JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 31, No. 1, MAY 1980

Computational Experience with Davidon's Least-Square Algorithm[1]

L. W. CORNWELL,[2] M. G. KOCMAN,[3] AND M. F. PROSSER[4]

Communicated by H. Y. Huang

Abstract. Computational results are presented for Davidon's new least-square algorithm. Computational experience with this algorithm is reported which motivated the development of a production code version of the algorithm. Several heuristic modifications, which have been added, are described. Fifteen zero-residual test problems have been used in comparing the new production code version with two established versions of the Levenberg–Marquardt algorithm. The production code version of Davidon's least-square algorithm performed faster and used less function evaluations than the Levenberg–Marquardt algorithm in almost every case of the test problems.

Key Words. Least-square methods, variable-metric methods, Levenberg–Marquardt methods, nonlinear programming, testing algorithms.

1. Introduction

Davidon (Ref. 1) has described a least-square algorithm for approximating the minimum of the sum of squares of M real and differentiable functions ϕ_m over an N-dimensional space. A unique characteristic of the algorithm is that it updates estimates for the location of a minimum after each function ϕ_m and its first derivatives are evaluated. Davidon has suggested that this algorithm is advantageous in those situations in which the following conditions are met:

[1] It is a pleasure to acknowledge and thank M. Thomas, R. & I. Consultant, Western Illinois University Computer Center, for writing the timing routine and taking the time to run the comparison tests on the IBM 360/50. Part of this work was also performed at the Applied Mathematics Division of Argonne National Laboratory under the auspices of the US Energy Research and Development Administration.

[2] Professor of Mathematics, Western Illinois University, Macomb, Illinois.

[3] Member of Technical Staff, Bell Laboratories, Denver, Colorado.

[4] Research Analyst, Management Research Department, Anheuser–Busch, St. Louis, Missouri.

0022-3239/80/0500-0027$03.00/0

(i) $M \gg N$, i.e., there are many more data points than parameters;

(ii) rapid approach to an approximate minimum is more important than final convergence to an exact minimum;

(iii) in many choices of weighting factors λ_m, the location of the minimum for the weighted sum of squares $\sum_m \phi_m^2 \lambda_m$ closely approximates that for the sum $\sum_m \phi_m^2$.

In the Gauss algorithm, the $(i+1)$th approximation for the function is obtained using

$$f_{i+1}(x) = \sum_m (\phi_m(x_i) + (x - x_i)^T \nabla \phi_m(x_i))^2, \tag{1}$$

where i is the index of the iteration and $\nabla \phi_m$ is the $N \times 1$ gradient vector whose components are the first derivatives of ϕ_m. In the new algorithm, the quadratic approximation f_{i+1} is obtained by adding just one term from the sum on the right of Eq. (1) to a multiple of the previous quadratic approximation, i.e.,

$$f_{i+1}(x) = (\phi_m(x_i) + (x - x_i)^T \nabla \phi_m(x_i))^2 + \lambda f_i(x), \tag{2}$$

where λ is a positive number less than one. It can be shown that, if the functions ϕ_m are all linear, if $f_0 = 0$, and if $\lambda = 1$, then the Mth quadratic approximation f_M defined by Eq. (1) is just $\sum_m \phi_m^2$, so that x_m is then the exact solution of the linear least-square problem.

The update functions for the scalar α_i, the $N \times 1$ column vector x_i, and the $N \times N$ positive definite matrix H_i (which are used to define the quadratic approximation f_i) are

$$\alpha_{i+1} = (\alpha i + \phi_i^2/\gamma_i)\lambda, \tag{3}$$

$$x_{i+1} = x_i - H_i \phi_i \nabla \phi_i / \gamma_i, \tag{4}$$

$$H_{i+1} = (H_i - H_i \nabla \phi_i (H_i \nabla \phi_i)^T / \gamma_i)/\lambda, \tag{5}$$

where

$$\phi_i = \phi_{m(i)}(x_i), \qquad \gamma_i = \lambda + (\nabla \phi_i)^T H_i \nabla \phi_i,$$

and $m(i)$ determines a sequence in which all m functions ϕ_m and their gradients $\nabla \phi_m$ are evaluated in m iterations. Note that Eq. (3) is not used in Eq. (4) and Eq. (5).

