E. D. EASON
Mechanical Engineering Dept.,
University of California,
Berkeley, Calif.

R. G. FENTON
Associate Professor,
University of Toronto,
Toronto, Ontario,
Canada

A Comparison of Numerical Optimization Methods for Engineering Design

Seventeen numerical optimization methods are compared by plotting their convergence characteristics when applied to design problems and test functions. Several ranking schemes are used to determine the most general, efficient, inexpensive, and convenient methods. Conclusions are presented in the form of a selection guide intended for general use.

Introduction

MANY methods are available for solving the constrained nonlinear optimization problems which arise in mechanical design. However, it is difficult to choose a generally useful algorithm, because (1) comparative data is limited, and (2) design problems may include difficulties not treated in the mathematical programming literature (such as implicit, nondifferentiable functions). This paper is intended to aid the selection of a useful code[1] by presenting the method and conclusions of the authors' comparative study [1],[2] with discussion of earlier studies.

Literature Survey. Prior to Colville's landmark study in 1968 [2], the available comparison studies [3–5] covered primarily unconstrained, mathematical problems. The basis of comparison was the number of function evaluations needed to obtain approximately the same answer, but none of the studies explicitly ranked the codes tested.

Colville sent eight constrained problems to the developers of thirty codes, requesting that they return their solution, preparation time, solution time, and number of function and constraint evaluations required. The results indicated that solution time was a more valid performance index than number of function evaluations, so a method for ranking timing results was developed. The procedure was to have each participant run a standard timing routine supplied by Colville, and then divide reported solution times by the time required for the timing routine. The resulting normalized times could then be compared even though many different computers were used. A ranking scheme based on solution time and generality (number of problems solved) was used for quantitative ranking.

Several recent studies have used Colville's methods and results. Abadie and Guigou [6] applied a different ranking scheme to Colville's data, including results on their new code. Stocker [7] used Colville's timing method and test problems; his results (and others) are summarized in Chapter 9 of [8]. Neither [7] nor [8] includes an overall ranking.

While the results of [2, 6–8] are more useful than the earlier studies, there are major difficulties. The accuracy of solution on each problem was not held constant for all codes, so that fast but inaccurate codes would appear superior. Those codes requiring derivatives of the objective function and constraints were sup-

[1] Code here means the FORTRAN implementation of an abstract algorithm.

[2] Numbers in brackets designate References at end of paper.

Contributed by the Design Automation Committee of the Design Engineering Division of THE AMERICAN SOCIETY OF MECHANICAL ENGINEERS and presented at the Design Engineering Technical Conference, Cincinnati, Ohio, September 9–12, 1973. Manuscript received at ASME Headquarters, June 6, 1973. Paper No. 73-DET-17.

Nomenclature

C = total running cost, dollars, equation (10)
C_p = estimated preparation cost, dollars
f_a = ratio of time for code "a" over fastest time, equation (7)
$\bar{f}_a$ = ratio of time for code "a" over average time, equation (8)
$f(x)$ = objective function to be minimized
$g(x)$ = constraints, feasible when $g(x) \geq 0$
m = total number of constraints
n = total number of independent variables
N = composite generality index, equation (6)
n_p = number of "P" ratings for given code
n_s = number of problems solved
t_{ap} = normalized time required by code "a" to solve problem "p"
T_a = total normalized execution time, equation (9)
$U(x)$ = artificial constrained function to minimize
x = vector of independent variables
x_i = the ith element of x
x^0 = the optimum point
ϵ = value of relative error at which comparisons are made

plied analytic derivatives, without penalty for the greatly increased preparation time. The results in [2] and [6] include unknown and variable efforts by the participants to optimize the convergence of their own codes. Finally, the overall rankings in [2] and [6] emphasize solution speed, though generality and ease of preparation are perhaps more important characteristics of a general optimizing code.

Scope of the Study. The authors' study is designed to test seventeen readily available codes on ten problems in such a way that the previous difficulties are avoided. The major differences are evaluation of solution time at precisely the same accuracy, and the inclusion of several ranking schemes, covering generality, efficiency, execution cost, and preparation cost. Other differences include the use of approximate derivatives, inclusion of mechanical design problems, and the attempt to give each code an equal chance to solve the problems, without modifying the code for optimum covergence. Less important details have been omitted here; the interested reader should refer to [1].

