

The Cross-Entropy Method for Continuous Multi-extremal Optimization

Dirk P. Kroese
Department of Mathematics
The University of Queensland
Brisbane 4072, Australia

Sergey Porotsky
Optimata Ltd.
11 Tuval St., Ramat Gan, 52522
Israel.

Reuven Y. Rubinstein
Faculty of Industrial Engineering and Management
Technion, Haifa
Israel

Abstract

In recent years, the cross-entropy method has been successfully applied to a wide range of discrete optimization tasks. In this paper we consider the cross-entropy method in the context of continuous optimization. We demonstrate the effectiveness of the cross-entropy method for solving difficult continuous multi-extremal optimization problems, including those with non-linear constraints.

Key words: Cross-entropy, continuous optimization, multi-extremal objective function, dynamic smoothing, constrained optimization, nonlinear constraints, acceptance–rejection, penalty function.

1 Introduction

The cross-entropy (CE) method (Rubinstein and Kroese, 2004) was motivated by Rubinstein (1997), where an adaptive variance minimization algorithm for estimating probabilities of rare events for stochastic networks was presented. It was modified in Rubinstein (1999) to solve combinatorial optimization problems.

The main idea behind using CE for continuous multi-extremal optimization is the same as the one for combinatorial optimization, namely to first associate with each optimization problem a rare event estimation problem — the so-called *associated stochastic problem* (ASP) — and then to tackle this ASP efficiently by an adaptive algorithm. The principle outcome of this approach is the construction of a random sequence of solutions which converges probabilistically to the optimal or near-optimal solution.

As soon as the ASP is defined, the CE method involves the following two iterative phases:

1. Generation of a sample of random data (trajectories, vectors, etc.) according to a specified random mechanism.
2. Updating the parameters of the random mechanism, typically parameters of pdfs, on the basis of the data, to produce a “better” sample in the next iteration.

The CE method has been successfully applied to a variety of problems in combinatorial optimization and rare-event estimation, the latter with both light- and heavy-tailed distributions. Applications areas include buffer allocation, queueing models of telecommunication systems, neural computation, control and navigation, DNA sequence alignment, signal processing, scheduling, vehicle routing, reinforcement learning, project management and reliability systems. References and more details on applications and theory can be found in the CE tutorial (de Boer *et al.*, 2004) and the recent CE monograph (Rubinstein and Kroese, 2004). It is important to note that the CE method deals successfully with both deterministic problems, such as the traveling salesman problem, and noisy (i.e., simulation-based) problems, such as the buffer allocation problem.

The goal of this paper is to demonstrate the quite accurate performance of the CE method for solving difficult *continuous* multi-extremal problems, both constrained and unconstrained. In particular we show numerically that typically it finds the optimal (or near-optimal) solution very fast and provides more accurate results than those reported in the literature.

It is out of the scope of this paper to review the huge literature on analytical/deterministic methods for solving optimization problem. Many of these methods are based on gradient or pseudo-gradient techniques. Examples are Line Search, Gradient Descent, Newton-Type Methods, Variable Metric, Conjugate Gradient etc. The main drawback of gradient-based methods is that they, by their nature, do not cope well with optimization problems that have non-convex objective functions and/or many local optima. Such multi-extremal continuous optimization problems arise abundantly in applications. A popular and convenient approach to these type of problems is to use random search techniques. The basic idea behind such methods is to systematically partition the feasible region into smaller subregions and then to move from one subregion to another based on information obtained by random search. Well-known examples include simulated annealing (Aarts and Korst, 1989), threshold acceptance, (Dueck and Scheur, 1990), genetic algorithms (Goldberg, 1989), tabu search (Glover and Laguna, 1993), ant colony method (Dorigo *et al.*, 1996), and the stochastic comparison method (Gong *et al.*, 1992).

The rest of this paper is organized as follows. Section 2 presents an overview of the CE method. In section 3 we give the main algorithm for continuous multi-extremal optimization using multi-dimensional normal sampling with independent components. A particular emphasis is put on the issue of different updating procedures for the parameters of the normal pdf, so-called *fixed* and *dynamic* smoothing. In Section 4 we present numerical results with the CE Algorithm for both unconstrained and constrained programs. We demonstrate the high accuracy of CE using two approaches to constrained optimization: the *acceptance-rejection* approach and the *penalty* approach.

2 The Main CE Algorithm for Optimization

The main idea of the CE method for optimization can be stated as follows: Suppose we wish to maximize some “performance” function $S(\mathbf{x})$ over all elements/states \mathbf{x} in some set \mathcal{X} . Let us denote the maximum by γ^* , thus

$$\gamma^* = \max_{\mathbf{x} \in \mathcal{X}} S(\mathbf{x}) . \tag{1}$$

To proceed with CE, we first randomize our deterministic problem by defining a family of pdfs

$\{f(\cdot; \mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ on the set \mathcal{X} . Next, we associate with (1) the estimation of

$$\ell(\gamma) = \mathbb{P}_{\mathbf{u}}(S(\mathbf{X}) \geq \gamma) = \mathbb{E}_{\mathbf{u}} I_{\{S(\mathbf{X}) \geq \gamma\}}, \quad (2)$$

the so-called *associated stochastic problem* (ASP). Here, \mathbf{X} is a random vector with pdf $f(\cdot; \mathbf{u})$, for some $\mathbf{u} \in \mathcal{V}$ (for example, \mathbf{X} could be a normal random vector) and γ is a known or unknown parameter. Note that there are in fact two possible estimation problems associated with (2). For a given γ we can estimate ℓ , or alternatively, for a given ℓ we can estimate γ , the root of (2). Let us consider the problem of estimating ℓ for a certain γ close to γ^* . Then, typically $\{S(\mathbf{X}) \geq \gamma\}$ is a rare event, and estimation of ℓ is a non-trivial problem. The CE method solves this efficiently by making adaptive changes to the probability density function according to the Kullback-Leibler CE, thus creating a sequence $f(\cdot; \mathbf{u}), f(\cdot; \mathbf{v}_1), f(\cdot; \mathbf{v}_2), \dots$ of pdfs that are “steered” in the direction of the theoretically optimal density $f(\cdot; \mathbf{v}^*)$ corresponding to the degenerate density at an optimal point. In fact, the CE method generates a sequence of tuples $\{(\gamma_t, \mathbf{v}_t)\}$, which converges quickly to a small neighborhood of the optimal tuple (γ^*, \mathbf{v}^*) . More specifically, we initialize by setting $\mathbf{v}_0 = \mathbf{u}$, choosing a not very small quantity ϱ , say $\varrho = 10^{-2}$, and then we proceed as follows:

1. **Adaptive updating of γ_t .** For a fixed \mathbf{v}_{t-1} , let γ_t be the $(1 - \varrho)$ -quantile of $S(\mathbf{X})$ under \mathbf{v}_{t-1} . That is, γ_t satisfies

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \geq \gamma_t) \geq \varrho, \quad (3)$$

$$\mathbb{P}_{\mathbf{v}_{t-1}}(S(\mathbf{X}) \leq \gamma_t) \geq 1 - \varrho, \quad (4)$$

where $\mathbf{X} \sim f(\cdot; \mathbf{v}_{t-1})$.

