Neural Networks, Vol. 3, pp. 467–483, 1990
Printed in the USA. All rights reserved.

0893-6080/90 $3.00 + .00

ORIGINAL CONTRIBUTION

Experiments in Nonconvex Optimization: Stochastic Approximation with Function Smoothing and Simulated Annealing

M. A. STYBLINSKI AND T.-S. TANG

Texas A&M University

(*Received* 3 *November* 1988; *revised and accepted* 27 *November* 1989)

Abstract—*The method of simulated annealing, to be referred to—after Szu (1986)—as classical simulated annealing (CSA), received much attention recently, especially for solving combinatorial problems (e.g., in the VLSI area), implementing the Boltzman machine for learning in neural networks, and recently for global function optimization. In Szu (1986a, 1987) the fast simulating annealing (FSA) algorithm was introduced—using the Cauchy rather than Gaussian sampling and fast cooling—and applied to finding the global minimum of a nonconvex multiextremal function. It has proved to be superior to the CSA technique. Recently, the method of stochastic approximation combined with convolution smoothing (to be referred to as SAS) was successfully used by Styblinski and Opalski (1984) for manufacturing yield optimization with deterministic parameters and nondifferentiable p.d.f's (Tang & Styblinski, 1988). In this presentation, some analogies between the CSA/FSA and SAS algorithms are discussed in application to global function minimization. It is demonstrated that the SAS approach is much more accurate and efficient on the test examples tried than the FSA algorithm. Convenient statistical criteria are proposed and used for algorithm performance evaluation, together with test examples of controlled properties.*

Keywords—Nonconvex and global optimization, Simulated and fast simulated annealing, Stochastic approximation, Convolution smoothing.

1. INTRODUCTION

In this paper we consider the following problem of global, unconstrained optimization: minimize the multiextremal (i.e., nonconvex) function $f(x) \in R^1$, $x \in R^n$, i.e.:

$$\min_{x \in R^n} f(x) \tag{1}$$

In recent years the method of simulated annealing (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983), closely related to the work of Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller (1953), gained popularity in successfully solving several difficult combinatorial problems (van Laarhoven & Aarts, 1987), such as the Traveling Salesman Problem (Kirkpatrick et al., 1983), VLSI circuit design (placement, routing, PLA folding, gate-matrix layout, etc.) (Wong, Leong, & Liu 1988), and others. Application of the SA technique to solving the general problem (1) has also been proposed in Szu (1986a, b, 1987), and Szu and Hartley (1987). The fast simulating annealing (FSA) algorithm was introduced and used for finding the global minimum of (1). FSA uses the Cauchy rather than Gaussian sampling and a fast (inversely linear in time rather than inversely logarithmic in time) cooling scheme. Due to these novel features, FSA proved to be much more efficient than the classical simulated annealing (CSA)—as the SA is called by Szu.

Recently, the method of stochastic approximation combined with convolution smoothing (to be referred to as SAS) was successfully used by Styblinski and Opalski (1984) for manufacturing yield optimization with deterministic parameters and nondifferentiable p.d.f.'s (Tang and Styblinski, 1988). In this paper some analogies between the SA and SAS al-

---

This is an expanded version of the paper: Stochastic Approximation Algorithm with Function Smoothing vs. Simulated Annealing, presented at The International Neural Network Society First Annual Meeting, September 6–10, 1988, Boston, MA. This work was supported in part by a grant from the Texas Advanced Technology Program (Project No. 70190 ATG).

Requests for reprints should be sent to M. A. Styblinski, Department of Electrical Engineering, Texas A&M University, College Station, TX 77843.

gorithms are discussed in application to solving the global function minimization problem (1). Specifically, the efficiency and accuracy of solving (1) using the SAS and FSA methods are studied on several multiextremal and multidimensional examples. It has been demonstrated that for all the examples statistically investigated in our experiments, the SAS approach was always much more efficient (in terms of the required number of function calculations) and accurate (in terms of the accuracy of the convergence to the global and local minima) than the FSA algorithm.

This paper is organized as follows: Basic principles and motivation beyond the convolution smoothing for global optimization are given first. Then gradient formulas for the resulting smoothed functional are developed and their accuracy is studied theoretically and experimentally. In section 4, CSA, FSA, and SAS algorithms are discussed, and the details of the stochastic approximation algorithm used are given. Several test examples are studied in section 5, where special statistical algorithm evaluation criteria are introduced, and the measure of global convergence quality (MGCQ) is proposed. Section 6 contains conclusions, as well as discussion of potential applications for the SAS techniques proposed. Important open research problems are also pointed out.

2. CONVOLUTION FUNCTION SMOOTHING

Motivation. A multiextremal function $f(x) \in R^1$, $x \in R^n$ can be represented as a superposition of a uniextremal function (i.e., having just one minimum) and other multiextremal functions that add some "noise" to the uniextremal function. The objective of convolution smoothing can be visualized as "filtering out" the noise and performing minimization on the "smoothed" uniextremal function (or on a family of these functions), in order to reach the global minimum. Since the minimum of the smoothed uniextremal function does not, in general, coincide with the global function minimum, a sequence of minimization runs is required with the amount of smoothing eventually reduced to zero in the neighborhood of the global minimum. The smoothing process is performed by averaging $f(x)$ over some region of the parameter space R^n using a proper weighting (or smoothing) function $\hat{h}(x)$, defined below. Formally, let us introduce a vector of random perturbations $\eta \in R^n$, and add η to x, thus creating the convolution function (Rubinstein, 1981):

$$\tilde{f}(x, \beta) = \int_{R^n} \hat{h}(\eta, \beta) f(x - \eta)\, d\eta = \int_{R^n} \hat{h}(x - \eta, \beta) f(\eta)\, d\eta. \quad (2)$$

Hence:

$$\tilde{f}(x, \beta) = E_\eta[f(x - \eta)], \quad (3)$$

where $\tilde{f}(x, \beta)$ is the *smoothed approximation* to the original multiextremal function $f(x)$, and the kernel function $\hat{h}(\eta,\beta)$ is the p.d.f. used to sample η. Note that $\tilde{f}(x, \beta)$ can be regarded as an averaged version of $f(x)$ weighted by $\hat{h}(\eta,\beta)$.

The parameter β controls the dispersion of $\hat{h}$, i.e., the degree of $f(x)$ *smoothing* (e.g., β can control the standard deviations of $\eta_1 \ldots \eta_n$).

$E_\eta[f(x - \eta)]$ is the expectation with respect to the random variable η. Therefore, an unbiased estimator $\tilde{f}(x, \beta)$ is the average:

$$\tilde{f}(x, \beta) = \frac{1}{N} \sum_{i=1}^{N} f(x - \eta^i), \quad (4)$$

where η is sampled with the p.d.f. $\hat{h}(\eta,\beta)$.

The kernel function $\hat{h}(x, \beta)$ should have the following properties (Rubinstein, 1981):

(a)

$$\hat{h}(\eta,\beta) = \frac{1}{\beta^n} h\left(\frac{\eta}{\beta}\right) \quad (5)$$

is piecewise differentiable with respect to η,
(b) $\lim_{\beta \to 0} \hat{h}(\eta,\beta) = \delta(\eta)$ (Dirac's delta functional),
(c) $\hat{h}(x, \beta)$ is a p.d.f.

Under these conditions $\lim_{\beta \to 0} \tilde{f}(x, \beta) = \int_{R^n} \delta(\eta) f(x - \eta)\, d\eta = f(x - 0) = f(x)$.

Several p.d.f.'s fulfill the conditions (a)–(c), for example, the Gaussian or uniform p.d.f.'s. As an example, consider the following function used by Szu (1986a) to compare the CSA and FSA techniques:

$$f(x) = x^4 - 16x^2 + 5x. \quad (6)$$

This function is continuous and differentiable, and it has two distinct minima, as shown in Figure 1. In Figure 2, the smoothed functional (2)—obtained using (6)—is plotted for different values of $\beta \to 0$, for Gaussian, uniform and Cauchy kernels. Except for the Cauchy kernel, the corresponding smoothed functionals were calculated from analytical formulas.

Observations. Smoothing is able to eliminate the local minima of $\tilde{f}(x, \beta)$, if β is sufficiently large. If $\beta \to 0$ then $\tilde{f}(x, \beta) \to f(x)$; this should actually happen at the end of optimization to provide convergence to the true function minimum.