The weighting factor λ is defined to be between zero and one. Davidon makes several suggestions about establishing a value for λ. In one variation, $\lambda_i = 1$ for all i not divisible by M and every Mth λ is a positive number close to zero. A second variation of the algorithm is to start with one value of λ, typically $N/(N=1)$, and then increase λ toward one as the search progresses.

An operation count was also performed on the new algorithm. Using a scaled representation for the matrix H, the number of multiplications or divisions needed in each iteration to update x_i and H_i is $(3N^2+7N+2)/2$. The Gauss least-square algorithm requires approximately $N^2/2$ multiplications after each evaluation of a function ϕ_m and its gradient. The Gauss algorithm also requires $N^3/2$ operations in order the invert the Hessian of the quadratic approximation. It should be noted that there is no matrix inversion or line search in the new algorithm.

Davidon also points out that the new algorithm generates a sequence of x_i which does not converge to the location of the minimum of $\sum_m \phi_m^2$, but fluctuates about it with an amplitude that depends on λ. Selecting λ close to zero enables the algorithm to locate the minimum relatively quickly, but causes large fluctuations about the minimum. Selecting λ close to one decreases the fluctuations but increases the number of iterations to locate the minimum.

Davidon defined *equivalent function evaluations* and *data cycles*. The calculation of the sum of squares of all m functions ϕ_m and the calculation of the N first derivatives of this sum are defined as $N+1$ equivalent function evaluations. A data cycle is defined as an evaluation of all M functions ϕ_m and their first derivatives, without regard to the points at which they are evaluated. Note that each iteration of other least-square algorithms constitutes at least one data cycle.

It has also been pointed out that the updating of α in Eq. (3) is not needed in Eq. (4) or Eq. (5), but may be useful in some criteria for termination. Davidon suggested that it is often satisfactory to stop when a specified number of successive iterations fail to decrease the smallest value yet obtained for α.

2. Early Computational Experience

To gain computational experience with the new least-square algorithm, a computer code was written which represented Eqs. (3), (4), and (5). A very simple version of the algorithm was written which did not alter the order of evaluations of ϕ_i's and did not represent the H matrix in the forms

$$H_i = A_i/\sigma_i \qquad \text{or} \qquad H_i = J_i J_i^T/\sigma_i.$$

The computer code was written in single precisions FORTRAN and executed on an IBM 370/MOD 195 computer. Although the estimate of the function was not needed for the implementation of the update formulas, it was included along with the exact evaluation of the function for comparison

purposes. The stopping criteria used was

$$f<10^{-10},$$

where f was the exact value of the function. This stopping criteria was quite artificial, but provided data for comparison with other algorithms. It would not be the stopping criteria for a production code of the algorithm.

Each problem selected was solved for several values of the parameter λ. Seven values of λ were selected. Those were $\lambda = 0.9$, 0.7, 0.5, 0.3, 0.1, 0.001, and 0.00001, where λ remained fixed for a particular run of the code. Fifteen problems were selected to test the computer code. The problems are described in the Appendix (Section 6).

Table 1 represents a summary of the number of data cycles generated to meet the stopping criteria for each problem and each value of λ. Reference 2 contains a more detailed description of the results generated for each problem.

In examining the information in Table 1 and Ref. 2, several instabilities were observed. Small values of λ ($\lambda < 0.1$) caused overflow in most problems. Different values of λ appeared to be the optimal choice for the problems presented. Cycling through the same sequence of x points occurred for one problem.

Table 1. Comparison of data cycles.

	λ						
Problem	0.9	0.7	0.5	0.3	0.1	0.001	0.00001
ROS	151	45	23	13.5	7	2.5	2.5
CUBE	104	33	17	10	6	2	2
BAD	702	177	90	52	56	14	*
TWO	57†	18	10.5	8.5	6.5	6.5	7
BEALE	39.67	12.67	7.33	4.67	3.33	3.33	3.33
HEL	77.67	24.67	13.67	8.67	7.33	8	*
PSF	66.33	22	13	10	9	*	*
POW 1	77.75	25.75	15.75	12.75	12.75	*	*
POW 2	64.75	21.75	13.75	11.75	10.75	*	*
POW 3	41.50	14.50	8.75	7.50	6.75	*	*
MIELE	49.60	18.00	12.20	10.20	10.20	*	*
WOOD	61.86	22.00	‡	‡	18.28	1.71	*
VAR 5	16.4	5.8	3.6	2.4	1.8	1.4	1
BOX	24	7.70	4.60	*	*	*	*
VAR 10	9.8	3.6	2.2	1.6	1.4	1.2	*

* Overflow occurred.

