A preliminary investigation of function optimisation by a combination of methods

D. A. Phillips

P.O. Box 1608, Ndola, Zambia

Function optimisation by a method comprised of several standard optimisation techniques is considered. The combined method attempts to use the techniques to which it has access, in an order, the order being defined by the requirement of optimal computing time.

The optimisation of functions of two to nine variables is considered. Results are given for a simple combined program containing three standard techniques and these results are compared with those of the individual methods. The three methods used are a method by Rosenbrock and modified by Swann, a Simplex technique presented by Nelder and Mead and a method by Powell. The Simplex method has been adapted to deal efficiently with initial step sizes that are too large. The modification produces a drastic improvement in the results of the test cases which have this property.

The results show that this simple version of a combined program is generally slower than the individual methods. Suggestions are given for improvements which may allow the combined method to give better results.

(Received December 1971)

1. Introduction

Consider two optimisation techniques, method A and method B. Each method generally gives a different rate of convergence to the optimum for a certain function; and for different regions within the function. Suppose that **Fig. 1** gives the behaviour of the two optimisation techniques attempting to minimise the same function.

The final results e and k (Fig. 1) are not separated by a large amount and so it is reasonable to consider both methods as being equally fast in arriving to within the same accuracy of the optimum (e and k).

The assumption that both methods follow the same path of descent but at different rates is generally not true. However the assumption is made that the paths are relatively close. This assumption validates a change from either of the graphs to the other by simply changing methods. The final values of one method are the initial values of the other method.

Thus starting with method A from point a in Fig. 1 (as it converges faster than method B in this region) and using this method until, for the same values, method B would converge faster, one arrives at point f at time t_1. Method B is now faster and so this method is used from point c. Carrying out the single changeover as described, the curve in **Fig. 2** is obtained.

This changeover provides a reduction in computing time while still achieving the same accuracy.

To this computing time there must be added any time taken up in changing from method A to method B, and also time required to determine the changeover point itself. Changeover time (CT) is generally small compared with decision time (DT) the former occurring at the changeover points, the latter being distributed throughout the computation. **Fig. 3** would be a more realistic representation of the situation under consideration, DT being added at the end for convenience. The smaller it is possible to keep the time CT and DT, the better will be the method.

The likelihood of being able to use really fast convergence rates will increase as the number of optimisation methods made available increases.

This advantage, however, must be weighed against the increase in DT. Hence, a compromise is required between the number of methods and complexity of the decision processes. The ideal giving small DT, a fast solution, and a high rate of success in optimising a general function.