Discussion. The simulated annealing algorithm is often interpreted in terms of the energy a "particle" has at any given temperature. Lowering the temperature reduces the particle energy. A similar interpretation can also be given to the smoothed approximation approach discussed. Perturbing x can be viewed as adding some random energy to a par-

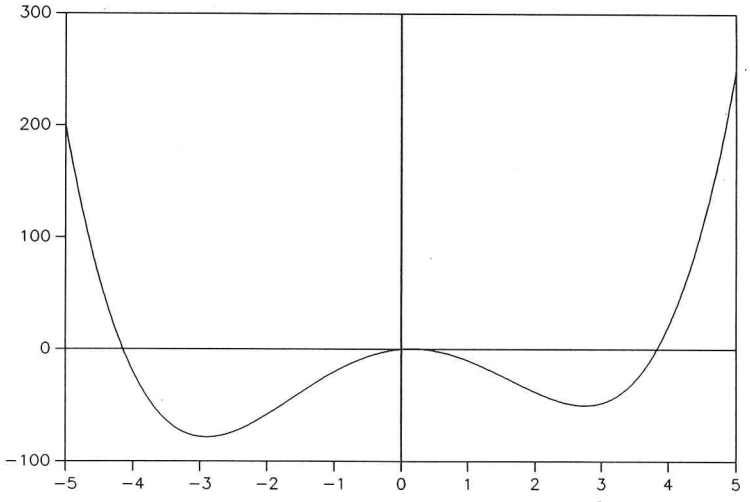

FIGURE 1. The multimodal function (6) used by Szu (1986).

ticle which x represents, thus randomly changing its state. The larger the β, the larger the energy (i.e., the larger the temperature in the SA interpretation), and the broader the range of x changes. Thus: *reducing β for the smoothed approximation approach corresponds to temperature reduction in the simulated annealing case.*

2.1. Reformulation of the Optimization Problem

Our objective now is to solve the following optimization problem: Minimize the smoothed functional $\tilde{f}(x, \beta)$ with $\beta \to 0$ as $x \to \hat{x}$, where $\hat{x}$ is the global minimum of the original function $f(x)$.

Formally, the modified optimization problem can

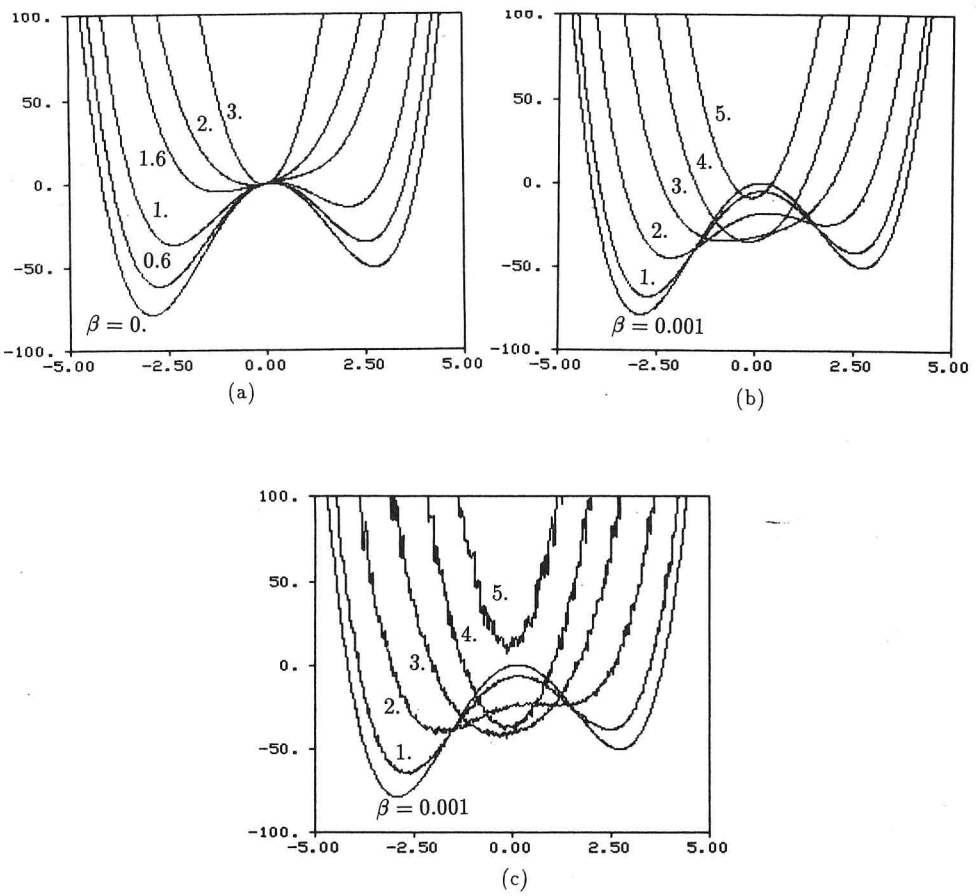

FIGURE 2. Smoothed functional $\tilde{f}(x, \beta)$ for different β's using the following p.d.f.'s for η: (a) Gaussian, (b) uniform, (c) Cauchy (the last obtained by Monte Carlo simulation).

be written as:

$$\min_{x \in R^n} \tilde{f}(x, \beta), \tag{7}$$

with $\beta \to 0$ as $x \to \hat{x}$.

3. GRADIENT CALCULATION

To apply the SAS algorithm to be discussed in section 4.2, the gradient of the smoothed functional $\tilde{f}(x, \beta)$ is needed. Differentiating (2) and using variable substitution, one obtains the following formulas for the gradient of (2) with respect to x, for two distinct cases encountered in practice:

(a) function values $f(\cdot)$ are only available

$$\nabla_x \tilde{f}(x, \beta) = \int_{R^n} \nabla_\eta \hat{h}(\eta, \beta) f(x - \eta)\, d\eta = \frac{1}{\beta} \int_{R^n} \nabla_\eta h(\eta) f(x - \beta\eta)\, d\eta, \tag{8a}$$

(b) gradient of $f(\cdot)$ is available

$$\nabla_x \tilde{f}(x, \beta) = \int_{R^n} h(\eta) \nabla_x f(x - \beta\eta)\, d\eta, \tag{8b}$$

where $\hat{h}(\cdot)$ is defined by (5).

One of the possible choices for $h(\eta)$ is a normalized multinormal p.d.f. (Rubinstein, 1981), which leads to the following expression for $\hat{h}(\eta, \beta)$:

$$\hat{h}(\eta, \beta) = \frac{1}{(2\pi)^{n/2}\beta^n} \exp\left[-\frac{1}{2} \sum_{i=1}^{n} \left(\frac{\eta_i}{\beta} \right)^2 \right], \qquad \beta > 0. \tag{9}$$

Then, after simple derivations, the gradient of $\tilde{f}(x, \beta)$ is expressed as:

(a) function values $f(\cdot)$ are only available

$$\nabla_x \tilde{f}(x, \beta) = \frac{-1}{\beta} \int_{R^n} \eta f(x - \beta\eta) h(\eta)\, d\eta = \frac{-1}{\beta} E_\eta[\eta f(x - \beta\eta)]. \tag{10a}$$

(b) gradient of $f(\cdot)$ is available

$$\nabla_x \tilde{f}(x, \beta) = E_\eta[\nabla_x f(x - \beta\eta)], \tag{10b}$$

where sampling is performed in the R^n space with the p.d.f. $h(\eta)$.

Therefore, the unbiased gradient estimators are, respectively:

$$\hat{\nabla}_x \tilde{f}(x, \beta) = \frac{-1}{\beta} \frac{1}{N} \sum_{i=1}^{N} \eta^i f(x - \beta\eta^i), \tag{11a}$$

for case (a), and

$$\hat{\nabla}_x \tilde{f}(x, \beta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_x f(x - \beta\eta^i), \tag{11b}$$

for case (b).

In (11a,b) N points η^i are sampled with the p.d.f. $h(\eta)$. Substituting $N = 1$ in (11a,b) one obtains the *one-sample* gradient estimators usually used in the stochastic approximation algorithms studied by Styblinski and Ruszczynski (1983); Styblinski and Opalski (1984); and Tang and Styblinski (1988), and implemented in our optimization system (Opalski & Styblinski, 1984).

It is important to point out that to find $\hat{\nabla}_x \tilde{f}(x, \beta)$, the original gradient $\nabla f(x)$ (which might not exist at all) does not have to be known. This is especially attractive if the analytic forms of $f(x)$ or its gradient are not known, and can only be simulated, which is a typical case, e.g., during optimization of electrical circuits.

As it will be shown below, the variability of the *single-sided* gradient estimators (11a,b) is rather large. Therefore, as suggested by Rubinstein (1981), the following *double-sided* smoothed gradient estimators were used in our experiments instead (assuming the Gaussian kernel (9)):

$$\hat{\nabla}_x \tilde{f}(x, \beta) = \frac{1}{2\beta} \frac{1}{N} \sum_{i=1}^{N} \eta^i [f(x + \beta\eta^i) - f(x - \beta\eta^i)], \tag{12a}$$

for case (a), and

$$\hat{\nabla}_x \tilde{f}(x, \beta) = \frac{1}{2N} \sum_{i=1}^{N} [\nabla_x f(x + \beta\eta^i) + \nabla_x f(x - \beta\eta^i)], \tag{12b}$$

for case (b).

Formulas (12a) and (12b) can readily be obtained from (10a), (10b), respectively, by changing variables, adding two identical integrals, and dividing the results by two; then the gradient estimators are obtained as averages of N samples in the same way as (11a) and (11b) were obtained.

3.1 Variability of Gradient Estimators

The variability of the four gradient estimators can be studied theoretically by considering the sample variance of formulas (11a,b) and (12a,b). Assuming that each sample is sampled independently with the same p.d.f., one obtains the variances of the jth components $[\hat{\nabla}_x \tilde{f}(x, \beta)]_j$ of the gradient vectors, for all four cases considered.

