DEVELOPMENT OF A FUZZY CALCULATOR FOR CONTINUOUS FUNCTIONS OF NON-INTERACTIVE FUZZY VARIABLES

Karolien Scheerlinck, Hilde Vernieuwe and Bernard De Baets
Department of Applied Mathematics, Biometrics and Process Control, Ghent University, Coupure links 653, Ghent, Belgium

Keywords: Extension principle, Particle swarm optimization, Fuzzy calculator, Non-interactive fuzzy variables.

Abstract: The goal of this paper is to develop a Fuzzy Calculator, making it possible to calculate functions of fuzzy intervals, as prescribed by the extension principle of Zadeh. The extension principle can be reversed, resulting in fixed α-levels for which the minimum and the maximum of the function has to be determined. This optimization problem can be tackled by different algorithms: Gradient Descent, SIMPSA, Particle Swarm Optimization and Particle Swarm optimization in combination with Gradient Descent. Two approaches are used to determine the number of α-levels: it is either fixed to a predetermined value, or it is initially chosen very small and subsequently expanded according to a suitable criterion. Both a non-parallel and a parallel implementation of the Fuzzy Calculator are designed. In the parallel version, communication is used to optimize the internal workings of PSO. The Fuzzy Calculator is applied to a number of test functions. The different combinations of optimization algorithms are evaluated, both by the final result and by the number of required model evaluations. The results indicate that the parallel implementation of the Fuzzy Calculator starting with a small number of α-levels and using PSO with Gradient Descent leads to the most accurate membership function.

1 INTRODUCTION

The concept of fuzzy information plays a central role in many engineering applications. This type of uncertainty can be captured by fuzzy intervals. Calculating with fuzzy intervals is in general a complex process, described by the extension principle of Zadeh. It is possible to reverse the extension principle, and to find, for each membership degree $\alpha \in]0,1]$, the corresponding interval of the membership function (Nguyen, 1978).

Several practical implementations of the extension principle based on the α-level concept are available for (locally) monotone continuous functions of non-interactive fuzzy intervals. The vertex method is developed for computing with monotone functions of non-interactive fuzzy intervals. This method can be extended to non-monotone functions by doing an extreme value analysis (Dong and Shah, 1987; Otto et al., 1993). However, this is not always possible, for example when dealing with complex functions. Another approach for (locally) monotone continuous functions is working with gradual numbers (Fortin et al., 2008; Dubois and Prade, 2008). However, for general functions, an optimisation algorithm is needed to determine the minimum and maximum of the function at each value of α (Maskey et al., 2004; Shrestha et al., 2007).

The objective of this paper is to develop a computationally efficient Fuzzy Calculator to construct the membership function of a fuzzy output interval that depends on non-interactive fuzzy intervals, through a general continuous function. In this paper four different optimisation algorithms are compared to determine the minimum and maximum value of the function at the different α-levels: (1) Gradient Descent based on Sequential Programming (GD) (Eaton, 2002; Nocedal and Wright, 1999), (2) the Simplex-Simulated Annealing approach (SIMPSA) (Cardoso et al., 1996), (3) Particle Swarm Optimisation (PSO) (Kennedy and Eberhart, 1995) and (4) Particle Swarm Optimisation in combination with Gradient Descent (PSO_GD). The first step is to determine the number of α-levels. In this paper, two approaches are used. Either we fix the number of α-levels at the beginning and kept constant throughout the algorithm or, alternatively, we start with only 3 α-levels and gradually increase this number as required by a criterion based on linear interpolation. The next step is to implement the Fuzzy Calculator. Both a non-parallel and a par-

Scheerlinck K., Vernieuwe H. and De Baets B..
DEVELOPMENT OF A FUZZY CALCULATOR FOR CONTINUOUS FUNCTIONS OF NON-INTERACTIVE FUZZY VARIABLES.
DOI: 10.5220/0003053800140020
In *Proceedings of the International Conference on Fuzzy Computation and 2nd International Conference on Neural Computation* (ICFC-2010), pages 14-20
ISBN: 978-989-8425-32-4

allel version are implemented. The latter application is only important when PSO is employed, such that several swarms simultaneously search at different α-levels and moreover communicate with each other as to locate the minimum and maximum of the function more accurately.

