

Western  Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

May 2012


Fuzzy Differential Evolution Algorithm

Dejan Vucetic
The University of Western Ontario

Supervisor
Dr. Slobodan Simonovic
The University of Western Ontario

Graduate Program in Civil and Environmental Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Master of
Engineering Science
© Dejan Vucetic 2012

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>

 Part of the [Civil and Environmental Engineering Commons](#), [Electrical and Computer Engineering Commons](#), and the [Operations Research, Systems Engineering and Industrial Engineering Commons](#)

Recommended Citation

Vucetic, Dejan, "Fuzzy Differential Evolution Algorithm" (2012). *Electronic Thesis and Dissertation Repository*. 503.
<https://ir.lib.uwo.ca/etd/503>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.

FUZZY DIFFERENTIAL EVOLUTION ALGORITHM

(Spine title: Fuzzy Differential Evolution Algorithm with Application in Water Resource
Systems)

(Thesis format: Monograph)

by

Dejan Vucetic

Graduate Program in Civil and Environmental Engineering

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Engineering Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies
CERTIFICATE OF EXAMINATION

Supervisor

Examiners

Dr. Slobodan Simonovic

Dr. Jagath Samarabandu

Supervisory Committee

Dr. Jason Gerhard

Dr. Ashraf Nassef

The thesis by

Dejan Vucetic

entitled:

Fuzzy Differential Evolution Algorithm

is accepted in partial fulfillment of the
requirements for the degree of
Master of Engineering Science

Date

Chair of the Thesis Examination Board

Abstract

The Differential Evolution (DE) algorithm is a powerful search technique for solving global optimization problems over continuous space. The search initialization for this algorithm does not adequately capture vague preliminary knowledge from the problem domain. This thesis proposes a novel Fuzzy Differential Evolution (FDE) algorithm, as an alternative approach, where the vague information of the search space can be represented and used to deliver a more efficient search. The proposed FDE algorithm utilizes fuzzy set theory concepts to modify the traditional DE algorithm search initialization and mutation components. FDE, alongside other key DE features, is implemented in a convenient decision support system software package. Four benchmark functions are used to demonstrate performance of the new FDE and its practical utility. Additionally, the application of the algorithm is illustrated through a water management case study problem. The new algorithm shows faster convergence for most of the benchmark functions.

Keywords

Fuzzy numbers, Genetic algorithms, Differential Evolution Algorithms, Fuzzy random variables, Fuzzy Set Theory, Optimization, Water Resource Management

Acknowledgments

I am very grateful for my supervisor, Dr. S.P. Simonovic for giving me the opportunity to do research in this exciting field. He has generously offered his time and followed my work with keen interest from its inception. He has shared his infinite knowledge and provided motivation to increase my own. I consider it a great privilege and honor to call myself one of his students.

I would also like to extend a special thanks to Mr. Mark Helsten from the Upper Thames Conservation Authority for contributing water reservoir operations insights and providing relevant data. In addition, I would like to thank Dr. Joran Velikonja for his detailed editing assistance. A heartfelt thanks goes out to everyone at FIDS for their pleasant company and assistance while working on my thesis.

Table of Contents

CERTIFICATE OF EXAMINATION	ii
Abstract	iii
Acknowledgments	iv
Table of Contents	v
List of Tables	viii
List of Figures	x
List of Appendices	xii
Chapter 1	1
1 Introduction	1
1.1 Organization of the Thesis	7
Chapter 2	8
2 Methodology	8
2.1 Differential Evolution Algorithm	8
2.1.1 DE Population Initialization	9
2.1.2 Mutation	11
2.1.3 Crossover	12
2.1.4 Selection	13
2.1.5 Termination	13
2.1.6 Illustrative Example of Classic DE Algorithm	14
2.2 Selected Differential Evolution Algorithm Variants	17
2.2.1 DE/best/1/bin	17
2.2.2 DE/local-to-best/1/bin	18

2.3	Setting Control Parameters	19
2.3.1	Fuzzy Adaptive Differential Evolution.....	21
2.4	Constraints	25
2.4.1	Search Space Constraint	26
2.4.2	Feasible Space Constraint.....	27
2.5	Fuzzy Differential Evolution Algorithm.....	30
2.5.1	Initialization	30
2.5.2	Mutation.....	34
2.5.3	Illustrative Example of FDE Algorithm	36
Chapter 3	42
3.1	Decision Support System Software Package	42
3.2	Differential Evolution Optimizer Overview	42
3.2.1	Algorithm Inputs	44
3.2.2	Optimization Inputs	48
3.2.3	Optimization Results.....	50
3.3	Illustrative Example	52
Chapter 4	56
4.1	Application.....	56
4.2	Benchmark Functions	56
4.2.1	Benchmark Function Results and Discussions	62
4.3	Case Study	68
4.3.1	Study Area Background.....	68
4.3.2	Problem Definition.....	71
4.3.3	Mathematical Formulation.....	71
4.3.4	Algorithm and Optimization Inputs	75
4.3.5	Study Results and Discussions	78

Chapter 5.....	84
5.1 Summary.....	84
5.2 Recommendations for Future Work.....	85
References.....	87
Appendices.....	92
Curriculum Vitae	105

List of Tables

Table 2.1. Population vector matrix for each generation.....	11
Table 2.2. An illustrative example.....	15
Table 2.3. Calculation of the weighted difference vector for the illustrative example.....	15
Table 2.4. Calculation of the mutated vector for the illustrative example.....	16
Table 2.5. Generation of the trial vector for the illustrative example.....	16
Table 2.6. New population for the next generation in the illustrative example.....	17
Table 2.7. Membership Functions	23
Table 2.8. The Fuzzy Rules	24
Table 2.9. An illustrative example.....	38
Table 2.10. Calculation of the weighted difference vector for the illustrative example.....	38
Table 2.11. Calculation of the mutated vector for the illustrative example.....	39
Table 2.12. Interval to single value mutated vector calculation	39
Table 2.13. Generation of the trial vector for the illustrative example.....	40
Table 2.14. New population for next generation for the illustrative example	41
Table 4.1. Algorithm settings.....	57
Table 4.2. Performance comparison of FDE and DE algorithms at various focus targets	63
Table 4.3. Performance comparison between the original DE algorithm with smaller bounds and FDE with a focus equal to one	66
Table 4.4. Constraints of the Wildwood reservoir (UTRCA, 1993).....	72
Table 4.5. DE algorithm inputs.....	75

Table 4.6. Storage initialization inputs for the year 2010 [10^3 m^3]	77
Table 4.7. Release initialization inputs for the year 2010 [10^3 m^3]	77
Table 4.8. Constraint satisfying release and storage target initialization inputs for the year 2010 [10^3 m^3]	77
Table 4.9. Release constraints [10^3 m^3]	78
Table 4.10. Monthly inflows for the Wildwood reservoir [10^3 m^3].....	78
Table 4.11. Wildwood reservoir objective functions and error after optimization.....	78
Table 4.12. Penalty constant selection	781

List of Figures

Figure 1.1. DE algorithm schematic.	5
Figure 2.1. Search space and feasible region.	26
Figure 2.2. Triangular fuzzy membership function.	31
Figure 2.3. The alpha-cut method schematic.	32
Figure 2.4. The alpha-cut intervals schematic.	33
Figure 3.1. Interface of DEO menu.	43
Figure 3.2. Algorithm inputs window.	44
Figure 3.3. Optimization inputs window.	48
Figure 3.4. Optimization results window.	48
Figure 3.5. Algorithm inputs for illustrative example.	53
Figure 3.6. Optimization inputs for illustrative example.	54
Figure 3.7. Optimization results for illustrative example.	54
Figure 4.1. First De Jong's function in 2 dimensions (Molga and Smutnicki, 2005).	58
Figure 4.2. Rosenbrock's function in 2 dimensions (Molga and Smutnicki, 2005).	59
Figure 4.3. Modified Third De Jong Function in 2 dimensions (Black, 1996).	60
Figure 4.4. Rastrigin's function in 2 dimensions (Molga and Smutnicki, 2005).	61
Figure 4.5. Location of the Upper Thames basin.	69
Figure 4.6. Wildwood reservoir schematic.	71
Figure 4.7. Wildwood reservoir optimization progress.	71

