AD-A046 252 RICE UNIV HOUSTON TEX AERO-ASTRONAUTICS GROUP F/G 12/1
ON TESTING ALGORITHMS FOR MATHEMATICAL PROGRAMMING PROBLEMS.(U)
1976 A MIELE, S GONZALEZ, A K WU AFOSR-76-3075
UNCLASSIFIED AAR-134 AFOSR-TR-77-1248 NL

| OF |
AD
A046 252

END
DATE FILMED
12-77
DDC

AFOSR-TR- 77-1248

(3)

AERO-ASTRONAUTICS REPORT NO. 134

ON TESTING ALGORITHMS FOR MATHEMATICAL PROGRAMMING PROBLEMS

by

A. MIELE, S. GONZALEZ, and A.K. WU

RICE UNIVERSITY

1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (19) (18) AFOSR-TR-77-1248	2. GOVT ACCESSION NO. 3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) ON TESTING ALGORITHMS FOR MATHEMATICAL PROGRAMMING PROBLEMS.	5. TYPE OF REPORT & PERIOD COVERED Interim rept.,
	6. PERFORMING ORG. REPORT NUMBER (14) AAR [illegible]-134
7. AUTHOR(s) (10) A./Miele, S./Gonzalez and A.K./Wu	8. CONTRACT OR GRANT NUMBER(s) (15) AFOSR-76-3075, NSF-MCS-76-21657
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rice University Department of Mechanical Engineering Houston, Texas 77001	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) 61102F 2304/A3 (17) A3
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Bolling AFB, DC 20332	12. REPORT DATE (11) 1976
	13. NUMBER OF PAGES 50
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 53p.	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Numerical analysis, numerical methods, computing methods, computing techniques, complexity of computation, philosophy of computation, comparison of algorithms, computational speed, (OVER)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper considers the comparative evaluation of algorithms for mathematical programming problems. It is concerned with the measurement of computational speed and examines critically the concept of equivalent number of function evaluations N_e. (OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

402 169

UNCLASSIFIED LB

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19. KEY WORDS (continued)

measurement of computational speed, number of function evaluations, equivalent number of function evaluations, unconstrained minimization, mathematical programming.

20. ABSTRACT (continued)

Does this quantity constitute a fair way of comparing different algorithms?

The answer to the above question depends strongly on whether or not analytical expressions for the components of the gradient and the elements of the Hessian matrix are available. It also depends on the relative importance of the computational effort associated with algorithmic operations vis-a-vis the computational effort associated with function evaluations.

Both theoretical considerations and extensive numerical examples carried out in conjunction with the Fletcher-Reeves algorithm, the Davidon-Fletcher-Powell algorithm, and the quasilinearization algorithm suggest the following: the N_e concept, while accurate in some cases, has drawbacks in other cases; indeed, it might lead to a distorted view of the relative importance of an algorithm with respect to another.

The above distortion can be corrected through the introduction of a more general parameter $\tilde{N}_e$. This generalized parameter is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations.

(OVER)

On Testing Algorithms for Mathematical Programming Problems[1]

A. MIELE[2], S. GONZALEZ[3], and A.K. WU[4]

Abstract. This paper considers the comparative evaluation of algorithms for mathematical programming problems. It is concerned with the measurement of computational speed and examines critically the concept of equivalent number of function evaluations (N_e). Does this quantity constitute a fair way of comparing different algorithms? → next page

---

[1]This work was supported by the Office of Scientific Research, Office of Aerospace Research, United States Air Force, Grant No. AF-AFOSR-76-3075, and by the National Science Foundation, Grant No. MCS-76-21657. Partial support for the junior author was provided by CONACYT, Consejo Nacional de Ciencia y Tecnologia, Mexico City, Mexico.

[2]Professor of Astronautics and Mathematical Sciences, Rice University, Houston, Texas.

[3]Graduate Student, Department of Electrical Engineering, Rice University, Houston, Texas.

[4]Graduate Student, Department of Mechanical Engineering, Rice University, Houston, Texas.

cont. The answer to the above question depends strongly on whether or not analytical expressions for the components of the gradient and the elements of the Hessian matrix are available. It also depends on the relative importance of the computational effort associated with algorithmic operations vis-a-vis the computational effort associated with function evaluations.

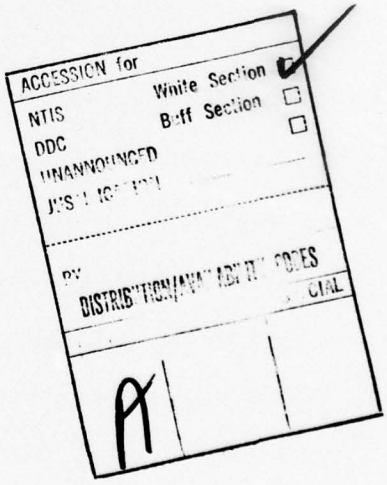
Both theoretical considerations and extensive numerical examples carried out in conjunction with the Fletcher-Reeves algorithm, the Davidon-Fletcher-Powell algorithm, and the quasilinearization algorithm suggest the following: the N_e concept, while accurate in some cases, has drawbacks in other cases; indeed, it might lead to a distorted view of the relative importance of an algorithm with respect to another.

The above distortion can be corrected through the introduction of a more general parameter $\tilde{N}_e$. This generalized parameter is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations.

From the analyses performed and the results obtained, it is inferred that, due to the weaknesses of the N_e concept, the use of the $\tilde{N}_e$ concept is advisable. In effect, this is the same as stating that, in spite of its obvious shortcomings,

the direct measurement of the CPU time is still the more reliable way of comparing different minimization algorithms.

Key Words. Numerical analysis, numerical methods, computing methods, computing techniques, complexity of computation, philosophy of computation, comparison of algorithms, computational speed, measurement of computational speed, number of function evaluations, equivalent number of function evaluations, unconstrained minimization, mathematical programming.



1. Introduction

Over the past two decades, a large number of mathematical programming problems have been studied both analytically and numerically. Generally speaking, these problems belong to four principal categories: (i) unconstrained minimization problems, (ii) constrained minimization problems involving equality constraints, (iii) constrained minimization problems involving inequality constraints, and (iv) constrained minimization problems involving both equality and inequality constraints.

