

COMPARISONS AMONG STOCHASTIC
OPTIMIZATION ALGORITHMS

By

DEBAO CHEN

Bachelor of Science

Shanghai Institute of Education

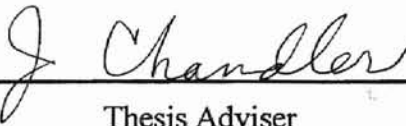
Shanghai, China

1984

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1997

COMPARISONS AMONG STOCHASTIC
OPTIMIZATION ALGORITHMS

Thesis Approved:



Thesis Adviser



H. Lu



Dean of the Graduate College

ACKNOWLEDGMENTS

First and foremost I am deeply grateful to Dr. John P. Chandler, my supervisor, for introducing me to such a beautiful area, for his encouragement, patience, and assistance throughout my studies in the Computer Science Department, and for his guidance in this research and the writing of this thesis.

My appreciation extend to my other committee members, Dr. K. M. George and Dr. H. Lu, as well. Their patience and guidance in reviewing this thesis were invaluable.

I am forever indebted to my wife Yijun Huang and our daughter Xuejing Chen, who gave me wholehearted support in order to make this work possible.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION	1
2. SOME STOCHASTIC ALGORITHMS.....	6
2.1 Random Pattern Search Algorithm.....	6
2.2 Torus Algorithm.....	7
3. SIMULATED ANNEALING ALGORITHMS.....	9
3.1 General Simulated Annealing Algorithms.....	9
3.2 Boltzmann Annealing (BA).....	11
3.3 Fast Annealing (FA).....	11
4. VERY FAST SIMULATED REANNEALING (VFSR) AND THE ASA CODE.....	13
4.1 Very Fast Annealing.....	13
4.2 Reannealing.....	15
4.3 ASA Code.....	18
5. TEST PROBLEMS.....	20
6. TEST RESULTS.....	25
6.1 Results of Random Pattern Search Algorithm.....	25
6.2 Results of Torus Algorithm.....	26
6.3 Results of SA.....	29
6.4 Results of ASA.....	29
6.5 Comparisons.....	32
7. CONCLUSIONS.....	37
REFERENCES.....	39
APPENDIX: PROGRAM LIST FOR TORUS.F.....	43

LIST OF TABLES

TABLE	PAGE
TABLE 6-1 RESULTS OF RANDOM SEARCH ALGORITHM	25
TABLE 6-2 RESULTS OF TORUS PROGRAM.....	27
TABLE 6-3 RESULTS ON OSBORNE FUNCTION 1	28
TABLE 6-4 RESULTS ON OSBORNE FUNCTION 2	28
TABLE 6-5 RESULTS OF CORANA'S ALGORITHM	29
TABLE 6-6 RESULTS OF ASA PROGRAM.....	31
TABLE 6-7 RESULTS ON OSBORNE FUNCTION 1	31
TABLE 6-8 RESULTS ON OSBORNE FUNCTION 2	32
TABLE 6-9 RESULTS ON ROSENBROCK FUNCTION.....	32
TABLE 6-10 RESULTS ON MODIFIED ROSENBROCK FUNCTION 1.....	33
TABLE 6-11 RESULTS ON BOHACHEVSKY FUNCTION.....	33
TABLE 6-12 RESULT ON POWELL FUNCTION	34
TABLE 6-13 RESULTS ON WOOD FUNCTION.....	34
TABLE 6-14 RESULTS ON BEALE FUNCTION	34
TABLE 6-15 RESULTS ON ENGVALL FUNCTION	35
TABLE 6-16 RESULTS ON OSBORNE FUNCTION 1	35
TABLE 6-17 RESULTS ON OSBORNE FUNCTION 2	36

1. Introduction

Global optimization is concerned with the determination of global optima, either maxima or minima, of a function. Such problems occur frequently in numerous disciplines which model real world systems. Mathematically, the global optimization problem can be defined as [15,41]:

Given a set $D \subseteq \mathbb{R}^n$, and given a real function $f: D \rightarrow \mathbb{R}$, find

$$\min_{x \in D} f(x).$$

Here f is called the objective function, or cost function. Since maximizing f is equivalent to minimizing to $-f$, this definition sufficiently includes the search for global maxima as well as global minima. Throughout this thesis, the function f will be referred to as our cost function, unless we mention otherwise.

Besides its importance, global optimization is also an extremely difficult problem. Various methods have been proposed to solve global optimization problems. However, there are no efficient algorithms which solve all general global optimization problems. In general, optimization algorithms can be classified as either stochastic or deterministic. In this thesis, we mainly consider the stochastic methods, which evaluate the cost function f at randomly sampled points from the feasible region D .

There are various stochastic methods [43]. We mainly discuss the simulated annealing method [3,5,27,30] and its variants [16,42]. Two other stochastic optimization algorithms are also discussed. One is the random pattern search algorithm [28,29],

which is one of the early stochastic algorithms. The other is the torus algorithm [39], which is a recent stochastic algorithm.

Originally, the simulated annealing algorithm was based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimization problems [27]. In condensed matter physics, annealing denotes a physical process in which a solid in a heat bath is heated up by increasing the temperature of the heat bath to a maximum value at which all particles of the solid to some extent randomly arrange themselves, followed by cooling through slowly lowering the temperature of the heat bath. In this way, all particles arrange themselves in the low energy ground state of a corresponding lattice, provided the maximum temperature is sufficiently high and the cooling is carried out sufficiently slowly. At each temperature value T , the solid is allowed to reach thermal equilibrium, characterized by a probability of being in a state with energy given by the Boltzmann distribution [30,32].

As the temperature decreases, the Boltzmann distribution concentrates on the states with lowest energy and finally, when the temperature approaches zero, only the minimum energy states have a non-zero probability of occurrence. However, it is well known [26] that if the cooling is too rapid, i.e., if the solid is not allowed to reach thermal equilibrium for each temperature, metastable amorphous structures can be reached rather than the low energy crystalline lattice structure.

The above ideas can be applied to combinatorial optimization problems in the following way [27]: First, a new point is randomly sampled. If it generates a new

minimum value, the new point is always accepted. If not, it is accepted provided that a random number between 0 and 1 is less than a probability defined by a mathematical function, usually the Boltzmann equation [32]. Early in the iterative process, the mathematical function generates values near unity, and most points are accepted. By adjusting a parameter in the probability function, usually referred to as the temperature, the probability function generates smaller values across successive iterations (cooling), and eventually, only points that produce better solutions are accepted. The procedure generates approximately optimal solutions for combinatorial problems, such as the traveling salesman problem, for which exact solutions are presently mathematically intractable [27].

Bohachevsky et al. [3], Corana et al. [5], and others extended simulated annealing ideas from combinatorial problems to the optimization of functions defined in a continuous domain. The mathematical model of the simulated annealing algorithm for continuous optimization based on the ergodic theory of Markov chains can be found in [8].

Szu and Hartley [42] introduced a fast simulated annealing. Fast annealing (FA) uses the Cauchy distribution, and is often superior to that of Boltzmann annealing. The fatter tail of the Cauchy distribution allows it to test states farther from the current local minima during the search process. In addition, fast annealing has an annealing schedule exponentially faster than the method of Boltzmann annealing.

Boltzmann annealing and fast annealing have distributions which sample infinite ranges, and there is no provision for considering differences in each parameter-dimension. Ingber [16] proposed a new probability distribution to accommodate these desired features. This algorithm is called very fast annealing, and is another variant of simulated annealing and is exponentially faster than fast annealing. The details are discussed in Chapter 4.

Various other stochastic algorithms have appeared in the literature [43]. Lee [29] studied a stochastic algorithm, called the random pattern search algorithm, which was first described by Lawrence and Steiglitz [28]. Rabinowitz [39] presented a stochastic algorithm called the torus algorithm, for finding the global optimum of a function of n variables. The performance of this algorithm was compared to that of the Nelder-Mead simplex algorithm [34] and the simulated annealing algorithm on a variety of nonlinear functions.

Many comparisons among different algorithms have been discussed by authors, for example, see [20,21,24,29,35,39,40]. There is no known optimization algorithm which is better than all other algorithms. For example, if one knows that a cost function to be optimized is unimodal and smooth everywhere, then a simple Newton iteration [41] might well be faster than most of the other optimization methods, deterministic or stochastic. Many stochastic algorithms perform poorly on ill-conditioned smooth functions, and are most useful on discontinuous, nonsmooth, or multi-modal problems that are not too ill-conditioned.

The main purpose of this thesis is to compare the simulated annealing algorithm, very fast simulated annealing algorithm, random pattern search algorithm, and torus algorithm on a variety of multi-modal, discontinuous, and/or ill-conditioned functions. In particular, we will test the Osborne functions [36], which are moderately ill-conditioned, smooth practical problems easily solved by deterministic methods [36]. We have not seen a satisfactory solution of optimizing Osborne functions given by any of the early stochastic algorithms [3,5]. Our test results show that these problems can be solved by the very fast annealing algorithm, the random pattern search algorithm, and the torus algorithm.

In Chapter 2, the random pattern search algorithm and the torus algorithm are explained. In Chapter 3, we discuss the general simulated annealing algorithms as well as Boltzmann annealing and fast annealing. In Chapter 4, we discuss very fast annealing in detail. In Chapter 5, we list the functions which we use to compare the algorithms. The comparisons, results, and conclusions are summarized in Chapter 6 and Chapter 7 of this thesis.

2. Some Stochastic Algorithms

In the past decade, simulated annealing algorithms have been studied in detail. In Chapter 3 and Chapter 4, we discuss the simulated annealing method and its variants. Besides that, there are various other stochastic programming methods proposed by various authors. There are collected more than two thousands references in [43], though it is not exhaustive, in this subject. In this chapter we study two stochastic algorithms, the random pattern search algorithm [28,29] and the torus algorithm [39].

2.1 Random Pattern Search Algorithm

One of the early stochastic optimization algorithms is the random pattern search algorithm, which was first described by Lawrence and Steiglitz [28]. It was successfully applied to the optimization of a variety of chemical engineering problems [12].

Lee [29] studied the random pattern search, and modified the random search procedure. He tested this procedure on various functions and compared its performance with the pattern search method [14] and DFP [6,9] method on several aspects, such as the final value of the objective function, the number of iterations of the algorithm, and the number of times the function is evaluated. Recently, Chandler [4] has tried unsuccessfully to reproduce Lee's results using Lee's program.

The random search algorithm came from a deterministic method, called pattern search, which was devised by Hooke and Jeeves [14].

The pattern search algorithm consists of two major phases, an exploratory move and a pattern move. The exploratory phase, moving from the base point, is designed to

explore the local behavior of the objective function. The pattern phase steps along the approximate negative gradient direction, which is determined from the results of the exploratory phase.

During the exploratory phase, if there are instances of successive successful searches, the step size will be extended. Should the next search with this expanded step size be successful, it is retained; otherwise the step size before extension will be applied again. If an unsuccessful search is encountered, the step size is decreased; if it becomes less than some small preset tolerance, then convergence is assumed to be achieved.

The random pattern search basically has the same searching procedures as pattern search, except that it searches in pseudorandom directions uniformly distributed over the surface of a sphere or hypersphere.

2.2 Torus Algorithm

Rabinowitz proposed a stochastic algorithm, called the torus algorithm, for finding the global optimum of a function of n variables [39].

Three computer functions (Controlling function, Multidimensional function, and Single-dimension function) constitute the core of the algorithm. The very detailed pseudo-codes of these three functions were given [39]. The algorithm is based on using an adaptive n -dimensional torus to surround and isolate the global minimum. In the controlling function, an n -dimensional torus moves in n -space and monotonically shrinks in size over trials, isolating the region containing the global minimum. This shrinkage is gradual, mimicking slow cooling in a multidimensional function and single-dimension

function which are repeatedly called from the controlling function; new points are randomly sampled around the currently best-fitting point. The permitted range of these sampled points shrinks logarithmically over iterations (mimicking rapid cooling). Detailed descriptions of the relevant user-specified parameters were given. However, it seems that it would be difficult to implement the algorithm independently [33]. The original program was written in Common Lisp [44]. In fact, there are some differences between the pseudo-codes in [39] and the Lisp program [44]. We translated the Lisp program into a FORTRAN program (see Appendix).

3. Simulated Annealing Algorithms

3.1 General Simulated Annealing Algorithms

In general, simulated annealing consists of three functional relationships [16]:

1. $g(x)$: The probability density function of the state space $x = \{x^i : i=1, 2, \dots, D\}$.
2. $h(x)$: The probability density function for accepting a new value given the just previous value.
3. $T(k)$: An annealing temperature (T) schedule in annealing-time step k .

General simulated annealing optimization methods choose new points at various distances from their current point x . Each new point x_{new} is generated probabilistically according to a given distribution g . These algorithms calculate the function value $E = f(x)$, and then probabilistically decide to accept or reject it. If accepted, the new point becomes the current point. The new point may be accepted even if it is worse and has a larger function value than the current point. The criterion for acceptance is determined by the acceptance function h , the temperature parameter T , and the difference in the function values of the two points. Initially, the temperature T is large. As the algorithm progresses, T is reduced, thus lowering the probability that the acceptance function will accept a new point if its function value is greater than that of the current point.

Let $E_k = f(x)$. The acceptance probability is based on the chances of obtaining a new state with “energy” E_{k+1} relative to a previous state with “energy” E_k .

$$h(\Delta E) = \frac{\exp(-E_{k+1}/T)}{\exp(-E_{k+1}/T) + \exp(-E_k/T)}$$

$$\begin{aligned}
&= \frac{1}{1 + \exp(\Delta E/T)} \\
&\approx \exp(-\Delta E/T),
\end{aligned} \tag{1}$$

where ΔE represents the “energy” difference between the present and previous values of the energies (considered here as cost functions) appropriate to the physical problem, i.e., $\Delta E = E_{k+1} - E_k$. This essentially is the Boltzmann distribution contributing to the statistical mechanical partition function of the system [32]. However, one may choose another function as the acceptance function [22].

