A Global Optimization Algorithm Using Stochastic Differential Equations

FILIPPO ALUFFI-PENTINI
Rome University
VALERIO PARISI
2nd Rome University
and
FRANCESCO ZIRILLI
Rome University

SIGMA is a set of FORTRAN subprograms for solving the global optimization problem, which implements a method founded on the numerical solution of a Cauchy problem for a stochastic differential equation inspired by statistical mechanics.

This paper gives a detailed description of the method as implemented in SIGMA and reports the results obtained by SIGMA attacking, on two different computers, a set of 37 test problems which were proposed elsewhere by the present authors to test global optimization software.

The main conclusion is that SIGMA performs very well, solving 35 of the problems, including some very hard ones.

Unfortunately, the limited results available to us at present do not appear sufficient to enable a conclusive comparison with other global optimization methods.

Categories and Subject Descriptors: G.1.6 [**Numerical Analysis**]: Optimization; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*; *certification and testing*

General Terms: Algorithms, Languages, Theory, Verification

Additional Key Words and Phrases: Global optimization, stochastic differential equations.

1. INTRODUCTION

In [4] a method for solving the global optimization problem was proposed. The method associates a stochastic differential equation with the function whose global minimizer we are looking for.

The research reported in this paper has been made possible through the support and sponsorship of the United States Government through its European Research Office of the U.S. Army under contract DAJA-37-81-C-0740 with the University of Camerino, Camerino, Italy.

Authors' present addresses: F. Aluffi-Pentini, Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate, Università di Roma, "La Sapienza," Via A. Scarpa 10, 00161, Roma, Italy; V. Parisi, Dipartimento di Fisica, 2ª Università di Roma "Tor Vergata," Via Orazio Raimondo, La Romanina, 00173 Roma, Italy; F. Zirilli, Dipartimento di Matematica, Università di Roma "La Sapienza," Piazzale Aldo Moro, 2, 00185 Roma, Italy.

The stochastic differential equation is a stochastic perturbation of a "steepest descent" ordinary differential equation and is inspired by statistical mechanics. In [4] the problem of the numerical integration of the stochastic equation introduced was considered, and a suitable "stochastic" variation of the Euler method was suggested.

SIGMA is a set of FORTRAN subprograms implementing the above method.

In Section 2 we describe the method as implemented in SIGMA, in Section 3 we give a general description of the method and some details on the implementation, in Section 4 we present some numerical experience on test problems, and in Section 5 we offer conclusions.

The SIGMA package and its usage are described in the accompanying algorithm.

2. THE METHOD

Let $\mathbb{R}^N$ be the N-dimensional real euclidean space, and let $f: \mathbb{R}^N \to \mathbb{R}$ be a real valued function, regular enough to justify the following considerations.

In this paper we consider the problem of finding a global minimizer of f, that is, the point $x^* \in \mathbb{R}^N$ (or possibly one of the points) such that

$$f(x^*) \leq f(x) \qquad \forall x \in \mathbb{R}^N, \tag{2.1}$$

and we propose a method introduced in [4], inspired by statistical mechanics, to compute numerically the global minimizers of f by following the paths of a stochastic differential equation.

The value of the global optimization problem both in mathematics and in many applications is well known and will not be discussed here. We just want to remark here that the root-finding problem for the system $g(x) = 0$, where $g: \mathbb{R}^N \to \mathbb{R}^N$, can be formulated as a global optimization problem considering the function $F(x) = \| g(x) \|_2^2$, where $\| \cdot \|_2$ is the Euclidean norm in $\mathbb{R}^N$.[1]

Despite its importance and the efforts of many researchers, the global optimization problem is still rather open and there is a need for methods with a solid mathematical foundation and a good numerical performance.

Much more satisfactory is the situation for the problem of finding the local minimizers of f, for which a large body of theoretical and numerical results exists; see for instance [7] and [12] and the references given therein.

Ordinary differential equations have been used in the study of the local optimization problem or of the root-finding problem by several authors; for a review see [15].

The above methods usually obtain the local minimizers or roots by following the trajectories of suitable ordinary differential equations. However, since the property (2.1) of being a global minimizer is a global one, that is, it depends on

[1] The present authors have considered this idea both from the mathematical point of view (for a review see [15]) and from the viewpoint of producing good software (see [1] and [2]). The method implemented in [1] and [2] is inspired by classical mechanics, uses ordinary differential equations, and can be regarded as a method for global optimization.

the behavior of f at each point of $\mathbb{R}^N$, and the methods that follow a trajectory of an ordinary differential equation are local, that is, they depend only on the behavior of f along the trajectory, there is no hope of building a completely satisfactory method for global optimization on the basis of ordinary differential equations.

The situation is different if we consider a suitable stochastic perturbation of an ordinary differential equation as explained in the following.

Let us first consider the (Ito) stochastic differential equation

$$d\xi = -\nabla f(\xi)dt + \epsilon dw \tag{2.2}$$

where ∇f is the gradient of f and $w(t)$ is a standard N-dimensional Wiener process, $\epsilon \in \mathbb{R}$, $\xi \in \mathbb{R}^N$.

In statistical mechanics, eq. (2.2) is known as the Smoluchowski–Kramers equation [14]; this equation is a singular limit of Langevin's second-order equation when the inertial (i.e., second-order) term is neglected.

The Smoluchowski–Kramers equation has been extensively used by solid state physicists and chemists to study physical phenomena such as atomic diffusion in crystals or chemical reactions. In these applications eq. (2.2) represents diffusion across potential barriers under the stochastic force ϵdw, where $\epsilon = (2kT/m)^{1/2}$, T is the absolute temperature, k is the Boltzmann constant, m is a suitable mass coefficient, and f is the potential energy.

From an optimization point of view eq. (2.2) can be viewed as the steepest-descent differential equation perturbed by adding a stochastic "white noise" term.

We assume that the minimizers of f are isolated and nondegenerate, and that

$$\lim_{\|x\|_2 \to \infty} f(x) = +\infty \tag{2.3}$$

in such a way that

$$\int_{\mathbb{R}^N} e^{-\alpha^2 f(x)}\, dx < \infty, \qquad \forall \alpha \in (\mathbb{R} \setminus \{0\}). \tag{2.4}$$

The above assumptions on the behavior of f at infinity are not overly restrictive: They are, for example, satisfied by any function f that grows at least as $(\| x \|_2)^\beta$, $\beta > 0$, when $| x |_2$ goes to infinity.

The use of eq. (2.2) for optimization purposes is suggested by the following facts.

It is well known that if $\xi^\epsilon(t)$ is the solution process of (2.2) starting from an initial point x_0, the probability density function $p^\epsilon(t, x)$ of $\xi^\epsilon(t)$ (which is the solution of the suitable Fokker–Planck partial differential equation) approaches as $t \to \infty$ the stationary solution given by the limit density

$$p_\infty^\epsilon(x) = A_\epsilon e^{-(2/\epsilon^2)f(x)} \tag{2.5}$$

where A_ϵ is a normalization constant. The way in which $p^\epsilon(t, x)$ for a class of one-dimensional systems approaches $p_\infty^\epsilon(x)$ has been studied in detail by considering the spectrum of the corresponding Fokker–Planck operators in [5].