For each category of problems, three types of methods have been developed, more specifically: (a) zeroth-order methods, (b) first-order methods, and (c) second-order methods. Methods of type (a) utilize only the functions under consideration and avoid the computation of derivatives. Methods of type (b) utilize the functions under consideration and their first derivatives. And methods of type (c) utilize the functions under consideration together with their first and second derivatives.

For each category of problems and each method, several classes of algorithms have been developed. As an example, with reference to the category of unconstrained minimization problems and first-order methods, the following classes of algorithms are available today: ordinary-gradient algorithm,

conjugate-gradient algorithm, variable-metric algorithm, memory-gradient algorithm, and supermemory-gradient algorithm.

It is clear that a bewildering combination of problems, methods, and algorithms exists and that the proliferation of these algorithms is bound to cause some confusion in the user, that is, the engineer, the chemist, or the economist who must solve problems of the real world. Faced with a given technical problem, the user would like to know an answer to the following question: what kind of algorithm should be selected to solve the problem under consideration?

Unfortunately, no clear-cut answer can be given to the above question. Nevertheless, the identification of potentially successful algorithms can be facilitated if the developer of an algorithm supplies sufficient information about the following items: (A) algorithm robustness or convergence range; (B) convergence rate; (C) computational speed; (D) memory requirements; and (E) programming complexity. Only for simply-structured problems (for instance, linear-quadratic problems), the above information can be predicted theoretically. For more general problems, the help of computer experimentation is nearly indispensable.

In this study, we are concerned with Item (C), the measurement of computational speed. In particular, we examine critically the concept of equivalent number of

function evaluations N_e and inquire whether this quantity constitutes a fair way of comparing different minimization algorithms. Then, we introduce a more general parameter $\tilde{N}_e$, which is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations. Next, we examine the N_e concept vis-a-vis the $\tilde{N}_e$ concept through several numerical examples. These examples include some widely used test functions (Rosenbrock, Wood, Powell, and Miele functions, generalized Rosenbrock function, and so on) and some widely used minimization algorithms (Fletcher-Reeves algorithm, Davidon-Fletcher-Powell algorithm, and quasilinearization algorithm).

Unconstrained Minimization. For the sake of simplicity, we consider in the following sections only one category of problems, namely, unconstrained minimization problems. More specifically, we consider the function

$$f = f(x) \quad , \tag{1}$$

where f is a scalar and x is an n-vector whose components are unconstrained. We denote by g(x) the gradient and by H(x) the Hessian. We observe that the gradient vector has n components and that the Hessian matrix has n^2 elements.

Of these n^2 elements, only

$$m = n(n+1)/2 \tag{2}$$

need to be calculated, owing to the symmetry of the Hessian matrix.

2. Measurement of Computational Speed

The methods employed for measuring the computational speed of different minimization algorithms can be grouped into two classes: (i) direct measurement and (ii) indirect measurement.

Direct Measurement. The most direct way for evaluating the computational speed of an algorithm is to measure the so called CPU time (the symbol CPU stands for central processing unit). The main advantage of this quantity is that it includes both function evaluation time and algorithmic time. The main disadvantage is that the CPU time is machine dependent as well as operator (programmer) dependent. The above difficulties can be removed to some degree if the comparison of different algorithms is done on a single computer, with the same programming language, with the same compiler with the same subroutines, under similar workload conditions of the computer, and by the same programmer. In other words, it is essential that the same experimental conditions be kept for all of the algorithms being investigated. For an example of comparative experiments done under these conditions, see Refs. 1-2.

Normalized Time. In an attempt to make the direct measurement of the CPU time independent of the particular computer, Colville introduced in Ref. 3 the concept of normalized time:

$$T_n = T/T_s \quad . \tag{3}$$

Here, T is the CPU time required to solve a particular test problem with the algorithm under consideration, and T_s is the CPU time required to execute a so-called standard program, devised by Colville. This standard program consists of inverting a 40x40 matrix ten times. Ideally, this parameter should be machine independent. In practice, it might still depend on the subroutines being used.

Indirect Measurement. There exist three major ways for evaluating the computational speed of an algorithm indirectly: (a) number of iterations; (b) number of function, gradient, and Hessian evaluations; and (c) equivalent number of function evaluations. All of these quantities are machine independent, operator independent, and simple to compute. However, their use implies the drawbacks discussed below.

Number of Iterations. Generally speaking, the time per iteration varies from one algorithm to another. Therefore, one cannot employ the number of iterations N as an indicator of computational speed, unless one can be reasonably sure that all of the algorithms being compared require approximately the same workload per iteration. For an example where this situation arises, see Ref. 4.

Number of Function, Gradient, and Hessian Evaluations. If one can be reasonably sure that the algorithmic time is

negligible by comparison with the function evaluation time, one can use the triplet composed of number of function evaluations (N_0), number of gradient evaluations (N_1), and number of Hessian evaluations (N_2) as a collective indicator of computational speed. The trouble is that the resulting indication is unclear, unless one is willing to attribute relative weights to function, gradient, and Hessian evaluations.

Equivalent Number of Function Evaluations. Let the relative weights (1,n,m) be attributed to the elements of the triplet (N_0, N_1, N_2). With this understanding, one can form the following linear combination:

$$N_e = N_0 + nN_1 + mN_2 \quad , \tag{4}$$

which is called the equivalent number of function evaluations. This parameter can be used as an indicator of computational speed providing the algorithmic time is negligible by comparison with the function evaluation time and providing the weights (1,n,m) measure correctly the relative importance of function evaluation, gradient evaluation, and Hessian evaluation. However, this depends on whether analytical expressions for the components of the gradient and the elements of the Hessian matrix are available or not.

3. Standard Definition of Equivalent Number of Function Evaluations

Assume that some particular algorithm is employed in order to obtain the minimum of the function (1) on a digital computer. The total CPU time T can be written as

$$T = T_a + T_e \quad . \tag{5}$$

Here, T_a is the algorithmic time, namely, the CPU time required to perform the arithmetic operations intrinsic to the algorithm being employed. And T_e is the function evaluation time, namely, the CPU time required to evaluate the function, the gradient, and the Hessian. Therefore, T_e can be written as

$$T_e = T_0 + T_1 + T_2 \quad . \tag{6}$$

Here, T_0 denotes the CPU time associated with function evaluations, T_1 denotes the CPU time associated with gradient evaluations, and T_2 denotes the CPU time associated with Hessian evaluations.