Suppose the function g is given. Let the state-generating probability at the cooling temperature $T(k)$ at the annealing-time k and within a neighborhood be $\geq g_k$; then the probability of not generating a state in the neighborhood is obviously $\leq (1 - g_k)$. Our purpose here is to choose suitable $T(k)$ such that it will suffice to give a global minimum of the cost function. In order to statistically guarantee to obtain the global minimum, we require that any point in x -space can be sampled infinitely often in annealing-time (IOT). It suffices to prove that the products of probabilities of not generating a state x IOT for all annealing-times successive to k_0 yield zero,

$$\prod_{k=k_0}^{\infty} (1 - g_k) = 0. \tag{2}$$

This is equivalent to

$$\sum_{k=k_0}^{\infty} g_k = \infty. \tag{3}$$

If the probability density function g is given, then the problem reduces to finding $T(k)$ to satisfy Equation (3).

3.2 Boltzmann Annealing (BA)

The Boltzmann algorithm chooses a Gaussian probability density function [32],

$$g(x) = (2\pi T)^{-D/2} \exp[-\Delta x^2/(2T)], \quad (4)$$

where $\Delta x = x_{\text{new}} - x$ is the deviation of x_{new} from the current accepted point x . It has been proven [10] that it suffices to obtain a global minimum of f if T is selected to be not faster than

$$T(k) = \frac{T_0}{\ln k}. \quad (5)$$

One can prove that this cooling schedule satisfies Equation (3) in the D -dimensional neighborhood for an arbitrary size $|\Delta x|$ and k . In fact we have

$$\sum_{k_0}^{\infty} g_k \geq \sum_{k_0}^{\infty} \exp(-\ln k) = \sum_{k_0}^{\infty} y_k = \infty. \quad (6)$$

3.3 Fast Annealing (FA)

There are sound physical principles underlying the choices of Equations (4) and (5) [27,32]. It was noted that this method of finding the global minimum in x -space is not limited to physics examples requiring “temperatures” and “energies”. Rather this methodology can be readily extended to any problem for which a reasonable probability density can be formulated [42]. It was also noted this methodology can be readily extended to use any reasonable generation function g .

Szu and Hartley [42] introduced a fast annealing method which uses the following Cauchy distribution as the generation function:

$$g(x) = \frac{T}{(\Delta x^2 + T^2)^{(D+1)/2}}. \quad (7)$$

The Cauchy distribution has some definite advantages over the Boltzmann form [42]. It has a “fatter” tail than the Gaussian form of the Boltzmann distribution, permitting easier access far from a local minimum in the search for the desired global minimum.

On the other hand, we can set the annealing schedule as

$$T(k) = \frac{T_0}{k}. \quad (8)$$

Then one can prove that this cooling schedule satisfies Equation (3). In fact we have

$$\sum_{k_0}^{\infty} g_k \approx \frac{T_0}{\Delta x^{D+1}} \sum_{k_0}^{\infty} \frac{1}{k} = \infty. \quad (9)$$

The method of FA is thus statistically seen to have an annealing schedule exponentially faster than the method of BA. This method has been tested on a variety of problems [42].

4. Very Fast Simulated Reannealing (VFSR) and the ASA Code

In a variety of physical problems, we have a D -dimensional parameter-space. Different parameters have different finite ranges, fixed by physical considerations, and different annealing-time-dependent sensitivities. BA and FA have g distributions which sample infinite ranges, and there is no provision for considering differences in each parameter-dimension. For example, different sensitivities might require different annealing schedules.

One might choose a D -product of one-dimensional Cauchy distributions because the one-dimensional Cauchy distribution has a few quick algorithms. This would also permit different T_0 's to account for different sensitivities

$$g_{ik} = \frac{T_{i0}}{(\Delta x^i)^2 + T_{i0}^2}. \quad (10)$$

But then we would require an annealing schedule:

$$T_i(k) = \frac{T_{i0}}{k^{1/D}} \quad (11)$$

which, although faster than BA, is still quite slow.

Motivated by the above considerations, Ingber introduced a very fast simulated annealing method [16].

4.1 Very Fast Annealing

As we mentioned previously, different parameters may have different annealing-time-dependent sensitivities. We consider a parameter a_k^i in dimension i generated at annealing-time k with the range

$$a_k^i \in [A_i, B_i].$$

The parameter a_{k+1}^i can be calculated from the random variable y^i

$$a_{k+1}^i = a_k^i + y^i (B_i - A_i),$$

$$y^i \in [-1, 1].$$

Define the generating function

$$g_T = \prod_{i=1}^D \frac{1}{2(|y^i| + T_i) \ln(1 + 1/T_i)} \equiv \prod_{i=1}^D g_T^i(y^i), \quad (12)$$

where the subscript i on T_i specifies the parameter index, and the k -dependence in $T_i(k)$ for the annealing schedule has been dropped for brevity. Its cumulative probability distribution is

$$G_T(y) = \int_{-1}^{y^1} \cdots \int_{-1}^{y^D} dy^{1'} \cdots dy^{D'} g_T(y') \equiv \prod_{i=1}^D G_T^i(y^i), \quad (13)$$

$$G_T^i(y^i) = \frac{1}{2} + \frac{\text{sgn}(y^i)}{2} \frac{\ln(1 + |y^i|/T_i)}{\ln(1 + 1/T_i)}. \quad (14)$$

y^i is generated from a u^i from the uniform distribution

$$u^i \in U[0, 1], \quad (15)$$

$$y^i = \text{sng}(u^i - \frac{1}{2}) T_i [(1 + 1/T_i)^{|2u^i - 1|} - 1]. \quad (16)$$

By a straightforward calculation, one can set the annealing schedule for T_i as

$$T_i(k) = T_{i0} \exp(-c_i k^{1/D}). \quad (17)$$

A global minimum statistically can be obtained; that is, the above annealing schedule satisfies Equation (3):

$$\sum_{k_0}^{\infty} g_k \approx \sum_{k_0}^{\infty} \left[\prod_{i=1}^D \frac{1}{2|y^i|c_i} \right] \frac{1}{k} = \infty. \quad (18)$$

It seems sensible to choose control over c_i such that

$$T_{fi} = T_{oi} \exp(-m_i), \quad (19)$$

$$k_f = \exp n_i, \quad (20)$$

$$c_i = m_i \exp(-n_i/D), \quad (21)$$

where m_i and n_i can be considered “free” parameters to help tune ASA for specific problems. Here ASA refers to Ingber’s adaptive simulated annealing code, which we will explain briefly in Section 4.3.

It has proven fruitful to use the same type of annealing schedule for the acceptance function h as is used for the generating function g , i.e., Equations (17) and (19), but with the number of acceptance points, instead of the number of generated points, used to determine the k for the acceptance temperature.

In one implementation of this algorithm, new parameters a'_{k+1} are generated from old parameters a'_k by generating the y^j until a set of D is obtained satisfying the range constraints.

4.2 Reannealing

Whenever doing a multi-dimensional search in the course of solving a real-world nonlinear physical problem, inevitably one must deal with different changing sensitivities of the a^i in the search. At any given annealing-time, it seems sensible to attempt to “stretch out” the range over which the relatively insensitive parameters are being searched, relative to the ranges of the more sensitive parameters.

It has proven fruitful to accomplish this by periodically rescaling the annealing-time k , essentially reannealing, every hundred or so acceptance-events (or at some user-defined modules of the number of accepted or generated states), in terms of the sensitivities s_i calculated at the most current minimum value of the cost function, f ,

$$s_i = \mathcal{J} / \partial a^i.$$

In terms of the largest $s_i = s_{\max}$, a default rescaling is performed for each k_i of parameter dimension, whereby a new index k'_i is calculated from each k_i .

$$k_i \rightarrow k'_i,$$

$$T'_{ik'} = T_{ik} (s_{\max} / s_i),$$

$$k'_i = (\ln(T_{i0} / T_{ik'}) / c_i)^D.$$

T_{i0} is set to unity to begin the search, which is ample to span each parameter dimension.

Recall that we use the Boltzmann acceptance criterion as the acceptance criterion.

That is, if

$$\exp(-\Delta f / T_{\text{cost}}) > \nu,$$

the new point is accepted as the new saved point for the next iteration. Otherwise, the last saved point is retained. Here T_{cost} is the “temperature” used in this test, and v is from the uniform distribution

$$v \in U[0,1].$$

The annealing schedule for the cost temperature (or, acceptance temperature) is developed similarly to the parameter temperature. However, the Boltzmann acceptance criterion uses an exponential distribution which is not as fat-tailed as the distribution used for the parameters. The index for reannealing the cost function, k_{cost} , is determined by the number of accepted points, instead of the number of generated points as used for the parameters.

There is still an unanswered question: How to choose the initial acceptance temperature? Ingber said: “The initial acceptance temperature is set equal to an initial trial value of f ” [16,21]. Here and in the following, we understand that only the quantities of the temperatures and the function values are compared.

The initial trial value of f is “typically very large relative to the current best minimum, which may tend to distort the scale of the region currently being sampled”. “Therefore, when this rescaling is performed, the initial acceptance temperature is reset to the maximum of the most current minimum and the best current minimum of f , and the annealing-time index associated with this temperature is reset to give a new temperature equal to the minimum of the current cost-function and the absolute values of the current best and last minima” [16,21].

We discussed the above problem with Chandler [4] and he made the following comments: “However, the cost function f may have units, such as M/sec., that are inappropriate for temperature. Further, the scaling (units) of f is arbitrary. Last, all values of f could be negative. Therefore, these remarks of Ingber’s make no sense to me, although his prescription no doubt will work in most cases.”

Also generated are the “standard deviations” of the theoretical forms, calculated as $[\partial^2 f / (\partial a_i)^2]^{-1/2}$, for each parameter a_i . This gives an estimate of the “noise” that accompanies fits to stochastic data or functions. At the end of the run, the off-diagonal elements of the “covariance matrix” are calculated for all parameters. This inverse curvature of the theoretical cost function can provide a quantitative assessment of the relative sensitivity of parameters to statistical errors in fits to stochastic systems.

4.3 ASA Code

The adaptive simulated annealing (ASA) code was first developed by Lester Ingber in 1987 as Very Fast Simulated Reannealing (VFSR) to deal with the necessity of performing adaptive global optimization on multivariate nonlinear stochastic systems [16]. Since 1993, many features have been added, leading to the current ASA code [22]. “Adaptive” in Adaptive Simulated Annealing refers to adaptive options available to a user to tune the ASA algorithm to optimize the code for application to specific systems. While the default options may suffice for many applications, this is not intended to imply that the code will automatically adaptively seek the best tuning options. There are many user options in the ASA code. Among them, there are only a few options which are very

influential [22]. However, it seems that it is not easy to grasp all the user options. In our testing we only use the most influential option “Temperature_Ratio_Scale”. For all other options we simply use the default values. Of course, we may not obtain the best results for some problems.

5. Test Problems

In this chapter we list the functions which we use to compare SA algorithm, VFSR algorithm, the random pattern search algorithm, and the torus algorithm. These functions exhibit moderate ill-conditioning, nonsmoothness, and multi-modality in various forms. For the detailed description of these functions, the readers are referred to [3,12,28,35,36,38]. We specify the range for each variable and the starting point for each function to be minimized. We also state the actual minimum or approximate minimum of each function.

1. Rosenbrock function:

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

variable range: $-2000 \leq x_1, x_2 \leq 2000$ [39].

starting point: $x = (-1.2, 1)$.

The actual minimum of this function is 0 at (1,1).

2. Modified Rosenbrock's function 1 ("flat-ground bent knife-edge function"):

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

variable range: $-2000 \leq x_1, x_2 \leq 2000$.

starting point: $x = (-1.2, 1)$.

The actual minimum of this function is 0 at (1,1). This function is not smooth.

3. Modified Rosenbrock's function 2 ("hollow-ground bent knife-edge function"):

$$f(x) = 100|x_2 - x_1^2|^{\frac{1}{2}} + (1 - x_1)^2$$

variable range: $-2000 \leq x_1, x_2 \leq 2000$.

starting point: $x = (-1.2, 1)$.

The actual minimum of this function is 0 at (1,1). This function is not smooth.

4. Bohachevsky function:

$$f(x) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.3 + 0.4$$

variable range: $-2000 \leq x_1, x_2 \leq 2000$.

starting point: $x = (-1, 1)$.

The actual minimum of this function is 0 at (0, 0).

5. Powell function

$$f(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4.$$

variable range: $-2000 \leq x_1, x_2, x_3, x_4 \leq 2000$.

starting point: $x = (3, -1, 0, 1)$.

The actual minimum of this function is 0 at (0, 0, 0, 0). The Hessian matrix of this function is singular at the minimum.

6. Wood function

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 + 10.1[(x_2 - 1)^2 + (x_4 - 1)^2] + 19.8(x_2 - 1)(x_4 - 1).$$

variable range: $-2000 \leq x_1, x_2, x_3, x_4 \leq 2000$.

starting point: $x = (-3, -1, -3, -1)$.

The actual minimum of this function is 0 at (1, 1, 1, 1).

7. Beale Function

$$f(x) = (1.5 - x_1(1 - x_2))^2 + (2.25 - x_1(1 - x_2^2))^2 + (2.625 - x_1(1 - x_2^3))^2.$$

variable range: $-2000 \leq x_1, x_2 \leq 2000$.

starting point: $x = (1, 0.8)$.

The actual minimum of this function is 0 at (3, 0.5).

8. Engvall function

$$f(x) = x_1^4 + x_2^4 + 2x_1^2x_2^2 - 4x_1 + 3.$$

variable range: $-2000 \leq x_1, x_2 \leq 2000$.

starting point: $x = (0.5, 2.0)$.

The actual minimum of this function is 0 at (1, 0).

The following two functions are the Osborne functions [36]. Osborne [36] studied a general method for minimizing a sum of squares which has the property that a linear least squares problem is solved at each stage and which includes the Gauss-Newton, Levenberg, Marquardt, and Morrison methods as particular special cases.

