A new and dynamic method for unconstrained minimization

J. A. Snyman

Department of Applied Mathematics, University of Pretoria, Pretoria, Republic of South Africa

(Received July 1981)

A new gradient method for unconstrained minimization is proposed. The method differs conceptually from other gradient methods. Here the minimization problem is solved via consideration of the analogous physical problem of the motion of a particle in a conservative force field, where the potential energy of the particle is represented by the function to be minimized. The method simulates the motion of the particle and by monitoring its kinetic energy an interfering strategy is adopted which ensures that the potential energy is systematically reduced. An algorithm, representing this approach, has been extensively tested using seven different kinds of test function. The results are compared with the performance of other quasi-Newton methods. Overall the results are encouraging showing the method to be competitive and particularly robust and reliable. The evidence that the computational time required by the new method for convergence increases roughly linearly with the dimension of the problem is of considerable significance.

Key words: unconstrained minimization, optimization, mathematical model

Introduction

In this paper a new method is proposed for solving the unconstrained minimization problem of finding the least value of a function $F(x)$ of n real variables $x = (x_1, x_2, \ldots, x_n)^T$. The method requires that only the gradient vector $\nabla F(x)$ be explicitly known. The function $F(x)$ need not necessarily be given explicitly since its evaluation is not required for the successful implementation of the algorithm. The method differs conceptually from other gradient methods in that it approaches the problem by considering the analogous physical problem of the motion of a particle of unit mass in a n-dimensional conservative force field, where the potential energy of the particle is represented by the function $F(x)$. In such a field the total energy of the particle, consisting of its potential and kinetic energy, is conserved. The method simulates the motion of the particle and by monitoring its kinetic energy an interfering strategy is adopted which ensures that the potential energy is systematically reduced. In this way the particle is forced to follow a trajectory towards a local minimum in potential energy at x^*.

It is clear that this new approach represents a natural method for solving the pure mathematical problem. Although it is based on physical reasoning the final algorithm, as it will be presented here, contains a number of features which were artificially and heuristically introduced. As will be apparent, the values for these heuristic parameters almost certainly do not represent an optimum choice. Nevertheless the current algorithm proves to be competitive with other gradient methods such as the quasi-Newton methods. The very strong indication that, if the individual derivatives have forms of more or less similar complexity, the computational time required for convergence increases roughly linearly with the dimension n of the problem is particularly encouraging. This should be seen in comparison with the quadratic behaviour exhibited by quasi-Newton methods. For the test problems considered in this paper the performance of the dynamic method will be shown to be spectacularly superior for large n. One should, however, bear in mind that for the cases to be considered the gradient evaluations are relatively inexpensive. The algorithm also appears to be extremely robust. In testing it on seven different kinds of test function with n varying from 2 to 150 and with approximately 140 different starting points in total, the algorithm never failed to converge. An extremely simple and easy to use FORTRAN program embodying the algorithm is also presented.

0307-904X/82/060449-14/$03.00

Dynamic model

We consider the problem:

minimize:

$$F(x) \quad x = (x_1, x_2, \ldots, x_n)^T \in R^n \tag{1}$$

where for the present, we assume $F \in C_1$.

Suppose $F(x)$ represents the potential energy of a particle of unit mass in a n-dimensional force field, then, if F has a local minimum at x^*, it follows that:

$$F(x) = -\int_{x^*}^{x} a(s)^T \, ds + F(x^*) \tag{2}$$

where $a(s)$ is the force acting on the particle at the point s.

The kinetic energy associated with the particle is defined by:

$$T(x) = \frac{1}{2}\sum_{i=1}^{n} \dot{x}_i^2 = \tfrac{1}{2}\|\dot{x}\|^2 \tag{3}$$

and the Lagrangian is given by:

$$L(x) = T(x) - F(x) \tag{4}$$

By applying Hamilton's principle we obtain the equations of motion:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}_i}\right) - \frac{\partial L}{\partial x_i} = 0 \quad i = 1, \ldots, n \tag{5}$$

On substituting (2) and (3) into (4) and (5) these equations may be written in vector form as:

$$\ddot{x} = -\nabla F = a \tag{6}$$

The trajectory $x(t)$ of the particle, as a function of the time t, is given by the solution of equation (6) subject to prescribed values for the starting point $x(0)$ and the initial velocity $v(0)$. In particular we consider the initial conditions:

$$x(0) = x_0$$
$$\dot{x}(0) = v(0) = v_0 = \mathbf{0} \tag{7}$$

where x_0 may be any arbitrarily prescribed starting point. Equation (6) implies conservation of energy, i.e., for any point x along the trajectory we have:

$$T(x) + F(x) = T(x_0) + F(x_0) = E_0 \tag{8}$$

where E_0 is the initial total energy. With $T(x_0) = 0$ it follows that along the particle path we have $F(x) \leqslant F(x_0)$. Since no frictional forces are present the particle will be in continual motion and its trajectory will not necessarily pass through x^* where we assume the gradient vector to be equal to zero.

It is now suggested that, if the trajectory of the particle is monitored, then one or other interfering strategy may be adopted by which the energy of the particle is systematically reduced. In this way the particle is forced to follow a path towards the local minimum at x^*. Various different strategies suggest themselves of which only two of the most promising will be discussed here. (i), The first and most obvious possibility is to introduce an artificial damping term into equation (6) giving:

$$\ddot{x} = -\nabla F - \alpha\dot{x} \tag{9}$$

where $\alpha > 0$ is some suitably chosen damping constant. The efficiency of this approach depends, however, on a favourable choice for α. If it is too small the particle will be too lightly damped and it will follow an oscillatory path about x^* (not necessarily through x^*) of slowly diminishing amplitude. On the other hand if α is too large the particle will be heavily damped and will consequently converge very slowly to x^*. An optimum choice for α is clearly dependant on both the position of the particle and the nature of the force field, and therefore constitutes a non-trivial problem for which a simple solution is not evident. (ii), A second strategy, and the one essentially adopted in the algorithm presented here, is based on the monitoring of the kinetic energy of the particle at fixed time intervals. Let x_k and v_k respectively denote the position and the velocity of the particle at time $t_k = k\Delta t$. By conservation of energy (8) it follows that for any two consecutive times t_k and t_{k+1} we have:

$$\begin{aligned}\Delta F_k &= F(x_{k+1}) - F(x_k) = -T(x_{k+1}) + T(x_k)\\ &= -\Delta T_k\end{aligned} \tag{10}$$

Hereafter we denote $F(x_k)$ by F_k and $T(x_k)$ by T_k. Clearly if $\Delta T_k > 0$ we obtain the desired effect $\Delta F_k < 0$. To secure this effect as far as possible we may adopt the following simple strategy:

$$\left.\begin{array}{l}\text{Monitor } \|v_{k+1}\| \quad k = 0, 1, \ldots\\ \text{If and only if } \|v_{k+1}\| \leqslant \|v_k\|:\\ \text{set } v_k = \mathbf{0} \text{ and repeat step } k \text{ to } k+1\end{array}\right\} \tag{11}$$

Lemma 1

Application of strategy (11) ensures that the sequence $\{F_k; k = 0, 1, 2, \ldots\}$ is decreasing.

Proof. If $\|v_{k+1}\| > \|v_k\|$ then by (10) $\Delta F_k < 0$.

If $\|v_{k+1}\| \leqslant \|v_k\|$ then setting $v_k = \mathbf{0}$ gives $T_k = 0$ and repeating step k to $k+1$ gives $T_{k+1} \geqslant 0$. It follows then again by (10), that $\Delta F_k \leqslant 0$, which completes the proof.

The strategy adopted in the final algorithm, of which the details are presented later in the paper appears to be more complicated than (11) in that it contains a number of heuristic elements which were introduced with the purpose of speeding up convergence. The basic reasoning embodied in the final algorithm is, however, essentially the same as in the simple approach outlined above.

Method of integration

The feasibility and success of the dynamic approach is dependent on the availability of an accurate, stable and economic numerical method for solving the equations of motion (6). In particular it is imperative that the integration method should retain, as far as possible, the energy conservation property (8). Clearly, for the present purpose of minimization, this property is more important than the actual accuracy of the computed trajectory. A highly accurate method (usually at the expense of economy) is therefore not our prime requirement but rather a simple method which, for all practical purposes, conserves energy and which, as a consequence, is relatively stable.