Table 1 Code descriptions

NAME & SOURCE	DESCRIPTION	ALGOR. CLASS[1]	PENALTY FUNCT.	EXTRA CODING[2]
ADRANS [10]	Random search followed by pattern moves	DS	Eq. (1)	
CLIMB [15]	Rosenbrock [14] procedure	DS	Special	
DAVID [10]	Davidon-Fletcher-Powell [12], num. deriv.	G	Eq. (11)	
DFMCG [16]	Fletcher-Reeves conjugate grad. w/Eq. (5)	G	Eq. (3)	P, O, D
DFMFP [16]	[12], using derivatives from Eq. (5)	G	Eq. (3)	P, O, D
FMIND [17]	Pattern search [11]	DS	Eq. (3)	P, O
GRAD4 [18]	Steepest descent method	G	Eq. (3)	P
GRID1 [18]	Grid & star network search, w/shrinkage	AR	Eq. (1)	P
MEMGRD [10]	[12], with retained step length info.	G	Eq. (11)	
NMSERS [15]	Simplex search [13]	DS	Eq. (3)	P, O
PATSH [19]	[11], w/relative steps, stepsize growth	DS	Eq. (3)	P
PATRN0 [18]	PATTRN with IRIDGE = 0, [11] dome strategy	DS	Eq. (1)	P
PATRN1 [18]	PATTRN with IRIDGE = 1, [11]ridge strategy	DS	Eq. (1)	P
RANDOM [10]	Random search with shrinkage	AR	Special	
SEEK1 [10]	[11], followed by random search & restart	DS	Eq. (1)	
SEEK3 [10]	Sequential pattern search [11] rel. steps	DS	Eq. (11)	
SIMPLX [10]	Sequential simplex search [13] rel. steps	DS	Eq. (11)	

NOTES: 1. DS = direct search (uses function only), G = gradient (also needs $\partial U/\partial x_i$), AR = area reduction (squeezes in on region containing x^0)
2. User must supply: P = penalty, O = output, D = derivatives $\partial U/\partial x_i$

Method

Preparation of Codes. Brief descriptions of the codes tested and the problems used are in Tables 1 and 2, respectively; listings and complete description are available in [9] or from the authors. Except for the following modifications, the codes were generally used as received, using the parameters suggested by documentation:

Code Modifications

1 Modified for IBM 370 model 165 FORTRAN IV (level G compiler), using implicit double precision.
2 Large blocks of comment cards removed.
3 All built-in output suppressed.
4 Penalty functions and secant derivative subroutines supplied as needed.

Modification 4 was necessary because some codes were designed for unconstrained problems or for analytic derivatives. Simple exterior penalty functions were used, of the form:

$$U(x) = f(x) + 10^{20} \sum_{i=1}^{m} a_i |g_i(x)| \tag{1}$$

with

$$a_i = \begin{cases} 1, & g_i(x) < d \\ 0, & g_i(x) \geq d \end{cases} \quad \text{and} \quad d = \begin{cases} -0.5 \times 10^{-6} & \text{for (1)} \\ 0 & \text{for (3)} \end{cases} \tag{2}$$

or

$$U(x) = f(x) + W \sum_{i=1}^{m} a_i [g_i(x)]^2 \tag{3}$$

with

$$W = \begin{cases} 4 \times 10^5 |f(x)|, & |f(x)| \geq 1 \\ 4 \times 10^5, & 0 \leq |f(x)| < 1 \end{cases} \tag{4}$$

Equation (4) assures that the weight W given to the penalty is not unreasonable compared to the current value of $f(x)$. The decision to use equation (1) or (3) and the choice of a reasonable value for W were based on experimentation.