The problem of minimizing a sum of squares arises naturally from the problem of determining parameters x_i , $i = 1, 2, \dots, p$ in the model equation

$$y(t) = F(t, x)$$

from observations

$$y_i = y(t_i) + \varepsilon_i, \quad (i = 1, 2, \dots, n),$$

where the ε_i (the experimental errors) are independent, normally distributed random variables with mean zero and standard deviation σ . In the case $n > p$ the appropriate maximum likelihood analysis indicates that x should be estimated by minimizing $f(x) = \|g(x)\|^2$, where

$$g_i(x) = y_i - F(t_i, x),$$

and

$$f(x) = \|g(x)\|^2 = \sum_{i=1}^n g_i(x)^2$$

This problem will be referred to as the *model problem*, and it is stressed that we have offered a statistical justification for minimizing a sum of squares. Osborne's two test problems are classic practical nonlinear least squares problems.

9. Osborne function 1

In this example, the data values $\{(t_i, y_i), (1 \leq i \leq 33)\}$, which are given in [36], are fitted by the model

$$F(t, x) = x_1 + x_2 \exp(-x_4 t) + x_3 \exp(-x_5 t).$$

Osborne's original method is a deterministic one, which does not have to specify the range of each variable. Based on the results in each stage of Osborne's problem, we may choose:

variable range: $0 \leq x_1 \leq 3, 0 \leq x_2 \leq 3, -3 \leq x_3 \leq 0, 0 \leq x_4 \leq 3, 0 \leq x_5 \leq 3$.

starting point: $x = (0.5, 1.5, -1, 0.01, 0.02)$.

The approximate minimum is 0.546E-4 at (0.3753, 1.9358, -1.4647, 0.01287, 0.02212). However, if we choose such a range and starting point, it is very difficult to obtain the global minimum by using a stochastic method. We cannot figure out what is the reason for this phenomenon.

In order to obtain a global minimum of Osborne function 1 by a stochastic method, we choose the following ranges and starting point to avoid the possible local minimum.

$$\text{variable range: } 0 \leq x_1 \leq 3, \quad -0.95 \leq x_2 \leq 1.95, \quad -3.45 \leq x_3 \leq -1.45, \quad 0 \leq x_4 \leq 3, \\ 0 \leq x_5 \leq 3.$$

starting point: (0.5, 1.5, -2, 0.01, 0.02).

10. Osborne function 2

In this example, the model has the form

$$F(t, x) = x_1 \exp(-x_5 t) + x_2 \exp[-x_6 (t - x_9)^2] \\ + x_3 \exp[-x_7 (t - x_{10})^2] + x_4 \exp[-x_8 (t - x_{11})^2]$$

The data values $\{(t_i, y_i), 1 \leq i \leq 65\}$ are also given in [36]. Osborne function 2 is easier than Osborne function 1 to minimize by stochastic methods. Based on the data in [36], we choose:

$$\text{variable range: } 0 \leq x_1 \leq 3, \quad 0 \leq x_2 \leq 3, \quad 0 \leq x_3 \leq 3, \quad 0 \leq x_4 \leq 3, \quad 0 \leq x_5 \leq 3, \\ 0 \leq x_6 \leq 3, \quad 0 \leq x_7 \leq 5, \quad 4 \leq x_8 \leq 7, \quad 0 \leq x_9 \leq 3, \quad 2 \leq x_{10} \leq 5, \quad 3 \leq x_{11} \leq 6.$$

starting point: (1.3, 0.65, 0.65, 0.7, 0.6, 3, 5, 7, 2, 4.5, 5.5).

The approximate global minimum is 0.0402 at (1.3100, 0.4315, 0.6336, 0.5993, 0.7539, 0.9056, 1.3651, 4.8248, 2.3988, 4.5689, 5.6754).

6. Test Results

In this chapter we run the random pattern search program, the torus program, the Corana's SA program, and the ASA program on the test functions we list in the previous chapter, and compare the results of these four programs.

6.1 Results of Random Pattern Search Algorithm

The random pattern search program in the M.S. report of Daniel Lee does not solve the modified Rosenbrock 1 and 2 problems, and it is clear that the algorithm used by Lee should not be able to handle nonsmooth problems. Perhaps Lee used some other version of his program to solve these two problems.

Lee's algorithm takes random steps only parallel to the coordinate axis. This is not consistent with the random pattern search algorithms of Lawrence and Steiglitz [28] and of Beltrami and Indusi [2], on which Lee's algorithm is supposed to be based.

Chandler [4] has programmed random pattern search [2,28]. Without trying many more search directions than prescribed, this algorithm should not be able to solve the two modified Rosenbrock problems, and it does not. The results of this algorithm on the other test problems are shown below.

Table 6-1 Results of Random Search Algorithm

Function	NF	$f(x)$
Rosenbrock	443	4.6398068E-9
Bohachevsky	237	2.287497
Powell	8093	1.5708663E-8
Wood	13587	2.5537496E-7
Beale	365	1.1568832E-15
Engvall	299	4.3298698E-14
Osborne1	23151	5.4730076E-5
Osborne2	30403	4.0137737E-2

Explanations:

$f(x)$: The minimum value we obtained.

NF: Number of function evaluation.

Random pattern search cannot be recommended in general for constrained or nonsmooth problems, which are the kinds of problems for which it was designed.

6.2 Results of Torus Algorithm

The torus algorithm can run on parallel processors. As the author pointed out, this approach is more of a Monte Carlo approach, and results are given by conducting function evaluations in a group to simulate parallel performance, rather than by actually running on a parallel computational system. For easily comparing with other algorithms, we only consider one processor.

We set all parameters to the default values except the parameters “scalar2” and “exit”. “Exit” is a stopping criterion with the default value 10^{-6} . For some problems we need to set “exit” to be smaller to get more accurate results. “Scalar2” is the weighting factor for the number of multiple-variable iterations per trial. The user controls the number of function calls of $f(x)$ made on each pass in multidimensional functions by adjusting the “scalar2” parameters. The “scalar2” parameter has the greatest impact on the outcome, and it should be increased for difficult problems.

There are four stopping criteria in the torus algorithm:

1. The number of trial blocks exceeds a pre-defined number “counter2”. The default value of “counter2” is 40.

2. The number of successive failures exceeds a pre-defined number “Flagz”. The default value of “Flagz” is 36.

3. The number of consecutive successes exceeds a pre-defined number “flag-count”. The default value of “flag-count” is 24.

4. The difference “last-minimum – best-minimum” is between zero and “exit” (success). Here, “last-minimum” refers to the smallest values returned by all prior calls of the single-dimension function, while “best-minimum” refers to the value returned by the current call of the single-dimension function.

The third criterion was not stated in the paper [39], but was actually coded in the author’s Lisp program [44].

We ran our FORTRAN torus program on our test functions and list the results in the following table:

Table 6-2 Results of Torus Program

Function	scalar2	exit	$f(x)$	NF	Stop Type
Rosenbrock	4	1.0E-6	1.62542E-8	11120	4
Rosenbrock1	5	1.0E-6	1.6606E-7	15220	4
Rosenbrock2	6.5	1.0E-6	8.71213E-3	27680	1
Bohachevsky	1	1.0E-6	3.67665E-7	3880	4
Powell	1	1.0E-6	4.51138E-7	9800	4
Wood	8	1.0E-6	3.19705E-7	73400	4
Beale	6	1.0E-6	7.30086E-8	11220	4
Engvall	8	1.0E-6	8.22563E-7	10720	4
Osborne1	14	1.0E-8	5.46544E-5	110450	4
Osborne2	6	1.0E-6	0.0401409	140030	4

Explanations:

Rosenbrock1 and 2 mean modified Rosenbrock function 1 and 2.

Stop type: Stopping type of the problem, as we explained before.

From the results in Table 6.2, we see that the torus problem converges for all test functions except the modified Rosenbrock function 2. The results are satisfactory. In particular, it converges for both Osborne functions.

Roughly speaking, for certain problem, the larger the value of “scalar2”, the larger the number of function evaluations, and the more accurate the results. However, if “scalar2” is set too large for a fixed problem, we may not get a more accurate answer, or may even not get the correct answer. In the following tables we list the results of the Osborne functions 1 and 2 for different values of “scalar2”.

Table 6-3 Results on Osborne Function 1

scalar2	$f(x)$	NF	stop type
13	5.67855E-5	73000	4
14	5.46544E-5	110450	4
15	5.46579E-5	110650	4
16	5.58468E-5	85400	4
20	5.48989E-5	152000	4

Table 6-4 Results on Osborne Function 2

scalar2	$f(x)$	NF	stop type
1	0.0413549	50490	4
2	0.107364	68310	3
3	0.0401507	63910	4
4	0.0401473	94820	4
5	0.0401482	154000	3
6	0.0401409	140030	4
7	0.0401395	231990	4
8	0.0401401	176550	4
9	0.0401428	187330	4
10	0.041452	183700	4
15	0.0401839	256190	4
20	0.0401398	389290	4

From the above tables we see that we get our best results for Osborne function 1 and Osborne function 2 when “scalar2” equals 14 or 7, respectively.

6.3 Results of SA

In [35], Ohm compared Bohachevsky’s simulated annealing algorithm and Corana’s simulated annealing algorithm on several test functions. The author concluded that the results of Corana’s program are more accurate than the results of Bohachevsky’s program.

In this section we only test Corana’s program on our test functions. The results are listed in following table. In the table, T_0 and V_0 are starting temperature and starting step vector, respectively.

As the author of [35] suggested, we choose best parameters T_0 and V_0 for each function. The first four functions were also tested in [35]. Since we use different ranges of the variables, our results are slightly different than the results in [35].

Table 6-5 Results of Corana’s Algorithm

Function	T_0	V_0	NF	$f(x)$
Rosenbrock	1000	0.01	220000	1.2776709E-8
Rosenbrock1	1000	0.01	216000	8.434226E-7
Rosenbrock2	1000	0.7	368000	5.6360820E-2
Bohachevsky	1000	0.7	180000	1.0778106E-8
Powell	1000	0.01	440000	5.2603830E-7
Wood	1000	0.01	500000	2.228336E-5
Beale	1000	0.01	124000	4.8416447E-9
Engvall	1000	0.01	152000	4.7172453E-8

The results are satisfactory except for the modified Rosenbrock function 2.

6.4 Results of ASA

The ASA code and its related documents are updated continually by the author, Lester Ingber. We use the most recent version, Version 15.10, which was released on June 20, 1997, to test our functions.

To use the ASA code, one has to set up the ASA interface. The program should be divided into two basic modules. (1) The user calling procedure, containing the cost function to be minimized, is contained in user.c, user.h and user_cst.h. (2) The ASA optimization procedure is contained in asa.c and asa.h. The file asa_user.h contains definitions and macros common to both asa.h and user.h. We simply defined our cost function in user_cst.h.

There are many user options in the ASA code, which allow the user to minimize very different functions. However, we cannot grasp all of the options. One of the very influential options is Temperature_Ratio_Scale, which determines the scale of parameter annealing. The default value of Temperature_Ratio_Scale is 10^{-5} . One may set a larger value than the default to slow down the annealing, or set a smaller value than the default to speed up the annealing.

In our test, we set different values of Temperature_Ratio_Scale for different functions to be minimized. For all other options, we simply use the default values.

For convenience, the author used a Makefile in the ASA code. In Makefile we set the following options:

```
DASA_TEST = FALSE
```

```
DOPTIONS_FILE = TRUE
```

```
DOPTIONS_FILE_DATA = TRUE
```

The first option tells the program to run our cost function, not the author's test function. The other two options tell the program to read the parameter values from the file `asa_opt`.

We list our ASA test results for some functions in the following table.

Table 6-6 Results of ASA Program

Function	TRS	$f(x)$	NF
Rosenbrock	0.2	1.136695E-9	59613
Rosenbrock	0.1	1.598384E-7	34217
Rosenbrock1	0.9	2.144271E-2	132875
Rosenbrock2	0.9	8.271849	125141
Beale	0.1	3.034363E-17	32202
Beale	0.08	6.620662E-18	26832
Beale	0.07	1.978987E-18	25760
Engvall	0.001	4.88498E-15	3704
Engvall	0.00001	5.52931E-11	1265
Osborne1	0.0001	5.4658E-5	86731
Osborne2	1.0E-10	0.04013813	312260

In the above table TRS means Temperature_Ratio_Scale.

For some functions the results are satisfactory. In particular, the ASA program also converges for both Osborne functions.

We also ran the program on the Osborne functions using different values of Temperature_Ratio_Scale, and list the results in the following tables:

Table 6-7 Results on Osborne Function 1

TRS	$f(x)$	NF
1.0E-2	5.474716E-5	268714
1.0E-3	5.464924E-5	147654
1.0E-4	5.4658E-5	86731
1.0E-5	5.465131E-5	59560
1.0E-6	5.468168E-5	33289
1.0E-7	8.742937E-5	12967

Table 6-8 Results on Osborne Function 2

TRS	$f(x)$	NF
1.0E-5	1.04042136	360069
1.0E-6	0.04017192	247986
1.0E-7	0.04020761	203693
1.0E-8	0.0402596	148501
1.0E-9	0.04017329	166660
1.0E-10	0.04013813	312260
1.0E-11	0.04017536	68672
1.0E-12	0.0405114	17414
1.0E-13	0.05284764	7848

Roughly speaking, the smaller the value of Temperature_Ratio_Scale, the smaller the number of function evaluations. However, if we set the value of Temperature_Ratio_Scale too small or too large for any particular problem, the results may not be correct.

From the above tables we see that the best TRS for Osborne function 1 is 10^{-3} , which is larger than the default value, while the best TRS for Osborne function 2 is 10^{-10} , which is much smaller than the default value.

6.5 Comparisons

In this section, we compare the results of the four programs.

For all the functions we tested, the simulated annealing program is much slower than the other three programs. Hence, in the following we mainly compare the results of the random pattern search program, the torus program, and the ASA program.

Since we do not know very well how to tune the ASA program, we cannot get satisfactory results for some functions, and we simply omit them in our comparisons.