A simple estimator of γ_t , denoted $\hat{\gamma}_t$, can be obtained by drawing a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from $f(\cdot; \mathbf{v}_{t-1})$ and evaluating the sample $(1 - \varrho)$ -quantile of the performances as

$$\hat{\gamma}_t = S_{(\lceil (1-\varrho)N \rceil)}. \quad (5)$$

2. **Adaptive updating of \mathbf{v}_t .** For fixed γ_t and \mathbf{v}_{t-1} , derive \mathbf{v}_t from the solution of the program

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v}_{t-1}} I_{\{S(\mathbf{X}) \geq \gamma_t\}} \ln f(\mathbf{X}; \mathbf{v}). \quad (6)$$

The stochastic counterpart of (6) is as follows: for fixed $\hat{\gamma}_t$ and $\hat{\mathbf{v}}_{t-1}$ (the estimate of \mathbf{v}_{t-1}), derive $\hat{\mathbf{v}}_t$ from the following program

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^N I_{\{S(\mathbf{X}_i) \geq \hat{\gamma}_t\}} \ln f(\mathbf{X}_i; \mathbf{v}). \quad (7)$$

Remark 2.1 (Smoothed Updating) Instead of updating the parameter vector \mathbf{v} directly via the solution of (7) we use the following *smoothed* version

$$\widehat{\mathbf{v}}_t = \alpha \tilde{\mathbf{v}}_t + (1 - \alpha) \widehat{\mathbf{v}}_{t-1}, \quad \forall i = 1, \dots, n, \quad (8)$$

where $\tilde{\mathbf{v}}_t$ is the parameter vector obtained from the solution of (7), and α is called the *smoothing parameter*, with $0.7 < \alpha \leq 1$. Clearly, for $\alpha = 1$ we have our original updating rule. The reason for using the smoothed (8) instead of the original updating rule is twofold: (a) to smooth out the values of $\widehat{\mathbf{v}}_t$, (b) to reduce the probability that some component $\widehat{v}_{t,i}$ of $\widehat{\mathbf{v}}_t$ will be zero or one at the first few iterations. This is particularly important when $\widehat{\mathbf{v}}_t$ is a vector or matrix of *probabilities*. Note that for $0 < \alpha \leq 1$ we always have that $\widehat{v}_{t,i} > 0$, while for $\alpha = 1$ one might have (even at the first iterations) that either $\widehat{v}_{t,i} = 0$ or $\widehat{v}_{t,i} = 1$ for some indices i . As result, the algorithm will converge to a wrong solution.

Thus, the main CE optimization algorithm, which includes smoothed updating of parameter vector \mathbf{v} can be summarized as follows.

Algorithm 1 Generic CE Algorithm for Optimization

1. Choose some $\widehat{\mathbf{v}}_0$. Set $t = 1$.
 2. Generate a sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from the density $f(\cdot; \widehat{\mathbf{v}}_{t-1})$ and compute the sample $(1 - \varrho)$ -quantile $\widehat{\gamma}_t$ of the performances according to (5).
 3. Use the **same** sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ and solve the stochastic program (7). Denote the solution by $\tilde{\mathbf{v}}_t$.
 4. Apply (8) to smooth out the vector $\tilde{\mathbf{v}}_t$.
 5. Repeat steps 2–4 until a pre-specified stopping criterion is met.
-

3 Continuous Multi-Extremal Optimization

Here we apply the CE Algorithm 1 to continuous multi-extremal optimization problems (1), for both (a) unconstrained and (b) constrained problems with non-linear boundaries. We assume henceforth that each $\mathbf{x} = (x_1, \dots, x_n)$ is a real-valued vector and that \mathcal{X} is a subset of \mathbb{R}^n .

(a) The Unconstrained Case

In this case generation of a random vector $\mathbf{X} = (X_1, \dots, X_n) \in \mathcal{X}$ is straightforward. The easiest way is to generate the coordinates independently (say from an arbitrary 2-parameter distribution), such that by applying Algorithm 1 the joint distribution converges to the degenerated distribution in “close” vicinity to the point \mathbf{x}^* where the global extremum is attained. Examples of such distributions are the normal, double-exponential and beta distributions.

The parameter updating step (7) translates into the following: Given a random sample of size N , the parameters are updated based on the $N^{\text{elite}} = \rho N$ best performing samples. These are called the *elite samples*. The updated parameters are found to be the *maximal likelihood estimates* (MLEs) of the elite samples (Rubinstein and Kroese, 2004). In particular, for the normal distribution, the parameter updating is especially simple. Namely, the parameters μ and σ^2 are updated as the sample mean and sample variance of the elite samples, see formulas (10) and (11) below.

While applying CE algorithm, the mean vector $\hat{\boldsymbol{\mu}}_t$ should converge to \mathbf{x}^* and the vector of standard deviations $\hat{\boldsymbol{\sigma}}_t$ to the zero vector. In short, we should obtain a degenerated pdf with all mass concentrated in the vicinity of the point \mathbf{x}^* .

Example 3.1 (Normal Updating) Consider optimization of the function S given by

$$S(x) = e^{-(x-2)^2} + 0.8e^{-(x+2)^2}, \quad x \in \mathbb{R}. \quad (9)$$

Note that S has a local maximum at point -2.00 (approximately) and a global maximum at 2.00 . At each stage t of the CE procedure we simulate a sample X_1, \dots, X_N from a $\mathbf{N}(\hat{\boldsymbol{\mu}}_{t-1}, \hat{\boldsymbol{\sigma}}_{t-1}^2)$ distribution, and update $\hat{\boldsymbol{\mu}}_t$ and $\hat{\boldsymbol{\sigma}}_t$ as the mean and standard deviation of the elite samples. A simple Matlab implementation is given in Appendix A. The CE procedure is illustrated in Figure 1, using starting values $\hat{\boldsymbol{\mu}}_0 = -6$, $\hat{\boldsymbol{\sigma}}_0 = 100$ and CE parameters $\alpha = 0.7$, $N^{\text{elite}} = 10$ and $N = 100$. Algorithm 2 is stopped when the standard deviation becomes smaller than $\varepsilon = 0.05$. We observe that the vector $(\hat{\boldsymbol{\mu}}_t, \hat{\boldsymbol{\sigma}}_t)$ quickly converges to the optimal $(\boldsymbol{\mu}^*, \boldsymbol{\sigma}^*) = (2.00, 0)$, easily avoiding the local maximum.

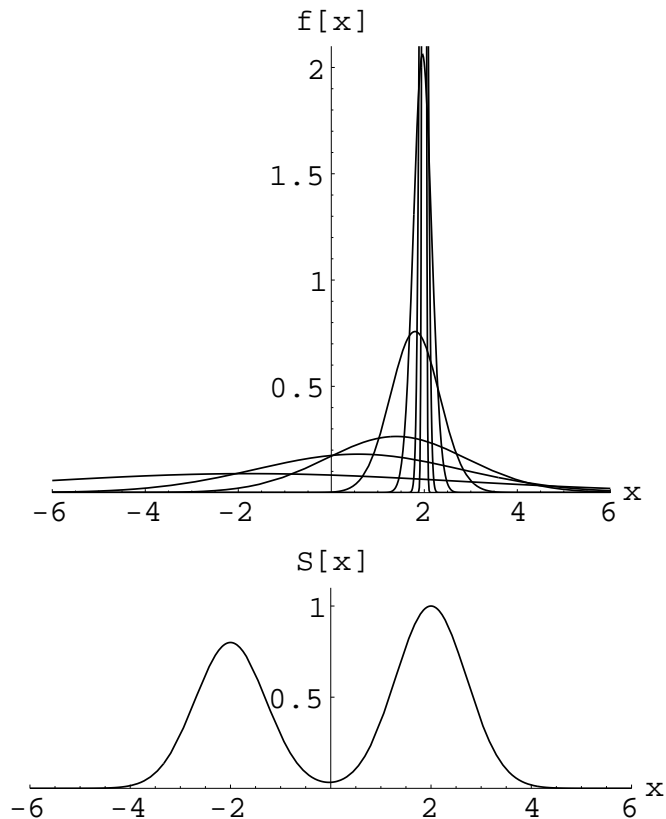


Figure 1: Continuous Multi-Extremal Optimization

The normal updating procedure above is easily adapted to the multidimensional case. Specifically, when the components of the random vectors $\mathbf{X}_1, \dots, \mathbf{X}_N$ are chosen independently, updating of the parameters can be done separately for each component. The pseudocode for the continuous CE procedure, using normal updating (with independent components) is given in Algorithm 2. The n -dimensional normal distribution with independent components, mean vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_n)$ and variance vector $\boldsymbol{\sigma}^2 = (\sigma_1^2, \dots, \sigma_n^2)$ is denoted by $N(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$.