Let τ_0, τ_1, τ_2 denote the basic times required to compute one function, one gradient, and one Hessian, respectively. Observe that the following relations hold:

$$T_0 = \tau_0 N_0 \quad , \quad T_1 = \tau_1 N_1 \quad , \quad T_2 = \tau_2 N_2 \quad . \tag{7}$$

As a consequence, Eq. (5) can be rewritten as

$$T = T_a + \tau_0 N_0 + \tau_1 N_1 + \tau_2 N_2 \quad . \tag{8}$$

Next, let the following assumptions be employed:

(A1) From the point of view of the CPU time, one gradient evaluation is equivalent to n function evaluations.

(A2) From the point of view of the CPU time, one Hessian evaluation is equivalent to m function evaluations.

(A3) The algorithmic time is negligible by comparison with the function evaluation time.

In equation form, the above assumption can be rewritten as follows:

(A1) $$\tau_1 = n\tau_0 \quad , \tag{9-1}$$

(A2) $$\tau_2 = m\tau_0 \quad , \tag{9-2}$$

(A3) $$T_a << T_e \quad . \tag{9-3}$$

As a consequence, Eq. (8) can be rewritten as

$$T = \tau_0 N_e \quad , \tag{10}$$

where

$$N_e = N_0 + nN_1 + mN_2 \quad . \tag{11}$$

This is the standard definition of equivalent number of function evaluations for second-order methods, which reduces to

$$N_e = N_0 + nN_1 \tag{12}$$

for first-order methods ($N_2 = 0$), and to

$$N_e = N_0 \tag{13}$$

for zeroth-order methods ($N_1 = N_2 = 0$).

4. Cases Where the Assumptions are Satisfied

In this section, we present some cases where the equivalent number of function evaluations (11) can be regarded to be a correct indicator of computational speed.

Example 4.1. Suppose that the function (1) is such that analytical expressions for the components of the gradient and the elements of the Hessian matrix are not available. Consequently, some numerical approximation scheme to the gradient and the Hessian is necessary. For example, suppose that a forward difference scheme is employed. Denote by ε some small number, and denote by u_i a unit vector in the x_i- direction. Then, the following relations hold:

$$f(x + \varepsilon u_i) - f(x) = \varepsilon u_i^T g(x) = \varepsilon g_i(x) \quad , \tag{14}$$

where $i = 1,2,\ldots,n$, and

$$f(x+\varepsilon u_i+\varepsilon u_j)-f(x+\varepsilon u_i)-f(x+\varepsilon u_j)+f(x) = \varepsilon^2 u_i^T H(x) u_j = \varepsilon^2 H_{ij}(x), \tag{15}$$

where $i = 1,2,\ldots,n$ and $j = i,i+1,\ldots,n$.

If $f(x)$ is known, Eq. (14) shows that the computation of the gradient $g(x)$ requires n additional function evaluations. In turn, if $f(x)$ and $g(x)$ are known, Eq. (15) shows that the computation of the Hessian $H(x)$ requires m additional function

evaluations. Clearly, Eqs. (14) and (15) illustrate the validity of Assumptions (A1) and (A2). Then, providing one can ascertain that Assumption (A3) is true, one concludes that the parameter (11) is a correct indicator of computational speed.

Example 4.2. Suppose that the function f depends on x, not directly, but indirectly through some variable y, which is a function of x defined by means of some definite integral. For simplicity, assume that both x and y are n-vectors. Then, the situation is as follows:

$$f = f(y) \quad , \tag{16}$$

where

$$y = \int_0^1 \phi(x,t)dt \quad . \tag{17}$$

Denote by

$$F(x) = f(y(x)) \tag{18}$$

the function obtained by combining (16)-(17) and eliminating y. Observe that

$$F_x = y_x f_y \quad , \quad F_{xx} = y_{xx} f_y + y_x f_{yy} y_x^T \quad , \tag{19}$$

where the square matrix y_x and the cubic array y_{xx} are given by

$$y_x = \int_0^1 \phi_x(x,t)\,dt \quad , \quad y_{xx} = \int_0^1 \phi_{xx}(x,t)\,dt \quad . \tag{20}$$

Next, assume that some particular algorithm is employed in order to find the minimum of the function F(x), utilizing the gradient (19-1) and perhaps the Hessian (19-2). Since the computation of F, F_x, F_{xx} requires previous numerical integrations, defined through (17) and (20), this example illustrates a situation where the validity of Assumption (A3) is plausible. Then, providing one can ascertain that Assumptions (A1) and (A2) are true, one concludes that the parameter (11) is a correct indicator of computational speed.

5. Cases Where the Assumptions Are Not Satisfied

In this section, we present some cases where the equivalent number of function evaluations (11) cannot be regarded to be a correct indicator of computational speed.

Example 5.1. Suppose that the function (1) is such that analytical expressions for the components of the gradient and the elements of the Hessian matrix are available. In particular, assume that the function (1) has the quadratic form

$$f(x) = a + b^T x + \frac{1}{2} x^T c x \quad , \tag{21}$$

with the implication that

$$g(x) = b + cx \quad , \quad H(x) = c \quad . \tag{22}$$

In (21)-(22), a,b,c are constants having appropriate dimensions.

Next, consider the operational count associated with function, gradient, and Hessian evaluations. Observe that the computation of the function requires $(n+1)^2$ multiplications and $n(n+1)$ sums and that the computation of the gradient requires n^2 multiplications and n^2 sums. If we neglect the addition times by comparison with the multiplication times, we arrive at the following conclusions for the ratios

of the basic times τ_0, τ_1, τ_2 :

$$\tau_1/\tau_0 = n^2/(n+1)^2 \quad , \quad \tau_2/\tau_0 = 0 \quad . \tag{23}$$

Hence, for n relatively large, we have

$$\tau_1/\tau_0 = 1 \quad , \quad \tau_2/\tau_0 = 0 \quad . \tag{24}$$

Therefore, it appears that, from the point of view of the CPU time, one gradient evaluation is equivalent to one (not n) function evaluation, and one Hessian evaluation is equivalent to zero (not m) function evaluations. As a conclusion, it appears that Assumptions (A1) and (A2) are not justified for the quadratic function (21).