Table 6-9 Results on Rosenbrock Function

Program	$f(x)$	NF
Random Pattern Search	4.6398068E-9	443

Torus	1.62542E-8	11120
Corana SA	1.2776709E-8	220000
ASA	1.136695E-9	59613

For the Rosenbrock function, the four programs have the similar accuracy. The random pattern search program is faster than the other three programs, while the Corana SA program is much slower than others. It seems that the torus program is faster than the ASA program on the Rosenbrock function. On one hand, the author of [39] tuned his program mainly based on the Rosenbrock function. On the other hand, we did not tune all the option parameters in the ASA code to get the best result for the Rosenbrock function. Actually, we do not know how to tune ASA optimally. These two reasons may explain why the torus program is superior to ASA on the Rosenbrock function.

Table 6-10 Results on Modified Rosenbrock Function 1

Program	$f(x)$	NF
Torus	1.6606E-7	15220
Corana SA	8.434226E-7	216000

For the modified Rosenbrock function 1, the torus program and the Corana SA program have the similar accuracy. The torus program is much faster than the Corana SA program.

All four programs do not solve the modified Rosenbrock function 2 very well.

Table 6-11 Results on Bohachevsky Function

Program	$f(x)$	NF
Random Pattern Search	2.287497	237
Torus	3.67665E-7	3880
Corana SA	1.0778106E-8	180000

The random pattern search program does not solve the Bohachevsky function , which has several local minimum. It may stops at local minimum. Both the torus program and Corana SA program solve the Bohachevsky function with the similar accuracy. But the tours program is much faster than the Corana program.

Table 6-12 Result on Powell Function

Program	$f(x)$	NF
Random Pattern Search	1.5708663E-8	8093
Torus	4.51138E-7	9800
Corana SA	5.2603830E-7	440000

For the Powell function, the random pattern search program, the torus program, and the Corana SA program have the similar accuracy. The first two programs are much faster than the Corana SA program.

Table 6-13 Results on Wood Function

Program	$f(x)$	NF
Random Pattern Search	2.5537496E-7	13587
Torus	3.19705E-7	73400
Corana SA	2.228336E-5	500000

For the Wood function, the random pattern search program and the torus program are more accurate and much faster than the Corana SA program. The random pattern search program is even faster than the torus program.

Table 6-14 Results on Beale Function

Program	$f(x)$	NF
Random Pattern Search	1.1568832E-15	365
Torus	7.30086E-8	11220
SA	4.8416447E-9	124000
ASA	1.978987E-18	25760

For the Beale function, the results of the random pattern search program and the ASA program are much more accurate than the results of the other programs. The random pattern search program is much faster than the other three programs. The torus program was tuned to produce median final function values in the range of the other algorithm, for example, Nelder-Mead simplex method and Corana SA algorithm [39]. The ASA program can produce final function values with arbitrary accuracy. If we adjust the related parameter in ASA, then ASA can produce a result similar to that of the torus program, but ASA is faster than the torus program on the Beale function.

Table 6-15 Results on Engvall Function

Program	$f(x)$	NF
Random Pattern Search	4.3298698E-14	299
Torus	8.22563E-7	10720
Corana SA	4.7172453E-8	152000
ASA	5.52931E-11	1265

For the Engvall function, the random pattern search program is more accurate and faster than the other three programs. The ASA is much more accurate and much faster than the torus program. This may be an example where ASA is more robust than the torus program.

Table 6-16 Results on Osborne Function 1

Program	$f(x)$	NF
Random Pattern Search	5.4730076E-5	23151
Torus	5.46544E-5	110450
ASA	5.4658E-5	86731

We are particularly interested in solving both Osborne functions. Except for the Corana SA program, all other three programs solve the Osborne function 1. The random

pattern search program is not accurate as the other two programs, but is faster than them. ASA and the torus program have similar accuracy, but ASA is faster than the torus program.

Table 6-17 Results on Osborne Function 2

Program	$f(x)$	NF
Random Pattern Search	0.040137737	30403
Torus	0.0401409	140030
ASA	0.04013813	312260

Except for the Corana SA program, all other three programs also solve the Osborne function 2 with similar accuracy. The random pattern search program is faster than the other two programs.

Finally, we mention again that, for the modified Rosenbrock function 2, all four programs do not find satisfactory results. Actually, the modified Rosenbrock function 2 is a very difficult function to minimize by a stochastic algorithm [35]. We do not know if we can solve this problem using ASA, even if we tune it accordingly. Fortunately, Rosenbrock 2 is not similar to functions that arise often in practical problems. Minimizing the L_p -norm of the residuals in a fitting problem, for $p=0.5$, would yield a problem similar to the modified Rosenbrock function 2. Values of p less than 1.0 seem never to have been used in the statistical literature, as far as we know.

7. Conclusions

To solve the global optimization problems, many stochastic optimization algorithms have been proposed in the past decades. In this thesis, we compare four such algorithms: the random pattern search algorithm, the torus algorithm, the Corana simulated annealing (SA) algorithm, and the ASA program. Brief explanations of these algorithms are given. Ten functions are chosen to test these algorithms. In particular, we test these algorithms on both Osborne functions.

All four programs fail to solve the modified Rosenbrock function 2, which is a very difficult function to minimize by any algorithm. In the remaining comments, we only consider the other nine functions.

The random pattern search program solved seven functions, but not the modified Rosenbrock function 1 and Bohachevsky function. The Corana simulated annealing program solved the seven functions, but solved neither Osborne function. Both the torus program and the ASA program solve nine functions.

For all functions it solved, the random pattern search program is faster or much faster than the other programs.

The torus program, the Corana simulated annealing program, and ASA program use the annealing principle and can jump out of a local minimum of a function. Theoretically, simulated annealing algorithm and the very fast annealing algorithm can find a global minimum of a function, and very fast simulated annealing is much faster than simulated annealing. The mathematical foundations of the torus algorithm were not given.

In our testing, the Corana simulated annealing program is much slower than the torus program and the ASA program, as one predicted . Since we do not know very well how to tune the ASA program, we failed to solve some of our test functions. For some functions, like the Rosenbrock function, the torus program is faster than the ASA program. For some other functions, like the Engvall function, the ASA program is much faster than the torus program. However, we believe that if one tuned the ASA program accordingly, the ASA program may be more robust and faster than the torus program.

In particular, the random pattern search program, the torus program, and the ASA program, solved both Osborne functions. The random pattern search program is much faster than the torus program and the ASA program on both Osborne functions. The torus program is faster than the ASA program on Osborne function 2, while the ASA program is faster than the torus program on Osborne function 1.

Suggestions for further study:

Design and/or find a stochastic optimization algorithm to solve the modified Rosenbrock function 2.

Give a mathematical foundation for the torus algorithm.

Find the reason we should specify the variable ranges when we use the torus program and the ASA program to test the Osborne function 1.

REFERENCES

- [1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, New York, 1989.
- [2] E. J. Beltrami and J. P. Indusi, *An adaptive random search algorithm for constrained minimization*, IEEE Transactions on Computers **21** (1972) 1004-1008.
- [3] I. O. Bohachevsky, M. E. Johnson, and M. L. Stein, *Generalized annealing for function optimization*, Technometrics **28** (1986) 209-217.
- [4] J. P. Chandler, private communication, 1997.
- [5] A. Corana, M. Marchesi, C. Martini, and S. Ridella, *Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithms*, ACM Transactions on Mathematical Software **13** (1987) 262-280.
- [6] W. C. Davidon, *Variable metric method for minimization*, AEC Research and Development Report, ANL-5990, Argonne National Laboratory, Lemont, Illinois, 1959.
- [7] M. H. A. Davis, *Markov Models and Optimization*, Chapman & Hall, London, 1993.
- [8] A. Dekkers and E. Arts, *Global optimization and simulated annealing*, Mathematical Programming **50** (1991) 367-393.
- [9] R. Fletcher and M. J. D. Powell, *A rapid descent method for minimization*, Computer Journal **6** (1963) 163-168.
- [10] W. L. Goffe, G. D. Ferrier, and John Rogers, *Global optimization of statistical functions with simulated annealing*, J. Econometrics **60** (1994) 65-99.
- [11] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images*, IEEE Transactions on Pattern Analysis and Machine Intelligence **6** (1984) 721-741.
- [12] M. W. Heuckroth, J. L. Gaddy, and L. D. Gains, *An examination of the adaptive random search technique*, AIChE Journal **22** (1976) 744-750.
- [13] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Springer-Verlag, New York, 1980.
- [14] R. Hooke and T. A. Jeeves, *Direct search solution of numerical and statistical problems*, Journal of Association for Computing Machinery **8** (1961) 212-229.

- [15] R. Horst and P. M. Pardalos, eds., *Handbook of Global Optimization*, Kluwer Academic Publications, Boston, 1995.
- [16] L. Ingber, *Very fast simulated re-annealing*, *Mathl. Comput. Modelling* **12** (1989) 967-973.
- [17] L. Ingber, *Statistical mechanical aids to calculating term structure models*, *Phys. Rev. A* **42** (1990) 7057-7064.
- [18] L. Ingber, *Statistical mechanics of neocortical interactions: A scaling paradigm applied to electroencephalography*, *Phys. Rev. A* **44** (1991) 4017-4060.
- [19] L. Ingber, *Generic Mesoscopic neural networks based on statistical mechanics of neocortical interactions*, *Phys. Rev. A* **45** (1992) 2183-2186.
- [20] L. Ingber, *Simulated annealing: Practice versus theory*, *Mathl. Comput. Modelling* **28** (1993) 29-57.
- [21] L. Ingber, *Adaptive simulated annealing (ASA): Lessons learned*, *Control and Cybernetics* **25** (1996) 33-54.
- [22] L. Ingber, *Adaptive Simulated Annealing (ASA)*, [ftp.alumni.caltech.edu:/pub/ingber/ASA-shar, ASA-shar.Z, ASA.tar.Z, ASA.tar.gz, ASA.zip], Mclean, VA, Lester Ingber Research.
- [23] L. Ingber, H. Fujio, and M. F. Wehner, *Mathematical comparison of combat computer models to exercise data*, *Mathl. Comput. Modelling* **15** (1991) 65-90.
- [24] L. Ingber, and B. Rosen, *Genetic algorithms and very fast simulated reannealing: A comparison*, *Mathl. Comput. Modelling* **16** (1992) 87-100.
- [25] L. Ingber and D. D. Sworder, *Statistical mechanics of combat with human factors*, *Mathl. Comput. Modelling* **15** (1991) 99-127.
- [26] L. Ingber, M. F. Wehner, G. M. Jabbour, and T. M. Barnhill, *Application of statistical mechanics methodology to term-structure bond-pricing models*, *Mathl. Comput. Modelling* **15** (1991) 77-98.
- [27] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, *Optimization by simulated annealing*, *Science* **220** (1983) 671-680.
- [28] J. P. Lawrence and K. Steiglitz, *Randomized pattern search*, *IEEE Transactions on Computers* **21** (1972) 382-385.

- [29] D. C. Lee, Computer Experimentation and Comparison of Random Search and Other Non-linear Optimization Methods, M. S. Report, Computer Science Department, Oklahoma State University, 1984.
- [30] P. J. M. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, Boston, 1988.
- [31] S. F. Masri and G. A. Bekey, *A global optimization algorithm using adaptive random search*, Applied Mathematics and Computation **7** (1980) 353-375.
- [32] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *Equation of state calculations by fast computing machines*, J. Chem. Phys. **21** (1953) 1087-1092.
- [33] M. Mintoff, *Algorithm 744: A stochastic algorithm for global optimization with constraints*, Computing Reviews, **37** (1996) 267-268.
- [34] J. A. Nelder and R. Mead, *A simplex method for function minimization*, Comput. J. **7** (1965) 308-313.
- [35] D. Y. Ohm, Generalized Simulated Annealing for Function Optimization over Continuous Variables, M. S. Thesis, Computer Science Department, Oklahoma State University, 1994.
- [36] M. R. Osborne, *Some aspects of non-linear least squares calculations*, Numerical Methods for Non-Linear Optimization, F. A. Lootsma ed., Academic Press, New York, 1971, 171-189.
- [37] R. H. J. M. Otten and L. P. P. van Ginneken, The Annealing Algorithm, Kluwer Academic Publishers, Boston, 1989.
- [38] J. D. Pintér, Global Optimization in Action, Kluwer Academic Publications, Boston, 1996.
- [39] F. M. Rabinowitz, *Algorithm 744: A stochastic algorithm for global optimization with constraints*, ACM Transactions on Mathematical Software **21** (1995) 194-213.
- [40] B. Rosen, *Function optimization based on advanced simulated annealing*, IEEE Workshop on Physics and Computation, IEEE Press, Dallas, 1992, 289-293.
- [41] L. E. Seales, Introduction To Non-Linear Optimization, Springer-Verlag, New York, 1985.
- [42] H. Szu and R. Hartley, *Fast simulated annealing*, Phys. Lett. A **122** (1987) 157-162.