† f was not less than 10^{-10}, but function and position vector did not change after 57 data cycles.

‡ Cycling occurred.

One serious problem with Davidon's least-square algorithm was the method of termination. Davidon suggested that the value of α in Eq. (3) could be used to terminate the algorithm. In a private communication, he pointed out that $M(1-\lambda)\alpha$ approximates the value of the function (sum of squares).

A simpler method of estimating the value of the function is to check the value of the actual sum of squares at the end of each data cycle. The value of ϕ_i is available at each iteration and the sum of squares can be collected over M iterations. This would be a crude estimate of the function, since the x values change each iteration.

To compare the crude estimate, Davidon's estimate, and the actual value of the function, the computer code of Davidon's least-square algorithm was run on the 15 test problems. The value of λ was fixed at $\lambda = 0.1$ and the value of ϵ used was $\epsilon = 10^{-5}$. The algorithm was changed to terminate when the absolute value of the previous estimated sum of squares and the new estimate are less than ϵ, i.e.,

$$|ss_{i+1} - ss_i| < \epsilon,$$

where ss_i is the estimated sum of squares in data cycle i. Note that termination will occur only at the end of a data cycle.

Table 2. Comparison of estimates of the sum of squares.

Problem	IT	DC	WVF	ESS	AVF
ROS	10	5	0.56×10^{-8}	0.26×10^{-9}	0.26×10^{-13}
CUBE	10	5	0.20×10^{-8}	0.32×10^{-11}	0.32×10^{-15}
BAD	30	15	0.18×10^{-7}	0.47×10^{-5}	0.86×10^{-8}
TWO	12	6	0.24×10^{-9}	0.21×10^{-12}	0.21×10^{-16}
BEALE	15	5	0.17×10^{-12}	0.25×10^{-16}	0.10×10^{-21}
HEL	36	12	0.89×10^{-14}	0.30×10^{-16}	0.30×10^{-22}
PSF	21	7	0.83×10^{-10}	0.92×10^{-7}	0.59×10^{-8}
POW 1	44	11	0.92×10^{-9}	0.64×10^{-7}	0.40×10^{-8}
POW 2	36	9	0.23×10^{-9}	0.16×10^{-6}	0.99×10^{-8}
POW 3	32	8	0.18×10^{-10}	0.66×10^{-7}	0.41×10^{-8}
MIELE	40	8	0.70×10^{-12}	0.39×10^{-6}	0.25×10^{-7}
WOOD	†				
VAR 5	20	2.86	0.63×10^{-18}	*	0.47×10^{-34}
BOX	40	4	0.71×10^{-27}	0.38×10^{-30}	0.42×10^{-39}
VAR 10	23	1.92	0.11×10^{-20}	*	0.47×10^{-30}

* Overflow occurred because the estimated sum of squares became too small before the completion of a data cycle. This supported the need for a check for termination after each iteration.

† Cycling occurred.

Table 2 represents the results of 15 test problems under the conditions listed above. In the table, column one contains the number of iterations IT, column two contains the number of data cycles DC, column three contains the value of Davidon's weighted value of the function WVF, column four contains the crude estimated sum of squares ESS, and column five contains the actual value of the function AVF. Note that Davidon's estimated value of the function fluctuates around the actual value and the crude estimate is consistently larger than the actual value. The crude estimate appears to be a conservative value for the function which requires very little additional computation.

3. Production Code Version

Several heuristic modifications have been used in developing a production code version of Davidon's least-square algorithm. The heuristic modifications were developed using the information gained from the empirical results of the 15 test problems.

