirably fulfilled, both in the general description of the methods they have used, and in the detailed flow charts from which their programs were coded.

The general technique of implementation was based on the work of E. W. Dijkstra and J. A. Zonneveld, who wrote the first ALGOL 60 translator for the X1 computer at the Mathematical Centre, Amsterdam. The translator is built up of a number of routines, each of which processes one of the delimiters of the language. Each routine ends with a transfer of control to the basic input routine, which reads in and assembles the source text as far as the next delimiter, and passes control to the corresponding delimiter routine to process it. Many of the delimiter routines make use of a global stack for the storage of information which will be needed later by another routine. The stack mechanism is admirably suited for dealing with recursively structured languages such as ALGOL, in which expressions, and even statements, may be bracketed one inside the other to any depth. However, the method chosen for specifying the use of a stack seems rather clumsy, since all operations of pushing information down on the stack and restoring it again when required have been inserted as explicit instructions in the flow charts. A more elegant way of using a stack is to write the translator as a set of procedures which can call each other and even themselves in a recursive manner; in this case, the whole of the stack administration is incorporated behind the scenes in the procedure entry and exit mechanism. In a recursively organized translator, each procedure can be designed to process the whole of an ALGOL syntactic entity, rather than a single delimiter. This makes it possible to abolish many of the markers which otherwise have to be set, stacked, unstacked and tested in order to establish context in the source program. Since the ALGOL language recognizes the usefulness of recursion, it seems a pity that an ALGOL translator should deny itself the benefits which it makes available to others.

The implementation of an advanced programming language involves a great deal more than translating it, since a considerable amount of book-keeping must remain to be done at run-time; and the specification of control routines to perform this task is a major part of the implementation. The division of labour between the translator and the control routines is one of the most characteristic features of any system. The authors have chosen to simplify the task of the translator as much as possible, and to place a correspondingly heavy burden on the control routines. In fact, the object program produced by the translator is not even framed in the KDF9 machine code at all, but in a sort of idealized machine code, specially suited to the needs of ALGOL; and the control routines have the job of interpreting this code at run-time in order to execute the program. The main justification for this use of interpretation is that the system is designed for use in program testing. It is therefore most important that the translation process should be as fast as possible, since the programs are likely to be altered every time that they are run. In addition, interpretation makes it possible to include some extremely powerful facilities for diagnostic printout at run-time.

As far as the reader of the book is concerned, the use of the idealized machine code is of great benefit, in that the description is almost entirely computer independent, and in no way involves the particular idiosyncrasies of the KDF9 machine code. This will be of particular interest to prospective implementors of ALGOL, who may wish to use the same flow charts on a different computer, as has already been done on three other machines, Deuce, Pegasus and ACE. However, as the authors point out, the use of interpretation involves a very severe penalty in efficiency at run-time, which is likely to be tolerated only during program checkout; and it is expected that a fully tested program will be retranslated by a more complex compiler into KDF9 machine code. In the absence of such an alternative compiler, the prospective implementor would be well advised to produce object programs in the machine code of the computer on which he is working.

The presentation of the idealized machine code will also be of great interest to prospective designers of future machine codes, for it points clearly the direction in which they must orient their design. It is becoming more obvious that the power of modern computers cannot be fully exploited without the use of advanced symbolic programming languages; yet at present, the use of these languages involves a considerable expense, either in a lengthy process of optimization, or in the inefficiency of object program. It is therefore a matter of urgent practical economics that computer designers should pay close attention to the needs of implementors and users of symbolic programming languages.

An even greater contribution is the clear and detailed manner in which the authors explain the nature of the problems they encountered, and the way in which they tackled them. The book should be read with the utmost interest by all programmers who are concerned with the development and use of automatic programming languages, and, in particular, those who have not themselves had the opportunity of implementing such a language. For the benefit of this class of reader, the book includes a brief but competent survey of other published techniques, and a comprehensive bibliography.

The authors must be highly praised for the delightful clarity of their English prose. The writing of the book has obviously been a pure labour of love, and the effort and care which has been expended on it at least equals that spent on the programs which it describes. In spite of the immense wealth of detail, the main thread of the description is always kept to the fore; and as an exercise in the documentation of a complex algorithm, a standard has been set that will not readily be equalled.

C. A. R. HOARE.