In general, results of optimisation methods are given as the

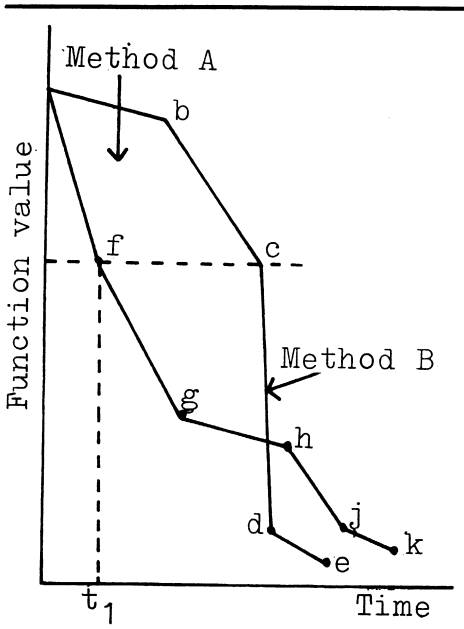


Fig. 1

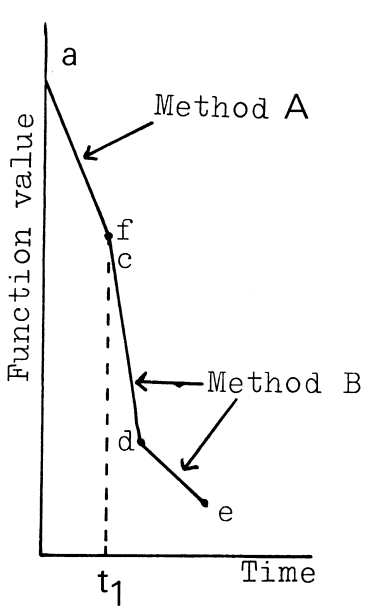


Fig. 2

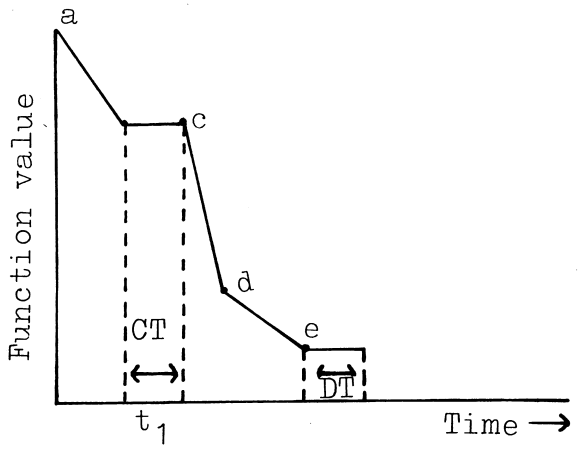


Fig. 3

function value (or the logarithm of the function value) at a certain number of function evaluations. A program with a fairly large function, requiring several hundred function evaluations, has its running time dominated by the time it takes to calculate this large number of function values. Consequently, function evaluations are almost equivalent to calculation time.

Further discussion about the equivalence of these factors is given in the Appendix.

2. Combined method

A combined method was designed which contained three standard optimisation techniques capable of optimising unconstrained analytic functions specified in procedure form.

The choice of three methods for a combined program was made from proven methods, each being basically different in its approach to the optimisation problem. Being basically different each method performs better than the others under certain circumstances. If this were not so then one or two methods would dominate the rest, whose inclusion would be pointless.

Consequently, if the function to be optimised is chosen at random, then each method has roughly the same *a priori* chance of being selected as the fastest.

The three methods chosen were

1. a method based on a quadratic convergence by Powell (1965),
2. a random search method by Nelder and Mead (1965) (Simplex method),
3. a rotating co-ordinate method, a modification by Swann (1964) of Rosenbrock's method (1960).

These methods will be referred to as Powell's, Simplex, and Rosenbrock's method throughout the text.

With the individual techniques in procedure form the combined program can call any one method and use it to optimise, for a given number of function evaluations and from a specified starting point, the function under investigation.

The control part of the combined program tests the velocity of convergence of each technique and selects the best procedure to carry out each stage of the optimisation.

Consider n individual methods M_i, $i = 1, \ldots, n$, with velocities v_i.

The velocity with which a particular method approaches the optimum is taken as the change in function value per unit time (or per unit function evaluation).

Initially the velocity for each procedure is required. This is calculated by optimising the function for a given test period t. Calculating the velocity of each procedure from the same starting point and for the same period t, would provide the best comparison. Interest lies in a fast program, however, so each procedure has its velocity calculated over consecutive test periods.

v_1 calculated in $[0, t]$
v_2 calculated in $[t, 2t]$
$\vdots$
v_n calculated in $[(n-1)t, nt]$

The final vector of one method provides the starting vector for the next. Now, providing t is small, each v_i can be regarded as the true v_i in $[0, nt]$. Hence the comparison of v_i, $i = 1, 2, \ldots, n$ is valid.

$$V_m = \frac{\max}{i} v_i,\ i = 1, 2, \ldots, n.\ 1 \leqslant m \leqslant n\,.$$

M_m is now used to optimise the function for a further length of time known as a run period. $[nt, nt + \lambda t]$.

V_m is re-calculated in $[(n-1)t + \lambda t, nt + \lambda t]$ and is now no longer necessarily a maximum.

V_i, $i = 1, 2, \ldots m-1, m+1 \ldots, n$, are now calculated over successive intervals

$[nt + \lambda t, (n+1)t + \lambda t], [(n+1)t + \lambda t, (n+2)t + \lambda t],$
$\ldots, [(2n-2)t + \lambda t, (2n-1)t + \lambda t]$

and a new V_m found.