For single-sided estimators one obtains:

(a) function values $f(\cdot)$ are only available

$$\mathrm{var}[\hat{\nabla}_x \tilde{f}(x, \beta)]_j = \frac{1}{\beta^2} \frac{1}{N} \mathrm{var}[\eta_j f(x - \beta\eta)], \tag{11a'}$$

(b) gradient of $f(\cdot)$ is available

$$\mathrm{var}[\hat{\nabla}_x \tilde{f}(x, \beta)]_j = \frac{1}{N} \mathrm{var}[\nabla_x f(x - \beta\eta)]_j. \tag{11b'}$$

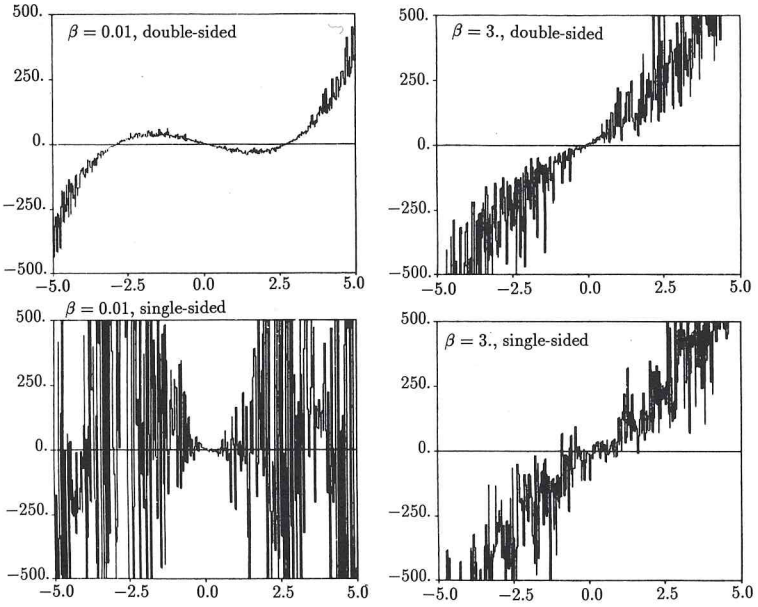

FIGURE 3. Gradient estimators (single- and double-sided) using *function values* only vs. x for function (6) and different values of β.

For double-sided estimators one obtains:

(a) function values $f(\cdot)$ are only available

$$\operatorname{var}[\hat{\nabla}_x \tilde{f}(x, \beta)]_j = \frac{1}{\beta^2}\frac{1}{N}\operatorname{var}\{\eta_j[f(x + \beta\eta) - f(x - \beta\eta)]\}, \quad (12a')$$

(b) gradient of $f(\cdot)$ is available

$$\operatorname{var}[\hat{\nabla}_x \tilde{f}(x, \beta)]_j = \frac{1}{4N}\operatorname{var}[\nabla_x f(x + \beta\eta) + \nabla_x f(x - \beta\eta)]_j. \quad (12b')$$

In order to learn about the practical variability of the gradient estimators introduced, their experimentally obtained values are plotted versus x in Figures 3 and 4 for function (6) and two different values of β. For each x point 100 Monte Carlo normally distributed samples were generated and formulas (11a, b), (12a,b) used. For each case the standard deviation of the Gaussian kernel used was equal to 1.

Discussion. From the variance formulas (11a′,b′), (12a′,b′), and Figures 3 and 4 the following obser-

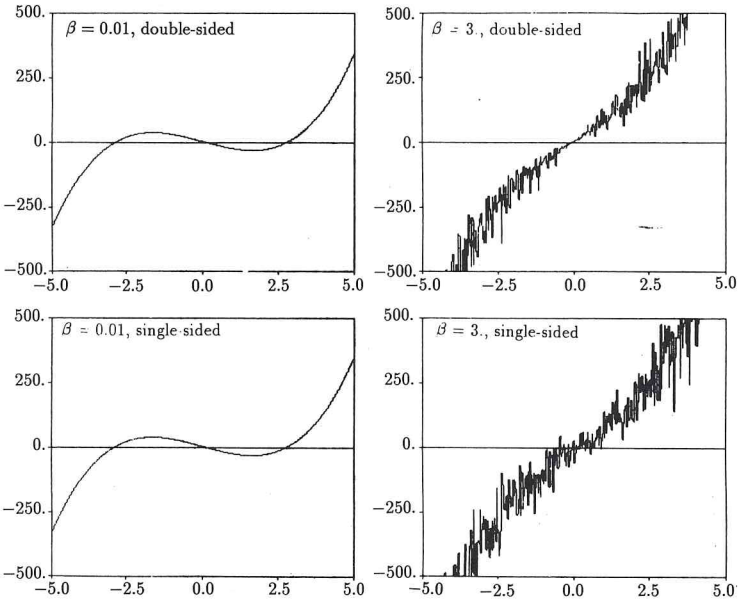

FIGURE 4. Gradient estimators (single- and double-sided) using *gradient values* vs. x for function (6) and different values of β.

vations can be made:

(1) The source of variability is the variance expression appearing in formulas (11a′,b′), (12a′,b′). For the same β *and* N values used, and a specific sampling p.d.f., the double-sided estimators should have smaller variability, due to a smaller, in general, variability of the function *difference* taken symmetrically at both sides of the current x value (this is the same property as the greater accuracy of double-sided versus single-sided finite difference deterministic derivative formulas). On the other hand, the required number of function or gradient evaluations is *doubled*. Based on our experience, this price is well worth paying in cases where the function values are only used, due a much worse performance of the single-sided estimators, as seen from Figure 3, especially for small values of β. In fact, estimators based on N double-sided samples (requiring $2N$ function evaluations) were as a rule more accurate than estimators based on $2N$ single-sided samples (i.e., requiring the same number of function evaluations). When the function gradient is available, double-sided estimators are still better, as expected, but the difference for small values of β is practically insignificant (Figure 4).

(2) Gradient-based estimators have significantly smaller variability than the function-value based, since they use *exact* values of the gradient, but calculated not at the actual point x, but at the points x shifted by + or $-\beta\eta$. Therefore, with β tending to zero, their variability also reduces, as seen from (11b′), (12b′) and Figure 4. For that reason the gradient of $f(x)$ should always be used, if available, since the final convergence will be faster due to the increasing accuracy of gradient estimators, as β tends to zero at the final stages of optimization.

(3) When the function values are only used, formulas (11a′), (12a′) suggest that the variance *increases* when β *decreases* due to the $1/\beta^2$ term. This might not be so, however, if the variance term decreases faster with the change of β than the $1/\beta^2$ term does. Such a situation can occur when the function $f(x)$ is highly nonlinear: larger values of β imply sampling in a broader range, which combined with a high function nonlinearity leads to more variability than when β is small. This is especially true in the double-sided case, where the *function difference* is much more sensitive to the change of nonlinearity than the *function* itself (e.g., when the function $f(x)$ is highly nonsymmetrical about the current x point).

(4) Comparing the single- and double-sided smoothed gradient estimators for small values of β (Figure 3), it is seen that the double-sided estimator has much smaller variability than the single-sided one. This can be explained as follows: for β very small, the function $f(x)$, even if nonlinear, will be quite symmetrical about any specific x; therefore, the difference appearing in (12a′) will be not much dependent on the actual value of the random variable η (compensation effect)—hence, very small variability, even if the $1/\beta^2$ term is quite large. This is not the case for the single-sided case, where there is no compensation and the effective variability is large.

4. COMPARISON OF THE FSA AND SAS ALGORITHMS

4.1. Simulated Annealing (CSA and FSA)

In what follows, the classical simulated annealing (CSA) and fast simulated annealing (FSA) are presented simultaneously, in order to demonstrate their similarities and differences. The algorithms below are *specific* versions presented in (Szu, 1986a, 1987).

ALGORITHM 1 (CSA and FSA algorithm model)

(0) Select the starting point x^0 and other parameters.

(1) Update the cooling temperature:
   (a) for CSA: $T(k) = \alpha/\log(1 + \sigma k)$("slow" cooling),
   (b) for FSA: $T(k) = \gamma/(1 + \sigma k)$ ("fast" cooling), where $T(\cdot)$ is temperature and k the iteration number, where $\alpha = 25$, $\gamma = 100$, $\sigma = 10^{-2}$, as assumed by Szu.

(2) Generate a random variable $\theta^k \in R^n$, according to the p.d.f.'s:
   (a) for CSA: Gaussian p.d.f., using equation (9) with $\beta = \sigma T(k)$,
   (b) for FSA: Cauchy p.d.f. with $\beta = \sigma T(k)$:

$$g(x_1, \ldots, x_n) = \frac{1}{\pi^{(n+1)/2}} \frac{\beta\Gamma^{(n+1)/2}}{(\beta^2 + \sum_{i=1}^{n} x_i^2)^{(n+1)/2}}. \quad \text{(A1.1)}$$

   To obtain this p.d.f. use Algorithm 1.1 below.

(3) Compute the function change:
$\Delta f_k = f_{k+1} - f_k = f(x^k + \theta^k) - f(x^k)$.

(4) Compute the hill-climbing acceptance probability:

$$\rho_k = 1/(1 + \exp(\Delta f_k/T(k))).$$

(5) Generate a uniformly distributed random vari-

able $\gamma_k \in [0,1]$, and:

if $\gamma_k < \rho_k$, let $x^{k+1} = x^k + \theta^k$
(i.e., accept the new point),

if $\gamma_k > \rho_k$, let $x^{k+1} = x^k$
(i.e., reject the new point).