This paper is organised as follows. Section 2 reminds the reader of the extension principle applied to construct the membership function of a continuous function of fuzzy intervals. Section 3 describes the different optimisation algorithms used by the Fuzzy Calculator. In Section 4, the implementation of the Fuzzy Calculator and the different optimisation algorithms are outlined in full detail. The test functions used to validate the Fuzzy Calculator are presented in Section 4. Section 5 describes the results and compares them in great detail. Finally, Section 6 contains our conclusion.

2 EXTENSION PRINCIPLE

The extension principle of Zadeh provides a general method to transfer uncertainty described by fuzzy intervals through a function. Concretely, the extension principle of Zadeh determines the membership function μ_Z of a fuzzy output quantity Z that depends on n fuzzy quantities $X_1,\ldots,X_n$ with known membership functions $\mu_{X_1},\ldots,\mu_{X_n}$ through a general function $Z = f(X_1,\ldots,X_n)$ (Zadeh, 1975):

$$\mu_Z(z) = \sup_{z=f(x_1,\ldots,x_n)} \min(\mu_{X_1}(x_1),\ldots,\mu_{X_n}(x_n)) \quad (1)$$

In the case of a continuous function f and upper semi-continuous membership functions, *i.e.* all α-cuts are closed: $\forall\alpha \in]0,1]$, $[\mu_{X_j}]_\alpha$ is closed, with a compact support $\overline{S}_{X_j}$, *i.e.* a bounded support, we can reverse the extension principle. Instead of determining the membership degree $\alpha = \pi_Z(z)$ of a certain value $z = f(x_1,\ldots,x_n)$, it is then possible to determine the interval of values $z \in [\underline{z}_\alpha, \overline{z}_\alpha]$ which have a membership degree $\mu_Z(z) \geq \alpha$ (Nguyen, 1978). Thus, by determining $\underline{z}_\alpha$ and $\overline{z}_\alpha$ for certain values of α, henceforth called α-levels, it is possible to reconstruct the membership function $\pi_Z(z)$. As the interval at $\alpha = 0$ is not closed, we choose the first α-level at a small value $\alpha = \delta > 0$. The range of membership degrees $[\delta,1]$ will be subdivided in m intervals of length $1/m$, with the $m+1$ α-levels $\alpha_j = j/m$, $j = \delta,1,\ldots,m$, as endpoints.

To simplify the search problem to determine the membership function of the fuzzy output quantity, we only use convex membership functions $\mu_X(x)$:

$$\begin{aligned} &\forall(x_1,x_2) \in \mathbb{R}^2, \forall\alpha \in]0,1] : \\ &\mu_X(\alpha x_1 + (1-\alpha)x_2) \geq \min(\mu_X(x_1), \mu_X(x_2)) \end{aligned} \quad (2)$$

Fuzzy quantities whose membership functions satisfy the above described conditions of upper-semi continuity and convexity and have a compact support are henceforth called fuzzy intervals.

For functions without internal minima or maxima for a certain α-level, the minimum $\underline{z}_\alpha$ and maximum $\overline{z}_\alpha$ will be found on the boundary of the corresponding α-cut of the input fuzzy intervals. For general functions, however, the minimum $\underline{z}_\alpha$ and maximum $\overline{z}_\alpha$ can either be located on the boundary or in the interior of the corresponding α-cut of the input fuzzy intervals, and optimisation algorithms will be necessary to try to locate these points efficiently (Maskey et al., 2004; Shrestha et al., 2007).

3 OPTIMISATION ALGORITHMS

As outlined in Section 2, the construction of the membership function of the fuzzy output interval of a general function of non-interactive fuzzy input intervals can be converted into a number of optimisation problems. Four different optimisation algorithms are applied to this problem: Gradient Descent based on Sequential Quadratic Programming (GD), Simplex-Simulated Annealing (SIMPSA), Particle Swarm Optimisation (PSO), and Particle Swarm Optimisation based on Gradient Descent (PSO_GD).

3.1 Gradient Descent based on Sequential Quadratic Programming

Gradient Descent based on Sequential Quadratic Programming (Eaton, 2002; Nocedal and Wright, 1999), a local optimisation algorithm, is an extension of the Quasi Newton method in order to handle constraints. Lagrange multipliers are used to incorporate (nonlinear and linear) equality constraints. In addition, inequality constraints are allowed. At the starting point, the objective function is approximated by a quadratic function, obtained through a Taylor expansion. The algorithm exactly determines the minimum or maximum of that quadratic function, which is the new starting point and the procedure is repeated until a convergence criterion is fulfilled. A possible convergence criterion is that the gradient is approximately zero, which is the mathematical condition for a local extremum (Eaton, 2002; Nocedal and Wright, 1999).

