University of Ostrava

Institute for Research and Applications of Fuzzy Modeling

Error Optimization of Fuzzy Transform Using Evolution Algorithms

František Huňka and Viktor Pavliska

Research report No. 135

2008

Submitted/to appear:

Information Sciences

Supported by:

Project MSM6198898701 of MŠMT ČR.

University of Ostrava
Institute for Research and Applications of Fuzzy Modeling
30. dubna 22, 701 03 Ostrava 1, Czech Republic

tel.: +420-59-7091401 fax: +420-59-6120478
e-mail: frantisek.hunka@osu.cz

Error Optimization of Fuzzy Transform Using Evolution Algorithms

Frantisek HUNKA*

Department of Informatics and Computers, University of Ostrava, 30. dubna 22, Ostrava 1

Viktor PAVLISKA

Institute of Research and Application of Fuzzy Modeling, University of Ostrava, 30. dubna 22, Ostrava 1

Abstract

Fuzzy transform is a transform that considerably simplifies solution of differential or integral-differential equations, by transforming them into a n-dimensional vectors that are easier to solve. The inverse fuzzy transform transforms the achieved results back into the original domain creating thus an approximated function of the original one. As the other approximations it also works with some level of error. With the F-transform the level of error is mainly caused by the distribution of the nodes in the searching area. Before the fuzzy transform starts, the universe has to be partitioned by the nodes. The main aim of the article is to propose and verify optimization of node distribution with respect to objective functions such that the difference between original and approximated function would be minimal. The article further discusses other possibilities of the distribution of the nodes as well as different heuristics, which may be used in finding the best distribution of the nodes so as the approximation error would be minimal. By this way the approximated function would be more closer to the original one.

Key words: fuzzy transform, fuzzy approximation, evolution algorithm, error optimization, differential evolution.

*Corresponding author

Email addresses: frantisek.hunka@osu.cz (Frantisek HUNKA), viktor.pavliska@osu.cz (Viktor PAVLISKA)

2008 MSC: 92D15, 78M50

1. Introduction

The main areas of the successful usage of fuzzy transform (F-transform for short) are filtering – removing noise, smooth logical deduction, time series prediction and image compression. Our research has been concerned only with one-dimensional F-transform. The main benefit of using fuzzy transform is that it converts an integrable function on $[a; b]$ into a n-dimensional vector. This approach enables to use methods of linear algebra to solve the problem instead of e.g. computation of definite integral or solution of differential equations. That is why there is a endeavor to make F-transform more precise more accurate.

The structure of the paper is following. In Section 2, the essence of fuzzy transform is given. In Section 3, basic properties of evolution algorithms are described with the focus on differential evolution and contest of heuristics. In Section 4, the process of optimization itself is explained and described. In Section 5, the optimization framework, which was used for F-transform optimization is described. Achieved results are described and explained in Section 6. In Section 7, all advancements and given results are discussed. Section 8, contains conclusion and further recommendation.

2. Fuzzy Transform

F-transform as mentioned earlier is a very effective tool that can be used as a bridge between fuzzy and classical mathematics because it converts classical mathematics domain into the fuzzy mathematics domain and visa versa. It is composed, as the other transform, of direct transform and inverse transform. Direct F-transform converts integrable function on $[a; b]$ into n-dimension vector. The inverse F-transform converts an n-dimension vector into a specially represented continuous function, which approximates the original one. The advantage of the F-transform is that the inverse F-transform produces a simple and unique approximate representation of the original function. Moreover this advancement considerably simplifies the solution of the original problem as the computation of a definite integral or solution of differential equations and transfers it to the solution of n-dimension vector space, which can be solved using methods of linear algebra.

