Università
degli Studi di Urbino
"Carlo Bo"

Facoltà
di Economia
Istituto di Scienze
Economiche

WP-EMS
Working Papers Series in Economics,
Mathematics & Statistics

# "Differential Evolution Methods for the Fuzzy Extension of Functions"

- Luciano Stefanini (U. Urbino)

WP-EMS # 2007/05

# Differential Evolution Methods for the Fuzzy Extension of Functions

Luciano Stefanini

University of Urbino "Carlo Bo", ITALY
lucste@uniurb.it

**Abstract.** The paper illustrates a differential evolution (DE) algorithm to calculate the level-cuts of the fuzzy extension of a multidimensional real valued function to fuzzy numbers. The method decomposes the fuzzy extension engine into a set of "nested" min and max box-constrained optimization problems and uses a form of the DE algorithm, based on multi populations which cooperate during the search phase and specialize, a part of the populations to find the the global min (corresponding to lower branch of the fuzzy extension) and a part of the populations to find the global max (corresponding to the upper branch), both gaining efficienty from the work done for a level-cut to the subsequent ones. A special version of the algorithm is designed to the case of differentiable functions, for which a representation of the fuzzy numbers is used to improve efficiency and quality of calculations. The included computational results indicate that the DE method is a promising tool as its computational complexity grows on average superlinearly (of degree less than 1.5) in the number of variables of the function to be extended.

## 1 Introduction

Appropriate use of fuzzy numbers in applications requires at least two features to be satisfied: (1) an easy way to represent and model fuzzy information with a high flexibility of shapes, e.g. allowing asymmetries or nonlinearities; (2) a relative simplicity and computational efficiency to perform exact fuzzy calculations or to obtain good approximations of the results. In general, the arithmetic operations on fuzzy numbers can be approached either by the direct use of the membership function (by the Zadeh extension principle) or by the equivalent use of the $\alpha - cuts$ representation. By the $\alpha - cuts$ approach, it is possible to define a parametric representation of fuzzy numbers with the advantage of obtaining a wide family of fuzzy numbers (see [12]). It is well known that the fuzzy extension principle requires to solve a set of optimization problems and different heuristic methods have been proposed to obtain good solutions with a small number of function evaluations. Well known fundamental algorithms are the vertex method and its modifications (see [15] and [10]); the transformation method (see [6]) in its general or reduced versions (see [8] for an efficient implementation); a sparse grids method (see [9]). We suggest here two procedures based on the differential evolution ($DE$) method of Storn and Price (see [13], [14], [11]) and adapted to

take into account both the nested property of $\alpha - cuts$ and the min and max problems over the same domains. In particular, we use simultaneous multiple populations that collaborate each other and specialize during the process to find all the required solutions. Computational results are reported that indicate the DE method as a promising tool, as it exhibits, on average, superlinear computational complexity (of degree less than 1.5) in the number of variables.

## 2 Fuzzy numbers and fuzzy extension principle

We will consider fuzzy numbers and intervals, i.e. fuzzy sets defined over the field $\mathbb{R}$ of real numbers having a particular form. A general *fuzzy set* over $\mathbb{R}$ is usually defined by its membership function $\mu : \mathbb{R} \longrightarrow \mathbb{T} \subseteq [0,1]$ and a fuzzy (sub)set $u$ of $\mathbb{R}$ is uniquely characterized by the pairs $(x, \mu_u(x))$ for each $x \in \mathbb{R}$; the value $\mu_u(x) \in [0,1]$ is the membership grade of $x$ to the fuzzy set $u$. Denote by $\mathcal{F}(\mathbb{R})$ the collection of the fuzzy sets over $\mathbb{R}$. Elements of $\mathcal{F}(\mathbb{R})$ will be denoted by letters $u$, $v$, $w$ and the corresponding membership functions by $\mu_u$, $\mu_v$, $\mu_w$.

Fundamental concepts in fuzzy theory are the *support*, the *level-sets* (or *level-cuts*) and the *core* of a fuzzy set:

**Definition 1.** *Let $\mu_u$ be the membership function of a fuzzy set $u$ over $\mathbb{R}$. The support of $u$ is the (crisp) subset of points of $\mathbb{R}$ at which the membership grade $\mu_u(x)$ is positive: $supp(u) = \{x | x \in \mathbb{R},\ \mu_u(x) > 0\}$. For $\alpha \in ]0,1]$, the $\alpha-$level cut of $u$ (or simply the $\alpha - cut$) is defined by $[u]_\alpha = \{x | x \in \mathbb{R},\ \mu_u(x) \geq \alpha\}$ and for $\alpha = 0$ by the closure of the support $[u]_0 = cl\{x | x \in \mathbb{R},\ \mu_u(x) > 0\}$. The core of $u$ is the set of elements of $\mathbb{R}$ having membership grade 1, i.e. $core(u) = \{x | x \in \mathbb{R},\ \mu_u(x) = 1\}$ and we say that $u$ is normal if $core(u) \neq \emptyset$.*

It is well-known that the $level - cuts$ are "nested", i.e. $[u]_\alpha \subseteq [u]_\beta$ for $\alpha > \beta$.

A particular class of fuzzy sets $u \in \mathcal{F}(\mathbb{R})$ is when the support is a convex set ($A$ is said *convex* if $(1-t)x' + tx'' \in A$ for every $x', x'' \in A$ and all $t \in [0,1]$) and the membership function is quasi-concave, i.e. $supp(u)$ is convex and $\mu_u((1-t)x' + tx'') \geq \min\{\mu_u(x'), \mu_u(x'')\}$ for every $x', x'' \in supp(u)$ and $t \in [0,1]$. Equivalently, $\mu_u$ is quasi-concave if the level sets $[u]_\alpha$ are convex for all $\alpha \in [0,1]$.

Finally, if the membership function is upper semi-continuous, then the level-cuts are closed.

**Definition 2.** *A fuzzy set $u$ is a fuzzy quantity if the $\alpha - cuts$ are nonempty, compact intervals of the form $[u]_\alpha = [u_\alpha^-, u_\alpha^+] \subset \mathbb{R}$. If $\exists \widehat{u} \in \mathbb{R}$ such that $core(u) = \{\widehat{u}\}$, $u$ is a fuzzy number and $u$ is called a fuzzy interval if $\exists \widehat{u}^-, \widehat{u}^+ \in \mathbb{R}$, $\widehat{u}^- < \widehat{u}^+$ such that $core(u) = [\widehat{u}^-, \widehat{u}^+]$.*

The "nested" property is the basis for the LU representation (L for lower, U for upper). We denote by $\mathbb{F}$ the set of fuzzy quantities.

**Definition 3.** *An LU-fuzzy quantity (number or interval) $u$ is completely determined by any pair $u = (u^-, u^+)$ of functions $u^-, u^+ : [0,1] \longrightarrow \mathbb{R}$, defining the end-points of the $\alpha-cuts$, satisfying the three conditions:(i) $u^- : \alpha \longrightarrow u_\alpha^- \in \mathbb{R}$ is a bounded monotonic nondecreasing left-continuous function $\forall \alpha \in ]0,1]$ and right-continuous for $\alpha = 0$;(ii) $u^+ : \alpha \longrightarrow u_\alpha^+ \in \mathbb{R}$ is a bounded monotonic non-increasing left-continuous function $\forall \alpha \in ]0,1]$ and right-continuous for $\alpha = 0$;(iii) $u_\alpha^- \leq u_\alpha^+ \ \forall \alpha \in [0,1]$.*

The support of $u$ is the interval $[u_0^-, u_0^+]$ and the core is $[u_1^-, u_1^+]$. We refer to the functions $u_{(.)}^-$ and $u_{(.)}^+$ as the lower and upper branches on $u$, respectively. If the two branches $u_{(.)}^-$ and $u_{(.)}^+$ are continuous invertible functions then $\mu_u(.)$ is formed by two continuous branches, the left being the increasing inverse of $u_{(.)}^-$ on $[u_0^-, u_1^-]$ and, the right, the decreasing inverse of $u_{(.)}^+$ on $[u_1^+, u_0^+]$.

There are many choices for $u_{(.)}^-$ and $u_{(.)}^+$. If we start with two decreasing shape functions $p(.)$ and $q(.)$ and with four numbers $u_0^- \leq u_1^- \leq u_1^+ \leq u_0^+$ defining the support and the core of $u$ then we can model $u_{(.)}^-$ and $u_{(.)}^+$ by $u_\alpha^- = u_1^- - (u_1^- - u_0^-)p(\alpha)$ and $u_\alpha^+ = u_1^+ - (u_1^+ - u_0^+)q(\alpha)$ for all $\alpha \in [0,1]$. The simplest fuzzy quantities have linear branches: a trapezoidal fuzzy interval, denoted by $u = \langle a, b, c, d \rangle$, where $a \leq b \leq c \leq d$, has $\alpha-cuts\ [u]_\alpha = [a + \alpha(b-a), d - \alpha(d-c)]$, $\alpha \in [0,1]$, obtaining a triangular fuzzy number if $b = c$.

Consider now the extension of function $f : \mathbb{R}^n \to \mathbb{R}$ to a vector of $n$ fuzzy numbers $u = (u_1, u_2, ..., u_n) \in (\mathbb{F})^n$, with $k-th$ component $u_k \in \mathbb{F}$ given by $[u_k]_\alpha = [u_{k,\alpha}^-, u_{k,\alpha}^+]$ for $k = 1, 2, ..., n$ or $\mu_{u_k} : supp(u_k) \longrightarrow [0,1]$ for $k = 1, 2, ..., n$ and denote $v = f(u_1, u_2, ..., u_n)$.

The extension principle introduced by Zadeh in [16] is the basic tool for fuzzy calculus; it states that $\mu_v$ is given by

$$\mu_v(y) = \begin{cases} \sup\{\min\{\mu_{u_1}(x_1), ..., \mu_{u_n}(x_x)\} | y = f(x_1, ..., x_n)\} & \text{if } y \in Range(f) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $Range(f) = \{y \in \mathbb{R} | \exists (x_1, ..., x_n) \in \mathbb{R}^n \text{ s.t. } y = f(x_1, ..., x_n)\}$.

For a continuous function $f : \mathbb{R}^n \to \mathbb{R}$, the $\alpha-cuts$ of the fuzzy extension $v$ are obtained by solving the following box-constrained global optimization problems ($\alpha \in [0,1]$)

$$v_\alpha^- = \min \{f(x_1, x_2, ..., x_n) | x_k \in [u_k]_\alpha,\ k = 1, 2, ..., n\} \tag{2}$$

$$v_\alpha^+ = \max \{f(x_1, x_2, ..., x_n) | x_k \in [u_k]_\alpha,\ k = 1, 2, ..., n\}. \tag{3}$$

If the function $f(x_1, x_2, ..., x_n)$ is sufficiently simple, the analytical expressions for $v_\alpha^-$ and $v_\alpha^+$ can be obtained, as it is the case for many unidimensional elementary functions.

For general functions, we need to solve numerically the global optimization problems (2) and (3) above; general methods have been proposed and a very extended scientific literature is available. It is clear that in these cases we have only the possibility of fixing a finite set of values $\alpha \in \{\alpha_0, ..., \alpha_M\}$ and obtain

the corresponding $v_\alpha^-$ and $v_\alpha^+$ pointwise; a sufficiently precise calculation requires $M$ in the range from 10 to 100 or more (depending on the application and the required precision) and the computational time may become very high. To reduce these difficulties, various specific heuristic methods have been proposed; among others, the vertex method and its variants (see [3], [1] and [10]), the transformation method (see [6], [7], [8]), the interval arithmetic optimization with sparse grids (see [9]).

All the specific methods try to take computational advantage from the specific structure of "nested" optimizations (2)-(3) intrinsic in the properties of the $\alpha-cuts$. We will see that, at least in the differentiable case, the advantages of the LU representation appear to be quite interesting, based on the fact that a small number of $\alpha$ points is in general sufficient to obtain good approximations (this is the essential gain in using the slopes to model fuzzy numbers), so reducing the number of constrained *min* (2) and *max* (3) problems to be solved directly. On the other hand, finding computationally efficient extension solvers is still an open research field in fuzzy calculations.

## 3 Representation of LU-fuzzy numbers

As we have seen in the previous section, the LU representations of fuzzy numbers require to use appropriate (monotonic) shape functions to model the lower and upper branches of the $\alpha - cuts$. In this section we present the basic elements of a parametric representation of the shape functions proposed in [5] and [12] based on monotonic Hermite-type interpolation. The parametric representations can be used both to define the shape functions and to calculate the arithmetic operations by error controlled approximations.

We first introduce some models for "standardized" differentiable monotonic shape functions $p : [0,1] \longrightarrow [0,1]$ such that $p(0) = 0$ and $p(1) = 1$ with $p(t)$ increasing on $[0,1]$; if interested to decreasing functions, we can start with an increasing function $p(.)$ and simply define corresponding decreasing functions $q : [0,1] \longrightarrow [0,1]$ by $q(t) = 1 - p(t)$ or $q(t) = p(\varphi(t))$ where $\varphi : [0,1] \longrightarrow [0,1]$ is any decreasing bijection (e.g. $\varphi(t) = 1 - t$).

Valid shape functions can be obtained by $p : [0,1] \longrightarrow [0,1]$, satisfying the four Hermite interpolation conditions $p(0) = 0$, $p(1) = 1$ and $p'(0) = \beta_0$, $p'(1) = \beta_1$ for any value of the two nonnegative parameters $\beta_i \geq 0$, $i = 0, 1$.

To explicit the parameters, we denote the interpolating function by $t \longrightarrow p(t; \beta_0, \beta_1)$ for $t \in [0,1]$.

We recall here two of the basic forms illustrated in [12]:

∘ (2,2)-rational spline: $p(t; \beta_0, \beta_1) = \frac{t^2 + \beta_0 t(1-t)}{1 + (\beta_0 + \beta_1 - 2)t(1-t)}$;

∘ mixed exponential spline: $p(t; \beta_0, \beta_1) = \frac{1}{a}[t^2(3-2t) + \beta_0 - \beta_0(1-t)^a + \beta_1 t^a]$ where $a = 1 + \beta_0 + \beta_1$.

