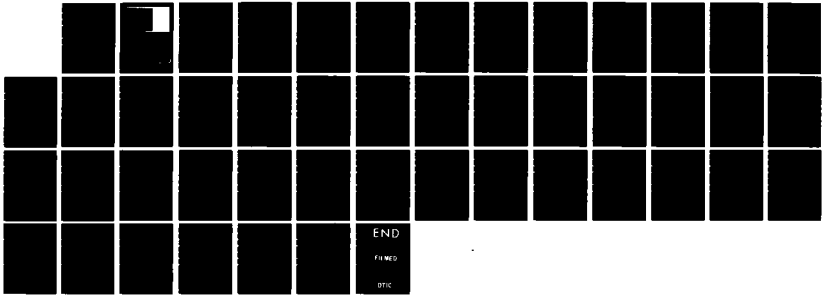
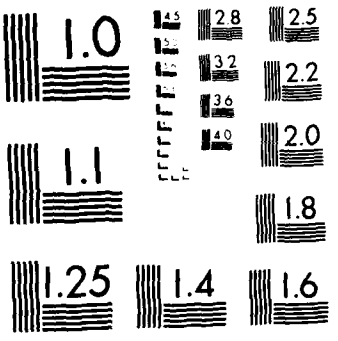
AD-A153 526 A GLOBAL OPTIMIZATION ALGORITHM USING STOCHASTIC DIFFERENTIAL EQUATIONS..(U) WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER F ALUFFI-PENTINI ET AL. 1/1

UNCLASSIFIED FEB 85 MRC/TSR-2791 DAAG29-80-C-0041 F/G 9/2 NL





MRC Technical Summary Report #2791

A GLOBAL OPTIMIZATION ALGORITHM USING STOCHASTIC DIFFERENTIAL EQUATIONS

Filippo Aluffi-Pentini, Valerio Parisi
and Francesco Zirilli

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53705

February 1985

(Received December 13, 1984)

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

A GLOBAL OPTIMIZATION ALGORITHM USING STOCHASTIC DIFFERENTIAL EQUATIONS

Filippo Aluffi-Pentini[1], Valerio Parisi[2], and Francesco Zirilli[3]

Technical Summary Report #2791
February 1985

ABSTRACT

SIGMA is a set of FORTRAN subprograms for solving the global optimization problem, which implement a method founded on the numerical solution of a Cauchy problem for stochastic differential equations inspired by quantum physics.

This paper gives a detailed description of the method as implemented in SIGMA, and reports on the numerical tests which have been performed while the SIGMA package is described in the accompanying Algorithm.

The main conclusions are that SIGMA performs very well on several hard test problems; unfortunately given the state of the mathematical software for global optimization it has not been possible to make conclusive comparisons with other packages.

AMS (MOS) Subject Classifications: 65K10, 60H10, 49D25

Key Words: Algorithms, Theory, Verification, Global Optimization, Stochastic Differential Equations,

Work Unit Number 5 (Optimization and Large Scale Systems)

---

[1]Dipartimento di Matematica, Università di Bari, 70125 Bari (Italy).
[2]Istituto di Fisica, 2[a] Università di Roma "Tor Vergata", Via Orazio Raimondo, 00173 (La Romanina) Roma (Italy).
[3]Istituto di Matematica, Università di Salerno, 84100 Salerno (Italy).

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041 and the U. S. Government through its European Research Office of the U. S. Army under Contract n. DAJA-37-81-C-0740 with the University of Camerino, Italy.

SIGNIFICANCE AND EXPLANATION

The paper reports about a new and very successful method for finding a "global" (or "absolute") minimum of a function of N real variables, i.e. the point x in N-dimensional space (or possibly, one of the points) such that not only the function increases if one moves away from x in any direction, ("local" or "relative" minimum), but also such that no other point exists where f has a lower value.

The method, which was first proposed by the present authors in a paper which is to appear in the Journal of Optimization Theory and Applications, is based on ideas from statistical mechanics, and looks for a point of global minimum by following the solution trajectories of a stochastic differential equation representing the motion of a particle (in N-space) under the action of a potential field and of a random perturbing force.

The paper gives a detailed description of the complete algorithm based on such a method, and summarizes the results of extensive numerical testing of the FORTRAN program implementing the algorithm (the FORTRAN program is described in a companion paper of the same authors: Algorithm SIGMA. A Stochastic-Integration Global Minimization Algorithm).

The tests have been performed by running the program on an extensive set of carefully selected test problems of varying difficulty, and the performance has been remarkably successful, even on very hard problems (e.g. problems with a single point of global minimum and up to about 10^{10} points of local minimum.)

The method is now being successfully tested on some real-world problems in applied chemistry, concerning the analysis of complex molecules, where one looks for spatial patterns which are not only stable (local minima of potential energy), but have also an absolute minimum of the potential energy.

More generally there are many problems in which the solution depends on the values of several parameters, and the quality of the solution can be measured by a single "performance figure" (which is therefore a function of the parameters), e.g. a cost, or a loss, or a cost/effectiveness ratio, which should be low, or a gain, a utility, which should be high.

In such situations the method can be usefully applied if one is not satisfied by finding a "sub-optimal" solution, i.e. a solution which is the best among many other solutions, but one requires a truly optimal solution, i.e. the best among all possible solutions.

It is finally to be noted that the majority of the optimization methods presently available deal with the local optimization problem, and that no methods of comparable power seem to be available in the field of global optimization.