(6) If a stopping criterion is fulfilled (e.g., the maximum permissible number of samples is exceeded)—STOP; otherwise, set $k = k + 1$, go to (1).

The joint multidimensional Gaussian p.d.f. was generated using a standard technique based on mapping the uniform p.d.f. through an inverse of the Gaussian cumulative distribution (Devroye, 1986), along each of the coordinate axes.

The multidimensional Cauchy p.d.f. *cannot* be generated by independently sampling each of the x variables according to the univariate Cauchy p.d.f. A more complicated procedure has to be used instead. Out of several possible methods, the algorithm described in Devroye (1986) was used:

ALGORITHM 1.1 (n-dimensional Cauchy random number generation)

(1) Generate an n-dimensional random vector X uniformly on the surface of the hyperball V_n in R^n (e.g., using a standard IMSL library subroutine (IMSL, 1987)).

(2) (a) Generate the scalar random variate B according to the Beta $\left(\frac{n}{2}, \frac{1}{2}\right)$ distribution.

(b) Calculate $r = \sqrt{B/(1 - B)}$.

(3) Return the product $y = \beta r X$; y is distributed according to the n-dimensional Cauchy p.d.f. (A1.1).

The above subroutine was used in all FSA experiments. In what follows, all examples are concerned with the FSA methodology only, since it has already been demonstrated by Szu (1986, 1987), and other authors (e.g., McBeth, 1988), as well as our own experiments that FSA is much more efficient than CSA.

4.2. Stochastic Approximation with Smoothing (SAS)

One class of methods to solve the modified problem (7)—to be called large-sample (LS) stochastic methods—can be characterized as follows: for each new point x (obtained, e.g., using one of the gradient methods of nonlinear programming), a *large* number of points sampled with the p.d.f. $\hat{h}(\eta, \beta)$ (eqn. (5)) is used to estimate $\hat{f}(x, \beta)$ (according to eqn (4)) and its gradient (according to (11a,b) or (12a,b)). The number of samples used should be sufficiently large to give small errors of the relevant estimators. Therefore, the LS methods (which seem to be intuitively most "natural"), are characterized by the following process: selecting x—averaging (i.e., finding large sample estimators)—selecting x—averaging—and so on. As seen, the two operations, namely, optimization and averaging are separated in this process. This makes it very inefficient.

Optimization and the averaging operations (4), (11a,b), or (12a,b) can be combined into *one* iterative process, leading to much more efficient "small-sample" (SS) methods of stochastic programming. A large class of SS methods, called *stochastic approximation*, was introduced by Robins and Monro (1951) and successfully applied to stochastic function minimization (or maximization) by Kiefer and Wolfowitz (1952) and other authors (see e.g., Kushner and Clark, 1978). Their basic principle of operation is that only a small number of samples (usually just one) is used in each iteration to find the necessary estimators, but all the information is being averaged over *many* steps, which themselves are sufficiently small to guarantee convergence to the solution.

In function minimization applications, the stochastic approximation methods create stochastic equivalents to the classical (e.g., gradient) methods of nonlinear programming. The advanced algorithms used by Styblinski and Ruszczynski (1983), are stochastic analogs of the method of conjugate gradients. To estimate the gradient of $\tilde{f}(x, \beta)$ for each $x = x^k$, they use a few (usually one) random samples. Moreover, they not only provide the standard averaging along the trajectory of x^k (as any other stochastic approximation method does), but also implement *gradient direction* averaging (or filtering), such that the amount of "zigzagging" and "directional noise" is reduced, which leads to greater directional stability and very good efficiency, as demonstrated by Styblinski and Ruszczynski (1983); Styblinski and Opalski (1984); and Tang and Styblinski (1988). As the algorithm progresses, $\beta \to 0$, reducing the degree of smoothing of the $f(x)$ function, and providing convergence to the true minimum.

It is worth pointing out that, in contrast to the SA algorithms which select the new x^k points completely randomly, the SAS algorithm implements (on average) a well-defined approximation to the conjugate gradient-like descent on the $\tilde{f}(x, \beta)$ surface. This is one of the primary reasons for its greater efficiency, as demonstrated below.

The stochastic approximation with smoothing (SAS) algorithm to be discussed, can be described

by the following model:

ALGORITHM 2 (SAS algorithm model)

(0) Select the initial values: $\beta = \beta^0$, $x = x^0$.
(1) Set $m = 0$.
(2) Set $\beta = \beta^m$ (according to a selected sequence: $\{\beta^0, \beta^1, \ldots\} \rightarrow 0$).
(3) Perform one cycle of the stochastic approximation optimization according to Algorithm 3, to find an approximation to $x = \hat{x}$ (the problem solution).
(4) Set $m = m + 1$; if $m =$ MMAX (maximum number of cycles)—STOP, otherwise—go to (2).

The general principles of operation of the stochastic approximation algorithm we used are described below:

ALGORITHM 3 (stochastic approximation algorithm model)

(1) Select $x = x^0$, and the algorithm control parameters:
MAXITER, EPS, STEP, NMCGR.
(2) Calculate the initial step size coefficient τ_0 based on the smoothed gradient $\xi^0 \equiv \hat{\nabla}_x \tilde{f}(x, \beta)$ estimation with NMCGR samples, using eqns (11a, b) or (12a,b), as follows

$$\tau_0 = \frac{\text{STEP}}{\|\xi^0\|}; \qquad \text{(A3.1)}$$

set $k = 0$.
(3) Use the following stochastic approximation iterative scheme, to find an approximation to the solution of the single optimization problem (7) (for a specific value of β):

$$x^{k+1} = x^k - \tau_k d^k, \qquad \text{(A3.2)}$$

$$d^k = (1 - \rho_k) d^{k-1} + \rho_k \xi^k, \quad 0 \le \rho_k \le 1, \qquad \text{(A3.3)}$$

where ξ^k is the gradient estimator found using eqns. (11a,b) or (12a,b) with $N = 1$ in our case, τ_k is the step length coefficient, controlled as described below; ρ_k is the gradient averaging coefficient, typically updated using the following formula:

$$\rho_k = \frac{\rho_{k-1}}{1 + \rho_{k-1} - R}, \qquad \text{(A3.4)}$$

where $0 < R < 1$ is a constant controlling the rate of change of ρ_k. k assumes values: 1, 2, . . . until $k =$ MAXITER or the step length $\|\tau_k d^k\| <$ EPS.

COMMENTS

- In this algorithm model $\{\tau_k\} \rightarrow 0$, and $\{\rho_k\} \rightarrow 0$.
- As seen from (A3.3), each new step is performed in the direction $\tau_k d^k$, where d^k is a convex combination of the previous (old) direction d^{k-1} and a new gradient estimate ξ^k (hence the algorithm is a stochastic analog of the deterministic conjugate gradient method); this provides the gradient direction averaging.
- ρ_k controls the "memory" or "inertia" of the search direction d^k (large inertia for small ρ's).
- τ_k and ρ_k are automatically determined, based on several heuristic principles; their actual implementation is quite complicated (especially in the most advanced algorithm version discussed in Ruszczynski & Syski (1984)), but the basic principles can be briefly characterized by the following heuristic reasoning: τ_k should be (on average) a maximizer of $f(x^{k+1}) = f(x^k + \tau_k d^k)$, hence the scalar product: $E[\langle d^k, \xi^{k+1} \rangle] = 0$ (on average). If $E[\langle d^k, \xi^{k+1} \rangle] < 0$, for x^{k+1}, τ_k is too large, otherwise it is too small. Some statistical tests are used to check the above condition and τ_k is suitably changed.
- From (A3.4) and (A3.3) it follows that R is responsible for the rate of change of ρ_k, that is, it modifies the search direction d^k and provides a suitable amount of inertia of the gradient direction (in other words it controls the amount of filtering of the random noise and zigzagging imposed on the gradient direction due to the random nature of the process and the type of the smoothed function surface). R should be chosen such that (on average) $E[\langle d^{k-1}, \xi^{k+1} \rangle] = 0$, since this condition provides a step toward the bottom of the valley of the $\tilde{f}(x)$ function); if $E[\langle d^{k-1}, \xi^{k+1} \rangle] < 0$, R is too small, otherwise R is too large. Statistical tests, similar to those used to control τ_k are used to determine the actual value of R.[1]

5. EXPERIMENTAL RESULTS

To study the behaviors of FSA and SAS, both approaches have been applied to several examples with diverse function shapes; only the most typical of them are presented here.

5.1. Single Optimization Runs

In all examples to follow, the fast simulated annealing (FSA) method introduced by Szu (1986a) was used. The first four examples show a typical behavior of both algorithms starting from a randomly chosen seed of the r.n.g. used (these are actually the first results we obtained studying both methods).

Specific sequences: $\{\beta\}$ (of the "cooling parameter"), and {MAXITER} (of the maximum permissible number of iterations) for the SAS algorithm are

[1] The above explanations present only the basic philosophy of the actual (Ruszczynski & Syski, 1984) more sophisticated algorithm implementation used in our tests.

given with each example. It turned out that the final solutions were not very sensitive to a specific choice of these sequences, therefore they were determined quite ad hoc, based on rough heuristic criteria such as: "low problem dimensionality requires a smaller number of function evaluations," "β should be large at the beginning of optimization (to determine the approximate location of the global minimum), and small at the end of optimization (for precision)," etc. The selected values are by no means optimal, so there is still room for improvements.