2.1. Fuzzy Partition of the Universe – Establishing Nodes

The interval $[a; b]$ is taken as a universe. This interval represents a common domain for all (real-valued) functions we are working with. The fuzzy partition of the universe is given by fuzzy subsets of the universe $[a; b]$ with given properties. By partitioning the common domain a series of so called nodes is given. These nodes are denoted as $x_1 < \cdots < x_n$ within $[a; b]$, such that $x_1 = a, x_n = b$ and $n \geq 2$. For each node x_i there is a corresponding fuzzy set denoted A_i with corresponding membership function $A_i(x)$. Detail definition that fuzzy sets and corresponding nodes have to fulfill are given at [3, 4]. The most important of these conditions is fulfilling so called Ruspini condition of orthogonality:

$$(\forall x \in [a; b]) : \sum_{k=1}^{n} A_k(x) = 1.$$

The membership functions $A_1(x), \ldots, A_n(x)$ are called *basic functions.* Basic functions are specified by a sequence of nodes $x_1, \ldots, x_n$ and further properties.

The F-transform is also influenced by a actual number of the nodes. This problem is discussed among others in [4]. The conclusion of the experiments is the more nodes the better adjustment of the original function to the function given by the inverse F-transform. By the simple experiment there may be seen that the small change of the node position call a change of the inverse transform and by this way in the approximated function.

2.2. Discrete F-Transform

There are both continuous and discrete F-transform. But due to experiments, which are made using discrete direct and inverse form of F-transform the article is focused on the discrete form. The other reason for using discrete F-transform is that with the practical experiments mostly discrete F-transform is used. In practice, the continuous case is frequently replaced by the discrete one in the following way. The original function $f(x)$ is replaced by points $p_1, \ldots, p_l \in [a; b]$ and their corresponding functional values. The number of the points used to be much greater than the number of later established nodes by the partition of the universe. This type of F-transform has wider practical use than continuous F-transform as many functional dependencies can be simply given by the pair composed of a point and corresponding functional value. However n-tuple of real numbers, so called *components*, denoted as $[F_1, \dots, F_n]$ is calculated according a slightly different formula as follows:

$$F_k = \frac{\sum_{j=1}^{l} f(p_j)A_k(p_j)}{\sum_{j=1}^{l} A_k(p_j)} \qquad \text{for } k = 1, \dots, n$$

The discrete F-transform is in any case simpler than the integral one. Moreover a certain sequence of the inverse discrete F-transform converges to an original function at all given points. Therefore, all computational algorithms may be based on the discrete type of F-transform. All experimental results given in this paper were calculated using discrete type of F-transform.

The discrete F-transform has its inverse form too. However there is no difference in mathematical expression (formalism) of this inverse form both for the continuous and the discrete form. The inverse F-transform fulfils the best approximation criterion, which can be called the piecewise integral least square criterion. The following function

$$f_{F,n}(x) = \sum_{k=1}^{n} F_k A_k(x)$$

is called the *inverse F-transform.* For each x point $x \in [a; b]$ we can calculate value of inverse F-transform, which will be compared with the original value. The result of the inverse function is as a sum through all nodes that multiply component of the F-transform with basic function (membership function).

3. Error Optimization

The main aim of our aspiration is to minimize the differences between the original function and its approximated form obtained by the F-transform. In order to formalize the optimization problem different types of objective functions can be used. By using objective functions we can exactly measure the differences between two sets of points representing the original and approximated function and also use tools (methods) optimizing the difference between the original and approximated functions.

Definition1 (Approximation error). Let f is a given function and f^A its approximation on the interval $[a; b]$. The error of approximation is defined such as maximal difference between the original function and its approximation:

$$\varepsilon = \bigvee_{x \in [a;b]} |f(x) - f^A(x)|$$

Optimization problems are ubiquitous in science and engineering. However it is not so easy to solve a difficult optimization problem. Therefore there is a demand for global optimization methods that should be simple to implement, easy to use, reliable and fast. Relatively for a long time the optimization problems were exclusively solved by classical mathematical methods. This approach enables to find global extremes for the simpler optimization problems and mostly local extremes for more difficult optimization problems. With this traditional approaches the computing complexity and in some way even awkwardness, increase not only with the growing complexity of the problem but also with the scope of the arguments of the objective functions. The range of these arguments can cover both different numerical types (integer, double, enumerated, logic) but also intervals, in which the given arguments can vary.