The responsibility for the wording and views expressed in this descriptive summary lies with MRC, and not with the authors of this report.

A GLOBAL OPTIMIZATION ALGORITHM USING STOCHASTIC DIFFERENTIAL EQUATIONS

Filippo Aluffi-Pentini[1], Valerio Parisi[2], and Francesco Zirilli[3]

1. Introduction.

In [1] 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 stochastic differential equation is a stochastic perturbation of a "steepest descent" ordinary differential equation and is inspired by quantum physics. In [1] the problem of the numerical integration of the stochastic equations 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 sect. 2 we describe the method as implemented in SIGMA; in sect. 3 we give a general description of the method and some details on the implementation; in sect. 4 some numerical experience on test problems is presented and in sect. 5 conclusions are given.

Unfortunately, given the state of the art of mathematical software in global optimization, it has not been possible to make conclusive comparisons with other packages.

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

---

[1]Dipartimento di Matematica, Università di Bari, 70125 Bari (Italy).
[2]Istituto di Fisica, 2ª Università di Roma "Tor Vergata", Via Orazio Raimondo, 00173 (La Romanina) Roma (Italy).
[3]Istituto di Matematica, Università di Salerno, 84100 Salerno (Italy).

Sponsored by the United States Army under Contract No. DAAG29-80-C-0041 and the U. S. Government through its European Research Office of the U. S. Army under Contract n. DAJA-37-81-C-0740 with the University of Camerino, Italy.

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 $\underline{x}^* \in \mathbb{R}^N$ (or possible one of the points) such that

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

and we propose a method introduced in [1] inspired by quantum physics to compute numerically the global minimizers of f by following the paths of a stochastic differential equation.

The interest of the global optimization problem both in mathematics and in many applications is well known and will not be discussed here.

We want just to remark here that the root-finding problem for the system $\underline{g}(\underline{x}) = \underline{0}$, where $\underline{g}:\mathbb{R}^N \to \mathbb{R}^N$ can be formulated as a global optimization problem considering the function $F(\underline{x}) = \|\underline{g}(\underline{x})\|_2^2$, where $\|\cdot\|_2$ is the euclidean norm in $\mathbb{R}^N$.*

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 solid mathematical foundation and good numerical performance.

---

* The present authors have considered this idea both from the mathematical point of view (for a review see [2]) and from the point of view of producing good software (see [3], [4]). The method implemented in [3], [4] is inspired by classical mechanics, uses ordinary differential equations, and can be regarded as a method for global optimization.

Much more satisfactory is the situation for the problem of finding the local minimizers of f, where a large body of theoretical and numerical results exists; see for instance [5], [6] 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 [2].

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, depends on the behaviour 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 behaviour of f along the trajectory, there is no hope of building a completely satisfactory method for global optimization based on 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

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

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

Equation (2.2) is known as the Smoluchowski-Kramers equation [7]; this equation is a singular limit of the Langevin's equation when the inertial terms are 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 (2.2) represents diffusion across potential barriers under the stochastic forces $\varepsilon d\underline{w}$, where $\varepsilon = \sqrt{\frac{2kT}{m}}$, T is the absolute temperature, k the Boltzmann constant, m a suitable mass coefficient, and f is the potential energy.

We assume that

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

in such a way that:

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

and that the minimizers of f are isolated and non degenerate.

It is well known that if $\underline{\xi}^\varepsilon(t)$ is the solution process of (2.2) starting from an initial point $\underline{x}_0$, the probability density function $p^\varepsilon(t,\underline{x})$ of $\underline{\xi}^\varepsilon(t)$ approaches as $t \to \infty$ the limit density $p_\infty^\varepsilon(\underline{x})$ where

$$p_\infty^\varepsilon(\underline{x}) = A_\varepsilon\, e^{-\frac{2}{\varepsilon^2} f(\underline{x})} \tag{2.5}$$

where A_ε is a normalization constant. The way in which $p^\varepsilon(t,\underline{x})$ for a class of one-dimensional systems approaches $p_\infty^\varepsilon(\underline{x})$ has been studied in detail by considering the spectrum of the corresponding Fokker-Planck operators in [8].

We note that p_∞^ε is independent of $\underline{x}_0$ and that as $\varepsilon \to 0$ p_∞^ε becomes more concentrated at the global minimizers of f. That is,

$$\lim_{t\to\infty} \underline{\xi}^\varepsilon(t) = \underline{\xi}_\infty^\varepsilon \quad \text{in law} \tag{2.6}$$

where $\underline{\xi}_\infty^\varepsilon$ has a probability density given by (2.5) and

$$\lim_{\varepsilon\to 0} \underline{\xi}_\infty^\varepsilon = \underline{\xi}_\infty^0 \quad \text{in law} \tag{2.7}$$

where $\underline{\xi}_\infty^0$ 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, x_2, with $\frac{d^2f}{dx^2}(x_i) = c_i > 0$, $i = 1,2$, we have (in distribution sense)

$$\lim_{\varepsilon\to 0} p_\infty^\varepsilon(x) = \gamma \quad \delta(x-x_1) + (1-\gamma)\,\delta(x-x_2) \tag{2.8}$$

where $\gamma = (1+\sqrt{c_1/c_2}\,)^{-1}$. In order to obtain the global minimizers of f as asymptotic values as $t \to \infty$ of a sample trajectory of a suitable system of stochastic differential equations it seems natural to try to perform the limit $t \to \infty$ (i.e. (2.6)) and the limit $\varepsilon \to 0$ (i.e. (2.7)) together.