All gradient methods (see Table 1) used secant approximations to the partial derivatives. DFMCG and DFMFP used the form

$$\frac{\partial U}{\partial x_i} \cong \frac{U(x_1, \ldots, x_i + \Delta, \ldots, x_n) - U(x_1, \ldots, x_i, \ldots, x_n)}{\Delta} \tag{5}$$

Table 2 Problem description

No. & Source	QUANTITY TO MINIMIZE;	CONSTRAINTS;	f(STARTING POINT), f°(OPTIMUM POINT)
1 [2]	$\left[\sum_{j=1}^{5} e_j x_j + \sum_{j=1}^{5}\sum_{i=1}^{5} c_{ij} x_i x_j + \sum_{j=1}^{5} d_j x_j^3\right]/10$;	Linear (see [2]);	Opt: f°(0.3, 0.33347, 0.4, 0.42831, 0.22397) = -3.2349
2 [14]	$-x_1 x_2 x_3/1000$ (Max. volume box to meet P.O. specifications);	$0 \leq x_1 + 2(x_2 + x_3) \leq 72$, $0 \leq x_2 \leq 11$; $0 \leq x_1 \leq 20$ $0 \leq x_3 \leq 42$;	f(10, 10, 10) = -1 f°(20, 11, 15) = -3.3
3 [2]	$[ax_3^2 + bx_1 x_5 + cx_1 + d]/10^4$ (Process design) ;	Quadratic (see [2]);	f(78.62, 33.44, 31.07, 44.18, 35.22) = -3.04 f°(78, 33, 29.995, 45, 36.776) = -3.0665
4 [2]	$100(x_2 - x_1^2)^2 + (1-x_1)^2 + 90(x_4 - x_3^2)^2 + (1-x_3)^2$ $+10.1[(x_2-1)^2 + (x_4-1)^2] + 19.8(x_2-1)(x_4-1)$;	$-10 \leq x_i \leq 10$ $i = 1, 2, 3, 4$;	f(-3, -1, -3, -1) = 2 x 10^4 f (1, 1, 1, 1) = 0
5 [14]	$100(x_2 - x_1^2)^2 + (1-x_1)^2$;	assume $-2 \leq x_i \leq 2$, $i = 1, 2$;	f(-1.2, 1) = 24.2 f°(1, 1) = 0
6 [20]	$(.44x_1^3 x_2^{-2} + 10x_1^{-1} + .592x_1 x_2^{-3})/10$ (Journal bearing design) ;	$1 - 8.62x_1^{-1}x_2^3 \geq 0$ $0 \leq x_i \leq 5$, $i = 1, 2$;	f(2.5, 2.5) = 0.51947 f°(1.2867, 0.53047) = 1.6206
7 [21]	$-.0201x_1^4 x_2 x_3^2/10^7$ (Flywheel design);	$675 - x_1^2 x_2 \geq 0$, $0 \leq x_1 \leq 36$, $0 \leq x_2 \leq 5$ $.419 - (x_1 x_3)^2/10^7 \geq 0$, $0 \leq x_3 \leq 125$;	f(22.3, 0.5, 125) = -4 f°(x) = - 5.684802
8 [18]	$[12 + x_1^2 + (1+x_2^2)/x_1^2 + (x_1^2 x_2^2 + 100)/(x_1 x_2)^4]/10$ (Gear train of minimum inertia)	$1 \leq x_i \leq 3$ $i = 1, 2$;	f(0.5, 0.5) = 2563.3 f°(1.7435, 2.0297) = 1.74
9 [18]	Design of smallest cam with specified follower function and limit on max. pressure angle. f(x) & g(x) implicitly coded in performance analysis. [1]		
10 [9]	Design of 4-bar mechanism to generate specified path with min. error and min. mechanism size. f(x) implicitly coded in performance analysis. [1]		

where Δ is the greater of $10^{-4}|x_i|$ and 10^{-6}.

Evaluation of Time to Specified Error. On any problem, it is unreasonable to expect all codes to terminate with the same accuracy without defeating the supplied termination test (which was done by [5]). Rather than make such a drastic modification, we repeatedly solve each problem, varying the termination parameter of each code in about five steps over a wide range. The time[3] required for the code to terminate and the relative error[4] of the solution (compared to the known optimum) is then plotted, as in Fig. 1. Using the resulting convergence characteristics, the time required by each code to precisely obtain an error ϵ can easily be found.