Single optimization runs, as described above, are not sufficient for a fair evaluation of the properties of the FSA and SAS algorithms, which are *stochastic* in nature, and whose convergence can only be proved for an infinite number of function evaluations. The algorithm behavior for a *finite* number of function evaluations is a completely different matter, and can only be tested experimentally, for several random algorithm runs. This was practically implemented starting each new run from a different initial seed of the r.n.g. used. This allowed us to determine the *average* properties of both algorithms. Such tests are demonstrated in Examples 5 and 6.

To make the results of the SAS method comparable with those obtained for the FSA method, only the *function values* were used for the SAS method, applying the double-sided gradient evaluation formula (12a). Therefore, for each iteration *two* function evaluations were performed. The starting points x^0 were always identical for both methods.

EXAMPLE 1. Let the function $f(x), x \in R^2$ be defined as:

$$f(x_1, x_2) = \frac{1}{2}\sum_{i=1}^{2} (x_i^4 - 16x_i^2 + 5x_i). \qquad (13)$$

This function is a 2-D version of the function (6) used by Szu (1986a). It has four local minima, with the global minimum located at $\hat{x}_i = -2.903534$, $i = 1,2$, and one maximum located close to the origin. The starting point was chosen as $x_1^0 = 4.0$, $x_2^0 = 6.4$. The final optimal points $\hat{x}_i$, $i = 1,2$—obtained applying both methods—together with their maximum deviations (Maximum Error) with respect to the coordinates of the true global minimum, are shown in Table 1. The values of β used for each subsequent optimization were as follows: $\{\beta\} = \{3., 1.5, 1., 0.75, 0.6, 0.51, 0.42, 0.05\}$. The corresponding maximum numbers of iterations MAXITER permitted for each optimization run were, respectively, determined by the sequence: {MAXITER} = {100, 100, 100, 100, 100, 100, 100, 100}. Two function evaluations per iteration were performed, as mentioned above. The actual number of iterations was in each optimization run determined by an error criterion used (the average minimum step length).

As seen, the SAS method gives much more accurate results with more than three times smaller number of function evaluations.

TABLE 1
Results of Example 1
(two dimensional)

SAS[1]	FSA[2]
$N_s^3 = 1498$	$N_s = 5000$
−2.900243	−2.667030
−2.902344	−2.950723
Maximum Error	Maximum Error
0.11%	8.1%

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.
[3] Number of function evaluations.

EXAMPLE 2. A large number of small local minima was added to (13), by introducing the cosine terms, as follows (the location of the global minimum is not changed):

$$f(x_1, x_2) = \frac{1}{2}\sum_{i=1}^{2} (x_i^4 - 16x_i^2 + 5x_i) - 10\cos(4(x_1 + 2.093534))\cos(2(x_2 + 2.903534)). \qquad (14)$$

The starting point was: $x_1^0 = 4.0$, $x_2^0 = 6.4$. $\{\beta\} = \{3., 1.5, 0.7, 0.1\}$, and {MAXITER} = {600, 400, 400, 200}. Function (14) is plotted in Figure 5, and the results are shown in Table 2.

As in Example 1, the SAS method is much more accurate and efficient. The existence of additional local minima did not change the behavior of both methods.

EXAMPLE 3. This is a ten-dimensional extension of

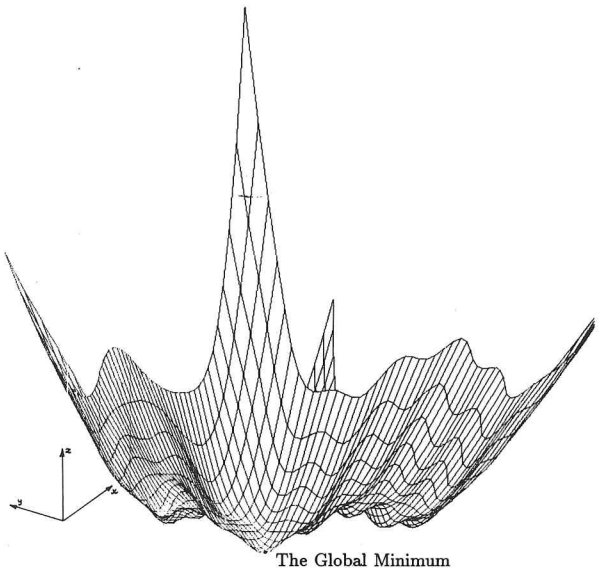

FIGURE 5. Plot of function (14) with a very large number of local minima.

TABLE 2
Results of Example 2 (two dimensional)

SAS[1]	FSA[2]
$N_s^3 = 2940$	$N_s = 5000$
−2.911005	−2.956368
−2.906121	−2.915492
Maximum Error	Maximum Error
0.26%	1.79%

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.
[3] Number of function evaluations.

Example 1, with $f(x)$, $x \in R^n$, defined as:

$$f(x_1, \ldots, x_{10}) = \frac{1}{10} \sum_{i=1}^{10} (x_i^4 - 16x_i^2 + 5x_i). \quad (15)$$

The total number of local minima of (15) is equal to $2^{10} = 1024$. The global minimum is located at $\hat{x}_i = -2.903534$, $i = 1, \ldots, 10$, and one local maximum close to the origin. $x_i^0 = 4.6$, $i = 1, \ldots, 10$. $\{\beta\} = \{5., 2.5, 1., 0.75, 0.1\}$, and $\{\text{MAXITER}\} = \{1000, 1000, 1000, 1000, 1000\}$.

This example is difficult, due to a very large number of local minima, where the function values are quite close to the global minimum value (see Example 5). Therefore, any global optimization method would have difficulties is distinguishing between these minima. A more thorough discussion of this case is given in Example 5. The results are shown in Table 3.

As seen, the SAS method performed *significantly* better in this case. A typical convergence process for both methods (i.e., actual function values corresponding to the subsequent values of x^k) are shown in Figure 6. As seen, the initial convergence of the FSA algorithm, for a large number of iterations (about 10,000 only shown), converges first to a local minimum; even in the range of about 8,000–10,000 iterations large "jumps" of the function values are recorded (see, e.g., the one close to the 10,000th iteration). This is due to the infinite variance property of the Cauchy p.d.f., which provides the locality of sampling with occasional "long jumps," as discussed by Szu (1986a). On the other hand, the SAS smoothing algorithm behaves in a very "stable" way, and it approaches the true global minimum already at about 4,000th iteration (about 8,000 function evaluations). The role of smoothing is clearly seen (moving out of a big local minimum at the beginning of optimization). The large plateau where the algorithm spends a large number of iterations is due to the function maximum occurring about the origin; a large number of iterations is needed at this point to detect the best search direction, based on an excessive gradient averaging from many randomly sampled points in a relatively broad neighborhood of the function maximum (due to still large value of β). Once this direction is found the algorithm accelerates and finds a minimum in a relatively small number of iterations, further refining the solution along its trajectory, as seen from the plot. Interestingly, as can be inferred from its behavior, the algorithm implements automatically the following methodology used in some global optimization schemes: "perform initial rough minimum estimation" (this corresponds to the time spent at the plateau to learn the approximate location of the global minimum), "jump to the most promising area" (acceleration period), and "refine the solution" (the final period of relatively slow convergence).

TABLE 3
Results of Example 3 (ten dimensional)

SAS[1]	FSA[2]		
$N_s^3 = 9228$	$N_s = 10005$	$N_s = 50000$	$N_s = 100000$
−2.899271	−2.742355	−2.617526	−2.727543
−2.900521	−3.798322	−2.914660	−2.745311
−2.902766	−1.698929	−2.937617	−2.827860
−2.895450	2.426616	−2.543482	−3.029721
−2.902119	−2.642577	−3.468538	−2.923877
−2.897861	−1.554575	−2.661655	−3.084294
−2.902274	−2.769585	−2.757708	−2.948352
−2.900122	−2.998929	−3.029382	−3.113862
−2.896537	3.092902	−2.798513	−3.053756
−2.901245	3.951455	−2.818866	−3.210182
Maximum Error	Maximum Error	Maximum Error	Maximum Error
0.24%	236.09%	19.46%	10.56%

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.
[3] Number of function evaluations.