That is, we want to consider:

$$d\underline{\xi} = -\nabla f(\underline{\xi})dt + \varepsilon(t)d\underline{w} \tag{2.9}$$

with initial condition

$$\underline{\xi}(0) = \underline{x}_0 \tag{2.10}$$

where

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

A-1

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 "frozen".

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). The way in which $\varepsilon(t)$ must go to zero, in order to have that when $t \to \infty$, the solution $\xi(t)$ of (2.9) becomes concentrated at the global minimizers of f, depends on f. In particular, it depends on the highest barrier in f to be overcome to reach the global minimizers.

This dependence has been studied using the adiabatic perturbation theory in [1]. Similar ideas in the context of combinatorial optimization have been introduced by Kirkpatrick, Gelatt, Vecchi in [9].

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), (2.10), disregarding mathematical problems such as the difference between the convergence in law of $\underline{\xi}(t)$ to a random variable concentrated at the global minimizers of f, and the convergence with probability one of the paths of $\underline{\xi}(t)$ 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), (2.10) using the Euler method, that is $\underline{\xi}(t_k)$ is approximated by the solution $\underline{\xi}_k$ of the following finite difference equations:

$$\underline{\xi}_{k+1} - \underline{\xi}_k = -h_k \nabla f(\underline{\xi}_k) + \varepsilon(t_k)(\underline{w}_{k+1} - \underline{w}_k) \qquad k = 0,1,2, \ldots \tag{2.12}$$

$$\underline{\xi}_0 = \underline{x}_0 \tag{2.13}$$

where $t_0 = 0$, $t_k = \sum_{i=0}^{k-1} h_i$, $h_k > 0$, and $\underline{w}_k = \underline{w}(t_k)$, $k = 0,1,2, \ldots$.

The computationally cheap Euler step seems a good choice here since in order to obtain the global minimizers of f as asymptotic values of the paths $\varepsilon(t)$ should go to zero very slowly when $t \to \infty$, and therefore a large number of time integration steps must be computed.

On the right hand side of (2.12) we add the random term $\varepsilon(t_k)(\underline{w}_{k+1} - \underline{w}_k)$ to the deterministic term $-h_k \nabla f(\underline{\xi}_k)$, which is computationally more expensive (e.g. N+1 function evaluations if a forward-difference gradient is used), so that the effort spent in evaluating $\nabla f(\underline{\xi}_k)$ is frequently lost.

In order to avoid this inconvenience we substitute the gradient $\nabla f(\underline{\xi})$ with a "random gradient" as follows. Let $\underline{r}$ be an N-dimensional random vector of length 1 uniformly distributed on the N-dimensional unit sphere. Then for any given (non-random) vector $\underline{v} \in \mathbb{R}^N$ its projection along $\underline{r}$ is such that:

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

where $E(\cdot)$ is the expected value, and $\langle \cdot, \cdot \rangle$ is the euclidean inner product in $\mathbb{R}^N$.

So that in order to save numerical work (i.e. functions evaluations) in (2.12) we substitute $\nabla f(\underline{\xi}_k)$ with the "random gradient"

$$\underline{\gamma}(\underline{\xi}_k) = N < \underline{r}, \nabla f(\underline{\xi}_k) > \underline{r}. \tag{2.15}$$

We note that since $\frac{1}{N}\,\underline{\gamma}(\underline{\xi}_k)$ is the directional derivative in the direction $\underline{r}$, it is computationally much cheaper (e.g. when forward differences are used, only 2 function evaluations are needed to approximate $\underline{\gamma}(\underline{\xi})$). Therefore, the paths are computed approximating $\underline{\xi}(t_k)$ with the solution $\underline{\xi}_k$ of the following differences equations:

$$\underline{\xi}_{k+1} - \underline{\xi}_k = -h_k\,\tilde{\underline{\gamma}}(\underline{\xi}_k) + \varepsilon(t_k)(\underline{w}_{k+1} - \underline{w}_k) \quad k = 0,1,2, \ldots \tag{2.16}$$

$$\underline{\xi}_0 = \underline{x}_0 \tag{2.17}$$

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

The complete algorithm is described in the next section.

3. The complete algorithm.

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

3.1. General description of the algorithm.

The basic time-integration step (eq. (2.16) and sect. 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 $\varepsilon(0)$ (note that even if the starting values $\varepsilon(0)$ are equal the trajectories evolve differently due to the stochastic nature of the integration steps).

The trajectories evolve (simultaneously but independently) during an "observation period" having a given duration (sect. 3.2.5), and within which the noise coefficient of each trajectory is kept at a constant value ε_p, while the values of the steplength h_k and of the spatial discretization increment Δx_k for computing the random gradient (eq. (2.15) and sect. 3.2.2) are automatically adjusted for each trajectory by the algorithm (sects. 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), all other trajectories are naturally continued in the next observation period, and one of the trajectories is selected for "branching" (sect. 3.2.6), that is for generating also a second continuation trajectory differing from the first one only in the starting values for ε_p and Δx_k (sect. 3.2.7), and which is considered as having the same "past history" of the first one.

Let λ_1 be the largest eigenvalue of the (symmetric and non-negative definite) matrix C.

We adopt the updating matrix

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

where I is the N×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 one of the possible 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 $\lambda = \frac{\beta\lambda_1}{2}$, and it therefore acts in the right direction to counter the ill-conditioning of $\tilde{f}$.