Algorithm 2 CE Algorithm for Continuous Optimization

1: **initialize:** Choose $\widehat{\boldsymbol{\mu}}_0$ and $\widehat{\boldsymbol{\sigma}}_0^2$. Set $t := 0$.

2: **repeat**

3: **draw:** Increase t by 1. Generate a random sample $\mathbf{X}_1, \dots, \mathbf{X}_N$ from the $N(\widehat{\boldsymbol{\mu}}_{t-1}, \widehat{\boldsymbol{\sigma}}_{t-1}^2)$ distribution.

4: **select:** Let \mathcal{I} be the indices of the N^{elite} best performing (=elite) samples.

5: **update:** for all $j = 1, \dots, n$ let

$$\widetilde{\boldsymbol{\mu}}_{tj} := \sum_{i \in \mathcal{I}} X_{ij} / N^{\text{elite}} \quad (10)$$

and

$$\widetilde{\boldsymbol{\sigma}}_{tj}^2 := \sum_{i \in \mathcal{I}} (X_{ij} - \mu_{tj})^2 / N^{\text{elite}}. \quad (11)$$

6: **smooth:**

$$\widehat{\boldsymbol{\mu}}_t := \alpha \widetilde{\boldsymbol{\mu}}_t + (1 - \alpha) \widehat{\boldsymbol{\mu}}_{t-1}, \quad \widehat{\boldsymbol{\sigma}}_t := \alpha \widetilde{\boldsymbol{\sigma}}_t + (1 - \alpha) \widehat{\boldsymbol{\sigma}}_{t-1} \quad (12)$$

7: **until** $\max_j (\sigma_{tj}) < \varepsilon$

It is important to note that we found numerically that the smoothing procedure with the *fixed* smoothing parameter α works well in many cases, but fails in some other cases. To overcome this difficulty we shall use two different smoothing schemes for μ and σ . In particular, for μ we shall use the *same fixed* smoothing parameter α , ($0.5 \leq \alpha \leq 0.9$) as in (12), while the variance σ^2 we shall use the following *dynamic* smoothing

$$\beta_t = \beta - \beta \left(1 - \frac{1}{t}\right)^q, \quad (13)$$

where q is an integer (typically between 5 and 10) and β is a smoothing constant (typically between 0.8 and 0.99).

The reason for using the dynamic smoothing (13) for σ instead of a fixed smoothing can be explained as follows. With a fixed α as per (12) the convergence to a degenerate distribution will typically happen too quickly, which will in turn result into a sub-optimal solution. The goal of (13) is precisely to prevent this. It is readily seen that by choosing for σ the β_t smoothed updating instead of α one, the convergence to the degenerate case will have a polynomial speed instead of exponential.

Examples where the fixed smoothing scheme works perfectly are simple functions such as (9) or the following *trigonometric* function

$$S(\mathbf{x}) = \sum_{i=1}^n 8 \sin^2(\eta(x_i - x_i^*))^2 + 6 \sin^2(2\eta(x_i - x_i^*))^2 + \mu(x_i - x_i^*)^2. \quad (14)$$

An example where the dynamic smoothing is essential is the *Rosenbrock* function:

$$S(\mathbf{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2. \quad (15)$$

Another example (arising from chemistry) where dynamic smoothing is important is the *Hougen least squares* function:

$$S(\mathbf{x}) = \frac{1}{13} \sum_{i=1}^{13} \left(r_i - \frac{x_1 z_{i2} - z_{i3}/x_5}{1 + x_2 z_{i1} + x_3 z_{i2} + x_4 z_{i3}} \right)^2, \quad (16)$$

where the r_i and z_i are given in Table 1.

Table 1: Data for the Hougen Function

z_1	z_2	z_3	r
470	300	10	8.55
285	80	10	3.79
470	300	120	4.82
470	80	120	0.02
470	80	10	2.75
100	190	10	14.39
100	80	65	2.54
470	190	65	4.35
100	300	54	13.00
100	300	120	8.50
100	80	120	0.05
285	300	10	11.32
285	190	120	3.13

The minimal value for S is 0.02299, which is attained at

$$\mathbf{x}^* = (1.2526, 0.0628, 0.0400, 0.1124, 1.1914).$$

This function arises from the following non-linear regression problem (Bates and Watts, 1988): The reaction rate r of a certain chemical reaction depends on three input variables: quantities of hydrogen z_1 , n-pentane z_2 , and isopentane z_3 . The functional relationship is given by the *Hougen* function:

$$r = \frac{x_1 z_2 - z_3/x_5}{1 + x_2 z_1 + x_3 z_2 + x_4 z_3},$$

where x_1, \dots, x_5 are the unknown parameters. The objective is to estimate the model parameters $\{x_i\}$ from the data, as given in Table 1. The estimation is done via the least squares method. The objective function in (16) is simply the average sum of the squared deviations, to be used in the least squares minimization problem.

The graphical representations of the Rosenbrock and the trigonometric functions for $\eta = 7$, $\mu = 1$, $x_i^* = x^* = 0.9$ in the two-dimensional case are given in Figures 2 and 3, respectively. It is not difficult to see that in the n -dimensional case the global minimum for the Rosenbrock and the trigonometric function is attained at points $\mathbf{x}^* = (1, 1, \dots, 1)$ and $\mathbf{x}^* = (0.9, 0.9, \dots, 0.9)$, respectively. The corresponding minimal function values are $S(\mathbf{x}^*) = 0$. If not stated otherwise we assume (as in Rubinstein (1999)) for the trigonometric function that $\eta = 7$, $\mu = 1$.