With the above in mind we investigate a very simple method, known as the 'leap-frog' method, and one which was found by Greenspan[1] to be exceptionally stable when applied to the study of nonlinear oscillators and vibrating strings. The method may be stated as follows:

Given x_0, v_0 and a time step Δt, compute for $k = 0, 1, 2, \ldots$

$$\left.\begin{aligned} &x_{k+1} = x_k + v_k \Delta t \quad \text{(Euler forward)} \\ &v_{k+1} = v_k + a_{k+1}\Delta t \quad \text{(Euler backward)} \\ &\text{where:} \\ &a_k = -\nabla F_k \end{aligned}\right\} \qquad (12)$$

The question now arises as to what extent the method given by (12) conserves energy. It can be shown (see appendix 1) that this method leads to the approximate energy conservation relationship:

$$\tfrac{1}{2}\|v_{k+1}\|^2 + F_{k+1} = \tfrac{1}{2}\|v_k\|^2 + F_k + \tfrac{1}{2}\|a_{k+1}\|^2(\Delta t)^2 - \tfrac{1}{2}\Delta x_k^T H(\bar{x})\,\Delta x_k \qquad (13)$$

where H is the Hessian matrix:

$$H(x) = \left[\frac{\partial^2 F(x)}{\partial x_i\, \partial x_j}\right]$$

and

$$\bar{x} = x_{k+1} - \theta\Delta x_k \quad 0 \leqslant \theta \leqslant 1$$

To assess this result we consider the case where F is a convex quadratic function. In this instance H is constant and positive semi-definite. For the trajectory computed by (12) we then have:

$$\Delta F_k = \tfrac{1}{2}\|v_k\|^2 - \tfrac{1}{2}\|v_{k+1}\|^2 + \tfrac{1}{2}\|a_{k+1}\|^2(\Delta t)^2 - \tfrac{1}{2}\Delta x_k^T H \Delta x_k \qquad (14)$$

This expression suggests that the following strategy be used in conjunction with method (12).

Monitor $\|v_{k+1}\|$ $\quad k = 0, 1, \ldots$

If and only if $\|v_{k+1}\|^2 \leqslant \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2$: $\quad$ (15)

set $v_k = \mathbf{0}$ and repeat step k to $k+1$

Lemma 2

If F is a convex quadratic function then application of strategy (15) together with method (12) ensures that the sequence $\{F_k; k = 0, 1, 2, \ldots\}$ is decreasing.

Proof. If $\|v_{k+1}\|^2 > \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2$ it follows directly from (14) and the fact that H is positive semi-definite that $\Delta F_k < 0$. If $\|v_{k+1}\|^2 \leqslant \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2$, then setting $v_k = \mathbf{0}$ and repeating step k to $k+1$ gives, by (12), $\Delta x_k = \mathbf{0}$ and $v_{k+1} = a_{k+1}\Delta t$. Substituting these values into (14) yields $\Delta F_k = 0$, which completes the proof.

In concluding this paragraph it should be stated that other simple integration methods which give better approximations to energy conservation may readily be constructed. Two such improvements are presented in appendix. Since they are not implemented in the algorithm that follows they will not be discussed here.

Construction of dynamic algorithm

Heuristic arguments

Still assuming F to be quadratic we consider the case where the test in (15) is relaxed such that no adjustment is made for:

$$\|v_k\|^2 < \|v_{k+1}\|^2 \leqslant \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2 \qquad (16)$$

It then follows from (14) and (16) that:

$$\tfrac{1}{2}\|a_{k+1}\|^2(\Delta t)^2 - \tfrac{1}{2}\Delta x_k^T H \Delta x_k > \Delta F_k \geqslant -\tfrac{1}{2}\Delta x_k^T H \Delta x_k$$

Thus if (16) applies ΔF_k may be negative or positive. We note that (16) implies that $a_{k+1}^T v_k \leqslant 0$, from which, for $n = 2$ or 3, we infer that the angle between the directions of v_k and a_{k+1} is $\geqslant 90°$. At the same time $\|v_{k+1}\| > \|v_k\|$. For a smooth trajectory and Δt sufficiently small the probability of (16) happening seems very small.

Far away from a minimum on the path one expects $v_k^T a_{k+1} > 0$ and on passing near x^* or a minimum on the actual trajectory it is more likely that $\|v_{k+1}\| < \|v_k\|$ than (16) actually occurring. Also bearing in mind that if (16) should occur the consequences would be minor we simplify our test in (15) to the simpler condition $\|v_{k+1}\| \leqslant \|v_k\|$. In any case for general nonlinear functions the argument presented above and in the previous paragraph for quadratic functions may not, of course, apply precisely. We notice that, since $\Delta x_k = v_k \Delta t$, we have for general functions:

$$\Delta F_k = \tfrac{1}{2}\|v_k\|^2 - \tfrac{1}{2}\|v_{k+1}\|^2 + \tfrac{1}{2}\{\|a_{k+1}\|^2 - v_k^T H(\bar{x})\, v_k\}(\Delta t)^2 \qquad (17)$$

We may therefore expect that in general, for Δt sufficiently small and with F possessing well behaved first and second derivatives, that procedure (15) with the simpler test:

$$\|v_{k+1}\| \leqslant \|v_k\| \qquad (18)$$

should be effective.

The second change we make to (15) relates to the adjustment of v_k if condition (18) is satisfied. If the particle is following a trajectory towards x^* along a narrow valley, say, then it may move slightly sideways in an uphill direction so that (18) may apply. If we now set $v_k = \mathbf{0}$ it is clear that information is lost as regards the general correct direction in which the particle was moving, and consequently convergence to x^* will be slowed down. A more sensible strategy would be to set v_k to a reduced velocity which, while not ensuring a negative ΔF_k, will make it highly probable that F will decrease in the next step. It was therefore arbitrarily decided that should (18) apply the strategy would be to set:

$$v_k = (v_k + v_{k+1})/4 \qquad (19)$$

for the restart. The effectiveness of this procedure was confirmed by carrying out a number of numerical experiments involving the two-dimensional Rosenbrock problem. In the same vein it was heuristically established that, for the restart setting:

$$x_{k+1} = (x_k + x_{k+1})/2 \qquad (20)$$

instead of equal to x_k, gave improved convergence.

If, however, in spite of the above adjustments, condition (18) persists one may conclude that the particle is no longer in a narrow valley but has passed near x^* and is steadily climbing uphill. It was therefore decided that if (18) should apply more than twice consecutively, then v_k should be set to $\mathbf{0}$ as before and the particle restarted from $(x_k + x_{k+1})/2$. Thereafter adjustment (19) will only be made on the first occasion at which (18) applies; otherwise v_k will be set to $\mathbf{0}$.

To prevent unrealistically large steps occurring in the case where the particle is in an area where $\|\nabla F\|$ is exceptionally large a limit is set to the maximum allowable step-size, i.e. it is required that:

$$\|\Delta x_k\| \leqslant \delta \qquad (21)$$

where δ is a prescribed maximum stepsize. The calculations are terminated when:

$$\|\nabla F_k\| = \|a_k\| \leqslant \epsilon \tag{22}$$

where ϵ is a small specified positive number.

Basic algorithm

Taking expressions (18) and (22) together we may now present a practical dynamic algorithm which may be considered a prototype for possible future improvements.

Algorithm (23)

```
(1) Assume $x_0$, $\Delta t$, $\delta$ and $\epsilon$ given; set $i = 0, j = 2$ and $k = -1$.
(2) Compute $a_0 = -\nabla F(x_0)$, $v_0 = \frac{1}{2} a_0 \Delta t$.
(3) Set $k = k + 1$ and compute $\|\Delta x_k\| = \|v_k\| \Delta t$.
(4) If $\|\Delta x_k\| < \delta$, go to (5);
    otherwise set $v_k = \delta v_k / (\Delta t \|v_k\|)$ and go to (5).
(5) Set $x_{k+1} = x_k + v_k \Delta t$.
(6) Compute $a_{k+1} = -\nabla F(x_{k+1})$, $v_{k+1} = v_k + a_{k+1} \Delta t$
(7) If $\|a_{k+1}\| \leqslant \epsilon$, stop; otherwise go to (8).*
(8) If $\|v_{k+1}\| > \|v_k\|$, set $i = 0$ and go to (3);
    otherwise set $x_{k+2} = (x_{k+1} + x_k)/2$, $i = i + 1$ and go to (9).
(9) If $i \leqslant j$, set $v_{k+1} = (v_{k+1} + v_k)/4$, set $k = k + 1$ and go (6);
    otherwise set $v_{k+1} = 0$, $j = 1$, $k = k + 1$ and go to (6).
```

In recognizing that, in practice, a restart represents an additional step, the indices on the left-hand-sides of (19) and (20) have been increased by one in implementing them in the above algorithm.

The performance of this algorithm is illustrated by plotting the computed trajectories for two different two-dimensional problems as depicted in *Figures 1a* and *b*. *Figure 1a* represents the particle path in the case of the well known two-dimensional Rosenbrock problem with starting point $(-1.2, 1)^T$. *Figure 1b* shows the trajectory for the function $F = 400x_1^2 + x_2^2$ with starting point $(5, 10)^T$. The values $\Delta t = 0.05$ and $\delta = 1.0$ were used in both cases. In each case the algorithm readily converges to the required minimum. In the Rosenbrock problem the particle initially jumps to and fro across the valley and after about 14 steps starts moving slowly along the bed of the parabolic valley and follows it more or less closely to the minimum at $(1, 1)^T$. The path clearly justifies the decision not to set the velocity equal to zero when $\|v_{k+1}\|$ is less than $\|v_k\|$. For example, at point 36 on the trajectory $\|v_{36}\| < \|v_{35}\|$. If we restarted with a velocity of zero the particle would again have jumped to and fro before moving slowly away along the valley. However, by reducing the velocity to $(v_{36} + v_{35})/4$ and starting at $(x_{36} + x_{35})/2$ the particle immediately moves away in the right general direction as shown. This procedure is repeated again at point 43. In *Figure 1b* we have a similar situation except that initially the particle keeps moving at maximum stepsize towards the valley and then zig-zags at maximum stepsize along the valley with the first velocity adjustment occurring after step 16.

