On PSO-Based Approximation of Zadeh's Extension Principle

Jiří Kupka and Nicole Škorupová(✉)

CE IT4I - IRAFM, University of Ostrava, 30. dubna 22, Ostrava, Czech Republic
{Jiri.Kupka,Nicole.Skorupova}@osu.cz

Abstract. Zadeh's extension is a powerful principle in fuzzy set theory which allows to extend a real-valued continuous map to a map having fuzzy sets as its arguments. In our previous work we introduced an algorithm which can compute Zadeh's extension of given continuous piecewise linear functions and then to simulate fuzzy dynamical systems given by them. The purpose of this work is to present results which generalize our previous approach to a more complex class of maps. For that purpose we present an adaptation on optimization algorithm called particle swarm optimization and demonstrate its use for simulation of fuzzy dynamical systems.

Keywords: Zadeh's extension · Particle swarm optimization · Fuzzy dynamical systems

1 Introduction

Zadeh's extension principle plays an important role in the fuzzy set theory. Mathematically it describes a principle due to which each map $f\colon X \to Y$ induces a map $z_f\colon \mathbb{F}(X) \to \mathbb{F}(Y)$ between some spaces of fuzzy sets $\mathbb{F}(X)$ (resp. $\mathbb{F}(Y)$) defined on X (resp. Y).

In general the calculation of Zadeh's extension principle is a difficult task. This is caused mainly by difficult computation of inverses of the map f. Only some cases, e.g. when f satisfies some assumptions like monotonicity, one can find an easier solution. Consequently, the problem of approximation of the image of a fuzzy set $A \in \mathbb{F}(X)$ under Zadeh's extension z_f has been attracted by many mathematicians. For example, in [2] and [3] the authors introduced a method approximating Zadeh's extension $z_f(A)$ which is based on decomposition of a fuzzy set A and multilinearization of a map f. Later in [1] another method using an optimization algorithm applied to α-cuts of a chosen fuzzy set was proposed and tested. Further some specific representations of fuzzy numbers have also been used. For example, a parametric LU-representation of fuzzy numbers

The support of the grant "Support of talented PhD students at the University of Ostrava" from the programme RRC/10/2018 "Support for Science and Research in the Moravian-Silesian Region 2018" is kindly announced.

M.-J. Lesot et al. (Eds.): IPMU 2020, CCIS 1239, pp. 267–280, 2020.
https://doi.org/10.1007/978-3-030-50153-2_20

was proposed and elaborated in [5] and [15]. The authors claim that the LU-fuzzy representation allows a fast and easy simulation of fuzzy dynamical systems and, thus, it avoids the usual massive computational work. However this method is restricted for fuzzy numbers only. Further, the authors of [14] presented another simple parametric representations of fuzzy numbers or intervals, based on the use of piecewise monotone functions of different forms. And finally, another more general procedure allowing to approximate Zadeh's extension of any continuous map using the F-transform technique was introduced in [9].

We contributed to the problem above in [10] where we introduced an algorithm which can compute Zadeh's extension of a given continuous piecewise linear map and, consequently, to simulate a fuzzy dynamical system given by this map. We first focused on continuous one-dimensional piecewise linear interval maps and piecewise linear fuzzy sets. These assumptions allowed us to precisely calculate Zadeh's extension for the class of piecewise linear fuzzy sets, for which we do not necessarily assume even the continuity. This feature should be considered as an advantage of our approach because discontinuities naturally appear in simulations of fuzzy dynamical systems. Still the algorithm proposed in [10] covered a topologically large, i.e. dense, class of interval maps.

The aim of this contribution is to extend the use of our previous algorithm (from [10]) for a more complex class of maps, namely, for the class of continuous interval maps. In order to do this, we intend to linearize a given map f, which is an approximation task leading to an optimization problem (of the determination of appropriate points of the linearization) minimizing the distance between the original function f and its piecewise linear linearization $\tilde{f}$. As there is no feasible analytical solution of the minimization problem, we can approach it from the perspective of the stochastic optimization. For that purpose, we chose the particle swarm optimization algorithm (PSO) that helps us to find appropriate distributions of points in a given space defining piecewise linear approximations as close as possible to the original function f. PSO is one of the most known stochastic algorithms from the group of swarm algorithms. We considered this algorithm due to several reasons. For example, it was the best among algorithms used for a similar task e.g. in [13]. Of course, there are naturally other options to be considered and deeper analysis in this direction is in preparation. Due to page limit we present several preliminary observations only, although we have prepared much more tests.

The structure of this paper is the following. In Sect. 2, some basic terms and definitions used in the rest of this manuscript are introduced. In Sect. 3, we introduce a modification of the original PSO algorithm applied to the problem mentioned in the previous paragraph and then we demonstrate the linearization procedure on a few examples. Further in Sect. 4 we provide a testing of the proposed algorithm, taking into account mainly its accuracy and the choice of parameters. And in the final section (Sect. 5) the proposed algorithm for approximation of Zadeh's extension is shortly introduced and, finally, the whole process is demonstrated on several examples.

2 Preliminaries

In this subsection we shortly introduce some elementary notions. For more detailed explanation we refer mainly to [8,9] and references therein.

A *fuzzy set* A on a given (compact) metric space (X, d_X), where X is a nonempty space (often called a *universe*), is a map $A\colon X \to [0,1]$. The number $A(x)$ is called a *membership degree* of a point $x \in X$ in the fuzzy set A. For a given $\alpha \in (0,1]$ an *α-cut* of A is the set $[A]_\alpha = \{x \in X \mid A(x) \geq \alpha\}$. Let us remark that if a fuzzy set A is upper semi-continuous then every α-cut of A is a closed subset of X. This helps us later to define a metric on the family of upper semi-continuous fuzzy sets on X which will be denoted by $\mathbb{F}(X)$. Note that if X is not compact then an assumption that every $A \in \mathbb{F}(X)$ has a compact support is required.