Note that in both forms we obtain a linear $p(t) = t$, $\forall t \in [0,1]$ if $\beta_0 = \beta_1 = 1$ and a quadratic $p(t) = t^2 + \beta_0 t(1-t)$ if $\beta_0 + \beta_1 = 2$.

In order to produce different shapes we can either fix the slopes $\beta_0$ and $\beta_1$ (if we have information on the first derivatives at $t = 0$, $t = 1$) or we can estimate them by knowing values of $p(t)$ in additional points.

The model functions above can be adopted to represent the functions "piecewise", on a decomposition of the interval $[0, 1]$ into $N$ subintervals $0 = \alpha_0 < \alpha_1 < ..... < \alpha_{i-1} < \alpha_i < .... < \alpha_N = 1$. It is convenient to use the same subdivision for both the lower $u_\alpha^-$ and upper $u_\alpha^+$ branches (we can always reduce to this situation by the union of two different subdivisions). In each subinterval $I_i = [\alpha_{i-1}, \alpha_i]$, the values and the slopes of the two functions are

$$\begin{array}{l} u^-_{(\alpha_{i-1})} = u^-_{0,i} \ \ , \ \ u^+_{(\alpha_{i-1})} = u^+_{0,i} \ \ , \ \ u^-_{(\alpha_i)} = u^-_{1,i} \ \ , \ \ u^+_{(\alpha_i)} = u^+_{1,i} \\ u'^-_{(\alpha_{i-1})} = d^-_{0,i} \ \ , \ \ u'^+_{(\alpha_{i-1})} = d^+_{0,i} \ \ , \ \ u'^-_{(\alpha_i)} = d^-_{1,i} \ \ , \ \ u'^+_{(\alpha_i)} = d^+_{1,i} \end{array} \tag{4}$$

and by the transformation $t_\alpha = \frac{\alpha - \alpha_{i-1}}{\alpha_i - \alpha_{i-1}}$, $\alpha \in I_i$, each subinterval $I_i$ is mapped into the standard $[0, 1]$ interval to determine each piece independently. Globally continuous or more regular $C^{(1)}$ fuzzy numbers can be obtained directly from the data (for example, $u^-_{1,i} = u^-_{0,i+1}$, $u^+_{1,i} = u^+_{0,i+1}$ for continuity and $d^-_{1,i} = d^-_{0,i+1}$, $d^+_{1,i} = d^+_{0,i+1}$ for differentiability at $\alpha = \alpha_i$).

Let $p_i^{\pm}(t)$ denote the model function on $I_i$; we obtain $p_i^-(t) = p(t; \beta^-_{0,i}, \beta^-_{1,i})$, $p_i^+(t) = 1 - p(t; \beta^+_{0,i}, \beta^+_{1,i})$ with $\beta^-_{j,i} = \frac{\alpha_i - \alpha_{i-1}}{u^-_{1,i} - u^-_{0,i}} d^-_{j,i}$ and $\beta^+_{j,i} = -\frac{\alpha_i - \alpha_{i-1}}{u^+_{1,i} - u^+_{0,i}} d^+_{j,i}$ for $j = 0, 1$ so that, for $\alpha \in [\alpha_{i-1}, \alpha_i]$ and $i = 1, 2., , , N$:

$$u^-_\alpha = u^-_{0,i} + (u^-_{1,i} - u^-_{0,i}) p_i^-(t_\alpha) \ , \ u^+_\alpha = u^+_{0,i} + (u^+_{1,i} - u^+_{0,i}) p_i^+(t_\alpha) \, . \tag{5}$$

The illustrated monotonic models suggest a first parametrization of fuzzy numbers on the trivial decomposition of interval $[0, 1]$, with $N = 1$ (without internal points) and $\alpha_0 = 0, \alpha_1 = 1$. In this simple case, $u$ can be represented by a vector of 8 components (the slopes corresponding to $u_i^-$ are denoted by $\delta u_i^-$ , etc)

$$u = (u_0^-, \delta u_0^-, u_0^+, \delta u_0^+; u_1^-, \delta u_1^-, u_1^+, \delta u_1^+) \tag{6}$$

with $u_0^-, \delta u_0^-, u_1^-, \delta u_1^-$ for the lower branch $u_\alpha^-$ and $u_0^+, \delta u_0^+, u_1^+, \delta u_1^+$ for the upper branch $u_\alpha^+$.

On a decomposition $0 = \alpha_0 < \alpha_1 < ... < \alpha_N = 1$ we can proceed piecewise. For example, a differentiable shape function requires $4(N + 1)$ parameters

$$\begin{array}{r} u = (\alpha_i; u_i^-, \delta u_i^-, u_i^+, \delta u_i^+)_{i=0,1,...,N} \text{ with} \\ u_0^- \le u_1^- \le ... \le u_N^- \le u_N^+ \le u_{N-1}^+ \le ... \le u_0^+ \text{ (data)} \\ \delta u_i^- \ge 0, \delta u_i^+ \le 0 \text{ (slopes).} \end{array} \tag{7}$$

and the branches are computed according to (5). In [5] and [12] we have analyzed the advantages of the LU representation in the computation of fuzzy expressions.

## 4 Differential Evolution algorithms for fuzzy arithmetic

In this section we adopt an algorithmic approach to describe the application of differential evolution methods to calculate the fuzzy extension of multivariable function, associated to the LU representation of the fuzzy quantities involved.

Let $v = f(u_1, u_2, ..., u_n)$ denote the fuzzy extension of a continuous function $f$ in $n$ variables; it is well known that the fuzzy extension of $f$ to normal upper semicontinuous fuzzy intervals (with compact support) has the level-cutting commutative property (see [4]), i.e. the $\alpha-cuts$ $v_\alpha = [v_\alpha^-, v_\alpha^+]$ of $v$ are the images of the $\alpha-cuts$ of $(u_1, u_2, ..., u_n)$ and are obtained by solving the box-constrained optimization problems

$$(EP)_\alpha : \begin{cases} v_\alpha^- = \min\left\{ f(x_1, x_2, ..., x_n) | x_k \in [u_{k,\alpha}^-, u_{k,\alpha}^+],\ k = 1, 2, ..., n \right\} \\ v_\alpha^+ = \max\left\{ f(x_1, x_2, ..., x_n) | x_k \in [u_{k,\alpha}^-, u_{k,\alpha}^+],\ k = 1, 2, ..., n \right\}. \end{cases} \tag{8}$$

For simplicity, we will illustrate the case of differentiable representations (7) and differentiable function $f$.

Let $u_k = (u_{k,i}^-, \delta u_{k,i}^-, u_{k,i}^+, \delta u_{k,i}^+)_{i=0,1,...,N}$ be the LU-fuzzy representations of the input quantities for $k = 1, 2, ..., n$ and $v = (v_i^-, \delta v_i^-, v_i^+, \delta v_i^+)_{i=0,1,...,N}$ ; the $\alpha-cuts$ of $v$ are obtained by solving the box-constrained optimization problems (8).

For each $\alpha = \alpha_i$, $i = 0, 1, ..., N$ the min and the max (8) can occur either at a point whose components $x_{k,i}$ are internal to the corresponding intervals $[u_{k,i}^-, u_{k,i}^+]$ or are coincident with one of the extremal values; denote by $\widehat{x}_i^- = (\widehat{x}_{1,i}^-, ..., \widehat{x}_{n,i}^-)$ and $\widehat{x}_i^+ = (\widehat{x}_{1,i}^+, ..., \widehat{x}_{n,i}^+)$ the points where the min and the max take place; then $v_i^- = f(\widehat{x}_{1,i}^-, \widehat{x}_{2,i}^-, ..., \widehat{x}_{n,i}^-)$ and $v_i^+ = f(\widehat{x}_{1,i}^+, \widehat{x}_{2,i}^+, ..., \widehat{x}_{n,i}^+)$ and the slopes $\delta v_i^-$, $\delta v_i^+$ are computed (as $f$ is differentiable) by

$$\delta v_i^- = \sum_{\substack{k=1 \\ \widehat{x}_{k,i}^- = u_{k,i}^-}}^{n} \frac{\partial f(\widehat{x}_{1,i}^-, ..., \widehat{x}_{n,i}^-)}{\partial x_k} \delta u_{k,i}^- + \sum_{\substack{k=1 \\ \widehat{x}_{k,i}^- = u_{k,i}^+}}^{n} \frac{\partial f(\widehat{x}_{1,i}^-, ..., \widehat{x}_{n,i}^-)}{\partial x_k} \delta u_{k,i}^+ \tag{9}$$

$$\delta v_i^+ = \sum_{\substack{k=1 \\ \widehat{x}_{k,i}^+ = u_{k,i}^-}}^{n} \frac{\partial f(\widehat{x}_{1,i}^+, ..., \widehat{x}_{n,i}^+)}{\partial x_k} \delta u_{k,i}^- + \sum_{\substack{k=1 \\ \widehat{x}_{k,i}^+ = u_{k,i}^+}}^{n} \frac{\partial f(\widehat{x}_{1,i}^+, ..., \widehat{x}_{n,i}^+)}{\partial x_k} \delta u_{k,i}^+. \tag{10}$$

If, for some reasons, the partial derivatives of $f$ at the solution points are not available we can produce an estimation of the shapes $\delta v_i^-$ and $\delta v_i^+$.

The idea of $DE$ to find $min$ or $max$ of $\{f(x_1, ..., x_n) | (x_1, ..., x_n) \in \mathbb{A} \subset \mathbb{R}^n\}$ is simple (see [13] and [14],[2] for constraints handling).

DE is a (parallel) direct search algorithm wich utilizes a "population" $x^{(1)} = (x_1, ..., x_n)^{(1)}, ..., x^{(NP)} = (x_1, ..., x_n)^{(NP)} \in \mathbb{A}$ of $NP$ feasible points for each generation (i.e. for each iteration) to obtain a new set of points by recombining randomly the individuals of the current population and by selecting the best

generated elements to continue in the next generation. The initial population is chosen randomly and should try to cover uniformly the entire parameter space.

Denote by $x^{(p,g)}$ the $p-$th vector if the population at iteration (generation) $g$ and by $x_j^{(p,g)}$ its $j-$th component ($j = 1, ..., n$).

At each iteration, the method generates a set of candidate points $y^{(p,g)}$ to substitute the elements $x^{(p,g)}$ of the current population, if $y^{(p,g)}$ is better.

To generate $y^{(p,g)}$ two operations are applied: recombination and crossover.

A typical recombination operates on a single component $j \in \{1, ..., n\}$ and generates a new perturbed vector of the form $v_j^{(p,g)} = x_j^{(r,g)} + \gamma[x_j^{(s,g)} - x_j^{(t,g)}]$, where $r, s, t \in \{1, 2, ..., p\}$ are chosen randomly and $\gamma \in ]0, 2]$ is a constant (eventually chosen randomly for the curent iteration) that controls the amplification of the variation.

The potential diversity of the population is controlled by a crossover operator, that construct the candidate $y^{(p,g)}$ by crossing randomly the components of the perturbed vector $v_j^{(p,g)}$ and the old vector $x_j^{(p,g)}$:

$$y_j^{(p,g)} = \begin{cases} v_j^{(p,g)} & \text{if } \ j \in \{j_1, j_2, ..., j_k\} \\ x_j^{(p,g)} & \text{if } \ j \notin \{j_1, j_2, ..., j_k\} \end{cases}$$

with $k$ random between 1 and $n$ and 0 with a crossover probability $q$ and $j_1, j_2, ..., j_k$ being random components if $k$ is not 0.

So, the components of each individual of the current population are modified to $y_j^{(p,g)}$ by a given probability $q$.

Typical values are $\gamma \in [0.2, 0.95]$, $q \in [0.7, 1.0]$ and $NP \geq 5n$ (the higher $NP$, the lower $\gamma$).

The candidate $y^{(p,g)}$ is then compared to the existing $x^{(p,g)}$ by evaluating the objective function at $y^{(p,g)}$ : if $f(y^{(p,g)})$ is better than $f(x^{(p,g)})$ then $y^{(p,g)}$ substitutes $x^{(p,g)}$ in the new generation $g + 1$, otherwise $x^{(p,g)}$ is retained.

Many variants of the recombination schemes have been proposed and some seem to be more effective than others. Examples are:

DE/rand/1: $v_j^{(p,g)} = x_j^{(r,g)} + \gamma[x_j^{(s,g)} - x_j^{(t,g)}]$;

DE/best/1: $v_j^{(p,g)} = x_j^{(best,g)} + \gamma[x_j^{(s,g)} - x_j^{(t,g)}]$ where $x^{(best,g)}$ is the current best solution;

DE/rand-best/2: $v_j^{(p,g)} = x_j^{(p,g)} + \gamma[x_j^{(best,g)} - x_j^{(p,g)} + x_j^{(s,g)} - x_j^{(t,g)}]$;

DE/best/2: $v_j^{(p,g)} = x_j^{(best,g)} + \gamma[x_j^{(s_1,g)} + x_j^{(s_2,g)} - x_j^{(t_1,g)} - x_j^{(t_2,g)}]$, $s_1, t_1, s_2, t_2 \in \{1, 2, ..., p\}$ are random;

DE/rand/2: $v_j^{(p,g)} = x_j^{(r,g)} + \gamma[x_j^{(s_1,g)} + x_j^{(s_2,g)} - x_j^{(t_1,g)} - x_j^{(t_2,g)}]$,

To take into account the particular nature of our problem, we modify the basic procedure and examine two different strategies:

1. SPDE (Single Population DE Procedure): start with the $(\alpha = 1) - cut$ back to the $(\alpha = 0) - cut$ so that the optimal solutions at a given level can be inserted into the "starting" populations of lower levels; use two distinct populations and

perform the recombinations such that, during generations, one of the populations specializes to find the minimum and the other to find the maximum.

2. MPDE (Multiple Populations DE Procedure): use $2(N+1)$ populations to solve simultaneously all the box-constrained problems (8); $N+1$ populations specialize for the min and the others for the max and the current best solution for level $\alpha_i$ is valid also for levels $\alpha_0, ..., \alpha_{i-1}$.