The magnitude of the counter-effect depends on β: the adopted value has been experimentally adjusted.

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

$$A'\underline{x} + \underline{b}' = A\underline{x} + \underline{b}.$$

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 x and y are sufficiently close to each other (within given tolerances) to be considered "numerically equal".

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

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 $\tilde{\underline{\gamma}}$ ($\tilde{\underline{\gamma}}^F$ or $\tilde{\underline{\gamma}}^C$) 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.

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 of 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 based on possible analytical or experimental evidence) an admissible region for the x-values (hopefully containing the looked-for global minimizer) within which the function values may be safely computed. We use an N-dimensional interval

$$R_i^{MIN} \leq x_i \leq R_i^{MAX}, \quad i = 1,2, \ldots , N,$$

where the interval boundaries must be given before trial start.

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 make ill-conditioned problems more tractable, rescaling is performed by the algorithm as follows.

the preceding trial, according to the outcome (stopping condition) of the preceding trial and to the number t of trials performed from algorithm start, as compared to the given maximum number of trials N_{TRIAL}

successful stop: $\alpha = 10^3$

unsuccessful uniform stop:

$\alpha = 10$ if $t < [[(2/5)\ N_{TRIAL}]]$

$\alpha = 10^{-4}$ otherwise,

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

unsuccessful non-uniform stop: $\alpha = 10^{-4}$

The initial point $\underline{x}_0$ is selected as follows:

if $t < [[(2/5)\ N_{TRIAL}]]$ take the value of $\underline{x}_0$ at algorithm start

otherwise take $\underline{x}_0 = \underline{x}_{OPT}$

where $\underline{x}_{OPT}$ is the current best minimizer found so far from algorithm start.

All other initial values are those of the first trial, except the initial values of h and $\Delta\alpha$ which are the values reached at the end of the preceding trial.

3.2.10 Stopping criteria for the algorithm.

The complete algorithm is stopped, at the end of a trial, if a given number N_{SUC} has been reached of uniform trial stops all at the current f_{OPT} level, or in any case if a maximum given number N_{TRIAL} of trials has been reached.

Success is claimed by the algorithm if at least one uniform stop occurred at the current f_{OPT} level.

and the best minimum function falue f_{OPT} found so far from algorithm start: if f_{TFMIN} and f_{OPT} satisfy at least one of the above criteria, with the same tolerances, the trial is considered successful at the level f_{OPT}; otherwise the trial is again considered unsuccessful.

Checking of the stopping criteria is activated only if a minimum given number N_{PMIN} of observation periods has been reached.

3.2.9 Characteristics of the successive trials.

The operating conditions which are changed when another trial is started are:

- seed of the random number generator
- maximum duration of the trial
- policy for choosing ε_p for the second continuation of a branched trajectory
- value of ε_p at trial start
- initial point $\underline{x}_0$.

The maximum duration of a trial, i.e. the maximum number N_{PMAX} of observation periods, is obtained as follows:

if the preceding trial had a uniform stop (sect. 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 sect. 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 updating factor F_ε for ε_p is as follows:

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

$F_\varepsilon = 10^{x-\frac{1}{2}}$ where x is a random sample from a standard normal distribution

for all other trials

$F_\varepsilon = 2^{y-\frac{1}{2}}$ where y is a random sample from a standard Cauchy distribution, i.e. with density

$f(y) = 1/(\pi(1+y^2))$

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, and after having discarded the worst trajectory, if all the final function values of the remaining trajectories (possibly at different points x) are "numerically equal", i.e. if the maximum, f_{TFMAX}, and the minimum, f_{TFMIN}, among the trial final values satisfy at least one of the criteria in sect. 3.2.13, the relative difference criterion with a given stopping tolerance τ_{REL} and/or the absolute difference criterion with given stopping tolerance τ_{ABS} ("uniform stop at the level f_{TFMIN}").

The trial is also anyway stopped, at the end of the observation period, if a maximum given number N_{PMAX} of observation periods has been reached.

In the latter case the trial is considered unsuccessful, while in the former case a comparison is made between the final value f_{TFMIN}

From the point of view 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, while the trajectory selected for branching is usually not the best one, to avoid to be stuck in a non-global minimum.

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

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

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.

In phase 6a: $\gamma = 0.1$

We finally remark that h_k and Δx_k are bounded by suitable constants to avoid computational failures.

3.2.5 Duration of the observation period.

The duration of 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 algorithm start among the following three formulas:

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

2) $N_{hp} = [k_p]$ ("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 point of view 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).

We test f_k and $\hat{f}_k = f_k + \hat{\Delta}f_k$ for numerical equality according to the relative difference criterion (sect. 3.2.13) with tolerances $\tau_{R1} = 10^{-11}$ and $\tau_{R2} = 10^{-5}$, and take

$\beta = 2$ if f_k and $\hat{f}_k$ are "equal" within τ_{R1}

$\beta = \frac{1}{2}$ if f_k and $\hat{f}_k$ are not "equal" within τ_{R2}

$\beta = 1$ otherwise.

The interval $(10^{-11}, 10^{-5})$ has been adopted since it contains both the square root and the cubic root of the machine precision of most computers in double precision (the square root is appropriate for forward differences, while the cubic root is appropriate for central differences).

Updating factors γ for h_k .

In phase 4a:

$\gamma = 1/1.05$ for the first attempt to the first half-step

$\gamma = \frac{1}{2}$ for the second attempt

$\gamma = 1/10$ for all other attempts

In phase 5 the value of γ 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 possible rejected while attempting the first step).