3.2 Simplex-simulated Annealing

The Simplex-Simulated Annealing (SIMPSA) algorithm (Cardoso et al., 1996) is an optimisation algorithm based on a combination of the non-linear Simplex algorithm (Nelder and Mead, 1965) and the Simulated Annealing algorithm (Kirkpatrick et al., 1983). The non-linear Simplex algorithm starts with a randomly chosen starting point $\vec{x} = (x_1, \ldots, x_n)$, for which a simplex, *i.e.* a polytope determined by $n+1$ n-dimensional vertices, is created. In a next step, the function values at the vertices of the simplex are compared. The objective is to move away from the worst point. However, in order to be able to find global optima, wrong-way movements must sometimes be allowed. This is regulated by combining the non-linear simplex algorithm with the Simulated Annealing algorithm. Simulated Annealing is a global optimisation algorithm based on the physical thermal process of annealing (Kirkpatrick et al., 1983). New configurations are accepted by the Simulated Annealing algorithm with a certain probability p, depending on the fitness of the solution and the current system temperature. In combination with the non-linear simplex algorithm, the possible configurations are represented by the vertices of the simplex. As for all heuristic optimisation algorithms, the performance of the SIMPSA algorithm largely depends on the choice of the parameter values inherent to the algorithm (Cardoso et al., 1996).

3.3 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) (Kennedy and Eberhart, 1995), a population-based global optimisation algorithm, starts with the initialisation of a population of N particles, numbered $i = 1, \ldots, N$, having randomly chosen position vectors $\vec{x}_i$ and velocity vectors $\vec{v}_i$. The position of each particle corresponds to a candidate solution of the optimisation problem. In each iteration, the particle's position vector is transported over its velocity. The velocity vector is redirected toward the particle's personal best position and local best position. The contributions of the particle's personal and local best position are weighted through the stochastic variables $c_1 r_1$ and $c_2 r_2$. The positive constants c_1 and c_2 are the cognitive and social parameters, the factors r_1 and r_2 are random numbers between 0 and 1, and are regenerated in each iteration step. When a particle is positioned outside the search space, it is repositioned on the boundary it crossed and its velocity in that direction is set to zero. An inertia weight w is used to decelerate the particle's velocity if a suitable solution is found. As with the other optimisation algorithms, the performance of the algorithm largely depends on the choice of the parameter values.

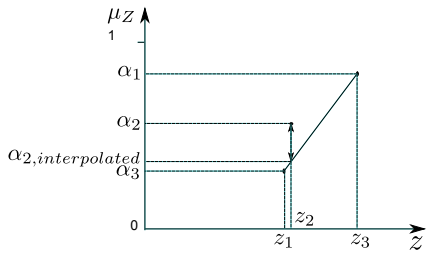

Figure 1: Linear interpolation of α_2 through α_1 and α_3.

4 METHODOLOGY

4.1 Subdivision in α-levels

The construction of the membership function of the output variable of a continuous function of non-interactive fuzzy variables described by fuzzy intervals can be handled through the application of the extension principle and the subdivision in α-levels. The number of α-levels that have to be determined is important, as for a finite number of $m+1$ α-levels, only an approximation for the true membership function $\mu_Z(z)$ will be obtained. Increasing the number of α-levels will improve the approximation.

To determine the number of α-levels, two approaches are followed. Firstly, the number of α-levels is set to a fixed number $m+1$, determined at the beginning of the algorithm. The corresponding α-levels are equidistantly distributed at values $\alpha_j = j/m$ for $j = \delta, \ldots, m$. Secondly, we start with $m+1=3$ and gradually increase the number of α-levels according to a criterion based on linear interpolation. More specifically, we compare α with a value $\tilde{\alpha}$ that is calculated by linear interpolation. We choose to compare α with its linear interpolation $\tilde{\alpha}$ based on z_α, instead of the other way around, since α is always in the interval $]0,1]$ and it is thus possible to work with an absolute tolerance level ε independent of the problem at hand. This is illustrated in Figure 1.