Let $[u^-_{j,i}, u^+_{j,i}]$, $j = 1, 2, ..., n$ and $f : \mathbb{R}^n \to \mathbb{R}$ be given; we have to find $v^-_i$ and $v^+_i$ according to (8) for $i = 0, 1, ..., N$. The slope parameters $\delta v^-_i$, $\delta v^+_i$ are computed by (9) and (10).

To handle the feasibility constraints we have to ensure either that at each generation we have $u^-_{j,i} \leq y^{(p,g)}_j \leq u^+_{j,i}$ or at least that the final solutions $\widehat{x}^-_i = (\widehat{x}^-_{1,i}, ..., \widehat{x}^-_{n,i})$ and $\widehat{x}^+_i = (\widehat{x}^+_{1,i}, ..., \widehat{x}^+_{n,i})$ are feasible.

An efficient constraint handling method has been examined in [2], where a tournament selection is proposed, based on three criteria:

(1) any feasible solution is preferred to any infeasible solution;

(2) among two feasible solutions, the one with better objective value is preferred;

(3) among two infeasible solutions, the one having smaller constraints violation is preferred.

In our case, as we have simple box-constraints, it is easy to produce feasible starting populations, as we have to generate random numbers $x^{(p,0)}_j$ between the lower $u^-_{j,i}$ and the upper $u^+_{j,i}$ values.

During the iterations, we use a variant of the method above, where the $y^{(p,g)}$ are progressively forced to be feasible or with small infeasibilities and a penalty is assigned to infeasible values:

*(i)* modify $y^{(p,g)}_j$ to fit $[u^-_{j,i} - \frac{\varepsilon}{g^2}, u^+_{j,i} + \frac{\varepsilon}{g^2}]$, $j = 1, 2, ..., n$ with small $\varepsilon \sim 10^{-2}(u^+_{j,i} - u^-_{j,i})$, so that the eventual infeasibilities decrease rapidly during the generation process;

*(ii)* if the candidate point $y^{(p,g)}$ is infeasible and has a value $f(y^{(p,g)})$ better than the current best feasible value $f(x^{(best,g)})$ then a penalty is added and the value of $y^{(p,g)}$ is elevated to $f(x^{(best,g)}) + \varepsilon'$ (for the min problems) or reduced to $f(x^{(best,g)}) - \varepsilon'$ (for the max problem), being $\varepsilon' \sim 10^{-3}$ a small positive number.

The first strategy SPDE is implemented in algorithm 1. Function $ran(0, 1)$ generates a random uniform number in [0,1].

**Algorithm 1:** (Single Population DE procedure: SPDE with DE/rand/1).
Choose $p \approx 10n$, $g_{\max} \approx 500$, $q$ and $\gamma$.
select $(x_1^{(l)}, ..., x_n^{(l)})$, $x_k^{(l)} \in [u_{k,N}^-, u_{k,N}^+]$ $\forall k$, $l = 1, ..., 2p$ (initial population)
evaluate $y^{(l)} = f(x_1^{(l)}, ..., x_n^{(l)})$
for $i = N, N-1, ..., 0$
 for $g = 1, 2, ..., g_{\max}$ (up to $g_{\max}$ generations or other stopping rule)
  for $l = 1, 2, ..., 2p$
   select (randomly) $r, s, t \in \{1, 2, ..., 2p\}$ and $j^* \in \{1, 2, ..., n\}$
   for $j = 1, 2, ..., n$
    if $(j = j^*$ or $ran(0,1) < q)$ then $x'_j = x_j^{(r)} + \gamma[x_j^{(s)} - x_j^{(t)}]$ else $x'_j = x_j^{(l)}$
    ensure that $u_{j,i}^- \le x'_j \le u_{j,i}^+$ (or other feasibility handling)
   end
   evaluate $y = f(x'_1, ..., x'_n)$
   if $l \le p$ and $y < y^{(l)}$ then substitute $(x_1, ..., x_n)^{(l)}$ with $(x'_1, ..., x'_n)$ (min)
   if $l > p$ and $y > y^{(l)}$ then substitute $(x_1, ..., x_n)^{(l)}$ with $(x'_1, ..., x'_n)$ (max)
  end
 end
$v_i^- = y^{(l^*)} = \min\{y^{(l)} | l = 1, 2, ..., p\}$ , $(\widehat{x}_{1,i}^-, ..., \widehat{x}_{n,i}^-) = (x_1, ..., x_n)^{(l^*)}$
$v_i^+ = y^{(l^{**})} = \max\{y^{(p+l)} | l = 1, 2, ..., p\}$ , $(\widehat{x}_{1,i}^+, ..., \widehat{x}_{n,i}^+) = (x_1, ..., x_n)^{(l^{**})}$
if $i < N$
  select $(x_1^{(l)}, ..., x_n^{(l)})$, $x_k^{(l)} \in [u_{k,i-1}^-, u_{k,i-1}^+]$ $\forall k$, $l = 1, ..., 2p$
  including $(\widehat{x}_{1,i}^-, ..., \widehat{x}_{n,i}^-)$ and $(\widehat{x}_{1,i}^+, ..., \widehat{x}_{n,i}^+)$
 endif
end

The second strategy MPDE is implemented in algorithm 2.

**Algorithm 2:** (Multi Populations DE procedure: MPDE with DE/rand/1).
Choose $p \approx 10n$, $g_{\max} \approx 500$, $q$ and $\gamma$.
select $(x_1^{(l,i)}, ..., x_n^{(l,i)})$, $x_k^{(l,i)} \in [u_{k,i}^-, u_{k,i}^+]$ $\forall k$, $l = 1, ..., 2p$, $i = 0, 1, ..., N$
let $y^{(l,i)} = f(x_1^{(l,i)}, ..., x_n^{(l,i)})$
let $v_i^- = \min\left\{y^{(l,j)} | j = 0, ..., i, \forall l\right\}$, $v_i^+ = \max\left\{y^{(l,j)} | j = 0, ..., i, \forall l\right\}$
denote by $\widehat{x}_i^-, \widehat{x}_i^+ \in \mathbb{R}^n$ the points where $v_i^-$ and $v_i^+$ are taken
for $g = 1, 2, ..., g_{\max}$ (up to $g_{\max}$ generations or other stopping rule)
 for $i = N, N-1, ..., 0$
  for $l = 1, 2, ..., p$
   select (randomly) $r, s, t \in \{1, 2, ..., p\}$ and $k^* \in \{1, 2, ..., n\}$
   for $k = 1, 2, ..., n$
    if ($k = k^*$ or $ran(0,1) < q$) then
     $x'_k = x_k^{(r,i)} + \gamma[x_k^{(s,i)} - x_k^{(t,i)}]$
     $x''_k = x_k^{(p+r,i)} + \gamma[x_k^{(p+s,i)} - x_k^{(p+t,i)}]$
     ensure $u_{k,i}^- \le x'_k, x''_k \le u_{k,i}^+$) (or other feasibility handling)
    else
     $x'_k = x_k^{(l,i)}$, $x''_k = x_k^{(p+l,i)}$
    endif
   end
   let $y' = f(x'_1, ..., x'_n)$ and $y'' = f(x''_1, ..., x''_n)$
   if $y' < y^{(l,i)}$ substitute $(x_1, ..., x_n)^{(l,i)}$ with $(x'_1, ..., x'_n)$ (min)
   if $y'' > y^{(p+l,i)}$ substitute $(x_1, ..., x_n)^{(p+l,i)}$ with $(x''_1, ..., x''_n)$ (max)
   update the values $\{v_j^-, v_j^+, \widehat{x}_j^-, \widehat{x}_j^+ | j = 0, ..., i\}$ if $y'$ or $y''$ are better
  end
 end
end

## 5 Computational results

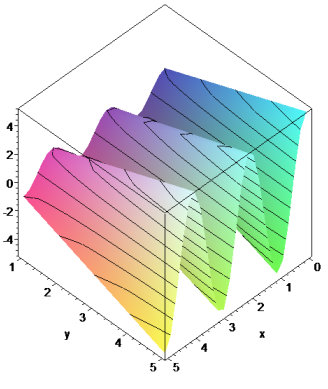
The two algorithms SPDE and MPDE have been implemented using C++ and Matlab and executed on a set of test functions with different dimension $n = 2, 4, 8, 16, 32$.

The problems are taken from the references [2], [5], [6], [7], [8], [9], [11], [12], [14].

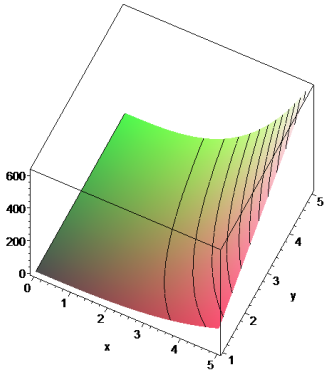
The first 20 problems are of dimension $n = 2$:

Problem 1: $f(x_1, x_2) = x_2 \cos(\pi x_1)$ over the support $(x_1, x_2) \in [0, 5] \times [1, 5]$

Problem 2:$f(x_1, x_2) = x_1^3 x_2$ over the support $(x_1, x_2) \in [0, 5] \times [1, 5]$



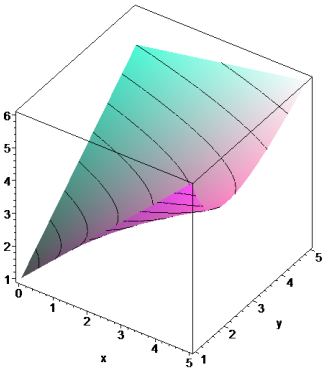

Function $f(x_1, x_2)$ of Problem 1.



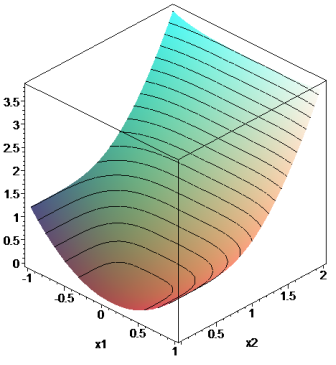

Function $f(x_1, x_2)$ of Problem 2.

Problem 3: $f(x_1, x_2) = x_2 + x_1/x_2$ over the support $(x_1, x_2) \in [0, 5] \times [1, 5]$

Problem 4: $f(x_1, x_2) = \sqrt{(x_1 - 0.1)^4 + (x_2 - 0.1)^4}$ over the support $(x_1, x_2) \in [-2, 2]^2$



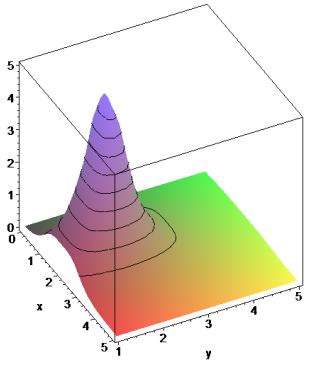

Function $f(x_1, x_2)$ of Problem 3.



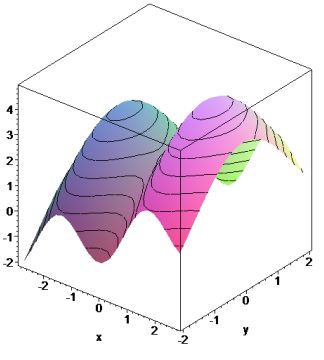

Function $f(x_1, x_2)$ of Problem 4.

Problem 5: $f(x_1, x_2) = \frac{1}{0.2+(x_1-2)^4+(x_2-2)^2}$ over the support $(x_1, x_2) \in [0, 5] \times [1, 5]$

Problem 6: $f(x_1, x_2) = 1 + \frac{1}{2}x_1 + \sin(2x_1 - \pi/2) + 2\cos(x_2)$ over the support $(x_1, x_2) \in [-2, 2]^2$



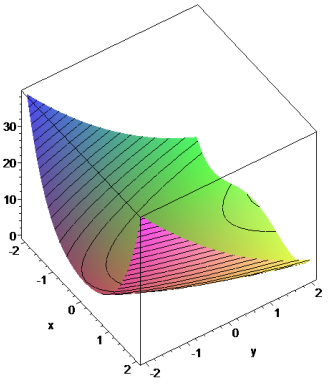

Function $f(x_1, x_2)$ of Problem 5.



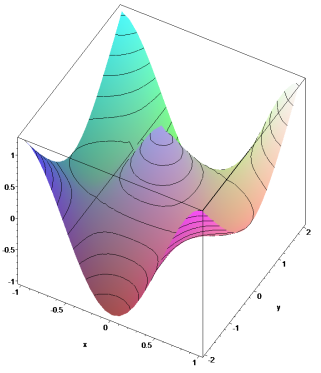

Function $f(x_1, x_2)$ of Problem 6.

Problem 7: $f(x_1, x_2) = (x_1^2 - x_2)^2 + 0.01(1 - x_1)^2$ over the support $(x_1, x_2) \in [-2, 2]^2$

Problem 8: $f(x_1, x_2) = (1 - \sqrt{x_1^2 + x_2^2})\sin(\pi(x_1 + \frac{1}{2}))$ over the support $(x_1, x_2) \in [-1, 1] \times [-2, 2]$



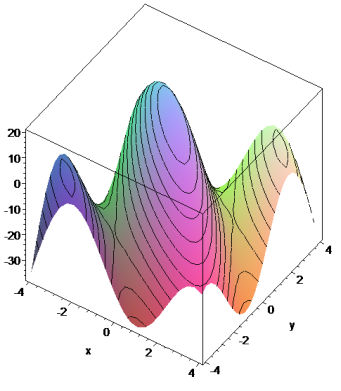

Function $f(x_1, x_2)$ of Problem 7.



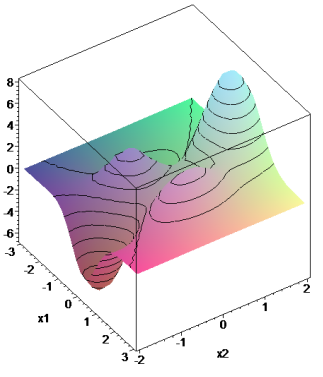

Function $f(x_1, x_2)$ of Problem 8.