Before to define fuzzy dynamical systems it is necessary to define a metric on the family of fuzzy sets $\mathbb{F}(X)$ and such metrics are usually based on the well known Hausdorff metric D_X which measures distance between two nonempty closed subsets of X. For instance, one of the most used metrics on $\mathbb{F}(X)$ is a *supremum metric* d_∞ defined as

$$d_\infty(A,B) = \sup_{\alpha\in(0,1]} D_X([A]_\alpha, [B]_\alpha),$$

for $A, B \in \mathbb{F}(X)$. Considering a metric topology on $\mathbb{F}(X)$ induced by some metric, e.g. by d_∞, we can obtain a topological structure on $\mathbb{F}(X)$.

Thus, let X be a (compact) metric space and $f\colon X \to X$ be a continuous map. Then a pair (X, f) is called a *(discrete) dynamical system*. Dynamics of an initial state $x \in X$ is given by a sequence $\{f^n(x)\}_{n\in\mathbb{N}}$ of forward iterates of x, i.e. $x, f(x), f^2(x) = f(f(x)), f^3(x) = f(f(f(x))), \ldots$. The sequence $\{f^n(x)\}_{n\in\mathbb{N}}$ is called a *forward trajectory* of x under the map f. Now properties of given points are given by properties carried by their trajectories. For instance, a *fixed point* of the function f is a point $x_0 \in X$ such that $f(x_0) = x_0$. A *periodic point* is a point $x_0 \in X$ for which there exists $p \in \mathbb{N}$ such that $f^p(x_0) = x_0$. However, usual trajectories are much more complicated as it is demonstrated on Fig. 1.

In this manuscript we deal with a fuzzy dynamical system which admits the standard definition of a discrete dynamical system and, at the same time, forms a natural extension of a given *crisp*, i.e. not necessarily fuzzy, discrete dynamical system on X. Discrete dynamical systems of the form (X, f) are used in many applications as mathematical models of given processes, see e.g. [11]. Fuzzy dynamical systems studied in this paper are defined with the help of Zadeh's extension which was firstly mentioned by L. Zadeh in 1975 [17] in a more general context. Later in [7] P. Kloeden elaborated a mathematical model of discrete fuzzy dynamical system $(\mathbb{F}(X), z_f)$, which is induced from a given discrete (crisp) dynamical system (X, f). This direction was further elaborated by many mathematicians.

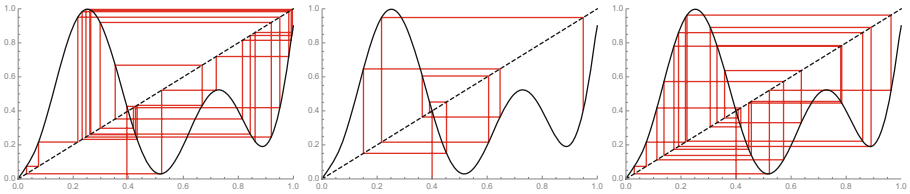

Fig. 1. Trajectories of the dynamical system of a given map, where the initial points are around the value 0.4.

So, formally, let a discrete dynamical system (X, f) be given. Then a continuous map $f\colon X \to X$ induces a map $z_f\colon \mathbb{F}(X) \to \mathbb{F}(X)$ by the formula

$$(z_f(A))(x) = \sup_{y \in f^{-1}(x)} \{A(y)\}.$$

The map z_f is called a *fuzzification* (or *Zadeh's extension*) of the map $f\colon X \to X$. The map z_f fulfils many natural properties, for instance the following equality $[z_f(A)]_\alpha = f([A]_\alpha)$, for any $A \in \mathbb{F}(X)$ and $\alpha \in (0, 1]$, which shows a natural relation to the family of compact subsets (α-cuts) of X. It was proved earlier (see e.g. [7] and [8]) that in the most common topological structures on $\mathbb{F}(X)$ (e.g. for the metric topology induced from the metric d_∞), the continuity of $f\colon X \to X$ is equivalent to the continuity of the fuzzification $z_f\colon \mathbb{F}(X) \to \mathbb{F}(X)$. Consequently a pair $(\mathbb{F}(X), z_f)$ fulfils a formal definition of a discrete dynamical system and this dynamical system is called a *fuzzy dynamical system*. For more information we again refer to [7], [8] and references therein.

3 Particle Swarm Optimization

Particle swarm optimization (abbr. PSO) is an evolutionary optimization algorithm based on stochastic searching in the domain which was originally attributed to R. Eberhart and J. Kennedy in 1995 [4]. The reason of the algorithm, whose behavior is inspired by a social behavior of species, is usually to optimize a given problem. For example, it can be used to find the global optimum of a function of one or more variables. Roughly speaking, population is composed from particles moving around in a given search space according to simple mathematical formulas. In every step of the algorithm, some characteristics (e.g. velocity, the best found solution of every particle, or the best solution in the population) are computed and used to show how the particles move towards the desired solutions. Naturally PSO can stop after a certain number of iterations or after fulfilling some predefined conditions like, for example, accuracy or required size of errors etc. [6,12,16].

3.1 Pseudocode of PSO

The aim of this algorithm is an optimization, i.e. searching for a global optima of a function of one or more variables, which is an essential thing in many applications.

Initially, a finite number of particles x_i is placed into the domain and each particle is evaluated by a given function. Each particle then determines its movement in the domain, with the help of its historical (personal) best position and of the neighbouring particles combined together with some random parameters.