Figure 4.8. Wildwood reservoir storage for a twelve-month time horizon.	82
Figure 4.9. Wildwood reservoir release for a twelve-month time horizon.	83
Figure 6.1. Fuzzification of scalar input from known membership function.	97
Figure 6.2. Fuzzy operator use for the generalized expression (6.5) of a rule.	99
Figure 6.3. Aggregation of rule outputs into a single fuzzy membership function.	100
Figure 6.4. Centroid method for defuzzification.	101

List of Appendices

Appendix A: Fuzzy Set Theory	91
Appendix B: Mamdani Fuzzy Inference	95
Appendix C: Decision Support System for Implementation of DEO	101
Appendix D: Wildwood Optimization Results	102

Chapter 1

1 Introduction

Water resources systems provide water for agricultural, industrial, household, recreational and environmental activities. Beside sustaining life, water has a high social, economic, cultural and aesthetic value for humans. However, water can also become a potential threat, such as in the event of flooding caused by a sudden abundance of water. Therefore it is no surprise that there is a great need for water resource systems management. Through the management activities we can appropriately allocate the water resources, increasing economic benefits while actively assuring the health and safety of humans and related environment.

Water-related problems can be addressed through structural measures (dikes, dams, sewers, etc.), but also through policy and operation decisions. However, before implementation of these aforementioned measures can take place, utilization of an approach such as system analysis is required. System analysis is defined as a set of mathematical planning and design techniques; its introduction has been viewed as the most important advance in the field of water management in the last century (Hall and Dracup, 1970; Loucks et al., 1981; Friedman et al., 1984; Yeh, 1985; Rogers and Fiering, 1986; Loucks and da Costa, 1991; Wurbs, 1998; Simonovic, 2009). Systems analysis is particularly promising when scarce resources must be used effectively. Resource allocation problems are very common in the field of water management, and affect both developed and developing countries, which today face increasing pressure to make efficient use of their resources (Simonovic, 2009).

System analysis techniques, often called operations research, management science and cybernetics, include simulation and optimization techniques that are used to analyze the quantitative and qualitative aspects of watershed runoff and stream flow processes, reservoir system operations, groundwater development and protection, water distribution

systems, water use and various other hydrological processes and management activities (Simonovic, 2009). The latter technique, optimization, is the focus of this thesis.

Optimization is a procedure defined as the selection of a set of decision variables falling within the feasible region that maximizes/minimizes the objective function (Simonovic, 2009). Optimization is very desirable as it improves efficiency, performance and revenue which finds application in a broad spectrum of fields, most commonly economics, engineering and operations research (including water management).

Optimization problems, once formulated through the creation of the objective function (and sometimes including constraints), may be solved using a wide variety of computational techniques. Most water resources allocation problems are addressed using linear programming (LP) solvers introduced in the 1950s (Dantzig, 1963). The objective function in the context of water management is usually to find the economically efficient water allocation (water supply, hydropower generation, irrigation, etc.) within a given time period in complex water systems (Simonovic, 2009). However, neither objective functions nor constraints are in a linear form in most practical water management applications. Many modifications have been used in real applications in order to convert nonlinear problems for the use of LP solvers. Examples include different schemes for the linearization of nonlinear relationships and constraints, and the use of successive approximations.

Nonlinear programming is an optimization approach used to solve problems when the objective function and the constraints are not all in linear form (Bazaraa et al., 2006). In general, the solution to a nonlinear problem is a vector of decision variables which optimizes a nonlinear objective function subject to a set of nonlinear constraints. No single universally applicable algorithm exists, that would solve every specific problem fitting this description. However, substantial progress has been made for some important special cases by making various assumptions about these functions. Successful applications are available for special classes of nonlinear programming problems such as unconstrained problems, linearly constrained problems, quadratic problems, convex problems, separable problems, non-convex problems and geometric problems.

The main limitation in applying nonlinear programming to water management problems lies in the fact that nonlinear programming algorithms generally are unable to distinguish between local optimum and global optimum (except by finding another better local optimum) (Simonovic, 2009). Therefore, where a global optimum solution is required, nonlinear programming may prove to be very inefficient due to the duration of computation.

Dynamic programming (DP) offers advantages over other optimization tools because the shape of the objective function and constraints do not affect it; hence, it has been used frequently in water management (Simonovic, 2009; Sniedovich, 2011). DP requires discretization of the problem into a finite set of stages. At every stage a number of possible conditions of the system states are identified and an optimal solution is identified at each individual stage, given that the optimal solution for the next stage is available. An increase in the number of discretizations and/or state variables would increase the number of evaluations of the objective function, as well as the core memory requirement per stage. This problem of rapid growth of computer time and memory requirement associated with multiple-state-variable DP problems is known as “the curse of dimensionality” (Sniedovich, 2011). This expression refers to the exponential growth of the search space volume as a function of dimensionality.

In the very recent past, most optimization practitioners and researchers have been looking for new approaches that combine efficiency and ability to find the global optimum. One group of such optimization algorithms, known as evolutionary algorithms (EA) has received praise for its efficiency and ability to find the global optimum for complex nonlinear systems (Back, 1996; Simonovic, 2009). Evolutionary algorithms are based on the biological evolutionary process and are therefore inherently stochastic in nature. In this concept, a population of individuals, each representing a search point in the space of feasible solutions, is exposed to a collective learning process, which proceeds from generation to generation. The population is arbitrarily initialized and subjected to the process of selection, recombination/crossover and mutation through stages known as generations, such that newly created generations evolve towards more favorable regions of the search space. The algorithm resembles the Darwinian concept known as “the

survival of the fittest”. This group of algorithms includes, among others, evolution strategies (ES) (Back, 1996), differential evolution (DE) (Storn and Price, 1995), evolutionary programming (EP) (Fogel et al, 1966; Fogel, 2005), genetic algorithms (GA) (Holland, 1975), and simulated annealing (Kirkpatrick et al, 1983; Lockwood and Moore, 1993).

Evolutionary algorithms have significant advantages over the other optimization methods discussed. Unlike LP, they are able to deal with complex nonlinear problems. Also, they are very likely to generate several solutions that are very close to the global optimum, as opposed to nonlinear programming, and, although not immune from the “curse of dimensionality”, they do not suffer from it to the extent of DP (Yu and Gen, 2010). In addition, evolutionary algorithms do not need an initial solution, and are able to produce acceptable results over longer time horizons (Simonovic, 2009). However, despite its ability to deal with unconstrained optimization problems very efficiently, EA suffers limitations like most traditional optimization techniques when dealing with constrained optimization problems. Most commonly, these limitations have been addressed by integrating additional algorithms with EA, such as the penalty function method, in order to transform a constrained optimization problem into an unconstrained one (Gen and Chen, 1997).

One of the above mentioned evolutionary algorithms, the differential evolution (DE) (Storn and Price, 1995; Storn and Price, 1997; Lampinen et al., 2005), is the main focus of this thesis. It has gained increasing popularity for solving optimization problems due to its robustness, simplicity, easy implementation and fast convergence. DE has been successfully applied to water resource management, mechanical engineering, sensor networks, scheduling and other domains (Arunachalam, 2008; Ilonen et al, 2003; Joshi and Sanderson, 1999; Onwubolu, 2008; Pan et al, 2009; Rogalsky et al, 1999; Storn, 1996).