Problem 9: $f(x_1, x_2) = 20\cos(x_1+x_2)-x_1^2-x_2^2$ over the support $(x_1, x_2) \in [-4, 4]^2$

Problem 10: $f(x_1, x_2) = 3(1 - x_1)^2 exp(-x_1^2 - (x_2 + 1)^2)$ $-10(\frac{1}{5}x_1 - x_1^3 - x_2^5)exp(-x_1^2 - x_2^2)$ $-exp(-x_2^2 - (1 + x_1)^2)/3$ over the support $(x_1, x_2) \in [-3, 3] \times [-2, 2]$



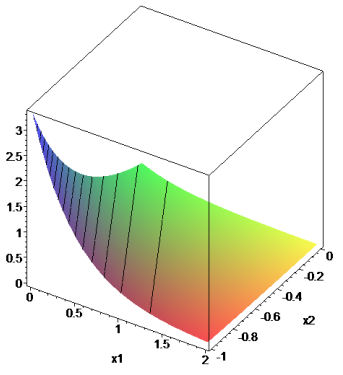

Function $f(x_1, x_2)$ of Problem 9.



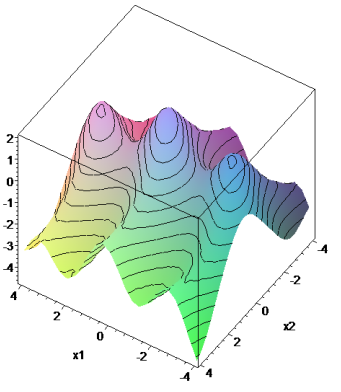

Function $f(x_1, x_2)$ of Problem 10.

Problem 11: $f(x_1, x_2) = exp(-2.1x_1 - 0.3)exp(-2.2x_2 - 0.7)$ over the support $(x_1, x_2) \in [0, 2] \times [-1, 0]$

Problem 12: $f(x_1, x_2) = \cos(2x_1 + \sin(x_2)) + \cos(x_2) - 0.1(x_1^2 + x_2^2)$ over the support $(x_1, x_2) \in [-4, 4]^2$



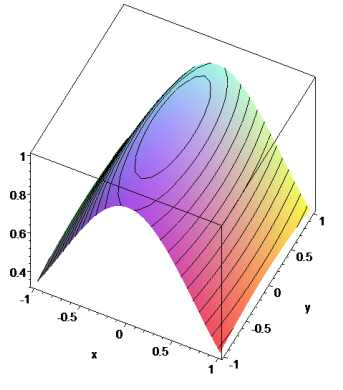

Function $f(x_1, x_2)$ of Problem 11.



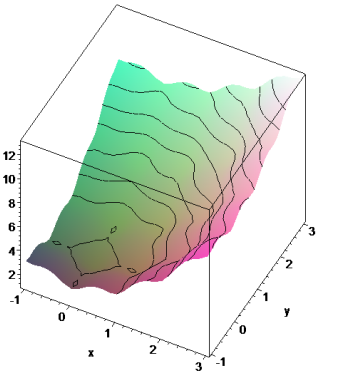

Function $f(x_1, x_2)$ of Problem 12.

Problem 13: $f(x_1, x_2) = exp(-x_1^2 - 0.1x_2^2)$ over the support $(x_1, x_2) \in [-1, 1]^2$

Problem 14: $f(x_1, x_2) = 20 + e - 20exp(-0.2(x_1^2 + x_2^2)/4) - exp((cos(2\pi x_1) + cos(2\pi x_2))/4)$ over the support $(x_1, x_2) \in [-1, 3]^2$



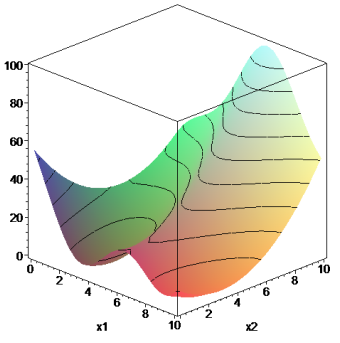

Function $f(x_1, x_2)$ of Problem 13.



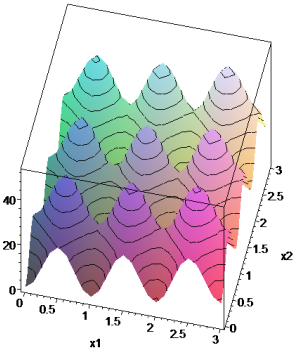

Function $f(x_1, x_2)$ of Problem 14.

Problem 15: $f(x_1, x_2) = (5x_1/\pi - \frac{5.1x_1^2}{4\pi^2} + x_2 - 6)^2 + 10(1 - \frac{1}{8\pi})cos(x_1) + 10$ over the support $(x_1, x_2) \in [0, 10]^2$

Problem 16: $f(x_1, x_2) = x_1^2 - 10cos(2\pi x_1) + 10 + x_2^2 - 10cos(2\pi x_2) + 10$ over the support $(x_1, x_2) \in [0, 3]^2$



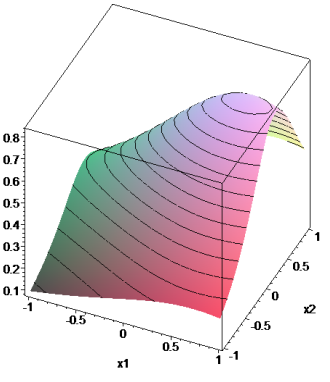

Function $f(x_1, x_2)$ of Problem 15.



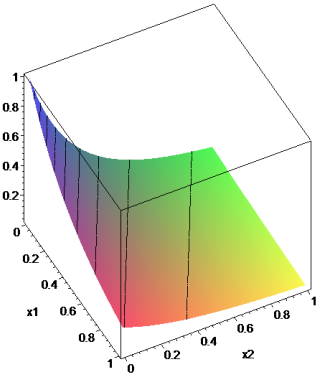

Function $f(x_1, x_2)$ of Problem 16.

Problem 17: $f(x_1, x_2) = \frac{1}{1/0.7^2+(x_1-0.7)^2} \frac{1}{1/1.3^2(x_2-0.3)^2}$ over the support $(x_1, x_2) \in [-1, 1]^2$

Problem 18: $f(x_1, x_2) = \frac{1}{(1+0.7x_2+1.3x_2)^3}$ over the support $(x_1, x_2) \in [0, 1]^2$



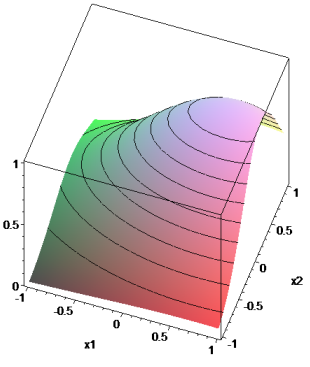

Function $f(x_1, x_2)$ of Problem 17.



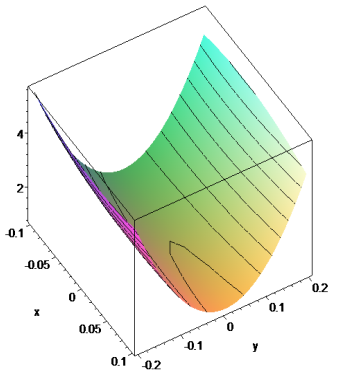

Function $f(x_1, x_2)$ of Problem 18.

Problem 19: $f(x_1, x_2) = exp[-(0.7(x_1-0.7))^2-(1.3(x_2-0.3))^2]$ over the support $(x_1, x_2) \in [-1, 1]^2$

Problem 20: $f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$ over the support $(x_1, x_2) \in [-0.1, 0.1] \times [-0.2, 0.2]$




Function $f(x_1, x_2)$ of Problem 19.




Function $f(x_1, x_2)$ of Problem 20.

The six problems of the second group have dimension $n = 4$.

Problem 21: $f(x_1, ..., x_4) = \sum_{i=1}^{4} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$ over the support $x_i \in [0, 3]$

Problem 22: $f(x_1, ..., x_4) = \prod_{i=1}^{4} (c_i^{-2} + (x_i - w_i)^2)^{-1}$ with $c = (0.8, 1.5, 2.3, 2.43)$ and $w = (0.2, 0.4, 0.3, 0.1)$ over the support $x_i \in [-1, 1]^4$

Problem 23: $f(x_1, ..., x_4) = (1 + \sum_{i=1}^{4} c_i x_i)^{-(1+n)}$ with $c = (0.8, 1.5, 2.3, 2.43)$ over the support $x_i \in [0, 1]^4$

Problem 24: $f(x_1, ..., x_4) = \exp(-\sum_{i=1}^{4} c_i^2 (x_i - w_i)^2)$ with $c = (0.8, 1.5, 2.3, 2.43)$ over the support $x_i \in [-1, 1]^4$

Problem 25: $f(x_1, ..., x_4) = 20 + e - 20\exp\left[-0.2\sqrt{\frac{1}{4}\sum_{i=1}^{4} x_i^2}\right] - \exp\left[\frac{1}{4}\sum_{i=1}^{4}\cos(2\pi x_i)\right]$ over the support $(x_1, x_2) \in [-1, 3]^4$

Problem 26: $f(x_1, ..., x_4) = \sum_{i=1}^{3} \left[10(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$ over the support $x_i \in [-0.2, 0.2]^4$.

The last three groups contain problems of dimension $n = 8$, $n = 16$ and $n = 32$ respectively.

They are constructed by the Rastrigin function, the Ackley function and a modified Rosenbrock function:

1. Rastrigin: $f(x_1, ..., x_n) = \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) + 10\right]$, $x_i \in [0, 3]$;

2. Ackley: $f = 20 + e - 20\exp\left[-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right] - \exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right]$, $x_i \in [-1, 3]$;

3. Modified Rosenbrock: $f(x_1, ..., x_n) = \sum_{i=1}^{n-1} \left[10(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right], x_i \in [-0.2, 0.2]$.

Problem 27: Rastrigin function with $n = 8$.

Problem 28: Ackley function with $n = 8$.

Problem 29: Modofied Rosenbrock function with $n = 8$.

Problem 30: Rastrigin function with $n = 16$.

Problem 31: Ackley function with $n = 16$.

Problem 32: Modofied Rosenbrock function with $n = 16$.

Problem 33: Rastrigin function with $n = 32$.

Problem 34: Ackley function with $n = 32$.

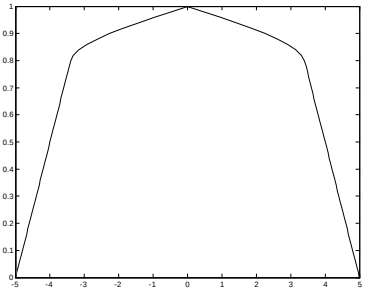
Problem 35: Modofied Rosenbrock function with $n = 32$.

In the computations, the input fuzzy numbers are triangular and symmetric of the form $u_i = \left\langle a_i, \frac{a_i+b_i}{2}, b_i \right\rangle$ with support given by the interval $[a_i, b_i]$; the applied supports are illustrated in the table.

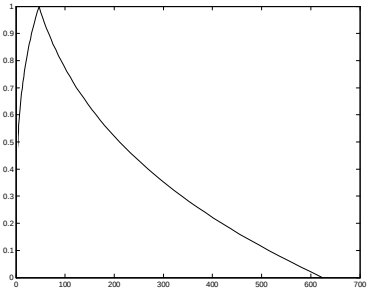
The extension algorithm is used in combinations with the LU-fuzzy representation for differentiable membership functions (and differentiable extended functions) and the number $N + 1$ of $\alpha - cuts$ (and correspondingly of min/max optimizations) can be sufficiently small. Experiments in [5] and [12] motivated that $N = 5$ is in general quite sufficient to obtain very good approximations.

In the present tests, $N = 10$ is used.

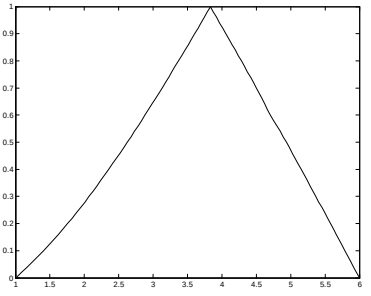
The figures here illustrate the LU-fuzzy extensions obtained by the two algorithms.