Remark. The result (24-1) represents a worst-case condition, because function evaluation and gradient evaluation have been regarded to be separate operations. Had one accounted for the commonality of the product cx to both f(x) and g(x), then Eqs. (24) would have been modified as follows:

$$\tau_1/\tau_0 = 0 \quad , \quad \tau_2/\tau_0 = 0 \quad , \tag{25}$$

thereby invalidating Assumptions (A1) and (A2) to an even larger degree.

Example 5.2. Suppose that the function (1) is such that the Hessian matrix has a banded structure. For example, consider the following generalized Rosenbrock function (see Oren, Ref. 5):

$$f(x) = \sum_{i=1}^{n-1} (x_i-1)^2 + 100 \sum_{i=1}^{n-1} (x_i^2 - x_{i+1})^2 \quad , \tag{26}$$

and observe that the associated Hessian matrix is tridiagonal. Therefore, only those elements H_{ij} which are located on the principal diagonal and on a contiguous subdiagonal need to be computed, since all of the remaining elements vanish. The number of nonzero elements of the Hessian matrix that need to be computed is

$$\tilde{m} = 2n-1 \quad , \tag{27}$$

instead of m. Table 1 shows the values of m, $\tilde{m}$, and $\tilde{m}/m$ for n ranging between 5 and 30. Note that the ratio $\tilde{m}/m$ decreases as n increases and becomes of order 1/10 for n=30. Therefore, even if finite-difference methods are employed in the computation of the Hessian matrix, it appears that Assumption (A2) is not justified for the generalized Rosenbrock function (26).

Table 1. Generalized Rosenbrock function.

n	$\tilde{m}$	m	$\tilde{m}/m$
5	9	15	0.600
10	19	55	0.345
15	29	120	0.242
20	39	210	0.186
25	49	325	0.151
30	59	465	0.127

6. New Definition of Equivalent Number of Function Evaluations

From the examples of the previous section , it appears that there are cases where Eq. (11) cannot be regarded as representative of the computational speed of an algorithm, in that it might overestimate the importance of the gradient contribution and the Hessian contribution to the equivalent number of function evaluations. In addition, Eq. (11) disregards the contribution due to algorithmic operations.

In an attempt to correct the above situation, we supply here a new definition of equivalent number of function evaluations. Specifically, we introduce a more general parameter $\tilde{N}_e$, which is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations.

While we retain Eqs. (5)-(8), we replace the assumptions expressed by Eqs. (9) with the following definitions:

(B1) $$\tau_1 = C_1 \tau_0 \quad , \tag{28-1}$$

(B2) $$\tau_2 = C_2 \tau_0 \quad , \tag{28-2}$$

(B3) $$T_a = C_3 T_e \quad . \tag{28-3}$$

Here, C_1, C_2, C_3 are coefficients to be determined experimentally

or theoretically through an operational count. With this understanding, Eq. (8) can be rewritten as

$$T = \tau_0 \tilde{N}_e \quad , \tag{29}$$

where

$$\tilde{N}_e = (1 + C_3)\ (N_0 + C_1 N_1 + C_2 N_2) \quad . \tag{30}$$

This expression constitutes a new definition of equivalent number of functions evaluations.

Alternative Definition. If one introduces the new coefficients

$$K_1 = C_1/n \quad , \quad K_2 = C_2/m \quad , \quad K_3 = 1+C_3 \quad , \tag{31}$$

Eq. (30) can be rewritten in the alternative form

$$\tilde{N}_e = K_3(N_0 + K_1 n N_1 + K_2 m N_2) \quad . \tag{32}$$

Remark. With the terminology of this section, Assumptions (A1), (A2), (A3) can now be restated as follows:

$$\text{(A1)} \qquad C_1 = n \quad \text{or} \quad K_1 = 1 \quad , \tag{33-1}$$

(A2) $C_2 = m$ or $K_2 = 1$, (33-2)

(A3) $C_3 = 0$ or $K_3 = 1$. (33-3)

Comment. The coefficients C_1 and C_2 measure the relative importance of the computational effort associated with gradient evaluation and Hessian evaluation vis-a-vis the computational effort associated with function evaluation. The coefficient C_3 measures the relative importance of the computational effort associated with algorithmic operations vis-a-vis the computational effort associated with function, gradient, and Hessian evaluations. While the coefficients C_1 and C_2 depend on the nature of the function $f(x)$, the coefficient C_3 depends also on the structure of the particular algorithm and search technique employed.

The above coefficients can be determined through either an operational count or computer experimentation. By determining the triplet (C_1, C_2, C_3) and the associated triplet (K_1, K_2, K_3), and by measuring the deviation of these coefficients from the idealized values (33), one can supply an answer to the basic questions formulated in this paper, namely, those concerning the correctness of Assumptions (A1), (A2), (A3).

7. Experimental Conditions and Test Functions

In the following sections, we describe some numerical experiments, leading to the computation of the coefficients C_i and K_i for several test functions and minimization algorithms.

Test Conditions. All computations were performed on the IBM 370/155 computer of Rice University. FORTRAN programming was employed in conjunction with double-precision arithmetic. A FORTRAN G1 compiler was used. A FORTRAN TIME subroutine was used in order to determine the CPU times. Note that the IBM 370/155 computer of Rice University has multiprogramming and time-sharing capabilities.

Test Functions. Fourteen test functions were employed. They are described below in the order of increasing dimension.

Example 7.1, Rosenbrock, n=2:

$$f = (x_1-1)^2 + 100(x_1^2-x_2)^2 \quad ; \tag{34}$$

Example 7.2, Himmelblau, n=2:

$$f = (x_1^2+x_2-11)^2 + (x_1+x_2^2-7)^2 \quad ; \tag{35}$$

Example 7.3, Beale, n=2:

$$f = [1.5-x_1(1-x_2)]^2 + [2.25 - x_1(1-x_2^2)]^2$$

$$+ [2.625-x_1(1-x_2^3)]^2 \quad ; \tag{36}$$

Example 7.4, trigonometric-exponential function, n=3:

$$f = \cos[\exp(x_1-x_2+2/3)] + (x_2-1)^4 + (x_3-2x_2)^2 \quad ; \tag{37}$$

Example 7.5, helical valley function, n=3:

$$f = 100[x_3-(10/2\pi)\arctan(x_2/x_1)]^2$$

$$+ 100[\sqrt{(x_1^2+x_2^2)}-1]^2 + x_3^2 \quad ; \tag{38}$$

Example 7.6, bowl function, n=3:

$$f = 1-10\exp[-(x-\pi u)^T M(x-\pi u)] \quad , \tag{39-1}$$

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{bmatrix} \quad , \qquad u = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad ; \tag{39-2}$$