For this more difficult optimizing problems a set of evolution algorithms emerged and was led to perfection. Among their benefits belong also that the user of the optimizing problem “can not” know classical optimizing methods but he is only required to have a good knowledge of the problem domain and the ability to define the objective function correctly. Among the further advantages of this approach can also be added the fact that evolution algorithms aim to find global extremes rather than local ones and they usually provide the user with more than one solutions.

The drawbacks involve the fact that evolution algorithms partly work with random, which causes the results will not be precisely estimated beforehand. Therefore the experience with the implementation of evolution algorithms plays an important role. On the other hand achieved results proved their applicability.

3.1. Optimization of Approximation Obtained by Inverse F-transform

Optimization process of approximation is based on the changing of the positions of the inner nodes. The number of nodes remains fixed for every optimization study. Actually there are no explicit conditions or limits on the nodes distribution except for being out of given interval representing the universe and being two or more nodes in the same position.

For these reasons the most perspective optimizing algorithms must be some of the family of the evolutionary optimizing algorithms. They are very effective for this class of the optimizing problems but some of them are too sophisticated mainly in setting their control parameters. Our aim was to use simple, reliable and effective optimizing algorithm. That is why we decided for differential evolution. The detail procedure of the algorithm will be described in the following section.

3.2. Differential Evolution

The differential evolution (DE) described in the book [5] has become one of the most popular algorithms for continuous global optimizing problems in the last decade. But in many cases the efficiency of the search for the global minimum is very sensitive to the setting of its control parameters. One of the means that eliminates this problem is self-adaptive DE. We used tested procedure called differential evolution with competitive control-parameter setting described in the papers [6, 7]. The essence of this approach is that the setting of the control parameters can be made adaptive through the implementation of a competition in differential evolution. Each setting has a given probability that is changing during the process of optimization. On the base of these probabilities the parameter settings is chosen for each evolutional step.

The competition provides an self-adaptive mechanism of setting control parameter appropriate to the problem actually solved. In the application of this algorithm there is necessary to have a different settings of values $F > 0$ (influencing mutation) and $C \in [0; 1]$ (affecting crossover) among them at random a choice is made with probability $q_i, i = 1, 2, ..., h$. The probabilities can be changed according to the success of the setting in preceding steps of the search process. The i-th setting is successful if it generates trial point y belonging to member x for which $f(y) < f(x)$. The probability q_i can be evaluated as the relative frequency

$$q_i = \frac{n_i + n_0}{\sum_{j=1}^{h}(n_j + n_0)},$$

where $n_0 > 0$ is a constant.

By [6] is supposed that such a competition of a different settings will prefer a successful setting. The competition will serve as an adaptive mechanism of setting control parameters suitable for the problem that is actually solved.

4. Application of the Differential Evolution Algorithm

There are three issues to be solved when applying evolution algorithms.

1. Representation and coding of the population member
2. Definition of objective function
3. Heuristics for initial population generation

Population member represents possible solution of the optimization problem. In our case the solution is given by the distribution of nodes defining the partition of the universe. The nodes themselves are coded into a vector, which dimension equals to the number of inner nodes. Objective function is in our case determined by an error of approximation, see def. 1. By our heuristics, initial population is composed of 16 members divided into three groups. The first group is made up only by 1 member representing equidistant distribution of nodes (uniform partition). The second group is formed by 5 random members, which means that their inner nodes are distributed completely at random. The third group contains 10 members that proceeds from slightly modified equidistant distribution. This modification can be explicitly expressed by rules of nodes movement. The nodes can move horizontally on the x-axes in the left (negative sign) and in the right (positive sign) of the axe. In order to mark node's movement relative to the physical position we use a percent expression. The distance between two neighboring nodes will be in the initial step 100%. So in this way -100% means a move of the node to the position of the left neighboring node and vise versa $+100\%$ means a move of the node to the position of the right neighboring node. In addition we introduced some restriction about the move of the nodes as we want to avoid some chaotic movement in this initial phase of the optimization.

1. The nodes will move but they will not "over-jump" each other. The sequence of the nodes remain unchangeable they can move only in their neighborhood.
2. The bordering (terminal) nodes are unmovable.
3. To support evolution progress of optimization we heuristically determined that the maximal move in one step will be only 50% either to the left or to the right side.