The value of ϵ used for ranking codes varied from problem to problem. It was chosen (after the convergence characteristics were plotted) to be the smallest error actually reached by every successful code (e.g., 2×10^{-4} for Fig. 1). A code was judged unsuccessful if it could not converge, did not progress toward the optimum, or had constraint violations greater than 10^{-6}. When convergence was the only difficulty (i.e., substantial progress and feasible result), the code would be rated "P" for progress. Table 3 reports the detailed results.

In addition to the execution time, the core required, number of cards read and listed, compilation time, and preparation time were noted for later ranking. The number of iterations and function evaluations were also recorded. The results confirmed [2] and [7] that number of function evaluations is not a valid basis of comparison, even on the most difficult problems.

Ranking Schemes

Generality. Number of problems solved, n_s, and

$$N = n_s + \frac{n_p}{2} \tag{6}$$

Equation (6) is motivated by the feeling that a "P" rating is halfway between failure and success.

Efficiency (Solution Time to Specified ϵ).

$$f_a = \frac{\sum_{p=1}^{10} b_{ap}\left[\dfrac{t_{ap}}{\min_a(t_{ap})}\right]}{n_s} \tag{7}$$

where $b_{ap} = 1$ if code "a" solved problem "p" (otherwise $b_{ap} = 0$), n_s is the number of problems solved by code "a" (4 minimum), and $\min_a(t_{ap})$ is the shortest time recorded by any code on problem "p." The interpretation given by [6] is that f_a indicates the average of the ratios of the time required by code "a" to the time required by a hypothetical code which would solve all problems as fast as the fastest code does.

An alternative form is

$$\bar{f}_a = \frac{\sum_{p=1}^{10} b_{ap}\left[\dfrac{t_{ap}}{\text{mean}_a(t_{ap})}\right]}{n_s} \tag{8}$$

where n_s and b_{ap} are as in equation (7), and $\text{mean}_a(t_{ap})$ is the average time of all codes solving problem "p." $\bar{f}_a$ represents the average ratio of time used by code "a" to the time which a hypothetical average code would use.

Efficiency, Generality, and Difficulty of Solved Problems.

$$T_a = \sum_{p=1}^{10} t_{ap} \tag{9}$$

[3] Normalized by Colville's method, where his timing program required 6.93 sec to execute in double precision.

[4] The relative error in $f(x)$ was used in problems 1, 2, 8; the other problems used relative error in x, calculated by