DE utilizes a parallel direct search method for generating population vectors for each generation G from NP , D -dimensioned parameter vectors, where NP is the number of members in a population which is fixed throughout the optimization process and D is the

number of optimization parameters known as individuals. The population vector is given as:

$$X_{i,G}, i = 1, 2, \dots, NP \quad (1.1)$$

Initialization of the algorithm occurs once the initial vector population is chosen at random from an assumed parameter range (i.e. a range of integers from -10 to 10). Alternatively, if the preliminary solution is known, the population vector is populated using a normally distributed random deviation to the nominal solution, $X_{nom,0}$. The initially generated population ($X_{i,0}$) is perturbed using mutation and crossover, leading to the evolution of a new trial population. A selection process takes place to determine the fittest population of the two. The fittest population is selected as the initial population for the subsequent generation. This process continues iteratively until a termination condition is met. Fig 1.1 summarizes the main components of the algorithm.

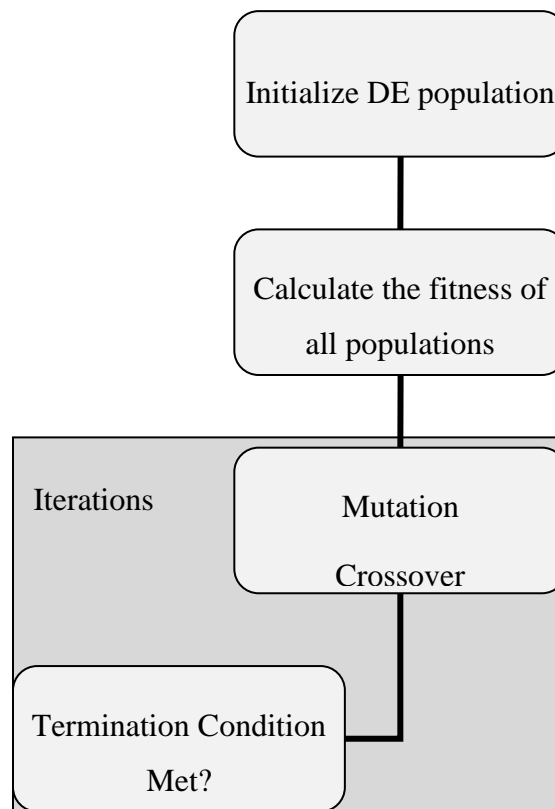


Figure 1.1. DE algorithm schematic.

The initialization strategies currently used with the DE address two specific scenarios: certainty or uncertainty. When preliminary information is available with certainty, the algorithm may be initialized using the nominal solution as discussed (Lampinen et al., 2005). Otherwise, if preliminary information is not available, the initialization will have to rely on a range of possible solutions (Lampinen et al., 2005).

However, when vague preliminary knowledge of the problem domain is available, neither method for initialization is ideal. Using such vague information to assume a nominal solution incorrectly implies more certainty than available. Alternatively assuming a range of solutions accounts for the uncertainty but may not utilize all available information to represent it correctly. The more knowledge one includes, the less uncertain will be the initialization and, consequently, the optimization.

The fuzzy set theory (Zadeh, 1965) offers a means to address the quantification of uncertainty from the available vague information. A brief overview of the main concepts of this theory is given in Appendix A. The fuzzy set theory offers unique possibilities for modifications of the traditional fundamentally stochastic DE algorithm. Some fuzzy practitioners have been already involved with evolutionary optimization. Some have utilized the existing algorithm to develop fuzzy models, like Kisi (2004) who found the parameters of membership functions for daily suspended sediment modeling. Others have joined the ongoing research that has resulted in modifications of the classic DE algorithm, such as Liu and Lampinen (2004). They proposed a fuzzy adaptive parameter control algorithm, based on feedback from the search behavior, to address the sensitivity of the DE to control parameter settings.

The objective of this thesis is to create a new DE initialization strategy that will be able to take advantage of the existing knowledge in the problem domain. The more knowledge is included, the more likely it becomes for the optimization to converge more efficiently. In conjunction with the new initialization technique, the mutation scheme will require modification in order to properly offer valuable guidance to the DE algorithm towards a

more efficient search strategy. For convenience, the novel algorithm is implemented into optimization decision support system software.

1.1 Organization of the Thesis

This thesis is organized into four additional chapters. Chapter Two gives an overview of the methodological background of the classic differential evolution algorithm strategy, as well as of several other selected strategies. An illustrative numerical example of the classic differential evolution algorithm is also presented here. The chapter also contains guidelines for setting the DE control parameters based on empirical evidence; in addition, the fuzzy adaptive differential evolution methodology is detailed. Constraint handling methodologies are also overviewed, including the random and bounce-back reinitialization approach for dealing with search space constraints and the penalty function method for dealing with feasible space constraints. Lastly, the methodology for the novel fuzzy differential evolution (FDE) algorithm for initialization and mutation is proposed. This approach uses prior knowledge of the problem domain for guiding the search towards the optimal solution. A numerical example of the fuzzy differential algorithm is also presented for illustrative purposes.

Chapter Three outlines the optimization decision support system software package developed by integrating all the features discussed in the methodology. An illustrative example is used to demonstrate the decision support system and a typical procedure required to find the optimal solution. Chapter Four details two applications of the novel fuzzy differential evolution algorithm. Included is the application of a set of standard benchmark functions, used to compare the performance of the classical DE algorithm (in terms of convergence speed) with the proposed FDE algorithm. The second example is a practical application of the proposed algorithm using a reservoir operation case study. The final Chapter Five is a summary of key contributions/findings with a view into possible directions for future research aimed at expanding the FDE concept.

Chapter 2

2 Methodology

In the following sections of this chapter an overview of the original differential evolution algorithm is presented, alongside several other common variants. Presented is also an overview of control parameter selection strategies. Additionally, the approach for handling constraints is detailed. Lastly, the contribution of this thesis, the novel fuzzy differential evolution algorithm methodological background is detailed.

2.1 Differential Evolution Algorithm

The DE algorithm after initialization has three main operations: (I) mutation, (II) crossover and (III) selection before finishing due to a termination condition. The fundamental idea behind DE is a specific way of generation of trial parameter vectors. This is achieved using mutation and crossover to generate new trial parameter vectors. Selection then determines which of the vectors will survive to be used in the next generation. Through repeated cycles of mutation, crossover and selection, DE is able to guide the search towards the vicinity of the global optimum.

The original DE algorithm scheme proposed by Storn and Price (1995) gave the working principles of DE. Subsequently, contributions of other variants or strategies have been made and continue to be made. Different DE strategies can be adopted in the DE algorithm depending upon the type of problem to which DE is applied. The strategies can vary based on the vector to be perturbed, the number of difference vectors considered for perturbation and the type of crossover used.

In order to differentiate the family of various available strategies for DE, a general notation convention used is $DE/x/y/z$ (Price and Storn, 1997). DE stands for Differential Evolution, it distinguishes that the notation presented follows the differential evolution algorithm principles. The x variable represents a string (rand:random;best), denoting how the vector is to be perturbed either using the best vector of the previous generation or

using any randomly chosen vector. The y variable is the number of difference vectors considered for the perturbation of x . Hence if it is a single vector difference, three distinct randomly chosen vectors are required, because the weighted differential of two vectors is added to the third one. Lastly, z stands for the type of crossover used: either exponential (exp) or binomial (bin). If exponential crossover is chosen, the crossover is performed on the D variables in one loop until it is within a given bound represented by the control parameter CR (crossover rate). The first time a randomly picked number between 0 and 1 exceeds the CR value, crossover is halted and the remaining D variables are left intact. If the crossover is binomial, it is performed on each of the D variables whenever a randomly picked number between 0 and 1 is within the CR value. Therefore for high values of CR , the exponential and binomial crossover methods yield similar results. In practice, the binomial crossover approach is used more frequently.