[43] URL, http://pc_mally.eco.rug.nl:80/biblio/stoprog.html, *Stochastic programming bibliography*

[44] URL, <http://www.acm.org/contents/journals/toms>

APPENDIX: Program List for TORUS.F

```
C
C   ALGORITHM 744: A STOCHASTIC ALGORITHM FOR GLOBAL OPTIMIZATION
C   WITH CONSTRAINTS, BY MICHAEL RABINOWITZ. PUBLISHED IN ACM
C   TRANSACTIONS ON MATHEMATICAL SOFTWARE, VOL 21, NO. 2, JUNE
C   1995, PAGES 194-213.
C
C
C   PROGRAM MAIN
C
C   THIS MAIN PROGRAM IS A DRIVER OF THE TORUS ALGORITHM.
C
C   IMPLICIT REAL*8 (A-H,O-Z)
C
C   INTEGER  N, II, COUNT1, COUNT2, FUNCNU, MAXFUN, SDITER, MDITER,
*   SDCONT, MDCONT, LOGSNG, LOGMUL, SEED, LP
C   DOUBLE PRECISION  BUMP, EPS, HIT, TRIAL, TORUS, SCAL1,
*   SCAL2, DATAN,  START(20), UPPER(20), LOWER(20),
*   CUTOFF(20), PI, Y1, Y2, SI, TI, CR, DI2, DI4, CUTALT(20),
*   MINTRS(20), CUT(20), RANGE(20), NRANGE(20), BSTSET(20),
*   LSTSET(20), BMPSET(20), TM1SET(20), TMPSET(20), TMP2(20),
*   VAR2(20), VARSET(20), DELTAS(20), TEMP(20)
C
C   COMMON /A0/LP
C   COMMON /A1/COUNT1, COUNT2, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C   COMMON /A2/FUNCNU, MAXFUN, PI
C   COMMON /A3/SDITER, MDITER, SDCONT, MDCONT, LOGSNG, LOGMUL
C   COMMON /COR2/SI, TI, CR, DI2(2), DI4(4)
C   COMMON /OSB/Y1(33), Y2(65)
C
C   CALL TRSET TO SET PARAMETER VALUES IN TABLE 1.
C
C   CALL TRSET
C
C   CALL OSBSET TO SET THE PARAMETERS OF THE TWO OSBORNE FUNCTIONS.
C
C   CALL OSBSET(Y1, Y2)
C
C   MAXIMUM NUMBER OF FUNCTIONS TO BE MINIMIZED.
C
C   MAXFUN = 13
C
C   SET THE UNIT NUMBER FOR THE PRINTING.
C
C   LP = 6
C
C   PI=4.0D0*DATAN(1.0D0)
C
C   DO 10 II=1,13
C
C   SET THE VALUE OF THE SEED.
C
```



```

SEED=5671
C
FUNCNU = II
CALL INIT(N, START, LOWER, UPPER, CUTOFF)
CALL CONTRO (SEED, START, N, LOWER, UPPER, CUTOFF, CUTALT, MINTRS,
*          CUT, RANGE, NRANGE, BSTSET, LSTSET, BMPSET, TM1SET, TMPSET,
*          TMP2, VAR2, VARSET, DELTAS, TEMP)
C
10 CONTINUE
C
STOP
END
SUBROUTINE OSBSET(Y1, Y2)
C
THIS SUBROUTINE IS USED TO SET THE PARAMETERS OF THE TWO
C OSBORNE FUNCTIONS.
C
IMPLICIT REAL*8 (A-H, O-Z)
INTEGER J
DOUBLE PRECISION Y1(33), Y2(65), YY1(33), YY2(65)
C
DATA OF OSBORNE FUNCTION 1.
C
DATA YY1/0.844D0, 0.908D0, 0.932D0, 0.936D0, 0.925D0, 0.908D0,
* 0.881D0, 0.850D0, 0.818D0, 0.784D0, 0.751D0, 0.718D0, 0.685D0,
* 0.658D0, 0.628D0, 0.603D0, 0.580D0, 0.558D0, 0.538D0, 0.522D0,
* 0.506D0, 0.490D0, 0.478D0, 0.467D0, 0.457D0, 0.448D0, 0.438D0,
* 0.431D0, 0.424D0, 0.420D0, 0.414D0, 0.411D0, 0.406D0/
C
DATA OF OSBORNE FUNCTION 2.
C
DATA YY2/1.366D0, 1.191D0, 1.112D0, 1.013D0, 0.991D0, 0.885D0,
* 0.831D0, 0.847D0, 0.786D0, 0.725D0, 0.746D0, 0.679D0, 0.608D0,
* 0.655D0, 0.616D0, 0.606D0, 0.602D0, 0.626D0, 0.651D0, 0.724D0,
* 0.649D0, 0.649D0, 0.694D0, 0.644D0, 0.624D0, 0.661D0, 0.612D0,
* 0.558D0, 0.533D0, 0.495D0, 0.500D0, 0.423D0, 0.395D0, 0.375D0,
* 0.372D0, 0.391D0, 0.396D0, 0.405D0, 0.428D0, 0.429D0, 0.523D0,
* 0.562D0, 0.607D0, 0.653D0, 0.672D0, 0.708D0, 0.633D0, 0.668D0,
* 0.645D0, 0.632D0, 0.591D0, 0.559D0, 0.597D0, 0.625D0, 0.739D0,
* 0.710D0, 0.729D0, 0.720D0, 0.636D0, 0.581D0, 0.428D0, 0.292D0,
* 0.162D0, 0.098D0, 0.054D0/
C
DO 10 J=1, 33
Y1(J)=YY1(J)
10 CONTINUE
C
DO 20 J=1, 65
Y2(J)=YY2(J)
20 CONTINUE
RETURN
END
SUBROUTINE TRSET
C
THIS SUBROUTINE SET THE DEFAULT VALUES OF THE PARAMETERS IN
C TABLE 1 (PAGE 197) .
C
THIS PROGRAM CAN SIMULATE THE PARALLEL PROCESSORS.
C HOWEVER, WE RUN OUR PROGRAM ONLY FOR ONE PROCESSOR. HENCE, WE
C SET COUNT1=1 AND SCAL2=4.
C
THESE DEFAULT VALUES CAN BE ADJUSTED FOR EACH FUNCTION TO BE
C MINIMIZED IN THE SUBROUTINE INIT.
C
FOR MOST OF THE PROBLEMS, WE HAVE TO ADJUST SCAL2. FOR A FEW

```

```

C PROBLEMS WE NEED TO ADJUST EPS.
C
C   IMPLICIT REAL*8 (A-H,O-Z)
C   INTEGER COUNT1, COUNT2
C   DOUBLE PRECISION BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C
C   COMMON/A1/COUNT1, COUNT2, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C
C   BUMP = 5.0D-1
C   COUNT1 = 1
C   COUNT2 = 40
C   EPS = 1.0D-06
C   SCAL1 = 1.0D0
C   SCAL2 = 4.0D0
C   HIT = 1.5D0
C   TRIAL = 1.5D0
C   TORUS = 4.0D+3
C
C   RETURN
C   END
C   SUBROUTINE INIT (N,X,LOWER,UPPER,CUTOFF)
C
C   IN THIS SUBROUTINE WE SET THE DIMENSION, START POINT, LOWER
C   BOUND, UPPER BOUND, AND CUTOFF FOR EACH FUNCTION TO BE MINIMIZED.
C   WE ALSO CHANGE THE DEFAUTE VALUES IN SUBROUTINE TRSET IF NECESSARY.
C   HOWEVER, IF THE VALUE OF A PARAMETER IN TRSET IS CHANGED FOR ONE
C   FUNCTION, WE HAVE TO RESET THIS VALUE FOR THE NEXT FUNCTION.
C
C   IMPLICIT REAL*8 (A-H,O-Z)
C
C   INTEGER N, J, MAXFUN, FUNCNU, LP, COUNT1, COUNT2
C   DOUBLE PRECISION X(20), LOWER(20), UPPER(20), CUTOFF(20), PI,
C   *   SI, TI, CR, DI2, DI4, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C
C   COMMON /A0/LP
C   COMMON /A1/COUNT1, COUNT2, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C   COMMON /A2/FUNCNU, MAXFUN, PI
C   COMMON /COR2/ SI, TI, CR, DI2(2), DI4(4)
C
C   IF(FUNCNU .LT.1 .OR. FUNCNU .GT. MAXFUN) STOP
C   GO TO (10,40,70,100,130,160,190,220,250,280,310,340,370), FUNCNU
C
C   FUNCNU = 1
C   ROSENBROCK FUNCTION
C
C   10 N=2
C     DO 20 J=1, N
C       LOWER(J)=-2.0D3
C       UPPER(J)=2.0D3
C       CUTOFF(J)=1.0D-7
C   20 CONTINUE
C     X(1)=-1.2D0
C     X(2)=1.0D0
C     WRITE(LP,30)
C   30 FORMAT(/' ROSENBROCK TEST FUNCTION')
C     GO TO 400
C
C   FUNCNU = 2
C   MODIFIED ROSENBROCK FUNCTION WITH OBLIQUE CREASE
C
C   40 N = 2
C     DO 50 J=1, N
C       LOWER(J)=-2.0D3
C       UPPER(J)=2.0D3
C       CUTOFF(J)=1.0D-7

```

```

50 CONTINUE
   X(1)=-1.2D0
   X(2)=1.0D0
C
C SET SCAL2 IN TRSET.
C
   SCAL2 = 5.0D0
   WRITE (LP,60)
60 FORMAT(/' MODIFIED ROSENBROCK TEST FUNCTION WITH
* OBLIQUE CREASE')
   GO TO 400
C
C FUNCNU = 3
C MODIFIED ROSENBROCK WITH CUSP
C
70 N=2
   DO 80 J=1, N
     LOWER(J)=-2.0D3
     UPPER(J)=2.0D3
     CUTOFF(J)=1.0D-7
80 CONTINUE
   X(1)=-1.2D0
   X(2)=1.0D0
C
C SET SCAL2 IN TRSET.
C
   SCAL2 = 6.5D0
   WRITE (LP,90)
90 FORMAT(/' MODIFIED ROSENBROCK TEST FUNCTION WITH CUSP')
   GO TO 400
C
C FUNCNU = 4
C BOHACHEVSKY FUNCTION
C
100 N=2
   DO 110 J=1, N
     LOWER(J)=-2.0D3
     UPPER(J)=2.0D3
     CUTOFF(J)=1.0D-7
110 CONTINUE
   X(1)=1.0D0
   X(2)=1.0D0
C
C SET SCAL2 IN TRSET
C
   SCAL2 = 1.0D0
   WRITE (LP,120)
120 FORMAT(/' BOHACHEVSKY TEST FUNCTION')
   GO TO 400
C
C FUNCNU = 5
C OSBORNE 1 FUNCTION
C THE MINIMUM IS APPROXIMATELY F(0.3754,1.9358,-1.4647,0.01287,
C 0.02212)=0.546D-4
130 N=5
C
   LOWER(1)=0.0D0
   UPPER(1)=3.0D0
   LOWER(2)=-0.95D0
   UPPER(2)=1.95D0
   LOWER(3)=-3.45D0
   UPPER(3)=-1.45D0
   LOWER(4)=0.0D0
   UPPER(4)=3.0D0
   LOWER(5)=0.0D0

```

```

UPPER(5)=3.0D0
C
DO 140 J=1,N
  CUTOFF(J)=1.0D-7
140 CONTINUE
C
X(1)=0.5D0
X(2)=1.5D0
X(3)=-2.0D0
X(4)=1.0D-2
X(5)=2.0D-2
C
C SET SCAL2 AND EPS IN TRSET
C
SCAL2 = 14.0D0
EPS = 1.0D-08
WRITE(LP,150)
150 FORMAT(//1X,' TEST FUNCTION OF OSBORNE 1')
GO TO 400
C
C FUNCNU=6
C OSBORNE 2 FUNCTION
C
C THE MINIMUM IS APPROXIMATELY F(1.3100,0.4315,0.6336,
C 0.5993,0.7539,1.3651,4.8248,2.3988,4.5689,5.6754)=0.0402
C
160 N=11
C
LOWER(1)=0.0D0
UPPER(1)=3.0D0
LOWER(2)=0.0D0
UPPER(2)=3.0D0
LOWER(3)=0.0D0
UPPER(3)=3.0D0
LOWER(4)=0.0D0
UPPER(4)=3.0D0
LOWER(5)=0.0D0
UPPER(5)=3.0D0
LOWER(6)=0.0D0
UPPER(6)=3.0D0
LOWER(7)=0.0D0
UPPER(7)=5.0D0
LOWER(8)=4.0D0
UPPER(8)=7.0D0
LOWER(9)=0.0D0
UPPER(9)=3.0D0
LOWER(10)=2.0D0
UPPER(10)=5.0D0
LOWER(11)=3.0D0
UPPER(11)=6.0D0
C
DO 170 J=1, N
  CUTOFF(J)=1.0D-7
170 CONTINUE
C
X(1)=1.3D0
X(2)=6.5D-1
X(3)=6.5D-1
X(4)=7.0D-1
X(5)=6.0D-1
X(6)=3.0D0
X(7)=5.0D0
X(8)=7.0D0
X(9)=2.0D0
X(10)=4.5D0

```

```

      X(11)=5.5D0
C
C  SET SCAL2 AND EPS IN TRSET
C
      SCAL2 = 6.0D0
      EPS = 1.0D-06
      WRITE(LP,180)
180  FORMAT('/' TEST FUNCTION OF OSBORNE 2')
      GO TO 400
C
C  FUNCNU = 7
C  CORANA FUNCTION WITH DIMENSION N=2
C
190  N=2
      SI = 2.0D-1
      TI = 5.0D-2
      CR = 1.5D-1
      DI2(1) = 1.0D+0
      DI2(2) = 1.0D+3
      DO 200 J=1, N
          LOWER(J)=-1.0D4
          UPPER(J)=1.0D4
          CUTOFF(J)=1.0D-4
200  CONTINUE
      X(1)=1.1D+3
      X(2)=1.1D+3
      WRITE(LP,210)
C
C  SET SCAL2 AND EPS IN TRSET
C
      SCAL2 = 4.0D0
      EPS = 1.0D-06
210  FORMAT('//1X, ' CORANA FUNCTION, N=2')
      GO TO 400
C
C  FUNCNU = 8
C  CORANA FUNCTION WITH DIMENSION N=4
C
220  N=4
      SI = 2.0D-1
      TI = 5.0D-2
      CR = 1.5D-1
      DI4(1) = 1.0D+0
      DI4(2) = 1.0D+3
      DI4(3) = 1.0D+2
      DI4(4) = 1.0D+1
      DO 230 J=1, N
          LOWER(J)=-1.0D4
          UPPER(J)=1.0D4
          CUTOFF(J)=1.0D-4
230  CONTINUE
      X(1)=-1.0D+03
      X(2)=1.0D+03
      X(3)=-1.0D+3
      X(4)=1.0D+03
      WRITE(LP,240)
C
C  SET SCAL2 AND EPS IN TRSET
C
      SCAL2 = 6.5D0
      EPS = 1.0D-06
240  FORMAT('//1X, ' CORANA FUNCTION, N=4')
      GO TO 400
C
C  FUNCNU = 9