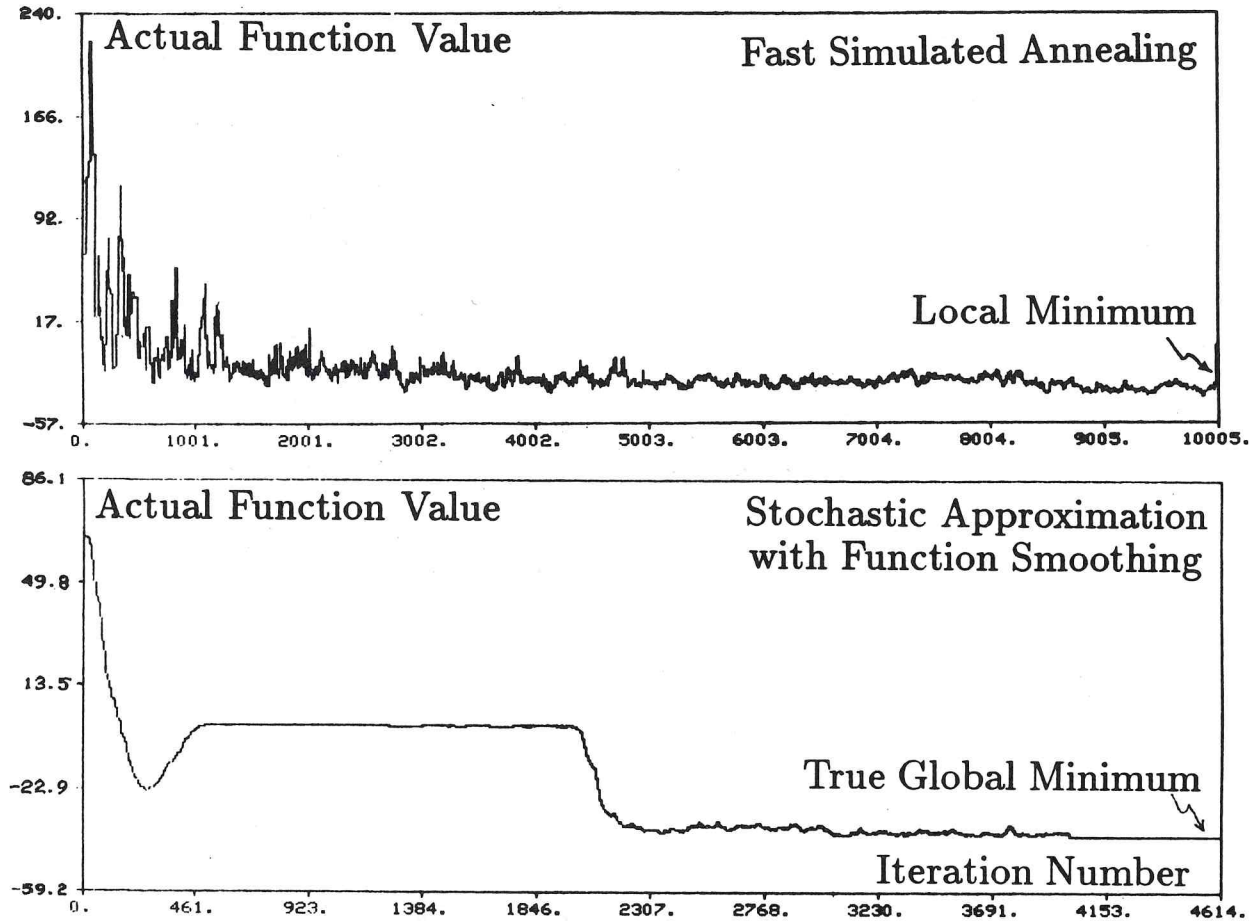

FIGURE 6. Actual function values calculated for the subsequent values of x^k vs. the current iteration number.

EXAMPLE 4. This is a five-dimensional example with the function:

$$f(x_1, \ldots, x_5) = \frac{1}{2}\sum_{i=1}^{2}(x_i^4 - 16x_i^2 + 5x_i) + \sum_{i=3}^{5}(x_i - 1)^2. \quad (16)$$

The function has 4 local minima, and the global minimum located at: $\hat{x}_1 = \hat{x}_2 = -2.903534$, $\hat{x}_3 = \hat{x}_4 = \hat{x}_5 = 1.0$; $x_i^0 = 4.6, i = 1, \ldots, 5$. The purpose of this example is to show the performance of the two methods for the case where the location of the global minimum is not identical along different coordinate axes. The results are shown in Table 4.

As seen, the convergence and accuracy of the SAS method are again superior.

5.2. Statistical Testing of the Algorithm Behavior

The objective of this section is to provide a simple statistical evaluation of the convergence properties of the two algorithms considered, on the difficult multiextremal function (15) used in Example 3. As it will be seen, both algorithms can converge to a local minimum on some occasions; this is unavoidable due to the stochastic nature of the optimization process. Real questions, therefore, are: (1) "What is a percentage of global solutions obtained out of a specific number of different random runs?" (i.e., runs starting from different values of the random number generator seeds, with all other algorithm parameters held constant). (2) "What is the 'quality'

TABLE 4
Results of Example 4
(five dimensional)

SAS[1]	FSA[2]
$N_s^3 = 3710$	$N_s = 10000$
−2.903506	−2.702844
−2.903527	−3.148829
1.000241	1.099552
0.9998547	1.355916
1.000194	1.485936
Maximum Error	Maximum Error
0.32%	48.6%

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.
[3] Number of function evaluations.

of the solutions other than the global?" (i.e., how far are they from the global solution—in some sense). In general, then, a criterion of an algorithm "goodness" is needed to quantitatively rank the algorithm convergence properties.

The first criterion (the percentage of global solutions reached) is simple and relatively easy to implement, even if the definition of "reaching the global minimum" is not very precise (i.e., how close to the global minimum is "close enough"). The second criterion is more difficult to evaluate. Moreover, a test function of special properties is needed (as it will be seen below), to make the evaluation easy and meaningful. There are reasons to believe (as discussed below) that the function (15) fulfills these requirements.

Our basic philosophy is to introduce a notion of subsets of solutions belonging to different "difficulty classes"; the global minimum (or several global minima of the same value, if present) belong to Class 0 (the most difficult). Classes 1, 2, ... are less and less difficult, in a sense, that the number of wrong coordinates (with respect to the global solution) is equal to 1, 2, ... , etc. The reason why such an approach is possible, is due to the fact that for the special test function (15) all the minima must have their coordinates equal either to $\hat{x}_i = -2.903534$, or $\hat{x}_i = 2.746803$—but the global solution is located at $\hat{x}_i = -2.903534$, $i = 1, \ldots, 10$ (all coordinates equal). All the remaining local minima have some combinations of the two values of $\hat{x}_i$. The smaller the *number* of "wrong coordinates" the better, since their particular combination is not important. This is due to the following: within a specific difficulty class the function values at all possible local minima are equal to each other, irrespective of the specific combination of the wrong coordinates. This is, obviously, a result of the sum form of the function (15), where each of the sum components is a function of just one variable. The higher the class number, the larger (i.e., the worse) the function value, if a true local minimum is reached during the optimization process. (Note: If a true local minimum is not reached, the classification to a specific class and the above conclusions are only approximate.)

The worst possible solution is in Class 10, where all coordinates are equal to $\hat{x} = 2.746803$. Since the starting point is $x_i^{(0)} = 4.6$, $i = 1, \ldots, 10$ (positive), the algorithm is first forced to pass through the vicinity of the worst ("all-positive-coordinates") solution, before it can reach the global ("all-negative-coordinates") solution. Taking this into account, the class number also says, in a sense, "how far was the algorithm able to go."

The quality of a specific algorithm can now be characterized by the percentage of the solutions of different classes obtained, for a specific number of runs. Let the numbers 0, 1, 2, . . . , n denote the classes 0, 1, 2, ... , n, where n is the highest-class number; these numbers are actually the values of a discrete random variable $\mathbf{c}$ assuming specific values $c_i = i$, $i = 0, 1, \ldots, n$. Let $p_i = M_i/M$ (where M_i is the number of solutions falling into the ith class, and M the total number of runs) denote the fraction of solutions falling into the ith class. Thus, p_i is the *sample probability* that $\mathbf{c} = c_i$, that is, the sample probability of a solution falling into the ith difficulty class, based on M random runs. This probability function can be then plotted, as a histogram, and used to compare different algorithms, or algorithm realizations.

In cases, where just *one number* is needed to characterize an algorithm, the following single measure of global convergence quality (MGCQ) is introduced as the sample average of the random variable $\mathbf{c}$:

$$\text{MGCQ} = \sum_{i=0}^{n} p_i c_i, \tag{17}$$

The best value of MGCQ is zero (all solutions are the global solutions), the worst is n (all solutions are at the worst local minimum). In order to characterize the dispersion of $\mathbf{c}$, the sample variance (or, better standard deviation) can also be used. This last parameter characterizes the amount of variability in the algorithm convergence to different local minima.

The above criteria are, by necessity, somehow heuristic, since general statistical methods of testing global optimization algorithms are not available at this writing (actually, most of the papers available in this area are concerned with global convergence properties when the sample sizes and the number of algorithm steps tend to infinity; no references are made to the behavior of different algorithms for finite sample sizes and a finite number of steps as required for practical applications).

EXAMPLE 5. As discussed above, the test function (15) was used to statistically evaluate the algorithm convergence properties. A total of 30 runs (10 runs in one case) were performed, starting with different r.n.g. seed values, for both FSA and SAS algorithms. For the FSA algorithm the basic cooling scheme (see step (1b) of Algorithm 1) was first used, with $\gamma = 100$ and $\sigma = 0.01$, and then different rates of cooling tried. For the SAS algorithm the following sequences were used: $\{\beta\} = \{5, 3, 1, 0.1, 0.01\}$, and {MAXITER} = {1000, 1000, 1000, 1000, 1000} (see Example 1). The total maximum number of function evaluations was limited for both methods to $N_s = 10{,}000$ (100,000 in one case). The results are summarized in Tables 5–9 and Figures 7 and 8, showing the histograms of p_i for the two methods.