Example 7.7, Wood, n=4:

$$f = (x_1-1)^2 + 100(x_1^2-x_2)^2 + (x_3-1)^2 + 90(x_3^2-x_4)^2$$

$$+ 10.1[(x_2-1)^2 + (x_4-1)^2] + 19.8(x_2-1)(x_4-1) \quad ; \quad (40)$$

Example 7.8, Jacobson, n=4:

$$f = [1/4+b^Tx+(1/2)x^TMx]^2 \quad , \quad (41\text{-}1)$$

$$M = \begin{bmatrix} 4.5 & 7.0 & 3.5 & 3.0 \\ 7.0 & 14.0 & 9.0 & 8.0 \\ 3.5 & 9.0 & 8.5 & 5.0 \\ 3.0 & 8.0 & 5.0 & 8.0 \end{bmatrix} \quad , \quad b = \begin{bmatrix} -0.5 \\ -1.0 \\ -1.5 \\ 0.0 \end{bmatrix} \quad ; \quad (41\text{-}2)$$

Example 7.9, Powell, n=4:

$$f = (x_1+10x_2)^4 + 5(x_3-x_4)^4 + (x_2-2x_3)^4$$

$$+ 10(x_1-x_4)^4 \quad ; \quad (42)$$

Example 7.10, Powell, n=4:

$$f = (x_1+10x_2)^2 + 5(x_3-x_4)^2 + (x_2-2x_3)^4$$

$$+ 10(x_1-x_4)^4 \quad ; \quad (43)$$

Example 7.11, Miele, n=4:

$$f = (\exp x_1 - x_2)^4 + 100(x_2 - x_3)^6 + \tan^4(x_3 - x_4) + x_1^8 + (x_4 - 1)^2 \quad ; \tag{44}$$

Example 7.12, quadratic function, n=4:

$$f = (x_1 + x_2 + 0.5x_4)^2 + (x_1 + 2x_2 + x_3 + x_4)^2 + (x_2 + x_3 + 1.5x_4)^2 + (0.5x_1 + x_2 + 1.5x_3 - 0.5)^2 \quad ; \tag{45}$$

Example 7.13, Bass, n=5 through 30:

$$f = \sum_{i=1}^{n} x_i^2 + \left[\sum_{i=1}^{n} (i)x_i\right]^2 + \left[\sum_{i=1}^{n} (i)x_i\right]^4 \quad ; \tag{46}$$

Example 7.14, generalized Rosenbrock function (Oren), n = 5 through 30:

$$f = \sum_{i=1}^{n-1} (x_i - 1)^2 + 100 \sum_{i=1}^{n-1} (x_i^2 - x_{i+1})^2 \quad . \tag{47}$$

8. Minimization Algorithms

Three unconstrained minimization algorithms were employed: The Fletcher-Reeves algorithm (FR), the Davidon-Fletcher-Powell algorithm (DFP), and the quasilinearization algorithm (QL). The FR and DFP algorithms exemplify first-order methods, and the QL algorithm exemplifies second-order methods.

For all of these algorithms, let x denote the nominal point, $\tilde{x}$ the varied point, $\hat{x}$ the previous point, p the search direction (an n-vector), and α the stepsize (a scalar). Then, the step leading from x to $\tilde{x}$ is given by

$$\tilde{x} = x-\alpha p \quad . \tag{48}$$

The specification of the search direction and the stepsize is given below.

Search Direction. For the FR algorithm, the computation of the search direction is described by the relations

$$p = g(x)+\gamma\hat{p} \quad , \tag{49-1}$$

$$\gamma = g^T(x)g(x)/g^T(\hat{x})g(\hat{x}) \quad . \tag{49-2}$$

For the first iteration, one sets

$$\gamma = 0 \quad . \tag{49-3}$$

For the DFP algorithm, let M denote a symmetric and positive-definite matrix (an approximation to the inverse of the Hessian matrix). Then, the computation of the search direction is described by

$$p = Mg(x) \quad , \tag{50-1}$$

$$M = \hat{M}+A-B \quad , \tag{50-2}$$

$$A = zz^T/y^Tz \quad , \tag{50-3}$$

$$B = \hat{M}yy^T\hat{M}/y^T\hat{M}y \quad , \tag{50-4}$$

$$y = g(x)-g(\hat{x}) \quad , \tag{50-5}$$

$$z = x-\hat{x} \quad . \tag{50-6}$$

For the first iteration, one sets

$$M = I \quad , \tag{50-7}$$

where I is the identity matrix of order n.

For the QL algorithm, the computation of the search direction is described by

$$H(x)p = g(x) \quad . \tag{51}$$

The solution of the above linear system is done by Gaussian elimination without pivoting.

Search Technique. For the FR and DFP algorithms, one-step cubic interpolation is employed in order to determine the optimum stepsize α_o . In the cubic interpolation process, two ordinates and two slopes are employed. Therefore, it is assumed that one iteration of each of these algorithms requires two function evaluations and two gradient evaluations. The details are given below.

Let $\tilde{f}(\alpha)$ and $\tilde{f}_\alpha(\alpha)$ denote the functions

$$\tilde{f}(\alpha) = f(\tilde{x}) = f(x-\alpha p) \quad , \tag{52-1}$$

$$\tilde{f}_\alpha(\alpha) = g^T(\tilde{x})p = g^T(x-\alpha p)p \quad . \tag{52-2}$$

Suppose that two stepsizes α_1 and α_2 have been found such that

$$\tilde{f}_\alpha(\alpha_1) < 0 \quad , \quad \tilde{f}_\alpha(\alpha_2) > 0 \quad . \tag{53}$$

Then, the optimal stepsize α_o is determined through the relation (Ref. 6)

$$\alpha_o = \alpha_1 + \beta_o(\alpha_2 - \alpha_1) \quad , \tag{54}$$

where

$$\beta_o = (1/3q_3)\left[-q_2 + \sqrt{(q_2^2 - 3q_1q_3)} \right] \quad , \tag{55}$$

with

$$q_1 = (\alpha_2 - \alpha_1)\tilde{f}_\alpha(\alpha_1) \quad , \tag{56-1}$$

$$q_2 = 3\left[\tilde{f}(\alpha_2) - \tilde{f}(\alpha_1) \right] - (\alpha_2 - \alpha_1)\left[2\tilde{f}_\alpha(\alpha_1) + \tilde{f}_\alpha(\alpha_2) \right] , \tag{56-2}$$

$$q_3 = 2\left[\tilde{f}(\alpha_1) - \tilde{f}(\alpha_2) \right] + (\alpha_2 - \alpha_1)\left[\tilde{f}_\alpha(\alpha_1) + \tilde{f}_\alpha(\alpha_2) \right] . \tag{56-3}$$

In the sequel, we assume that the stepsizes

$$\alpha_1 = 0 \quad , \quad \alpha_2 = 1 \tag{57}$$

are such that Ineqs. (53) are satisfied.