```

```

C POWELL'S SINGULAR TEST FUNCTION
C
C THE MINIMUM IS F(0.0,0.0,0.0,0.0)=0.0 .
C
250 N=4
    X(1)=3.0D0
    X(2)=-1.0D0
    X(3)=0.0D0
    X(4)=1.0D0
C
C SET SCAL2 AND EPS IN TRSET
C
    SCAL2 = 1.00D0
    EPS = 1.0D-06
    WRITE(LP,260)
260 FORMAT(' SINGULAR TEST FUNCTION OF POWELL')
    DO 270 J=1, N
        LOWER(J)=-2.0D3
        UPPER(J)=2.0D3
        CUTOFF(J)=1.0D-7
270 CONTINUE
    GO TO 400
C
C FUNCNU=10
C WOOD'S TEST FUNCTION
C
C THE MINIMUM IS F(1.0,1.0,1.0,1.0)=0.0 .
C
280 N=4
    X(1)=-3.0D0
    X(2)=-1.0D0
    X(3)=-3.0D0
    X(4)=-1.0D0
C
C SET SCAL2 AND EPS IN TRSET
C
    SCAL2 = 8.00D0
    EPS = 1.0D-06
C
    WRITE(LP,290)
290 FORMAT(' TEST FUNCTION OF WOOD')
    DO 300 J=1, N
        LOWER(J)=-2.0D3
        UPPER(J)=2.0D3
        CUTOFF(J)=1.0D-7
300 CONTINUE
    GO TO 400
C
C FUNCNU=11
C HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL
C THE MINIMUM IS F(1.0,0.0,0.0)=0.0 .
C
310 N=3
    X(1)=-1.0D0
    X(2)=0.0D0
    X(3)=0.0D0
C
C SET SCAL2 AND EPS IN TRSET
C
    SCAL2 = 1.00D0
    EPS = 1.0D-06
C
    WRITE(LP,320)
320 FORMAT(' HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL')
    DO 330 J=1, N

```

```

        LOWER(J)=-2.0D3
        UPPER(J)=2.0D3
        CUTOFF(J)=1.0D-7
330 CONTINUE
    GO TO 400
C
C   FUNCNU=12
C   BEALE'S TEST FUNCTION
C
C       THE MINIMUM IS F(3.0,0.5)=0.0 .
C
340 N=2
    X(1)=0.1D0
    X(2)=0.1D0
C
C   SET SCAL2 AND EPS IN TRSET
C
    SCAL2 = 6.00D0
    EPS = 1.0D-06
C
    WRITE(LP,350)
350 FORMAT(' BEALE TEST FUNCTION')
    DO 360 J=1, N
        LOWER(J)=-2.0D3
        UPPER(J)=2.0D3
        CUTOFF(J)=1.0D-7
360 CONTINUE
    GO TO 400
C
C   FUNCNU =13
C   ENGVALL TEST FUNCTION
C
370 N = 2
    X(1) = 5.0D-1
    X(2) = 2.0D0
C
C   SET SCAL2 AND EPS IN TRSET
C
    SCAL2 = 8.00D0
    EPS = 1.0D-06
C
    WRITE(LP,380)
380 FORMAT(' ENGVALL TEST FUNCTION')
    DO 390 J=1, N
        LOWER(J)=-2.0D3
        UPPER(J)=2.0D3
        CUTOFF(J)=1.0D-7
390 CONTINUE
    GO TO 400
C
C   PRINT FINAL RESULT
C
400 WRITE(LP,410)
410 FORMAT(/ 21X,'THE INITIAL VALUES ARE ',20X)
    WRITE(LP,420)
420 FORMAT(21X,'===== ',20X)
    WRITE(LP,430)(X(J),J=1,N)
430 FORMAT(' X =',1PG14.6,4G14.6/(4X,5G14.6))
C
    RETURN
    END
    DOUBLE PRECISION FUNCTION TEFNC(N,X)
C
C   FUNCTION TEFNC IS USED TO SET THE FUNCTIONS TO BE MINIMIZED
C

```

```

      IMPLICIT REAL*8 (A-H,O-Z)
C
      INTEGER N, FUNCNU, MAXFUN, KI(10), INDEX, I, J
      DOUBLE PRECISION X(N), PI, TEMP(4), Y1, Y2, T(65), DABS, DATAN, IDINT,
*      DCOS, DEXP, DSQRT, R, S, FTX, SI, TI, DI2, DI4, ZI(10), CR
C
      COMMON /A2/ FUNCNU, MAXFUN, PI
      COMMON /COR2/ SI, TI, CR, DI2(2), DI4(4)
      COMMON /OSB/Y1(33), Y2(65)
C
      IF (FUNCNU .LT. 1 .OR. FUNCNU .GT. MAXFUN) STOP
      GO TO (10, 20, 30, 40, 50, 80, 120, 190, 260, 270, 280, 290, 300), FUNCNU
C
C  FUNCNU = 1
C  ROSENBROCK'S TEST FUNCTION
C
      10  TESHNC=1.0D2*(X(2)-X(1)**2)**2+(1.0D0-X(1))**2
          RETURN
C
C  FUNCNU = 2
C  MODIFIED ROSENBROCK WITH OBLIQUE CREASE
C
      20  TESHNC=1.0D2*DABS(X(2)-X(1)**2)+(1.0D0-X(1))**2
          RETURN
C
C  FUNCNU = 3
C  MODIFIED ROSENBROCK WITH CUSP
C
      30  TESHNC=1.0D2*DSQRT(DABS(X(2)-X(1)**2))+(1.0D0-X(1))**2
          RETURN
C
C  FUNCNU = 4
C  BOHACHEVSKY FUNCTION
C
      40  TESHNC=X(1)**2+2*X(2)**2-3.0D-1*DCOS(3.0D0*PI*X(1))
*      -4.0D-1*DCOS(4.0D0*PI*X(2))+3.0D-1+4.0D-1
          RETURN
C
C  FUNCNU=5
C  OSBORNE FUNCTION 1
C
      50  TESHNC = 0.0D0
          DO 60 J=1, 33
              T(J)=10.0D0*(J-1)
      60  CONTINUE
          DO 70 J=1, 33
              R=DEXP((-1)*X(4)*T(J))
              S=DEXP((-1)*X(5)*T(J))
              FTX=X(1)+X(2)*R+X(3)*S-Y1(J)
              TESHNC = TESHNC+FTX**2
      70  CONTINUE
          RETURN
C
C  FUNCNU=6
C  OSBORNE FUNCTION 2
C
      80  TESHNC=0.0D0
          DO 90 J=1, 65
              T(J)=0.1D0*(J-1)
      90  CONTINUE
          DO 110 J=1, 65
              TEMP(1) = -X(5)*T(J)
              TEMP(2) = -X(6)*(T(J)-X(9))**2
              TEMP(3) = -X(7)*(T(J)-X(10))**2
              TEMP(4) = -X(8)*(T(J)-X(11))**2

```



```

      IF (TEMP(1) .LT. -69) TEMP(1) = -69
      IF (TEMP(2) .LT. -69) TEMP(2) = -69
      IF (TEMP(3) .LT. -69) TEMP(3) = -69
      IF (TEMP(4) .LT. -69) TEMP(4) = -69
      TEMP(1)=DEXP(TEMP(1))
      TEMP(2)=DEXP(TEMP(2))
      TEMP(3)=DEXP(TEMP(3))
      TEMP(4)=DEXP(TEMP(4))
      FTX=0.0D0
      DO 100 I=1, 4
        FTX=FTX+X(I)*TEMP(I)
100    CONTINUE
      FTX=FTX-Y2(J)
      TESHFNC = TESHFNC+FTX**2
110 CONTINUE
      RETURN
C
C  FUNCNU=7
C  CORANA FUNCTION, N=2
C
120 TESHFNC = 0.0D0
      INDEX = 2
      DO 130 I=1, 2
        IF(X(I) .GT. 0.0D0) THEN
          KI(I)=IDINT(X(I)/SI + 0.5D+0)
        ELSE IF(X(I) .LT. 0.0D0) THEN
          KI(I)=IDINT(X(I)/SI - 0.5D+0)
        ELSE
          KI(I)=0
        END IF
        IF(KI(I) .EQ. 0) INDEX= INDEX-1
130 CONTINUE
C
      IF (INDEX .EQ. 0) THEN
        DO 140 I = 1,2
          TESHFNC = TESHFNC+DI2(I)*X(I)**2
140    CONTINUE
        GO TO 180
      END IF
C
      INDEX = 2
      DO 150 I=1, 2
        IF(DABS(KI(I)*SI - X(I)) .LT. TI) INDEX=INDEX - 1
150 CONTINUE
      IF (INDEX .EQ. 0) THEN
        DO 160 I = 1,2
          IF(KI(I) .LT. 0) THEN
            ZI(I) = KI(I)*SI + TI
          ELSE IF(KI(I) .GT. 0) THEN
            ZI(I) = KI(I)*SI -TI
          ELSE
            ZI(I) = 0.0D0
          END IF
          TESHFNC = TESHFNC+CR*DI2(I)*ZI(I)**2
160    CONTINUE
        ELSE
          DO 170 I = 1,2
            TESHFNC = TESHFNC+DI2(I)*X(I)**2
170    CONTINUE
          END IF
180 RETURN
C
C  FUNCNU=8
C  CORANA FUNCTION, N=4
C

```

```

190 TEFUNC = 0.0D0
    INDEX = 4
    DO 200 I=1, 4
        IF(X(I) .GT. 0.0D0) THEN
            KI(I)=IDINT(X(I)/SI + 0.5D+0)
        ELSE IF(X(I) .LT. 0.0D0) THEN
            KI(I)=IDINT(X(I)/SI - 0.5D+0)
        ELSE
            KI(I)=0
        END IF
        IF(KI(I) .EQ. 0) INDEX= INDEX-1
200 CONTINUE
C
    IF (INDEX .EQ. 0) THEN
        DO 210 I = 1,4
            TEFUNC = TEFUNC+DI4(I)*X(I)**2
210 CONTINUE
        GO TO 250
    END IF
C
    INDEX = 4
    DO 220 I=1, 4
        IF(DABS(KI(I)*SI - X(I)) .LT. TI) INDEX=INDEX - 1
220 CONTINUE
    IF (INDEX .EQ. 0) THEN
        DO 230 I = 1,4
            IF(KI(I) .LT. 0) THEN
                ZI(I) = KI(I)*SI + TI
            ELSE IF(KI(I) .GT. 0) THEN
                ZI(I) = KI(I)*SI -TI
            ELSE
                ZI(I) = 0.0D0
            END IF
            TEFUNC = TEFUNC+CR*DI4(I)*ZI(I)**2
230 CONTINUE
        ELSE
            DO 240 I = 1,4
                TEFUNC = TEFUNC+DI4(I)*X(I)**2
240 CONTINUE
        END IF
250 RETURN
C
C FUNCNU=9
C POWELL'S SINGULAR TEST FUNCTION
C
260 TEFUNC=(X(1)+10.0D0*X(2))**2+5.0D0*(X(3)-X(4))**2+
    * (X(2)-2.0D0*X(3))**4+10.0D0*(X(1)-X(4))**4
    RETURN
C
C FUNCNU=10
C WOOD'S TEST FUNCTION
C
270 TEFUNC=100.0D0*(X(2)-X(1)**2)**2+(1.0D0-X(1))**2+
    * 90.0D0*(X(4)-X(3)**2)**2+(1.0D0-X(3))**2+
    * 10.1D0*((X(2)-1.0D0)**2+(X(4)-1.0D0)**2)+
    * 19.8D0*(X(2)-1.0D0)*(X(4)-1.0D0)
    RETURN
C
C FUNCNU=11
C HELICAL VALLEY TEST FUNCTION OF FLETCHER AND POWELL
C
280 R=DSQRT(X(1)**2+X(2)**2)
C
    IF(X(1).EQ.0.0D0) THEN
        S=0.25D0

```

```

ELSE
  S=DATAN(X(2)/X(1))/(2.0D0*PI)
ENDIF
C
IF(X(1).LT.0.0D0) S=S+0.5D0
TESFNC=100.0D0*((X(3)-10.0D0*S)**2+(R-1.0D0)**2)+X(3)**2
RETURN
C
C  FUNCNU=12
C  BEALE'S TEST FUNCTION
C
290  TESFNC = (1.5D0-X(1)*(1.0D0-X(2)))**2+
*   (2.25D0-X(1)*(1.0D0-X(2)**2))**2+
*   (2.625D0-X(1)*(1.0D0-X(2)**3))**2
RETURN
C
C  FUNCNU = 13
C  ENGVALL FUNCTION
C
300  TESFNC = (X(1)**2+X(2)**2)**2-4.0D0*X(1)+3.0D0
RETURN
C
END
DOUBLE PRECISION  FUNCTION RAND(SEED)
C
C  THIS FUNCTION IS USED TO GENERATE A RANDOM NUMBER BETWEEN -1 AND 1.
C
  IMPLICIT REAL*8(A-H,O-Z)
C
  INTEGER SEED
  SEED = 2045*SEED + 1
  SEED = SEED - (SEED/1048576)*1048576
  RAND = 2*(SEED+1)/1048577.0 -1.0
  RETURN
  END
  SUBROUTINE CONTRO(SEED,START,N,LOWER,UPPER,CUTOFF,CUTALT,
*  MINTRS,CUT,RANGE,NRANGE,BSTSET,LSTSET,BMPSET,TM1SET,TMPSET,
*  TMP2,VAR2,VARSET,DELTAS,TEMP)
C
C  THIS IS THE CONTROLLING FUNCTION (PAGE 199).
C
C  ALGORITHM 744: A STOCHASTIC ALGORITHM FOR GLOBAL OPTIMIZATION
C  WITH CONSTRAINTS, BY MICHAEL RABINOWITZ. PUBLISHED IN ACM
C  TRANSACTIONS ON MATHEMATICAL SOFTWARE, VOL 21, NO. 2, JUNE
C  1995, PAGES 194-213.
C
C  THE ORIGINAL PROGRAM WAS WRITTEN IN COMMON LISP.
C  WE TRANSLATE IT INTO A FORTRAN PROGRAM.
C
C  SUBROUTINES AND FUNCTIONS:
C  OSBSET() ---- SET THE PARAMETERS OF THE TWO OSBORNE FUNCTIONS.
C  TRSET() ---- SET THE PARAMETERS.
C  INIT() ---- SET THE DIMENSION, LOWER, UPPER, CUTOFF, START
C  POINT FOR EACH FUNCTION TO BE MINIMIZED.
C  TEFNC() ---- DEFINE THE FUNCTIONS TO BE MINIMIZED.
C  RAND() ---- GENERATE A RANDOM NUMBER BETWEEN -1 AND 1.
C  MULTDM() ---- THE DRIVER OF THE MULTIDIMENSION FUNCTION.
C  MULT1() ---- MULTIDIMENSION FUNCTION.
C  SINGER() ---- SINGER-DIMENSION FUNCTION.
C  SNG1() ---- THE INNER LOOP OF THE SINGER-DIMENSION FUNCTION.
C
C  VARIABLES:
C  THE FOLLOWING VARIABLES ARE GIVEN IN TABLE 1 (PAGE 197):
C
C  BUMP ---- *BUMP*. USED TO DISPLACE A VARIABLE FROM THE BEST-