Below we can see the pseudocode of the original version of PSO which can search for a global optima of a given function. Let $f\colon \mathbb{R} \to \mathbb{R}$ be a function for which we search for a global optimum. Now, a number of particles is equal to n and a vector $x = (x_1, \ldots, x_n)$ gives a position of each particle $x_i \in \mathbb{R}$, $p = (p_1, \ldots, p_n)$ is a vector of the best found positions of each particle in its history, p_g is the best position of a particle from the population called *the best neighbour* and v_i is a *velocity* of each particle. Because it is an iterative algorithm, in each iteration, the function f is evaluated in all particles. If the value $f(x_i)$ is better than the previous best value p_i, then this point is rewritten as the best point ($p_i := x_i$) and the value of the function $f(p_i)$ is saved to the value of p_{best_i}. The new position of the particles is then reached by updating the velocity v_i. Now we can explain the parameters which are used in the equation defining the velocity v_i of the next iteration of a particle position. The elements U_{Φ_1}, U_{Φ_2} indicate random points given by a uniform distribution of each component from intervals $[0, \Phi_1]$, $[0, \Phi_2]$, where $\Phi_1, \Phi_2 \in \mathbb{R}$. Parameters Φ_1, Φ_2 are called *acceleration coefficients*, where Φ_1 gives the importance of the personal best value and Φ_2 gives the importance of the neighbors best value. If both of these parameters are too high then the algorithm can be unstable because the velocity could grow up faster. Parameter χ called a *constriction factor*, multiplies the newly calculated velocity and it can affect the movement propagation given by the last velocity value. The original version of PSO works with $\chi = 2/(\Phi - 2 + \sqrt{\Phi^2 - 4\Phi})$, where $\Phi = \Phi_1 + \Phi_2$. The value of this parameter is not changed in time and it has restrictive effect to the result.

1. Initialization (functions, variables, constants, ...)
2. Cycle - for all i calculate $f(x_i)$
3. Comparison
   - compare p_{best_i} and $f(x_i)$
   - if $p_{best_i} \leq f(x_i)$, then $p_i := x_i$ and $p_{best_i} := f(p_i)$
4. Best neighbour
   - find the best neighbour of i and assign it j
   - if $f(p_g) \leq f(x_j)$, then $p_g := x_j$ and $f(p_g) := f(x_j)$
5. Calculation
   - $v_i := \chi(v_i + U_{\Phi_1}(p_i - x_i) + U_{\Phi_2}(p_g - x_i))$
   - $x_i := x_i + v_i$

3.2 The Use of PSO for Linearization

This algorithm is an adaptation of the one-dimensional algorithm introduced above to a higher-dimensional case. We have a map given by a formula $f(x)$ and we want to search for a particle, i.e. a vector of ℓ points, which gives us the best possible linearization of f. These points indicate a *dimension* ℓ of the problem

under consideration and it defines a size of each particle $\mathbf{x}$ in the population. Because the idea is to get points (particles) which give us a piecewise linear function, it becomes a problem in which we want to minimalize distances between the original function f and approximating functions given by particles.

The distance between the initial function f and the approximating piecewise linear function is calculated with the help of the following *Manhattan metric* d_M on a finite number of points D in the domain of f. This metric is given by a function $d_M \colon \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ defined by

$$d_M(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} |x_i - y_i|,$$

where $\mathbf{x} = \{x_1, x_2, \ldots, x_D\}$ and $\mathbf{y} = \{y_1, y_2, \ldots, y_D\}$.

4 Testing

4.1 Parameter Selection

In this subsection we briefly describe the choice of selected parameters and then we study their influence to the accuracy of the proposed algorithm. For each parameter settings the results are calculated 50 times and then evaluated by means of mean and standard deviation.

In general the choice of PSO parameters can have a large impact on optimization performance. In our optimization problem we look for the best setting of the constriction factor χ and acceleration coefficients Φ_1, Φ_2. If both of the parameters Φ_1, Φ_2 are too high then the algorithm can be unstable because the related velocity could grow up faster. There are some recommendations that can be found in the literature. For example, the authors of the original algorithm [4] recommended the values of Φ_1, Φ_2 to be set to 2.05 and they also recommended the following equation $\Phi_1 + \Phi_2 > 4$ to be satisfied. However, parameter setting can be different model by model and therefore we did it for our purpose as well.

In our testing we set the parameter $\chi \in \{0.57, 0.61, 0.65, 0.69, 0.73\}$ and parameters $\Phi_1, \Phi_2 \in \{1.65, 1.85, 2.05, 2.25, 2.45\}$. Thus we have 125 possible combinations of parameters (i.e. 25 combinations for Φ_1, Φ_2 and 5 possibilities for χ). For each of these combinations the proposed algorithm runs 50 times to get the mean and standard deviation from computed outcomes. Some of the initial parameters are fixed - namely, all results are calculated for fixed numbers defining linear parts of approximating function ($\ell = 12$), the number of particles in population ($\mathbf{x} = 25$), the number of iterations of PSO ($I = 100$) and the number of points at which the metric d_M is computed ($D = 80$). These parameters are chosen only for our testing, therefore for the use of this algorithm it should always be considered what the best initial parameters are for our linearized function. For instance the recommended number of linear parts ℓ should definitely be much bigger that the number of monotone parts of the function under consideration to be able somehow cover at least all monotone parts of the approximated function.

4.2 Testing Functions

For the purpose of testing we have chosen functions g_1, g_2 given by the following expressions:

$$g_1(x) = 0.9 + (-1 + x)(0.9 + (-0.16 + (5.4 + (-27 + (36 + (510 + (-120 - 2560(-0.9 + x))(-0.1 + x))(-0.6 + x))(-0.2 + x))(-0.8 + x))(-0.4 + x))x),$$

$$g_2(x) = 1/25(\sin 20x + 20x \cdot \sin 20x \cdot \cos 20x) + 1/2.$$

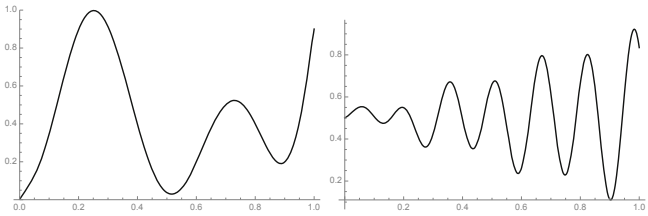

Fig. 2. The graphs of the functions $g_1(x)$ (the left one) and $g_2(x)$ (the right one).