For the QL algorithm, the stepsize

$$\alpha = 1 \tag{58}$$

is employed. Therefore, it is assumed that one iteration requires one function evaluation, one gradient evaluation, and one Hessian evaluation.

9. Experimental Determination of the Coefficients

In this section, we describe the technique employed to determine the coefficients C_1, C_2, C_3 experimentally.

Coefficients C_1 and C_2. These coefficients can be computed with the aid of Eqs. (28-1) and (28-2) as follows:

$$C_1 = \tau_1/\tau_0 \quad , \quad C_2 = \tau_2/\tau_0 \quad . \tag{59}$$

For the basic times τ_0, τ_1, τ_2 to be sufficiently precise, it is necessary that the computation of f(x), g(x), H(x) be repeated a large number of times at the same nominal point x (for instance, 1000 times).

With the above considerations in mind, the basic times τ_0, τ_1, τ_2 were determined as follows: (i) by evaluating separately the function, the gradient, and the Hessian 1000 times; (ii) by monitoring the associated CPU times, which include the do-loop time; (iii) by determining separately the do-loop time; and (iv) by subtracting the do-loop time from the experimentally determined CPU times.

With the basic times τ_0, τ_1, τ_2 known, the coefficients C_1 and C_2 can be computed with (59). Then, the coefficients K_1 and K_2 can be determined with (31-1) and (31-2).

Coefficient C_3. This coefficient can be computed with the aid of Eqs. (5), (6), and (28-3) as follows:

$$C_3 = (T-T_0-T_1-T_2)/(T_0+T_1+T_2) \quad . \tag{60}$$

Next, we invoke Eqs. (7) and the definition

$$T = \tau N \quad , \tag{61}$$

where N denotes the number of iterations and τ denotes the time required to perform one iteration (this includes both the algorithmic time and the function evaluation time). In the light of (7) and (61), Eq. (60) can be rewritten as

$$C_3 = (\tau N-\tau_0 N_0-\tau_1 N_1-\tau_2 N_2)/(\tau_0 N_0+\tau_1 N_1+\tau_2 N_2) \quad . \tag{62}$$

Examination of one iteration of the algorithms under consideration shows that, for the search conditions assumed,

$$N_0/N = 2 \quad , \quad N_1/N = 2 \quad , \quad N_2/N = 0 \tag{63}$$

for the FR and DFP algorithms and that

$$N_0/N = 1 \quad , \quad N_1/N = 1 \quad , \quad N_2/N = 1 \tag{64}$$

for the QL algorithm. As a consequence, Eq. (62) simplifies to

$$C_3 = (\tau-2\tau_0-2\tau_1)/(2\tau_0+2\tau_1) \tag{65}$$

for the FR and DFP algorithms, and to

$$C_3 = (\tau-\tau_0-\tau_1-\tau_2)/(\tau_0+\tau_1+\tau_2) \qquad (66)$$

for the QL algorithm. For the basic time τ to be sufficiently precise, it is necessary that one iteration of each of the algorithms under consideration be repeated a large number of times at the same nominal point x (for instance, 1000 times).

With the above considerations in mind, the basic time τ was determined as follows: (i) by executing one iteration of each of the algorithms under consideration 1000 times; (ii) by monitoring the associated CPU times, which include the do-loop time; (iii) by determining separately the do-loop time; and (iv) by subtracting the do-loop time from the experimentally determined CPU times.

With the basic time τ known, and with τ_0, τ_1, τ_2 also known, the coefficient C_3 can be computed with (65) for the FR and DFP algorithms and with (66) for the QL algorithm. Then, the coefficient K_3 can be determined with (31-3).

10. Numerical Results

Careful numerical experiments were performed along the lines outlined in Sections 7-9, and the results are given in Tables 2-5. Tables 2-3 give the coefficients C_1, C_2 and K_1, K_2 for the test functions (34)-(47). Tables 4-5 give the coefficient C_3 for the test functions (34)-(47) and the FR, DFP, and QL algorithms. The coefficient K_3 is not given, since it can be computed with the simple relation (31-3).

Coefficient C_1. Table 2 shows that the coefficient C_1 for the Rosenbrock function has the value 0.90 (instead of n=2), and the coefficient C_1 for the Powell function has the value 1.17 (instead of n=4). Thereby, use of the standard definition (11) of equivalent number of function evaluations overestimates the effort associated with gradient computation by a factor of 2.2 for the Rosenbrock function and by a factor of 3.4 for the Powell function.

With reference to the generalized Rosenbrock function, Table 3 shows that, for the case n=5, the coefficient C_1 has the value 1.05 (instead of 5); for the case n=30, the coefficient C_1 has the value 1.11 (instead of 30). Thereby, use of the standard definition (11) of equivalent number of function evaluations overestimates the effort associated with gradient computation by a factor of 4.8 for n=5 variables and by a factor of 27 for n=30 variables.

In conclusion, under the hypothesis that analytical expressions for the components of the gradient are available, and for Examples 7.1 through 7.14, it does not appear that Assumption (A1) is satisfied.

Coefficient C_2. Table 2 shows that the coefficient C_2 for the Rosenbrock function has the value 1.03 (instead of m=3), and the coefficient C_2 for the Powell function has the value 0.92 (instead of m=10). Thereby, use of the standard definition (11) of equivalent number of function evaluations overestimates the effort associated with Hessian computation by a factor of 2.9 for the Rosenbrock function and by a factor of 11 for the Powell function.