We note that p_∞^ϵ is independent of x_0 and that, as $\epsilon \to 0$, p_∞^ϵ becomes more concentrated at the global minimizers of f. That is,

$$\lim_{t\to\infty} \xi^\epsilon(t) = \xi_\infty^\epsilon \qquad \text{in law} \tag{2.6}$$

where ξ_∞^ϵ has a probability density given by (2.5) and

$$\lim_{\epsilon\to 0} \xi_\infty^\epsilon = \xi_\infty^0 \qquad \text{in law} \tag{2.7}$$

where ξ_∞^0 is a random variable having as its probability density a weighted sum of Dirac's deltas concentrated at the global minimizers of f. For example if $N = 1$ and f has two global minimizers x_1 and x_2, with $(d^2f/dx^2)(x_i) = c_i > 0$, $i = 1, 2$, we have (in a distribution sense)

$$\lim_{\epsilon\to 0} p_\infty^\epsilon(x) = \gamma\delta(x - x_1) + (1 - \gamma)\delta(x - x_2)$$

where

$$\gamma = (1 + \sqrt{c_1/c_2})^{-1}. \tag{2.8}$$

In order to obtain the global minimizers of f it seems natural to think of performing simultaneously the limit in t (i.e., eq. (2.6)) and the limit in ϵ (i.e., eq. (2.7)); this suggests trying to obtain a global minimizer of f as the asymptotic value, as $t \to \infty$, of a sample trajectory of the stochastic differential equation obtained by replacing the constant noise coefficient ϵ in eq. (2.2) with a time-varying noise coefficient $\epsilon(t)$, which tends to zero.

We therefore consider the equation

$$d\xi = -\nabla f(\xi)dt + \epsilon(t)dw \tag{2.9}$$

with initial condition

$$\xi(0) = x_0 \tag{2.10}$$

where

$$\lim_{t\to\infty} \epsilon(t) = 0. \tag{2.11}$$

In physical terms condition (2.11) means that the temperature T is decreased to 0 (absolute zero) when $t \to \infty$, that is, the system is being "frozen."

In optimization terms condition (2.11) means that the stochastic perturbation, which is suggested by the need for preventing the steepest descent trajectory from being captured by local minima, must asymptotically vanish in order to perform the final approach to a (hoped-for global) minimum.

Since we want to end up in a global minimizer of f, that is, a global minimizer of the (potential) energy, the system has to be frozen very slowly (adiabatically).

This means that, in order for the solution trajectory $\xi(t)$ of problem (2.9), (2.10) to become concentrated around a global minimizer of f, it is necessary that $\epsilon(t)$ go to zero very slowly as $t \to \infty$. The way in which $\epsilon(t)$ must go to zero depends on the function f. In particular, it depends on the highest barrier in f

that should be overcome in reaching the global minimizers. This dependence has been studied in [4] using the adiabatic perturbation theory.[2]

In this paper we restrict our attention to the numerical implementations of the previous ideas, that is, the computation of the global minimizers of f, by following the paths defined by (2.9) and (2.10) and disregarding mathematical problems such as the difference between the convergence in law of $\xi(t)$ to a random variable concentrated at the global minimizers of f, and the convergence with probability of one of the paths of $\xi(t)$ to the global minimizers of f.

We consider the problem of how to compute numerically these paths, keeping in mind that we are not really interested in the paths, but only in their asymptotic values.

We discretize (2.9) and (2.10) using the Euler method; that is, $\xi(t_k)$ is approximated by the solution ξ_k of the following finite-difference equations:

$$\xi_{k+1} - \xi_k = -h_k \nabla f(\xi_k) + \epsilon(t_k)(w_{k+1} - w_k), \qquad k = 0, 1, 2, \ldots \tag{2.12}$$

$$\xi_0 = x_0 \tag{2.13}$$

where

$$t_0 = 0, \qquad t_k = \sum_{i=0}^{k-1} h_i, \qquad h_k > 0,$$

and

$$w_k = w(t_k), \qquad k = 0, 1, 2, \ldots$$

The choice of the Euler method, which is computationally cheap but slowly convergent, is suggested by the fact that the noise coefficient $\epsilon(t)$ should decrease to zero very slowly as $t \to \infty$, and, therefore, a large number of steps must be computed anyway. Thus cheap steps are clearly useful, and there is no need to resort to more expensive methods designed for rapid convergence.

The speed of convergence of the Euler method could become slower than the decrease of $\epsilon(t)$ zero only in extremely ill-conditioned problems, degrading the performance of the method. In order to cope with such situations, the algorithm implementation provides some form of rescaling (see Section 3.2.12).

A further step can be made in the direction of avoiding unnecessary computational effort if we observe that a correct numerical computation of the gradient in eq. (2.12) (which requires $N + 1$ function evaluations if a forward-difference approximation is used) is not really needed, since we add a stochastic term to it.

---

[2] We are indebted to A. H. G. Rinnooy Kan for bringing to our attention, after the submission of the present paper, the existence of the work of Kirkpatric, Gelatt, and Vecchi [9] in which similar ideas are exploited in the different context of combinatorial optimization. Their "simulated annealing" method is based on the Metropolis algorithm for the numerical simulation of the behavior of a collection of atoms tending to thermal equilibrium at a given constant temperature. It consists of first "melting" the system being optimized, then slowly lowering the "temperature" by small steps, each time waiting to reach the equilibrium, and continuing according to a given empirical "annealing schedule" until the system "freezes," one hopes in the lowest energy configuration.

The main differences with respect to our method are the discrete variable nature of combinatorial optimization problems and the fact that our method looks for an optimum by following the solution trajectories of the stochastic differential equation governing a diffusion process in which the "annealing schedule" is given by a continuous decrease of the temperature parameter $\epsilon(t)$.

We therefore replace the gradient $\nabla f(\xi)$ with a "random gradient" as follows. Let r be an N-dimensional random vector of length 1 uniformly distributed on the N-dimensional unit sphere. Then for any given (nonrandom) vector $v \in \mathbb{R}^N$ its projections along r is such that

$$N \cdot \mathrm{E}(\langle r, v \rangle r) = v \tag{2.14}$$

where $\mathrm{E}(\cdot)$ is the expected value, and $\langle \cdot, \cdot \rangle$ is the Euclidean inner product in $\mathbb{R}^N$. This suggests replacing the gradient $\nabla f(\xi_k)$ with the "random gradient"

$$\gamma(\xi_k) = Nr, \nabla f(\xi_k) r. \tag{2.15}$$

The random error

$$e(\xi_k) = \gamma(\xi_k) - \nabla f(\xi_k) \tag{2.16}$$

in the computation of the gradient is a vector random variable whose components have the expected value 0 (γ is an unbiased estimator of ∇f) and variance of the same order as $| \nabla f |_2$.

We note that since $(1/N)\, \gamma(\xi_k)$ is the directional derivative in the direction r, it is computationally much cheaper (e.g., when forward differences are used, only two function evaluations are needed to approximate $\gamma(\xi)$). Therefore, the paths are computed approximating $\xi(t_k)$ with the solution ξ_k of the following difference equations:

$$\xi_{k+1} - \xi_k = -h_k \tilde{\gamma} \xi_k) + \epsilon(t_k)(w_{k+1} - w_k), \qquad k = 0, 1, 2, \ldots \tag{2.17}$$

$$\xi_0 = x_0 \tag{2.18}$$

where $\tilde{\gamma}(\xi_k)$ is a finite difference (forward or central) approximation to $\gamma(\xi_k)$.