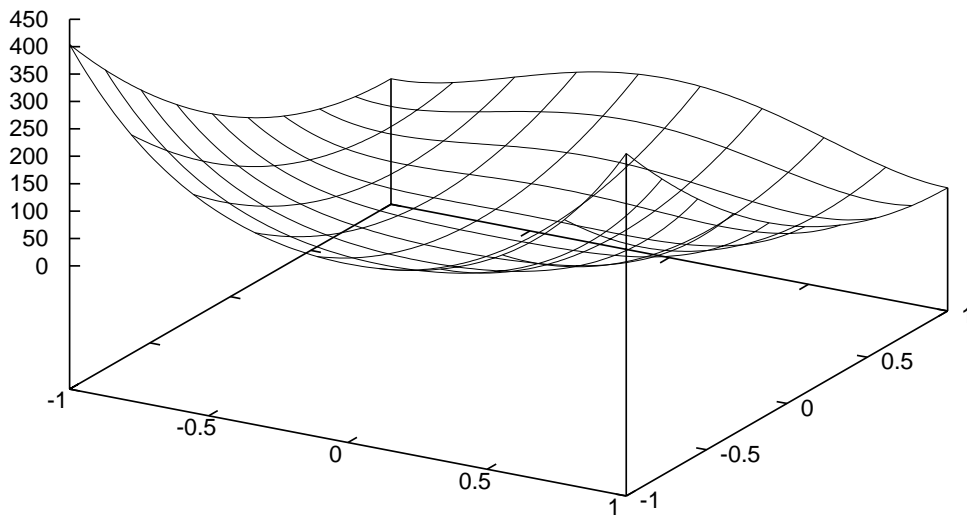


Figure 2: Rosenbrock's function in \mathbb{R}^2 for $-1 \leq x_i \leq 1$

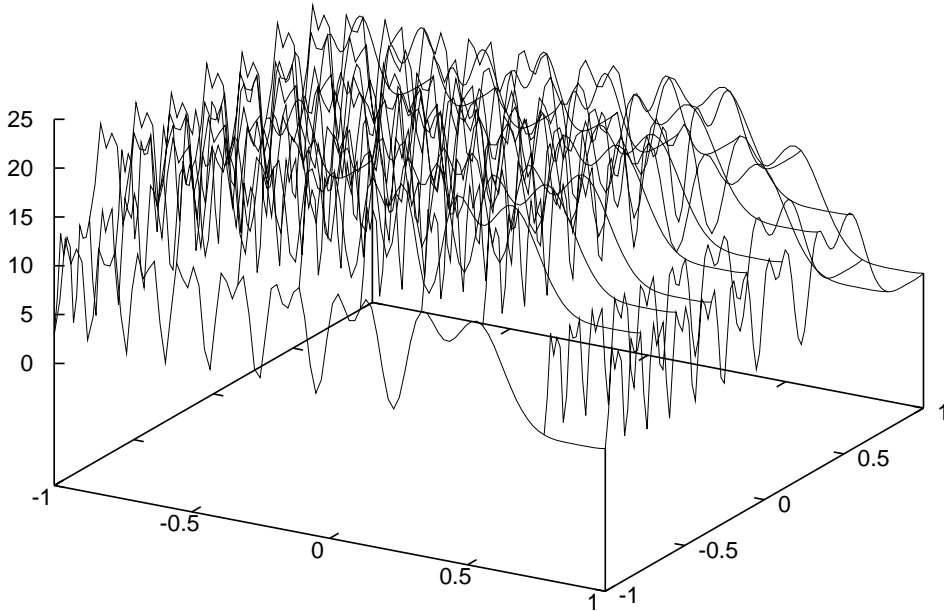


Figure 3: The trigonometric function in \mathbb{R}^2 with $\eta = 7$, $\mu = 1$, $x_i^* = x^* = 0.9$ and $-1 \leq x_i \leq 1$

(b) The Constrained Case

We consider the case where \mathcal{X} in (1) is a (non-linear) region defined by the following system of inequalities:

$$G_i(\mathbf{x}) \leq 0, \quad i = 1, \dots, L. \quad (17)$$

To solve the program (1) with constraints (17), we can apply two approaches: the *acceptance–Rejection* and the *penalty* approach.

ACCEPTANCE–REJECTION APPROACH.

Suppose that the feasible region is simply an n -dimensional rectangle: $\mathcal{X} = [\mathbf{a}, \mathbf{b}]^n$. A straightforward method for solving (1) with such simple constraints is use the acceptance–rejection (AR) method, which works as follows: Generate a random vector \mathbf{X} from a normal distribution with given parameters μ and σ^2 , then accept or reject it depending on whether the sample falls or not in the interval of interest. The accepted sample can be viewed as the one generated from the truncated normal distribution. A pleasant feature of such acceptance–rejection method is that the updating rules for the CE method remain exactly the same as for the untruncated case, see Rubinstein and Kroese (2004). In short, sampling from a truncated normal distribution on the interval $[a, b]$ (we

write $\mathbf{N}(\mu, \sigma^2, a, b)$ incurs the same updating rules (mean and variance of the elite samples) as for the untruncated case.

It is important to note that the AR method can also be used when \mathcal{X} is not a rectangle. We can simply define a rectangle \mathcal{R} that (hopefully) contains the optimal value and sample from \mathcal{R} , e.g., via the AR method, or by sampling directly from the truncated distribution, and by rejecting samples from \mathcal{R} that do not satisfy the constraints. However, \mathcal{R} should not be chosen too large either, because else too many samples are rejected.

PENALTY APPROACH.

This approach is more generally applicable. the idea is to modify the objective function as follows:

$$\tilde{S}(\mathbf{x}) = S(\mathbf{x}) + \sum_{i=1}^L P_i(\mathbf{x}), \quad (18)$$

where the $\{P_i\}$ are *penalty functions*. Specifically, the i -th penalty function P_i (corresponding to the i -th constraint) is defined as

$$P_i(\mathbf{x}) = H_i \max(G_i(\mathbf{x}), 0) \quad (19)$$

and $H_i > 0$ measures the importance (cost) of the i -th penalty. We shall call such penalty approach, the *proportional* penalty approach to distinguish it from the so-called *constant* penalty approach, where the penalty is constant, i.e., $\tilde{S}(\mathbf{x}) = S(\mathbf{x}) + H I_{\{\mathbf{x} \notin \mathcal{X}\}}$, for some constant H .

Clearly that as soon as the constrained problem (1), (17) is reduced to the unconstrained one (1) — using (18) instead of S — we can apply again Algorithm 2.

4 Numerical Results

In this section we present numerical results with the CE Algorithm 2 for both unconstrained and constrained programs (1) and (1), (17), respectively. As mentioned, the latter case is solved directly via the AR method, or is translated into an unconstrained problem by using the penalty method (18). We shall apply the AR approach to several examples where the constrains are relatively simple.

For each test case we used the following parameters: $N = 100n$, $\alpha = 0.8$, $\beta = 0.7$. For $n < 50$, we set $N^{\text{elite}} = 10$ and $q = 5$, while for $50 \leq n \leq 100$, we set $N^{\text{elite}} = 20$ and $q = 6$. We found that

(5–10)% deviation of all the above parameters from result to similar accuracy of the Algorithm 2. But, more importantly, we did not have to “tweak” each set of parameter to each problem. For each problem we first attempted the simple (fixed) smoothing, and if this was unsatisfactory we applied the dynamic smoothing. The CE algorithm is very insensitive to the choice of initial means and standard deviations, provided that the initial standard deviation is chosen large enough. For example, on an interval $[l, r]$ the initial standard deviation could be chosen as $5(r - l)$ and the initial mean uniformly in $[l, r]$.

In the tables S_t^* denotes the best (that is, smallest) function value found in iteration t , $\hat{\gamma}_t$ the worst of the elite performances, and $\hat{\boldsymbol{\mu}}_t$ the vector of means at the t -th iteration. The experiments were conducted on a 2.4GHz computer, using a similar Matlab implementation as in appendix B.