If $r > 0$

$\gamma = 1$ (if $a \leq 2r$)

$\gamma = 1.1$ (if $2r < a \leq 3r$)

$\gamma = 2$ (if $3r < a$)

If $r = 0$

$\gamma = 2$ (if $a = 1$)

$\gamma = 10$ (if $a > 1$)

6a. If the half-step is rejected: reject also the first half-step, update (decrease) h_k, and go back to 1.

6b. Otherwise: accept the whole step and try the next one.

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 sect. 3.2.3, is as follows:

Updating factors β for Δx_k

In phase 1b: $\beta = 10^6$

In phase 2a: $\beta = 10$

In phase 4b: $\beta = 10^{-4}$

In phase 5 the value of β 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 $\tilde{\eta}_k^F$ or $\tilde{\eta}_k^C$ as appropriate), and the function value $f_k = f(\underline{\xi}_k)$.

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 $\underline{r}_k$.

1a. Compute the random increment $\underline{s}_k$ (sect. 3.2.2).

1b. If $\underline{s}_k$ (and therefore Δx_k) is too small (i.e. if the square of the euclidean norm of the difference between the computed values of $\underline{\xi}_k + \underline{s}_k$ and $\underline{\xi}_k$ is zero, due to the finite arithmetics of the machine): update (increase) Δx_k and go back to 1a.

2. Compute $\tilde{\eta}_k^F$ (eq. (3.2.2.2)).

2a. If the computed value of $(\tilde{\eta}_k^F)^2$ is zero (due to the finite arithmetics): update (increase) Δx_k and go back to 1a.

3. Compute the first half-step with $\tilde{\underline{\gamma}}_k^F$.

Compute $\Delta' f_k$ (eq. (3.2.3.1)).

3a. If $\Delta' f_k \leqslant |\tilde{\eta}_k^F| \, \Delta x_k$

accept the first half-step and jump to 5.

4. Compute the first half-step with $\tilde{\underline{\gamma}}_k^C$ to check the appropriateness of Δx_k.

Compute $\Delta' f_k$ (eq. (3.2.3.1)).

4a. If $\Delta' f_k > |\tilde{\eta}_k^F - \tilde{\eta}_k^C| \, \Delta x_k$

reject the half-step, update (decrease) h_k, and go back to 1.

4b. Otherwise: accept the half-step, and update (decrease) Δx_k.

5. Update (increase) h_k.

Update (decrease) Δx_k.

6. Compute the second half-step.

Compute $\Delta'' f_k$ (eq. (3.2.3.2)).

and the forward- and central-differences random gradients

$$(3.2.2.3) \qquad \underline{\tilde{\gamma}}_k^F = N\,\tilde{\eta}_k^F\,\underline{r}_k \qquad \underline{\tilde{\gamma}}_k^C = N\,\tilde{\eta}_k^C\,\underline{r}_k$$

We use $\underline{\tilde{\gamma}}_k^F$ or $\underline{\tilde{\gamma}}_k^C$ for $\underline{\tilde{\gamma}}(\underline{\xi}_k)$ 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-differences random gradient ($\underline{\tilde{\gamma}}_k^F$ or $\underline{\tilde{\gamma}}_k^C$ eq. (3.2.2.3)) as described below.

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

$$(3.2.3.1) \qquad \Delta' f_k = f(\underline{\xi}_k') - f(\underline{\xi}_k)$$

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

The second half-step is rejected if the corresponding function increment

$$(3.2.3.2) \qquad \Delta'' f_k = f(\underline{\xi}_{k+1}) - f(\underline{\xi}_k')$$

is positive and too large (greater than $100\,\varepsilon_p^2$ in the present implementation).

The sequence of attempts affects the updating of h_k and Δx_k as described below; the amount of the updating is described in sect. 3.2.4.

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

$$\underline{\xi}'_k = \underline{\xi}_k - h_k \, \underline{\tilde{Y}}(\underline{\xi}_k) \qquad \text{(first half-step)} \tag{3.2.1.2}$$

and

$$\underline{\xi}_{k+1} = \underline{\xi}'_k + \varepsilon_p \sqrt{h_k} \, \underline{u}_k \quad . \qquad \text{(second half-step)} \tag{3.2.1.3}$$

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 $\underline{\tilde{Y}}(\underline{\xi}_k)$.

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

3.2.2 The finite-differences random gradient.

Given the current value Δx_k of the spatial discretization increment for the trajectory under consideration, we consider the random increment vector

$$\underline{s}_k = \Delta x_k \cdot \underline{r}_k$$

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

$$\begin{cases} \Delta^F f_k = f(\underline{\xi}_k + \underline{s}_k) - f(\underline{\xi}_k) \\ \Delta^C f_k = \frac{1}{2}[f(\underline{\xi}_k + \underline{s}_k) - f(\underline{\xi}_k - \underline{s}_k)] \end{cases} \tag{3.2.2.1}$$

the forward- and central-differences directional derivatives

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

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

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

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

3.2. Implementation details.

3.2.1 The time-integration step.

The basic time-integration step (eq. (2.16)) is used, for the trajectory under consideration, in the form

$$\underline{\xi}_{k+1} = \underline{\xi}_k - h_k \, \underline{\tilde{\gamma}}(\underline{\xi}_k) + \varepsilon_p \sqrt{h_k} \, \underline{u}_k \quad (k = 0,1,2, \ldots) \tag{3.2.1.1}$$

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 (sect. 3.1)); $\underline{u}_k$ is a random vector sample from an N-dimensional standard Gaussian distribution, and

$$\sqrt{h_k} \, \underline{u}_k = \underline{w}_{k+1} - \underline{w}_k$$

due to the properties of the Wiener process.