3.1. Selecting λ. Table 1 indicates that overflow in the computer occurred for small selections of λ. The table also indicates that a small value of λ provides excellent results provided overflow does not occur. The following heuristic procedure was developed to take advantage of using as small a λ as possible. Select an initial value for λ. During the execution of the algorithm, monitor the computation for overflow. If an overflow occurs, restart the algorithm at the initial x vector, and let $\lambda = \lambda + \Delta\lambda$, where $\Delta\lambda$ is a fixed increment. In the present version of the algorithm, the initial values of λ and $\Delta\lambda$ are 0.1.

3.2. Random Selection of Terms. In Davidon's original article, he suggests using a scrambled sequence for the order of evaluating the M terms ϕ_i and their gradients $\nabla\phi$. He made this recommendation to reduce the fluctuations in the x vector. From the computational experience, the selection of a random sequence for each M terms removed the problem of cycling found with one of the problems in Table 1. A subroutine was added to generate pseudo random numbers; and, for each data cycle, a new random sequence was generated. The feature was added to the algorithm to guard against cycling.

3.3. Termination of Algorithm. Two checks were used to terminate the algorithm. The estimated sum of squares described in Section 3 is used at the end of each data cycle. A check on the change in the x vector is also used.

The check on the estimate of the sum of squares was initially used in terminating the algorithm. Since this value was not computed until the end of each data cycle, the termination criteria was not checked with sufficient frequency when M was large. Computational experience also indicated that a minimum was located in the middle of a data, cycle, but an overflow occurred before the data cycle was complete. The overflow occurred because of the small values in the components in the gradient $\nabla\phi_i$. The overflow caused the algorithm to restart, which required additional computation to find the minimum already located.

The second criterion was added to the algorithm to check consecutive x vectors after each iteration. In comparing consecutive x vectors, early termination occurred in a few problems because some terms ϕ_i made very small changes in the x vectors. The comparison was then made between the current x vector and the x vector from k previous iterations, where

$$k = \min(M, 4).$$

The lag check on the x vectors would monitor when the fluctuations have settled.

The termination criterion used in the production code version to check on the estimated sum of squares is

$$|ss_{i+1} - s_i|/|s_i| < \epsilon, \tag{6}$$

where i represents the number of the data cycle; the termination criterion based on the comparison of the two x vectors is

$$|\bar{x}_j - \bar{x}_{j-k}|/|\bar{x}_{j-k}| < \epsilon, \tag{7}$$

where j represents the current iteration and

$$k = \min(M, 4).$$

In the case where $|s_i|$ or $|\bar{x}_{j-k}|$ becomes small, Eqs. (7) and (8) are replaced by

$$|s_{i+1} - s_i| < \epsilon \tag{8}$$

and

$$|\bar{x}_j - \bar{x}_k| < \epsilon, \tag{9}$$

respectively.

4. Computational Results

In an attempt to make a valid comparison of the production code version of Davidon's least-square algorithm (PCVDLS) with other computer

codes, two least-square codes were obtained from the Applied Mathematics Division, Argonne National Laboratory, Argonne, Illinois. These two codes (LMCHOL and LMGENV) are variations of the Levenberg–Marquardt algorithm and are widely used for nonlinear least-square problems. LMCHOL is a subroutine which is a modularized modification of the nonlinear least-square optimization algorithm (VA07A) by Fletcher (Ref. 3). LMGENV is based on a restricted step hybrid algorithm, combining the Levenberg–Marquardt and steepest-descent algorithms. It is a subroutine which is a modularized modification of the nonlinear least-square optimization algorithm VA05A by Powell, contained in the Harwell Subroutine Library (Ref. 4). Since PCVDLS uses gradient information, the only valid code for comparison is LMCHOL. LMGENV uses forward difference approximations to the partial derivatives and should not be used in a direct comparison. The information on LMGENV is provided for a rough comparison.

All three codes were written in FORTRAN IV and implemented on an IBM/MOD 50 computer. Double-precision arithmetic was used, and all termination checks used $\epsilon = 10^{-5}$. The initial x vectors for the 15 zero-residual test problems are described in the Appendix. In PCVDLS, the order of the ϕ_i terms was selected randomly, and the initial H matrix was the identity matrix.

A timing routine was written and tested to obtain valid CPU times for the three computer codes. The times on all problems are accurate within one-sixtieth of a second. Each problem was solved 50 times, and the times reported do not include input/output times. Although the computer has several partitions, the computer times were obtained with only the single job being executed.