Since the membership function $\mu_Z(z)$ should be upper semi-continuous and convex, it is impossible that $\underline{z}_{\alpha_{j-1}} > \underline{z}_{\alpha_j}$ or that $\overline{z}_{\alpha_{j-1}} < \overline{z}_{\alpha_j}$. Therefore, it is not possible to determine the different α-levels independently and we have to correct the endpoints of the α-levels when this situations occurs. There are two ways to correct for these inconsistencies. If for example $\overline{z}_{\alpha_{j+1}} > \overline{z}_{\alpha_j}$, $\overline{z}_{\alpha_j}$ can be reset to $\overline{z}_{\alpha_{j+1}}$. Another possibility is to discard the old result at α-level α_j and

recalculate $\overline{z}_{\alpha_j}$. If the optimisation algorithm accepts a starting point, then providing the location $\overline{x}_{\alpha_{j+1}}$ will ensure that the inconsistency is solved.

4.2 Non-parallel versus Parallel Implementation

The Fuzzy Calculator is implemented in the programming environment Octave (Eaton, 2002) and accepts a general *n*-ary function as input. Both a non-parallel, and a parallel version were designed using the Message Passing Interface (MPI) of Octave.

Non-parallel. As mentioned above, in the α-level approach the number of α-levels is fixed to $m+1$. The non-parallel version of the algorithm then starts with searching the optima (the minimum and maximum) of the function f at the level $\alpha_m = 1$. As this is the smallest interval, chances are higher to find the correct optima $\underline{z}_{\alpha_m}$ and $\overline{z}_{\alpha_m}$. The algorithm then continues with the determination of the optima for the other α-levels, for decreasing values of α. When allowed by the optimisation algorithm, the optima of the function at the previous α-level can be provided as starting point. With this approach, the inconsistency mentioned in the last paragraph of Section 4.1 cannot occur.

Parallel. The parallel version of the implementation is based on a master-slave configuration. For the fixed number of $m+1$ α-levels, we used $2m+3$ processes: one master and $2(m+1)$ slaves for the determination of the left (right) sides $\underline{z}_{\alpha_j}$ ($\overline{z}_{\alpha_j}$), for $j = \delta, \ldots, m$. The master sends the input intervals belonging to the $m+1$ α-levels to the slaves and the slaves optimise the function for these intervals and send these optima back to the master. When the master receives the optima for all α-levels, these values are corrected for inconsistencies if needed. In this part, the first correction approach is applied.

In the second α-level approach, we start from $m+1=3$ α-levels and expand this number through linear interpolation when necessary. This algorithm starts with searching the optima of the function at the α-levels 1, 0.5 and δ. The obtained values are compared and corrected if inconsistencies should occur. Next, a linear interpolation is applied to examine whether an even finer sampling (additional α-levels) is required. Each time the master receives new results, it checks for possible inconsistencies and subsequently examines whether it has to request the calculation of additional α-levels. The algorithm stops when the convergence criterion of the linear interpolation is fulfilled for all α-levels.

4.3 Optimisation Algorithms

Gradient Descent based on Sequential Quadratic Programming (GD) is a standard function of Octave, namely `sqp` (Eaton, 2002). This function has no extra parameters that have to be determined and can thus be applied directly to our optimisation problem.

The implementation of the Simplex-Simulated Annealing (SIMPSA) algorithm was taken from (Donckels, 2009; Donckels et al., 2009) with the author's permission. As mentioned in Section 3.2, the maximal number of iterations, the cooling ratio, the freezing temperature and the tolerance of the convergence criterion have to be determined. After some test simulations we decided to set the maximal number of iterations for each simulated annealing cycle to 2500, the cooling ratio to 0.7 and the freezing temperature to 1. The tolerance level for convergence was set to 10^{-6}.

The implementation of Particle Swarm Optimisation (PSO) was taken from previous work of the present authors (Scheerlinck et al., 2009) and appropriately modified. This algorithm requires the determination of a number of parameters inherent to the algorithm. After some test simulations, we decided to work with fixed parameter values $c_1 = 1$, $c_2 = 1.5$ and $w = 0.7$, while different population sizes of $N = 10$, $N = 15$ and $N = 20$ were used. The convergence criterion used requires that half of the population has the same position (with tolerance level 10^{-6}). Explicitly, the algorithm stops if the mean distance between the particles of the best half of the population is smaller than 10^{-6}. The parallel Fuzzy Calculator using PSO can be interpreted as several swarms that search on different α-levels at the same time and are thus able to communicate about candidate solutions. We have modified the PSO algorithm such that each swarm broadcasts its current global best position to the other running PSO processes. When a swarm receives a global best position located in its search space and which is better than its own global best position, the swarm will change its global best position. When communication occurs, the remaining particles are reinitialised. This is necessary to prevent slow convergence, for example when the current swarm is already close to converging to a local optimum (far) away from the newly obtained optimum. In this way, a new parameter is introduced, namely the frequency of communication. We varied this parameter by running instances in which the swarms communicate at every 2, 5 or 10 iterations.