The iteration continues until optimisation results are obtained within the required specification.

The combined method requires a procedure to evaluate the function at any point in an unconstrained region and an initial starting vector from which optimisation proceeds.

It is a peculiarity of the Simplex method that if the initial Simplex is too large, then the first function evaluations are wasted in slowly contracting the Simplex; with no improvement in the approximation to the optimum. After this initial contraction, when an acceptable Simplex has been found the method is generally efficient.

In the combined method, as the Simplex procedure is used for short periods of time t, this peculiarity often results in the procedure never improving the approximate optimum. This gives the test velocity of the Simplex procedure as zero. Even with a good step size initially, on re-entries (with the same initial step size) the new simplicities will eventually become and remain too large. Hence, the method has a limited usefulness for any one step size.

To overcome this difficulty the step size is reduced, using a multiplier of $-\frac{1}{2}$, whenever a Simplex is initially too large. The Simplex is then re-formed using the modified step size and re-tested for acceptability. This reduction continues until an acceptable size is obtained and this value replaces the initial step size for subsequent entries to this procedure.

It should be noted that this alteration does not affect the program once an acceptable Simplex is found or if the initial Simplex is satisfactory.

Basically, the modification means that the step size is contracted at a much faster rate than in the original method, and in the combined method the Simplex procedure is never left before some improvement is achieved and hence a zero velocity is never obtained.

3. Test functions

All the functions used have previously been employed in other papers concerned with optimisation. These functions were chosen for the characteristic difficulties that each presents to the optimisation technique.

Function 1

Rosenbrock's valley function (Powell, 1965)

$$\text{Funct}\,(x_1, x_2) = 100(x_2 - x^2{}_1)^2 + (1 - x_1)^2$$

starting point $(-1.2, 1)$, minimum at $(1, 1)$.

Function 2

Helical valley function (Fletcher, 1969)

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

Table 1 Results

FUNCTION	INDIVIDUAL METHODS						COMBINED METHOD	
	ROSENBROCK		SIMPLEX (*Unmodified)		POWELL			
	X	F	X	F	X	F	X	F
Rosenbrock	10^{-5}	140	10^{-5}	125	10^{-5}	138	10^{-4}	124
	10^{-10}	152			10^{-7}	145	10^{-6}	220
Helical valley	10^{-4}	160	10^{-4}	130	10^{-4}	140	10^{-3}	219
	10^{-10}	187	10^{-10}	199	10^{-6}	149		
Powell 4-D	10^{-3}	105	10^{-3}	73	10^{-3}	102	10^{-3}	155
	10^{-5}	154	10^{-5}	109	10^{-5}	138	10^{-5}	226
Chebychev 2-D	10^{-7}	43	10^{-7}	58	10^{-7}	22	0	34
	10^{-12}	103	10^{-12}	95	10^{-9}	25		
			$*10^{-5}$	*170				
Chebychev 4-D	10^{-5}	61	10^{-5}	73	10^{-5}	46	10^{-3}	25
	10^{-8}	121	10^{-8}	121	10^{-6}	52	10^{-4}	64
	10^{-11}	181	10^{-11}	164			10^{-5}	180
			$*10^{-6}$	*182				
Chebychev 6-D	10^{-5}	127	$*10^{-5}$	*228	10^{-5}	113	10^{-5}	220
	10^{-8}	235	$*10^{-8}$	*346			10^{-8}	330
	10^{-11}	415	$*10^{-10}$	*399				
			10^{-5}	138				
			10^{-8}	258				
			10^{-13}	382				
Chebychev 9-D	10^{-4}	273	10^{-3}	135	10^{-3}	109	10^{-3}	147
	10^{-6}	524	10^{-4}	350	10^{-4}	190	10^{-3}	305
			$*10^{-3}$	*157				

X = order of distance from optimum in terms of function value. F = number of function evaluations.

where

$$x_1 = r \cos 2\pi\theta, \; x_2 = r \sin 2\pi\theta$$

i.e.

$$2\pi\theta = \arctan(x_2/x_1), \; x > 0$$
$$= \pi + \arctan(x_2/x_1), \; x < 0$$

and

$$r = (x_1^2, +x_2^2)^{\frac{1}{2}}$$

starting point (−1, 0, 0,), minimum at (1, 0, 0,).