That are all restrictions we determined for initial population.

In DE algorithm the inner nodes can move within the whole interval on the x axe. After generating initial population the further populations are evolved following the rules of the DE modified by competition of heuristics. The aim of the optimization is to find such a position of the nodes (they will not be equidistant) that the objective function will be minimum.

The number of generated members in each population as well as the number of nodes, which remains the same for the whole process of optimization, is given as input parameters for evolution algorithm.

Search space is n-dimensional where n is equal to the number of declared nodes. For each single node there is one dimension of the search box. Each node is associated with a single dimension (intervals are identical so far).

5. Optimization Framework

As we intend to study and examine the optimizing problems thoroughly we designed and implemented optimization framework. The aim of the framework is to cover broader set of optimizing algorithms based on populations of possible solutions in the form of population members. These population members influence each other their quality in iterations (derived from evolutionary principles), which are usually called in the context of evolution algorithms as generations. The aim of the whole process is to find the best solution in the scope of the defined objective function.

To achieve flexibility and adaptability of the framework we decided to proceed gradually from the abstract levels to the more specific level of the framework. Layered design proved to be effective and perspective method of solution [1, 2]. The framework itself was designed in two layers, the *abstract* layer and the *implementation* layer. There is also the third layer used by the user, in our design called the *application* layer. This layer utilizes the two bottom layers for using a genetic based algorithms for different purposes.

5.1. Abstract Layer

Analyzing the problem domain and the demands of optimization we identified five basic abstract classes characterizing the framework. This layer comprises three principle classes and two auxiliary classes for search space description in the form of box constraints.

The two basic abstract classes are the *Algorithm* and the *Problem* classes. The task of the first one is to describe optimizing algorithms the other one represents the problem that should be optimized. There is association between these two classes and class *Problem* is accessible from the *Algorithm* class, which corresponds to the procedure of the optimization computing.

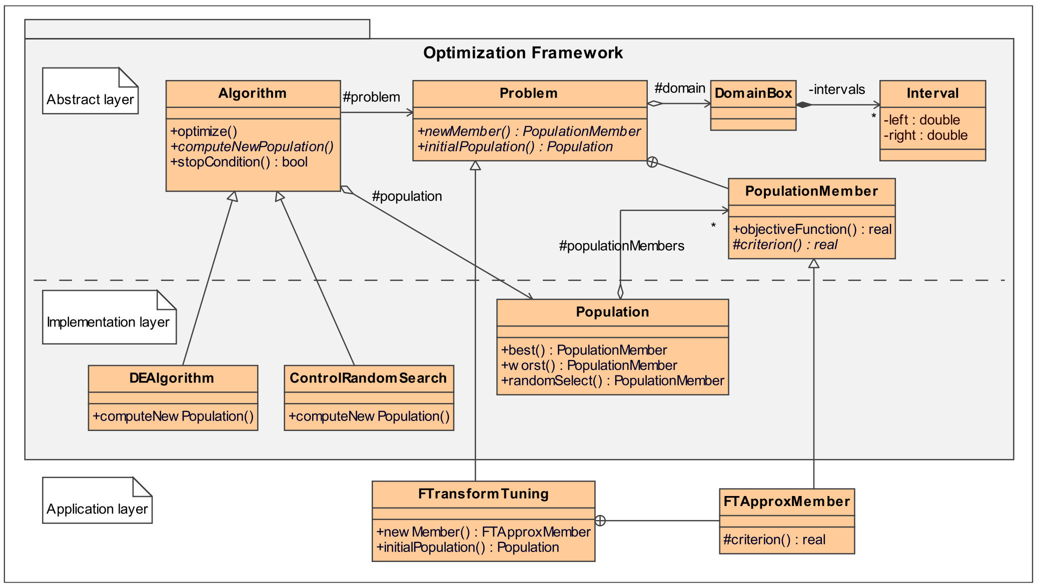

Figure 1: Optimization Framework – UML Class Diagram

Abstract class *Algorithm* declares two methods. The *optimize()* methods, which manages the whole optimization process and the *computeNewPopulation()* method, which is responsible for the computing of *new population* representing next generation. Both methods are abstract so they will be overridden or further extended in their more specific subclasses. The family of genetic algorithms also needs explicitly declared *stop condition*, which terminates computing iteration. In the current version of the framework the *stopCondition()* was designed as an attribute of the Algorithm class. This solution can not be the definite one as the *stop condition* could express more complex logic expressions for terminating computing process, given by combining different types. In this cases *stop condition* should be designed as a self independent class with an association to the Algorithm class.