Tables 3 and 4 provide comparisons of the three least-square codes. A comparison of the number of function evaluations is described in Table 3. The table present number of iterations IT, number of data cycles DC, number of function evaluations NF, and number of Jacobian evaluations NJ. Table 4 compares the final values of the objective function FVF and execution time ET (in seconds).

Examination of Table 3 supports the effectiveness of the PCVDLS code. Since both PCVDLS and LMCHOL use derivative information, a valid conservative comparison would be between the DC and NJ columns. PCVDLS evaluates the function and the Jacobian matrix for each DC, and LMCHOL usually makes more calls for the objective function than the Jacobian matrix. In this case, the DC count is smaller than the NJ count for every problem solved by LMCHOL. The execution times in Table 4 also support the effectiveness of the PCVDLS code. As one might conjecture, the two codes which use gradient information take less time in almost all cases. Execution time is valid for comparing all three codes. With one exception, PCVDLS is

Table 3. Comparison of function evaluations.

	PCVDLS		LMCHOL		LMGENV
Problem	IT	DC	NF	NJ	NF
ROS	10	5	18	14	27
CUBE	12	6	16	11	20
BAD	34	17	47	36	46
TWO	13	6.5	9	7	11
BEALE	16	5.33	7	6	9
HEL	22	7.33	13	9	27
PSF	21	7	36*	1	24
POW 1	48	12	23	22	40
POW 2	36	9	19	18	20
POW 3	32	8	18	17	13
MIELE	40	8	28	27	35
WOOD	26	3.71	70	63	19*
VAR 5	16	2.29	6	5	15
BOX	29	2.9	17*	2	63
VARIO	23	1.92	7	6	39

* Failure of the algorithm.

Table 4. Comparison of final value of function and execution time (seconds).

	PCVDLS		LMCHOL		LMGENV	
Problem	FVF	ET	FVF	ET	FVF	ET
ROS	0.145E-12	4.12	0.0	13.95	0.465E-12	34.78
CUBE	0.105E-15	4.83	0.173E-32	12.33	0.780E-12	24.75
BAD	0.418E-8	15.18	0.287E-29	44.53	0.357E-5	65.32
TWO	0.117E-13	5.12	0.493E-31	6.90	0.187E-5	11.67
BEALE	0.469E-16	6.72	0.0	6.03	0.562E-5	11.28
HEL	0.512E-13	14.48	0.493E-29	19.52	0.195E-6	59.58
PSF	0.347E-7	14.75	1.25*	43.25	0.506E-5	53.6
POW 1	0.688E-8	44.62	0.520E-20	58.75	0.246E-5	160.47
POW 2	0.142E-7	33.45	0.341E-19	48.12	0.698E-5	62.78
POW 3	0.626E-8	29.67	0.115E-18	45.82	0.951E-5	36.08
MIELE	0.106E-7	38.33	0.185E-26	83.25	0.135E-4	143.80
WOOD	0.421E-15	23.82	0.173E-22	218.60	7.877*	85.45
VAR 5	0.961E-24	21.07	0.909E-55	34.03	0.313E-5	71.06
BOX	0.152E-20	25.88	34.95*	55.97	0.294E-5	472.57
VAR 10	0.925E-31	86.38	0.230E-46	143.65	0.136E-12	677.13

* Failure of the algorithm.

faster than both Levenberg–Marquardt codes. The FVF columns indicate that LMCHOL finds the smallest value of the objective function, but the additional execution times are not entirely attributable to the additional precision. The fact that PCVDLS uses no matrix inversion or one-dimensional search surely accounts for much of the differences in execution times.

For an overall evaluation of the three computer codes on the 15 test problems, one can make two comparisons. On the 13 problems solved by LMCHOL, PCVDLS used 327.79 sec and LMCHOL used 735.48 sec. This represents a 124% increase in CPU time. For the 14 problems solved by LMGENV, PCVDLS used 344.60 sec and LMGENV used 1884.86 sec for a 447% increase in CPU time.

5. Conclusions

Early work with Davidon's least-square algorithm produced encouraging results. The algorithm itself has several desirable properties, including that it is simple, short, requires no matrix inversion, and uses no one-dimensional search. In addition, it produced a satisfactory solution for several of the zero-residual problems in considerably fewer equivalent function evaluations.