Function 3

Powell's function of four variables (Nelder and Mead, 1965)

$$\text{Funct}(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

starting point (3, −1, 0, 1) minimum at (0, 0, 0, 0).

Function 4

Chebyquad function (Fletcher, 1965).

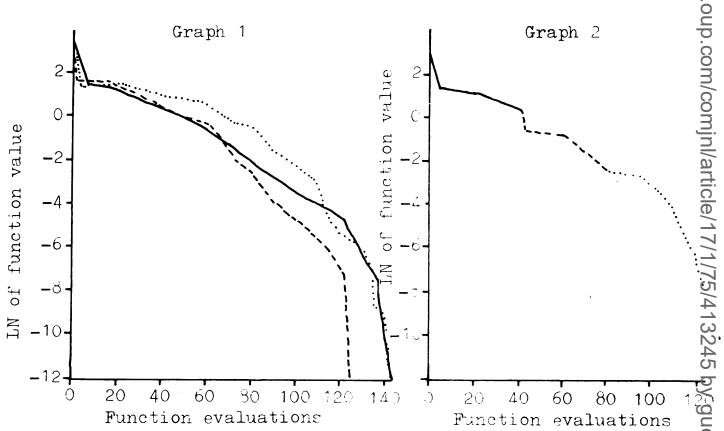
This function, given in the above reference along with an ALGOL program for its evaluation, is of a type which is more regular than the others and often occurs in practice. The number of variables (n) involved in the function is easily changed. This function was tested with 2, 4, 6 and 9 variables using the convenient initial approximation for a starting vector of $x_i = i/(n + 1), i = 1, \ldots, n$. The minimisation is valid for all n.

4. Results

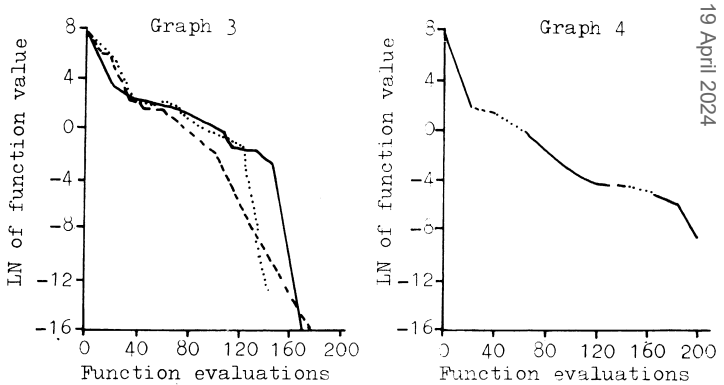
The results are shown in **Table 1** and in the fourteen graphs, using the following conventions:

	Log.	
Odd number graphs	—	separate methods
Even number graphs	—	combined method

Graphs based on	*Graph notation*
Swann and Rosenbrock	————
Simplex	--------
Powell	. . . . .