With reference to the generalized Rosenbrock function, Table 3 shows that, for the case n=5, the coefficient C_2 has the value 1.47 (instead of m=15); for the case n=30, the coefficient C_2 has the value 3.68 (instead of m=465). Thereby, use of the standard definition (11) of equivalent number of function evaluations overestimates the effort associated with Hessian computation by a factor of 10 for n=5 variables and by a factor of 126 for n=30 variables.

In conclusion, under the hypothesis that analytical expressions for the elements of the Hessian matrix are available, and for Examples 7.1 through 7.14, it does not appear that Assumption (A2) is satisfied.

Coefficient C_3. Inspection of Tables 4-5 shows that the coefficient C_3 is never negligible by comparison with 1, meaning that the algorithmic time is never negligible by comparison with the function evaluation time. Indeed, in many cases, C_3 can be larger than 1, meaning that the algorithmic time can be larger than the function evaluation time.

For example, consider the Wood function. Table 4 shows that the coefficient C_3 has the value 0.87 for the FR algorithm, 2.88 for the DFP algorithm, and 2.12 for the QL algorithm.

As another example, consider the generalized Rosenbrock function. Table 5 shows that, for values of n ranging between 5 and 30, the coefficient C_3 ranges between 0.55 and 0.86 for the FR algorithm, between 3.33 and 12.92 for the DFP algorithm, and between 2.24 and 23.85 for the QL algorithm.

In conclusion, under the hypothesis that analytical expressions for the components of the gradient and the elements of the Hessian matrix are available, and for Examples 7.1 through 7.14, it does not appear that Assumption (A3) is satisfied for the FR, DFP, and QL algorithms.

Comments. With reference to the previous results, the following comments are pertinent.

(i) In devising the subroutines necessary to the computation of the functions (34)-(47) and their first and second derivatives, an effort was made to render the CPU time as small as possible. The commonality existing between the components of the gradient was exploited. The commonality existing between the elements of the Hessian matrix being computed was also exploited. Finally, the fact that the Hessian matrix might have many elements which vanish was taken into consideration.

(ii) For the subroutines mentioned in (i), the computation of the function, the computation of the gradient, and the computation of the Hessian matrix were conceived to be separate operations. No attempt was made to exploit the commonality between function and gradient nor the commonality between function, gradient, and Hessian matrix. In other words, a worst-case situation was postulated, and conservative estimates of C_1, C_2, C_3 were arrived at.[5] Had the above commonality been exploited, the values of C_1, C_2 would have been smaller than those given in Tables 2-3 and the values of C_3 would have been larger than those given in Tables 4-5.

---

[5] Here, the adjective conservative is employed in the sense of "favorable" to the old definition (11) of equivalent number of function evaluations.

(iii) For the generalized Rosenbrock function, Table 5 illustrates in a striking way the effect of the size of the problem on the relative importance of algorithmic time vis-a-vis function evaluation time. As n increases, C_3 tends to be constant for the FR algorithm; it increases at a linear rate for the DFP algorithm; and it increases at a faster-than-linear rate for the QL algorithm.

The explanation for this result is simple. As n increases, the function evaluation time increases linearly for all of the algorithms under consideration.[6] On the other hand, as n increases, the algorithmic time increases linearly for the FR algorithm, quadratically for the DFP algorithm, and cubically for the QL algorithm.

(iv) The coefficients C_1 and C_2 depend mostly on the nature of the function f(x). On the other hand, the coefficient C_3 for the FR and DFP algorithms depends also on the search technique employed to determine the stepsize α; should a different search technique be employed, the value of C_3 would change. By the same token, the coefficient C_3 for QL algorithm depends on the subroutine used to solve the linear system governing the components of the search direction; should a different subroutine be employed, the value of C_3 would change.

---

[6] For the QL algorithm, the linear increase of the function evaluation time is due to the tridiagonal structure of the Hessian matrix associated with the generalized Rosenbrock function.

Table 2. Results for Examples 7.1 through 7.12.

Example	n	m	C_1	C_2	K_1	K_2
7.1	2	3	0.90	1.03	0.45	0.34
7.2	2	3	1.41	1.34	0.71	0.45
7.3	2	3	1.42	1.85	0.71	0.62
7.4	3	6	1.10	1.45	0.37	0.24
7.5	3	6	1.18	1.63	0.39	0.27
7.6	3	6	1.13	1.69	0.38	0.28
7.7	4	10	1.06	0.83	0.26	0.08
7.8	4	10	1.10	2.02	0.27	0.20
7.9	4	10	1.27	1.32	0.32	0.13
7.10	4	10	1.17	0.92	0.29	0.09
7.11	4	10	1.50	1.50	0.38	0.15
7.12	4	10	1.13	0.32	0.28	0.03

Table 3. Results for Examples 7.13 and 7.14.

Example	n	m	C_1	C_2	K_1	K_2
7.13	5	15	1.90	2.94	0.38	0.20
7.13	10	55	1.99	4.89	0.20	0.09
7.13	15	120	2.05	7.05	0.14	0.06
7.13	20	210	2.18	9.36	0.11	0.04
7.13	25	325	2.17	11.78	0.09	0.04
7.13	30	465	2.06	12.92	0.07	0.03
7.14	5	15	1.05	1.47	0.21	0.10
7.14	10	55	1.09	1.86	0.11	0.03
7.14	15	120	1.09	2.38	0.07	0.02
7.14	20	210	1.12	2.81	0.06	0.01
7.14	25	325	1.09	3.11	0.04	<0.01
7.14	30	465	1.11	3.68	0.04	<0.01

Table 4. Results for Examples 7.1 through 7.12.

Algorithm			FR	DFP	QL
Example	n	m	C_3	C_3	C_3
7.1	2	3	1.65	2.51	1.17
7.2	2	3	1.15	1.97	0.66
7.3	2	3	0.65	1.41	0.51
7.4	3	6	0.27	0.77	0.37
7.5	3	6	0.28	0.78	0.37
7.6	3	6	0.30	0.77	0.40
7.7	4	10	0.87	2.88	2.12
7.8	4	10	0.37	1.00	0.64
7.9	4	10	1.02	3.31	1.94
7.10	4	10	1.13	3.42	2.14
7.11	4	10	0.36	1.04	0.57
7.12	4	10	0.99	3.08	2.58

Table 5. Results for Examples 7.13 and 7.14.