The main aim of the *Problem* abstract class is to declare *newMember()* and *initialPopulation()* methods. The *newMember()* represents the factory method creating adequate instance depending on solving optimizing task. The *initialPopulation()* method is responsible for creating an initial population. Both methods are abstract and will be further extended in their subclasses.

Main function of the *PopulationMember* class is to declare objective function in the form of *objectiveFunction()* method, which is necessary for evaluating individuals in the population. This method utilizes abstract method *criterion()*, which is further specialized in the application layer by the user. In this way the user can influence evaluation of fitness of each population member.

The area in which the global extremes are looking for is declared by the *DomainBox* class, which precisely describes searching space. The *Problem* class aggregates the *DomainBox* class. Aggregation is relationship that expresses the fact that the *DomainBox* class can be replaced by a different more specific class regarding the solving problem. Class *DomainBox* usually represents an n-dimensional searching space. The single dimension of the searching space is described by the *Interval* class, which bounds one dimension of the space. The relationship between the *DomainBox* class and the *Interval* class is composition.

The structure of the whole framework using the UML class diagram can be seen in the figure 1.

5.2. Implementation Layer

This layer is created by two sorts of more specific classes. The first one is the *Population* class that actually works as a container of population members. Its methods are used for finding the best or the worst population member regarding to the value of their objective function and selection the population member at random. The *randomSelect()* method is utilized in the core of evolution operation of mutation.

The second sort forms a family of subclasses of the *Algorithm* class. They specify in a detail way the particular optimization algorithm on the basis of optimizing approach. Classes differ in the implementation of the *computeNewPopulation()* method declared in the superclass *Algorithm* as abstract. It is supposed that with the enlargement of the optimizing framework the number of subclasses will increase in order to implement additional optimizing approaches.

The *ControlRandomSearch* class implements rather simple *computeNewPopulation()* method based on the fact that the worst population member, evaluated by the given objective function, is replaced by the new better one. This method is also known as the *worst-member-substitution* method.

The *DEAlgorithm* class implements algorithms of differential evolution mostly used in practical testing of this approach, which were described in section 3.2.

5.3. Application Layer

Application layer represents classes that the user utilizing the optimizing framework has to implement by himself. In this way the user defines his specific requirements for the optimizing task. Therefore this layer does not actually belong to the optimizing framework. In our case the layer is composed of the *FTransformTuning* class, which is a subclass of the *Problem* class and the *FTApproxMember* class, which is a subclass of the *PopulationMember* class. As can be seen from the figure 1, the *FTApproxMember* class is an inner class of the *FTransformTuning* class. The construction of the inner (sometimes also called nested) class is in this case more natural and effective than the other ones.

The *FTApproxMember* implements the *criterion()* method, which is used together with the *objectiveFunction()* method to evaluate members of population. The *criterion()* method actually implements the specific function, by which the *objectiveFunction()* completes its evaluation. By this way the *criterion()* method can be easily replaced by other specific function.

The *FTransformTuning* class further specializes (extends) the *newMember()* and *initialPopulation()* methods, originally declared in the superclass. Both extensions are done with the aim to complete required optimizing demands.

6. Achieved Results

First of all we would like to illustrate in the form of figures results achieved applying F-transform with uniform partition (without optimization) see Fig. 2 and result achieved after optimization see Fig. 3. The whole number of the inner nodes was reduced to seven to make the distribution of nodes more visible. Comparing both figures it can be seen how the move of the inner nodes can influence the whole transformation.