Time step selection

The above results obtained with the basic algorithm are certainly encouraging. One serious problem remains, namely, how to select an optimum value for the time step Δt. If Δt is too small then many unnecessary steps will have to be computed and as a result the algorithm will become uneconomic. On the other hand if Δt is too large the trajectory will become seriously inaccurate, even near x^*, so that convergence becomes unattainable. What is required is a method by which the algorithm (or computer program) may automatically select a suitable or optimum value for Δt. A complete and satisfactory answer to this problem is not self evident. At this stage the best we can do is to propose a heuristically developed method which, although it was found to work satisfactorily in practice, certainly does not present an optimum strategy.

The time step routine which is implemented is based on the following commonsense argument. If a relative large value for the maximum stepsize δ is prescribed then the number of consecutive times this maximum step is taken,

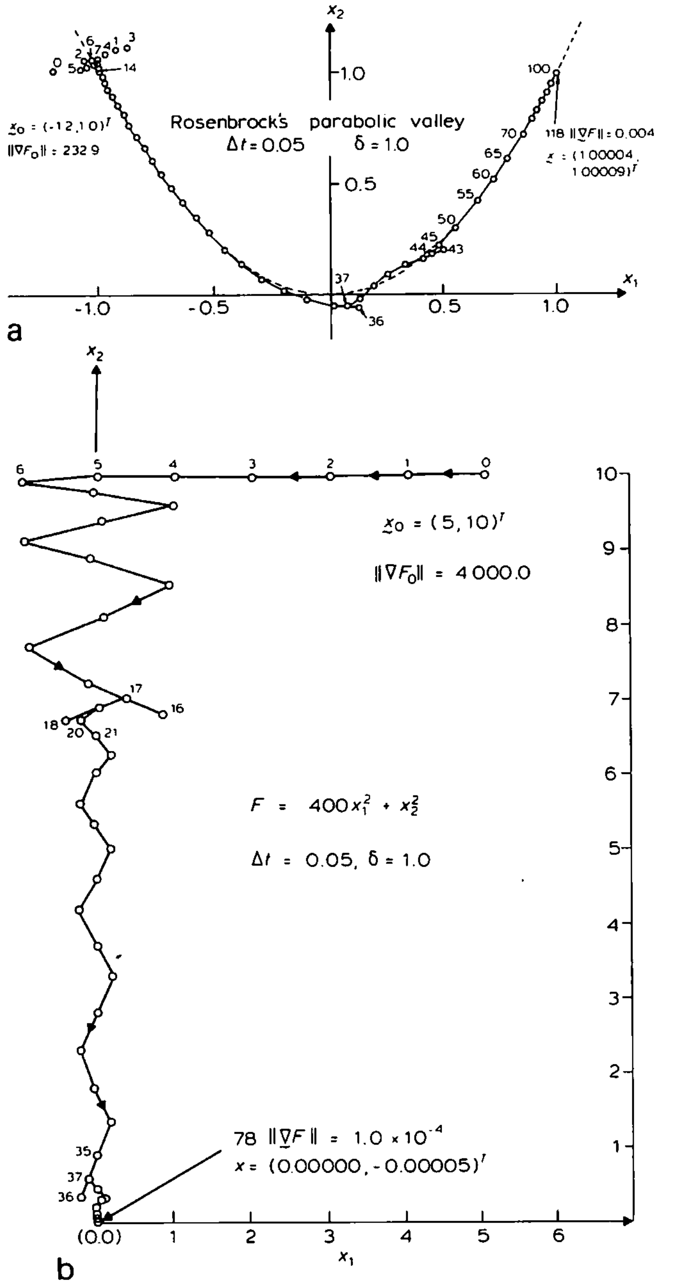

Figure 1 (a) Computed particle trajectory for two-dimensional Rosenbrock problem using no time step control. (b) Computed particle path for function $F = 400x_1^2 + x_2^2$ using no time step control

* See footnote to section on computational results

may be an indication of the inaccuracy of the method and suggest that a reduction in time step is necessary. On the other hand the repetitive execution of steps of maximum size may be completely justified if the particle is very far from x^* and has accelerated along a smooth path towards x^*. In this event it would be unwise to reduce the time step excessively. To cater for both these eventualities the following time step routine was built into the basic algorithm (23).

Time step routine (24)

```
Assume initial $\Delta t$ given and integers $M$ and $N$ specified, set
    $m = 0, n = 0$
Insert the following steps between steps (3) and (4) of
    algorithm (23)
(4a) If $\|\Delta x_k\| \geq \delta$, set $m = m + 1$ and go to (4b);
     otherwise go to (4);
(4b) If $m < M$, go to (4);
     otherwise set $n = n + 1$ and go to (4c).
(4c) If $n > N$, go to (4);
     otherwise set $x_k = (x_k + x_{k-1})/2$, $v_k = (\delta v_k/(\Delta t\|v_k\|)$
     $+ v_{k-1})/4$, $\Delta t = \Delta t/4$, $m = 0$ and go to (5).
```

Typically we may choose $M = 10$ and $N = 2$. This results in the time step being reduced to one-quarter of its current value if the maximum stepsize occurs for 10 consecutive steps. The value $N = 2$ guarantees that the above reduction is not carried out more than twice. This practical procedure for stepsize control proved satisfactory. Indeed, for the seven different kinds of test functions and in total about 140 different starting positions considered in this study the choice of $\Delta t(\text{initial}) = 0.5, \delta = 1.0, M = 10$ and $N = 2$ never failed to produce convergence.

The functioning of the time step control for the parameters set at the above values is illustrated in *Figures 2a* and *b*. The functions considered are again the Rosenbrock problem and $F = 400x_1^2 + x_2^2$. *Figure 2a* shows that an initial choice of $\Delta t = 0.5$ for the Rosenbrock problem is obviously too large. The particle jumps about at maximum stepsize in an almost random fashion although showing a slight tendency to move in the right direction. After the first reduction at step 10 the value of $\Delta t = 0.125$ is still too large. On the second reduction ($\Delta t = 0.03125$) occurring at point 21, the particle starts moving towards the bed of the valley reaching it at about step 30 and starts moving slowly but steadily along the parabolic valley towards the minimum at $(1, 1)^T$. After 127 steps it satisfies the stopping criterion $\|\nabla F\| \leq \epsilon = 10^{-5}$ yielding $x^* = (1.00000, 1.00000)^T$. The performance for the second problem, depicted by the solid line in *Figure 2b*, is similar with two reductions being applied. The only difference is that initially, with the particle far away from the bed of the valley, we have a smoother movement at maximum stepsize towards the valley.