4.3 Parameters Selection

In Tables 1 and 2 below one can see that the choice of parameters χ, Φ_1, Φ_2 can affect the results of the linearization procedure. After some preliminary testing on several functions we can provide some introductory observations. Namely, the best setting was found when the constriction factor χ was taken among values $0.57, 0.61, 0.65$ and in a few cases also for the value 0.69. However for χ lying outside of the interval $[0.61, 0.73]$, the results were much worse and did not meet our expectations. Concerning parameters Φ_1, Φ_2, again, our general observation is that better results are obtained when $\Phi_1 \geq \Phi_2$.

It is natural for stochastic methods, that in particular tasks you can get more specific combination of parameters - in the following two tables we emphasize in bold the best 3 combinations of attributes for particular functions g_1 and g_2. More extensive testing is planned as the continuation of this manuscript.

4.4 Examples

In this subsection we demonstrate the use of the proposed algorithm on three nontrivial functions g_1, g_2 and g_3, where the first two functions were defined in Subsect. 4.2, and we consider the best settings of random parameters selected in Subsect. 4.3.

Table 1. The values of mean and standard deviation for the function $g_1(x)$.

χ	$\Phi_1 = 1.65, \Phi_2 = 1.65$	$\Phi_1 = 1.65, \Phi_2 = 2.05$	$\Phi_1 = 1.65, \Phi_2 = 2.45$	$\Phi_1 = 2.05, \Phi_2 = 2.05$	$\Phi_1 = 2.05, \Phi_2 = 2.45$
0.57	1.63547±0.238939	1.67338±0.266557	1.65651±0.266593	1.57143±0.213432	1.54162±0.198801
0.61	1.62098±0.281265	1.60798±0.247689	1.59534±0.222001	1.60351±0.226622	1.6108±0.252729
0.65	1.68093±0.257731	1.58062±0.214022	1.59761±0.21376	1.57592±0.224728	1.52531±0.189646
0.69	1.62679±0.227925	1.59228±0.235681	1.9139±0.34237	1.65493±0.276727	1.98223±0.355166
0.73	1.62422±0.214448	1.83744±0.276758	3.10602±0.67275	2.26001±0.571314	4.03971±0.787626
χ	$\Phi_1 = 2.45, \Phi_2 = 2.45$	$\Phi_1 = 2.45, \Phi_2 = 2.05$	$\Phi_1 = 2.45, \Phi_2 = 1.65$	$\Phi_1 = 2.05, \Phi_2 = 1.65$	$\Phi_1 = 1.65, \Phi_2 = 1.85$
0.57	1.63376±0.249797	1.53581±0.194398	1.54222±0.197587	1.58635±0.233554	1.59812±0.229893
0.61	1.55107±0.191518	1.55327±0.205339	1.58552±0.22103	1.57803±0.227844	1.60375±0.242386
0.65	1.57619±0.219947	1.57579±0.204595	**1.51653±0.178176**	1.56608±0.222457	1.63805±0.218378
0.69	3.421±0.689219	1.57886±0.239874	1.54552±0.217573	1.53172±0.20908	1.617±0.223034
0.73	5.72016±0.834381	2.50061±0.643045	1.60521±0.212556	1.53968±0.200264	1.54681±0.205785
χ	$\Phi_1 = 1.65, \Phi_2 = 2.25$	$\Phi_1 = 1.85, \Phi_2 = 1.85$	$\Phi_1 = 1.85, \Phi_2 = 2.05$	$\Phi_1 = 1.85, \Phi_2 = 2.25$	$\Phi_1 = 1.85, \Phi_2 = 2.45$
0.57	1.59498±0.216631	1.60299±0.243964	1.62142±0.226532	1.56213±0.22595	1.66831±0.217123
0.61	1.62145±0.209883	1.57229±0.184818	1.58244±0.228172	1.57339±0.212051	1.57717±0.229499
0.65	1.63451±0.232793	1.595±0.241515	1.53383±0.189491	1.60258±0.228966	1.58079±0.222188
0.69	1.65213±0.278563	1.58664±0.221945	1.59723±0.234288	1.61548±0.218672	1.86881±0.415271
0.73	1.99725±0.391979	1.6464±0.257361	1.89429±0.301735	2.86462±0.545212	3.5004±0.789913
χ	$\Phi_1 = 2.05, \Phi_2 = 2.25$	$\Phi_1 = 2.25, \Phi_2 = 2.25$	$\Phi_1 = 2.25, \Phi_2 = 2.45$	$\Phi_1 = 2.45, \Phi_2 = 2.25$	$\Phi_1 = 2.45, \Phi_2 = 1.85$
0.57	1.53667±0.217165	1.57757±0.208589	1.63313±0.241569	1.56437±0.223043	1.59828±0.211077
0.61	1.57841±0.220252	1.51553±0.19691	1.57972±0.230095	1.54784±0.250593	1.57163±0.223167
0.65	1.56308±0.216044	1.56444±0.235687	1.53595±0.189361	1.62629±0.235062	1.58514±0.220281
0.69	1.70348±0.29063	1.64649±0.242506	2.72718±0.626525	2.27222±0.527745	1.5791±0.202365
0.73	3.2433±0.546781	3.93251±0.809978	5.04399±0.684256	3.92034±0.91482	2.20619±0.495479
χ	$\Phi_1 = 2.25, \Phi_2 = 2.05$	$\Phi_1 = 2.25, \Phi_2 = 1.85$	$\Phi_1 = 2.25, \Phi_2 = 1.65$	$\Phi_1 = 2.05, \Phi_2 = 1.85$	$\Phi_1 = 1.85, \Phi_2 = 1.65$
0.57	1.57289±0.220332	**1.49649±0.18915**	1.5457±0.195436	1.56029±0.208285	1.61812±0.239638
0.61	1.56538±0.204659	1.56143±0.209023	1.57824±0.221359	1.59234±0.234838	1.53362±0.194484
0.65	1.60941±0.238231	1.60536±0.219927	1.52257±0.213785	1.57849±0.21733	1.58347±0.247829
0.69	1.54909±0.20935	1.53217±0.192154	1.5495±0.180544	1.58344±0.210707	1.57927±0.2096
0.73	2.61903±0.579384	1.71432±0.33481	**1.50373±0.166186**	1.73587±0.267968	1.53825±0.173687