TABLE 5
Results of Example 5 (ten dimensional) for Stochastic Approximation with Smoothing (SAS) Algorithm Based on 30 Random Runs

Class	0	1	2
p_i	$\frac{22}{30}$	$\frac{5}{30}$	$\frac{3}{30}$
Maximal Coordinate Error	0.25%	0.08%	0.07%
Average Function Value	−78.33218	−75.50499	−72.67763
Error of Average Function Value	2.3×10^{-4}%	1.6×10^{-4}%	4.1×10^{-5}%
Actual Functional Value at Minimum	−78.33236	−75.5051	−72.67767

Discussion. It is seen from the function values for the minima of different classes (Tables 5–9) that the problem is difficult, since the differences between those values are small (about 4% between the global minimum and the best local minimum). Therefore, optimization algorithms will have difficulties in distinguishing between these minima, especially that their total number is equal to 1,024.

The SAS method is clearly superior in this example: the number of correct SAS global solutions is 22 out of 30 runs (only 3 for FSA, for the better of two cases with $N_s = 10{,}000$); the accuracy of the SAS solutions is excellent, with the maximum coordinate error of *all* runs equal to 0.25%, and the average function value accuracy in the order of 10^{-4} (for FSA the best coordinate accuracy is 17.38%, but was also as bad as 92.6%, and the best function value accuracy was 2.98%).

For FSA, increasing the cooling rate two times (from $\sigma = 0.01$ to 0.02) *reduced* the number of global solutions from 3 (10%) to 1 (3.33%) out of 30 runs (Table 7), and increased the MGCQ measure from 1.73 to 2.53 (Table 9). However, the maximum coordinate error, and the average error of the corresponding locally minimal function values was significantly reduced. Therefore, the fast ("inversely linear in time") cooling scheme proposed by Szu (1986) (see Algorithm 1), leads to a trade-off between the global and local algorithm convergence properties (better global convergence → poorer local convergence and vice versa). Many other choices of the parameters of the inverse linear cooling scheme we tried did not generate any better results than those presented in Tables 6 and 7.

After the number of FSA functions evaluations was increased to $N_s = 100{,}000$, with $\sigma = 0.01$, the number of global solutions increased to 3 out of 10 runs tried in this case (30%), but the MCGQ measure stayed about the same as for $N_s = 10{,}000$ (1.8 vs. 1.73), due to a relatively large number of poor local solutions obtained. (Note: Due to the reduced number of runs from 30 to 10 in the last case, some caution should be exercised in interpreting these results.)

Using the MGCQ measure and its dispersion as a basis for the method comparison, it is seen that MGCQ is about 5 times and its standard deviation at least 2 times smaller for SAS than for FSA. These results visualize, in a condensed form, the better (for this example) global convergence properties and smaller variability of the final results for the SAS method.

EXAMPLE 6. The objective of this example is to study the algorithm behavior on a ten-dimensional multimodal function which is a "multimodally corrupted" version of a simple convex quadratic function defined as follows:

$$f(x_1, \ldots, x_n) = \frac{1}{2n} \sum_{i=1}^{n} x_i^2 - 4n \prod_{i=1}^{n} \cos(x_i), \quad (18)$$

where $n = 10$ in our case. This function has the global minimum at the origin, and a large (unknown) number of local minima, whose "deepness" and "fre-

TABLE 6
Results of Example 5 (ten dimensional) for Fast Simulated Annealing (FSA) Algorithm Based on 30 Random Runs, with $N_s = 10{,}000$ (total number of function evaluations), and $\sigma = 0.01$ (standard cooling—as in Szu (1986))

Class	0	1	2	3	4	5	6
p_i	$\frac{3}{30}$	$\frac{13}{30}$	$\frac{9}{30}$	$\frac{1}{30}$	$\frac{3}{30}$	$\frac{0}{30}$	$\frac{1}{30}$
Maximal Coordinate Error	79.10%	92.63%	65.40%	76.72%	44.09%		34.41%
Average Function Value	−73.835	−68.710	−67.244	−63.202	−61.276		−57.423
Error of Average Function Value	6.10%	9.89%	8.08%	10.52%	9.38%		6.78%
Actual Functional Value at Minimum	−78.33236	−75.5051	−72.67767	−69.85033	−67.02297	−64.19563	−61.36829

TABLE 7
Results of Example 5 (ten dimensional) for Fast Simulated Annealing (FSA) Algorithm Based on 30 Random Runs, with N_s = 10,000 (total number of function evaluations), and σ = 0.02 (two times faster cooling)

Class	0	1	2	3	4	5	6
p_i	$\frac{1}{30}$	$\frac{6}{30}$	$\frac{10}{30}$	$\frac{5}{30}$	$\frac{5}{30}$	$\frac{3}{30}$	$\frac{0}{30}$
Maximal Coordinate Error	29.87%	42.57%	90.06%	34.05%	27.13%	17.38%	
Average Function Value	−76.077	−72.334	−68.904	−67.399	−64.892	−62.211	
Error of Average Function Value	2.96%	4.38%	5.48%	3.64%	3.28%	3.19%	
Actual Functional Value at Minimum	−78.33236	−75.5051	−72.67767	−69.85033	−67.02297	−64.19563	−61.36829

quency'' depend on a specific choice of the coefficients of the sinusoidal part. To give an idea of the function behavior, its two-dimensional version is plotted in Figure 9.

Ten runs were performed for both SAS and FSA algorithms. For the SAS algorithm we used $\{\beta\}$ = {4, 2, 1, 0.5, 0.25, 0.1, 0.01}, and {MAXITER} = {500, 500, 500, 500, 500, 500, 500}. For FSA the same cooling scheme as before (with β = 0.01) was used. The total maximum number of function evaluations was limited to N_s = 7,500 for SAS and to 50,000 for FSA.

In Table 10 the final function values for each of the 10 runs are given. As seen, the accuracy in reaching the global minimum of −40.000 is almost perfect for the SAS algorithm, with the total actual average number of function evaluations equal to only 6,620 (this was due to the stopping criterion used). On the other hand the FSA method used 50,000 function evaluations, with much more inferior results as far as the final function values were concerned. The results suggest that the region of the global function minimum has never been reached in this case. Also, the variability of the FSA results is about 28 times greater than those of the SAS method (compare the standard deviations of the final function values for both methods listed in Table 10).

6. CONCLUSIONS

In this paper the method of fast simulated annealing, applied to global minimization of a multiextremal function was compared with an approach based on stochastic approximation and convolution function smoothing. It has been shown that:

(a) Simulated annealing (or FSA) and stochastic approximation with smoothing, are both based on random sampling about a current vector x^k of the independent variables, and both reduce the magnitude of perturbations as the algorithm progresses towards the solution (a "cooling" scheme).
(b) SA (or FSA) and SAS *differ* in the way the next point x^{k+1} is selected: in the SA algorithm the direction is chosen entirely at random, with a finite but decreasing probability of accepting a direction of function *increase*; the SAS algorithm relys upon the function smoothing principle, and uses the negative of an "averaged" gradient direction of the smoothed functional, obtained from a current and many *previous* iterations, i.e., more information is used in each step.
(c) In the global optimization experiments performed with complicated multiextremal func-

TABLE 8
Results of Example 5 (ten dimensional) for Fast Simulated Annealing (FSA) Algorithm Based on 10 Random Runs, with N_s = 100,000 (total number of function evaluations), and σ = 0.01 (standard cooling—as in Szu (1986))

Class	0	1	2	3	4	5	6
p_i	$\frac{3}{10}$	$\frac{3}{10}$	$\frac{0}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{0}{10}$
Maximal Coordinate Error	17.85%	13.38%		13.39%	6.72%	16.31%	
Average Function Value	−77.5615	−74.6129		−68.9350	−66.8273	−62.8909	
Error of Average Function Value	0.99%	1.20%		1.33%	0.29%	2.07%	
Actual Functional Value at Minimum	−78.33236	−75.5051	−72.67767	−69.85033	−67.02297	−64.19563	−61.36829

TABLE 9
Measure of Global Convergence Quality (MGCQ) and Its Standard Deviation for Different Cases Studied in Example 5

		FSA[2]		
Cases	SAS[1]	$\sigma = 0.01$ $N_s = 10{,}000$	$\sigma = 0.02$ $N_s = 10{,}000$	$\sigma = 0.01$ $N_s = 100{,}000$
MGCQ	$\frac{11}{30} \cong 0.3667$	$\frac{52}{30} \cong 1.7333$	$\frac{76}{30} \cong 2.5333$	$\frac{18}{10} = 1.8$
S.D.[3]	0.6574	1.3149	1.3345	1.7205

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.
[3] Standard deviation.

tions, the SAS technique has proved to be always much more efficient and accurate than the FSA method (which is itself faster than the classical simulated annealing).

We believe that the promising results obtained for the different examples studied provide a sufficiently strong evidence of the SAS method utility for solving many nonconvex global optimization problems. They also give a strong hint that serious theoretical and practical studies of these methods should be undertaken. This is especially important as far as the small sample convergence properties of these methods are concerned, since the infinite sample convergence proofs are already available for several special cases (see, e.g., Rubinstein (1981); Styblinski and Opalski (1984)). The knowledge of small sample convergence properties is essential for efficient practical applications, but it has not yet been investigated in the available literature. This should be done for different classes of functions (e.g., "multimodally corrupted" convex quadratic functions), since a general approach will most likely be theoretically intractable.