The results obtained by experiments are shown in the two following tables. Table 1 gives results with respect to the required precision while Table 2 gives results with respect to the constant number of generations. The number of population members in all generations remains the same for all experiments and equals to 16. This number can be seen in the Table 1 in the column *Avg No of Objective Function Evaluation*, in cases where the average number of generations is equal to 1, which represents only initial population. It happens when the number of the inner nodes is inadequately big with respect to required precision, which leads to fulfilling stop condition even for equidistant

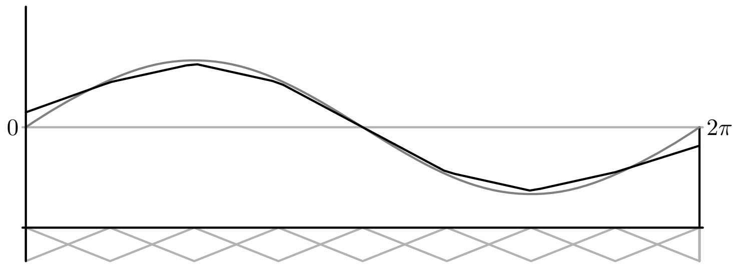

Figure 2: F-transform with uniform partition

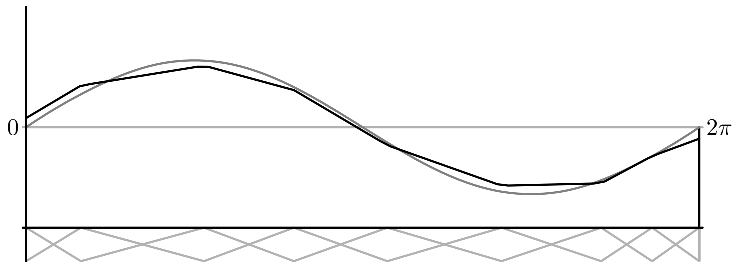

Figure 3: F-Transform with optimized partition

partition that is included in initial population. This was mentioned in the description of its construction in the section 4. The number of points also remains the same for all experiments and it equals to 62. However the number of nodes varies from 20 nodes to 50 nodes.

The results given in the Table 1 determine the success rate regarding the given precision. Each optimization study is in the separate line marked as *study n* and relates to one task optimized by one algorithm. Optimization experiment represents one run of the optimization task. Optimizing study represents statistical evaluation of the optimization experiments – in our case 100 experiments. That is why the average values (Avg) are used in the heading of the Table 1. From the experiments it can be judged that for each number of nodes there is a restricted precision, which can be achieved. We can see that with the growing number of nodes the higher precision of optimization can be received.

The results given in the Table 2 show how the number of inner nodes influences achieved precision. The precision is given both in the form of the average achieved precision (last but one column of the table) and in the form of the best achieved precision (last column of the table). During the experiments the number of generations remains constant, equals to 10 000. The number of objective function evaluation is also the same for all performed studies of experiments.

Table 1: Experimental results with respect to required precision

Optimization study no	No of inner nodes	Avg no of generations	Avg no of objective function evaluation	Success rate	Required precision
Study 1	20	–	–	0%	0.01
Study 2	20	–	–	0%	0.02
Study 3	20	1373	21986	84%	0.03
Study 4	20	520	8344	97%	0.04
Study 5	30	–	–	0%	0.01
Study 6	30	1784	28575	94%	0.02
Study 7	30	396	6355	95%	0.03
Study 8	30	59	974	100%	0.04
Study 9	40	4887	78208	3%	0.01
Study 10	40	469	7532	100%	0.02
Study 11	40	66	1078	100%	0.03
Study 12	40	1	16	100%	0.04
Study 13	50	1396	22366	85%	0.01
Study 14	50	132	2139	100%	0.02
Study 15	50	1	16	100%	0.03
Study 16	50	1	16	100%	0.04

7. Discussion

In the presented experiments it was proved that the distribution of nodes plays a crucial role in the F-transform. Uniform distribution of nodes is taken somehow as a standard distribution and in our view as a starting point for further improvements of the F-transform. Using evolution algorithms in the scope of the optimization framework proved its ability to solve such problems. As was mentioned in the chapter dealing with evolution algorithms, the most

Table 2: Experimental results with respect to constant number of generations

Optimization study no	No of inner nodes	No of generations	No of objective function evaluation	Avg achieved precision	Best achieved precision
Study 1	20	10 000	160 032	0.0258	0.0226
Study 2	30	10 000	160 032	0.0162	0.0103
Study 3	40	10 000	160 032	0.0130	0.00798
Study 4	50	10 000	160 032	0.0089	0.00639

important is to have a good knowledge of the problem domain and the ability to define the objective function correctly.