$$\sqrt{\sum_{i=1}^{n}\left[\frac{x_i - x_i^0}{x_i^0}\right]^2}$$

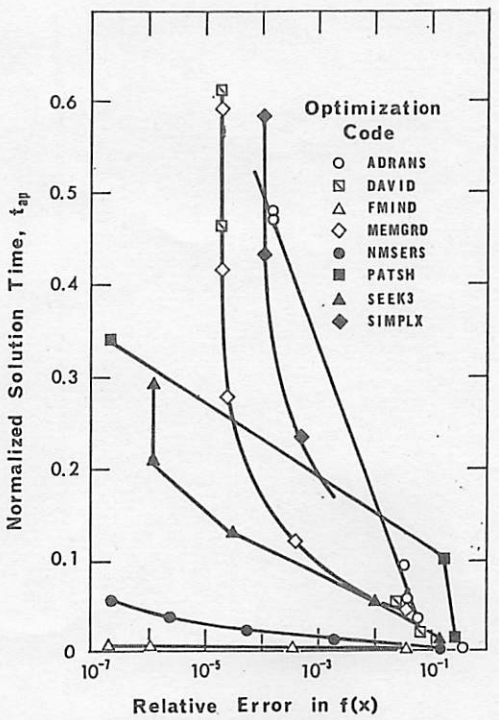

Fig. 1 Convergence characteristics on problem 2

Table 3 Performance of optimization codes

CODE NAME	PROBLEM NUMBER AND ERROR VALUE ϵ									
	1 10^{-4}	2 2×10^{-4}	3 10^{-3}	4 10^{-3}	5 10^{-3}	6 2×10^{-3}	7 10^{-5}	8 10^{-5}	9 2×10^{-3}	10 10^{-4}
ADRANS	2.64	0.458	1.65	F	0.100	0.654	0.159	0.069	F	F
CLIMB				0.015	0.005			0.007		
DAVID		0.188	0.84	1.132	0.075	0.046				
DFMCG			F	0.015	F			0.004		
DFMFP			F	0.038	0.250		F	0.387		
FMIND		0.004	F		0.140		0.003	0.003	F	3.74
GRAD4		F	F	0.283	0.393		F	0.004		F
GRID1	F	F		0.033	F	F		0.037	F	F
MEMGRD		0.143	0.91	0.059	0.066	0.067	F			
NMSERS		0.019	0.045	0.060	0.002	0.012	F	0.007	0.39	F
PATSH	1.78	0.220	1.00	0.013	0.018	0.020	0.010	0.009	F	3.54
PATRN0	F	F	F	0.021	F			0.002	F	1.95
PATRN1	F	F	F	0.008	0.002	F		0.001	1.02	1.20
RANDOM		F		F	F	0.024	1.20	0.013		F
SEEK1	F	F	F		0.010		0.010	0.007	F	1.53
SEEK3		0.102	0.14	0.212	0.035	0.191	0.141	0.013	4.20	F
SIMPLX	2.97	0.297	1.31	0.196	0.035	0.191	0.262	0.050	F	

where t_{ap} is set to twice the time of the slowest code solving problem "p" if code "a" did not solve problem "p."

Equation (9) is a simple way to penalize codes which could not solve many problems, or were slow on the difficult problems. Adding twice the slowest time is intended to represent the execution time lost when code "a" fails and a second code must be tried. By summing, the difficult problems will exert greater influence, reflecting the reality that the execution time must be significant for it to matter whether a code is fast or only average. T_a may be thought of as the total time required to solve the 10 problems, using more than one code as needed.

Total Running Cost.

$$C = 8.5\,(\text{CPU}) + 0.0105\,(\text{CORE USAGE}) + 0.0008\,(\text{I/O}) \quad (10)$$

where CPU is the sum of compilation and execution (T_a) times in min, CORE USAGE is basically the product of the number of storage locations and the time they are in use, and I/O is the number of cards read plus lines printed. The resulting cost C is in dollars.

This is the charging formula actually applied to the authors' computer runs. As such, it provides a realistic weighted function for ranking execution and compilation times, core usage, and physical deck size. Generality was also included by the use of T_a and application of multiplicative penalties to CORE USAGE and I/O, depending on number of problems solved. C is intended to represent the overall running cost to solve the 10 problems, using two or more codes as needed.

Estimated Preparation Cost. As indicated in Table 1, the OPTISEP [10] codes did not require any extra coding; they are also well documented. In an attempt to compare preparation costs, we consider a problem requiring 1.5 hr preparation for any OPTISEP code (short main program, short function and constraint subroutines). This same time would be used for the same amount of preparation on all the other codes except CLIMB and NMSERS, which are penalized for their inadequate documentation by the addition of 1.3 and 1.0 hr, respectively. Then from Table 1, 0.5 hr are added to all methods not providing built-in output, 4 hr are added if a penalty function was not supplied, and 3 hr if gradient routines were required, but not supplied.

The resulting time estimates represent the time required by a good programmer to set up each code to solve a constrained problem, starting with just the optimization subroutine and the supplied documentation. To facilitate comparisons with the running cost, the time estimates are multiplied by a preparation charge of \$10 per hr before entry in Table 4.

Discussion

General. The results of all ranking schemes are in Table 4. The tested codes present a wide variation in generality and preparation cost, with less variation in running cost. The running cost is actually less than the preparation cost for most of the originally unconstrained, inconvenient, or poorly documented codes. Had analytic derivatives been used, C_p would have further increased for some codes. Obviously, future effort should stress convenience to reduce total optimization cost.

All comparisons of execution time are only valid for the values of ϵ selected, since the convergence curves (Fig. 1) intersect. However, rankings valid at a particular ϵ are better than rankings based on unequal values of ϵ (as in the earlier studies [2, 6–8]). Fig. 1 conclusively shows that a time comparison of the leftmost point of PATSH with any nearby point by DAVID, MEMGRD, or SEEK3 will be totally misleading. The shape of the curves for the latter codes indicates they may *never* reach the same precision as PATSH, though they might appear faster for greater errors. The same situation undoubtedly occurs for any group of codes and problems, casting doubt on the time rankings in [2] and [6].