As it is not certain that PSO will converge to a local/global optimum (Engelbrecht, 2006), it may be recommended to combine PSO with a local optimisation algorithm such as GD. We performed GD at several positions in the algorithm, in order to have a final best solution which is assured to be a local optimum. First, we performed GD on the initial particles. As after this, all the particles will be positioned in a local optimum, we only kept the particle with the best position in the population. The other particles of the population are repositioned at their original position received during the initialisation of the algorithm. Then, we performed GD each time the swarm changes its global best position through communication. After convergence takes place, we perform GD on the global best position of the swarm one last time.

4.4 Test Functions

In order to compare the different optimisation algorithms, we made use of 9 different test functions, such as the cosine function in one dimension, an arbitrary function in 2 dimensions with multiple minima and maxima, the alpine function in 2 dimensions for different intervals and the alpine function in 3, 4 and 5 dimensions.

The membership functions of the non-interactive fuzzy input intervals are chosen to be normal, *i.e.* $\exists x \in \mathbb{R}$ such that $\mu_X(x) = 1$, and have a trapezoidal shape. The interval $[\underline{x}_{i,\delta}, \overline{x}_{i,\delta}]$ at $\alpha = \delta$, corresponds to the intervals of the test functions. At $\alpha = 1$, this interval is reduced to $[\underline{x}_{i,1}, \overline{x}_{i,1}]$ with

$$\underline{x}_{i,1} = \frac{\underline{x}_{i,\delta} + \overline{x}_{i,\delta}}{2} - \frac{1}{10}\left(\overline{x}_{i,\delta} - \underline{x}_{i,\delta}\right),$$
$$\overline{x}_{i,1} = \frac{\underline{x}_{i,\delta} + \overline{x}_{i,\delta}}{2} + \frac{1}{10}\left(\overline{x}_{i,\delta} - \underline{x}_{i,\delta}\right).$$

In general, our implementation can deal with any upper semi-continuous membership functions. When the fuzzy intervals X_j are interactive, *i.e.* a joint membership function can be specified, however, using the Fuzzy Calculator for such type of membership function will be harder, as the search region at each α-level is no longer hyper-rectangular.

5 RESULTS

The membership functions of the fuzzy output intervals of the test functions are constructed with the Fuzzy Calculator using the optimisation algorithms discussed above. As for each α-level, the error on the determination of $\underline{z}_\alpha$ and $\overline{z}_\alpha$ will depend on the optimisation algorithm, the lowest minimum and highest maximum found by any of the optimisation algorithm can be used as reference. We can thus use the area under the membership function as a global quality measure. The Fuzzy Calculators are also compared on the level of number of function evaluations. To allow for a statistical comparison between the Fuzzy Calculators using different optimisation algorithms or between the non-parallel and parallel implementation, each algorithm is repeated 50 times. To compare the different Fuzzy Calculators, a mixed ANOVA model with the test functions as random effect and a Satterthwaite correction for unequal variances is used (Neter et al., 2004).

5.1 Fixed Number of α-levels

In this section, the number of α-levels is fixed at a value of $m+1 = 11$.

5.1.1 Non-parallel Implementation

This paragraph examines the capability of GD, SIMPSA, PSO with a population of 10, 15 and 20 particles and PSO_GD with a population of 10, 15 and 20 particles to construct the membership functions of the fuzzy output intervals of the 9 test functions. Firstly, a mixed ANOVA model with the different test functions as random effect is applied on the data of the area under the membership function composed by the Fuzzy Calculator using these optimisation algorithms. This test indicates that the application of the Fuzzy Calculator using the algorithms PSO_GD with a population size of 15 and 20 particles and SIMPSA are significantly better in constructing the membership functions than the Fuzzy Calculator using the other optimisation algorithms.