4.1 Unconstrained Optimization

Here we present simulation results with the trigonometric and Rosenbrock function. Table 2 presents the evolution of Algorithm 1 (with fixed smoothing) for the trigonometric function with $\eta = 7$, $\mu = 1$ and $n = 10$. The algorithm was stopped when all standard deviations were smaller than $\varepsilon = 10^{-5}$. In repeated experiments the global maximum was consistently found in less than 1 second. Different values for the parameters μ and ν had little effect on the excellent accuracy of the method. The final solution as observed to be accurate to at least 5 digits.

Table 2: The Evolution of Algorithm 2 for the Trigonometric Function with $\eta = 7$, $\mu = 1$ and $n = 10$, Using a Fixed Smoothing Parameter $\alpha = 0.8$

t	$\hat{\gamma}_t$	S_t^*	$\hat{\mu}_t$									
1	26217.239	16407.105	-32.34	3.39	-27.24	-7.59	9.50	33.26	1.17	22.70	19.48	-13.05
2	9547.867	7301.766	8.64	-12.50	-10.70	-8.59	-1.07	-7.05	2.16	13.70	15.84	-23.97
3	3223.558	1728.216	-4.19	-4.65	2.23	0.55	-1.91	-0.26	-4.65	-3.57	1.96	0.31
4	945.186	461.574	1.69	-3.18	-1.84	1.46	-0.21	-3.15	0.00	-1.68	3.82	-2.01
5	394.990	248.219	-2.62	-2.10	1.04	3.88	1.89	0.23	1.08	-0.49	-0.84	-0.63
6	167.775	116.924	0.77	0.92	-0.81	1.23	1.03	1.14	2.30	0.80	1.04	-0.02
7	87.474	66.761	0.16	0.09	0.80	0.38	0.91	1.02	0.95	0.50	0.71	1.53
8	56.999	41.834	0.56	0.57	1.15	0.13	0.90	0.64	1.15	0.60	0.24	0.99
9	44.042	27.412	0.43	0.56	0.59	0.32	1.05	1.54	0.66	1.00	0.09	1.27
10	40.506	27.386	0.80	0.83	0.72	0.47	0.82	1.33	1.34	0.88	0.41	0.80
11	35.428	21.344	0.62	0.96	0.66	0.58	0.97	1.03	0.92	0.87	0.63	0.86
12	28.879	22.246	0.60	0.81	0.76	0.46	1.01	0.91	1.15	0.81	0.31	0.76
13	23.012	14.374	0.98	0.81	0.63	0.69	0.97	0.91	1.18	0.91	0.48	0.97
14	15.462	3.787	0.82	0.91	0.72	0.77	0.88	0.90	1.07	0.92	0.94	0.95
15	7.937	1.510	0.89	0.91	0.90	0.84	0.85	0.91	0.97	0.93	0.91	0.97
16	1.823	0.731	0.90	0.90	0.84	0.90	0.90	0.92	0.90	0.85	0.91	0.99
17	0.342	0.080	0.92	0.90	0.89	0.92	0.90	0.92	0.91	0.88	0.89	0.91
18	0.041	0.015	0.90	0.91	0.89	0.91	0.91	0.91	0.88	0.89	0.90	0.90
19	0.005	0.003	0.89	0.90	0.90	0.90	0.91	0.90	0.90	0.90	0.89	0.90
20	0.001	0.001	0.90	0.91	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
21	0.000	0.000	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
22	0.000	0.000	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90
23	0.000	0.000	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90

Table 3 presents the evolution of Algorithm 2 (with fixed smoothing) for the minimization of the Rosenbrock function. The tolerance ε is taken to be 10^{-5} . In contrast to the Trigonometric case, the standard (constant α) implementation does not work. Note that in this case the algorithm happens to get stuck near function value 7.5; repeated experiments gave other sub-optimal values.

Table 3: The Evolution of Algorithm 2 with Fixed Smoothing, for the Rosenbrock Function with $n = 10$

t	S_t^*	$\hat{\boldsymbol{\mu}}_t$									
1	4356855562.857	29.68	15.70	-8.65	-7.49	-12.20	20.56	-2.68	18.48	25.82	28.04
2	230125218.080	9.25	-6.33	-9.58	6.75	11.89	3.50	-1.10	0.72	4.54	58.64
3	9907244.594	-2.18	-8.36	2.14	-0.26	2.13	4.20	5.14	0.69	0.12	84.64
4	1455485.756	2.50	-1.53	-1.61	0.96	3.04	2.32	0.77	3.04	-1.12	81.56
5	239936.836	0.40	-1.99	-0.91	-1.02	3.06	0.24	-0.39	1.90	1.00	128.16
6	91115.542	0.71	-1.07	-0.16	-0.20	0.13	0.22	-0.68	0.97	-4.05	71.74
7	24308.911	-0.32	-0.78	0.74	0.33	0.07	0.00	-0.24	0.61	0.63	37.18
8	3122.847	0.02	-0.26	0.38	0.17	-0.10	0.18	-0.36	1.28	2.93	15.13
9	1510.490	0.14	0.33	-0.07	0.05	0.21	0.15	-0.47	1.43	1.94	8.83
10	293.475	0.13	0.33	0.19	0.04	0.39	-0.19	-0.23	0.85	1.20	2.55
11	166.794	0.01	0.36	0.19	0.15	0.25	-0.08	-0.17	0.89	1.15	1.80
12	101.828	0.13	0.24	0.11	0.19	0.30	0.01	0.07	0.71	0.57	0.66
13	39.909	0.06	0.16	0.05	0.15	0.12	0.10	0.19	0.54	0.51	0.32
14	40.796	0.03	0.20	0.11	0.13	0.13	0.00	0.04	0.43	0.36	0.23
15	23.482	0.05	0.17	0.03	0.11	0.07	0.00	0.10	0.32	0.25	0.10
16	18.294	0.21	0.16	0.05	0.09	0.05	0.04	0.02	0.23	0.16	0.08
17	12.354	0.28	0.11	0.05	0.05	0.04	0.01	0.03	0.14	0.12	0.06
18	9.127	0.35	0.13	0.04	0.03	0.03	0.01	0.03	0.08	0.07	0.05
19	8.724	0.39	0.15	0.04	0.03	0.02	0.02	0.04	0.05	0.02	0.02
20	8.047	0.44	0.19	0.04	0.02	0.01	0.03	0.02	0.03	0.03	0.01
21	7.820	0.47	0.22	0.05	0.01	0.01	0.02	0.02	0.02	0.02	0.01
22	7.726	0.50	0.25	0.06	0.01	0.01	0.01	0.01	0.02	0.02	0.01
23	7.634	0.51	0.26	0.07	0.01	0.01	0.01	0.01	0.01	0.02	0.01
24	7.578	0.52	0.27	0.07	0.02	0.01	0.01	0.01	0.01	0.01	0.01
25	7.555	0.52	0.27	0.07	0.02	0.01	0.01	0.01	0.01	0.01	0.00
26	7.519	0.53	0.28	0.08	0.02	0.01	0.01	0.01	0.01	0.01	0.00
27	7.521	0.53	0.28	0.08	0.02	0.01	0.01	0.01	0.01	0.01	0.00
28	7.512	0.53	0.28	0.08	0.02	0.01	0.01	0.01	0.01	0.01	0.00
29	7.503	0.53	0.28	0.08	0.02	0.01	0.01	0.01	0.01	0.01	0.00

However, with dynamic smoothing the CE algorithm finds the the global maximum consistently, in less than 3 seconds, using $\varepsilon = 10^{-3}$. It is interesting to note that always the first component

converges first, then the second, third etc.