In Fig. 1, the shape of the curve for each code is fairly typical of that code on all test problems. By noting whether this shape is concave upward, downward, or linear, we can predict the relative rate of convergence for ϵ larger and smaller than the values of ϵ used in this study, as in Table 5.

Table 4 Relative ranking of optimization codes

GENERALITY		EFFICIENCY		GEN. + EFF.	RUNNING COST	PREP. COST
n_p	N (Eq.(6))	f_a (Eq.(7))	$\bar{f}_a$ (Eq.(8))	T_a (Eq.(9))	C (Eq.(10))	C_p (Est.)
9 PATSH	9.5 PATSH	1.3 PATRN1	0.1 NMSERS	15 PATSH	23 PATSH	ADRANS
8 SEEK3, SIMPLX	8.5 ADRANS, SEEK3, SIMPLX	3.3 NMSERS	0.2 PATRN1	16 NMSERS, PATRN1	28 NMSERS, SEEK3	DAVID, MEMGRD
7 ADRANS, NMSERS	8.0 NMSERS	4.2 SEEK1	0.3 SEEK1	18 SEEK3	32 SIMPLX	15 RANDOM, SEEK1, SEEK3, SIMPLX
5 DAVID, FMIND, MEMGRD, PATRN1	7.0 PATRN1	12 PATSH	0.6 PATSH	21 SIMPLX	33 PATRN1	
	6.0 FMIND, SEEK1	16 FMIND	0.8 MEMGRD	24 ADRANS, SEEK1	35 FMIND	
	5.5 MEMGRD, PATRNO	20 MEMGRD, SEEK3	0.9 FMIND		37 ADRANS	
4 SEEK1		38 SIMPLX	1.0 SEEK3	25 FMIND, PATRNO	39 MEMGRD	
3 CLIMB, DFMFP, GRAD4, PATRNO, RANDOM	5.0 DAVID, GRAD4, GRID1, RANDOM	50 DAVID	1.5 SIMPLX		40 SEEK1	28 CLIMB
	4.0 DFMFP	54 ADRANS	1.7 DAVID	26 MEMGRD	41 DAVID	GRAD4, GRID1
2 DFMCG, GRID1	3.0 CLIMB, DFMCG		2.4 ADRANS	27 DAVID	46 PATRNO	55 PATRNO, PATRN1, PATSH
				30 CLIMB, DFMFP, GRAD4, RANDOM	53 DFMCG, DFMFP, RANDOM	
					54 CLIMB	60 FMIND
				31 DFMCG, GRID1	55 GRAD4	70 NMERS
					56 GRID1	90 DFMCG, DFMFP

Table 5 Predicted speed of solution

Code	Small ϵ	Moderate[a] ϵ	Large ϵ
PATSH	average to fast	average to fast	slow to average
NMSERS	fast	fast	fast
SEEK3	slow or incapable	average	average to fast
SIMPLX	slow or incapable	slow to average	average

[a] Based on $\bar{f}_a$ in Table 4.

We can also compare the efficiency of the penalty functions used by noting that all of the codes with steep concave upward curves (e.g., SEEK3, SIMPLX) use a sequential interior penalty of the form:

$$U(x) = f(x) + r_k \sum_{i=1}^{m} \frac{1}{|g_i(x)|} \quad (11)$$

where $U(x)$ is repeatedly minimized as the weight r_k is reduced, using each $(x^0)_k$ as the starting point for the $k+1$ minimization. These codes are effectively prohibited from obtaining highly accurate solutions by the steep upturn in their convergence curve. It is significant that all codes using equation (11) have this problem, while similar codes using equation (3) (PATSH, NMSERS, FMIND) do not. On the strength of this evidence and the comparable generality of codes using either penalty, we judge equation (3) superior to equation (11).

The effect of scaling is apparent in the rather different rankings (see Table 4) of PATSH and FMIND, which use the same basic algorithm [11]. PATSH scales the step lengths to a fraction of the current x_i (similar to Δ in equation (5)), while FMIND does not. Other successful codes (e.g., SEEK3, SIMPLX) use step lengths that are a fraction of the feasible range for each x_i. Scaling is essential when the magnitudes of the variables are widely different; automatic scaling as in PATSH and SEEK3 is highly desirable to reduce preparation time.