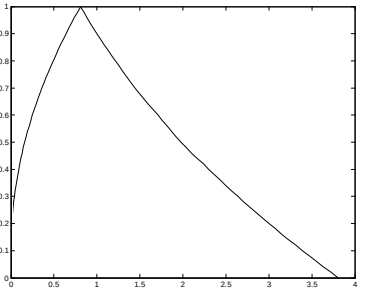

Fuzzy extension of problem 1



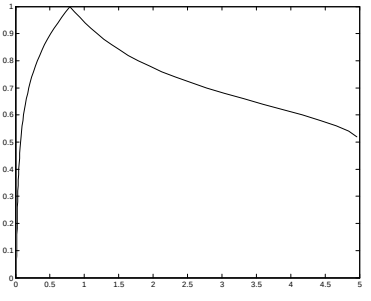

Fuzzy extension of problem 2



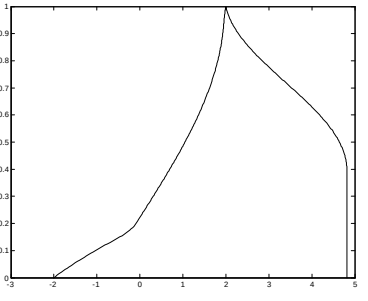
Fuzzy extension of problem 3



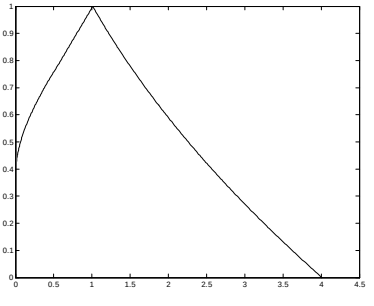
Fuzzy extension of problem 4



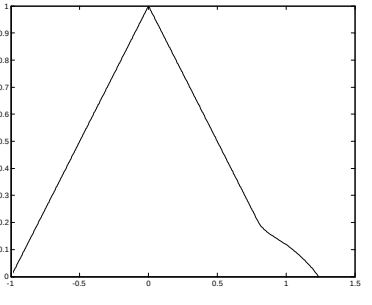
Fuzzy extension of problem 5



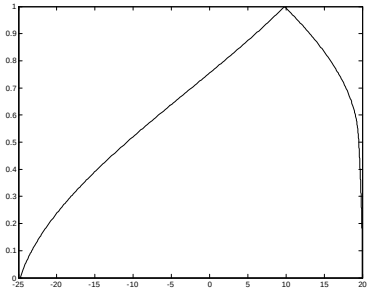
Fuzzy extension of problem 6



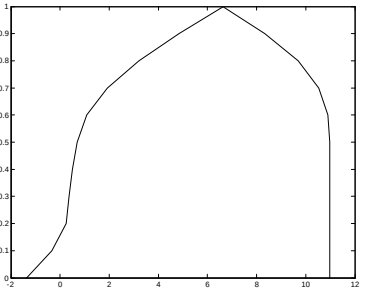
Fuzzy extension of problem 7



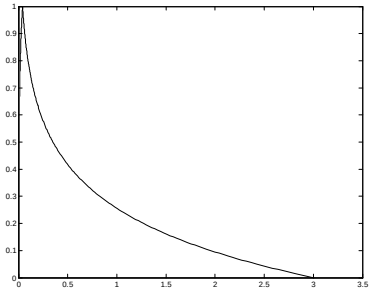
Fuzzy extension of problem 8



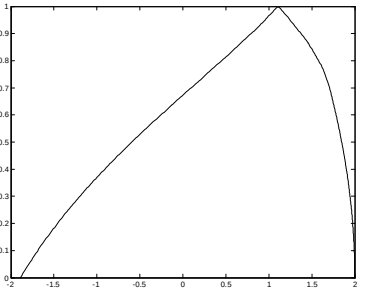

Fuzzy extension of problem 9



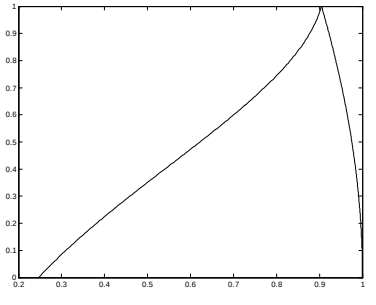

Fuzzy extension of problem 10



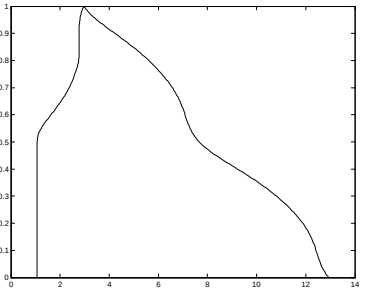

Fuzzy extension of problem 11



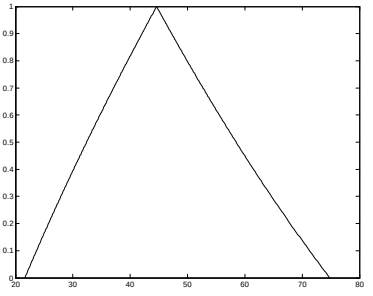

Fuzzy extension of problem 12



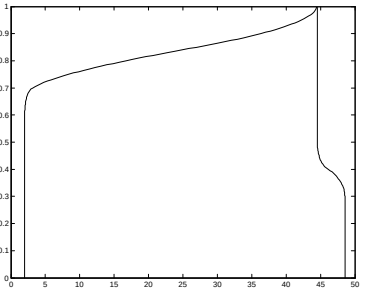

Fuzzy extension of problem 13



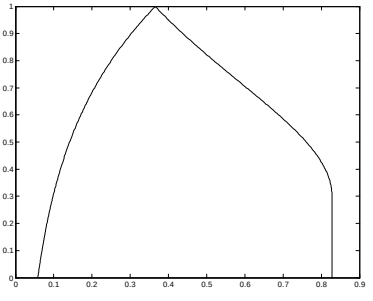

Fuzzy extension of problem 14



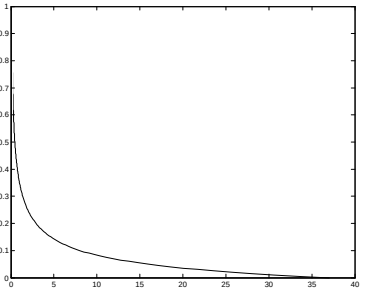
Fuzzy extension of problem 15



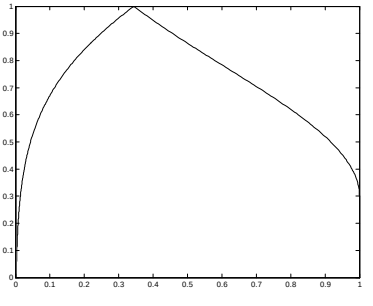
Fuzzy extension of problem 16



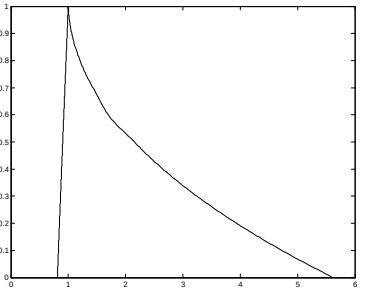
Fuzzy extension of problem 17



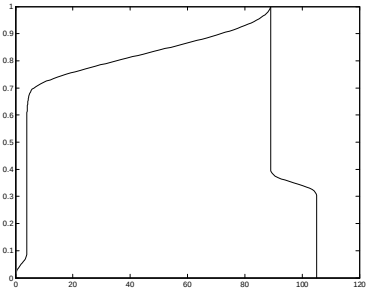
Fuzzy extension of problem 18



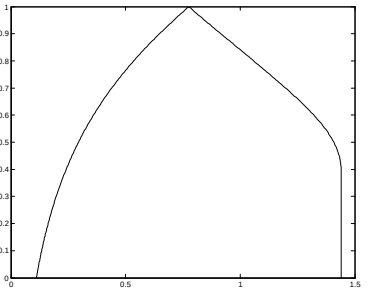
Fuzzy extension of problem 19



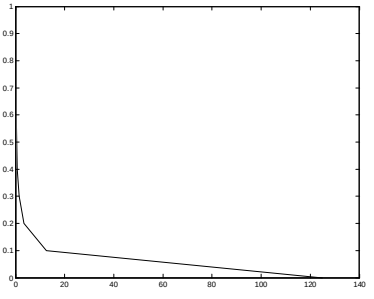
Fuzzy extension of problem 20



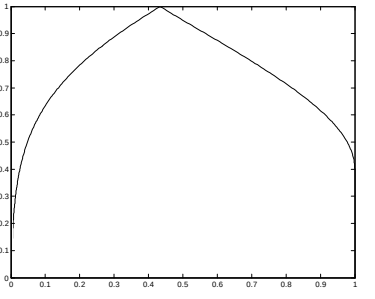

Fuzzy extension of problem 21



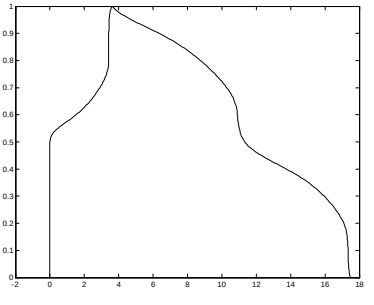

Fuzzy extension of problem 22



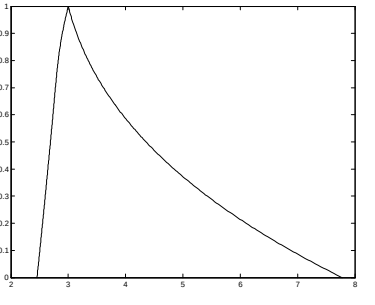

Fuzzy extension of problem 23



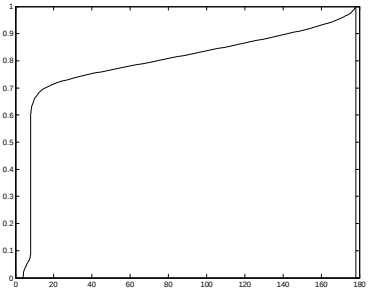

Fuzzy extension of problem 24



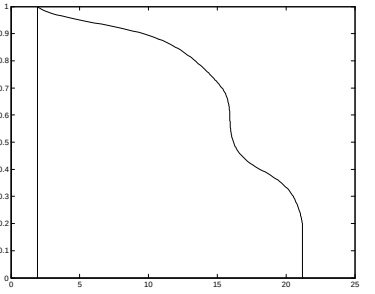

Fuzzy extension of problem 25



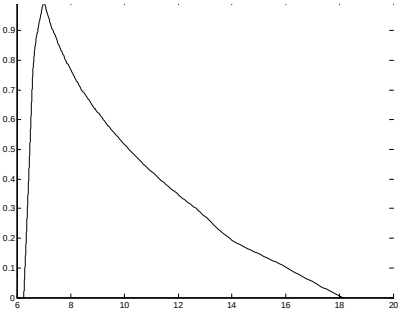

Fuzzy extension of problem 26



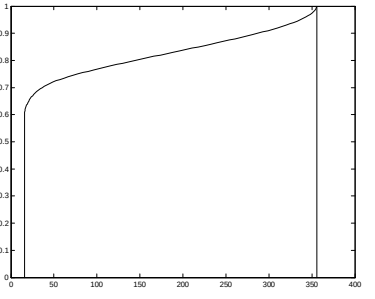

Fuzzy extension of problem 27



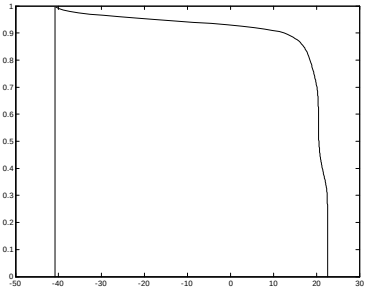

Fuzzy extension of problem 28



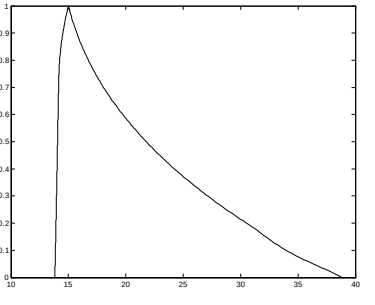

Fuzzy extension of problem 29



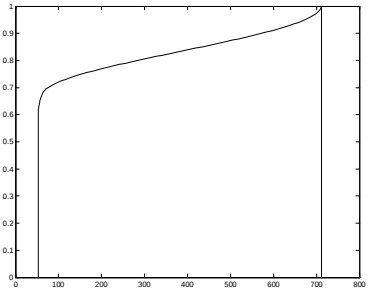

Fuzzy extension of problem 30



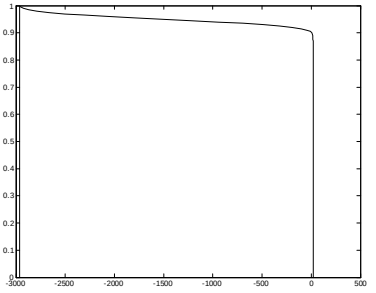

Fuzzy extension of problem 31



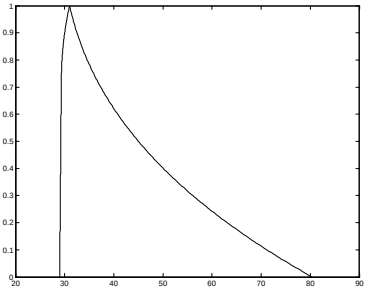

Fuzzy extension of problem 32




Fuzzy extension of problem 33




Fuzzy extension of problem 34




Fuzzy extension of problem 35

.

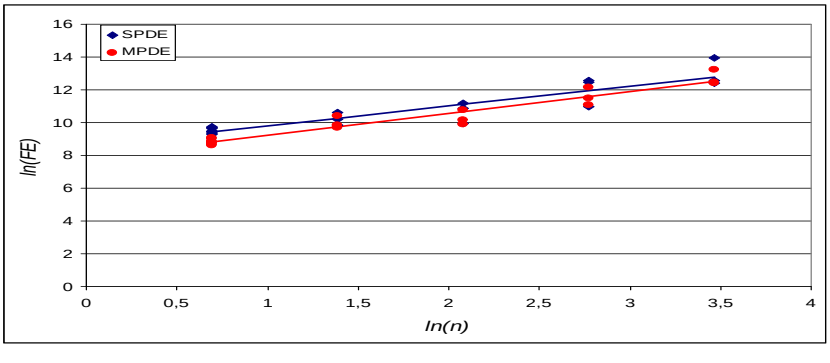
The table below reports the number of function evaluations $FE_{SPDE}$ and $FE_{MPDE}$ needed to the two algorithms $SPDE$ and $MPDE$ to reach the solution of the nested min/max optimization problems corresponding to the 11 $\alpha - cuts$ of the uniform $\alpha-$decomposition $\alpha_i = \frac{i}{10}$, $i = 0, 1, ..., 10$ ($N = 10$ subintervals).