The number of function evaluations needed to construct the membership function is a measure for the computational cost. The number of function evaluations of the Fuzzy Calculator using PSO_GD with a population size of 15 particles (28551) and 20 particles (37662) and SIMPSA (72390) are significantly different. As the Fuzzy Calculator using SIMPSA needs a very high mean number of function evaluations, we can conclude that this algorithm is computationally inefficient. Therefore, the SIMPSA algorithm will not be used in the parallel Fuzzy Calculator.

5.1.2 Parallel Implementation

For the parallel Fuzzy Calculator, we restrict to the optimisation algorithms PSO and PSO_GD.

Again, a mixed ANOVA model with the test functions as random effect is performed. Since the factors population and communication are available in our

two optimisation algorithms, we can put them in this model as nested factors. This leads to a more correct estimate of the *p*-values in our model. The results of the mixed ANOVA model indicate that in all possible combinations of population size and communication strategy, PSO_GD is significantly better than PSO. Furthermore, communication at every 5 iterations and a population size of 20 particles is significantly better than the other parameter combinations. This leads to the conclusion that for the parallel Fuzzy Calculator using PSO_GD with a population size of 20 particles and communication at every 5 iterations is the best algorithm to construct the membership functions of the test functions. For the number of function evaluations, more frequent communication or a larger population size needs significantly more function evaluations.

The last step is the comparison of the parallel results with the non-parallel results. We compared the results of the non-parallel Fuzzy Calculator using PSO_GD with a population size of 15 and 20 particles and the parallel Fuzzy Calculator using PSO_GD with a population size of 20 particles and communication at every 5 iterations. The mean area obtained with the parallel Fuzzy Calculator using PSO_GD with a population size of 20 particles is significantly higher than the mean areas obtained with the non-parallel Fuzzy Calculator using PSO_GD with a population size of 15 and 20 particles, which leads to the conclusion that our parallel implementation gives the best results. However, the number of function evaluations of the parallel Fuzzy Calculator using PSO_GD with communication at every 5 iterations is significantly higher than the non-parallel Fuzzy Calculator using PSO_GD with a population size of 15 and 20 particles.

5.2 Starting from 3 α-levels

A disadvantage of working with a fixed number of α-levels is that this number has to be determined in advance. Consequently, it is possible that too many α-levels are used to determine the membership function of the fuzzy output interval of a simple function whereas too few α-levels are used for more difficult functions. A solution to this problem is starting with 3 α-levels and expanding this number through linear interpolation when necessary. For this linear interpolation, a convergence criterion has to be determined. As convergence criterion, we decided to set a tolerance of 0.01 between α and the guess of $\tilde{\alpha}$ calculated through linear interpolation. With the use of the mixed ANOVA model with the different test functions as random effect, we compared the parallel Fuzzy Calculator starting with 3 α-levels with the parallel Fuzzy Calculator using the fixed number of 11 α-levels. The difference between the mean area under the membership function for the Fuzzy Calculator starting with 3 α-levels and the Fuzzy Calculator using 11 α-levels is significant, which leads to the conclusion that the Fuzzy Calculator starting with 3 α-levels and expanding this number through linear interpolation gives a more accurate membership function than the Fuzzy Calculator using 11 α-levels.

The number of function evaluations is significantly higher for the Fuzzy Calculator starting with 3 α-levels (145074 versus 45119). An explanation is that much more α-levels are optimised when starting with 3 α-levels and expanding this number through linear interpolation (Table 1).

Table 1: Mean number of α-levels needed to construct the left (# α_{min} levels) and right (#α_{max} levels) side of the membership function of the fuzzy output interval for the different test functions with the Fuzzy Calculator starting with 3-α-levels.

Test function	# α_{min} levels	#α_{max} levels
1	21	3
2	6.2	37.6
3	33.6	40.72
4	31.16	19.88
5	21.12	43.56
6	40.28	46.24
7	44.32	62.6
8	45.8	64.16
9	49.64	72.4

As mentioned in Section 4.2, there are two ways to correct for inconsistencies between the α-levels. In the previous results, the optima of all α-levels are compared and replaced by the optima of a higher α-level if necessary. The other approach is to recalculate the optima of the inconsistent α-levels with as starting point the location of the optima of the above lying α-level. We used this last approach in combination with the Fuzzy calculator starting from 3 α-levels and expanding this number through linear interpolation when necessary. We compared these results to those of the Fuzzy Calculator starting with 3 α-levels, using the first approach to correct for inconsistencies between the α-levels. The difference between the mean area under the possibility distribution, obtained with the two approaches, is very small and not significant (p-value > 0.05).