Comparison by Algorithm Class. The best four codes by any ranking[5] are direct search algorithms, and this class dominates the upper half of every column in Table 4. In contrast with earlier studies [2–8], none of the gradient methods performed better than average.

The major reason for the poor showing by gradient methods seems to be the use of secant derivative approximations. All three tested versions of the Davidon-Fletcher-Powell [12] algorithm performed poorly, yet the algorithm is highly rated when analytic derivatives are used [2–6]. Colville [2] provides additional evidence, since his group of "small-step gradient methods" (which use derivative approximations) also behaved badly compared to the codes using analytic derivatives. For conclusive proof, it would be necessary to try the same codes with both types of derivatives; this cannot be done for all test problems.[6]

[5] C_p is ignored throughout this section.

[6] Particularly, problems 9 and 10. The original study [1] included two other nondifferentiable problems, as well.

The better direct search methods are based on the pattern search [11] and polyhedral simplex [13] algorithms. The Rosenbrock [14] code (CLIMB) was far less successful, confirming earlier studies [2, 3, 5, 7]. The pattern search algorithm is not represented in the recent studies [2, 7], but the simplex algorithm is, ranking about average in [2] and earning highest recommendations in [7]. Our results indicate that either algorithm can perform better than gradient methods if the latter use approximate derivatives.

Best Code(s). The ideal computer code for design optimization should solve any problem conveniently and at moderate cost. Since no single code has yet demonstrated complete generality, several codes must be used, with different methods for handling constraints. For convenience, the codes should interchange easily when failure occurs (or to verify results), and ample documentation must be supplied. Preparation time should be minimized by automatic scaling and by avoiding codes requiring analytic derivatives of the function and/or constraints. A highly convenient package with minimum preparation, but average execution speed would be perfectly acceptable, because total cost in use would be low.

A package approach using OPTISEP [10] has the features listed above, with better than average speed. An OPTISEP user need only write subroutines containing his function and constraints and a main program with dimensioning and a call statement. Once set up this way, any OPTISEP routine can be used, by changing only the calling statements. A particular package strategy might first try the fastest code (SEEK1), then more general codes (SEEK3, SIMPLX) in case SEEK1 fails or to check results. Using this strategy on the nine test problems solved by PATSH (see Table 3), and including the time to failure before trying the next code, the sum of t_{ap} would be 5.55 for the package, compared to 6.61 for PATSH. The package time would be further reduced if the improved point found by SEEK1 on problems 1, 2, 3, and 9 were used as the starting point for the next code. More importantly, the package strategy would have solved every test problem, and the solution could be immediately verified by an independent algorithm.

Though no single code can match the package approach, one code does stand out above the others in this study. PATSH solved or made progress on every problem, with better than average solution speed ($\bar{f}_a$). This is remarkable, considering that PATSH is less than half as long as most codes—72 FORTRAN statements, including the penalty function equation (3), but excluding prints and comments. This makes PATSH attractive for small computers and repeated use. In addition, the shape of the convergence curve for PATSH is linear to slightly concave downward, indicating that arbitrarily high precision may be obtained.

Conclusions

The authors' study has produced a number of observations related to choosing a useful optimization method. These are:

1 Codes requiring analytic derivatives should be avoided if possible, in order to reduce preparation cost and chance of human error, and to permit application to general design problems.

2 Pattern [11] and Simplex [13] direct search methods are generally better than gradient methods using secant derivative approximations.

3 Penalty functions similar to equation (11) may not permit high-precision solutions; equation (3) does not have this problem.

4 Built-in problem scaling increases generality and efficiency.

5 Comparisons of efficiency may not be meaningful unless convergence characteristic curves are used to compare times at the same error. The curves also permit detailed performance analysis over a range of accuracies.

6 Several ranking schemes should be considered to fully evaluate generality, efficiency, and cost.

7 The most general code need not be large and slow.

8 Execution cost may be small compared to preparation cost—favoring convenient codes with adequate documentation.

9 A package strategy is preferred for design optimization because of greater generality, ease of checking results, and lower total cost.