The performance of the various DE variants is highly dependent on the given problems, so that a suitable one for any particular problem may not be as suitable for another. This assertion is reinforced by the no free lunch theorem (NFL) which states that no single search algorithm exists that can solve all problems efficiently (Wolpert and Macready, 1997). With that in mind, the importance and amount of research into strategies and control parameters for the best convergence efficiency is hardly surprising. The strategy and control parameter selection with best performance for a given problem is typically unknown, though some guidance exists. The usual approach is trial-and-error. However, the original DE algorithm strategy, under the notation DE/rand/1/bin by Storn and Price (1995), appears to be the most successful and the most widely used. The following presentation is based on the original/classic DE scheme.

2.1.1 DE Population Initialization

A common starting point with implementing any evolutionary algorithm is the initialization of the population. Initialization has two main issues that need to be decided upon: (a) “How to initialize each gene of the individual?” and (b) “How many genes should be used in the population?” (Iba and Noman, 2012). Discussed here is only the

first issue. The latter, which is related to population size, a critical parameter of DE, will be focused on in Section 2.3.

As stated in earlier sections, each gene of each individual is initialized using a uniform random generator within the search ranges. This concept is the same for all evolutionary algorithms and DE is no exception. Let us assume that we are working in a D -dimensional problem. Then each individual of the DE population, P_G , would be a D -dimensional vector which can be initialized as follows:

$$x_{i,t}^{G=1} = \text{Rand}_t(LB_t, UB_t) \quad (2.1)$$

Such that $x_{i,t}^{G=1}$ denotes the t th gene ($t=1,2,\dots,D$) of the i th individual ($i=1,2,\dots, NP$) in generation $G=1$. $\text{Rand}_t(a,b)$ denotes the uniform random number generator that returns a uniformly distributed random number from $[a, b]$. The subscript in Rand_t is used to clarify that a separate random number is drawn for each gene in each individual. LB_t and UB_t denote the lower and upper limits of the search ranges for gene j , respectively. It is critical that the bounds are set sufficiently high enough, so that the initial bounding box contains the optimum solution. In many cases the existence of natural physical limits or logical constraints makes prescribing bounds for each parameter straightforward. In circumstances where the bounds for a specific parameter are not known this may be particularly difficult.

A population vector with its gene and individual components is presented in Table 2.1 for clarity.

Table 2.1. Population vector matrix for each generation

Gene	1	2	D
Individual			
1	$X_{1,1}$	$X_{1,2}$	$X_{1,D}$
2	$X_{2,1}$	$X_{2,2}$	$X_{2,D}$
NP	$X_{NP,1}$	$X_{NP,2}$	$X_{NP,D}$

2.1.2 Mutation

DE derived its name from the mutation operator it applies to mutate its individual. Mutation is the first of two main operators (the other being crossover) required to alter the “genetic code” of current individuals to improve diversity of a population. A mechanism for evolving the population of vectors is essential. There is the possibility that re-selection of vectors already chosen can occur along with other vectors being omitted from the search. Vectors that are not chosen are deprived of passing on potential diversity to the next generation. Re-selection of vectors causes the potential to lose diversity in the next generation due to over sampling of the same vector. DE ensures that this does not happen by comparing vectors from competing populations by their index.

The mutation operator is called “differential mutation” and generates the mutated individual (also known as mutated vector) $m_{i,G+1}$, for the principal parent (also known as target vector) $x_{i,G}$ according to the following equation (Storn and Price, 1997):

$$m_{i,G+1} = x_{r1,G} + F(x_{r2,G} - x_{r3,G}) \quad r1 \neq r2 \neq r3 \quad (2.2)$$

where $F \in [0, 2]$ is a real number that controls the amplification of the difference vector $(x_{r2, G} - x_{r3, G})$, while $r1, r2, r3 \in [1, NP]$ represent randomly chosen indexes, where $r1$ corresponds to the base vector. The indexes have to be different from each other and from the running index i . That way, a parent pool of four individuals is required to breed an offspring.

2.1.3 Crossover

To complement the differential mutation search strategy, DE then uses a crossover operation, in which the mutated individual is mated with the principal parent and generates the offspring or “trial individual”. This crossover operation for classic DE as reviewed here is known as binomial crossover.

The target vector $x_{i,G}$ is mixed with the mutated vector, $m_{i,G}$, using the following scheme, to yield the trial vector (Storn and Price, 1997)

$$u_{i,G+1} = (u_{i,1,G+1}, u_{i,2,G+1}, \dots, u_{i,D,G+1}) \quad (2.3)$$

where

$$u_{i,G+1} = \begin{cases} m_{i,t,G+1} & \text{if } rand_t < CR \text{ or } t = rn_i \\ x_{i,t,G} & \text{if } rand_t > CR \text{ and } t \neq rn_i \end{cases} \quad (2.4)$$

CR is the crossover constant $\in [0, 1]$ (to be specified by the user), $t=1, 2, \dots, D$ and $rand_t$ is the t th evaluation of a uniform random generator number $\in [0, 1]$. Lastly, to guarantee that a new altered population vector is produced, a randomly chosen index $rn_i \in [1, 2, \dots, D]$ is used, ensuring that $u_{i,G+1}$ gets at least one element from $m_{i,G+1}$.

2.1.4 Selection

DE uses a selection mechanism to ensure that the individuals promoted to the next generation are strictly those with the best fitness values in the population. A knockout competition is played between each individual (target vector) $x_{i,G}$ and its offspring (trial vector) $u_{i,G+1}$. The survival criteria can be described as follows (Storn and Price, 1997):

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & \text{if } f(u_{i,G+1}) < f(x_{i,G}) \\ x_{i,G}, & \text{otherwise} \end{cases} \quad (2.5)$$

where $f(\cdot)$ indicates the objective function that is being optimized (minimized here). This one-to-one selection mechanism ensures that the selected individuals are strictly those with the best fitness values in the population. That is to say, the trial vector $u_{i,G+1}$ must yield a better fitness value than $x_{i,G}$, for $x_{i,G+1}$ to be set to $u_{i,G+1}$; otherwise, the old value $x_{i,G}$ is retained. Practicing this one-to-one selection mechanism thus enables DE to exercise elitism on its population. Due to its positional elitism strategy it discards an offspring which is better than most of the current population but worse than its parent. However, such rejected individuals could be useful to accelerate the search for the global optimum (Iba and Noman, 2012).

2.1.5 Termination

Termination of the algorithm ideally takes place after the global optimum is achieved, but this may not always be the case. Frequently, termination of the algorithm is a user-defined input and the user can limit the number of iterations of the algorithm. This is a trial-and-error approach, in that a sufficient number of iterations are required to ensure the best known results are returned. Another method for termination is when the objective has been met. In some objective functions, the optimal value can already be known. For example, some functions such as benchmark functions may have a known minimum value, meaning as soon as the search algorithm reaches this known minimum value it will

terminate. Additionally, feedback provided by the objective function can determine that no further optimization is possible. For example, if the optimization stalls and thus many subsequent objective function values are the same, the algorithm may be terminated. Also, human monitoring can determine when optimization is over.