The computation of the finite-differences random gradient $\underline{\tilde{\gamma}}(\underline{\xi}_k)$ is described in the next section.

a) Relative difference criterion

$$|x-y| \leq \tau_{REL} \quad (|x| + |y|)/2$$

b) Absolute difference criterion

$$|x-y| \leq \tau_{ABS}$$

where τ_{REL} and τ_{ABS} are given non-negative tolerances.

4. Numerical Testing.

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

4.1. Test problems.

The set of test problems is fully described in [10] 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 $a = 10000$ (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 minima (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 used the above functions, and the standard initial points as they are coded in the subroutines GLOMTF and GLOMIP, which are available in [10].

4.2. Test computers.

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

— UNIVAC 1100/82 with EXEC8 operating system and FORTRAN (ASCII) computer (level 10R1) ("large" dynamic range)

— D.E.C. VAX 11/750 with VMS operating system (vers. 3.0) and FORTRAN compiler (vers. 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. All results were obtained under the following operating conditions.

The easy-to-use driver subroutine SIGMA1 (described in the accompanying algorithm) was used, with $N_{SUC} = 1,2,3,4,5$. 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.

All numerical results are reported on Tables 1, 2, and 3. Table 1 reports some performance data (i.e. output indicator IOUT and number of functions evaluations) as obtained from SIGMA output for each of the 37 test problems and for the testing both on the "large" and "small" dynamic range machines. 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 solutions — 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 occurrs at any point of the iteration.

Table 2 and Table 3 report for each problem and summarized for all problems data concerning the effectiveness, dependability and robustness — in the form of total numbers of correctly claimed successes, correctly claimed failures, incorrect success or failure claims and total number of overflows — for the two machines.

TABLE 1

UNIVAC

N_{SUC} =		1		2		3		4		5	
NPROB	N	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie
1	1	3,588	0	11,467	0	23,067	0	32,520	0	58,751	0
2	1	3,254	0	9,509	0	20,893	0	32,910	0	72,015	0
3	1	8,638	0	17,741	0	23,814	0	57,187	0	67,621	0
4	2	6,594	0	15,898	0	30,589	0	69,489	0	101,633	0
5	2	12,680	0	23,221	0	38,362	0	95,423	0	104,391	0
6	2	2,697	0	8,343	0	19,660	0	57,728	0	78,090	0
7	2	32,185	0	35,256	0	49,153	0	59,983	0	139,675	0
8	2	5,600	0	347,039	0	348,301	0	359,642	0	392,466	0
9	2	6,180	0	83,625	0	470,130	0	699,767	0	701,051	0
10	2	3,596	0	6,731	0	12,958	0	61,753	0	66,855	0
11	2	3,191	0	8,384	0	23,196	0	40,808	0	56,958	0
12	2	4,799	0	7,296	0	18,902	0	29,315	0	47,216	0
13	2	7,105	0	10,287	0	20,605	0	27,838	0	43,505	0
14	2	6,671	0	10,654	0	15,102	0	31,322	0	47,051	0
15	2	7,747	0	11,631	0	16,227	0	23,587	0	38,362	0
16	2	16,021	0	26,560	0	58,401	0	67,865	0	115,350	0
17	2	2,700	0	6,670	0	14,388	0	28,275	0	80,826	0
18	4	4,674	0	16,556	0	101,828	0	209,177	0	282,950	0
19	4	4,759	0	54,559	0	131,350	0	224,028	0	306,327	0
20	4	9,955	0	90,092	0	262,616	0	278,385	0	327,392	0
21	3	3,416	0	12,520	0	27,472	0	66,044	0	86,482	0
22	6	4,729	0	10,488	0	20,318	0	36,981	0	52,364	0
23	2	11,888	0	16,660	0	32,579	0	84,168	0	92,194	0

[illegible]

UNIVAC (continued)

N_{SUB}		1		2		3		4		5	
NPROB	N	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie
24	3	8,099	0	36,057	0	47,619	0	69,901	0	92,104	0
25	4	11,954	0	54,212	0	71,655	0	97,724	0	191,722	0
26	5	43,083	0	284,104	0	347,056	0	450,102	0	464,611	0
27	8	2,324	0	21,124	0	75,728	0	635,990	0	654,436	0
28	10	50,975	0	426,171	0	454,808	0	474,323	0	479,817	0
29	2	25,462	0	35,675	0	98,944	0	111,447	0	167,728	0
30	3	15,734	0	113,789	0	177,970	0	257,904	0	286,273	0
31	4	11,516	0	143,757	0	208,217	0	264,834	0	296,663	0
32	5	50,911	0	176,840	0	275,852	0	357,089	0	679,442	0
33	6	53,178	0	102,652	0	272,642	0	303,267	0	454,543	0
34	7	14,594	0	298,256	0	357,878	0	409,949	0	520,641	0
35	5	33,635	0	50,348	0	70,105	0	127,091	0	183,864	0
36	2	3,102	0	10,176	0	23,283	0	72,931	0	79,481	0
37	5	6,938	0	12,469	0	25,175	0	64,639	0	92,407	0