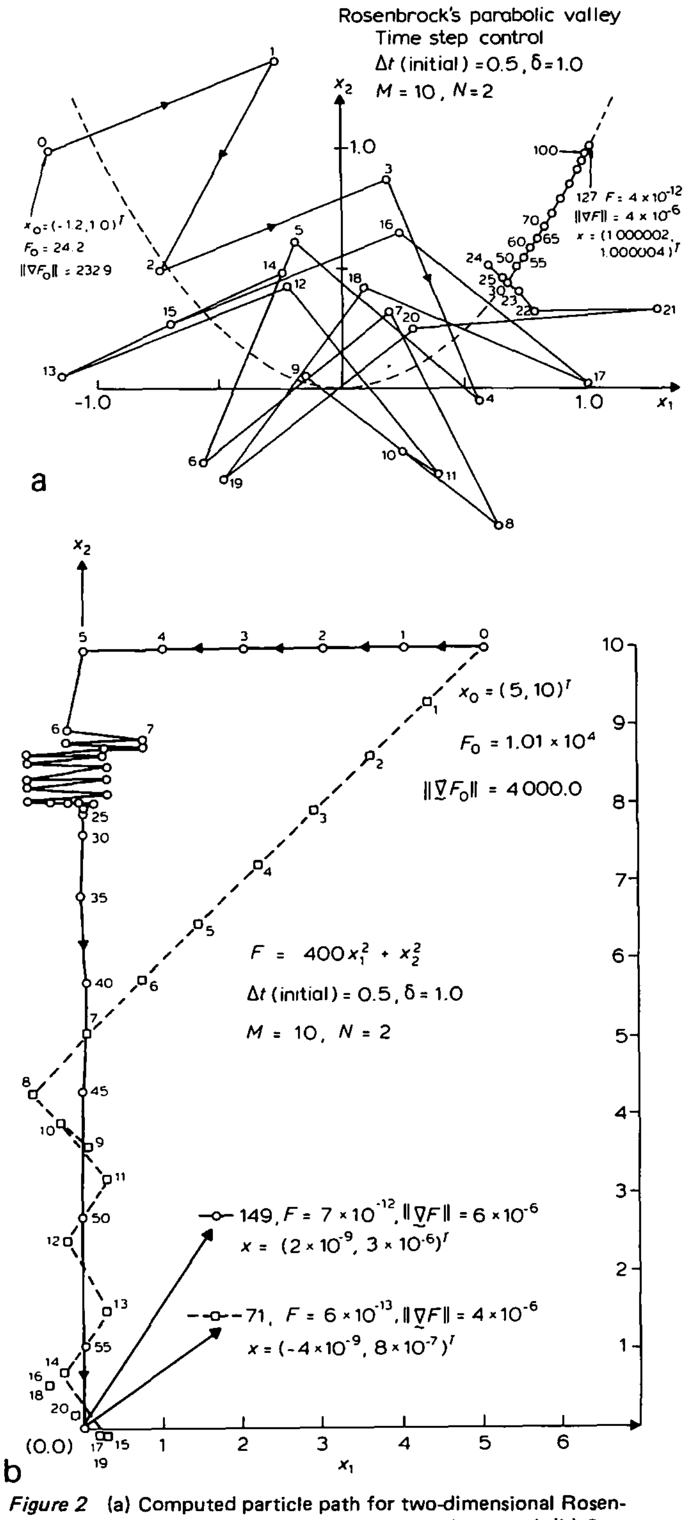

Figure 2 (a) Computed particle path for two-dimensional Rosenbrock problem in case where time step control is applied. (b) Computed particle trajectory for function $F = 400x_1^2 + x_2^2$ in case where time step control is applied. (——), path obtained using the basic algorithm (23). (– – –), trajectory obtained when modification (25) is used

A modification: equally weighted acceleration components

We conclude this section by very briefly discussing a modification to the algorithm which may be of importance in cases where the function is relatively insensitive to one (or more) of the variables. If we apply our algorithm to such a function then in general one of the gradient components will be much smaller in magnitude than the other and the motion of the particle is initially such as to ignore the smaller component. Thus we see in *Figute 2b*, for $F = 400x_1^2 + x_2^2$, that the particle initially moves more or less parallel to the x_1 axis towards the valley at $x_1 = 0$, and then starts moving along the valley towards x^* at the origin.

It is now suggested that a motion may be obtained in which the particle moves more directly towards x^* by giving equal weight to the individual acceleration components, i.e., we apply modified acceleration components a_{ki}^* given by:

$$a_{ki}^* = a_{ki}\|a_k\|/|a_{ki}| \quad i = 1, 2, \ldots, n \tag{25}$$

In *Figure 2b* the dashed lines indicate the trajectory which results from incorporating (25) into the dynamic algorithm. The particle now follows a more direct path and reaches the vicinity of x^* after 38 steps with $\|\nabla F\| = 0.025$, $x_1 = 0.0000$ and $x_2 = 0.0004$. This should be seen in comparison to the 96 steps required to yield $\|\nabla F\| = 0.022$, $x_1 = 0.0000$ and $x_2 = -0.0090$ when the unmodified algorithm is used. The effect of modification (25) is clearly spectacular for the single case considered here and this approach definitely deserves future investigation. In what follows, however, results will only be quoted for the unmodified algorithm.

Computational results

A FORTRAN IV program (see Appendix 2) embodying the basic algorithm (23)† and the time step routine (24) was written and tested on several standard functions in the optimization literature. The size of the test problems varied from $n = 2$ to $n = 150$. To assess the efficiency of the new method all the problems were run using the new program and Fletcher's FORTRAN IV program VA09A,[2] which is based on the quasi-Newton algorithm of Gill and Murray.[3] For the extended Rosenbrock functions and the homogeneous quadratic functions (see below) some of the problems were also run using the FORTRAN IV Davidon-Fletcher-Powell code of Himmelblau[4] with a golden section unimodal search. The programs were compiled using IBM's FORTRAN G compiler and all computations were performed in double precision on an IBM 370 model 148 computer. In the execution of the programs all intermediate output was suppressed and the CPU times were recorded. For both the new method and Fletcher's program the termination criterion was $\|\nabla F\| \leqslant 10^{-5}$. For the DFP program termination occurred on $\|\nabla F\| \leqslant 10^{-4}$. Unless otherwise stated the parameter settings for the new method were kept fixed at the following values: $\Delta t(\text{initial}) = 0.5$, $\delta = 1.0$, $M = 10$ and $N = 2$. The test functions used are listed below.

Test function 1 (extended Rosenbrock functions[4,5])

$$F(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$

The minimum of this function is located at $x^* = (1, 1, \ldots, 1)^T$ with minimum value $F(x^*) = 0$. For $n = 2$ we have the classic Rosenbrock parabolic valley:

$$F(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Test function 2 (cubic valley[4])

$$F(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2$$
$$x^* = (1, 1)^T, \quad F(x^*) = 0$$

† The algorithm actually used in the computations differs slightly from algorithm (23). The difference involves step 7 of (23) which was removed and inserted just after 'set $i = 0$' in the first line of step 8 with 'otherwise go to (8)' replaced by 'otherwise go to (3)'. This change has no influence on the practical efficiency of the method. Algorithm (23) as given is, however, preferable since it is more complete in allowing for the unlikely, but possible event of a starting point which coincides exactly with the minimum. The FORTRAN program listed in Appendix 2 reflects algorithm (23)

Test function 3 (Beale's function[4])

$$F(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + [2.625 - x_1(1 - x_2^3)]^2$$
$$x^* = (3, \tfrac{1}{2})^T \quad F(x^*) = 0$$

Test function 4 (Powell's function[4])

$$F(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$
$$x^* = (0, 0, 0, 0)^T \quad F(x^*) = 0$$

Test function 5 (Wood's function[4])

$$F(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1)$$
$$x^* = (1, 1, 1, 1)^T \quad F(x^*) = 0$$

Test function 6 (homogeneous quadratic[6])

$$F(x) = \sum_{i=1}^{n} i x_i^2$$
$$x^* = (0, 0, \ldots, 0)^T \quad F(x^*) = 0$$

Test function 7 (Oren's power function[5])

$$F(x) = \left[\sum_{i=1}^{n} i x_i^2\right]^2$$
$$x^* = (0, 0, \ldots, 0)^T \quad F(x^*) = 0$$

To enable a fair evaluation of the new method it was decided not to limit the tests to the classical starting points. Consequently some of the functions were tested with additional arbitrarily selected starting points. Starting points very far from x^* were also used. For example, the Rosenbrock problem was tested with a starting point $(1000, 1000)^T$ as opposed to the classical starting point $(-1.2, 1)^T$ giving a highly significant result. The computed results are tabulated in *Tables 1–7* for the following functions: function 1, $n = 2, 4$ and 24; functions 2, 3, 4 and 5; function 6, $n = 40$; and function 7, $n = 20$. The entries in the tables are from left to right: starting point x_0; for the new method: number of time steps, $F(x^*)$ and CPU time; Fletcher's program: number of iterations, number of function and gradient evaluations, $F(x^*)$ and CPU time.

In addition to the above, the following experiments were performed to determine the influence of the dimension of the problem on the efficiency of the new method. The extended Rosenbrock problem (function 1), with starting point $x_0 = (-1.2, 1, -1.2, 1, \ldots)^T$, was run for 18 different values of n ranging from $n = 2$ to $n = 100$ and the CPU time was recorded for each individual run. For the homogeneous quadratic function (function 6) the value for the starting point was fixed at $x_0 = (3, 3, 3, \ldots)^T$ and the method applied for 28 different values of n ranging from $n = 2$ to $n = 150$. For Oren's power function (function 7) the starting point was also $x_0 = (3, 3, 3, \ldots)^T$ and the program was run for 22 different values of n in the range 2 to 130. The results obtained are graphically displayed in *Figures 3a, b* and *c*. The corresponding results obtained by using Fletcher's program and the DFP method are also shown.

Table 1a Function 1, $n = 2$ (parabolic valley)

x_0^T	New method			VAO9A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(−1.2, 1)	127	4.0×10^{-12}	0.53	36	46	3.2×10^{-11}	0.37
(−8.2, 0)	196	2.0×10^{-12}	0.73	109	134	7.7×10^{-13}	0.92
(−2.547, 1.489)	202	9.3×10^{-13}	0.82	56	69	1.9×10^{-10}	0.50
(5.621, −3.635)	213	9.4×10^{-14}	0.84	87	116	1.5×10^{-11}	0.77
(−2, −2)	233	1.7×10^{-11}	0.93	60	73	4.7×10^{-9}	0.54
(6.39, −0.221)	126	2.3×10^{-12}	0.53	81	101	2.1×10^{-12}	0.72
(10, −10)	218	2.2×10^{-11}	0.89	131	172	2.7×10^{-11}	1.13
(−10, 10)	363	3.8×10^{-11}	1.42	122	156	4.3×10^{-11}	0.98
(30, −20)	278	6.2×10^{-11}	1.01	254	323	2.5×10^{-8}	1.94
(−30, −10)	161	3.7×10^{-11}	0.63	264	345	1.0×10^{-10}	2.04
(1000, −1000)	2176	6.6×10^{-11}	7.69	2491	3254	1.1×10^{-12}	19.17