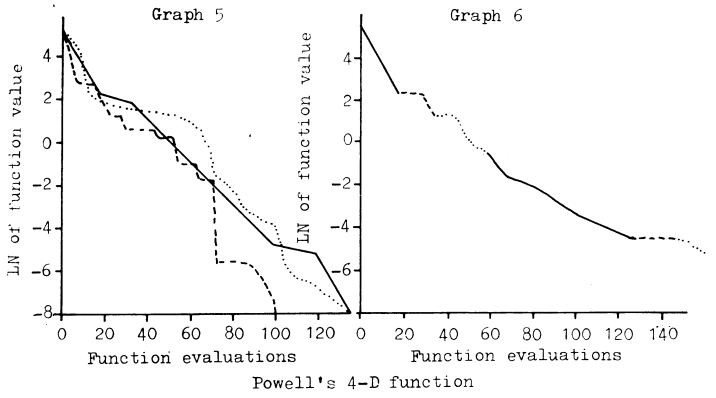
Rosenbrock's function

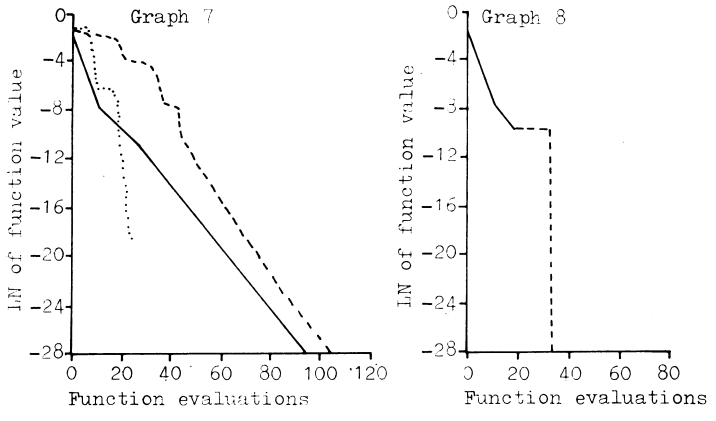


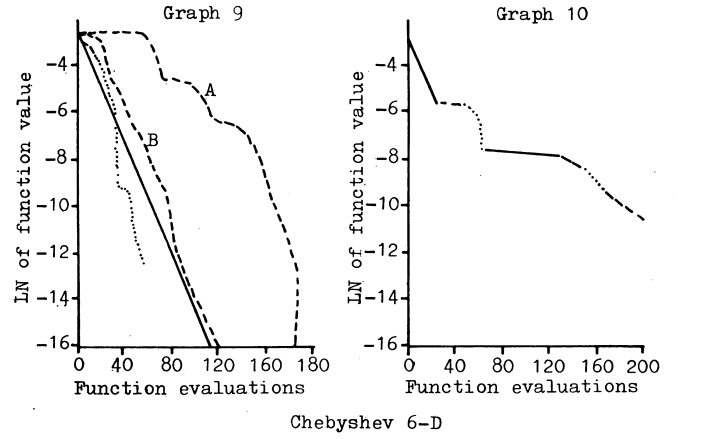
Helical valley function

5. Discussion of results

The Simplex method, modified by the author to deal with large initial step sizes, was found to give considerable improvement in both the individual and combined programs. Graphs 9, 11 and 13 show this for the individual program, the origi-



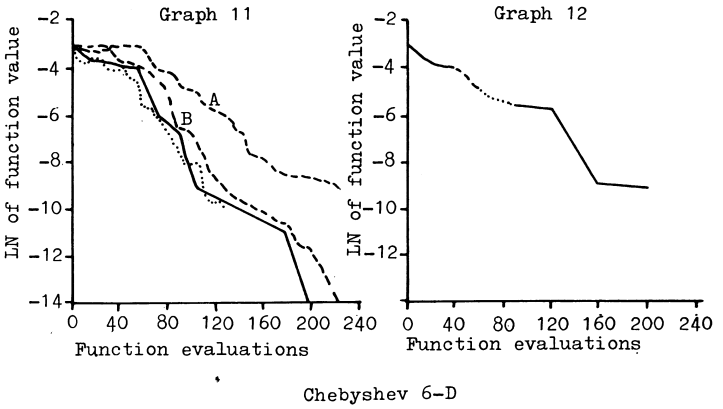


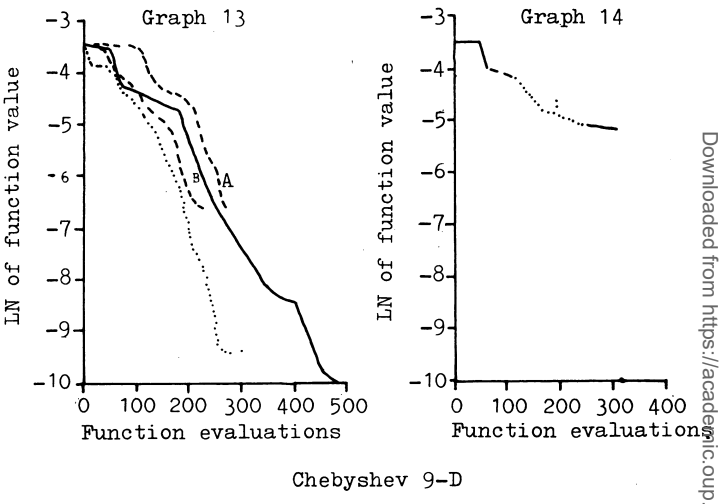


nal method giving curve A and the modified one giving curve B. The table of results also gives unmodified results distinguished with an asterisk.