The complete algorithm is described in the next section.

3. THE COMPLETE ALGORITHM

We give in Section 3.1 a general description of the algorithm, while implementation details are given in Section 3.2.

3.1 General Description of the Algorithm

The basic time-integration step (see eq. (2.17) and Section 3.2.1) is used to generate a fixed number N_{TRAJ} of trajectories, which start at time zero from the same initial conditions, with possibly different values of $\epsilon(0)$ (note that even if the starting values $\epsilon(0)$ are equal, the trajectories evolve differently owing to the stochastic nature of the integration steps).

The trajectories evolve (simultaneously but independently) during an "observation period" having a given duration (see Section 3.2.5) and within which the noise coefficient $\epsilon(t)$ of each trajectory is kept at a constant value ϵ_p. At the same time the values of the steplength h_k and of the spatial discretization increment Δx_k for computing the random gradient (see eq. (2.15) and Section 3.2.2) are automatically adjusted for each trajectory by the algorithm (see Sections 3.2.3 and 3.2.4).

At the end of every observation period the corresponding trajectories are compared, one of them is discarded (and will not be considered any more), and all other trajectories are naturally continued in the next observation period. In addition, one of the trajectories is selected for "branching" (see Section 3.2.6), that is, for generating a second continuation trajectory differing from the first one only in the starting values for ϵ_p and Δx_k (see Section 3.2.7), and is considered as having the same "past history" of the first one.

The set of simultaneous trajectories is considered as a single trial, which is stopped as described in Section 3.2.8 and is repeated a number of times with different operating conditions (Section 3.2.9).

The stopping criteria for the complete algorithm are described in Section 3.2.10.

The use of an admissible region for the x values is described in Section 3.2.11, scaling is described in Section 3.2.12, and criteria for numerical equality, in Section 3.2.13.

3.2 Implementation Details

3.2.1 *The Time-Integration Step.* The basic time-integration step of eq. (2.17) is used for the trajectory under consideration in the form

$$\xi_{k+1} = \xi_k - h_k\tilde{\gamma}(\xi_k) + \epsilon_p\sqrt{h_k}u_k \qquad (k = 0, 1, 2, \ldots) \qquad (3.2.1.1)$$

where h_k and ϵ_p are the current values of the steplength and of the noise coefficient (the noise coefficient has a constant value ϵ_p throughout the current observation period (Section 3.1)) u_k is a random vector sample from an N-dimensional standard Gaussian distribution, and $h_k^{1/2}u_k = w_{k+1} - w_k$ is due to the properties of the Wiener process.

We note that, neglecting the discretization error $\tilde{\gamma} - \gamma$, the magnitude of the contribution of the random error (eq. 2.16) to eq. 3.2.1.1 becomes negligible with respect to the Wiener noise term as $h_k \to 0$.

The computation of the finite-difference random gradient $\tilde{\gamma}(\xi_k)$ is described in the next section.

The basic step (3.2.1.1) is actually performed in two half steps

$$\xi_k' = \xi_k - h_k\tilde{\gamma}(\xi_k) \qquad \text{(first half step)} \qquad (3.2.1.2)$$

and

$$\xi_{k+1} = \xi_k' + \epsilon_p\sqrt{h_k}u_k \qquad \text{(second half step)} \qquad (3.2.1.3)$$

Both half steps depend on h_k, while the first depends also on the current value Δx_k of the spatial discretization increment used in computing $\gamma(\xi_k)$.

Either half step can be rejected if deemed not satisfactory, as described in Section 3.2.3.

3.2.2 *The Finite-Difference Random Gradient.* Given the current (scalar) value Δx_k of the spatial discretization increment for the trajectory under consideration, we consider the random increment vector

$$s_k = \Delta x_k \cdot r_k$$

where r_k is a random sample of a vector uniformly distributed on the unit sphere in $\mathbb{R}^N$, the forward and central differences

$$\begin{cases} \Delta^F f_k = f(\xi_k + s_k) - f(\xi_k) \\ \Delta^C f_k = \dfrac{1}{2}\,[f(\xi_k + s_k) - f(\xi_k - s_k)] \end{cases} \tag{3.2.2.1}$$

the forward- and central-difference directional derivatives

$$\tilde{\eta}_k^F = \Delta^F f_k / \Delta x_k, \qquad \tilde{\eta}_k^C = \Delta^C f_k / \Delta x_k \tag{3.2.2.2}$$

and the forward- and central-difference random gradients

$$\tilde{\gamma}_k^F = N \tilde{\eta}_k^F r_k, \qquad \tilde{\gamma}_k^C = N \tilde{\eta}_k^C r_k. \tag{3.2.2.3}$$

We use $\tilde{\gamma}_k^F$ or $\tilde{\gamma}_k^C$ for $\tilde{\gamma}(\xi_k)$ in the first half step as described in the next section.

3.2.3 *Accepting and Rejecting the Half Steps.* The computation of the first half step can be attempted with the forward- or central-difference random gradient ($\tilde{\gamma}_k^F$ and $\tilde{\gamma}_k^C$ of eq. (3.2.2.3)) as described below.

In either case the half step is accepted or rejected according to the function increment

$$\Delta' f_k = f(\xi_k') - f(\xi_k) \tag{3.2.3.1}$$

Since $\Delta' f_k$ should be nonpositive for a sufficiently small value of h_k, the half step is rejected if $\Delta' f_k$ is "numerically positive," that is, larger than a given positive small tolerance.

The second half step is rejected in the corresponding function increment

$$\Delta'' f_k = f(\xi_{k+1}) - f(\xi_k') \tag{3.2.3.2}$$

is positive and too large (greater than 100 ϵ_p^2 in the present implementation).

The sequence of attempts affects the updating of the time-integration steplength h_k and of the spatial discretization increment Δx_k as described below; the amount of the updating is described in Section 3.2.4. All attempts are with the current (i.e., updated) values of h_k and Δx_k.

The sequence of attempts is as follows:

(1) Pick up a random unit vector r_k.
   (a) Compute the random increment $s_k = \Delta x_k r_k$ for computing the random gradient (Section 3.2.2).
   (b) If s_k (and therefore Δx_k) is too small (i.e., if the square of the euclidean norm of the difference between the computed values of $\xi_k + s_k$ and ξ_k is zero because of the finite arithmetic of the machine) update (increase) Δx_k and go back to (1a).

(2) Compute the forward-difference directional derivative $\tilde{\eta}_k^F$ (eq. (3.2.2.2)).
   If the computed value of $(\tilde{\eta}_k^F)^2$ is zero (owing to the finite arithmetic) update (increase) Δx_k and go back to (1a).

(3) Compute the first half step with the forward-difference random gradient $\tilde{\gamma}_k^F$.
   Compute the first half-step function increment $\Delta' f_k$ (eq. (3.2.3.1)).
   If $\Delta' f_k \leq |\tilde{\eta}_k^F|\,\Delta x_k$ accept the first half step and jump to (5).

(4) Compute again the first half step with the central-difference random gradient $\tilde{\gamma}_k^C$ to check the appropriateness of Δx_k. Compute the function increment $\Delta' f_k$ (eq. (3.2.3.1)).
   (a) If $\Delta' f_k > |\tilde{\eta}_k^F - \tilde{\eta}_k^C| \Delta x_k$ reject the first half step, update (decrease) h_k, and go back to (1).
   (b) Otherwise accept the first half step and update (decrease) Δx_k.