[a] Compiled in *Mathematica* 12.0 on a laptop with processor 1,8 GHz Intel Core i5.

Table 2. The values of mean and standard deviation for the function $g_2(x)$.

χ	$\Phi_1 = 1.65, \Phi_2 = 1.65$	$\Phi_1 = 1.65, \Phi_2 = 2.05$	$\Phi_1 = 1.65, \Phi_2 = 2.45$	$\Phi_1 = 2.05, \Phi_2 = 2.05$	$\Phi_1 = 2.05, \Phi_2 = 2.45$
0.57	4.48464±0.808152	4.33324±0.670205	4.43686±0.774412	4.24214±0.707924	4.43853±0.737978
0.61	4.31768±0.682493	4.49225±0.749727	4.21937±0.63087	4.24245±0.755797	4.30317±0.677813
0.65	4.53499±0.531912	4.14321±0.657339	4.29982±0.670138	4.18762±0.674639	4.64488±0.94961
0.69	4.43666±0.719598	4.18925±0.727941	5.7266±1.06469	4.70045±0.902873	6.64708±1.16885
0.73	4.23356±0.74098	5.4096±1.10863	7.88197±0.951119	6.76307±1.05201	8.72105±0.956886
χ	$\Phi_1 = 2.45, \Phi_2 = 2.45$	$\Phi_1 = 2.45, \Phi_2 = 2.05$	$\Phi_1 = 2.45, \Phi_2 = 1.65$	$\Phi_1 = 2.05, \Phi_2 = 1.65$	$\Phi_1 = 1.65, \Phi_2 = 1.85$
0.57	4.20244±0.667743	4.22678±0.549085	4.18617±0.510273	4.4319±0.685112	4.4517±0.706558
0.61	4.20342±0.764773	4.22028±0.520342	4.10346±0.603738	4.23213±0.711974	4.32719±0.636389
0.65	5.42939±1.16048	4.4184±0.802053	**3.99467±0.590382**	4.18327±0.635378	4.26412±0.781714
0.69	8.30996±0.983165	5.46925±0.995767	**3.90281±0.646766**	**4.03962±0.527906**	4.17317±0.555974
0.73	8.31745±1.04308	8.02816±1.0483	6.02647±1.24507	4.58364±0.833236	4.56542±0.978583
χ	$\Phi_1 = 1.65, \Phi_2 = 2.25$	$\Phi_1 = 1.85, \Phi_2 = 1.85$	$\Phi_1 = 1.85, \Phi_2 = 2.05$	$\Phi_1 = 1.85, \Phi_2 = 2.25$	$\Phi_1 = 1.85, \Phi_2 = 2.45$
0.57	4.76712±0.819473	4.46511±0.841069	4.19868±0.648816	4.33206±0.617926	4.32131±0.780307
0.61	4.46591±0.645347	4.47089±0.791886	4.33041±0.483959	4.21253±0.815479	4.26108±0.471753
0.65	4.08529±0.640999	4.39325±0.749726	4.08447±0.641931	4.30325±0.79269	4.47726±0.719004
0.69	4.86003±0.80498	4.09351±0.600797	4.61469±0.862227	5.39318±0.93059	6.07631±1.1166
0.73	6.91749±1.08444	5.06067±1.0387	5.79045±0.991023	7.78185±1.09352	8.15509±1.11355
χ	$\Phi_1 = 2.05, \Phi_2 = 2.25$	$\Phi_1 = 2.25, \Phi_2 = 2.25$	$\Phi_1 = 2.25, \Phi_2 = 2.45$	$\Phi_1 = 2.45, \Phi_2 = 2.25$	$\Phi_1 = 2.45, \Phi_2 = 1.85$
0.57	4.33195±0.763994	4.23792±0.618683	4.27062±0.71422	4.26326±0.707052	4.31927±0.729022
0.61	4.27319±0.596503	4.29391±0.724273	4.14659±0.812527	4.20238±0.573447	4.18677±0.682581
0.65	4.30942±0.741097	4.25068±0.734095	4.73966±0.798268	4.6961±0.934194	4.10398±0.530543
0.69	5.87982±0.967094	6.35378±1.00249	7.23311±1.22546	7.43348±1.08212	4.84346±0.956979
0.73	8.18012±0.796192	8.64843±0.990123	8.77355±0.937052	8.30777±1.15618	6.78641±1.22497
χ	$\Phi_1 = 2.25, \Phi_2 = 2.05$	$\Phi_1 = 2.25, \Phi_2 = 1.85$	$\Phi_1 = 2.25, \Phi_2 = 1.65$	$\Phi_1 = 2.05, \Phi_2 = 1.85$	$\Phi_1 = 1.85, \Phi_2 = 1.65$
0.57	4.26443±0.661071	4.30843±0.643455	4.24803±0.727079	4.16216±0.593539	4.4644±0.712637
0.61	4.22633±0.68801	4.34254±0.67098	4.20053±0.648986	4.25709±0.687653	4.31687±0.55673
0.65	4.23431±0.701167	4.22943±0.691592	4.14209±0.633141	4.12123±0.646354	4.15398±0.577304
0.69	5.41382±1.04234	4.28741±0.7065	4.13273±0.713539	4.23241±0.641686	4.33662±0.707127
0.73	6.98054±1.18231	6.47896±1.11268	5.11513±0.968546	5.39849±1.12682	4.22943±0.885804

[a] Compiled in *Mathematica* 12.0 on a laptop with processor 1,8 GHz Intel Core i5.