Table 4: The Evolution of Algorithm 2 with Dynamic Smoothing for the Rosenbrock Function with $n = 10$

t	$\hat{\gamma}_t$	S_t^*	$\hat{\boldsymbol{\mu}}_t$										
100	21.097	16.287	0.31	0.15	0.06	-0.06	0.01	0.02	-0.00	-0.07	0.11	0.03	
200	7.219	6.391	0.86	0.74	0.53	0.29	0.10	0.04	0.00	-0.00	0.02	-0.00	
300	3.403	3.002	0.98	0.96	0.93	0.86	0.74	0.54	0.30	0.10	0.03	-0.01	
400	1.063	0.864	1.00	0.99	0.99	0.97	0.94	0.88	0.77	0.59	0.35	0.12	
500	0.418	0.312	1.00	0.99	0.99	0.98	0.97	0.94	0.88	0.77	0.59	0.35	
600	0.249	0.193	1.00	1.00	1.00	0.99	0.98	0.96	0.93	0.85	0.73	0.52	
700	0.115	0.070	1.00	1.00	1.00	1.00	0.99	0.98	0.96	0.92	0.84	0.69	
800	0.063	0.043	1.00	1.00	1.00	1.00	0.99	0.99	0.97	0.95	0.90	0.81	
900	0.037	0.023	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.98	0.96	0.91	
1000	0.024	0.012	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.98	0.95	

In this case the final mean vector is $(1.0013, 1.0002, 0.9995, 1.0002, 0.9980, 0.9969, 0.9929, 0.9865, 0.9738, 0.9466)$, with a function value of 0.014.

4.2 Acceptance–Rejection

We use the AR method to minimize the Hougen least squares function, constrained to the 5-dimensional rectangle $[0, \mathbf{2}]$. Table 5 depicts the evolution of the Algorithm 2 (with dynamic smoothing). The tolerance was set to $\varepsilon = 10^{-7}$. In repeated experiments the global minimum was reliably found. The initial parameter vectors are $\hat{\boldsymbol{\mu}}_0 = \mathbf{1}$ and $\hat{\boldsymbol{\sigma}}_0 = \mathbf{2}$. The simulation time for 8000 iterations was 30 seconds.

Table 5: The Evolution of Algorithm 2 for the Hougen Least Squares Function Using Constant Smoothing

t	S_t^*	$\hat{\boldsymbol{\mu}}_t$				
500	0.02311	1.45482	0.07294	0.04723	0.12983	1.02368
1000	0.02304	1.37263	0.06881	0.04431	0.12277	1.08589
1500	0.02301	1.32930	0.06663	0.04277	0.11903	1.12177
2000	0.02300	1.30294	0.06531	0.04184	0.11676	1.14476
2500	0.02300	1.28579	0.06444	0.04123	0.11528	1.16019
3000	0.02299	1.27412	0.06386	0.04081	0.11427	1.17098
3500	0.02299	1.26619	0.06346	0.04053	0.11359	1.17843
4000	0.02299	1.26092	0.06319	0.04034	0.11314	1.18341
4500	0.02299	1.25739	0.06302	0.04022	0.11283	1.18677
5000	0.02299	1.25526	0.06291	0.04014	0.11264	1.18881
5500	0.02299	1.25406	0.06285	0.04010	0.11254	1.18996
6000	0.02299	1.25339	0.06282	0.04008	0.11248	1.19061
6500	0.02299	1.25303	0.06280	0.04006	0.11245	1.19095
7000	0.02299	1.25278	0.06279	0.04005	0.11243	1.19120
7500	0.02299	1.25269	0.06278	0.04005	0.11242	1.19128
8000	0.02299	1.25264	0.06278	0.04005	0.11242	1.19132

Below we consider two more examples, where the AR method has been successfully applied.

Example 4.1 The following optimization test problem is selected from Hock and Schittkowski (1981), Problem 112 on page 121, which uses optimization methods from Schittkowski (1980), such as line-search algorithms, unconstrained optimization, quadratic programming, penalty methods, multiplier methods and the generalized reduced gradient methods. Some other references (that indicate where this problem is originally from and where additional information or test results can be found) are Bracken and McCormick (1968); Himmelblau (1972); White (1992).

The nonlinear programming problem is to find \mathbf{x} so as to minimize the objective function

$$S(\mathbf{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \dots + x_{10}} \right),$$

subject to the following set of constraints:

$$\begin{aligned}x_1 + 2x_2 + 2x_3 + x_6 + x_{10} - 2 &= 0, \\x_4 + 2x_5 + x_6 + x_7 - 1 &= 0, \\x_3 + x_7 + x_8 + 2x_9 + x_{10} - 1 &= 0, \\x_j &\geq 0.000001, \quad j = 1, \dots, 10,\end{aligned}$$

where the constants $\{c_i\}$ are given in Table 6.

Table 6: Constants for Test Problem 112

$c_1 = -6.089; c_2 = -17.164; c_3 = -34.054; c_4 = -5.914; c_5 = -24.721;$ $c_6 = -14.986; c_7 = -24.100; c_8 = -10.708; c_9 = -26.662; c_{10} = -22.179.$

The best known solution in Hock and Schittkowski (1981) was

$$\mathbf{x}^* = (0.01773548, 0.08200180, 0.8825646, 0.0007233256, 0.4907851, \\ 0.0004335469, 0.01727298, 0.007765639, 0.01984929, 0.05269826),$$

with $S(\mathbf{x}^*) = -47.707579$. However, using genetic algorithms (Michalewicz, 1996) finds a better solution:

$$\mathbf{x}^* = (0.04034785, 0.15386976, 0.77497089, 0.00167479, 0.48468539, \\ 0.00068965, 0.02826479, 0.01849179, 0.03849563, 0.10128126),$$

with $S(\mathbf{x}^*) = -47.760765$. A single run of 1000 iterations took 56 seconds of CPU time. Using the CE method — with constant smoothing — we can find an even better solution (using a comparable amount of time as Michalewicz (1996)):

$$\mathbf{x}^* = (0.04067247, 0.14765159, 0.78323637, 0.00141368, 0.48526222, \\ 0.00069291, 0.02736897, 0.01794290, 0.03729653, 0.09685870)$$

and $S(\mathbf{x}^*) = -47.76109081$ (using $\varepsilon = 10^{-8}$).

We now explain in more detail how the CE method is applied to this problem. The first step is to reduce the search space by expressing x_1 , x_4 , x_8 in terms of the other seven variables:

$$\begin{aligned}x_1 &= 2 - (2x_2 + 2x_3 + x_6 + x_{10}), \\x_4 &= 1 - (2x_5 + x_6 + x_7), \\x_8 &= 1 - (x_3 + x_7 + 2x_9 + x_{10}).\end{aligned}$$

Hence, we have reduced the original problem of ten variables to that of a function of seven variables, which are subject to the following linear constraints:

$$\begin{aligned}x_2, x_3, x_5, x_6, x_7, x_9, x_{10} &\geq 0.000001, \\2 - (2x_2 + 2x_3 + x_6 + x_{10}) &\geq 0.000001, \\1 - (2x_5 + x_6 + x_7) &\geq 0.000001, \\1 - (x_3 + x_7 + 2x_9 + x_{10}) &\geq 0.000001.\end{aligned}$$

The next step is to choose an appropriate rectangular search space \mathcal{R} . The samples are drawn from a truncated normal distribution (with independent components) on this space. Thus, each component is drawn independently from a truncated normal distribution on some interval, either by sampling directly from this distribution, or by sampling from the ordinary normal distribution followed by the acceptance–rejection method. The rectangle \mathcal{R} does not have to be a subspace of the 7-dimensional search space given above. In this particular case we choose \mathcal{R} such that

$$\begin{aligned}0.000001 &\leq x_2 \leq 0.5, \\0.5 &\leq x_3 \leq 0.9, \\0.000001 &\leq x_5 \leq 0.5, \\0.000001 &\leq x_6 \leq 0.001, \\0.000001 &\leq x_7 \leq 0.05, \\0.000001 &\leq x_9 \leq 0.05, \\0.000001 &\leq x_{10} \leq 0.5.\end{aligned}$$

We now apply Algorithm 2 on the multi-dimensional rectangle \mathcal{R} , rejecting samples that do not satisfy the constraints.