To decide that a solution is found, we use the following simple rule: choose a fixed tolerance $tol \sim 10^{-3}, 10^{-4}$ and a number $\widehat{g} \sim 20, 30$ of generations; if for $\widehat{g}$ subsequent iterations all the values $v_i^-$ and $v_i^+$ are changed less than $tol$, then the procedure stops and the found solution is assumed to be optimal. In any case, no more than 500 iterations are performed (but this limit was never reached during the computations).

| $Problem$ | $n$ | $FE_{SPDE}$ | $FE_{MPDE}$ | $Problem$ | $n$ | $FE_{SPDE}$ | $FE_{MPDE}$ |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 15400 | 8800 | 19 | 2 | 11880 | 6820 |
| 2 | 2 | 12760 | 7260 | 20 | 2 | 11440 | 6600 |
| 3 | 2 | 11880 | 6820 | 21 | 4 | 40040 | 32560 |
| 4 | 2 | 12760 | 6380 | 22 | 4 | 31240 | 22000 |
| 5 | 2 | 16280 | 6600 | 23 | 4 | 15840 | 15840 |
| 6 | 2 | 12760 | 6380 | 24 | 4 | 19360 | 14520 |
| 7 | 2 | 12320 | 5720 | 25 | 4 | 27720 | 16280 |
| 8 | 2 | 11000 | 5500 | 26 | 4 | 19360 | 18920 |
| 9 | 2 | 16720 | 7040 | 27 | 8 | 72160 | 47520 |
| 10 | 2 | 18920 | 10560 | 28 | 8 | 51392 | 25344 |
| 11 | 2 | 13640 | 7700 | 29 | 8 | 20416 | 19712 |
| 12 | 2 | 13200 | 8140 | 30 | 16 | 292160 | 186560 |
| 13 | 2 | 10120 | 5280 | 31 | 16 | 255552 | 98560 |
| 14 | 2 | 16280 | 6380 | 32 | 16 | 59136 | 63360 |
| 15 | 2 | 16280 | 7920 | 33 | 32 | 1122176 | 560384 |
| 16 | 2 | 23760 | 9020 | 34 | 32 | 283008 | 252032 |
| 17 | 2 | 11880 | 6600 | 35 | 32 | 250624 | 243584 |
| 18 | 2 | 11880 | 6600 | | | | |

The figure below represents the logarithm of the number of function evaluations vs the logarithm of the number $n$ of arguments. It appears an almost linear relationship $\ln(FE_{SPDE}) = a + b\ln(n)$ and $\ln(FE_{MPDE}) = c + d\ln(n)$ : the estimated coefficients are $a = 8.615, b = 1.20$ and $c = 7.869, d = 1.34$. The computational complexity of the proposed algorithms ( on average for the 22 test problems) grows less then quadratically with the dimension $n$ (SPDE is less efficient but grows slowly than MPDE). This is an interesting result, as all the existing methods for the fuzzy extension of functions are essentially exponential in $n$.

The C++ source codes are available on request to the author; also a MatLab implementation is available.



## References


1. H. K. Chen, W. K. Hsu, W. L. Chiang, A comparison of vertex method with JHE method, *Fuzzy Sets and Systems*, 95, 1998, 201-214.
2. K. Deb, An efficient constraint handling method for genetic algorithms, *Computer methods in applied mechanics and engineering*, 186, 2000, 311-338.
3. W. M. Dong, H. C. Shah, Vertex method for computing functions of fuzzy variables, *Fuzzy Sets and Systems*, 24, 1987, 65-78.
4. D. Dubois, H. Prade (ed), *Fundamentals of Fuzzy Sets*, Kluwer, Boston, The Handbooks of Fuzzy Sets Series, 2000.
5. M. L. Guerra, L. Stefanini, Approximate Fuzzy Arithmetic Operations Using Monotonic Interpolations, *Fuzzy Sets and Systems*, 150, 2005, 5-33.
6. M. Hanss, The transformation method for the simulation and analysis of systems with uncertain parameters, *Fuzzy Sets and Systems*, 130, 2002, 277-289.
7. M. Hanss, A. Klimke, On the reliability of the influence measure in the transformation method of fuzzy arithmetic, *Fuzzy Sets and Systems*, 143, 2004, 371-390.
8. A. Klimke, An efficient implementation of the transformation method of fuzzy arithmetic, Extended Preprint Report, 2003/009, Institut of Applied Analysis and Numerical Simulation, University of Stuttgard, Germany, 2003.
9. A. Klimke, B. Wohlmuth, Computing expensive multivariate functions of fuzzy numbers using sparse grids, *Fuzzy Sets and Systems*, 153, 2005, 432-453.
10. E. N. Otto, A. D. Lewis, E. K. Antonsson, Approximating $\alpha - cuts$ with the vertex method, *Fuzzy Sets and Systems*, 55, 1993, 43-50.
11. K. Price, An introduction to differential evolution, in D. Corne, M. Dorigo, F. Glover (ed.), *New Ideas in Optimization*, McGraw Hill, 1999, 79-108.
12. L. Stefanini, L. Sorini, M. L. Guerra, Parametric representation of fuzzy numbers and application to fuzzy calculus, *Fuzzy Sets and Systems*, 157, 2006, 2423 – 2455.
13. R. Storn, K. Price, Differential Evolution: a simple and efficient heuristic for global optimization over continuous spaces, ICSI technical report TR-95-012, Berkeley University, 1995. Also, *Journal of Global Optimization*, 11, 1997, 341-359.
14. R. Storn, System design by constraint adaptation and differential evolution, *IEEE Transactions on Evolutionary Computation*, 3, 1999, 22-34.
15. K. L. Wood, K. N. Otto, E. K. Antonsson, Engineering design calculations with fuzzy parameters, *Fuzzy Sets and Systems*, 52, 1992, 1-20.
16. L. A. Zadeh, Fuzzy Sets, *Information and Control*, 8, 1965, 338-353.

APPENDIX 1
Matlab Implementation of the Single Population DE Algorithm: SPDE

```
function [fU, nGen, nfe] = ...
    FuzzySPDE(fname,D,Extra,N,U,Pfact,strategy,reiter,replot,maxGen,gTol,noGen)
% Fuzzy Extension of a D-dimensional function defined by
%      f(x) = fname(x,D,Extra,k),
% using the Single Population differential evolution (SPDE) algorithm.
% The D-dimensional fuzzy argument U and result fU are in LU-fuzzy format:
%       U = (Um,dUm,Up,dUp;i=1:N+1) over N uniform alpha's subintervals
%           in 3-dimensional matrix form
%           U(i=1:N+1;j=1:D;k=1:4)
%           where    i  is the index of alpha-cut corresponding to
%                       alpha(i) = (i-1)/N. (i starts from 1 to N+1)
%                    j  is the index of a component of U
%                    k  is 1 for Um, 2 for dUm, 3 for Up, 4 for dUp
%                       so that  U(i,j,:) are the four LU-fuzzy parameters
%                       defining U(j) at i-th alpha-cut
%      fU = (fUm,dfUm,fUp,dfUp;i=1:N+1) over N uniform alpha's subintervals
%           so that  fU(i,:) are the four LU-fuzzy parameters defining
%           f(U) at i-th alpha-cut
%
% Output:
%    fU        Fuzzy Extended value f(U) in LU-fuzzy form
%    nf        number of function evaluations needed
%
% Inputs:
%    fname     string naming a function f(x,y) to minimize
%    D         number of parameters of the function
%    Extra     vector of extra values eventually needed to calculate f(x)
%    Pfact     factor for the number NP = Pfact*D of population members
%    maxGen    maximum number of generations (>= 200)
%
% Value function f(x) has to be defined by the user
% define fname.m as
%               function fv = fname(x,D,Extra,k)
%                  fv = function value f at x if k=0
%                  fv = partial first derivative of f at x wrt x(k) if k=1:D
%               end
%
% Internal Important Parameters:
%    F         DE-stepsize (in [0, 2])
%    CR        crossover probability (in [0, 1])
%    strategy    1 --> DE/best/1/exp           6 --> DE/best/1/bin
%                2 --> DE/rand/1/exp           7 --> DE/rand/1/bin
%                3 --> DE/rand-to-best/1/exp   8 --> DE/rand-to-best/1/bin
%                4 --> DE/best/2/exp           9 --> DE/best/2/bin
%                5 --> DE/rand/2/exp           else  DE/rand/2/bin
%    reiter      intermediate output will be produced after "reiter"
%                iterations. No intermediate output will be produced
%                if reiter is < 1
%    replot      same as for reiter, to plot the populations of level k=1
%
% Notes:
%       The code is adapted from
%             function devec3, (1997), by R. Storn (ICSI, Berkeley)
%       author: Luciano Stefanini, University of Urbino, Italy, 2006.
%               lucste@uniurb.it
%       A good initial guess is to choose F from interval [0.5, 1],
%       the crossover probability CR from interval [0, 1] helps to maintain
%       the diversity of the population and is rather uncritical. The
%       number of population members NP is also not very critical. A
%       good initial guess is 10*D. Depending on the difficulty of the
%       problem NP can be lower than 10*D or must be higher than 10*D
%       to achieve convergence.
%       If the parameters are correlated, high values of CR work better.
%       The reverse is true for no correlation.
%
% This program is free software; you can redistribute and/or modify it.
% It is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
```

```
% default values:
NP = Pfact*D;
F = 0.8;
CR = 0.5;

if (NP < 5)
   NP=5;
end
if (maxGen <= 0)
   maxGen = 200;
end
reiter = floor(reiter);
replot = floor(replot);
if ( D < 2  ||  D > 2 )
    replot = 0;
end

% Initialize arrays
Um  = zeros(N+1,D);
Up  = zeros(N+1,D);
dUm = zeros(N+1,D);
dUp = zeros(N+1,D);
for k=1:N+1
    for j=1:D
        Um(k,j)  = U(k,j,1);
        Up(k,j)  = U(k,j,3);
        dUm(k,j) = U(k,j,2);
        dUp(k,j) = U(k,j,4);
    end
end
fU  = zeros(N+1,4);

pop     = zeros(2*NP,D);
bminval = zeros(1,N+1);
bmaxval = zeros(1,N+1);
minmemit = zeros(1,D);
minmem   = zeros(N+1,D);
maxmemit = zeros(1,D);
maxmem   = zeros(N+1,D);
val      = zeros(1,2*NP);
lx      = zeros(1,D);
ux      = zeros(1,D);
tmpx    = zeros(1,D);
% populations 1 to NP for lower branch and NP+1 to 2NP for upper branch

nfe       = 0;            % number of function evaluations
nGen      = 0;

for k = (N+1):-1:1

    lx  = reshape(Um(k,:),1,D);
    ux  = reshape(Up(k,:),1,D);
    for i=1:2*NP
        pop(i,:) = lx + rand(1,D).*(ux - lx);
    end
    if k < N+1
       pop(1,:)    = minmem(k+1,:);
       pop(NP+1,:) = maxmem(k+1,:);
   end
    % Evaluate initial populations
    for i=1:2*NP
        tmpx = pop(i,:);
        val(i) = feval(fname,tmpx,D,Extra,0);
        nfe  = nfe + 1;
    end
    % identify best initial values for each cut
    imin    = 1;
    imax    = 1;
    minval = val(1);
    maxval = val(1);
    for i=2:2*NP
        if (val(i) < minval)
            imin   = i;
            minval = val(i);
        end
        if (val(i) > maxval)
            imax   = i;
            maxval = val(i);
```

```
    end
end
bminval(k)    = minval;           % best min value ever
bmaxval(k)    = maxval;           % best max value ever
minmemit      = pop(imin,:);      % best min member of current iteration
minvalit      = minval;           % best min value of current iteration
minmem(k,:)   = minmemit;         % best min member ever
maxmemit      = pop(imax,:);      % best max member of current iteration
maxvalit      = maxval;           % best max value of current iteration
maxmem(k,:)   = maxmemit;         % best max member ever


%------popold is the population which has to compete. It is--------
%------static through one iteration. pop is the newly--------------
%------emerging population.-----------------------------------------

popold = zeros(NP,D);        % toggle populations

pm1 = zeros(NP,D);              % initialize population matrix 1
pm2 = zeros(NP,D);              % initialize population matrix 2
pm3 = zeros(NP,D);              % initialize population matrix 3
pm4 = zeros(NP,D);              % initialize population matrix 4