Example 1. We have the function g_1 whose graph is depicted in Fig. 2. The initial parameters are set to $\chi = 0.69, \Phi_1 = 2.45, \Phi_2 = 1.65$. In our testing we choose $\ell = 6, 12, 18$, $I = 100$ and $D = 80$. This function g_1 has 5 monotone parts, thus if our intention is to linearize the function g_1 the smallest number ℓ to be considered is 6. Naturally, the higher ℓ we take, the smoother result we obtain. This is demonstrated on the following figure (Fig. 3).

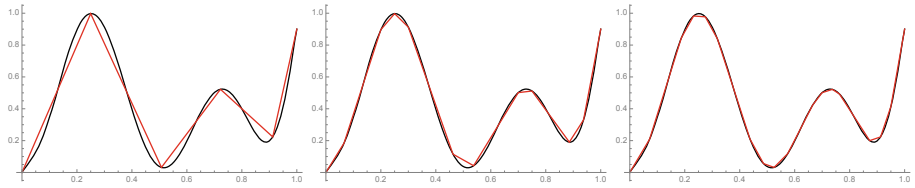

Fig. 3. Graphs of the original function g_1 (black lines) and the piecewise linear functions (red lines) approximating g_1, where $\ell = 6, 12, 18$. (Color figure online)

Example 2. Consider a function g_2 (see Fig. 2) and take the initial parameters $\chi = 0.69, \Phi_1 = 2.45, \Phi_2 = 1.65$, $D = 80$, $I = 100$, $\ell = 15, 18, 25$ (see Fig. 4).

In this example, 14 monotone parts are divided almost equidistantly. Naturally, for better accuracy the number of linear parts should be higher as we can see in the next figure.

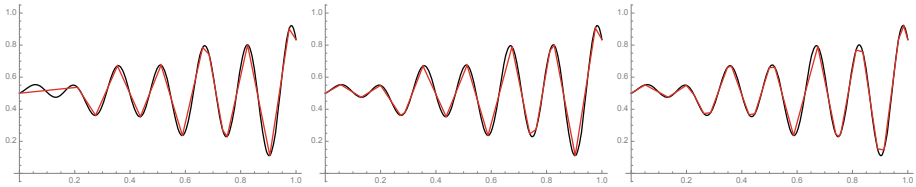

Fig. 4. Graphs of the original function g_2 (black lines) and its piecewise linear approximations (red lines), where $\ell = 15, 18, 25$. (Color figure online)

Another simple observation is that for better accuracy, it need not help to increase the number of linear parts only, but we need to increase also the number D of discretization points accordingly. To demonstrate this, we choose D to be equal to 200, where $\ell = 25$, $\ell = 50$ and $I = 100$ (see Fig. 5).

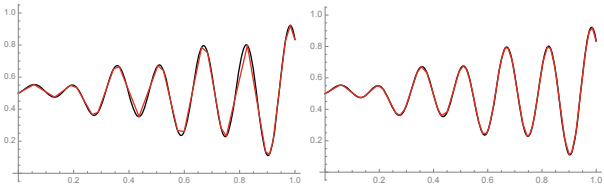

Fig. 5. Graphs of the original function g_2 (black lines) and its piecewise linear approximations of g_2 (red lines). (Color figure online)

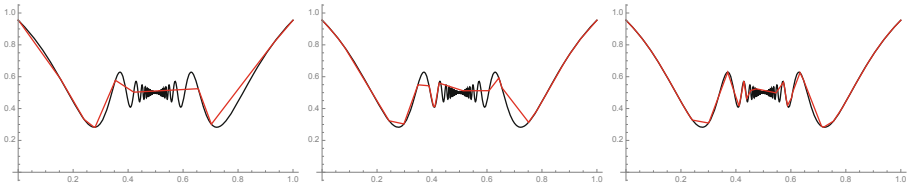

Fig. 6. Graphs of the original function g_3 (black lines) and the piecewise linear functions (red lines) approximating g_3, where $\ell = 12, 25, 40$ and $D = 80$. (Color figure online)

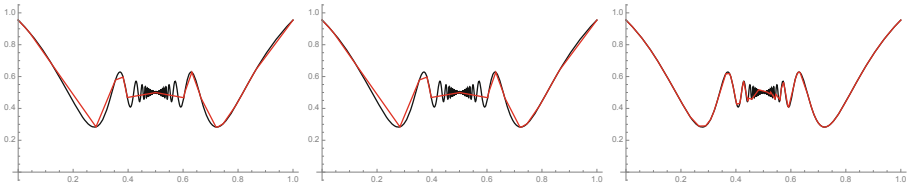

Fig. 7. Graphs of the original function g_3 (black lines) and the piecewise linear functions (red lines) approximating g_3, where $\ell = 25, 40, 100$ and $D = 1000$. (Color figure online)

Example 3. Consider a function $g_3(x) = (x - 1/2)(\sin(1/(x - 1/2))) + 1/2$ (see the picture below). The initial parameters are $\chi = 0.69, \Phi_1 = 2.45, \Phi_2 = 1.65$. In Fig. 6, $\ell = 12, 25, 40$, $I = 100$ and $D = 80$ while in Fig. 7, $\ell = 25, 40, 60$, $I = 100$ and $D = 1000$.