Example 4.2 This test example with non-linear constraints is taken from Hock and Schittkowski (1981), Problem 63 on page 85. Other sources of reference are Himmelblau (1972); Paviani (1969); Sheela and Ramaoorthy (1975). The objective function is:

$$S(\mathbf{x}) = 1000 - x_1^2 - 2x_2^2 - x_3^2 - x_1x_2 - x_1x_3,$$

subject to the constraints:

$$\begin{aligned} 8x_1 + 14x_2 + 7x_3 - 56 &= 0, \\ x_1^2 + x_2^2 + x_3^2 - 25 &= 0, \\ x_j &\geq 0, \quad j = 1, 2, 3. \end{aligned}$$

Similar to the previous test example, we eliminate two variables x_2 and x_3 and express them in terms of x_1 , which we constrain to an interval. For the latter we choose $0 \leq x_1 \leq 5$. For x_2 and x_3 we have either

$$\begin{aligned} x_2 &= \frac{224 - 32x_1 - 2\sqrt{2989 + 896x_1 - 309x_1^2}}{70}, \\ x_3 &= \frac{2\left(28 - 4x_1 + \sqrt{2989 + 896x_1 - 309x_1^2}\right)}{35}, \end{aligned}$$

or

$$\begin{aligned} x_2 &= \frac{224 - 32x_1 + 2\sqrt{2989 + 896x_1 - 309x_1^2}}{70}, \\ x_3 &= \frac{2\left(28 - 4x_1 - \sqrt{2989 + 896x_1 - 309x_1^2}\right)}{35}. \end{aligned}$$

Each vector \mathbf{X} in the CE algorithm is now drawn as follows: First, we draw X_1 according to a truncated normal distribution on $[0,5]$. Then, we choose either the first or the second solution for (X_2, X_3) above, with equal probability. The best known global solution in Hock and Schittkowski (1981) is

$$\mathbf{x}^* = (3.512118414, 0.2169881741, 3.552174034),$$

with $S(\mathbf{x}^*) = 961.7151721$.

Table 7 shows the progress of the CE algorithm (using dynamic smoothing). The initial parameters are: mean $\hat{\boldsymbol{\mu}}_0 = (0, 0, 0)$ and standard deviation $\hat{\boldsymbol{\sigma}}_0 = (1, 1, 1)$. It took less than 1 second to find the solution

$$\mathbf{x}^* = (3.512120196, 0.216988032, 3.552172282),$$

with $S(\mathbf{x}^*) = 961.715172130054$, using a tolerance of $\varepsilon = 10^{-6}$. Fixed smoothing is somewhat less accurate, giving typically only the first 3 significant digits.

Table 7: Evolution of the CE Method for Problem 63, Using Dynamic Smoothing

t	S_t^*	\mathbf{x}_t		
3	961.715568341919	3.496826535945	0.218483551317	3.566659713430
6	961.715172566952	3.511034947979	0.217074975981	3.553238678918
9	961.715172130248	3.512301551911	0.216973631926	3.551993819679
12	961.715172130932	3.512049391288	0.216993664593	3.552241937912
15	961.715172130091	3.512143711972	0.216986164164	3.552149143705
18	961.715172130114	3.512117071139	0.216988281388	3.552175355923
21	961.715172130124	3.512126761278	0.216987510817	3.552165822620

4.3 Penalty Method

Here we apply the proportional penalty approach to the constrained minimization of the Rosenbrock function of dimension 10.

Table 8 displays the results of 8 experiments, listing for each experiment the constraints, the minimal value obtained by Algorithm 2 (with dynamic smoothing), and the CPU time. In all experiments we used $\varepsilon = 10^{-3}$ and $H = 1000$, except in the third one, where $H = 2000$. Repeated experiments found consistently the same values, indicating that the true global minimum was reached in each case.

Table 8: Constrained Rosenbrock Problems

Constraints	S_T^*	secs
$\sum_{j=1}^{10} x_j \leq -8$	1517.8	20
$\sum_{j=1}^{10} x_j \leq -10$	2677.4	14
$\sum_{j=1}^{10} x_j \leq -15$	7489.4	20
$\sum_{j=1}^{10} x_j \geq 15$	1.32	4
$\sum_{j=1}^{10} x_j \leq -8, \sum_{j=1}^{10} x_j^2 \geq 8$	1517.8	27
$\sum_{j=1}^{10} x_j \leq -8, \sum_{j=1}^{10} x_j^2 \geq 15$	1764.0	4
$\sum_{j=1}^{10} x_j \leq -8, \sum_{j=1}^{10} x_j^2 \geq 22.5$	2337.6	5
$\sum_{j=1}^{10} x_j \geq 15, \sum_{j=1}^{10} x_j^2 \leq 22.5$	0.241	4

We note that results using constant penalty approach, that is, using performance function $\tilde{S}(\mathbf{x}) = S(\mathbf{x}) + H I_{\{\mathbf{x} \in \mathcal{X}\}}$, were not as good as the ones listed above. The reason is that with the constant penalty approach it can happen that (almost) all samples fall outside \mathcal{X} and therefore incur a (large) penalty. This is very similar to the AR approach when (almost) all samples are rejected.

We have repeated Example 4.1 with the penalty approach. Specifically, x_1, x_4 and x_8 are again expressed in terms of the other seven variables, but now the latter ones are generated according to an ordinary (untruncated) normal distribution; and a proportional penalty is added to the objective function, for each of the 10 constraints. Actually, because of the occurrence of the logarithm in the objective function, many samples yield *complex* values. For this reason we only consider the *real* part of the modified function. We found that the penalty approach with constant smoothing works even better than the AR approach, provided that the smoothing parameter for σ is chosen not too

large, say 0.2. Setting $\varepsilon = 10^{-8}$ and $H_i = 1000$, the penalty approach yielded the solution

$$\mathbf{x}^* = (0.0406680568, 0.1477303255, 0.7831533568, 0.00141421020, 0.4852466432, \\ 0.0006931926, 0.0273993108, 0.0179472407, 0.0373143529, 0.0968713859),$$

with $S(\mathbf{x}^*) = -47.7610908594$, in 12 seconds. This is the best known solution thus far.

5 Conclusions and Directions for Future Research

We applied the CE method for continuous multi-extremal unconstrained and constrained optimization problems with linear and nonlinear constraints. We demonstrated its high accuracy and gave an example where CE finds a better solution than reported in the literature. We have focused on two different updating schemes — called *fixed* and *dynamic* smoothing — for the parameters μ and σ^2 of the normal pdf. Fixed smoothing is typically significantly faster than dynamic smoothing, and should be attempted first. We have showed that dynamic smoothing consistently gives accurate answers, in cases where the fixed smoothing approach fails. We have deliberately refrained from “tuning” the parameters. Undoubtedly this will further improve the accuracy and speed of the algorithm, but by using the *same* CE parameters for all problems, and by employing the (in essence) same simple Matlab implementation throughout, we emphasized the robustness, elegance and versatility of the CE method.