The difference between the number of function evaluations, however, is significant (p-value < 0.05). The second approach for dealing with inconsistent α-levels needs significantly less function evaluations (102101 versus 145074). This is caused by the fact

that in general less α-levels are needed when using the second approach for dealing with inconsistencies between α-levels (Table 2).

Table 2: Mean number of α-levels needed to constructing the left (# α_{min} levels) and right (#α_{max} levels) side of the membership function of the fuzzy output interval for the different test functions with the Fuzzy Calculator starting with an expanding number of α-levels with recalculating incorrectly found optima.

Test function	# α_{min} levels	#α_{max} levels
1	21	3
2	4.9	34.76
3	26.26	38.38
4	19.54	19.22
5	10.24	45.56
6	51.92	54.78
7	25.58	42.82
8	23.4	43.36
9	22.28	46.64

6 CONCLUSIONS

The results indicate that the parallel Fuzzy Calculator is the best way to construct the membership function of the fuzzy output interval of a continuous function of non-interactive fuzzy intervals. The best approach is an expanding number of α-levels, with Particle Swarm Optimisation in combination with Gradient Descent as optimisation algorithm, using a population size of 20 particles and communication at every 5 iterations, and by recalculating inconsistent α-levels. The number of function evaluations, however, can be quite high, depending on the number of α-levels that will be constructed. This can be regulated by the tolerance level in the criterion that determines the insertion of additional α-levels. In addition, as the implementation is parallel and several processors can be used, an elevated number of function evaluations will not pose a major problem for most applications if a high performance facility is available.

ACKNOWLEDGEMENTS

This work was supported by the Special Research Fund of Ghent University and the Belgian Science Policy (STEREO-project SR/00/100).

REFERENCES

Cardoso, M., Salcedo, R., and de Azevedo, S. F. (1996). The simplex-simulated annealing approach to continuous non-linear optimization. *Computers and Chemical Engineering*, 20:1065–1080.

Donckels, B. (2009). *Optimal experimental design to discriminate among rival dynamic mathematical models*. PhD thesis, Ghent University.

Donckels, B., De Pauw, D., Vanrolleghem, P., and De Baets, B. (2009). A kernel-based method to determine optimal sampling times for the simultaneous estimation of the parameters of the rival mathematical models. *Journal of Computational Chemistry*, 30:2064–2077.

Dong, W. and Shah, H. (1987). Vertex method for computing functions of fuzzy variables. *Fuzzy Sets and Systems*, 24:65–78.

Dubois, D. and Prade, H. (2008). Gradual elements in a fuzzy set. *Soft Computing*, 12:165–175.

Eaton, J. W. (2002). *GNU Octave Manual*. Network Theory Limited.

Engelbrecht, A. (2006). *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons Ltd.

Fortin, J., Dubois, D., and Fargier, H. (2008). Gradual numbers and their application to fuzzy interval analysis. *IEEE Transactions on Fuzzy Systems*, 16:388–402.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *IEEE International Conference on Artificial Neural Networks*, pages 1942–1948, Piscataway, NJ.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by Simulated Annealing. *Science*, 220:671–680.

Maskey, S., Guinot, V., and Price, R. (2004). Treatment of precipitation uncertainty in rainfall-runoff modelling: a fuzzy set approach. *Advances in Water Resources*, 27:889–898.

Nelder, J. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. (2004). *Applied Linear Statistical Models*. McGraw-Hill/Irwin.

Nguyen, H. (1978). A note on the extension principle for fuzzy sets. *Mathematical Analysis and Applications*, 64:369–380.

Nocedal, J. and Wright, S. (1999). *Numerical Optimization*. Springer Verlag.

Otto, K., Lewis, A., and Antonsson, E. (1993). Approximating α-cuts with the vertex method. *Fuzzy Sets and Systems*, 55:43–50.

Scheerlinck, K., Pauwels, V., Vernieuwe, H., and De Baets, B. (2009). Calibration of a water and energy balance model: Recursive parameter estimation versus particle swarm optimization. *Water Resources Research*, 45, W10422.

Shrestha, R. R., Brdosst, A., and Nestmann, F. (2007). Analysis and propagation of uncertainties due to the stage-discharge relationship: a fuzzy set approach. *Hydrological Sciences*, 52:595–610.

Zadeh, L. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, 8:199–249.