Though strictly valid only for the present study, it is hoped that these comments may help others to choose useful codes, without a lengthy comparison study.

Acknowledgments

The authors are grateful for the codes contributed by C. R. Mischke, J. N. Siddall, D. E. Whitney, and others, for the computer time donated by the University of Toronto Computer Center, and for the fellowship support of the National Science Foundation.

References

1 Eason, E. D., and Fenton, R. G., "Testing and Evaluation of Numerical Methods for Design Optimization," UTME-TP 7204, University of Toronto, Sept. 1972.

2 Colville, A. R., "A Comparative Study on Nonlinear Programming Codes," IBM New York Scientific Center Report No. 320–2949, June 1968.

3 Kowalik, J., and Osborne, M. R., Chapter 6, *Methods for Unconstrained Optimization Problems*, American Elsevier, New York, 1968.

4 Leon, A., "A Comparison among Eight Known Optimizing Procedures," *Recent Advances in Optimization Techniques*, Lavi, A., and Vogl, T. P., ed., Wiley, New York, 1966, pp. 23–41.

5 Box, M. J., "A Comparison of Several Current Optimization Methods, and the use of Transformations in Constrained Problems," *Computer Journal*, Vol. 9, No. 1, May 1966, pp. 67–77.

6 Abadie, J., and Guigou, J., "Numerical Experiments with the GRG Method," Appendix III of *Integer and Nonlinear Programming*, Abadie, J., ed., North-Holland, Amsterdam, 1970.

7 Stocker, D. C., *A Comparative Study of Nonlinear Programming Codes*, MS thesis, University of Texas at Austin, 1969.

8 Himmelblau, D. M., Chapter 9, *Applied Nonlinear Programming*, McGraw-Hill, New York, 1972.

9 Eason, E. D., *An Experimental Study of Numerical Methods for Design Optimization*, M. Eng. Project Report, University of Toronto, 1972.

10 Siddall, J. N., *"OPTISEP" Designers' Optimization Subroutines*, ME/71/DSN/REP1, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, 1971.

11 Hooke, R., and Jeeves, T. A., " 'Direct Search' Solution of Numerical and Statistical Problems," *Journal of the Association for Computing Machinery*, Vol. 8, No. 2, Apr. 1961, pp. 212–229.

12 Fletcher, R., and Powell, M. J. D., "A Rapidly Convergent Descent Method for Minimization," *Computer Journal*, Vol. 6, No. 2, July 1963, pp. 163–168.

13 Nelder, J. A., and Mead, R., "Simplex Method for Function Minimization," *Computer Journal*, Vol. 7, No. 4, Jan. 1965, pp. 308–313.

14 Rosenbrock, H. H., "An Automatic Method for Finding the Greatest or Least Value of a Function," *Computer Journal*, Vol. 3, No. 3, Oct. 1960, pp. 175–184.

15 WATLIB, University of Toronto Computer Center, Toronto 5, Ontario, Canada.

16 IBM System/360 Scientific Subroutine Package (SSP). 360A-CM-03X Version 3, 6th Edition, Mar. 1970.

17 MIT Information Processing Center, Applications Program Series AP-78 (APD-46), Cambridge, MA 02139, U.S.A.

18 Mischke, C. R., *An Introduction to Computer-Aided Design*, Prentice-Hall, Englewood Cliffs, N. J., 1968 (or Mechanical Engineering Department, Iowa State University, Ames, Iowa 50010, U.S.A.).

19 Whitney, D. E., Joint Mechanical and Civil Engineering Computing Facility, MIT, 77 Massachusetts Ave., Cambridge, MA 02139, U.S.A.

20 Beightler, C. S., Ta-Chen Lo, and Rylander, H. G., "Optimal Design by Geometric Programming," *Journal of Engineering for Industry*, TRANS ASME Series B, Vol. 92, No. 1, Feb. 1970, pp. 191–196.

21 Siddall, J. N., *Analytical Decision Making in Engineering Design*, Section 5.3, Prentice-Hall, Englewood Cliffs, N. J., 1972.