(5) Update (increase) h_k. Update Δx_k.

(6) Compute the second half step. Compute the second half-step function increment $\Delta'' f_k$ (eq. (3.2.3.2)).
   (a) If $\Delta'' f_k > 100\ \epsilon_p^2$, reject the second half step, and reject also the first half step, update (decrease) h_k, and go back to (1).
   (b) Otherwise accept the whole step.

Note however that, if the same half step is rejected too many times, the half step is nevertheless accepted in order not to stop the algorithm. This is not too harmful since several trajectories are being computed, and a "bad" one will be eventually discarded (in the present implementation the bound is given explicitly for the first half step (50 times) and implicitly for the second half step (if h_k becomes smaller than 10^{-30})).

3.2.4 *The Updating of h_k and Δx_k.* The time-integration steplength h_k and the spatial discretization increment Δx_k for the trajectory under consideration are updated while performing the integration step, as described in the preceding section.

Updating is always performed by means of a multiplicative updating factor which is applied to the old value to obtain the new one.

The magnitude of the updating factors as used in the various phases of the sequence in the preceding Section, 3.2.3, is as follows:

3.2.4.1 *Updating Factors for Δx_k.*

In phase (1b):	10^6
In phase (2a):	10
In phase (4b):	10^{-4}

In phase 5 the value of the updating factor is chosen among the values 1/2, 1, 2: The choice depends on the magnitude of the current estimated function increment $\hat{\Delta} f_k = |\tilde{\eta}_k| \Delta x_k$ (where $\tilde{\eta}_k$ is the forward- or central-difference directional derivative, $\tilde{\eta}_k^F$ or $\tilde{\eta}_k^C$, as appropriate), and of the function value $f_k = f(\xi_k)$.

The choice is performed as follows: We test f_k and $\tilde{f}_k = f_k + \Delta f_k$, for numerical equality according to the relative difference criterion (see Section 3.2.13) with tolerances $\tau_{R1} = 10^{-11}$ and $\tau_{R2} = 10^{-5}$ and take the updating factor

2	if f_k and $\hat{f}_k$ are "equal" within τ_{R1}
1/2	if f_k and $\hat{f}_k$ are not "equal" within τ_{R2}
1	otherwise.

We note that the performance of the present method, unlike that of quasi-Newton methods, is not critically degraded, owing to the stochastic nature of the integration step, by a not overly accurate computation of the gradient; it is,

therefore, only necessary to prevent the space discretization increment Δx_k from becoming abnormally large or small. The above interval $(10^{-11}, 10^{-5})$ represents therefore a relatively wide "indifference region" within which no control action for Δx_k is taken, which is to be considered machine independent, since it contains both the square root and cube root of the machine precision of most computers in double precision (the square root is appropriate for forward differences, whereas the cube root is appropriate for central differences).

3.2.4.2 *Updating Factors for h_k.* In phase (4a) the updating factor for h_k is

1/1.05	for the first attempt to the first half step
1/2	for the second attempt
1/10	for all other attempts

In phase 5 the updating factor depends on the current number a of accepted time-integration steps in the current observation period and on the current total number r of half steps rejected so far in the current trial (excluding those possibly rejected while attempting the first step).

If $r > 0$ the updating factor is

1.0	(if $a \leq 2r$)
1.1	(if $2r < a \leq 3r$)
2.0	(if $3r < a$)

If $r = 0$ the updating factor is

2.0	(if $a = 1$)
10.0	(if $a > 1$)

In phase (6a) the updating factor is 0.1.

3.2.5 *Duration of the Observation Period.* The duration of the observation period numbered k_p from trial start, defined as the number N_{hp} of time-integration steps in period k_p, is computed as a function of k_p by means of a formula which must be chosen, before the algorithm starts, from among the following three formulas:

(1) $N_{hp} = 1 + [\log_2(k_p)]$ ("short" duration)

(2) $N_{hp} = [k_p^{1/2}]$ ("medium-size" duration)

(3) $N_{hp} = k_p$ ("long" duration)

where $k_p = 1, 2, \ldots$, and $[x]$ is the largest integer not greater than x.

3.2.6 *Trajectory Selection.* In order to decide, at the end of an observation period, which trajectory is to be discarded and which one should be selected for branching, we compare the trajectories on the basis of the values of their noise coefficient in the observation period and of the function values obtained from trial start.

From the viewpoint of past function values, a trajectory is considered better than another if it has attained a lower function value than the other (excluding a possible initial part common to both trajectories).

From the viewpoint of the noise coefficient ϵ_p, a trajectory with larger ϵ_p is considered better if the comparison is made in an early observation period (as long as $k_p < M_p \cdot I_b$, where k_p is the number of elapsed observation periods and M_p, I_b are defined below) and worse otherwise.

A basic partial ordering of the trajectories is first obtained on the basis of past function values, and a final total ordering is then obtained, if needed, by suitably exploiting the noise-based ordering.

The discarded trajectory is always the worst in the ordering, whereas the trajectory selected for branching is usually not the best one so as to avoid sticking in a nonglobal minimum.

Normal branching is performed on the trajectory which, in the ordering, occupies the place I_b (a given integer); exceptional branching, in which the best trajectory is selected, occurs for the first time at the end of the observation period k_{po} and then at every M_p period (k_{po} and M_p are given integers), so that exceptional observation periods are those numbered

$$k_p = k_{po} + jM_p, \qquad (j = 0, 1, 2, \ldots).$$

3.2.7 *The Second Continuation of a Branched Trajectory.* While the first (unperturbed) continuation of a trajectory that undergoes branching starts with the current values of ϵ_p and Δx_k, the second continuation starts with values obtained by means of multiplicative random updating factors applied to the current values.

The updating factor F_ϵ for ϵ_p is as follows:

For the first trial and for any trial following an unsuccessful trial:

$F_\epsilon = 10^{X-1/2}$ where X is a random sample from a standard normal distribution

For all other trials:

$F_\epsilon = 2^{Y-1/2}$ where Y is a random sample from a standard Cauchy distribution, that is, with density $p(Y) = 1/(\pi(1 + Y^2))$

The updating factor for Δx_k is

$F_{\Delta x} = 10^{3Z}$ where Z is a random sample from a standard normal distribution.

3.2.8 *Stopping Criteria for a Trial.* A trial is stopped, at the end of an observation period, if one of the following two stopping criteria is satisfied:

(1) *Uniform Stop:* The final function values of all the trajectories (except the discarded one) are "numerically equal" (possibly at different points x), that is, the maximum f_{TFMAX} and the minimum f_{TFMIN} among the final function values are numerically equal as defined in Section 3.2.13, with given tolerances τ_{ABS} and τ_{REL}.

(2) *Maximum Trial Duration:* A given maximum number N_{PMAX} of observation periods has been reached.

The stopping criteria are checked in the stated order, and checking is activated only after a minimum trial duration, that is, a minimum given number N_{PMIN} of observation periods has been reached.

The stop is considered successful only if it is a uniform stop and if the final function values can be considered numerically equal to the current best minimum function value f_{OPT} found so far from the algorithm start; that is, if f_{TFMIN} and f_{OPT} are numerically equal (see Section 3.2.13) with the same tolerances.