pm5 = zeros(NP,D);              % initialize population matrix 5
bm  = zeros(NP,D);              % initialize bestmember  matrix
ui  = zeros(NP,D);              % intermediate population of perturbed vectors
mui = zeros(NP,D);              % mask for intermediate population
mpo = zeros(NP,D);              % mask for old population
rot = (0:1:NP-1);               % rotating index array (size NP)
rotd= (0:1:D-1);                % rotating index array (size D)
rt  = zeros(NP);                % another rotating index array
rtd = zeros(D);                 % rotating index array for exponential crossover
a1  = zeros(NP);                % index array
a2  = zeros(NP);                % index array
a3  = zeros(NP);                % index array
a4  = zeros(NP);                % index array
a5  = zeros(NP);                % index array
ind = zeros(4);

iter = 1;
noValid = 0;
while ((iter <= maxGen) & (noValid < noGen))
    improved = 0;
    ind = randperm(4);                % index pointer array
    a1  = randperm(NP);               % shuffle locations of vectors
    rt = rem(rot+ind(1),NP);          % rotate indices by ind(1) positions
    a2  = a1(rt+1);                   % rotate vector locations
    rt = rem(rot+ind(2),NP);
    a3  = a2(rt+1);
    rt = rem(rot+ind(3),NP);
    a4  = a3(rt+1);
    rt = rem(rot+ind(4),NP);
    a5  = a4(rt+1);

   % process lower branches, nested Min problems
   for i=1:NP
      popold(i,:) = pop(i,:);  % save the old population
   end
   pm1 = popold(a1,:);       % shuffled population 1
   pm2 = popold(a2,:);       % shuffled population 2
   pm3 = popold(a3,:);       % shuffled population 3
   pm4 = popold(a4,:);       % shuffled population 4
   pm5 = popold(a5,:);       % shuffled population 5
   for i=1:NP    % population filled with best member of last iteration
       for j=1:D
           bm(i,j) = minmemit(j);
       end
   end
   mui = rand(NP,D) < CR;  % all random numbers < CR are 1, 0 otherwise
   if (strategy > 5)
      st = strategy-5;           % binomial crossover
   else
      st = strategy;                % exponential crossover
      mui=sort(mui');         % transpose, collect 1's  in each column
      for i=1:NP
         n=floor(rand*D);
         if n > 0
            rtd = rem(rotd+n,D);
            mui(:,i) = mui(rtd+1,i); %rotate column i by n
```

```
      end
    end
    mui = mui';                       % transpose back
 end
 mpo = mui < 0.5;                % inverse mask to mui
 if (st == 1)                      % DE/best/1
    ui = bm + F*(pm1 - pm2);
    ui = popold.*mpo + ui.*mui;
 elseif (st == 2)                  % DE/rand/1
    ui = pm3 + F*(pm1 - pm2);
    ui = popold.*mpo + ui.*mui;
 elseif (st == 3)                  % DE/rand-to-best/1
    ui = popold + F*(bm-popold) + F*(pm1 - pm2);
    ui = popold.*mpo + ui.*mui;
 elseif (st == 4)                  % DE/best/2
    ui = bm + F*(pm1 - pm2 + pm3 - pm4);
    ui = popold.*mpo + ui.*mui;
 elseif (st == 5)                  % DE/rand/2
    ui = pm5 + F*(pm1 - pm2 + pm3 - pm4);
    ui = popold.*mpo + ui.*mui;
 end
 % force feasibility of current population for level k
 for i=1:NP
     for j=1:D
         if ( ui(i,j) < lx(j) )
             ui(i,j) = lx(j);
         end
         if ( ui(i,j) > ux(j) )
             ui(i,j) = ux(j);
         end
     end
 end

 if (replot > 0)
   if (rem(iter,replot) == 0 && k==1)
      figure(3);
      xplot = zeros(NP);
      yplot = zeros(NP);
      xplot = reshape(ui(:,1),1,NP);
      yplot = reshape(ui(:,2),1,NP);
      subplot(1,1,1);
      plot(xplot,yplot,'or');
      drawnow; %---Draws current graph now
   end
 end

 % Select which vectors are allowed to enter the new min population
 for i=1:NP
    tmpx = ui(i,:);
    tempval = feval(fname,tmpx,D,Extra,0);   % check cost of competitor
    nfe     = nfe + 1;
    if (bminval(k) > tempval)
        if (tempval + gTol < bminval(k))
            improved = 1;
        end
        bminval(k) = tempval;
        minmem(k,:) = tmpx;
    end
    if (tempval < val(i)) % if competitor is better
       pop(i,:) = tmpx;    % replace old vector (for new iteration)
       val(i)   = tempval;
    end
    if (bmaxval(k) < tempval)
        if (tempval - gTol > bmaxval(k))
            improved = 1;
        end
        bmaxval(k) = tempval;
        maxmem(k,:) = tmpx;
    end
    if (tempval > val(NP+i))  % if competitor is better
       pop(NP+i,:) = tmpx;    % replace old vector (for new iteration)
       val(NP+i)   = tempval;
    end
end            % end for member i=1:NP

 % process upper branches, nested Max problems
 for i=1:NP
     popold(i,:) = pop(NP+i,:);  % save the old population
```

```
end
pm1 = popold(a1,:);             % shuffled population 1
pm2 = popold(a2,:);             % shuffled population 2
pm3 = popold(a3,:);             % shuffled population 3
pm4 = popold(a4,:);             % shuffled population 4
pm5 = popold(a5,:);             % shuffled population 5
for i=1:NP                           % population filled with the best member
    for j=1:D
        bm(i,j) = maxmemit(j);
    end
end
mui = rand(NP,D) < CR;          % all random numbers < CR are 1, 0 otherwise
if (strategy > 5)
   st = strategy-5;                 % binomial crossover
else
   st = strategy;                       % exponential crossover
   mui=sort(mui');              % transpose, collect 1's in each column
   for i=1:NP
      n=floor(rand*D);
      if n > 0
         rtd = rem(rotd+n,D);
         mui(:,i) = mui(rtd+1,i); %rotate column i by n
     end
   end
   mui = mui';                      % transpose back
end
mpo = mui < 0.5;                % inverse mask to mui
if (st == 1)                      % DE/best/1
   ui = bm + F*(pm1 - pm2);
   ui = popold.*mpo + ui.*mui;
elseif (st == 2)                  % DE/rand/1
   ui = pm3 + F*(pm1 - pm2);
   ui = popold.*mpo + ui.*mui;
elseif (st == 3)                  % DE/rand-to-best/1
   ui = popold + F*(bm-popold) + F*(pm1 - pm2);
   ui = popold.*mpo + ui.*mui;
elseif (st == 4)                  % DE/best/2
   ui = bm + F*(pm1 - pm2 + pm3 - pm4);
   ui = popold.*mpo + ui.*mui;
elseif (st == 5)                  % DE/rand/2
   ui = pm5 + F*(pm1 - pm2 + pm3 - pm4);
   ui = popold.*mpo + ui.*mui;
end
% force feasibility of current population for level k
for i=1:NP
    for j=1:D
        if ( ui(i,j) < lx(j) )
            ui(i,j) = lx(j);
        end
        if ( ui(i,j) > ux(j) )
            ui(i,j) = ux(j);
        end
    end
end
if (replot > 0)
   if (rem(iter,replot) == 0  &&  k==1)
       figure(4);
       xplot = zeros(NP);
       yplot = zeros(NP);
       xplot = reshape(ui(:,1),1,NP);
       yplot = reshape(ui(:,2),1,NP);
       subplot(1,1,1);
       plot(xplot,yplot,'ob');
       drawnow; %---Draws current graph now
   end
end

% Select which vectors are allowed to enter the new max population
for i=1:NP
   tmpx = ui(i,:);
   tempval = feval(fname,tmpx,D,Extra,0);   % check cost of competitor
   nfe     = nfe + 1;
   if (bmaxval(k) < tempval)
       if (tempval - gTol > bmaxval(k))
           improved = 1;
       end
       bmaxval(k) = tempval;
       maxmem(k,:) = tmpx;
```

```
          end
          if (tempval > val(NP+i))  % if competitor is better
             pop(NP+i,:) = tmpx;     % replace old vector (for new iteration)
             val(NP+i)   = tempval;
          end
          if (bminval(k) > tempval)
              if (tempval + gTol < bminval(k))
                  improved = 1;
              end
              bminval(k) = tempval;
              minmem(k,:) = tmpx;
          end
          if (tempval < val(i))  % if competitor is better
             pop(i,:) = tmpx;     % replace old vector (for new iteration)
             val(i)   = tempval;
          end
       end            % end for member i=1:NP
       maxmemit(k,:) = maxmem(k,:); % freeze the best member of this iteration for the coming
                                    % iteration. This is needed for some of the strategies.
       minmemit(k,:) = minmem(k,:); % freeze the best member of this iteration for the coming
                                    % iteration. This is needed for some of the strategies.

       % Intermediate Output section
       if (reiter > 0)
          if (rem(iter,reiter) == 0)
              fprintf(1,'\nGeneration: %d,  Function Evaluations: %d\n',iter, nfe);
              fprintf(1,'Level: %d,  BestMin: %f,  BestMax: %f\n',k-1,bminval(k),bmaxval(k));
          end
       end
       % next generation
       noValid = noValid + 1;
       if (improved == 1)
           noValid = 0;
       end
       iter = iter + 1;
    end
     if nGen < iter - 1
        nGen = iter-1;
    end

end    % next level



% Construct result fuzzy number fU
iter = 1;
while (iter > 0)
    iter = 0;
    for k = 1:N
        if  ( bminval(k+1) < bminval(k) )
            iter = 1;
            bminval(k) = bminval(k+1);
            minmem(k,:) = minmem(k+1,:);
        end
        if  ( bmaxval(k+1) > bmaxval(k) )
            iter = 1;
            bmaxval(k) = bmaxval(k+1);
            maxmem(k,:) = maxmem(k+1,:);
        end
    end
end

TolBound = 0.0001;
for k=1:N+1
    dlval = 0.0;
    lval = bminval(k);
    tmpx = minmem(k,:);
    for j=1:D
        go=0;
        if ( tmpx(j) < Um(k,j) + TolBound )
            go = go+1;
        end
        if ( tmpx(j) > Up(k,j) - TolBound)
            go = go+2;
        end
        if (go>0)
            dval = feval(fname,tmpx,D,Extra,j);
            if (go == 1)
```

```
                dlval = dlval + dval * dUm(k,j);
            elseif (go == 2)
                dlval = dlval + dval * dUp(k,j);
            elseif (go == 3)
                if (dval > 0.0)
                   dlval = dlval + dval*dUm(k,j);
                else
                   dlval = dlval + dval*dUp(k,j);
               end
            end
        end
        if dlval < 0
            dlval = 0;
        end
    end

    duval = 0.0;
    uval = bmaxval(k);
    tmpx = maxmem(k,:);
    for j=1:D
        go=0;
        if ( tmpx(j) > Um(k,j) - TolBound )
            go = go+2;
        end
        if ( tmpx(j) < Up(k,j) + TolBound)
            go = go+1;
        end
        if (go>0)
            dval = feval(fname,tmpx,D,Extra,j);
            if (go == 1)
                duval = duval + dval * dUm(k,j);
            elseif (go == 2)
                duval = duval + dval * dUp(k,j);
            elseif (go == 3)
                if (dval > 0.0)
                   duval = duval + dval*dUp(k,j);
                else
                   duval = duval + dval*dUm(k,j);
               end
            end
        end
        if duval > 0
            duval = 0;
        end
    end
    fU(k,1)  =  lval;
    fU(k,2)  =  dlval;
    fU(k,3)  =  uval;
    fU(k,4)  =  duval;
end
for k=1:N
    if fU(k,1) == fU(k+1,1)
        fU(k,2) = 0;
        fU(k+1,2) = 0;
    end
    if fU(k,3) == fU(k+1,3)
        fU(k,4) = 0;
        fU(k+1,4) = 0;
    end
end
```