2.1.6 Illustrative Example of Classic DE Algorithm

A simple numerical example adopted from Arunachalam (2008) is presented to illustrate the classic DE algorithm. Let us consider the following objective function for optimization:

$$\text{Minimize } f(x) = x_1 + x_2 + x_3 \quad (2.6)$$

The initial population is chosen randomly between the bounds of decision variables, in this case x_1 , x_2 and $x_3 \in [0, 1]$. The population along with its respective objective function values is shown in Table 2.2. The first member of the population, “Individual 1”, is set as the target vector.

In order to generate the mutated vector, three individuals (“Individual 2”, “Individual 4” and “Individual 6”) from the population size are selected randomly (ignoring “Individual 1”, since it is set as the target vector). The weighted difference between “Individual 2” and “Individual 4” is added to the third randomly chosen vector “Individual 6” to generate the mutated vector. The weighting factor F is chosen as 0.80 and the weighted difference vector is obtained in Table 2.3 and the mutated vector in Table 2.4.

Table 2.2. An illustrative example

Population Size NP=6 (user defined), D=3						
	Individual	Individual	Individual	Individual	Individual	Individual
	1	2	3	4	5	6
x_1	0.68	0.92	0.22	0.12	0.40	0.94
x_2	0.89	0.92	0.14	0.09	0.81	0.63
x_3	0.04	0.33	0.40	0.05	0.83	0.13
$f(x)$	1.61	2.17	0.76	0.26	2.04	1.70

Table 2.3. Calculation of the weighted difference vector for the illustrative example

	Individual	Individual	Difference	Weighted
	2	4	Vector	Difference
				Vector
x_1	0.92	0.12	= 0.80	= 0.64
x_2	0.92	- 0.09	= 0.83	= 0.66
				(F= 0.80)
x_3	0.33	0.05	= 0.28	= 0.22

Table 2.4. Calculation of the mutated vector for the illustrative example

	Weighted Difference Vector		Individual 6	Mutated Vector
x_1	0.64		0.94	= 1.58
x_2	0.66	+	0.63	= 1.29
x_3	0.22		0.13	= 0.35

The mutated vector does a crossover with the target vector to generate the trial vector, as shown in Table 2.5. This is carried out by (1) generating random numbers equal to the dimension of the problem (2) for each of the dimensions: if random number $> CR$; copy the value from the target vector, else copy the value from the mutated vector into the trial vector. In this example, the crossover constant CR is chosen as 0.50.

Table 2.5. Generation of the trial vector for the illustrative example

	Target Vector		Mutated Vector	Trial Vector
x_1	0.68		1.58	= 1.58
x_2	0.89	Crossover	1.29	= 0.89
x_3	0.04	(CR= 0.50)	0.35	= 0.04
$f(x)$	1.61		3.22	2.51

The objective function of the trial vector is compared with that of the target vector and the vector with the lowest value of the two (minimization problem) becomes “Individual 1” for the next generation. To evolve “Individual 2” for the next generation, the second member of the population is set as target vector (see Table 2.6) and the above process is repeated. This process is repeated NP times until the new population set array is filled,

which completes one generation. Once the termination criterion is met, the algorithm ends.

Table 2.6. New population for the next generation in the illustrative example

	New Population for the Next Generation					
	Individual	Individual	Individual	Individual	Individual	Individual
	1	2	3	4	5	6
x_1	0.68					
x_2	0.89					
x_3	0.04					
$f(x)$	1.61					

2.2 Selected Differential Evolution Algorithm Variants

In addition to the classical DE strategy DE/rand/1/bin, there are many derivative strategies for perturbation of the population vectors. The motivation to develop such strategies has come from the fact that no single perturbation method has turned out to be best for all problems (Chakraborty, 2008). Discussed here is DE/best/1/bin and DE/current(local)-to-best/1/bin, two very popular mutation strategies for addressing optimization problems that the original strategy may not perform adequately. These two strategies benefit in faster convergence by incorporating the best solution information in the evolutionary search. However the best solution information may also cause problems such as premature convergence due to the resultant decreased population diversity.

2.2.1 DE/best/1/bin

The strategy DE/best/1/bin is very popular. It was proposed after the initial formulation of the DE algorithm (Price, 1996). The fundamental difference between the original DE

scheme and this variant is based on the perturbation of the vectors. In the DE/best/1/bin scheme only the mutation component of the algorithm is modified with respect to the original, incorporating information from the objective function. Instead of randomly populating the base vector from randomly chosen indexes in the current generation (as in the original scheme), in DE/best/1/bin the algorithm always selects the best-so-far vector (best) as the base vector, adds a single scaled vector difference to it, then creates a trial vector by uniformly crossing the resulting mutant with the target vector. Thus the base vector always has the best (fittest) objective function value in the current population. Compared to random base vector selection, using the best-so-far vector lowers the diversity of the pool of potential trial vectors (Lampinen et al., 2005).

The above description is expressed in the formula below, where for each target vector $x_{i,G}$, a mutation vector $m_{i,G}$ is generated according to (Price, 1996)

$$m_{i,G+1} = x_{best,G} + F(x_{r1,G} - x_{r2,G}) \quad r1 \neq r2 \quad (2.7)$$

where $F \in [0, 2]$ is a real number that controls the amplification of the difference vector $(x_{r1,G} - x_{r2,G})$ and $r1, r2 \in [1, NP]$ represent randomly chosen indexes. The indexes have to be different from each other and from the running index i so that NP must be at least three. $x_{best,G}$ corresponds to the best vector from the best population solution in the current generation.

2.2.2 DE/local-to-best/1/bin

This DE variant computes the difference between the i th member (target vector) and the best-so-far member of the current population (Lampinen et al, 2005). This method attempts to balance robustness with fast convergence and is a popular choice in most studies of DE.

$$m_{i,G+1} = x_{i,G} + F(x_{best,G} - x_{i,G}) + F(x_{r1,G} - x_{r2,G}) \quad r1 \neq r2 \quad (2.8)$$

2.3 Setting Control Parameters

Control parameters have already been briefly mentioned, but due to their importance to the performance of DE algorithms a more detailed explanation is given here. The values of population size (NP), crossover constant (CR) and weighing factor or mutation scale factor (F) are fixed empirically, following certain heuristics. Proper tuning of these parameters is essential for the reliable performance of the algorithm. Trying to tune these three main control parameters and finding bounds for their values has been a topic of intensive research (Chakraborty, 2008).

The mutation scale factor F controls the speed and robustness of the search. A lower value for F increases the convergence rate but it does so at the risk of getting stuck into a local optimum and therefore failing to find the true global solution. Parameters CR and NP have a similar effect on the convergence rate as F . High values of CR favor a higher mutated element crossover to current elements; as a result, the mutation factor F has a greater impact on the search. As well, an increased value of NP increases the diversity of the population and with it the potential to find the true optimal solution from the greater search space but at the cost of longer computation time.

The control parameter selection is a difficult task due to their interdependence with each other and the fact that some objective functions are sensitive to proper settings (Liu and Lampinen, 2002). Traditionally, the control parameters have been held fixed during the whole execution of the algorithm.

The rule-of-thumb values for the control parameters given by Storn and Price (1997) for F is usually between 0.5 and 1.0 and CR between 0.8 and 1.0. These authors have proposed that the population size NP should be between $5 \times D$ and $10 \times D$ and not less than 4 to ensure that the mutation operation can be carried out. If mis-convergence occurs, NP should be increased; however, beyond a certain limit it is not useful to increase the population size any more (Iba and Noman, 2012). The suggestions by Storn and Price for the control parameters are valid for many practical purposes but still lack generality. This means that, in practice, many time-consuming trial runs are required to find optimal

parameters for each problem setting. As a result of the difficulty of setting appropriate control variables, research has focused on finding parameters such as F and CR settings automatically (Zhang and Sanderson, 2009).