The test functions optimised did not indicate any one method as being outstandingly suitable in their solution. Indeed, it was generally found that the three methods used to optimise a particular function gave very close results. Only in graphs 7 and 13 does one method stand out. This method is Powell's and in general is the better, making good progress in graphs 3, 7, 9, 11, 13.

A reason for the combined program's slowness is that it has been deliberately made simple, as initially it was felt that the transference of too much data would slow down the method. The results, however, show that the entry into a different optimisation procedure drastically decelerates the process. This is because search directions, gradients, simplexes, or in general a knowledge of the local landscape are lost or not known on transference. Consequently this information has to be laboriously calculated from the 'setting up' section of the new procedure. To overcome this it is felt that as many global variables as possible should be used and minor modifications of procedures undertaken to allow for this. Also, it is likely that





the old values of variables (still available in the procedure's storage area, when specified as 'own' variables in ALGOL) more closely represent those required than an arbitrary starting system. Hence by by-passing the 'setting-up' stage of the procedures and using old values still present in the machine, the amount of computation would be reduced whilst increasing the rate of optimisation.

Other slightly more sophisticated modifications in the change-over system which would provide the new procedure with accurate directions, step-sizes, etc. would undoubtedly require extra changeover time, but if done wisely this could be negligible when compared with the time saved through the procedure having a flying start.

One aspect mentioned in this paper has not been tested by the functions given in Section 3. It is thought that a combined method would prove to be a more general technique, in that it could optimise functions when some or all of the individual methods could not find a solution. This type of situation occurs in Box's paper (1966) where he shows that some optimisation methods fail when started at certain points of specific functions, but succeed from others. In such cases, then, even though all methods fail individually from a given starting vector, by solving with a combined program one in effect changes the starting vector each time a procedure is re-entered and hence from one of these points one would expect to find the true optimum.

6. Conclusions

The combined method was found to be generally not as good as the individual methods. The universality of a combined method has yet to be proven. At this stage, therefore, the combined method seems to have no future since no efficient changeover process is available, resulting in loss of information.

The individual methods, however, can be used independently and efficient programs have been formed for these with modi-

fications as outlined in the text. The improvement to the Simplex method is the most significant.

7. Acknowledgements

I am grateful to the English Electric Company for allowing me to publish this work, and especially Dr. N. R. Tomlinson of the Computer Centre, The University of Aston in Birmingham and Mr. J. S. Glass of the Nelson Research Centre, English Electric (Stafford) for their encouragement and assistance.

Appendix

Optimisation techniques usually have to complete several hundred evaluations of the function under investigation before the optimum is achieved. Even when the calculations are done on a modern high speed computer, this work still takes considerable time and costs a lot of money. Consequently the length of time that a program takes to compute is of great importance.

The argument has been that an optimisation program, investigating a fairly large function, has its run time dominated by the time it takes to calculate the large number of function evaluations required by the method. This means that the run time is practically proportional to the number of function evaluations.

Many published results are given in terms of function evaluations and function values, even for small functions, and it was found that some such results give the reader a very inaccurate picture of the comparative speed of different methods.

For example, using Rosenbrock's valley function, Rosenbrock's method (Swann, 1964) is found to get close to the optimum 2·5 times faster than Powell's method (Powell, 1965). In terms of function evaluations, however, the methods appear to be equally fast in reducing the function to 10^{-7}.

This type of inaccuracy shows the need for a knowledge of the run time, preferably available in the program, during any computation. In a combined program run time would not only give accurate times for the procedures, but it would automatically take care of changeover and decision times.