Table 1b Function 1, $n = 4$

x_0^T	New method			VAO9A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(−3, −1, −3, −1)	308	2.8×10^{-13}	1.70	50	63	5.3×10^{-13}	0.82
(−3, 1, −3, 1)	313	2.6×10^{-12}	1.54	54	63	1.7×10^{-11}	0.85
(−1.2, 1, −1.2, 1)	267	1.5×10^{-12}	1.42	40	47	1.3×10^{-12}	0.67
(−1.2, 1, 1.2, 1)	343	6.6×10^{-13}	1.84	31	39	3.7014[a]	0.54
(10, −10, 10, −10)	278	9.0×10^{-12}	1.53	87	107	1.9×10^{-13}	1.36
(−30, −10, −30, −10)	291	8.4×10^{-12}	1.60	97	117	1.2×10^{-11}	1.45
(−30, −10, 30, −10)	249	6.0×10^{-14}	1.40	200	247	3.7014[a]	2.92
(−30, −10, −30, 10)	364	8.6×10^{-12}	1.97	107	131	1.2×10^{-12}	1.64
(100, −50, 50, −100)	497	1.3×10^{-12}	2.56	77	88	1.6×10^{-12}	1.18

[a] Reached a near stationary point in a neighbourhood of $(-0.77565, 0.61309, 0.38206, 0.14597)^T$

Table 1c Function 1, $n = 24$

x_0^T	New method			VAO9A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(−1.2, 1, ...)	642	3.4×10^{-13}	12.1	154	172	1.3×10^{-14}	27.5
(−8.2, 0, ...)	603	5.1×10^{-12}	11.7	259	297	9.2×10^{-14}	43.3
(−1.2, 0, ...)	735	6.6×10^{-13}	14.0	148	179	9.6×10^{-13}	24.6
(−2.547, 1.489, ...)	588	1.3×10^{-13}	11.4	184	212	7.5×10^{-13}	30.2
(1.489, −2.547, ...)	680	5.2×10^{-12}	13.0	189	218	6.5×10^{-13}	31.4
(5.621, −3.635, ...)	596	1.1×10^{-13}	11.6	163	183	5.4×10^{-13}	26.1
(−3.635, 5.621, ...)	656	2.4×10^{-12}	12.5	112	134	1.6×10^{-13}	19.2
(6.39, −0.221, ...)	439	1.4×10^{-13}	8.3	172	209	4.9×10^{-14}	28.9
(2, −2, ...)	622	1.5×10^{-13}	11.8	173	199	1.6×10^{-13}	28.9
(10, −10, ...)	659	2.2×10^{-12}	12.3	331	376	5.8×10^{-13}	54.1
(−30, 10, ...)	775	1.4×10^{-13}	14.8	397	453	3.9866[b]	65.5

[b] Reached a near stationary point in the neighbourhood of $(-1, 1, 1, \ldots)^T$

Table 2 Function 2 (cubic valley)

x_0^T	New method			VAO9A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(−1.2, 1)	185	8.8×10^{-13}	0.75	43	59	5.0×10^{-12}	0.54
(3, 3)	283	1.7×10^{-11}	1.11	131	176	1.0×10^{-10}	1.14
(8, 8)	406	1.0×10^{-14}	1.60	485	631	1.2×10^{-11}	3.68
(−10, 0)	229	2.5×10^{-10}	0.89	605	804	5.9×10^{-11}	4.71
(10, −10)	197	5.3×10^{-14}	0.82	650	853	3.7×10^{-11}	5.08
(100, −100)	14 849	1.2×10^{-11}	53.91	12 873	16 708	6.0×10^{-13}	103.57
(100, −100)	1 515	2.2×10^{-13}	5.66[c]				

[c] $\delta = 3$

Table 3 Function 3 (Beale's function)

x_0^T	New method			VA09A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(0, 0)	96	6.7×10^{-11}	0.54	10	14	7.5×10^{-9}	0.18
(0, −1)	120	5.9×10^{-11}	0.65	14	16	1.2×10^{-8}	0.22
(5, 0.8)	96	6.1×10^{-11}	0.52	17	19	5.2×10^{-10}	0.25
(8, 0.2)	156	1.3×10^{-10}	0.81	28	36	1.3×10^{-9}	0.34
(8, 0.8)	133	3.6×10^{-13}	0.71	30	34	6.0×10^{-12}	0.34
(10, −10)	151	9.1×10^{-11}	0.79	65	92	1.4×10^{-10}	0.72
(30, 30)	2330	5.3×10^{-11}	11.07		Failed		
(100, 100)	8172	1.4×10^{-10}	35.76		Failed		
(100, 100)	2006	1.3×10^{-12}	8.74[d]				

[d] $\delta = 3$

Table 4 Function 4 (Powell's function)

x_0^T	New method			VA09A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(1, 1, 1, 1)	442	1.6×10^{-9}	2.22	36	43	5.9×10^{-12}	0.60
(3, −1, 0, 1)	439	5.5×10^{-9}	2.14	33	36	1.8×10^{-7}	0.54
(10, 10, 10, 10)	1902	2.4×10^{-8}	9.01	53	58	3.1×10^{-11}	0.83
(1, 1, 1, 1)[e]	81	3.1×10^{-6}	0.48	18	23	4.3×10^{-6}	0.34
(3, −1, 0, 1)[e]	103	4.6×10^{-6}	0.60	27	30	8.4×10^{-6}	0.45
(10, 10, 10, 10)[e]	432	6.8×10^{-6}	2.15	31	36	1.6×10^{-6}	0.50

[e] $\epsilon = 10^{-3}$

Table 5 Function 5 (Wood's function)

x_0^T	New method			VA09A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(−1.2, 1, 1.2, 1)	406	1.6×10^{-12}	2.09	37	49	8.8×10^{-12}	0.63
(−3, −1, −3, −1)	337	5.9×10^{-11}	1.75	95	105	9.8×10^{-11}	1.38
(−3, 1, −3, 1)	423	4.8×10^{-12}	2.17	78	103	2.2×10^{-12}	1.19
(10, 10, 10, 10)	375	4.8×10^{-12}	1.94	45	55	1.1×10^{-12}	0.72

Table 6 Function 6, n = 40 (homogeneous quadratic)

x_0^T	New method			VA09A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(1, 1, ...)	158	5.9×10^{-12}	3.65	47	58	1.4×10^{-9}	17.97
(3, 3, ...)	523	1.8×10^{-12}	11.34	48	59	8.3×10^{-12}	17.99
(10, 5, ...)	578	3.9×10^{-12}	12.45	48	57	4.0×10^{-10}	18.66
(10, 10, ...)	688	2.5×10^{-12}	14.70	48	59	9.2×10^{-11}	17.98

Table 7 Function 7, n = 20 (Oren's power function)

x_0^T	New method			VA09A			
	Steps	$F(x^*)$	Time (s)	Iter.	F.g. eval.	$F(x^*)$	Time (s)
(1, 1, ...)	927	2.6×10^{-8}	14.52	85	99	1.6×10^{-8}	10.13
(2, 2, ...)	866	2.1×10^{-8}	13.88	57	75	2.1×10^{-8}	7.06
(3, 3, ...)	1025	1.6×10^{-8}	16.14	85	105	9.3×10^{-9}	10.51
(100, 100, ...)	1091	2.5×10^{-8}	17.00	128	156	8.4×10^{-9}	15.75

Discussion of results

In *Table 1a* the results for the classical Rosenbrock problem, $n = 2$, indicate that the new method is competitive with the quasi-Newton method for this function. Fletcher's program performs better when the starting point is relatively near x^* while the dynamic method is superior when the starting point is far from x^*. For the classical starting point $(-1.2, 1)^T$ the respective times are 0.53 s for the new method and 0.37 s for the quasi-Newton method. On the other hand for the extreme case where we choose $x_0 = (1000, -1000)^T$, the dynamic method converges within 7.69 s while Fletcher's program requires 19.17 s. The results for the extended Rosenbrock function with $n = 4$, shown in *Table 1b*, are similar. The quasi-Newton method still performs better for 'near' starting points while the dynamic method is competitive when x_0 is chosen far away from x^*. It is interesting to note that for two starting points in *Table 1b* the quasi-Newton method converges to a near stationary point ($\|\nabla F\| < 10^{-5}, F = 3.7014$) different from $x^* = (1, 1, 1, 1)^T$, while the dynamic method converges successfully to x^* in all cases. *Table 1c* shows that for the extended Rosenbrock function with $n = 24$ the efficiency of the dynamic method is superior for all 11 starting points used and does almost three times as well overall. Again we note that for one of the starting points Fletcher's program converges to a near stationary point in the neighbourhood of $(-1, 1, 1, \ldots, 1)^T$, having a function value of approximately 3.9866. The dynamic method never fails to converge to x^*.

The results for the steep cubic valley of function 2 are listed in *Table 2*. This function represents a severe challenge to a minimization algorithm. With one exception, namely 0.75 s compared to 0.54 s for the classical starting point of $(-1.2, 1)^T$, the new method performs better than the quasi-Newton method. For the 'far' starting point of $(100, -100)^T$ the respective times are 53.9 s and 103.6 s. Re-running the latter problem with the dynamic method but using $\delta = 3.0$ instead of $\delta = 1.0$ gives an even more dramatically reduced time of 5.7 s.