For example, Brest et al. (2006) proposed a self-adaptive version of DE that automatically adjusts its control parameters F and CR at an individual level. Likewise, a feedback update rule for F was proposed by Zaharie (2003), designed to maintain the population diversity at a given level, thereby reducing a premature convergence of the search. Fuzzy adaptive differential evolution (FADE), introduced by Liu and Lampinen (2004), is another example of methods that determine the control parameters automatically and is discussed in detail in the following section.

2.3.1 Fuzzy Adaptive Differential Evolution

Fuzzy logic is a means of transforming linguistic knowledge into a mathematical model. It has been used extensively in the field of automatic control where it succeeded in the modeling and control of many systems that cannot be described using classical control techniques. Therefore fuzzy logic offers a means of rendering control parameters more adaptive to each optimization problem. The result of implementing fuzzy adaptive differential evolution (FADE) is a more efficient search (a lesser number of function evaluations) (Liu and Lampinen, 2004).

FADE uses a fuzzy knowledge-based system to adapt dynamically the control parameters F and CR for the mutation and crossover operations. It uses a series of fuzzy rules developed based on existing empirical evidence to infer appropriate values of F and CR for each generation, based on parameter and objective function difference vector from subsequent generations. The adaptive parameters using FADE accelerate the convergence velocity of DE.

FADE uses Mamdani's inference method to establish the control parameter (Mamdani and Assilian, 1975). Mamdani's fuzzy inference method is detailed in Appendix B.

FADE establishes inputs for fuzzy inference by using the mean square root concerning the change between successive generations over the whole population during the optimization process:

$$PC = \sqrt{\frac{1}{NP} \sum_{i=1}^{NP} \sum_{j=1}^D (x_{i,j}^{(n)} - x_{i,j}^{(n-1)})^2} \quad (2.9)$$

$$FC = \sqrt{\frac{1}{NP} \sum_{i=1}^{NP} (f_i^{(n)} - f_i^{(n-1)})^2}$$

and

$$\begin{aligned}
 d_{11} &= 1 - (1 + PC) * e^{-PC} \\
 d_{12} &= 1 - (1 + FC) * e^{-FC} \\
 d_{21} &= 2 * (1 - (1 + PC) * e^{-PC}) \\
 d_{22} &= 2 * (1 - (1 + FC) * e^{-FC})
 \end{aligned} \tag{2.10}$$

where PC is called the parameter vector change in magnitude and is transformed into the range of $[0,1]$ as d_{11} and the range of $[0,2]$ as d_{21} ; FC is called the function value change and is transformed into $[0,1]$ as d_{12} and $[0,2]$ as d_{22} ; $f_i^{(n)}$ is the i th component of the function value vector for the n th generation, $i = 1, 2, \dots, NP$; $x_{i,j}^{(n)}$ is the component in the i th row and j th column of the parameter matrix $X_{NP \times D}$ for the n th generation, $i = 1, 2, \dots, NP$, $j = 1, 2, \dots, D$; n is the generation index; NP and D represent the population size and dimensionality of the problem, respectively.

Actual input values for the fuzzy inference are the numerical values as stated in Eq. (2.10); output variables are the parameter values for F and CR , whose ranges are sets of real numbers.

Each of the variables (d_{11} , d_{12} , d_{21} , d_{22} , F , CR) has a corresponding fuzzy membership function with 3 fuzzy subsets, where S is “small”, M is “middle” and B is “big”. These membership functions are developed by Lampinen and Liu (2004), based on existing empirical knowledge. A Gaussian curve membership function, f_g is used for every input and output and is defined in Table 2.7.

Table 2.7. Membership Functions

Inputs, Outputs	Membership Functions
d_{11}	$\mu_S(d_{11}) = f_g(d_{11}, 0.25, 0.05)$
	$\mu_M(d_{11}) = f_g(d_{11}, 0.25, 0.5)$
	$\mu_B(d_{11}) = f_g(d_{11}, 0.25, 0.9)$
d_{21}	$\mu_S(d_{21}) = f_g(d_{21}, 0.5, 0.1)$
	$\mu_M(d_{21}) = f_g(d_{21}, 0.5, 0.8)$
	$\mu_B(d_{21}) = f_g(d_{21}, 0.5, 1.5)$
d_{12}	$\mu_S(d_{12}) = f_g(d_{12}, 0.35, 0.1)$
	$\mu_M(d_{12}) = f_g(d_{12}, 0.35, 0.5)$
	$\mu_B(d_{12}) = f_g(d_{12}, 0.35, 0.9)$
d_{22}	$\mu_S(d_{22}) = f_g(d_{22}, 0.5, 0.1)$
	$\mu_M(d_{22}) = f_g(d_{22}, 0.5, 0.8)$
	$\mu_B(d_{22}) = f_g(d_{22}, 0.5, 1.5)$
F	$\mu_S(F) = f_g(F, 0.5, 0.3)$
	$\mu_M(F) = f_g(F, 0.5, 0.6)$
	$\mu_B(F) = f_g(F, 0.5, 0.9)$
CR	$\mu_S(CR) = f_g(CR, 0.35, 0.4)$

	$\mu_M(CR) = f_g(CR, 0.35, 0.7)$
	$\mu_B(CR) = f_g(CR, 0.35, 1.0)$

The values of F and CR are adapted based on d_{11} , d_{12} , d_{21} , d_{22} and a series of fuzzy rules used to describe the characteristics of the system. There are a total of 18 rules for determining F and CR values, 9 each. Each rule has two inputs and one output which represent the mapping from the input space to the output space. The “9×2” rules are given in Table 2.8.

Table 2.8. The Fuzzy Rules

Rule	Fuzzy Sets		
	d_{i1}	d_{i2}	F or CR
1	S	S	S
2	S	M	M
3	S	B	B
4	M	S	S
5	M	M	M
6	M	B	B
7	B	S	B
8	B	M	B
9	B	B	B

Note: S = small; M = middle; B = big, $i = 1, 2$ the first and second fuzzy logic control system. d_{ij} = the j th input of the i th fuzzy logic control system.

Finally, the adaptive parameters may be found given the supplied information and Mamdani's inference in conjunction with a centroidal defuzzification technique.

Defuzzification is mapping from a space of fuzzy output into a space of real output. The result is a single number y^* which represents the value of the mutation amplification F or crossover factor CR .

2.4 Constraints

Constrained optimization problems, especially nonlinear optimization problems, where objective functions are to be optimized under given constraints, are very important and frequently appear in the real world. For this reason, DE has had significant research invested into dealing with optimization problems, with inequality constraints, equality constraints, as well as upper and lower bound constraints (Chakraborty, 2008; Lampinen et al., 2005). Constrained optimization problems are mathematically expressed as

$$\begin{aligned}
 & \text{minimize(maximize)} \ f(x) & (2.11) \\
 & \text{subject to} \\
 & g_j(x) \leq 0, \quad j = 1, \dots, n \\
 & h_j(x) = 0, \quad j = n + 1, \dots, m \\
 & l_i \leq x_i \leq u_i, \quad i = 1, \dots, k
 \end{aligned}$$

Where $x = (x_1, x_2, \dots, x_k)$ is a k -dimensional vector, $f(x)$ is an objective function, $g_j(x) \leq 0$ and $h_j(x) = 0$ are n inequality constraints and m equality constraints, respectively.

Functions f , g_j and h_j are linear or nonlinear real-valued functions. Values u_i and l_i are upper and lower bounds of x_i , respectively.