We note that f_{OPT} is updated whenever a lower function value is reached, and that a past trial that had a successful stop may not continue to be considered a successful trial because of the updating of f_{OPT} (Section 3.2.10).

3.2.9 *Characteristics of the Successive Trials.* The operating conditions that are changed when another trial is started are

—the seed of the random number generator
—the maximum duration of the trial
—the policy for choosing ϵ_p for the second continuation of a branched trajectory
—the value of ϵ_p at the start of the trial
—the initial point x_0.

The maximum duration of a trial, that is, the maximum number N_{PMAX} of observation periods, is obtained as follows: If the preceding trial had a uniform stop (Section 3.2.8), take the value of the preceding trial; otherwise, take a value obtained by adding to the preceding value a fixed given increment I_{NPMAX}.

The policy for selecting ϵ_p for the second continuation of a branched trajectory was described in Section 3.2.7.

The value of ϵ_p at the start of a new trial is obtained by means of a multiplicative updating factor α applied to the starting value of the preceding trial, according to the outcome (stopping condition) of the preceding trial and to the number t of trials performed from the algorithm start, as compared to the given maximum number of trials N_{TRIAL}.

Successful stop:

$$\alpha = 10^3.$$

Unsuccessful uniform stop:

$$\begin{aligned} \alpha &= 10 && \text{if } \ t < [[(2/5)N_{\mathrm{TRIAL}}]] \\ \alpha &= 10^{-4} && \text{otherwise,} \end{aligned}$$

where $[[x]]$ is the smallest integer not smaller than x.

Unsuccessful nonuniform stop:

$$\alpha = 10^{-4}.$$

The initial point x_0 is selected as follows: If $t < [[(2/5)N_{\mathrm{TRIAL}}]]$, take the value of x_0 at the algorithm start; otherwise, take $x_0 = x_{\mathrm{OPT}}$ where x_{OPT} is the current best minimizer found so far from the algorithm start.

All other initial values are those of the first trial, except the initial values of h_k and Δx_k, which are the values reached at the end of the preceding trial.

3.2.10 *Stopping Criteria for the Algorithm.* The final stopping and evaluation of the complete algorithm is based on the value of f_{OPT}, the current best minimum function value found so far from the algorithm start (possibly not at the end of a trial) and on the values of two quantities, f_{TFOPT} and K_{SUC}, which summarize the outcomes of the past trials and which are defined below.

f_{TFOPT} is the current best end-of-trial function value found so far; that is, the minimum among the values found so far for f_{TFMIN} in the uniformly stopping trials (note that f_{TFOPT} is initialized with the function values at the starting point, and that, before the first uniform stop occurs, f_{TFOPT} is set to the value of f_{TFMIN} at every trial).

K_{SUC} is the current count of successful trials, which is set to zero at the algorithm start, and is updated at every uniform stop, on the basis of comparison of the last trial to those among the preceding trials which stopped uniformly.

If the last trial is sufficiently better, that is, if its final value f_{TFMIN} is smaller than f_{TFOPT} without being "numerically equal" to it (see Section 3.2.13, usual tolerances), then K_{SUC} is reset to zero, thus forgetting the past trials; otherwise, the current value is retained.

In any case K_{SUC} is then incremented by one only if the last trial stopped successfully.

The complete algorithm is stopped, at the end of a trial, if one of the following two stopping criteria (checked in the stated order) is satisfied:

(1) *Sufficient Number of Successful Trials:* The current count K_{SUC} of successful trials has reached a given requested number of successful trials N_{SUC}.

(2) *Maximum Duration:* A given maximum number N_{TRIAL} of trials has been reached.

The successful trials, as counted by K_{SUC}, are considered "not valid" whenever f_{OPT} differs too much (owing to nonfinal function values) from f_{TFMIN}; that is, f_{OPT} and f_{TFMIN} are not numerically equal (as in Section 3.2.13, always defined with the same tolerances).

Success is claimed by the algorithm if the current count K_{SUC} of successful trials is at least one and if such trials are currently valid.

We note that each integration step can be rejected only a finite number of times, each observation period lasts a finite number of accepted integration steps, and there is a finite number of observation periods in a trial; since a finite number of trials is allowed, the algorithm will stop after a finite total number of steps and function evaluations.

3.2.11 *Admissible Region for the x Values.* In order to help the user in trying to prevent computation failures (e.g., overflow) the present implementation of the method gives the possibility of defining (for any given problem and machine dynamic range and on the basis of possible analytical or experimental evidence) an admissible region for the x values (containing, one hopes, the looked-for global minimizer) within which the function values may be safely computed. We use an N-dimensional interval

$$R_i^{\mathrm{MIN}} \leq x_{(i)} \leq R_i^{\mathrm{MAX}}, \qquad i = 1, 2, \ldots, N,$$

where the interval boundaries must be given before the trial starts.

Outside the admissible region, the function $f(x)$ is replaced by an exponentially increasing function in such a way that the values of f and of the external function are matched at the boundary of the region.

3.2.12 *Scaling.* In order to counter the possible degradation of the performance of the method on extremely ill-conditioned problems (see the comments after eq. (2.13)), some form of rescaling is performed by the algorithm as follows:

We consider (for each trajectory) the rescaled variable $\tilde{x} = Ax + b$, where A is an $N \times N$ rescaling matrix and $b \in \mathbb{R}^N$ is a bias vector, and, instead of $f(x)$, we minimize, with respect to x, the function $\tilde{f}(x) = f(\tilde{x}) = f(Ax + b)$, and we try to counter the ill conditioning of $\tilde{f}$ with respect to x by suitably adjusting A (and b is adjusted in order not to alter $\tilde{x}$).

The updating of A is obtained by means of an updating matrix F_A and is performed at the end of an observation period if sufficient data are available (see below) and if the number of elapsed observation periods is not less than a given number K_{pasca} and greater than $7N$.

The updating matrix F_A is computed as described below, keeping in mind that the random gradients are the only simply usable data on the behavior of $\tilde{f}$ computed by the algorithm.

Let $\gamma_{(i)}$, $i = 1, 2, \ldots, N_g$ be the column vectors of the components of all the N_g finite-difference random gradients $\tilde{\gamma}$ ($\tilde{\gamma}^{\mathrm{F}}$ or $\tilde{\gamma}^{\mathrm{C}}$) evaluated along the trajectory (also for rejected steps) from the last scaling.

If sufficient data are available (i.e., if $N_g \geq 2N^2$) we compute the average

$$\bar{\gamma} = \frac{1}{N_g} \sum_{i=1}^{N_g} \gamma_{(i)}$$

and the estimated covariance matrix

$$C = \frac{1}{N_g} \sum_{i=1}^{N_g} [(\gamma_{(i)} - \bar{\gamma})(\gamma_{(i)} - \bar{\gamma})^T],$$

which seems to be a reasonable indicator, given the available data, of the average ill conditioning of $\tilde{f}$ as having the larger eigenvalues associated with the directions along which the second directional derivative of $\tilde{f}$ is on the average larger.

Let λ_1 be the largest eigenvalue of the (symmetric and nonnegative definite) matrix C.