Several difficulties were encountered before a satisfactory code was produced. Cycling was eliminated by a random selection of the order in which the terms were examined. To encourage rapid convergence, it was decided that the weight λ given to the previous terms was to be one-tenth. If overflow was encountered, the process was to be restarted with an increased value of λ. Finally, a combination of two stopping criteria was settled upon. The first criterion stopped when the solution vector did not vary by more than ϵ after several iterations. The other criterion stopped when approximation of the function value did not vary by more than ϵ.

The computational results in this paper indicate that Davidon's least-square algorithm should be considered seriously for nonlinear least-square problems. Although the authors are quite optimistic about the future of PCVDLS, they realize the need for further testing of the code on nonlinear problems which do not have the zero-residual property.

6. Appendix: Description of Test Problems

6.1. Rosenbrock's function (ROS), Ref. 5:
$f = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$;
minimum $f = 0$ at (1.0, 1.0);
starting point is (−1.2, 1.0).

6.2. Cube (CUBE), Ref. 6:
$f = 100(x_2 - x_1^3)^2 + (1 - x_1)^2$;
minimum $f = 0$ at (1.0, 1.0);
starting point is (0.5, 0.5).

6.3. Badly scaled function (BAD), Ref. 7:
$f = (10{,}000 x_1 x_2 - 1)^2 + [\exp(-x_1) + \exp(-x_2) - 1.0001]^2$;
minimum $f = 0$ at (0.00001098, 9.106) or (9.106, 0.00001098);
starting point is (0.0, 1.0).

6.4. Two-global minima function (TWO), Ref. 7:
$f = (x_1^2 - x_2 - 1)^2 + [(x_1 - 2)^2 + (x_2 - 0.5)^2 - 1]^2$;
minimum $f = 0$ at (1.546343, 1.391176) or (1.067346, 0.1392277);
starting point is (0.1, 2.0).

6.5. Beale's function (BEALE), Ref. 8:
$f = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2$
$+ [2.625 - x_1(1 - x_2^3)]^2$;
minimum $f = 0$ at (3.0, 0.5);
starting point is (2.0, 0.7).

6.6. Helical-valley function (HEL), Ref. 9:
$f = 100(x_3 - 10\theta)^2 + 100(R - 1)^2 + x_3^2$,

$$\theta = \begin{cases} \tan^{-1}(x_2/x_1)/2\pi, & x_1 > 0, \\ 0.75, & x_1 = 0, \\ \tan^{-1}(x_2/x_1)2\pi + 0.5, & x_1 < 0, \end{cases}$$

$R = (x_1^2 + x_2^2)^{1/2}$;
minimum $f = 0$ at (1.0, 0.0, 0.0);
starting point is (−1.0, 0.0, 0.0).

6.7. Powell's second function (PSF), Ref. 10:
$f = [(1 + (x_1 - x_2)^2)^{-1} - 1]^2 + [\sin(\pi x_2 x_3) - 1]^2$
$+ [\exp - \{(x_1 + x_2)/x_2 - 2\}^2 - 1]^2$;
minimum $f = 0$ at (x_1, x_2, x_3),
where $x_1 = x_2 = x_3 = \pm(4n + 1)^{1/2}$;
starting point is (0.0, 1.0, 2.0).

6.8. Powell's four-parameter function (POW 1), Ref. 11:
$f = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$;
minimum $f = 0$ at (0.0, 0.0, 0.0, 0.0);
starting point is (10.0, 10.0, 10.0, −10.0).

6.9. Powell's four-parameter function (POW 2), Ref. 11:
f = same type as in Problem 6.8;
minimum = same as in Problem 6.8;
starting point is (3.0, −1.0, 0.0, 1.0).

6.10. Powell's four-parameter function (POW 3), Ref. 11:
f = same type as in Problem 6.8;
minimum = same as in Problem 6.8;
starting point is (−0.1, −0.1, 1.0, 1.0).