Discussed here will be the methodological background on defining constraints for (I) the feasible space in which every point satisfies constraint functions denoted by F and (II) the

search space in which every point satisfies upper and lower boundary constraints denoted by $S(\supset F)$. Fig. 2.1 shows graphically the search space and feasible region.

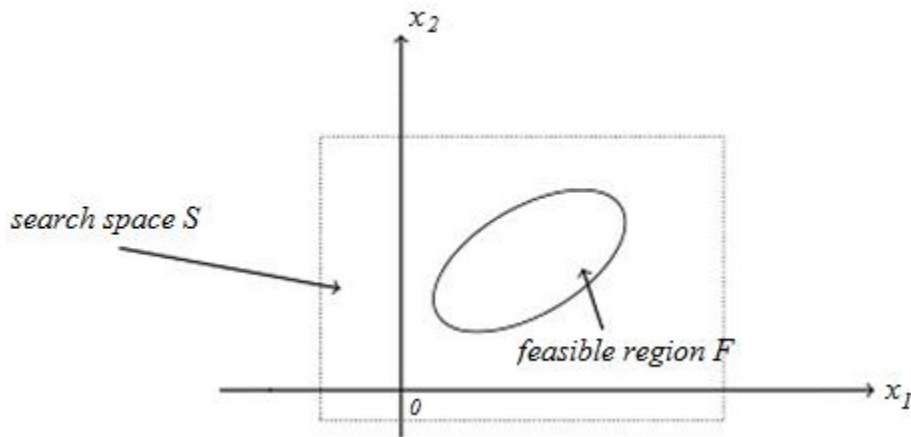


Figure 2.1. Search space and feasible region.

2.4.1 Search Space Constraint

After initialization, the algorithm may produce mutated vectors in subsequent generations that fall outside of the initial search boundaries. The initial search bounds give information on the assumed feasible search space for the problem and thus can be used to define the low and high limits put on each individual. In some cases it may be desirable for the search to be able to have the freedom to surpass the set bounds. This may be in instances where the search space is improperly preset due to a lack of knowledge about the problem domain. However, in all other cases this is harmful and non-desirable. For example, a negative value for discharge for a reservoir operation problem is absolutely inadmissible; as such, the lower bound constraints must be maintained, $LB_i = 0$.

Two approaches are surveyed here for regularization of infeasible mutant vectors. These fall into the hard constraint handling methods, where the infeasible solutions are rejected. The first approach is random reinitialization. Any infeasible optimization parameter value of the mutant vector, $m_{i,G+1}$ that does not fall within upper and lower bounds is replaced by a value randomly generated with a uniform distribution from the initial bounds.

$$m_{i,t}^{G+1} = \begin{cases} \text{Rand}_t(LB_t, UB_t) & \text{if } m_{i,t}^{G+1} \notin [LB_t, UB_t] \\ m_{i,t}^{G+1} & \text{otherwise} \end{cases} \quad (2.12)$$

The other approach to regularize infeasible mutant vectors is called bounce-back. Bounce-back replaces the offending parameter with another, chosen between the boundary and the base vector.

If the mutated vector exceeds the lower bound:

$$m_{i,G+1} = x_{r1,G} + \text{Rand}_i(LB - x_{r1,G}) \quad (2.13)$$

If the mutated vector exceeds the upper bound:

$$m_{i,G+1} = x_{r1,G} + \text{Rand}_i(UB - x_{r1,G}) \quad (2.14)$$

Bounce-back may be preferred over random reinitialization as it is able to preserve the direction of the current search. As a result, the convergence speed using bounce-back may be favorable to random reinitialization.

2.4.2 Feasible Space Constraint

Some problems have constraint functions which cannot be dealt with utilizing the search space boundary constraints. The penalty function method is widely used for constrained optimization problems, not just in differential evolution algorithms but in other optimization algorithms as well. The penalty function method transforms the constrained problem into an unconstrained one by penalizing infeasible solutions, in which a penalty term is added to the objective function for any violation of the constraints (Gen and Chen, 1997).

The additional penalties added to the objective function force the solution to fall into the feasible space after a few generations. This results from solutions that have the penalty

added on to the objective failing in order to compete with solutions without penalty in the selection process of DE. It needs to be emphasized that infeasible solutions may not be rejected outright in each generation, as they may provide much more useful information about optimal solution than some feasible solutions. The major concern is how to determine the penalty term so as to strike a balance between keeping some infeasible solutions and rejecting others. An overly low penalty term constant may keep too many infeasible solutions, whereas a very high penalty constant may reject all the solutions preventing the optimization from convergence to an optimal solution.

Careful selection of the penalty control parameters is required for the proper convergence to a feasible optimal solution and is very much problem-dependent.

The differential evolution algorithm is modified to take account of constraint functions using the penalty function method. The fitness function modified for taking account of the penalty function may be expressed as follows (Gen and Chen, 1997):

$$eval(x) = f(x) + p(x) \quad (2.15)$$

where x represents the genes parameter vector, $f(x)$ the objective function of the problem and $p(x)$ the penalty function. For an optimization problem, it is required that

$$p(x) = 0 \text{ if } x \text{ is feasible} \quad (2.16)$$

$$p(x) > 0 \text{ for minimization problems}$$

$$p(x) < 0 \text{ for maximization problems}$$

To demonstrate how the function in Eq. (2.16) may be formulated consider the example problem where the initial parameter values for x_1 and x_2 are found to be 5 and 2 respectively:

$$\text{minimize } f(x) = x_1^2 + x_2^2 \quad (2.17)$$

subject to

$$x_1 \leq 4$$

$$x_1 + x_2 = 1$$

The above two constraints would be transformed to an unconstrained problem and multiplied by a penalty constant as follows:

$$\text{eval}(x) = 5^2 + 2^2 + P_1(5 - 4) + P_2(5 + 2 - 1) \quad (2.18)$$

where P_1 and P_2 are the user specified penalty constants for each constraint, these values for convenience can be chosen the same, let say $P_1 = P_2 = 10$. The terms in parentheses in the penalty functions are the values of the constraint violations. Evaluating Eq. (2.18) yields a fitness value of 99, much less favorable for a minimization problem than if the solution were feasible. In such a case, no constraints would be violated which would result in a fitness value of 29.

2.5 Fuzzy Differential Evolution Algorithm

The novel fuzzy differential evolution (FDE) algorithm proposed here allows a novel approach for additional problem domain information to be communicated to the DE algorithm for optimization. Doing so results in better overall performance.

Differential evolution is fundamentally a stochastic based algorithm. The name FDE may suggest a full deviation to the fuzzy domain. However, this is not the case. The proposed method may be better described as a stochastic and fuzzy hybrid. The (I) initialization and (II) mutation procedures are modified so that they utilize both, the fuzzy and the stochastic theory.

2.5.1 Initialization

Initialization is done in order to seed the population NP , D -dimensional parameter vector of the algorithm. Traditionally performed through using $rand_i \in [0, 1]$, a uniform probabilistic distribution to randomly select within upper (b_U) and lower bounds (b_L) agents is to be carried through subsequent algorithm components:

$$x_{i,0} = b_L + rand_i(b_U - b_L) \quad i = 1, 2, \dots, NP \quad (2.19)$$

Instead, in FDE, initialization is carried out by using two fuzzy concepts; (I) a normal continuous-valued fuzzy set characterized by a membership function and (II) alpha-cuts. Membership functions in this case are used to describe the convex single-point normal fuzzy sets defined on the real line, often termed fuzzy numbers (i.e. vague values such as a flow of *about* $5 \text{ m}^3/\text{s}$) (Ross, 2004). Therefore, the membership functions are used to capture the available knowledge and transfer it to the optimization algorithm. The membership functions and the alpha-cuts are both used to support the initialization step within the optimization algorithm. The use of alpha-cuts allows for the creation of multiple unique population vectors from the singular supplied fuzzy set. Through these fuzzy concepts, the FDE algorithm initialization is able to take advantage of the available domain knowledge, no matter how uncertain.