We adopt the updating matrix

$$F_A = \beta \lambda_1 I - C$$

where I is the $N \times N$ identity matrix, $\beta > 1$ ($\beta = 1.3$ in the present implementation), and we obtain the updated value A' of A by means of the formula

$$A' = \alpha A F_A$$

where α is a normalization factor such that the sum of the squares of the elements of A' is equal to N (as in the identity matrix).

The matrix F_A seems to be one of the reasonable choices since it is positive definite for $\beta > 1$, it has the same set of eigenvectors as C, its eigenvalue spectrum is obtained from the spectrum of C by reflection around $\lambda = \beta\lambda_1/2$, and it therefore acts in the right direction to counter the ill conditioning of $\tilde{f}$.

The magnitude of the countereffect depends on β: The adopted value has been experimentally adjusted.

The updated bias vector b' is chosen in order that the scaling at x does not alter $\tilde{x}$, that is, in order that

$$A'x + b' = Ax + b.$$

3.2.13 *Criteria for Numerical Equality.* The following two criteria are used in a number of places in the algorithm to decide if two given numbers p and q are sufficiently close to each other (within given tolerances):

(a) Relative difference criterion:

$$|p - q| \leq \tau_{\text{REL}}(|p| + |q|)/2.$$

(b) Absolute difference criterion:

$$|p - q| \leq \tau_{\text{ABS}},$$

where τ_{REL} and τ_{ABS} are given nonnegative tolerances.

We simply say "p and q are numerically equal" (within tolerances τ_{REL} and τ_{ABS}) if p and q satisfy at least one of the above criteria with the given tolerances.

4. NUMERICAL TESTING

SIGMA has been numerically tested on a number of test problems run on two computers. The test problems are described in Section 4.1, the computers in Section 4.2, and some numerical results are reported in Section 4.3.

4.1 Test Problems

The set of test problems is fully described in [3], together with the initial points; the test problems are

(1) A fourth-order polynomial ($N = 1$)
(2) Goldstein sixth-order polynomial ($N = 1$)
(3) One-dimensional penalized Shubert function ($N = 1$)
(4) A fourth-order polynomial in two variables ($N = 2$)
(5) A function with a single row of local minima ($N = 2$)
(6) Six-hump camel function ($N = 2$)
(7) Two-dimensional penalized Shubert function, $\beta = 0$ ($N = 2$)
(8) Two-dimensional penalized Shubert function, $\beta = 0.5$ ($N = 2$)
(9) Two-dimensional penalized Shubert function, $\beta = 1$ ($N = 2$)
(10) A function with three ill-conditioned minima, $a = 10$ ($N = 2$)
(11) A function with three ill-conditioned minima, $a = 100$ ($N = 2$)
(12) A function with three ill-conditioned minima, $a = 1000$ ($N = 2$)
(13) A function with three ill-conditioned minima, $a = 10{,}000$ ($N = 2$)
(14) A function with three ill-conditioned minima, $a = 10^5$ ($N = 2$)
(15) A function with three ill-conditioned minima, $a = 10^6$ ($N = 2$)
(16) Goldstein–Price function ($N = 2$)
(17) Penalized Branin function ($N = 2$)
(18) Penalized Shekel function $M = 5$ ($N = 4$)
(19) Penalized Shekel function $M = 7$ ($N = 4$)

(20) Penalized Shekel function $M = 10$ ($N = 4$)
(21) Penalized three-dimensional Hartman function ($N = 3$)
(22) Penalized six-dimensional Hartman function ($N = 6$)
(23) Penalized Levy–Montalvo function, type 1 ($N = 2$)
(24) Penalized Levy–Montalvo function, type 1 ($N = 3$)
(25) Penalized Levy–Montalvo function, type 1 ($N = 4$)
(26) Penalized Levy–Montalvo function, type 2 ($N = 5$)
(27) Penalized Levy–Montalvo function, type 2 ($N = 8$)
(28) Penalized Levy–Montalvo function, type 2 ($N = 10$)
(29) Penalized Levy–Montalvo function, type 3, range 10 ($N = 2$)
(30) Penalized Levy–Montalvo function, type 3, range 10 ($N = 3$)
(31) Penalized Levy–Montalvo function, type 3, range 10 ($N = 4$)
(32) Penalized Levy–Montalvo function, type 3, range 5 ($N = 5$)
(33) Penalized Levy–Montalvo function, type 3, range 5 ($N = 6$)
(34) Penalized Levy–Montalvo function, type 3, range 5 ($N = 7$)
(35) A function with a cusp-shaped minimum ($N = 5$)
(36) A function with a global minimum having a small region of attraction, $a = 100$ ($N = 2$)
(37) A function with a global minimum having a small region of attraction, $a = 10$ ($N = 5$).

We use the above functions and the standard initial points as they are coded in the subroutined GLOMTF and GLOMIP, which are available in [3].

4.2 Test Computers

We consider two typical machines of "large" and "small" dynamic range, that is, with 11 and 8 bits for the exponent (biased or signed) of a double-precision number and a corresponding dynamic range of about $10^{\pm 308}$ and $10^{\pm 38}$. The tests were actually performed on

—A UNIVAC 1100/82 with an EXEC8 operating system (level 38R5) and a FORTRAN (ASCII) computed (level 10R1A) ("large" dynamic range), and
—A D.E.C. VAX 11/750 with VMS operating system (version 3.0) and a FORTRAN compiler (version 3) ("small" dynamic range).

4.3 Numerical Results

Numerical results of running SIGMA on the above problems and on the above machines are described below.

The easy-to-use driver subroutine SIGMA1 (described in the accompanying algorithm) was used with N_{SUC}, the requested number of successful trials (Section 3.2.10), set to 5. All numerical values used for the parameters are set in the driver SIGMA1 and in the other subroutines which are described in the accompanying algorithm.

In order to evaluate the performance of SIGMA, we consider all the cases in which the program claimed a success (output indicator IOUT > 0) or a failure (IOUT ≤ 0) and, by comparing the final point with the known solution, we identify the cases in which such a claim is clearly incorrect (i.e., success claim

when the final point is not even approximately close to the known solution or failure claim when the final point is practically coincident with the known solution). It is also meaningful to consider all the cases in which a computational failure due to overflow actually occurs at any point of the computation.

We therefore provide a "success indicator" Is defined by

Is = 1—success, correctly claimed

Is = 2—failure, correctly claimed

Is = 3—incorrect claim

Is = 4—overflow failure

and we note that effectiveness, dependability, and robustness can be estimated by looking at the numbers of correctly claimed successes, incorrect claims, and overflow failures.

The results of the testing are reported in Tables I, II, and III (including the "intermediate" results relative to $N_{SUC} = 1, 2, 3, 4$, that is, the results that one would have obtained by setting N_{SUC} equal to 1, 2, 3, or 4).

The performance of SIGMA on the UNIVAC 1100/82 is reported in Table I, where for each one of the 37 test problems we give the total number Nf of function evaluations (including the ones needed to evaluate the random gradient) and the value of the success indicator Is; the performance on the VAX 11/750 is reported in Table II.

Summarized data are given in Table III, where for each test machine we give the total number of times in which, over the whole set of test problems, the success indicator had the value 1, 2, 3, or 4, that is, the total number of correctly claimed successes, of correctly claimed failures, of incorrect claims, and of overflow failures. We note that success was always claimed by the algorithm and that no overflow occurred.