We intentionally consider the function g_3 to demonstrate limits of the proposed algorithm because g_3 has infinitely many monotone parts at arbitrary small neighborhood of the point $1/2$. Consequently, in this case it is not possible to approximate all monotone parts correctly. Despite of this drawback we can see in Figs. 6 and 7 that when we increase the number D of discretization points and the number ℓ of linear pairs appropriately, the proposed algorithm works smoothly outside of some neighborhood of the "oscillating" point $1/2$.

4.5 Computational Complexity

In this section, we briefly discuss computation complexity of the proposed algorithm. Naturally, the computation time depends on more factors, mainly on the

Table 3. Computing time in seconds.

Function g_1	$D = 80$	$D = 200$	$D = 500$	$D = 1000$
$\ell = 12$	35.92	78.38	189.95	372.82
$\ell = 18$	50.33	111.92	273.8	537.57
$\ell = 25$	65.75	163.33	364.63	734.85
$\ell = 50$	126.28	322.48	775.28	1462.72

[a] Compiled in *Mathematica* 12.0 on a laptop with processor 1,8 GHz Intel Core i5.

number ℓ of linear parts, the number I of iterations, the number D of discretization points and also on computer which is used for compiling. In the table below, we show the time in dependence on number of pairs ℓ and number of discretization points D, which are the most important parameters for the accuracy of this algorithm. The test was executed on function g_1 defined above and with parameters $\chi = 0.69, \Phi_1 = 2.45$, $\Phi_2 = 1.65$ and $I = 100$ (Table 3).

5 Approximation of Zadeh's Extension

5.1 Algorithm

In this subsection we briefly recall an algorithm for calculation of Zadeh's extension of a given function f. For a more detailed description of this algorithm we refer to [10]. The algorithm in [10] was proposed for one-dimensional continuous functions $f\colon X \to X$, i.e. we assume $X = [0, 1]$, but one can consider any closed subinterval of $\mathbb{R}$. The algorithm was proposed for piecewise linear maps f and piecewise linear fuzzy sets A. In this section we demonstrate a generalization of the algorithm from [10] to maps which are not necessarily piecewise linear.

The purpose of the algorithm was to compute a trajectory of a given discrete fuzzy dynamical subsystem $(\mathbb{F}(X), z_f)$, which is obtained as a unique and natural extension of a given discrete dynamical system (X, f).

Thus we consider a continuous map $f\colon [0, 1] \to [0, 1]$ and a piecewise linear fuzzy set A representing an initial state of induced fuzzy dynamical system $(\mathbb{F}([0, 1]), z_f)$. First we use the PSO-based linearization (described in Sect. 5) to get an approximated piecewise linear function $\tilde{f}$, and then we use the algorithm from [10] to calculate a trajectory of the initial state A in the fuzzy dynamical system $(\mathbb{F}([0, 1]), z_{\tilde{f}})$. This simple and natural generalization is demonstrated in the following subsection.

5.2 Examples

Let us see two examples of the procedure described in the previous subsection.

Example 4. Let a function f_1 be given by a formula

$$f_1(x) = (-2.9 + (-4.1 + (-15.6 - 14(-0.8 + x))(-0.2 + x))(-0.6 + x))(-1 + x)x$$

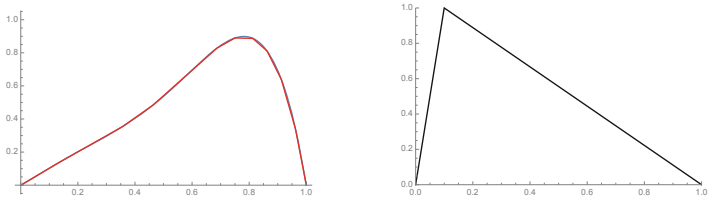

Fig. 8. The graph of a function f_1 (the left figure, black line) and the linearization of f_1 given by PSO (the left figure, red line), the graph of a fuzzy set A (the right figure). (Color figure online)

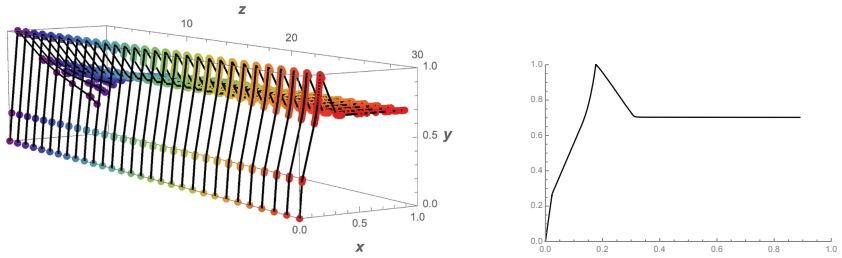

Fig. 9. The graphs of $A, z_{\tilde{f}_1}(A), \ldots, z^{30}_{\tilde{f}_1}(A)$ (the left one) and $z^{30}_{\tilde{f}_1}(A)$ (the right one).

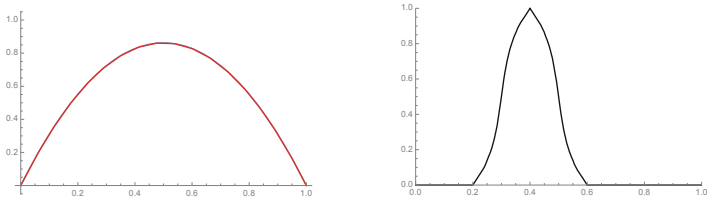

Fig. 10. The graph of a function f_2 (the left figure, black line) and the linearization of f_2 given by PSO (the left figure, red line), the graph of a fuzzy set A (the right figure). (Color figure online)

and let $A(y)$ be a fuzzy set depicted on Fig. 8. As the first step of the algorithm we linearize the function f_1. For that reason, we use the PSO-based algorithm with parameters $\ell = 12, D = 80, I = 100$. After the linearization process we obtain a piecewise linear function $\tilde{f}_1$ and we can compute a plot containing the first 30 iterations of the fuzzy set A (see Fig. 9).