We believe that the CE method for continuous optimization is especially useful for tackling problems with complicated constraints. We have focused on two approaches: the acceptance-rejection and the proportional penalty approach for the constrained optimization. Both methods have their merits, although the constant penalty approach is more generally applicable.

The theoretical convergence properties of the CE method are not yet fully understood. A few results on this can be found in Rubinstein and Kroese (2004) and Margolin (2004). Although this will be an important topic for future research, we believe that from a practical point of view the merit of the CE method has been demonstrated clearly by the increasing body of numerical evidence. However, the CE method is still evolving, and many new modifications and applications are being developed. An extensive numerical study on the various CE modifications and the choice of parameters, for a large range of test problems, is another topic for future research.

A Matlab Example Code

An example Matlab CE program, to find the global maximum of the function S in (9).

```
S = inline('exp(-(x-2).^2) + 0.8*exp(-(x+2).^2)');
mu = -6;
sigma = 10;
Nel = 10;
N = 100;
eps = 1E-8;
tic
t=0;
while sigma > eps
    t = t+1;
    x = mu + sigma*randn(N,1);
    SX = S(x);
    sortSX = sortrows([x SX],2);
    Xel = sortSX((N - Nel + 1):N,1);
    mu = mean(Xel);
    sigma = std(Xel);
    fprintf('%g %6.9f %6.9f %6.9f \n', t, S(mu),mu, sigma)
end
mu
toc
```

B Main CE Program

All the Matlab programs used for this paper we based on the program below.

```
clear all
format long g
n=10; % select dimension
Nel = 10, N = 100*n, alpha = 0.8 , beta = 0.7, q = 5;
eps = 1e-3;
mu = -2 + 4*rand(1,n); % select initial mu
sigma = 100.0*ones(1,n); % select initial sigma
```

```

mu_last = mu;
sigma_last = sigma;
X_best_overall = zeros(1,n);
S_best_overall = 1E10;
t = 0
tic
while sigma > eps
    t = t+1;
    mu = alpha*mu + (1-alpha)*mu_last;
    B_mod = beta - beta*(1-1/t)^q;
    sigma = B_mod*sigma + (1-B_mod)*sigma_last ; % dynamic smoothing
% sigma= alpha*sigma + (1-alpha)*sigma_last; % fixed smoothing
    X = ones(N,1)*mu + randn(N,n)*diag(sigma); % generate samples
    SA = Rosen(X); % select performance function
    [S_sort,I_sort] = sort(SA);
    gam = S_sort(Nel);
    S_best = S_sort(1);
    if (S_best < S_best_overall)
        S_best_overall = S_best;
        X_best_overall = X(I_sort(1),:);
    end
    mu_last = mu;
    sigma_last = sigma;
    Xel = X(I_sort(1:Nel),:);
    mu = mean(Xel);
    sigma = std(Xel);
    if mod(t,100)==0 % print each 100 iterations
        fprintf('%d %5.4f',t,S_best);
        fprintf(' %5.4f',mu)
        fprintf('\n')
    end;
end
toc
fprintf('%d %9.8f\n',t,S_best_overall);

```

```
fprintf(' %9.8f',X_best_overall)
fprintf('\n')
```

C Functions

Below we list three test functions (the trigonometric, the Rosenbrock, and the Hougen least squares function) that can be used in conjunction with the main CE program above.

```
function out = Trigo(X) %trigonometric function
r = [];
for i = 1:size(X,2) ,
r = [8*sin(7*(X(:,i)- 0.9).^2).^2 + ...
      + 6*sin(14*(X(:,i)- 0.9).^2).^2 + (X(:,i) - 0.9).^2, r];
end;
out = sum(r,2);

function out = Rosen(X)
r=[];
for i = 1:size(X,2)-1
r = [100*(X(:,i+1)-X(:,i).^2).^2+(X(:,i)-1).^2,r];
end
out = sum(r,2);

function out = Hougen1(X)
r = [8.55, 3.79, 4.82, 0.02, 2.75, 14.39, 2.54, 4.35, 13.00, 8.50, ...
      0.05, 11.32, 3.13];
z = [470, 300, 10;
      285, 80, 10;
      470, 300,120;
      470, 80,120;
      470, 80, 10;
      100, 190, 10;
      100, 80, 65;
```

```

470, 190, 65;
100, 300, 54;
100, 300,120;
100, 80,120;
285, 300, 10;
285, 190,120 ]';
s = 0;
for i = 1:13
    s = s + (r(i) - ((X(:,1)*z(2,i) - z(3,i)./X(:,5) )./(1 + X(:,2)*z(1,i) ...
        + X(:,3)*z(2,i)+ X(:,4)*z(3,i))))).^2 ;
end
out = (s + 10*sum(max(-X(:, :),0),2) + 10*sum(max(X(:, :)-2,0),2))/13;

```

Acknowledgment: We would like to thank Jenny Liu and Thomas Taimre for conducting various CE experiments on constrained optimization. This Research was supported by the Israel Science Foundation (ISF) Grant No. 191-565.

References

- Aarts, E. H. L. and Korst, J. H. M. (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley & Sons.
- Bates, D. and Watts, D. (1988). *Nonlinear Regression Analysis and Its Applications*. Wiley.
- Bracken, J. and McCormick, G. P. (1968). *Selected Applications of Nonlinear Programming*. John Wiley & Sons, Inc., New York.
- de Boer, P. T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. (2004). A tutorial on the cross-entropy method. *Annals of Operations Research*. To appear.
- Dorigo, M., Maniezzo, V., and Colorni, A. (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, **26**(1), 29–41.

- Dueck, G. and Scheur, T. (1990). Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, **90**.
- Glover, F. and Laguna, M. L. (1993). *Modern Heuristic Techniques for Combinatorial Optimization*, chapter Chapter 3: Tabu search. Blackwell Scientific Publications.
- Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley.
- Gong, W. B., Ho, Y. C., and Zhai, W. (1992). Stochastic comparison algorithm for discrete optimization with estimation. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 795–800.
- Himmelblau, D. M. (1972). *Applied Nonlinear Programming*. McGrawHill Book-Comapany, New York.
- Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*, volume 197. Springer-Verlag, New York.
- Margolin, L. (2004). On the convergence of the cross-entropy method. *Annals of Operations Research*.
- Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 3rd edition.
- Paviani, D. A. (1969). *A New Method for the Solution of the General Nonlinear Programming Problem*. Ph.D. thesis, The University of Texas, Austin, Texas.
- Rubinstein, R. Y. (1997). Optimization of computer simulation models with rare events. *European Journal of Operational Research*, **99**, 89–112.
- Rubinstein, R. Y. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, **2**, 127–190.
- Rubinstein, R. Y. and Kroese, D. P. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning*. Springer-Verlag, New York.

Schittkowski, K. (1980). *Nonlinear Programming Codes*, volume 183. Springer-Verlag, New York.

Sheela, B. V. and Ramaoorthy, P. (1975). Swift - a new constrained optimization technique. *Computer Methods in Applied Mechanics and Engineering*, **6**(3), 309–318.

White, D. (1992). *Markov Decision Process*. J. Wiley & Sons.