```



```

*   BSTSET (N) , LSTSET (N) , BMPSET (N) , TMP2 (N) ,
*   LSTMIN , DFMIN , DATAN , TMLSET (N) , TMPSET (N) ,
*   VAR2 (N) , VARSET (N) , DELTAS (N) , TEMP (N) ,
*   DLOG , DMAX1 , DTEMP
LOGICAL  DOWNUP
C
COMMON  /A0/LP
COMMON  /A1/COUNT1 , COUNT2 , BUMP , EPS , HIT , TRIAL , TORUS , SCAL1 , SCAL2
COMMON  /A3/SDITER , MDITER , SDCONT , MDCONT , LOGSNG , LOGMUL
C
DO 10 I=1 , N
    MINTRS (I) = 64*CUTOFF (I)
    RANGE (I) = UPPER (I) - LOWER (I)
    CUTALT (I) = DMAX1 (RANGE (I) /TORUS , CUTOFF (I))
10 CONTINUE
C
C   ROUND TO NEAREST INTEGER.
C
    SDITER = 10.0D0*SCAL1 +0.5D0
    MDITER = 10.0D0*SCAL2*N**2 + 0.5D0
C
C   NUMBER OF FUNCTION EVALUATIONS IN EACH CALL OF
C   SINGER-DIMENSION FUNCTION.
C
    SDCONT = N*COUNT1*SDITER
C
C   NUMBER OF FUNCTION EVALUATIONS IN EACH CALL OF
C   MULTIDIMENSION FUNCTION.
C
    MDCONT = COUNT1*MDITER
C
C   SET LOG-CONSTANTS :
C   SDITER = LN (SINGLE-DIMENSION-ITERATIONS)  (TABLE 5, PAGE 204)
C   MDITER = LN (MULTIPLE-DIMENSION-ITERATIONS)  (TABLE 4, PAGE 203)
C
    DTEMP=SDITER
    LOGSNG=DLOG (DTEMP)
    DTEMP=MDITER
    LOGMUL=DLOG (DTEMP)
C
C   PASS START-VALUE TO THE MULTIDIMENSION FUNCTION, AND GET
C   THE FIRST LAST-MINIMUM AND THE FIRST BEST-SET.
C
    CALL MULTDM (N , START , BSTMIN , BSTSET , LOWER , UPPER ,
    *   RANGE , CUTALT , SEED , TMPSET , VARSET , DELTAS , TEMP)
    LSTMIN = BSTMIN
C
C   SET THE STARTING VALUES OF EACH VARIABLE IN THE LOOP.
C
    INTRAL = 1
    INTRCT = 0
    FLGZCT = 0
    COUNT = 0
C
C   IN LISP, THE INDEX OF THE FIRST ELEMENT OF AN ARRAY IS 0.
C   IN FORTRAN, THE INDEX OF THE FIRST ELEMENT OF AN ARRAY IS 1.
C   SO, THE STARTING VALUE OF NEXTVA IS 1, NOT 0.
C
    NEXTVA = 1
    FLGDIR = 0
C
DO 20 I=1 , N
    BMPSET (I) = BSTSET (I)
    LSTSET (I) = BMPSET (I)
    CUT (I) = CUTALT (I)

```

```

        NRANGE(I) = RANGE(I)
20 CONTINUE
C
    DOWNUP = .FALSE.
C
    CALL SINGER(N,BMPSET,BSTMIN,LSTSET, LOWER,UPPER,NRANGE,
*    CUT,SEED,NEXTVA,FLGDIR,TMP2,VAR2)
C
    IF(BSTMIN .LT. LSTMIN) THEN
        FLAGZ = 0
    ELSE
        FLAGZ = 1
    END IF
C
    TOTAL = MDCONT + SDCONT
C
C    LOOP
C
30 CONTINUE
C
STEP 1. SET THE WITH-TRIAL PARAMETER (STARTING VALUE=1).
C
    IF(FLAGZ .EQ. 0 .AND. INTRAL .GT. 1) THEN
        INTRAL = INTRAL
    ELSE IF(INTRAL .EQ. 3 ) THEN
        INTRAL = 1
    ELSE
        INTRAL = INTRAL + 1
    END IF
C
STEP 2. SET THE WITHIN-TRIAL-COUNT PARAMETER (STARTING VALUE = 0).
C
    IF(INTRAL .GT. 1) THEN
        INTRCT = INTRCT + 1
    ELSE
        INTRCT = 0
    END IF
C
STEP 2+ IN THE PSEUDOCODE [PAGE 199], THIS STEP WAS MISSING.
C    SET THE FLGZCT PARAMETER (STARTING VALUE = 0).
C
    IF(FLAGZ .EQ. 0) THEN
        FLGZCT = FLGZCT + 1
    ELSE
        FLGZCT = 0
    END IF
C
STEP 3. SET THE COUNT PARAMETER (STARTING VALUE = 0).
C
    IF(INTRAL .EQ. 1) COUNT = COUNT + 1
C
STEP 4. SET THE NEXT-VARIABLE PARAMETER (STARTING VALUE = 1).
C
    NEXTVA = MOD(COUNT, N) + 1
C
STEP 5. SET THE FLAG-DIRECTION PARAMETER (STARTING VALUE = 0).
C
    IF(FLGDIR .EQ. 0) THEN
        FLGDIR = 1
    ELSE
        FLGDIR = 0
    END IF
C
STEP 6 AND STEP 7.
C    SET THE CUT PARAMETER (STARTING VALUE = CUTOFF).

```

```

C          SET THE NEW-RANGE PARAMETER (STARTING VALUE = RANGE)
C
DO 40 I=1, N
  IF (INTRCT .GT. 1 .AND. FLAGZ .EQ. 0) THEN
    CUT(I) = DMAX1 (CUT(I)/HIT, CUTOFF(I))
    NRANGE(I) = DMAX1 (NRANGE(I)/HIT, MINTRS(I))
  ELSE IF (INTRAL .EQ. 1) THEN
    CUT(I) = DMAX1 (CUT(I)/TRIAL, CUTOFF(I))
    NRANGE(I) = DMAX1 (NRANGE(I)/TRIAL, MINTRS(I))
  END IF
40 CONTINUE
C
C STEP 8. SET THE DOWN-UP PARAMETER (STARTING VALUE = 0).
C
  IF (FLAGZ .EQ. 0 .AND. INTRCT .GT. 1) THEN
    IF (BSTSET (NEXTVA) .LT. LSTSET (NEXTVA)) THEN
      DOWNUP = .FALSE.
    ELSE
      DOWNUP = .TRUE.
    END IF
  ELSE IF (INTRAL .EQ. 2) THEN
    DOWNUP = .TRUE.
  ELSE IF (DOWNUP) THEN
    DOWNUP = .FALSE.
  ELSE
    DOWNUP = .TRUE.
  END IF
C
C STEP 9. SET THE BUMP PARAMETER (STARTING VALUE = 0).
C
  IF (DOWNUP) THEN
    BUMPV = NRANGE (NEXTVA) * BUMP
  ELSE
    BUMPV = -NRANGE (NEXTVA) * BUMP
  END IF
C
C IN LISP PROGRAM, STEP 10 IS AFTER STEP 11 AND STEP 12.
C IT SEEMS IT WORKS BETTER.
C
C STEP 11 AND STEP 12.
C   SET THE LAST-MINIMUM PARAMETER (STARTING VALUE = FIRST
C                                     LAST-MIN ).
C   SET THE BEST-SET PARAMETER (STARTING VALUE= FIRST
C                                     BEST-SET).
C
  IF (FLAGZ .EQ. 0) THEN
    LSTMIN = BSTMIN
    DO 50 I=1, N
      BSTSET (I) = LSTSET (I)
50 CONTINUE
  END IF
C
C STEP 10. SET THE BUMP-SET PARAMETER (STARTING VALUE =
C                                     FIRST BEST-SET)
C
DO 60 I=1, N
  BMPSET (I) = BSTSET (I)
60 CONTINUE
  IF (INTRAL .NE. 1) THEN
    IF (BSTSET (NEXTVA) + BUMPV .LT. UPPER (NEXTVA) .AND.
*   BSTSET (NEXTVA) + BUMPV .GT. LOWER (NEXTVA)) THEN
      BMPSET (NEXTVA) = BSTSET (NEXTVA) + BUMPV
    ELSE IF (BSTSET (NEXTVA) - BUMPV .LT. UPPER (NEXTVA) .AND.
*   BSTSET (NEXTVA) - BUMPV .GT. LOWER (NEXTVA)) THEN
      BMPSET (NEXTVA) = BSTSET (NEXTVA) - BUMPV
  
```

```

        END IF
        END IF
        TOTAL = TOTAL + SDCONT
C
C STEP 13. SET THE LAST-SCORE AND LAST-SET PARAMETERS (STARTING
C VALUES RETURNED BY A CALL TO THE SINGLE -DIMENSION FUNCTION).
C
        IF (INTRAL .EQ. 1) THEN
            CALL SINGER (N, BMPSET, BSTMIN, LSTSET, LOWER, UPPER, NRANGE,
*           CUT, SEED, NEXTVA, FLGDIR, TMP2, VAR2)
        ELSE
            CALL MULTDM (N, BMPSET, BSTMIN, TM1SET, LOWER, UPPER,
*           NRANGE, CUT, SEED, TMPSET, VARSET, DELTAS, TEMP)
            CALL SINGER (N, TM1SET, BSTMIN, LSTSET, LOWER, UPPER, NRANGE,
*           CUT, SEED, NEXTVA, FLGDIR, TMP2, VAR2)
            TOTAL = TOTAL + MDCONT
        END IF
C
C STEP 14. SET THE FLAGZ PARAMETER (STARTING VALUE = 0 IF NEW
C LAST-MIN OBTAINED BY THE FIRST CALL TO THE
C SINGLE-DIMENSION FUNCTION OTHERWISE, STARTING VALUE = 1).
C
        IF (BSTMIN .LT. LSTMIN) THEN
            FLAGZ = 0
        ELSE
            FLAGZ = FLAGZ + 1
        END IF
C
C STEP 15. EXIT TEST: RETURN TO THE BEGINING OF THE LOOP UNLESS ONE
C OF FOLLOWING CRITERIA IS MET:
C 1. COUNT=COUNT2 (TOO MANY COMPLETE TRIALS).
C 2. FLAGZ=36 (TOO MANY SUCCESSIVE FAILURES).
C 3. FLGZCT=24 (TOO MANY CONSECUTIVE SUCCESSES).
C THIS THIRD CRITERION WAS NOT STATED IN THE PAPER.
C 4. 0<LSTMIN - BSTMIN<EPS (SUCCESS).
C
        STPVAR = 0
        IF (COUNT .EQ. COUNT2) STPVAR = 1
        IF (FLAGZ .EQ. 36) STPVAR = 2
        IF (FLGZCT .EQ. 24) STPVAR=3
        DFMIN = LSTMIN-BSTMIN
        IF (DFMIN .GT. 0 .AND. DFMIN .LT. EPS) STPVAR = 4
        IF (STPVAR .GT. 0) GO TO 70
C
        GO TO 30
C
        70 WRITE (LP, 80)
        80 FORMAT (/// 21X, 'THE FINAL RESULT IS ', 20X)
           WRITE (LP, 90)
        90 FORMAT (21X, '=====', 20X)
           WRITE (LP, 100) BSTMIN, DFMIN
        100 FORMAT (' FUNCTION VALUE = ', 1G14.6, 2X, 'DFMIN=', 1G14.6)
           WRITE (LP, 110) TOTAL, STPVAR, COUNT, FLAGZ, FLGZCT
        110 FORMAT (' NF=', I6, 3X, 'STPVAR=', I2, 3X, 'COUNT=', I2, 3X,
*           'FLAGZ=', I2, 3X, 'FLGZCT=', I2)
           WRITE (LP, 120) (LSTSET (I), I=1, N)
        120 FORMAT (' X =', 1PG14.6, 4G14.6 / (4X, 5G14.6))
C
        RETURN
        END
        SUBROUTINE MULTDM (N, BMPSET, BSTVAL, BSTSET, LOWER, UPPER,
*           NRANGE, CUTALT, SEED, TMPSET, VARSET, DELTAS, TEMP)
C
C THIS SUBROUTINE IS A DRIVER OF THE FOLLOWING MULTIDIMENSION
C FUNCTION. COUNT1 IS THE NUMBER OF PARALLEL PROCESSORS SIMULATED.