Example 5. Let a function f_2 be given by the following formula $f_2(x) = 3.45(x - x^2)$ and $A(y)$ be a fuzzy set depicted in Fig. 10. Again, we need to linearize the function f_2. To do this we use the PSO-based algorithm with the following parameters $\ell = 18, D = 80, I = 100$.

Finally we can see a plot of the images of the fuzzy set A for the first 30 iterations (see Fig. 11).

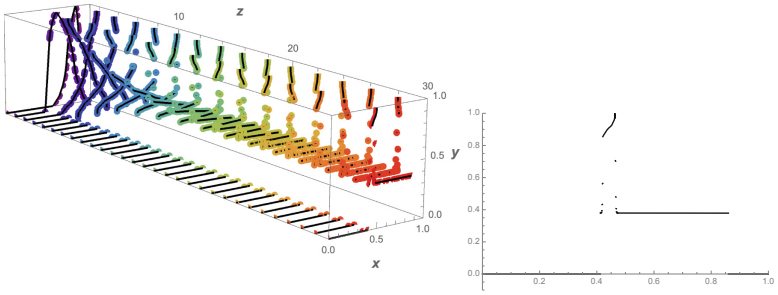

Fig. 11. The graphs of $A, z_{\tilde{f}_2}(A), \ldots, z^{30}_{\tilde{f}_2}(A)$ (the left one) and $z^{30}_{\tilde{f}_2}(A)$ (the right one).

6 Conclusion

In this contribution we generalized our previous algorithm from [10]. The main idea of this algorithm was to calculate Zadeh's extension for a piecewise linear function and a fuzzy set. Because we restricted our attention only to piecewise linear functions the next natural step becomes a generalization of the previous approach to arbitrary continuous functions. Consequently, we took an evolutionary algorithm called particle swarm optimization and we adapt this algorithm to searching for the best possible linearization of a given function. This naturally extends the use of our previous algorithm of an approximation of Zadeh's extension, which now gives us an approximated trajectory of the initial state A in a more general fuzzy dynamical system.

The newly proposed algorithm has been briefly tested from several points of view, mainly parameters selection has been taken into account. In our future work we plan more extensive testing involving also computational complexity of the algorithm given by Big O notation which will deal, for example, with the size of the population, number of iterations, number of linear parts, etc. Another natural step is to provide a deep comparison of the original trajectory derived by the Zadeh's extension with the one given by our algorithm, to provide various comparisons to previously known approaches, and, eventually, to involve, dynamic adaptation of parameters in our PSO-based algorithm. After that, the algorithm should be naturally extended to higher dimensions.

References

1. Ahmad, M.Z., Hasan, M.K.: A new approach for computing Zadeh's extension principle. Matematika **26**, 71–81 (2010)
2. Chalco-Cano, Y., Misukoshi, M.T., Román-Flores, H., Flores-Franulic, A.: Spline approximation for Zadeh's extensions. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **17**(02), 269–280 (2009)
3. Chalco-Cano, Y., Román-Flores, H., Rojas-Medar, M., Saavedra, O., Jiménez-Gamero, M.D.: The extension principle and a decomposition of fuzzy sets. Inf. Sci. **177**(23), 5394–5403 (2007)

4. Eberhart, R., Kennedy, J.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948. Citeseer (1995)
5. Guerra, M.L., Stefanini, L.: Approximate fuzzy arithmetic operations using monotonic interpolations. Fuzzy Sets Syst. **150**(1), 5–33 (2005)
6. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of Machine Learning, pp. 760–766 (2010)
7. Kloeden, P.: Fuzzy dynamical systems. Fuzzy Sets Syst. **7**(3), 275–296 (1982)
8. Kupka, J.: On fuzzifications of discrete dynamical systems. Inf. Sci. **181**(13), 2858–2872 (2011)
9. Kupka, J.: A note on the extension principle for fuzzy sets. Fuzzy Sets Syst. **283**, 26–39 (2016)
10. Kupka, J., Škorupová, N.: Calculations of Zadeh's extension of piecewise linear functions. In: Kearfott, R.B., Batyrshin, I., Reformat, M., Ceberio, M., Kreinovich, V. (eds.) IFSA/NAFIPS 2019 2019. AISC, vol. 1000, pp. 613–624. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21920-8_54
11. Lynch, S.: Dynamical Systems with Applications using MATLAB. Springer, Boston (2004). https://doi.org/10.1007/978-0-8176-8156-2
12. Olivas, F., Valdez, F., Castillo, O., Melin, P.: Dynamic parameter adaptation in particle swarm optimization using interval type-2 fuzzy logic. Soft Comput. **20**(3), 1057–1070 (2016)
13. Scheerlinck, K., Vernieuwe, H., De Baets, B.: Zadeh's extension principle for continuous functions of non-interactive variables: a parallel optimization approach. IEEE Trans. Fuzzy Syst. **20**(1), 96–108 (2011)
14. Stefanini, L., Sorini, L., Guerra, M.L.: Parametric representation of fuzzy numbers and application to fuzzy calculus. Fuzzy Sets Syst. **157**(18), 2423–2455 (2006)
15. Stefanini, L., Sorini, L., Guerra, M.L.: Simulation of fuzzy dynamical systems using the LU-representation of fuzzy numbers. Chaos Solitons Fractals **29**(3), 638–652 (2006)
16. Valdez, F.: A review of optimization swarm intelligence-inspired algorithms with type-2 fuzzy logic parameter adaptation. Soft Comput. **24**(1), 215–226 (2020)
17. Zadeh, L.A.: Fuzzy logic and approximate reasoning. Synthese **30**(3–4), 407–428 (1975)