Algorithm			FR	DFP	QL
Example	n	m	C_3	C_3	C_3
7.13	5	15	0.48	1.80	1.03
7.13	10	55	0.42	3.14	2.26
7.13	15	120	0.41	4.27	3.76
7.13	20	210	0.37	5.72	4.82
7.13	25	325	0.39	6.90	6.59
7.13	30	465	0.41	8.06	8.28
7.14	5	15	0.86	3.33	2.24
7.14	10	55	0.72	4.92	5.08
7.14	15	120	0.58	6.93	8.82
7.14	20	210	0.66	8.82	13.15
7.14	25	325	0.55	10.89	18.96
7.14	30	465	0.60	12.92	23.85

11. Conclusions and Recommendations

In this paper, we have considered the comparative evaluation of algorithms for mathematical programming problems from the point of view of computational speed. We have examined critically the indirect measurement of computational speed through the equivalent number of function evaluations N_e, and we have found that the N_e concept, while accurate in some cases, has drawbacks in other cases. Indeed, it might lead to a distorted view of the relative importance of an algorithm with respect to another.

In an effort to correct the above distortion, we have imbedded the parameter N_e into a more general parameter $\tilde{N}_e$, which is constructed so as to reflect accurately the computational effort associated with function evaluations and algorithmic operations. This new parameter $\tilde{N}_e$ includes coefficients C_1, C_2, C_3 which can be determined either experimentally or through an operational count. When the triplet (C_1, C_2, C_3) takes on the values $(n,m,0)$, the new parameter $\tilde{N}_e$ reduces to the old parameter N_e.

We have determined experimentally the coefficients C_1, C_2, C_3 for fourteen test functions and three minimization algorithms. And we have found that the deviations of these coefficients from the idealized values n,m,0 can be substantial.

More specifically, the experimental values found for C_1 are 1.4 to 27 times smaller than the idealized value n. The experimental values found for C_2 are 1.6 to 126 times smaller than the idealized value m. And the experimental values found for C_3 are never negligible with respect to 1; they are of order 1 for the FR algorithm, and of order 1 to 10 for the DFP and QL algorithms.

Obviously, the experimental values of C_1, C_2, C_3 are subject to errors due, among other things, to the multiprogramming and time-sharing capabilities of the IBM 370/155 computer of Rice University. However, the basic fact remains that the deviations detected for C_1, C_2, C_3 from the idealized values n,m,0 are so large that the use of the N_e concept is open to serious question.

From the analyses performed and the results obtained, it is inferred that, due to the weakness of the N_e concept, the use of the $\tilde{N}_e$ concept is advisable as a means for comparing different algorithms from the point of view of computational speed. In effect, this is the same as stating that, in spite of its obvious shortcomings, the direct measurement of the CPU time is still the most reliable way of comparing different minimization algorithms.

However, for the direct measurement of the CPU time to be really meaningful, provisions similar to those implemented

in Refs. 1-2 should be employed. That is, it is necessary that the comparison of different algorithms be done on a single computer, with the same programming language, with the same compiler, with the same subroutines, under similar workload conditions of the computer, and by the same programmer.

In closing, these authors stress that the conclusions of this paper should not be interpreted as an invitation to other authors to disregard reporting on number of iterations N, number of function evaluations N_0, number of gradient evaluations N_1, and number of Hessian evaluations N_2. By all means, these are useful quantities, which should be reported because their knowledge does shed some light on the comparative behavior of different algorithms. Nevertheless, none of the methods for the indirect measurement of the computational speed is truly satisfactory, and these authors feel that there exists no reliable alternative to the direct measurement of the CPU time.

References

1. LEVY, A.V., and MONTALVO, A., Comparison of Multiplier and Quasilinearization Methods, I&EC Process Design and Development, Vol. 14, No. 4, 1975.

2. LEVY, A.V., and GUERRA, V., On the Optimization of Constrained Functions: Comparison of Sequential Gradient-Restoration Algorithm and Gradient-Projection Algorithm, Applied Mathematics and Computation, Vol. 2, No. 3, 1976.

3. COLVILLE, A.R., A Comparative Study of Nonlinear Programming Codes, IBM, New York Scientific Center, Yorktown Neights, New York, Technical Report No. 320-2949, 1968.

4. MIELE, A., TIETZE, J.L., and LEVY, A.V., Comparison of Several Gradient Algorithms for Mathematical Programming Problems, Omaggio a Carlo Ferrari, Edited by G. Jarre, Libreria Editrice Universitaria Levrotto e Bella, Torino, Italy, 1974.

5. OREN, S.S., On the Selection of Parameters in Self-Scaling, Variable-Metric Algorithms, Mathematical Programming, Vol. 7, No. 3, 1974.

6. MIELE, A., BONARDO, F., and GONZALEZ, S., Modifications and Alternatives to the Cubic Interpolation Process for One-Dimensional Search, Rice University, Aero-Astronautics Report No. 135, 1976.

Additional Bibliography

7. BOX, M.J., A Comparison of Several Current Optimization Methods and the Use of Transformations in Constrained Problems, Computer Journal, Vol. 9, No. 1, 1966.

8. HAARHOFF, P.C., and BUYS, J.D., A New Method for the Optimization of a Nonlinear Function Subject to Nonlinear Constraints, Computer Journal, Vol. 13, No. 2, 1970.

9. HIMMELBLAU, D.M., A Uniform Evaluation of Unconstrained Optimization Techniques, Numerical Methods for Nonlinear Optimization, Edited by F.A. Lootsma, Academic Press, New York, New York, 1972.

10. HUANG, H.Y., and CHAMBLISS, J.P., Quadratically Convergent Algorithms and One-Dimensional Search Schemes, Journal of Optimization Theory and Applications, Vol. 11, No. 2, 1973.

11. HUANG, H.Y., and NAQVI, S., Unconstrained Approach to the Extremization of Constrained Functions, Journal of Mathematical Analysis and Applications, Vol. 39, No. 2, 1976.

12. BOTSARIS, C.A., and JACOBSON, D.H., A Newton-Type Curvilinear Search Method for Optimization, Journal of Mathematical Analysis and Applications, Vol. 54, No. 1, 1976.

13. NORRIS, D.O., and GERKEN, J.D., A Rank-One Algorithm for Unconstrained Function Minimization, Journal of Optimization Theory and Applications (to appear).