More experimental studies are required in order to develop good heuristic rules for controlling various parameters, such as β (cooling strategies), MAXITER (maximum permissible number of samples), stopping criteria, etc. These parameters should be *automatically* adjusted during several optimization runs (as it was implemented in the examples shown in this paper), or (better) *dynamically* during a *single* optimization run.[2] The statistical criteria introduced in this paper, and especially the measure of global convergence quality (MGCQ) and its variance can conveniently be used for automated comparison of various strategies, and selecting the best one for a given class of applications. Also, an optimal choice of the type of the kernel p.d.f. $\hat{h}(\eta)$—which might be problem dependent—requires further investigation.

More research is also needed to identify the limitations of the proposed approach. It is felt, for instance, that the SAS method might have difficulties in locating a narrow and deep global minimum "well" on a relatively flat area, especially starting from a neighborhood of a broad local minimum, due to a low probability of sampling a point in the region

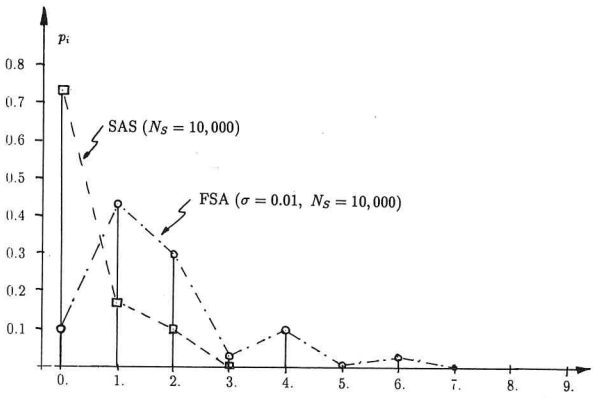

FIGURE 7. Example 5: Sample probability p_i of a solution falling into the ith class for the SAS and the FSA methods, based on 30 random algorithm runs and the maximum total permissible number of function evaluations equal to 10,000 for both methods. The results from Tables 5 and 6 were used.

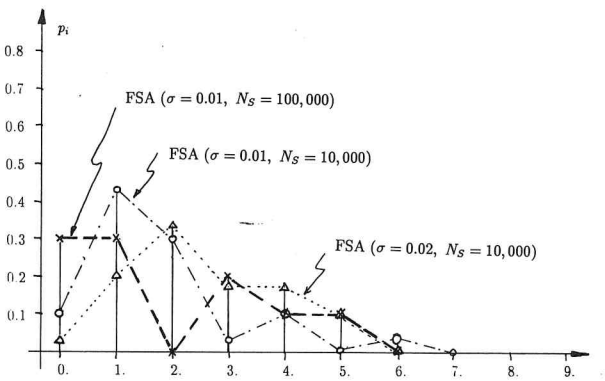

FIGURE 8. Example 5: Sample probability p_i of the solution falling into the ith class for different variants of the FSA method, based on the results from Tables 6–8.

[2] We have tried this method already with a cooling scheme similar to the "inversely linear in time" strategy of FSA, with very encouraging results; more studies in this direction are needed.

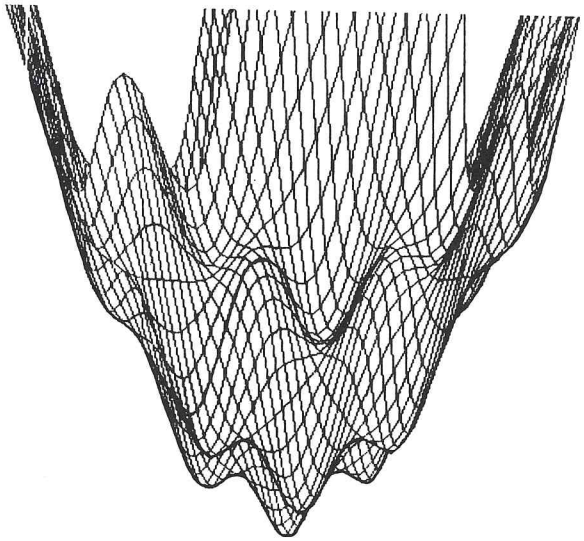

FIGURE 9. 2-D version of function (11) for Example 6.

of attraction of the narrow global minimum (this is, of course, also a limitation of other probabilistic methods). On the other hand, for *technical* applications, such as circuit or system design, locating the nominal point of the designable parameters in a narrow global minimum is not very attractive, due to a locally large sensitivity of the system performance to small technological parameter variations. This would lead to a great variability of the product performance, which is usually unacceptable in practice.

It has also to be noticed that the SAS approach requires a larger number of optimization parameters to be controlled (four: β, STEP, EPS, R) than the FSA algorithm (two: γ and σ). This is inconvenient for the user, but it also gives more flexibility in algorithm tuning to a specific problem.

The SAS methodology is important due to its potential practical applications: as an alternative to the Boltzman machine (Hinton & Sejnowski, 1986) and back propagation approaches to learning connection strength for multilayer neural networks, for solving combinatorial problems (applying mapping of the discrete-space problem into a continuous space of solutions—as implemented by Hopfield and Tank (1985) for the solution of the Traveling Salesman Problem using a neural network approach), for global optimization of electronic circuits where multiextremal objective functions are a rule rather than an exception, and other practical problems. More research in these areas is envisaged.

TABLE 10
Results of Example 6 (ten dimensional); Final Function Values for 10 Random Runs are Listed

Run	SAS[1]	FSA[2]
1	−39.6815	−24.8972
2	−40.0000	−15.5196
3	−40.0000	−28.9899
4	−40.0000	−32.0695
5	−40.0000	−30.1582
6	−40.0000	−30.0533
7	−39.9846	−27.8960
8	−39.4235	−20.9150
9	−40.0000	−28.4537
10	−40.0000	−34.6122
Average Value	−39.90895	−27.35646
Standard Deviation	0.1871	5.3084
Average Number of Function Evaluations	6620	50,000

[1] Stochastic approximation with smoothing.
[2] Fast simulated annealing.

REFERENCES

Devroye, L. (1986). *Non-uniform random variate generation*. New York: Springer-Verlag.

Hinton, G. E. & Sejnowski, T. J. (1986), Learning and relearning in Boltzman machines. In: *Parallel distributed processing*, Vol. 1, Ed. by: Rumelhart, D. E. and McClelland, J. L., Cambridge, Mass.: The MIT Press, pp. 282–317.

Hopfield, J. J., & Tank, D. W. (1985). 'Neural' computation of decisions in optimization problems. *Biological Cybernetics*, **52**, 141–142.

IMSL (1987). *User's Manual: STAT/LIBRARY Fortran Subroutines for Statistical Analysis*. IMSL Inc., Houston, TX.

Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, **23**, 462–466.

Kirkpatrick, S., Gelatt, C. D., Jr., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680.

Kushner, H. K., & Clark, D. S. (1978). *Stochastic approximation methods for constrained and unconstrained systems*. New York: Springer-Verlag.

McBeth, J. H. (1988). *Simulated Annealing*. General Dynamics, Electronics Division, MZ 7202-K, P.O. Box 85310, San Diego, CA 92138 (pre-print of a conference publication).

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, **21**, 1087–1092.

Opalski, L. J., & Styblinski, M. A. (1984). A user-reprogrammable interactive interface system for computer-aided optimization of ICs. *Proceedings of the 27th Midwest Symposium on Circuits Systems* (pp. 587–590), Morgantown, WV, June 11–12.

Robins, H., & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, **22**, 400–407.

Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo method*. New York: John Wiley.

Ruszczynski, A., & Syski, W. (1984). Stochastic approximation algorithm with gradient averaging and on-line step-size rules. *9th World Conference of the IFAC* (Vol. VII, pp. 230–234), Budapest, Hungary, July 2–6.

Styblinski, M. A., & Opalski, L. J. (1984). A random perturbation method for IC yield optimization with deterministic process parameters. *Proceedings of the International Symposium on Circuits and Systems* (pp. 977–980), Montreal, Canada, May 7–10.

Styblinski, M. A., & Ruszczynski, A. (1983). Stochastic approximation approach to statistical circuit design. *Electronics Letters*, **19**(8), 300–302.

Szu, H. (1986a). Fast simulated annealing. *AIP Conference Proceedings 151: Neural Network for Computing*, Snowbird, UT.

Szu, H. (1986b). Non-convex optimization. *Proceedings of the SPIE*, Real time signal processing (Vol. 698, IX pp. 59–65).

Szu, H. (1987). Nonconvex optimization by fast simulated annealing. *Proceedings of the IEEE* (Vol. 75, No. 11, pp. 1538–1540).

Szu, H., & Hartley, R. (1987). Fast simulated annealing. *Physics Letters A*, **122**(3,4), 157–162.

Tang, T. S., & Styblinski, M. A. (1988). Yield optimization for non-differentiable density functions using convolution techniques. *IEEE Transactions on CAD of IC and Systems* (Vol. 7, No. 10, pp. 1053–1067).

van Laarhoven, P. J. M., & Aarts, E. H. L. (1987). *Simulated annealing: Theory and applications*. Hingham, MA: Kluwer Academic Publishers.

Wong, D. F., Leong, H. W., & Liu, C. L. (1988). *Simulated annealing for VLSI design*. Hingham, MA: Kluwer Academic Publishers.