Concerning the first point we set as members of initial population different node distributions (coded into vectors of nodes partition distribution). We used our own heuristics by setting one node distribution with the uniform distribution. The second group of the node distributions was created completely at random and finally the third group was generated using equidistant distribution of nodes, which can be moved at random up to $\pm 50\%$ towards their neighboring nodes. In terms of evolution algorithms this phase is called initial population.

The initial distribution of nodes expressed through initial population fulfills all requirements of the evolution algorithms but does not fully takes into account the behavior of the original real function itself. We intuitively think that the function behavior should be taken into account too in the phase of initial population and that it could influenced the results of the optimization process. For this reason it could be advisable somehow formalize the function behavior and use it for initial population heuristics.

From the simple view it is clear that places, where the function is linear or even constant there is no need to place nodes densely. In short the more non-linear function behavior is the more nodes are necessary. From the mathematical point of view in places, where the function behavior noticeably differ from the linear behavior, the values of the second derivations are distinctively different from zero. We can say that the measure of the nodes density should reply the absolute value of the second derivation.

The second point that is necessary for proper using of evolution algorithms concerns the correct objective function definition. For our purpose we defined maximal deviation among discrete points expressing the original function, which is a consequence of the definition 1.

The results of the error optimization of the F-transform should be used in more complex tasks connecting with utilizing the whole F-transform such as filtering-removing noise, smooth logical deduction and image compressing. To use the results properly there is a need to safely store optimized node distribution. For each of the nodes there are two values to be stored. The first one is the x-axis position (could be e.g. expressed relatively regarding to the left neighboring node and the right bordering node) and value of the component in the node. This can be solved by using simple list or map of these values, which may be stored either in a compressed sequential file or in a persistent object in the object oriented environment or in an array of values in case of relation database environment.

8. Conclusions

The article proves that the better results of the F-transform can be achieved by its optimization. The task of the optimization is to find the better inner nodes distribution so that the objective function (error of approximation) is minimal. In the section *Discussion* further possible improvements are indicated. Except for these improvements we also plan to extend optimization to approximation of n-dimensional functions too.

Acknowledgements:
This research has been supported by the project MSM6198898701 of the MŠMT ČR.

References

[1] F. Hunka, Object-oriented Modeling in Cluster Analysis, In *Proceedings of 28th ASU Conference The Simulation Languages,* Brno 2002, pp. 71–78.

[2] F. Hunka, V. Pavliska, Object Oriented Approach in Optimization of Fuzzy Transform, In *Proceedings of 12th WSEAS International Conference on Computers,* Heraklion, Greece 2008, pp. 1066–1071.

[3] I. Perfiljeva, Approximating models based on fuzzy transforms, In *Joint EUSFLAT-LFA Conference,* 2005, Barcelona, Spain, Universitat Politecnica de Catalunya & EUSFLAT, 2005, pp. 645–650.

[4] I. Perfiljeva, Fuzzy Transform – a Powerful Tool in Modelling, In *East West Fuzzy Colloquium,* Univ. of Applied Sciemces, Zittau, 2006, pp. 2–3.

[5] K.–V. Price, R.–M. Storm and J.–A. Lampinen, *Differential Evolution,* Springer Verlag, Berlin–Heidelberg, 2005

[6] J. Tvrdik, Adaptive Differential Evolution: Application to Nonlinear Regression, In *Proceedings of the International Multiconference on Computer Science and Information Technology,* Vol. 1, No. 2, 2007, pp. 193–202.

[7] J. Tvrdik and I. Krivy, Competitive Self-Adaptation in Evolutionary Algorithms, In *5th Conference of European Society for Fuzzy Logic and Technology,* 2007, Ostrava, University of Ostrava, 2007, pp. 251–258.