As run time was not readily available to the program during execution, this initial investigation was carried out using function evaluations. The need for measurement of run time should, however, be noted and developed in future work.

References

BOX, H. J. (1966). A comparison of several current optimisation methods. *The Computer Journal* Vol. 9, p. 67.

FLETCHER, R. (1965). Function minimisation without evaluating derivatives—a review. *The Computer Journal*, Vol. 8, pp. 33-41.

FLETCHER, R. (Editor) (1969). *Optimisation*, Academic Press, London and New York. (Symposium of the I.M.A., University of Keele, England 1968).

FLETCHER, R. and POWELL, M. J. D. (1963). Rapidly convergent descent methods for minimisation. *The Computer Journal*, Vol. 6, pp. 163-168.

NELDER, J. A. and MEAD, R. (1965). A simplex method for function minimisation. *The Computer Journal*, Vol. 7, pp. 308-313.

POWELL, M. J. D. (1962). An iterative method for finding stationary value of a function of several variables. *The Computer Journal*, Vol. 5, pp. 147.

POWELL, M. J. D. (1965). An efficient method for finding the minimum of a function of several variables without derivatives. *The Computer Journal*, Vol. 7, pp. 155-162.

ROSENBROCK, H. H. (1960). An automatic method for finding the greatest and least value of a function. *The Computer Journal*, Vol. 3, pp. 175-184.

SWANN, W. H. (1964). *Report on the development of a new direct search method of optimisation.* I.C.I. Limited, Central Inst. Lab. Research notes 64/3.

Book review

High Level Programming Languages—the way ahead. Proceedings of a BCS conference, 1973; 140 pages. (*NCC*, £5·00)

This publication is based on the recorded proceedings of a British Computer Society conference of the same title held at the University of York in October 1972. The conference itself was an enjoyable occasion, and some of this enjoyment comes across when transferred to cold print.

Sandwiched between brief opening and closing remarks by Dr. D. F. Hartley, the papers are concerned with 'The Way Behind' by C. A. R. Hoare, 'Objectives for a General Purpose Language' by E. H. Sibley, 'FORTRAN' by B. H. Shearing, 'ALGOL 68' by P. M. Woodward, 'PL/I' by D. Beech, 'SIMULA' by G. M. Birtwistle, 'APL' and 'POP-2' by D. W. Barton, 'System Programming Requirements' by I. C. Pyle and 'Future Prospects' by B. L. Neff.

A major disappointment is the absence of Commander (now Captain) Grace Hopper's stimulating paper on COBOL, which is reduced to merely a one-page summary of nine points from her talk, as unfortunately the 'edited paper was not available at the time of going to press'. Personally I find it hard to imagine anything less stimulating than COBOL, or anyone much more stimulating than Captain Hopper. The combination of the two has a strange savour that this summary cannot begin to capture.

Each paper is followed by an edited version of the discussion that followed it at the conference, and there is also a report of the main discussion session that was so ably conducted by T. W. Olle, with Commander Hopper, D. Beech, B. H. Shearing and P. M. Woodward as panellists. The editing of these discussions has been well done to produce a meaningful result, not the garbage that often appears in conference reports.

The text is in typescript, with certain characters added in manuscript. Nowadays, when devices are available that enable typewriters to type a wide range of characters, there seems little excuse for these excursions into manuscript; but at least the accuracy of both the typing and the manuscript insertions, while not perfect, is considerably better than one often sees. (Beech and Woodward each have their names mis-spelled on at least one occasion. In a computing context 'Woodware' is at least a good joke.)

The way ahead? Judging from the papers presented here it is hard to see how the title could logically be awarded to any of them but ALGOL 68, although some of its apparent stature relative to the others may be due merely to Dr. Woodward's admirable presentation. In practice the language that triumphs is much more likely to depend on 'computer politics' than upon intrinsic merits.

Perhaps it is symbolic that the front cover design shows FORTRAN, POP-2, SIMULA, APL, ALGOL 68 and PL/I all setting off in different directions—none of them corresponding to the way ahead.

I. D. HILL (London)