5. CONCLUSIONS

The SIGMA package presented here seems to perform quite well on the proposed test problems.

As shown in [3], some of the test problems are very hard; for example, problem 28 ($N = 10$) has a single global minimizer and a number of local minimizers of order 10^{10} in the region $|x_i| \leq 10$, $i = 1, 2, \ldots, 10$.

Table III shows that, from the point of view of effectiveness as measured by the number of correctly claimed successes, the performance of SIGMA is very satisfactory (35 out of 37 problems were correctly solved); moreover, SIGMA is remarkably machine independent (note that completely different random step sequences are generated by the algorithm on the two test machines[3]). The results of Table III also suggest that the performance of SIGMA is very satisfactory from the point of view of robustness (no overflow on both machines) and of dependability (only 2 incorrect claims on the "large" dynamic range machine when $N_{SUC} > 3$ and on the "small" dynamic range machine when $N_{SUC} > 4$). It

[3] This is due to the fact that different pseudorandom number sequences are generated on the two test machines and to the unavoidable integration effects that would occur even with identical pseudo-random number generators.

Table I. Performance Data on UNIVAC 1100/82

N_{SUC}		1		2		3		4		5	
NPROB	N	Nf	Is	Nf	Is	Nf	Is	Nf	Is	Nf	Is
1	1	3,588	1	11,467	1	23,067	1	32,520	1	58,751	1
2	1	3,254	1	9,509	1	20,893	1	32,910	1	72,015	1
3	1	8,638	1	17,741	1	23,814	1	57,187	1	67,621	1
4	2	6,594	1	15,898	1	30,589	1	69,489	1	101,633	1
5	2	12,680	1	23,221	1	38,362	1	95,423	1	104,391	1
6	2	2,697	1	8,343	1	19,660	1	57,728	1	78,090	1
7	2	32,185	1	35,256	1	49,153	1	59,983	1	139,675	1
8	2	5,600	3	347,039	1	348,301	1	359,642	1	392,466	1
9	2	6,180	3	83,625	3	470,130	1	699,767	1	701,051	1
10	2	3,596	1	6,731	1	12,958	1	61,753	1	66,855	1
11	2	3,191	1	8,384	1	23,196	1	40,808	1	56,958	1
12	2	4,799	1	7,296	1	18,902	1	29,315	1	47,216	1
13	2	7,105	1	10,287	1	20,605	1	27,838	1	43,505	1
14	2	6,671	1	10,654	1	15,102	1	31,322	1	47,051	1
15	2	7,747	1	11,631	1	16,227	1	23,587	1	38,362	1
16	2	16,021	1	26,560	1	58,401	1	67,865	1	115,350	1
17	2	2,700	1	6,670	1	14,388	1	28,275	1	80,826	1
18	4	4,674	3	16,556	3	101,828	1	209,177	1	282,950	1
19	4	4,759	3	54,559	1	131,350	1	224,028	1	306,327	1
20	4	9,955	3	90,092	1	262,616	1	278,385	1	327,392	1
21	3	3,416	1	12,520	1	27,472	1	66,044	1	86,482	1
22	6	4,729	1	10,488	1	20,318	1	36,981	1	52,364	1
23	2	11,888	1	16,660	1	32,579	1	84,168	1	92,194	1
24	3	8,099	1	36,057	1	47,619	1	69,901	1	92,104	1
25	4	11,954	1	54,212	1	71,655	1	97,724	1	191,722	1
26	5	43,083	1	284,104	1	347,056	1	450,102	1	464,611	1
27	8	2,324	3	21,124	3	75,728	3	635,990	1	654,436	1
28	10	50,975	1	426,171	1	454,808	1	474,323	1	479,817	1
29	2	25,462	1	35,675	1	98,944	1	111,447	1	167,728	1
30	3	15,734	3	113,789	1	177,970	1	257,904	1	286,273	1
31	4	11,516	3	143,757	1	208,217	1	264,834	1	296,663	1
32	5	50,911	1	176,840	1	275,852	1	357,089	1	679,442	1
33	6	53,178	1	102,652	1	272,642	1	303,267	1	454,543	1
34	7	14,594	3	298,256	1	357,878	1	409,949	1	520,641	1
35	5	33,635	1	50,348	1	70,105	1	127,091	1	183,864	1
36	2	3,102	3	10,176	3	23,283	3	72,931	3	79,481	3
37	5	6,938	3	12,469	3	25,175	3	64,639	3	92,407	3

Notes:

NPROB problem number given in Section 4.1

N problem dimension (number of variables)

Nf total number of function evaluations including the ones needed to compute the "random" gradient

N_{SUC} requested number of successful trials (see Section 3.2.10)

Is success indicator (1 = success, correctly claimed; 2 = failure, correctly claimed; 3 = incorrect claim; 4 = overflow failure)

can also be seen that increasing N_{SUC} gives a better performance at the expense of a greater number of function evaluations.

It would be interesting to compare the performance of SIGMA against other global optimization methods; we think, however, that both our results and the

Table II. Performance Data on VAX 11/750

N_{SUC}		1		2		3		4		5	
NPROB	N	Nf	Is	Nf	Is	Nf	Is	Nf	Is	Nf	Is
1	1	7,522	1	12,657	1	21,643	1	31,787	1	46,954	1
2	1	3,131	1	6,542	1	14,171	1	27,023	1	70,953	1
3	1	11,526	1	15,342	1	20,457	1	57,342	1	67,423	1
4	2	9,265	1	17,713	1	28,667	1	80,161	1	144,719	1
5	2	12,094	1	19,716	1	36,426	1	59,214	1	100,336	1
6	2	4,650	1	11,040	1	25,772	1	57,087	1	62,099	1
7	2	10,543	1	44,408	1	82,833	1	130,859	1	156,208	1
8	2	27,044	3	76,348	3	189,195	3	521,474	3	604,401	1
9	2	24,348	1	35,885	1	71,593	1	165,393	1	225,842	1
10	2	4,114	1	9,959	1	19,363	1	42,409	1	91,572	1
11	2	3,254	1	7,901	1	12,795	1	28,450	1	42,615	1
12	2	6,711	1	9,949	1	18,191	1	28,788	1	44,896	1
13	2	6,771	1	11,031	1	15,508	1	22,629	1	39,765	1
14	2	6,208	1	9,443	1	17,719	1	23,579	1	39,721	1
15	2	6,313	1	13,581	1	17,631	1	30,648	1	42,360	1
16	2	5,439	1	10,491	1	24,055	1	80,137	1	101,441	1
17	2	2,790	1	11,006	1	18,444	1	51,305	1	61,100	1
18	4	2,446	3	36,252	1	129,264	1	269,925	1	290,805	1
19	4	4,778	1	19,951	1	44,198	1	109,747	1	273,679	1
20	4	4,741	1	11,312	1	24,947	1	78,820	1	125,407	1
21	3	4,334	3	27,816	1	44,893	1	82,640	1	150,742	1
22	6	3,975	1	8,613	1	16,514	1	54,082	1	68,270	1
23	2	5,534	1	29,691	1	52,042	1	65,588	1	75,379	1
24	3	10,050	1	18,170	1	80,567	1	109,726	1	190,920	1
25	4	10,657	1	37,655	1	56,285	1	129,598	1	227,682	1
26	5	59,689	1	369,912	1	460,779	1	476,325	1	585,249	1
27	8	168,933	1	259,950	1	302,092	1	310,718	1	330,192	1
28	10	43,466	1	393,550	1	411,854	1	454,095	1	474,422	1
29	2	15,223	1	55,718	1	95,254	1	131,418	1	171,995	1
30	3	12,641	1	54,822	1	118,927	1	201,965	1	333,175	1
31	4	26,235	1	123,716	1	148,183	1	242,535	1	397,066	1
32	5	35,365	1	86,758	1	186,860	1	285,371	1	316,702	1
33	6	49,087	1	71,418	1	159,987	1	184,087	1	296,770	1
34	7	50,237	1	132,768	1	173,955	1	204,081	1	348,809	1
35	5	14,815	1	53,394	1	79,235	1	129,480	1	206,980	1
36	2	3,744	3	9,574	3	23,355	3	55,947	3	82,518	3
37	5	3,847	3	12,239	3	29,411	3	70,599	3	107,264	3