```



```

C
  IMPLICIT REAL*8 (A-H,O-Z)
  INTEGER N, I, J, SEED, COUNT1, COUNT2
  DOUBLE PRECISION BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2,
*   BMPSET (N), BSTVAL, BSTSET (N), NRANGE (N), UPPER (N),
*   LOWER (N), CUTALT (N), TMPBST, TMPSET (N),
*   VARSET (N), DELTAS (N), TEMP (N)
  COMMON /A1/COUNT1, COUNT2, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C
  DO 30 I=1, COUNT1
    CALL MULT1 (N, BMPSET, TMPBST, TMPSET, LOWER, UPPER, NRANGE,
*   CUTALT, SEED, VARSET, DELTAS, TEMP)
    IF (I .EQ. 1) THEN
      BSTVAL = TMPBST
      DO 10 J=1, N
        BSTSET (J) = TMPSET (J)
10     CONTINUE
      ELSE IF (TMPBST .LT. BSTVAL) THEN
        BSTVAL = TMPBST
        DO 20 J=1, N
          BSTSET (J) = TMPSET (J)
20     CONTINUE
      END IF
30  CONTINUE
  RETURN
  END
C
  SUBROUTINE MULT1 (N, BMPSET, BSTVAL, BSTSET, LOWER, UPPER,
*   NRANGE, CUTALT, SEED, VARSET, DELTAS, TEMP)
  IMPLICIT REAL*8 (A-H, O-Z)
C
  THIS IS A MONTE CARLO ALGORITHM FOR CHANGING THE VALUES OF
C   ALL THE VARIABLES. ANNEALING PRINCIPLES ARE USED IN
C   CALCULATING THE MAXIMUM RANGE OF THE VARIABLES ON EACH
C   ITERATION. FAST COOLING IS IMPLEMENTED AS THE RANGE IS
C   LOGARITHMICALLY REDUCED. RANDOMNESS IS INTRODUCED INTO THE
C   FIT BY MULTIPLYING THE MAXIMUM RANGE OF THE FIT VARIABLE BY
C   A RANDOM NUMBER BETWEEN -1 AND 1 TO COMPUTE THE DELTA VALUE
C   FOR EACH ITERATION. THE BEST FITTING SCORE AND RELATED
C   VARIABLE SET ARE RETURNED BY THE FUNCTION.
C
  INTEGER I, N, ITER, FUNCNU, MAXFUN, SDITER, MDITER,
*   SDCONT, MDCONT, LOGSNG, LOGMUL, SEED
  DOUBLE PRECISION VALUE, TEFUNC, RAND, BSTVAL, BMPSET (N),
*   BSTSET (N), NRANGE (N), UPPER (N), LOWER (N),
*   CUTALT (N), VARSET (N), ITERDT,
*   DELTAS (N), TEMP (N), DTEMP
  LOGICAL VARFLG, VALFLG
C
  COMMON /A3/SDITER, MDITER, SDCONT, MDCONT, LOGSNG, LOGMUL
C
  ITER=0
  VARFLG = .FALSE.
  VALFLG = .FALSE.
C
  DO 10 I=1, N
    BSTSET (I) = BMPSET (I)
    VARSET (I) = BSTSET (I)
10  CONTINUE
C
  LOOP. IN THE FIRST ITERATION (ITER=0), WE SET THE STARTING VALUES
C   OF EACH VARIABLES. SOME VARIABLES, SUCH THAT ITERATE-DELTA,
C   DELTA, ARE NOT NECESSARILY TO BE INITIALIZED.
C
  20 CONTINUE

```

```

C
C STEP 1: SET THE ITERATE PARAMETER TO COUNT THE NUMBER OF ITERATION
C
      IF(.NOT. VARFLG) ITER=ITER+1
      IF(ITER .EQ. 1) GO TO 50
C
C STEP 2: SET THE ITERATE-DELTA PARAMETER TO WEIGHT THE VALUES
C           OF THE VARIABLES.
C
      DTEMP=ITER
      ITERDT=1.0D0-DLOG(DTEMP)/LOGMUL
C
C STEP 3: SET THE DELTA PARAMETER TO CALCULATE A SET OF BOUNDED
C           STOCHASTIC VALUES BY WHICH THE VALUES IN VARIABLE-SET
C           CAN BE MODIFIED.
C           THERE IS A SLIGHT DIFFERENCE BETWEEN PSEUDOCODE AND
C           LISP PROGRAM.
C
      DO 30 I=1, N
        DELTAS(I) = ITERDT*NRANGE(I)*RAND(SEED)
        IF(DABS(DELTAS(I)) .LT. CUTALT(I)) THEN
          DELTAS(I) = 4 * CUTALT(I) * RAND(SEED)
          IF(DABS(DELTAS(I)) .LT. CUTALT(I)) THEN
            IF(DELTAS(I) .LT. 0 ) THEN
              DELTAS(I) = -CUTALT(I)
            ELSE
              DELTAS(I) = CUTALT(I)
            END IF
          END IF
        END IF
      END IF
C
C STEP 4: SET THE VARIABLE-SET PARAMETER TO CALCULATE A SET
C           OF BOUNDED STOCHASTIC VALUES BY WHICH THE VALUES
C           IN VARIABLE-SET CAN BE MODIFIED.
C
      TEMP(I) = BSTSET(I) + DELTAS(I)
      IF(TEMP(I) .GT. LOWER(I) .AND. TEMP(I) .LT. UPPER(I)) THEN
        VARSET(I) = TEMP(I)
      ELSE
        VARSET(I) = BSTSET(I)
      END IF
      30 CONTINUE
C
C STEP 5: SET THE VARIABLE-FLAG PARAMETER TO FLAG IF THE VALUES
C           IN VARIABLE-SET AND VALUE-SET ARE IDENTICAL FOR ALL
C           VARIABLES.
C
      DO 40 I=1, N
        IF(VARSET(I) .NE. BSTSET(I)) THEN
          VARFLG = .FALSE.
          GO TO 50
        END IF
      40 CONTINUE
C
C IF THE VALUES IN VARIABLE-SET AND VALUE-SET ARE IDENTICAL FOR
C ALL VARIABLES, THE FOLLOWING STEPS ARE NOT NECESSARY.
C
      VARFLG = .TRUE.
      GO TO 20
C
C STEP 6: SET THE VALUE PARAMETER TO COMPUTE A VALUE FOR EACH
C           FUNCTION CALL.
C
      50 VALUE = TESFNC(N, VARSET)
C

```

```

C STEP 7-STEP 9:
C STEP 7: SET THE VALUE-FLAG PARAMETER TO FLAG IF A NEW MINIMUM
C OBTAINED.
C STEP 8: SET THE BEST-VALUE PARAMETER TO STORE THE MINIMUM
C VALUE OBTAINED ACROSS ITERATIONS.
C STEP 9: SET VALUE-SET PARAMETER TO STORE THE VALUES OF THE
C VARIABLE SET THAT GENERATES THE BEST VALUE.
C
C IF(ITER .EQ. 1) BSTVAL = VALUE
C IF(VALUE .LT. BSTVAL) THEN
C VALFLG = .TRUE.
C BSTVAL = VALUE
C DO 60 I=1, N
C BSTSET(I) = VARSET(I)
60 CONTINUE
C END IF
C
C STEP 10: EXIT TEST: RETURN TO THE BEGINING OF THE LOOP UNLESS
C MULTIPLE-DIMENSION-ITERATIONS = ITERATE.
C
C IF(ITER .LT. MDITER ) GO TO 20
C
C RETURN
C END
C
C SUBROUTINE SINGER(N, START, BSTVAL, BSTSET, LOWER, UPPER,
C * NRANGE, CUT, SEED, NEXTVA, FLGDIR, TMP2, VAR2)
C
C THIS IS A MONTE CARLO ALGORITHM FOR CHANGING THE VALUES OF
C ONE VARIABLES. ANNEALING PRINCIPLES ARE USED IN
C CALCULATING THE MAXIMUM RANGE OF THE VARIABLES ON EACH
C ITERATION. FAST COOLING IS IMPLEMENTED AS THE RANGE IS
C LOGARITHMICALLY REDUCED. RANDOMNESS IS INTRODUCED INTO THE
C FIT BY MULTIPLYING THE MAXIMUM RANGE OF THE FIT VARIABLE BY
C A RANDOM NUMBER BETWEEN -1 AND 1 TO COMPUTE THE DELTA VALUE
C FOR EACH ITERATION. THE BEST FITTING SCORE AND RELATED
C VARIABLE SET ARE RETURNED BY THE FUNCTION.
C
C THIS IS THE OUTER LOOP OF THE SINGER-DIMENSION FUNCTION.
C
C IMPLICIT REAL*8(A-H,O-Z)
C
C INTEGER I, J, N, COUNT1, COUNT2, CNT1, CNT2, NEXTVA, FLGDIR, MOD,
C * SDITER, MDITER, SDCONT, MDCONT, LOGSNG, LOGMUL, SEED
C DOUBLE PRECISION BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2,
C * BSTVAL, START(N), BSTSET(N), NRANGE(N), VAR2(N),
C * UPPER(N), LOWER(N), CUT(N), TMP2(N), TMPBST
C LOGICAL VARFLG, VALFLG
C
C COMMON /A1/COUNT1, COUNT2, BUMP, EPS, HIT, TRIAL, TORUS, SCAL1, SCAL2
C COMMON /A3/SDITER, MDITER, SDCONT, MDCONT, LOGSNG, LOGMUL
C
C DO 50 J=1, COUNT1
C DO 20 CNT1=1, N
C IF( FLGDIR .EQ. 0 ) THEN
C CNT2 = MOD((NEXTVA+CNT1), N) + 1
C ELSE
C CNT2 = MOD((NEXTVA-CNT1+N), N) + 1
C END IF
C
C CALL SNG1(CNT2, N, START, TMPBST, TMP2, LOWER, UPPER,
C * NRANGE, CUT, SEED, VAR2)
C DO 10 I=1, N
C START(I) = TMP2(I)
10 CONTINUE

```

```

20    CONTINUE
      IF(J.EQ.1) THEN
          BSTVAL = TMPBST
          DO 30 I=1, N
              BSTSET(I) = TMP2(I)
30    CONTINUE
          ELSE IF(TMPBST.LT.BSTVAL) THEN
              BSTVAL = TMPBST
              DO 40 I=1, N
                  BSTSET(I) = TMP2(I)
40    CONTINUE
          END IF
50    CONTINUE
      RETURN
      END
      SUBROUTINE SNG1(CNT2,N,START,BSTVAL,BSTSET,LOWER,UPPER,
*    NRANGE,CUT,SEED,VAR2)
C
C    THIS IS THE INNER LOOP OF THE SINGLE-DIMENSION FUNCTION. IT IS
C    IDENTICAL TO THE MULTIDIMENSION FUNCTION EXCEPT THAT LOG-CONSTANT
C    IS SET TO LN(SINGLE-DIMENSION-ITERATIONS), AND ONLY THE VARIABLES
C    DESIGNATED BY CNT2 IS CHANGED ON EACH PASS.
C
      IMPLICIT REAL*8(A-H,O-Z)
C
      INTEGER I,N,CNT2,ITER,FUNCNU,MAXFUN,SDITER,
*    MDITER,SDCONT,MDCONT,LOGSNG,LOGMUL,SEED
      DOUBLE PRECISION VALUE,TESFNC,RAND,BSTVAL,START(N),
*    DABS,BSTSET(N),NRANGE(N),UPPER(N),LOWER(N),
*    DLOG,CUT(N),VAR2(N),ITERDT,DELTA,TEMP,DTEMP
      LOGICAL VARFLG,VALFLG
C
      COMMON /A3/SDITER,MDITER,SDCONT,MDCONT,LOGSNG,LOGMUL
C
      ITER = 0
      VARFLG = .FALSE.
      VALFLG = .FALSE.
C
      DO 10 I=1, N
          BSTSET(I) = START(I)
          VAR2(I) = BSTSET(I)
10    CONTINUE
C
20    CONTINUE
      IF(.NOT.VARFLG) ITER = ITER + 1
      IF(ITER.EQ.1) GO TO 30
C
      DTEMP=ITER
      ITERDT=1.0D0-DLOG(DTEMP)/LOGSNG
C
      DELTA = ITERDT * NRANGE(CNT2) * RAND(SEED)
      IF((DABS(DELTA)).LT.CUT(CNT2)) THEN
          DELTA = 16 * CUT(CNT2) * RAND(SEED)
          IF((DABS(DELTA)).LT.CUT(CNT2)) THEN
              IF(DELTA.LT.0) THEN
                  DELTA = -CUT(CNT2)
              ELSE
                  DELTA = CUT(CNT2)
              END IF
          END IF
      END IF
C
      TEMP = BSTSET(CNT2) + DELTA
      IF(TEMP.GT.LOWER(CNT2).AND.TEMP.LT.UPPER(CNT2)) THEN
          VAR2(CNT2) = TEMP

```

```

ELSE
  VAR2 (CNT2) = BSTSET (CNT2)
END IF
C
IF (VAR2 (CNT2) .NE. BSTSET (CNT2)) THEN
  VARFLG = .FALSE.
  GO TO 30
END IF
VARFLG = .TRUE.
GO TO 20
C
30 VALUE = TEFNC (N, VAR2)
IF (ITER .EQ. 1) BSTVAL = VALUE
IF (VALUE .LT. BSTVAL) THEN
  VALFLG = .TRUE.
  BSTVAL = VALUE
  BSTSET (CNT2) = VAR2 (CNT2)
END IF
C
IF (ITER .LT. SDITER ) GO TO 20
C
RETURN
END

```

VITA

Debao Chen

Candidate for the Degree of

Master of Science

Thesis: COMPARISONS AMONG STOCHASTIC OPTIMIZATION ALGORITHMS

Major Field: Computer Science

Biographical:

Education: Graduated from Shanghai Second Institute of Education, China, and Shanghai Institute of Education, China, in July 1980 and July 1984, respectively. Graduated from Fudan University, China, in January 1986; received a diploma in graduate program in Mathematics for Assistant Professors. Graduated from The University of Texas at Austin in August 1994 and received a Philosophy of Doctor degree in Mathematics. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in December 1997.

Professional Experience: Employed as an assistant professor by Shanghai Second Institute of Education, China, August 1980 to August 1988. Employed as a graduate teaching assistant and a graduate research assistant by the Mathematics Department, The University of Texas at Austin, August 1988 to August 1994. Employed as an assistant professor by the Mathematics Department, Oklahoma State University, August 1994 to August 1995.

Professional Membership: America Mathematical Society.