Table 1 (continued)

N_{SUC} =		VAX 1		2		3		4		5	
NPROB	N	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie
1	1	7,522	0	12,657	0	21,643	0	31,787	0	46,954	0
2	1	3,131	0	6,542	0	14,171	0	27,023	0	70,953	0
3	1	11,526	0	15,342	0	20,457	0	57,342	0	67,423	0
4	2	9,265	0	17,713	0	28,667	0	80,161	0	144,719	0
5	2	12,094	0	19,716	0	36,426	0	59,214	0	100,336	0
6	2	4,650	0	11,040	0	25,772	0	57,087	0	62,099	0
7	2	10,543	0	44,408	0	82,833	0	130,859	0	156,208	0
8	2	27,044	0	76,348	0	189,195	0	521,474	0	604,401	0
9	2	24,348	0	35,885	0	71,593	0	165,393	0	225,842	0
10	2	4,114	0	9,959	0	19,363	0	42,409	0	91,572	0
11	2	3,254	0	7,901	0	12,795	0	28,450	0	42,615	0
12	2	6,711	0	9,949	0	18,191	0	28,788	0	44,896	0
13	2	6,771	0	11,031	0	15,508	0	22,629	0	39,765	0
14	2	6,208	0	9,443	0	17,719	0	23,579	0	39,721	0
15	2	6,313	0	13,581	0	17,631	0	30,648	0	42,360	0
16	2	5,439	0	10,491	0	24,055	0	80,137	0	101,441	0
17	2	2,790	0	11,006	0	18,444	0	51,305	0	61,100	0
18	4	2,446	0	36,252	0	129,264	0	269,925	0	290,805	0
19	4	4,778	0	19,951	0	44,198	0	109,747	0	273,679	0
20	4	4,741	0	11,312	0	24,947	0	78,820	0	125,407	0
21	3	4,334	0	27,816	0	44,893	0	82,640	0	150,742	0
22	6	3,975	0	8,613	0	16,514	0	54,082	0	68,270	0
23	2	5,534	0	29,691	0	52,042	0	65,588	0	75,379	0

Table 1 (continued)

VAX (continued)

N_{SUC} =		1		2		3		4		5	
NPROB	N	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie	Nf	Ie
24	3	10,050	0	18,170	0	80,567	0	109,726	0	190,920	0
25	4	10,657	0	37,655	0	56,285	0	129,598	0	227,682	0
26	5	59,689	0	369,912	0	460,779	0	476,325	0	585,249	0
27	8	168,933	0	259,950	0	302,092	0	310,718	0	330,192	0
28	10	43,466	0	393,550	0	411,854	0	454,095	0	474,422	0
29	2	15,223	0	55,718	0	95,254	0	131,418	0	171,995	0
30	3	12,641	0	54,822	0	118,927	0	201,965	0	333,175	0
31	4	26,235	0	123,716	0	148,183	0	242,535	0	397,066	0
32	5	35,365	0	86,758	0	186,860	0	285,371	0	316,702	0
33	6	49,087	0	71,418	0	159,987	0	184,087	0	296,770	0
34	7	50,237	0	132,768	0	173,955	0	204,081	0	348,809	0
35	5	14,815	0	53,394	0	79,235	0	129,480	0	206,980	0
36	2	3,744	0	9,574	0	23,355	0	55,947	0	82,518	0
37	5	3,847	0	12,239	0	29,411	0	70,599	0	107,264	0

NPROB = problem number given in sect. 4.1.

Ie = 0 success (IOUT > 0) (claimed by SIGMA)

= 1 failure (IOUT ≤ 0) (claimed by SIGMA)

NSUC = see sect. 3.2.10.

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

TABLE 2

UNIVAC

N_{SUC} =		1	2	3	4	5
NPROB	N					
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	2	1	1	1	1	1
5	2	1	1	1	1	1
6	2	1	1	1	1	1
7	2	1	1	1	1	1
8	2	3	1	1	1	1
9	2	3	3	1	1	1
10	2	1	1	1	1	1
11	2	1	1	1	1	1
12	2	1	1	1	1	1
13	2	1	1	1	1	1
14	2	1	1	1	1	1
15	2	1	1	1	1	1
16	2	1	1	1	1	1
17	2	1	1	1	1	1
18	4	3	3	1	1	1
19	4	3	1	1	1	1
20	4	3	1	1	1	1
21	3	1	1	1	1	1
22	6	1	1	1	1	1
23	2	1	1	1	1	1
24	3	1	1	1	1	1
25	4	1	1	1	1	1
26	5	1	1	1	1	1
27	8	3	3	3	1	1
28	10	1	1	1	1	1

Table 2 (continued)

UNIVAC (continued)

		1	2	3	4	5
[illegible]	N					
[illegible]	2	1	1	1	1	1
[illegible]	3	3	1	1	1	1
[illegible]	4	3	1	1	1	1
[illegible]	5	1	1	1	1	1
[illegible]	6	1	1	1	1	1
[illegible]	7	3	1	1	1	1
[illegible]	8	1	1	1	1	1
[illegible]	2	3	3	3	3	3
[illegible]	3	3	3	3	3	3

Table 2 (continued)