Table 3 lists the results for Beale's two-dimensional function. Here the quasi-Newton method is consistently better for 'near' starting points but fails for 'far' points. Again for the 'far' point of $(100, 100)^T$ the computational time for the dynamic method is drastically reduced when using a larger value for δ.

The results in *Table 4* for the Powell quartic function favours the quasi-Newton method. If, however, the stopping criterion is made less stringent with $\epsilon = 10^{-3}$ instead of 10^{-5}, then the dynamic method becomes more competitive. The probable reason for the relative poor performance of the new method, for this problem, is the fact that the function has a flat minimum which is naturally to the disadvantage of the dynamic method.

For Wood's function (*Table 5*) the quasi-Newton method gives faster convergence although the performance of the dynamic method, considering the small size of the problem, is still competitive and certainly no less reliable.

In *Table 6* the results are listed for the homogeneous quadratic function with $n = 40$. Here the dynamic method performs better for all four starting points considered, doing almost twice as well overall.

The results listed in *Table 7* for Oren's power function with $n = 20$ slightly favour the quasi-Newton method. This advantage, as will be shown below, is lost when the dimension of the problem is increased.

The experiments in which the efficiency of the dynamic method was studied as a function of the dimension of the problem, and of which the results are displayed in *Figures 3a, b* and *c*, probably bring to the fore the strongest advantage of the new method. This is the definite indication that, for the dynamic method the computational time required for the convergence increases roughly linearly with the dimension of the problem, as opposed to the quadratic behaviour exhibited by quasi-Newton methods.

The results for function 1, depicted in *Figure 3a*, show that while the quasi-Newton methods may perform slightly better for small values of n, this advantage is overshadowed by the superior performance of the dynamic method for large n. To put the performances in perspective we quote the results for $n = 50$. The dynamic method requires 31.4 s for convergence while the Fletcher and DFP programs require 163.4 and 311.7 s respectively. For $n = 100$ the dynamic method readily converges in 74.0 s while it is clear from *Figure 3a*, that the times that would be required for the quasi-Newton methods would be prohibitive.

Figure 3b shows a similar behaviour for the homogeneous quadratic function. For n less than 10 the quasi-Newton methods do slightly better but for large n the performance of the dynamic method is again spectacularly superior. For

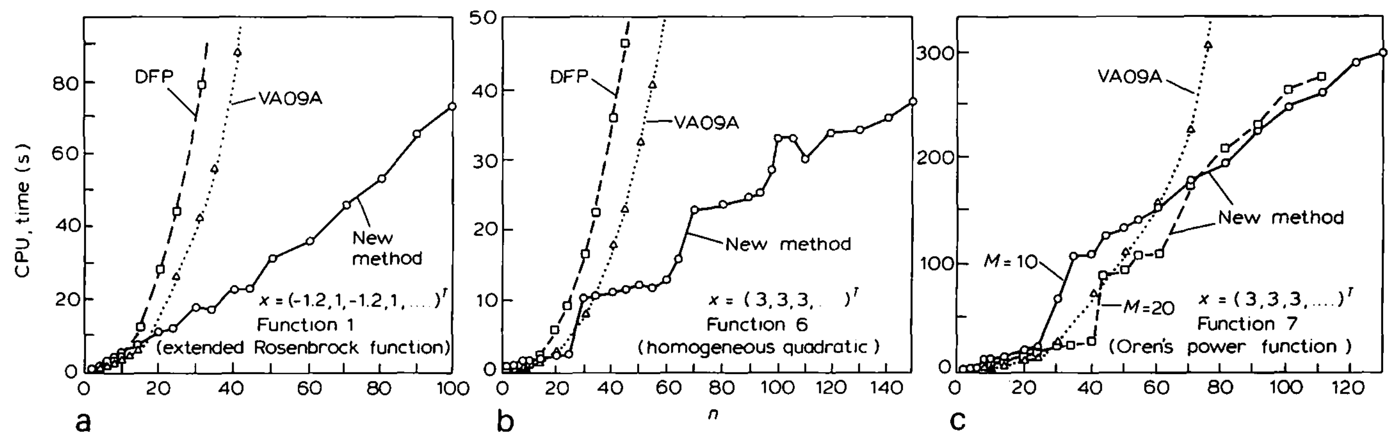

Figure 3 (a) Computational times required for convergence by new method, Fletcher's program and DFP method, plotted against dimension of problem for extended Rosenbrock function. (b) Computational times required for convergence by new method, Fletcher's program and DFP method, plotted against dimension of problem for homogeneous quadratic function. (c) Computational times required for convergence by new method and Fletcher's program, plotted against dimension of problem for Oren's power function.

$n = 70$ the dynamic method converges in 23.0 s while VA09A requires 86.5 s and the DFP program fails. For $n = 150$ the dynamic method readily converges in 38.2 s which is in fact less than twice the time required for $n = 70$. An interesting feature of *Figure 3b* is the fact that, although in general the dependence is more or less linear, it exhibits a steplike and jerky behaviour. The first two steps occurring at $n = 8$ and $n = 30$ are related to time step reductions. For n less than 8 we have $\Delta t = 0.5$ while for n in the range $8 \leqslant n < 30$ the time step is $\Delta t = 0.125$. For $n \geqslant 30$ we have $\Delta t = 0.03125$. Otherwise the jerky behaviour cannot be explained although it is probably due to the inherent random nature of the particle motion and the arbitrary character of the interfering strategy adopted.

Figure 3c depicts the behaviour for Oren's power function. The two steps occurring at $n = 12$ and $n = 30$ are again related to time step reductions. The large step at $n = 30$, which is reponsible for the fact that the dynamic method does not compare favourably with the quasi-Newton method over the range $n = 30$ to $n = 50$, suggests that the time step reduction is being applied too soon. If we now change M from 10 to 20 the picture changes significantly as can be seen in *Figure 3c* and the new method is competitive and better over the whole range. For large values of n the choice of M, all other parameters being fixed, does not appear to play an important role. For both values of M considered here the new method gives a superior performance for large n.

Conclusions

The proposed dynamic method is obviously a practical method for finding the local minimum of a general function of a large number of variables whose first derivatives can be evaluated quickly. In spite of the fact that the new method clearly represents an initial prototype dynamic method, which may certainly be improved, it already has several clear advantages. The method has proved itself to be extremely robust and reliable and with the standard parameter settings (see section on computational results) it never failed to give convergence during extensive testing done on standard test functions. In a sense it proved more reliable than Fletcher's program since it never converged to a near stationary point as happened with the quasi-Newton method, but always yielded the required local minimum. In addition the method is conceptually very straightforward and the FORTRAN program embodying the algorithm is extremely simple and easy to use. The storage requirements are minimal with only five vectors of dimension n to be stored as opposed to the storage of, for example, the estimate to the Hessian matrix required by quasi-Newton methods.

Overall the performance of the dynamic method is competitive with the quasi-Newton methods, performing particularly well for functions representing steep narrow valleys and in the cases where the starting point is far from the local minimum. In instances where the function has a relatively flat minimum the performance of the new method is not quite as efficient but no less reliable. Of great significance is the encouraging evidence that the computational time required by the new method for convergence increases roughly linearly with the dimension of the problem. Since the multiplications required per iteration is $O(n^2)$ for quasi-Newton methods it is not surprising that for large n the performance of the new method is vastly superior. One should, however, bear in mind that, for the test functions considered here, the gradient evaluations are relatively inexpensive which favours the new method. If the gradient evaluations become more expensive one may expect the dynamic method to compare less favourably for intermediate values of n, becoming more competitive and superior as the dimension increases, because of the quadratic increase in the multiplications required per iteration by the quasi-Newton methods.

The applications of the dynamic method to constrained optimization problems via the penalty function formulation would be of great interest. It is well known that the application of gradient methods based on quadratic theory becomes difficult in the case where the shape of the penalty function cannot adequately be approximated by quadratic functions. Since from a physical point of view the shape of the function should not affect the reliability of the dynamic method and in the light of its excellent performance in the case of steep functions, one may expect the new method to perform relatively well for penalty functions. An obvious extension of the present study is therefore the application of the dynamic method to penalty function formulations of constrained problems. In addition a more detailed investigation into the effect of the various parameters in the algorithm on its efficiency is also required. In particular, further attention will be given to the development of a more sophisticated time step selection routine. It may also be worthwhile investigating the application of this new approach with other integration methods such as those developed in appendix 1.

Acknowledgement

The author thanks the South African Council for Scientific and Industrial Research for financial assistance which made this research possible.