6.11. Miele's function (MIELE), Ref. 12:
$$f=[\exp(x_1)-x_2]^4+100(x_2-x_3)^6+\tan^4(x_3-x_4)+x_1^8+(x_4-1)^2;$$
minimum $f=0$ at (0.0, 1.0, 1.0, 1.0);
starting point is (1.0, 2.0, 2.0, 2.0).

6.12. Wood's function (WOOD), Ref. 9:
$$f=100(x_2-x_1^2)^2+(1-x_1)^2+90(x_4-x_3^2)^2+(1-x_3)^2+0.2(x_2-1)^2+0.2(x_4-1)^2+9.9(x_2+x_4-2)^2;$$
minimum $f=0$ at (1.0, 1.0, 1.0, 1.0);
starting point is (−3.0, −1.0, −3.0, −1.0).

6.13. Variable-dimension problem with five variables (VAR 5), Ref. 7:

$$f=\sum_{i=1}^{5}x_i^2+\left(\sum_{i=1}^{5}i^{1/2}x_i\right)^2+\left(\sum_{i=1}^{5}i^{1/2}x_i\right)^4;$$

minimum $f=0$ at (0.0, 0.0, 0.0, 0.0, 0.0);
starting point is (0.1, 0.1, 0.1, 0.1, 0.1).

6.14. Box's function (BOX), Ref. 7:

$$f=\sum_{i=1}^{10}[\exp(-x_1\zeta_i)-\exp(x_2\zeta_i)-x_3(\exp(-\zeta_i)-\exp(-10\zeta_i)]^2, \qquad \text{where } \zeta_i=i/10;$$

minimum $f=0$ at (1.0, 10.0, 1.0), or (10.0, 1.0, −1.0), or $(x_1, x_2, 0.0)$, where $x_1=x_2$;
starting point is (0.0, 20.0, 20.0).

6.15. Variable-dimension problem with ten variables (VAR 10), Ref. 7:

$$f=\sum_{i=1}^{10}x_i^2+\left(\sum_{i=1}^{10}i^{1/2}x_i\right)^2+\left(\sum_{i=1}^{10}i^{1/2}x_i\right)^4;$$

minimum $f = 0$ at (0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0); starting point is (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1).

References

1. DAVIDON, W. C., *New Least Squares Algorithms*, Journal of Optimization Theory and Applications, Vol. 18, No. 2, 1976.
2. CORNWELL, L. W., BABINGTON, W., and KOCMAN, M., *Computational Experience with Davidon's Least Squares Algorithm*, submitted at the operations Research Society of America/The Institute of Management Science National Meeting, November, 1976.
3. FLETCHER, R., *A Modularized Marquardt Subroutine for Nonlinear Least-Squares*, Harwell Report AERE-R.6799, 1971.
4. HOPPER, M. J., *The Harwell Subroutine Library*, Harwell Report AERE-R.7477, pp. 36, 65, 1973.
5. ROSENBROCK, H. H., *An Automatic Method for Finding the Greatest and Least Value of a Function*, Computer Journal, Vol. 3, No. 2, 1960.
6. WITTE, B. F., and HOLST, W. R., *Two New Direct Minimum Search Procedures for Functions of Several Variables*, submitted at the 1964 Spring Joint Computer Conference in Washington, D.C.
7. HILLSTROM, K. E., *MINIPACK I, A Study in the Modularization of a Package of Computer Algorithms for the Unconstrained Nonlinear Optimization Problem*, Technical Memorandum TM-252, Argonne National Laboratory, 1974.
8. BEALE, E. M. L., *On an Iterative Method for Finding a Local Minimum of a Function of More Than One Variable*, Princeton University, Princeton, Technique Report 25, 1958.
9. COLVILLE, A. R., *A Comparative Study on Nonlinear Programming Codes*, IBM N.Y. Science Center Report 320–2949, June, 1968.
10. POWELL, M. J. D., *An Iterative Method for Finding Stationary Values of a Function of Several Variables*, Computer Journal, Vol. 5, No. 5, 1962.
11. FLETCHER, R., and POWELL, M. J. D., *A Rapidly Convergent Descent Method for Minimization*, Computer Journal, Vol. 6, No. 6, 1963.
12. HUANG, H. Y., *Numerical Experiments on Quadratically Convergent Algorithms for Function Minimization*, Journal of Optimization Theory and Applications, Vol. 6, No. 4, 1970.