VAX

N_{SUC} =		1	2	3	4	5
NPROB	N					
1	1	1	1	1	1	1
2	1	1	1	1	1	1
3	1	1	1	1	1	1
4	2	1	1	1	1	1
5	2	1	1	1	1	1
6	2	1	1	1	1	1
7	2	1	1	1	1	1
8	2	3	3	3	3	1
9	2	1	1	1	1	1
10	2	1	1	1	1	1
11	2	1	1	1	1	1
12	2	1	1	1	1	1
13	2	1	1	1	1	1
14	2	1	1	1	1	1
15	2	1	1	1	1	1
16	2	1	1	1	1	1
17	2	1	1	1	1	1
18	4	3	1	1	1	1
19	4	1	1	1	1	1
20	4	1	1	1	1	1
21	3	3	1	1	1	1
22	6	1	1	1	1	1
23	2	1	1	1	1	1
24	3	1	1	1	1	1
25	4	1	1	1	1	1
26	5	1	1	1	1	1
27	8	1	1	1	1	1
28	10	1	1	1	1	1

Table 2 (continued)

VAX (continued)

N_{SUC} =		1	2	3	4	5
NPROB	N					
29	2	1	1	1	1	1
30	3	1	1	1	1	1
31	4	1	1	1	1	1
32	5	1	1	1	1	1
33	6	1	1	1	1	1
34	7	1	1	1	1	1
35	5	1	1	1	1	1
36	2	3	3	3	3	3
37	5	3	3	3	3	3

1 = success correctly claimed

2 = failure correctly claimed

3 = incorrect claim

4 = overflow

TABLE 3

		UNIVAC					VAX				
N_{SUC} =		1	2	3	4	5	1	2	3	4	5
Totals	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

1 = success correctly claimed

2 = failure correctly claimed

3 = incorrect claim

4 = overflow

5. Conclusions.

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

As it is shown in [10] 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 $|x_i| < 10$ $i = 1,2, \ldots, 10$.

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

Unfortunately, given the state of the art on mathematical software for global optimization, it has not been possible to make conclusive comparisons with other packages.

Finally, we note that a smaller value of N_{SUC} gives a much cheaper method (less function evaluations) at the expense of a loss in effectiveness (greater number of failures).

ACKNOWLEDGEMENT: One of us (F.Z.) gratefully acknowledges the hospitality and the support of the Mathematics Research Center of the University of Wisconsin and of the Department of Mathematical Sciences of Rice University where part of this work was done, and Prof. A. Rinooy Kan for bringing to our attention ref. [9].

REFERENCES

[1] Aluffi-Pentini F., Parisi V., Zirilli F.: "Global optimization and stochastic differential equations," submitted to Journal Optimization Theory and Applications.

[2] Zirilli F.: "The use of ordinary differential equations in the solution of nonlinear systems of equations," in "Nonlinear Optimization 1981," M.J.D. Powell, Editor, Academic Press, London, 1982, 39-47.

[3] Aluffi-Pentini F., Parisi V., Zirilli F.: "A differential equations algorithm for nonlinear equations," ACM Transactions on Mathematical Software, 10, (1984), 299-316.

[4] Aluffi-Pentini F., Parisi V., Zirilli F.: "Algorithm 617 DAFNE. A differential equations algorithm for nonlinear equations," ACM Transactions on Mathematical Software, 10, (1984), 317-324.

[5] Powell M.J.D. (Editor): "Nonlinear Optimization 1981," Academic Press, London, 1982.

[6] Dennis J. E., Schnabel R. B.: "Numerical methods for unconstrained optimization and nonlinear equations," Prentice Hall, Inc., London, 1983.

[7] Schuss Z.: "Theory and applications of stochastic differential equations," J. Wiley & Sons, New York, 1980, chapter 8.

[8] Angeletti A., Castagnari C., Zirilli F.: "Asymptotic eigenvalue degeneracy for a class of one dimensional Fokker-Planck operators," to appear in J. of Math. Phys.

[9] Kirkpatric S., Gelatt Jr. C.D., Vecchi M.P.: "Optimization by simulated annealing," Science 220 , (1983), 671-680.

[10] Aluffi-Pentini F., Parisi V., Zirilli F.: 'Test problems for global optimization software," submitted to A.C.M. Transactions on Mathematical Software.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 2791	2. GOVT ACCESSION NO.: AD-A153526
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A GLOBAL OPTIMIZATION ALGORITHM USING STOCHASTIC DIFFERENTIAL EQUATIONS	5. TYPE OF REPORT & PERIOD COVERED: Summary Report - no specific reporting period
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Filippo Aluffi-Pentini, Valerio Parisi and Francesco Zirilli	8. CONTRACT OR GRANT NUMBER(s): DAAG29-80-C-0041; DAJA-37-81-C-0740
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin, 610 Walnut Street, Madison, Wisconsin 53706	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit Number 5 - Optimization and Large Scale Systems
11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, P.O. Box 12211, Research Triangle Park, North Carolina 27709	12. REPORT DATE: February 1985
	13. NUMBER OF PAGES: 41
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

Algorithms, Theory, Verification, Global Optimization, Stochastic Differential Equations

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

SIGMA is a set of FORTRAN subprograms for solving the global optimization problem, which implement a method founded on the numerical solution of a Cauchy problem for stochastic differential equations inspired by quantum physics.

(continued)

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (cont.)

This paper gives a detailed description of the method as implemented in SIGMA, and reports on the numerical tests which have been performed while the SIGMA package is described in the accompanying Algorithm.

The main conclusions are that SIGMA performs very well on several hard test problems; unfortunately given the state of the mathematical software for global optimization it has not been possible to make conclusive comparisons with other packages.

END

FILMED

6-85

DTIC