APPENDIX 2
Matlab Implementation of the Multiple Populations DE Algorithm: MPDE

```
function [fU, nGen, nfe, nvc] = ...
    FuzzyMPDE(fname,D,Extra,N,U,Pfact,strategy,reiter,replot,maxGen,gTol,noGen)
% Fuzzy Extension of a D-dimensional function defined by
%      f(x) = fname(x,D,Extra,k),
% using the Single Population differential evolution (SPDE) algorithm.
% The D-dimensional fuzzy argument U and result fU are in LU-fuzzy format:
%       U = (Um,dUm,Up,dUp;i=1:N+1) over N uniform alpha's subintervals
%           in 3-dimensional matrix form
%           U(i=1:N+1;j=1:D;k=1:4)
%           where    i  is the index of alpha-cut corresponding to
%                       alpha(i) = (i-1)/N. (i starts from 1 to N+1)
%                    j  is the index of a component of U
%                    k  is 1 for Um, 2 for dUm, 3 for Up, 4 for dUp
%                       so that  U(i,j,:) are the four LU-fuzzy parameters
%                       defining U(j) at i-th alpha-cut
%      fU = (fUm,dfUm,fUp,dfUp;i=1:N+1) over N uniform alpha's subintervals
%           so that  fU(i,:) are the four LU-fuzzy parameters defining
%           f(U) at i-th alpha-cut
%
% Output:
%    fU        Fuzzy Extended value f(U) in LU-fuzzy form
%    nf        number of function evaluations needed
%
% Inputs:
%    fname     string naming a function f(x,y) to minimize
%    D         number of parameters of the function
%    Extra     vector of extra values eventually needed to calculate f(x)
%    Pfact     factor for the number NP = Pfact*D of population members
%    maxGen    maximum number of generations (>= 200)
%
% Value function f(x) has to be defined by the user
% define fname.m as
%               function fv = fname(x,D,Extra,k)
%                  fv = function value f at x if k=0
%                  fv = partial first derivative of f at x wrt x(k) if k=1:D
%               end
%
% Internal Important Parameters:
%    F         DE-stepsize (in [0, 2])
%    CR        crossover probability (in [0, 1])
%    strategy    1 --> DE/best/1/exp           6 --> DE/best/1/bin
%                2 --> DE/rand/1/exp           7 --> DE/rand/1/bin
%                3 --> DE/rand-to-best/1/exp   8 --> DE/rand-to-best/1/bin
%                4 --> DE/best/2/exp           9 --> DE/best/2/bin
%                5 --> DE/rand/2/exp           else  DE/rand/2/bin
%    reiter      intermediate output will be produced after "reiter"
%                iterations. No intermediate output will be produced
%                if reiter is < 1
%    replot      same as for reiter, to plot the populations of level k=1
%
% Notes:
%       The code is adapted from
%             function devec3, (1997), by R. Storn (ICSI, Berkeley)
%       author: Luciano Stefanini, University of Urbino, Italy, 2006.
%               lucste@uniurb.it
%       A good initial guess is to choose F from interval [0.5, 1],
%       the crossover probability CR from interval [0, 1] helps to maintain
%       the diversity of the population and is rather uncritical. The
%       number of population members NP is also not very critical. A
%       good initial guess is 10*D. Depending on the difficulty of the
%       problem NP can be lower than 10*D or must be higher than 10*D
%       to achieve convergence.
%       If the parameters are correlated, high values of CR work better.
%       The reverse is true for no correlation.
%
% This program is free software; you can redistribute and/or modify it.
% It is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
```

```
% default values:
NP = Pfact*D;
F = 0.8;
CR = 0.5;

if (NP < 5)
   NP=5;
end
if (maxGen <= 0)
   maxGen = 200;
end
reiter = floor(reiter);
replot = floor(replot);
if ( D < 2  ||  D > 2 )
    replot = 0;
end

% Initialize arrays
Um  = zeros(N+1,D);
Up  = zeros(N+1,D);
dUm = zeros(N+1,D);
dUp = zeros(N+1,D);
for k=1:N+1
    for j=1:D
        Um(k,j)  = U(k,j,1);
        Up(k,j)  = U(k,j,3);
        dUm(k,j) = U(k,j,2);
        dUp(k,j) = U(k,j,4);
    end
end
fU  = zeros(N+1,4);


pop = zeros(2*NP,N+1,D);
bminval = zeros(1,N+1);
bmaxval = zeros(1,N+1);
lx  = zeros(1,D);
ux  = zeros(1,D);
% populations 1 to NP for lower branch and NP+1 to 2NP for upper branch
for k=1:N+1
    lx  = reshape(Um(k,:),1,D);
    ux  = reshape(Up(k,:),1,D);
    for i=1:2*NP
        pop(i,k,:) = lx + rand(1,D).*(ux - lx);
    end
end

nfe       = 0;            % number of function evaluations
nvcross   = 0;            % number of valid improvement across levels
% Evaluate initial populations
tmpx      = zeros(1,D);
for k=1:N+1
    for i=1:2*NP
        tmpx = reshape(pop(i,k,:),1,D);
        val(i,k) = feval(fname,tmpx,D,Extra,0);
        nfe  = nfe + 1;
    end
end
% identify best initial values for each cut
for k=1:N+1
    imin    = 1;
    imax    = 1;
    kmin    = k;
    kmax    = k;
    minval = val(1,k);
    maxval = val(1,k);
    for kk=1:N+1
        for i=1:2*NP
            % test if pop(i,kk,:) is feasible for level k
            feas = 1;
            if (kk < k)
                for j=1:D
                    if ( (pop(i,kk,j)<Um(k,j) ) || (pop(i,kk,j)>Up(k,j) ) )
                        feas = 0;
                    end
                end
            end
            if (feas == 1)      % if yes, use to find min and max
```

```
                if (val(i,kk) < minval)
                    imin   = i;
                    kmin   = kk;
                    minval = val(i,kk);
                end
                if (val(i,kk) > maxval)
                   imax   = i;
                   kmax   = kk;
                   maxval = val(i,kk);
                end
             end
        end
    end
    bminval(k)    = minval;           % best min value ever
    bmaxval(k)    = maxval;           % best max value ever
    minmemit(k,:) = pop(imin,kmin,:); % best min member of current iteration
    minvalit(k)   = minval;           % best min value of current iteration
    minmem(k,:)   = minmemit(k,:);    % best min member ever
    maxmemit(k,:) = pop(imax,kmax,:); % best max member of current iteration
    maxvalit(k)   = maxval;           % best max value of current iteration
    maxmem(k,:)   = maxmemit(k,:);    % best max member ever

end

%------popold is the population which has to compete. It is--------
%------static through one iteration. pop is the newly--------------
%------emerging population.----------------------------------------

% for min subpopulations
popoldmin = zeros(NP,D);        % toggle populations
pm1min = zeros(NP,D);              % initialize population matrix 1
pm2min = zeros(NP,D);              % initialize population matrix 2
pm3min = zeros(NP,D);              % initialize population matrix 3
pm4min = zeros(NP,D);              % initialize population matrix 4
pm5min = zeros(NP,D);              % initialize population matrix 5
bmmin  = zeros(NP,D);              % initialize bestmember  matrix
uimin  = zeros(NP,D);              % intermediate population of perturbed vectors
% for max subpopulations
popoldmax = zeros(NP,D);        % toggle populations
pm1max = zeros(NP,D);              % initialize population matrix 1
pm2max = zeros(NP,D);              % initialize population matrix 2
pm3max = zeros(NP,D);              % initialize population matrix 3
pm4max = zeros(NP,D);              % initialize population matrix 4
pm5max = zeros(NP,D);              % initialize population matrix 5
bmmax  = zeros(NP,D);              % initialize bestmember  matrix
uimax  = zeros(NP,D);              % intermediate population of perturbed vectors
mui = zeros(NP,D);              % mask for intermediate population
mpo = zeros(NP,D);              % mask for old population
rot = (0:1:NP-1);               % rotating index array (size NP)
rotd= (0:1:D-1);                % rotating index array (size D)
rt  = zeros(NP);                % another rotating index array
rtd = zeros(D);                 % rotating index array for exponential crossover
a1  = zeros(NP);                % index array
a2  = zeros(NP);                % index array
a3  = zeros(NP);                % index array
a4  = zeros(NP);                % index array
a5  = zeros(NP);                % index array
ind = zeros(4);

iter = 1;
noValid = 0;
while ((iter <= maxGen) & (noValid < noGen))
    improved = 0;
    ind = randperm(4);                % index pointer array
    a1  = randperm(NP);               % shuffle locations of vectors
    rt = rem(rot+ind(1),NP);          % rotate indices by ind(1) positions
    a2  = a1(rt+1);                   % rotate vector locations
    rt = rem(rot+ind(2),NP);
    a3  = a2(rt+1);
    rt = rem(rot+ind(3),NP);
    a4  = a3(rt+1);
    rt = rem(rot+ind(4),NP);
    a5  = a4(rt+1);

    for k=1:N+1           % for each level k=1:N+1

      % process lower and upper branches together, nested Min problems
      for i=1:NP
```

```
    popoldmin(i,:) = pop(i,k,:);     % save the old min population
    popoldmax(i,:) = pop(NP+i,k,:);  % save the old max population
end
pm1min = popoldmin(a1,:);       % shuffled population 1
pm2min = popoldmin(a2,:);       % shuffled population 2
pm3min = popoldmin(a3,:);       % shuffled population 3
pm4min = popoldmin(a4,:);       % shuffled population 4
pm5min = popoldmin(a5,:);       % shuffled population 5
pm1max = popoldmax(a1,:);       % shuffled population 1
pm2max = popoldmax(a2,:);       % shuffled population 2
pm3max = popoldmax(a3,:);       % shuffled population 3
pm4max = popoldmax(a4,:);       % shuffled population 4
pm5max = popoldmax(a5,:);       % shuffled population 5
for i=1:NP    % population filled with best member of last iteration
    bmmin(i,:) = minmemit(k,:);
    bmmax(i,:) = maxmemit(k,:);
end
mui = rand(NP,D) < CR;  % all random numbers < CR are 1, 0 otherwise
if (strategy > 5)
   st = strategy-5;            % binomial crossover
else
   st = strategy;                  % exponential crossover
   mui=sort(mui');         % transpose, collect 1's in each column
   for i=1:NP
      n=floor(rand*D);
      if n > 0
         rtd = rem(rotd+n,D);
         mui(:,i) = mui(rtd+1,i); %rotate column i by n
     end
   end
   mui = mui';                      % transpose back
end
mpo = mui < 0.5;                % inverse mask to mui
if (st == 1)                      % DE/best/1
   uimin = bmmin + F*(pm1min - pm2min);
   uimin = popoldmin.*mpo + uimin.*mui;
   uimax = bmmax + F*(pm1max - pm2max);
   uimax = popoldmax.*mpo + uimax.*mui;
elseif (st == 2)                  % DE/rand/1
   uimin = pm3min + F*(pm1min - pm2min);
   uimin = popoldmin.*mpo + uimin.*mui;
   uimax = pm3max + F*(pm1max - pm2max);
   uimax = popoldmax.*mpo + uimax.*mui;
elseif (st == 3)                  % DE/rand-to-best/1
   uimin = popoldmin + F*(bmmin-popoldmin) + F*(pm1min - pm2min);
   uimin = popoldmin.*mpo + uimin.*mui;
   uimax = popoldmax + F*(bmmax-popoldmax) + F*(pm1max - pm2max);
   uimax = popoldmax.*mpo + uimax.*mui;
elseif (st == 4)                  % DE/best/2
   uimin = bmmin + F*(pm1min - pm2min + pm3min - pm4min);
   uimin = popoldmin.*mpo + uimin.*mui;
   uimax = bmmax + F*(pm1max - pm2max + pm3max - pm4max);
   uimax = popoldmax.*mpo + uimax.*mui;
elseif (st == 5)                  % DE/rand/2
   uimin = pm5min + F*(pm1min - pm2min + pm3min - pm4min);
   uimin = popoldmin.*mpo + uimin.*mui;
   uimax = pm5max + F*(pm1max - pm2max + pm3max - pm4max);
   uimax = popoldmax.*mpo + uimax.*mui;
end
% force feasibility of current population for level k
for i=1:NP
    for j=1:D
        if ( uimin(i,j) < Um(k,j) )
            uimin(i,j) = Um(k,j);
        end
        if ( uimin(i,j) > Up(k,j) )
            uimin(i,j) = Up(k,j);
        end
        if ( uimax(i,j) < Um(k,j) )
            uimax(i,j) = Um(k,j);
        end
        if ( uimax(i,j) > Up(k,j) )
            uimax(i,j) = Up(k,j);
        end
    end
end

if (replot > 0)
```

```
 if (rem(iter,replot) == 0 && k==1)
     figure(3);
     xplot = zeros(NP);
     yplot = zeros(NP);
     xplot = reshape(uimin(:,1),1,NP);
     yplot = reshape(uimin(:,2),1,NP);
     subplot(1,1,1);
     plot(xplot,yplot,'or');
     drawnow; %---Draws current graph now
     figure(4);
     xplot = reshape(uimax(:,1),1,NP);
     yplot = reshape(uimax(:,2),1,NP);
     subplot(1,1,1);
     plot(xplot,yplot,'or');
     drawnow; %---Draws current graph now
 end
end

% Select which vectors are allowed to enter the new min/max population
for i=1:NP
   tmpxmin = uimin(i,:);
   tempvalmin = feval(fname,tmpxmin,D,Extra,0);   % check cost of competitor
   nfe     = nfe + 1;
   tmpxmax = uimax(i,:);
   tempvalmax = feval(fname,tmpxmax,D,Extra,0);   % check cost of competitor
   nfe     = nfe + 1;
   for kk=1:N+1   % consider all cuts for which new element is feasible
       feasmin = 1;
       feasmax = 1;
       if (kk > k)
           for j=1:D
               if ( tmpxmin(j) < Um(kk,j) || tmpxmin(j) > Up(kk,j) )
                   feasmin = 0;
               end
               if ( tmpxmax(j) < Um(kk,j) || tmpxmax(j) > Up(kk,j) )
                   feasmax = 0;
               end
           end
       end
       if ( feasmin == 1 )
          if (bminval(kk) > tempvalmin)
              if (tempvalmin + gTol < bminval(kk))
                  improved = 1;
              end
              bminval(kk) = tempvalmin;
              minmem(kk,:) = tmpxmin;
              nvcross = nvcross + 1;
          end
          if (bmaxval(kk) < tempvalmin)
              if (tempvalmin - gTol > bmaxval(kk))
                  improved = 1;
              end
              bmaxval(kk) = tempvalmin;
              maxmem(kk,:) = tmpxmin;
              nvcross = nvcross + 1;
          end
          if (tempvalmin < val(i,kk)) % if competitor is better
             pop(i,kk,:) = tmpxmin;    % replace old vector (for new iteration)
             val(i,kk)   = tempvalmin;
          end
          if (tempvalmin > val(NP+i,kk)) % if competitor is better
             pop(NP+i,kk,:) = tmpxmin;    % replace old vector (for new iteration)
             val(NP+i,kk)   = tempvalmin;
          end
       end
       if ( feasmax == 1)
          if (bmaxval(kk) < tempvalmax)
              if (tempvalmax - gTol > bmaxval(kk))
                  improved = 1;
              end
              bmaxval(kk) = tempvalmax;
              maxmem(kk,:) = tmpxmax;
              nvcross = nvcross + 1;
          end
          if (bminval(kk) > tempvalmax)
              if (tempvalmax + gTol < bminval(kk))
                  improved = 1;
              end
```

```
                    bminval(kk) = tempvalmax;
                    minmem(kk,:) = tmpxmax;
                    nvcross = nvcross + 1;
                end
                if (tempvalmax > val(NP+i,kk)) % if competitor is better
                   pop(NP+i,kk,:) = tmpxmax;    % replace old vector (for new iteration)
                   val(NP+i,kk)   = tempvalmax;
                end
                if (tempvalmax < val(i,kk)) % if competitor is better
                   pop(i,kk,:) = tmpxmax;    % replace old vector (for new iteration)
                   val(i,kk)   = tempvalmax;
                end
             end
          end
       end            % end for member i=1:NP
       minmemit(k,:) = minmem(k,:);  % freeze the best member of this iteration for the coming
                                     % iteration. This is needed for some of the strategies.
       maxmemit(k,:) = maxmem(k,:);  % freeze the best member of this iteration for the coming
                                     % iteration. This is needed for some of the strategies.
   end    % next level

   % Intermediate Output section
   if (reiter > 0)
      if (rem(iter,reiter) == 0)
          fprintf(1,'\nGeneration: %d,  Function Evaluations: %d\n',iter, nfe);
          for k=1:N+1
              fprintf(1,'Level: %d,  BestMin: %f,  BestMax: %f\n',k-1,bminval(k),bmaxval(k));
          end
      end
   end

   noValid = noValid + 1;
   if (improved == 1)
       noValid = 0;
   end
   iter = iter + 1;
end      % next generation

nGen = iter - 1;
nvc = nvcross;

% Construct result fuzzy number fU
iter = 1;
while (iter > 0)
    iter = 0;
    for k = 1:N
        if  ( bminval(k+1) < bminval(k) )
            iter = 1;
            bminval(k) = bminval(k+1);
            minmem(k,:) = minmem(k+1,:);
        end
        if  ( bmaxval(k+1) > bmaxval(k) )
            iter = 1;
            bmaxval(k) = bmaxval(k+1);
            maxmem(k,:) = maxmem(k+1,:);
        end
    end
end

TolBound = 0.0001;
for k=1:N+1
    dlval = 0.0;
    lval = bminval(k);
    tmpx = minmem(k,:);
    for j=1:D
        go=0;
        if ( tmpx(j) < Um(k,j) + TolBound )
            go = go+1;
        end
        if ( tmpx(j) > Up(k,j) - TolBound)
            go = go+2;
        end
        if (go>0)
            dval = feval(fname,tmpx,D,Extra,j);
            if (go == 1)
                dlval = dlval + dval * dUm(k,j);
            elseif (go == 2)
                dlval = dlval + dval * dUp(k,j);
```

```
            elseif (go == 3)
                if (dval > 0.0)
                   dlval = dlval + dval*dUm(k,j);
                else
                   dlval = dlval + dval*dUp(k,j);
               end
            end
        end
        if dlval < 0
            dlval = 0;
        end
    end

    duval = 0.0;
    uval = bmaxval(k);
    tmpx = maxmem(k,:);
    for j=1:D
        go=0;
        if ( tmpx(j) > Um(k,j) - TolBound )
            go = go+2;
        end
        if ( tmpx(j) < Up(k,j) + TolBound)
            go = go+1;
        end
        if (go>0)
            dval = feval(fname,tmpx,D,Extra,j);
            if (go == 1)
                duval = duval + dval * dUm(k,j);
            elseif (go == 2)
                duval = duval + dval * dUp(k,j);
            elseif (go == 3)
                if (dval > 0.0)
                   duval = duval + dval*dUp(k,j);
                else
                   duval = duval + dval*dUm(k,j);
               end
            end
        end
        if duval > 0
            duval = 0;
        end
    end
    fU(k,1)  =  lval;
    fU(k,2)  =  dlval;
    fU(k,3)  =  uval;
    fU(k,4)  =  duval;
end
for k=1:N
    if fU(k,1) == fU(k+1,1)
        fU(k,2) = 0;
        fU(k+1,2) = 0;
    end
    if fU(k,3) == fU(k+1,3)
        fU(k,4) = 0;
        fU(k+1,4) = 0;
    end
end
```