Notes:

NPROB problem number given in Section 4.1

N problem dimension (number of variables)

Nf total number of function evaluations including the ones needed to compute the "random" gradient

N_{SUC} requested number of successful trials (see Section 3.2.10)

Is success indicator (1 = success, correctly claimed; 2 = failure, correctly claimed; 3 = incorrect claim; 4 = overflow failure)

other results we are aware of (see [6], [8], [10], [11], and [13]) do not appear sufficient for enabling a conclusive comparison.

A reliable comparison—when stochastic methods are involved—should in fact be based on results averaged over a sufficient number of repeated trials in order

Table III. Summarized Performance Data

	Results on UNIVAC 1100/82					Results on VAX 11/750				
N_{SUC}	1	2	3	4	5	1	2	3	4	5
Outcome										
1	26	32	34	35	35	32	34	34	34	35
2	0	0	0	0	0	0	0	0	0	0
3	11	5	3	2	2	5	3	3	3	2
4	0	0	0	0	0	0	0	0	0	0

Notes:
N_{SUC} requested number of successful trials (Section 3.2.10)
Is success indicator (1 = success, correctly claimed; 2 = failure, correctly claimed; 3 = incorrect claim; 4 = overflow failure)

to avoid effects mainly due to the variability of sample results (as it appears to be the case, for example, for the differences in performance of SIGMA on our two tests machines).

In order to attempt a rough comparison, we could consider the results with the greatest number of common test problems, which are those in [10] and [11], where Levy and Montalvo report the results of their tunneling algorithm on 4 problems taken from the literature (problems 6 to 9 in Section 4.1) and 12 problems proposed by Levy and Montalvo (problems 23 to 34 also in Section 4.1), based on functions of essentially the same analytical form. The results, however, are not truly comparable, since Levy and Montalvo made 4 runs for each problem, starting from different initial points whose choice (first described in [11]) was different for each problem, while we made one run for each problem starting always from the center of the observation region defined in [10], that is, from the vector zero.

Moreover, Levy and Montalvo give the total number of function evaluations and of gradient evaluations separately and do not state how the gradients were computed.

If we want to attempt a comparison, we may observe what follows. As for the number of failures, Levy and Montalvo fail in about 10 percent of the runs but with no "problem-failure" (since on every problem they never fail in more than two of the four runs), while our results depend on the control parameter N_{SUC}, ranging from 6 failures on 16 problems with $N_{SUC} = 1$ to no failures with $N_{SUC} = 4$.

As for the number of function evaluations, if we assume for the sake of comparison that central-differences gradients were used by Levy and Montalvo, we see that, for $N_{SUC} = 1$, our results are comparable with their results but become more expensive with growing N_{SUC}.

This may be due in part to the fact that, as explained in more detail in [3], our method is designed to perform a truly unconstrained optimization (i.e., over $\mathbb{R}^n$), and although the restriction to a given compact n-dimensional observation interval is easily performed by adding suitable penalization functions, this does not prevent wasting some search effort outside the region of interest.

We may also point out that our method can easily cope with nondifferentiable minima as in problem 35.

We think that a really conclusive comparison is very difficult but would in any case be best performed when the algorithms to be compared are available to a third party for extensive testing on a larger and commonly accepted set of standard test problems.

ACKNOWLEDGMENTS

F. Zirilli gratefully acknowledges the hospitality and the support of the Department of Mathematical Sciences of Rice University where part of this work was done.

REFERENCES

1. ALUFFI-PENTINI, F., PARISI, V., AND ZIRILLI, F. A differential-equations algorithm for nonlinear equations. *ACM Trans. Math. Softw. 10*, 3 (Sept. 1984), 299–316.
2. ALUFFI-PENTINI, F., PARISI, V., AND ZIRILLI, F. Algorithm 617. DAFNE—A differential-equations algorithm for nonlinear equations. *ACM Trans. Math. Softw. 10*, 3 (Sept. 1984), 317–324.
3. ALUFFI-PENTINI, F., PARISI, V., AND ZIRILLI, F. Test problems for global optimization software. Tech Rep. ROM2F/85/030, Dip. di Fisica, 2ª Univ. di Roma (Tor Vergata), Dec. 1985. To appear in the *Computer J.*
4. ALUFFI-PENTINI, F., PARISI, V., AND ZIRILLI, F. Global optimization and stochastic differential equations. *J. Optim. Theo. Appl. 47* (1986), 1–16.
5. ANGELETTI, A., CASTAGNARI, C., AND ZIRILLI, F. Asymptotic eigenvalue degeneracy for a class of one dimensional Fokker–Planck operators. *J. Math. Phys. 26* (1985), 678–691.
6. BOENDER, C. G. E., RINNOOY KAN, A. H. G., TIMMER, G. T., AND STOUGIE, L. A stochastic method for global optimization. *Math. Program. 22* (1982), 125–140.
7. DENNIS, J. E., AND SCHNABEL, R. B. Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall, Englewood Cliffs, N.J., 1983.
8. DIXON, L. C. W., AND SZEGÖ, G. P., Eds. *Towards Global Optimization 2.* North-Holland, Amsterdam, The Netherlands, 1978.
9. KIRKPATRICK, S., GELATT, C. D., JR., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220* (1983), 671–680.
10. LEVY, A. V., AND MONTALVO, A. Algoritmo de tunelizaciòn para la optimizaciòn global de funciones. Comunicaciones técnicas, Serie Naranja, n. 204. IIMAS-UNAM, Mexico, D.F., 1979.
11. LEVY, A. V., AND MONTALVO, A. The tunneling algorithm for the global minimization of functions. *SIAM J. Sci. Stat. Comput. 6* (1985), 15–29.
12. POWELL, M. J. D., Ed. *Nonlinear Optimization 1981.* Academic Press, London, 1982.
13. RINNOOY KAN, A. H. G., AND TIMMER, G. T. Stochastic methods for global optimization. Tech. Rep. 8317/0, Erasmus Univ., Rotterdam, The Netherlands, 1984.
14. SCHUSS, Z. *Theory And Applications Of Stochastic Differential Equations.* J. Wiley, New York, 1980, Chap. 8.
15. ZIRILLI, F. The use of ordinary differential equations in the solution of nonlinear systems of equations. In *Nonlinear Optimization 1981*, M. J. D. Powell, Ed., Academic Press, London, 1982, 39–47.

Received January 1985; revised April 1987; accepted April 1988