References

1 Greenspan, D. 'Discrete models', Addison-Wesley, Massachusetts, 1973
2 Fletcher, R. 'FORTRAN subroutines for minimization by quasi-Newton methods', *Rep.* R7125, AERE, Harwell, UK, 1972
3 Gill, P. E. and Murray, W. 'Quasi-Newton methods for unconstrained optimization', *J. Inst. Math. Appl.* 1972, 9, 91
4 Himmelblau, D. M. 'Applied nonlinear programming', McGraw-Hill, New York, 1972
5 Shanno, D. F. and Phua, K. H. 'Matrix conditioning and nonlinear optimization', *Math. Programming* 1978, 14, 149.
6 Shanno, D. F. and Phua, K. H. 'Numerical comparison of several variable-metric algorithms', *J. Optimization Theory Appl.* 1978, 25, 507

Appendix 1

Energy relationships for simple integration methods

The leap-frog method

We restate the integration steps of algorithm (12):

$$x_{k+1} = x_k + v_k \Delta t \tag{A1}$$

$$v_{k+1} = v_k + a_{k+1} \Delta t \tag{A2}$$

where: $a_k = -\nabla F_k$ (A3)

It follows from (A2) that:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_{k+1}\|^2 (\Delta t)^2 + 2v_k^T a_{k+1} \Delta t \tag{A4}$$

Substituting the expression for $v_k \Delta t$, obtained from (A1), into (A4) yields:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2 + 2(x_{k+1} - x_k)^T a_{k+1}$$

which, by (A3) and interchanging x_{k+1} and x_k, may be written as:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2 + 2(x_k - x_{k+1})^T \nabla F_{k+1} \tag{A5}$$

By the second mean value theorem we have:

$$F_k = F_{k+1} + (x_k - x_{k+1})^T \nabla F_{k+1} + \tfrac{1}{2}(x_k - x_{k+1})^T H(\bar{x})(x_k - x_{k+1}) \tag{A6}$$

where:

$$\bar{x} = x_{k+1} - \theta\,\Delta x_k \qquad 0 \leqslant \theta \leqslant 1$$

and H the Hessian matrix:

$$H(x) = \left[\frac{\partial^2 F(x)}{\partial x_i\,\partial x_j}\right]$$

(A6) may be rewritten as:

$$(x_k - x_{k+1})^T \nabla F_{k+1} = F_k - F_{k+1} - \tfrac{1}{2}(x_k - x_{k+1})^T \times H(\bar{x})(x_k - x_{k+1}) \tag{A7}$$

Substituting (A7) into (A5) gives:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_{k+1}\|^2(\Delta t)^2 + 2\left[F_k - F_{k+1} - \tfrac{1}{2}\Delta x_k^T H(\bar{x})\,\Delta x_k\right].$$

On rearranging the latter equation and dividing by 2 we finally obtain:

$$\tfrac{1}{2}\|v_{k+1}\|^2 + F_{k+1} = \tfrac{1}{2}\|v_k\|^2 + F_k + \tfrac{1}{2}\|a_{k+1}\|^2(\Delta t)^2 - \tfrac{1}{2}\Delta x_k^T H(\bar{x})\,\Delta x_k \tag{A8}$$

This is the energy relationship (13) used earlier in the paper.

An Euler-trapezium method

As an alternative to the leap-frog method we propose the following scheme:

$$x_{k+1} = x_k + v_k\,\Delta t \tag{A9}$$

$$v_{k+1} = v_k + \tfrac{1}{2}(a_k + a_{k+1})\,\Delta t \tag{A10}$$

where:

$$a_k = -\nabla F_k \tag{A11}$$

It follows from (A10) that:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_k'\|^2(\Delta t)^2 + 2v_k^T a_k'\,\Delta t \tag{A12}$$

where:

$$a_k' = \tfrac{1}{2}(a_k + a_{k+1}) \tag{A13}$$

Again, by substituting the expression for $v_k\,\Delta t$, obtained from (A9), into (A12) gives:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_k'\|^2(\Delta t)^2 + (x_{k+1} - x_k)^T(a_k + a_{k+1}) \tag{A14}$$

which, by (A11) may be written as:

$$\|v_{k+1}\|^2 = \|v_k\|^2 + \|a_k'\|^2(\Delta t)^2 - (x_{k+1} - x_k)^T(\nabla F_k + \nabla F_{k+1}) \tag{A15}$$

By the second mean value theorem it follows that:

$$\left.\begin{aligned} -(x_{k+1} - x_k)^T \nabla F_k &= F_k - F_{k+1} + \tfrac{1}{2}\Delta x_k^T H(\bar{x}_0)\,\Delta x_k \\ \text{and}\qquad & \\ -(x_{k+1} - x_k)^T \nabla F_{k+1} &= F_k - F_{k+1} - \tfrac{1}{2}\Delta x_k^T H(\bar{x}_1)\,\Delta x_k \end{aligned}\right\} \tag{A16}$$

where: $\bar{x}_0 = x_k + \theta_0\Delta x_k$, $0 \leqslant \theta_0 \leqslant 1$; and $\bar{x}_1 = x_{k+1} - \theta_1\Delta x_k$, $0 \leqslant \theta_1 \leqslant 1$.

Substituting equations (A16) into (A15) and rearranging finally gives the energy relationship:

$$\tfrac{1}{2}\|v_{k+1}\|^2 + F_{k+1} = \tfrac{1}{2}\|v_k\|^2 + F_k + \tfrac{1}{2}\|a_k'\|^2(\Delta t)^2 + \tfrac{1}{4}\Delta x_k^T\{H(\bar{x}_0) - H(\bar{x}_1)\}\,\Delta x_k \tag{A17}$$

It is interesting to compare this relationship with the corresponding energy relationship (A8) of the leap-frog method. If F is quadratic then H is constant and (A17) simplifies to:

$$\tfrac{1}{2}\|v_{k+1}\|^2 + F_{k+1} = \tfrac{1}{2}\|v_k\|^2 + F_k + \tfrac{1}{2}\|a_k'\|^2(\Delta t)^2 \tag{A18}$$

We then have $\Delta F_k \leqslant 0$ whenever:

$$\|v_{k+1}\|^2 \geqslant \|v_k\|^2 + \|a_k'\|^2(\Delta t)^2 \tag{A19}$$

This condition may easily be monitored.

An exact energy conserving method for quadratic functions

We notice that equation (A18) implies an increase in the total energy of order $(\Delta t)^2$ during the step k to $k+1$. We now attempt to modify the Euler-trapezium method in such a way that energy is conserved exactly in the case where F is a quadratic function. This is done by introducing a damping force equal to $-\alpha v_k$ over the interval t_k to t_{k+1} such that our integration equations become:

$$x_{k+1} = x_k + v_k\,\Delta t \tag{A20}$$

$$v_{k+1} = v_k + a_k''\,\Delta t \tag{A21}$$

where:

$$a_k'' = \tfrac{1}{2}(a_k + a_{k+1}) - \alpha v_k = a_k' - \alpha v_k \tag{A22}$$

From (A21) and (A22) it follows that:

$$\begin{aligned}\|v_{k+1}\|^2 &= \|v_k\|^2 + \|a_k''\|^2(\Delta t)^2 + 2v_k^T\{a_k' - \alpha v_k\}\,\Delta t \\ &= \|v_k\|^2 + \|a_k''\|^2(\Delta t)^2 + 2v_k^T a_k'\,\Delta t - 2\alpha\|v_k\|^2\,\Delta t\end{aligned} \tag{A23}$$

As before, since:

$$2v_k^T a_k'\,\Delta t = -(x_{k+1} - x_k)^T(\nabla F_k + \nabla F_{k+1}) \tag{A24}$$

it follows by the second mean value theorem that:

$$2v_k^T a_k'\,\Delta t = 2F_k - 2F_{k+1} + \tfrac{1}{2}\Delta x_k^T\{H(\bar{x}_0) - H(\bar{x}_1)\}\,\Delta x_k \tag{A25}$$

By substituting (A25) into (A23) and assuming F to be quadratic (H constant), we finally obtain the following energy relationship:

$$\tfrac{1}{2}\|v_{k+1}\|^2 + F_{k+1} = \tfrac{1}{2}\|v_k\|^2 + F_k + \tfrac{1}{2}\|a_k''\|^2(\Delta t)^2 - \alpha\|v_k\|^2\,\Delta t$$

i.e.

$$E_{k+1} = E_k + \tfrac{1}{2}\|a_k''\|^2(\Delta t)^2 - \alpha\|v_k\|^2\,\Delta t \tag{A26}$$

The total energy is therefore conserved from step to step provided we choose α such that:

$$\|a_k''\|^2\,\Delta t - 2\alpha\|v_k\|^2 = 0 \quad v_k \neq \mathbf{0} \tag{A27}$$

Clearly α will, in general, have a different value, $\alpha = \alpha_k$, for each step taken and its value is given by the solution to (A27). Using (A22) the above becomes:

$$\{\|a_k'\|^2 + \alpha_k^2\|v_k\|^2 - 2\alpha_k a_k'^T v_k\}\,\Delta t - 2\alpha_k\|v_k\|^2 = 0$$

which may be written as:

$$A\alpha_k^2 - B\alpha_k + C = 0 \tag{A28}$$

where:

$$A = \|v_k\|^2\,\Delta t \qquad B = 2\{a_k'^T v_k\,\Delta t + \|v_k\|^2\}$$
$$C = \|a_k'\|^2\,\Delta t \tag{A29}$$

The solutions of quadratic equation (A28) are:

$$\alpha_k = \{B \pm \sqrt{B^2 - 4AC}\}/2A \tag{A30}$$

We require the root which will result in the smallest perturbation to a_k', i.e., we choose the root of smallest absolute magnitude. This choice depends on the sign of B. If $B > 0$ we select the negative sign in (A30) and if $B < 0$ we choose the positive sign (the choice of a real root α_k depends, of course, on Δt being sufficiently small such that $B^2 - 4AC > 0$).

We conclude by summarizing that equations (A20)-(A22) together with the choice:

$$\alpha_k = \{B - \sqrt{B^2 - 4AC}\}/2A \quad \text{if} \quad B > 0$$

and

$$\alpha_k = \{B + \sqrt{B^2 - 4AC}\}/2A \quad \text{if} \quad B < 0$$

constitute an energy conserving integration method for cases where F is a quadratic function.

Appendix 2

```
C$JOB
C*****************************************************************************C
C  PROGRAM IMPLEMENTING DYNAMIC METHOD FOR UNCONSTRAINED MINIMIZATION.   C
C       BY J. A. SNYMAN , DEPT. OF APPLIED MATHEMATICS, UNIV. OF PRETORIA.      C
C             FORTRAN IV PROGRAM , VERSION : LFOP1 : MAY 1981.                  C
C     PROGRAM FINDS MINIMUM OF FUNCTION(FUNC) OF N VARIABLES(X(I),I=1,N).       C
C       GRADIENT VECTOR(GF(I),I=1,N) SUPPLIED BY SUBROUTINE DER(N,X,GF).        C
C            FUNCTION VALUE SUPPLIED BY SUBROUTINE FUN(N,X,FUNC).               C
C              SUBROUTINES DER AND FUN MUST BE PROVIDED BY USER.                C
C   EXAMPLE GIVEN HERE IS FOR ROSENBROCK'S EXTENDED FUNCTION WITH N=24.         C
C *************************************************************************** C
C                                                                               C
        IMPLICIT REAL*8 (A-H,O-Z),INTEGER(I-N)
        DOUBLE PRECISION X(100),V(100),GF(100),VO(100),XO(100)
C                                                                               C
C *************************************************************************** C
C         IPRINT CONTROLS PRINTING : PRINTING OCCURS EVERY |IPRINT| STEPS.      C
C               IPRINT=KMAX+1:PRINTING ON STEP 0 AND ON EXIT ONLY.              C
C            VALUES OF X SUPPRESSED ON INTERMEDIATE STEPS IF IPRINT.LT.O        C
C *************************************************************************** C
C                                                                               C
        IPRINT= 50
C                                                                               C
C *****SET STARTING POINT **************************************************** C
C                                                                               C
        N=24
        N1=N/2
        DO 1 I=1,N1
        X(2*I-1)=-1.2D0
      1 X(2*I)=1.0D0
C                                                                               C
C *****SET IM ,IN AND IX : INITIALIZE COUNTERS ******************************** C
C                                                                               C
        IM=10
        IN=2
        IX=2
        IS=0
        ID=0
        IND=0
        KOUNT=0
C                                                                               C
C *****SET TO MAXIMUM NUMBER OF STEPS ***************************************** C
C                                                                               C
        KMAX=1000
C                                                                               C
C *****SET CONVERGENCE CRITERION FOR NORM OF GRADIENT VECTOR ************ C
C                                                                               C
```

```
      EG=1.0D-5
C                                                                          C
C *****SET MAXIMUM STEP SIZE ************************************************ C
C                                                                          C
      DELT= 1.0D0
C                                                                          C
C *****SET INITIAL TIME STEP ************************************************ C
C                                                                          C
      DT=0.5D0
    5 CALL DER(N,X,GF)
      DO 6 I=1,N
    6 V(I)=-GF(I)*DT/2
      EPS=0.D0
      DO 7 I=1,N
    7 EPS=EPS+GF(I)**2
      EPS=EPS**0.5D0
      CALL FUN(N,X,FUNC)
      VR=0.D0
      DO 8 I=1,N
    8 VR=VR+V(I)**2
      VR=VR**0.5D0
C                                                                          C
C *****PRINT INITIAL GRAD. NORM , TIME STEP & FUNCTION VALUES *****************C
C                                                                          C
      WRITE(6,55) KOUNT ,EPS ,DT
      WRITE (6,50) FUNC
C                                                                          C
C *****PRINT STARTING POINT VECTOR ******************************************* C
C                                                                          C
      WRITE (6,100) (X(I),I=1,N)
   11 DX=VR*DT
      IF  (DX.LT.DELT) GO TO 12
      IS=IS+1
      DX=DELT
      GO TO 13
   12 IS=0
   13 VM=DX/DT
      IF (VR.EQ.O.ODO) GO TO 16
      DO 14 I=1,N
   14 V(I)=V(I)*VM/VR
      IF (IS.LT.IM) GO TO 16
      ID=ID+1
      IF (ID.GT.IN) GO TO 16
      DT=DT/4
      DO 15 I=1,N
      X(I)=(X(I)+XO(I))/2
   15 V(I)=(V(I)+VO(I))/4
      VM=0.D0
      DO 26 I=1,N
   26 VM=VM+V(I)**2
      VM=VM**0.5D0
      IS=0
   16 DO 17 I=1,N
      XO(I)=X(I)
      VO(I)=V(I)
      X(I)=X(I)+V(I)*DT
   17 CONTINUE
   18 CALL DER(N,X,GF)
      DO 19 I=1,N
   19 V(I)=V(I)- GF(I)*DT
      KOUNT=KOUNT+1
      VR=0.D0
      EPS=0.D0
      DO 20 I=1,N
      VR=VR+V(I)**2
   20 EPS=EPS+GF(I)**2
      VR=VR**0.5D0
      EPS=EPS**0.5D0
      IF (IPRINT.GT.KMAX) GO TO 63
      IF (MOD(KOUNT,IPRINT).NE.0) GO TO 63
C                                                                          C
C *****PRINT STEP NUMBER ,NORM OF GRADIENT VECTOR AND TIME STEP ************* C
C                                                                          C
```

```
   44 WRITE(6,55)KOUNT,EPS,DT
C                                                                          C
C ***** PRINT VARIABLES X(I) ****************************************************** C
C                                                                          C
      IF (IPRINT.LT.0) GO TO 63
      WRITE(6,100)(X(I),I=1,N)
   50 FORMAT(' ','FUNCTION VALUE=',D12.4)
   55 FORMAT(' ','STEP=',I6,5X, 'GRADIENT NORM =',D12.4,5X, 'TIME STEP=', D12.4)
   63 IF (KOUNT.GT.KMAX) GO TO 90
      IF(EPS.LF.EG) GO TO 90
   64 IF ( VR .GT.VM ) GO TO 80
      DO 65 I=1,N
   65 X(I)=(XO(I)+X(I))/2
      IND=IND+1
      IF (IND.LE.IX)GO TO 67
      DO 66 I=1,N
   66 V(I)=0.D0
      IX=1
      GO TO 69
   67 FAC=0.5D0
      DO 68 I=1,N
   68 V(I)=(V(I)+VO(I))*FAC/2
   69 DO 70 I=1,N
   70 VO(I)=V(I)
      VM=0.0D0
      DO 71 I=1,N
   71 VM=VM+V(I)**2
      VM=VM**0.5DO
      GO TO 18
   80 IND=0
      GO TO 11
   90 CALL FUN(N,X,FUNC)
C                                                                          C
C ***** PRINT FINAL STEP NO., FUNC. & GRAD. NORM VALUES & TIME STEP ************* C
C                                                                          C
   99 WRITE(6,55) KOUNT,EPS ,DT
      WRITE(6,50) FUNC
C                                                                          C
C ***** PRINT VARIABLES X(I) ****************************************************** C
C                                                                          C
      WRITE(6,100)(X(I),I=1,N)
  100 FORMAT(5D16.8)
      STOP
      END

      SUBROUTINE DER(N,X,GF)
C                                                                          C
C ***** CALCULATES FIRST DERIVATIVES OF ROSENBROCK'S EXTENDED FUNCTION ***** C
C                                                                          C
      IMPLICIT REAL*8 (A-H,O-Z),INTEGER(I-N)
      DIMENSION X(N),GF(N)
      M=N-1
      GF(1)=400*X(1)**3-400*X(1)*X(2)+2*X(1)-2
      IF (M.EQ.1) GO TO 2
      DO 1 I=2,M
    1 GF(I)=400*X(I)**3-400*X(I+1)*X(I)+202*X(I)-200*X(I-1)**2-2
    2 GF(N)=200*X(N)-200*X(M)**2
      RETURN
      END

      SUBROUTINE FUN(N,X,F)
C                                                                          C
C ***** CALCULATES FUNCTION VALUE OF ROSENBROCK'S EXTENDED FUNCTION ******* C
C                                                                          C
      IMPLICIT REAL*8 (A-H, O-Z),INTEGER(I-N)
      DIMENSION X(N)
      F=0.0D0
      M=N-1
      DO 3 I=1,M
    3 F=F+100*(X(I+1)-X(I)**2)**2+(1-X(I))**2
      RETURN
      END
```