Membership functions describe the degree of membership or truth in each value corresponding to a parameter. Many shapes of membership functions may be used. In this paper, for illustration and convenience, we are limiting our discussion to the triangular membership function. A fuzzy triangular number $A = (a_1, a_2, a_3)$ can be represented by an ordered triplet or by a triangular membership function

$$A(x) = \begin{cases} 0, & \text{if } x < a_1 \\ \left(\frac{x - a_1}{a_2 - a_1}\right), & \text{if } a_1 \leq x \leq a_2 \\ \left(\frac{a_3 - x}{a_3 - a_2}\right), & \text{if } a_2 \leq x \leq a_3 \\ 0, & \text{if } x > a_3 \end{cases} \quad (2.20)$$

Fig. 2.2 shows a triangular membership function defined by Eq. (2.20) where a_2 holds the highest degree of membership in x (membership, $\mu = 1$) comparatively a_1 and a_3 hold no degree of membership ($\mu = 0$). Within the FDE algorithm a_1 and a_3 are called the initial parameter range while a_2 is called the focus or target parameter.

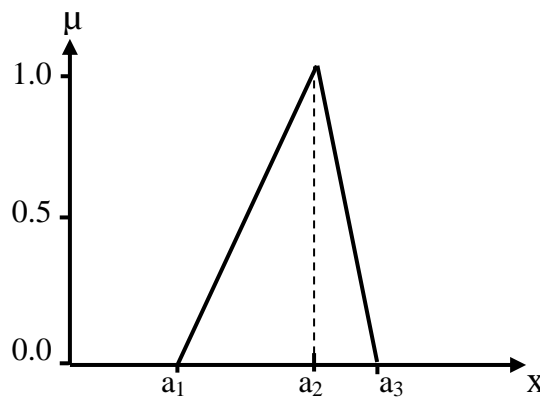


Figure 2.2. Triangular fuzzy membership function.

Alpha-cuts are mostly used to extract information from a membership function and are rarely used for defuzzifying the fuzzy sets (converting fuzzy numbers into crisp form). The alpha-cut describes a fuzzy set using a set of sharp sets. The main idea is to fix a certain membership degree α and thus to obtain a crisp set, which is defined as the set of values that have a membership degree higher or equal to α . Fig. 2.3 illustrates the concept of alpha-cuts. The membership function is cut horizontally at a finite number of regular α -levels, or cuts, between 0 and 1. This process generates a number of crisp interval sets as shown in Fig. 2.4.

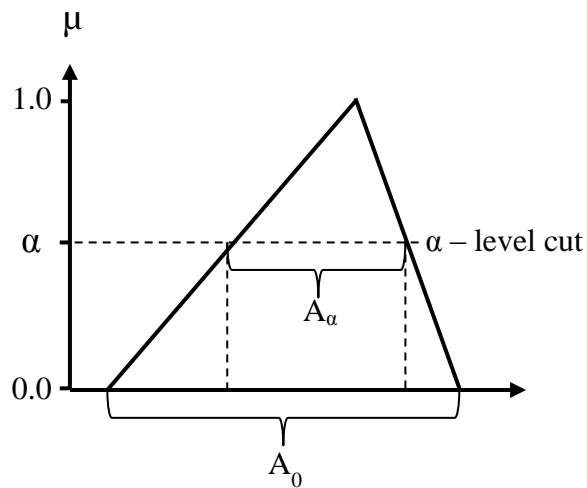


Figure 2.3. The alpha-cut method schematic.

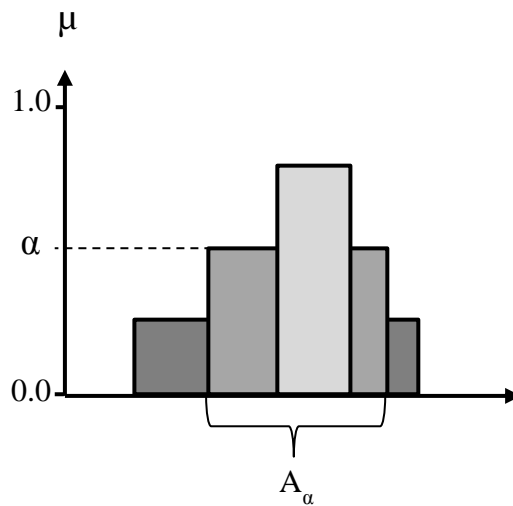


Figure 2.4. The alpha-cut intervals schematic.

Taking an arbitrary alpha-cut $\epsilon \in [0, 1]$ in A (a triangular fuzzy number), a confidence fuzzy interval, A_α is obtained, defined as

$$A_\alpha = [a_1^\alpha, a_3^\alpha] = [((a_2 - a_1)\alpha + a_1), (-(a_3 - a_2)\alpha + a_3)] \quad (2.21)$$

Relating to FDE, parameters are described using triangular fuzzy numbers in the form of inputs for the triangular membership function. To start the algorithm, the initial population vector needs to be generated from these membership functions. This is achieved by using the alpha-cut method NP times at random α -levels to create alpha-cut intervals for each parameter. This allows for a unique individual to be generated NP times from the same parameter membership function input (fuzzy number). The alpha-cut interval is assumed to belong to a unique fuzzy number. In essence, the initial fuzzy number is used to seed NP unique incomplete fuzzy numbers defined only by a single discrete alpha-cut level.

The alpha-cut interval population vector, $A_{i,0}^\alpha$, is found by modifying Eq. (2.21).

$$A_{i,0}^\alpha = (A_{i,0}^{L\alpha}, A_{i,0}^{U\alpha}) \quad (2.22)$$

$$A_{i,0}^{L\alpha} = a_1 + rand_i * (a_2 - a_1)$$

$$A_{i,0}^{U\alpha} = a_3 - rand_i * (a_3 - a_2)$$

Where $i = 1, 2, \dots, NP$ and α is the alpha-cut level such that it is equal to a uniform random number generated, $rand_i \in [0,1]_i$. $A_{i,0}^{L\alpha}$ and $A_{i,0}^{U\alpha}$ are the lower and upper interval bounds for each alpha-cut. The parameters a_1, a_2, a_3 are the values representing the fuzzy number triplet for each individual parameter.

In singular value form, the alpha-cut intervals are converted to the familiar population vector where neutral preference is given to the upper and lower intervals

$$x_{i,0} = \frac{1}{2} * (A_{i,0}^{U\alpha} + A_{i,0}^{L\alpha}) \quad (2.23)$$

In order for a unique singular value to be generated, an asymmetrical triangular membership function must be used.

2.5.2 Mutation

The mutation component of the algorithm allows for new population vectors to be generated in order to investigate the feasible region in search for the optimal solutions. FDE utilizes the alpha-cut intervals from the initialization stage and performs mutation on them by using fuzzy arithmetic. Performing the mutation in the fuzzy domain allows for the algorithm to take advantage of the focused search benefits given by the uncertain or vague available knowledge from the problem domain. The mutation that is carried out is based on a modification of DE/rand/1/bin, a classical, widely used and successful strategy. Therefore the full notation for the proposed strategy can be stated as

FDE/rand/1/bin. A similar modification to the one presented here could be performed for several other DE variants available, but that is beyond the scope of this paper.

DE/rand/1/bin defines the weighted differential of two different randomly chosen vectors and is used to perturb another randomly chosen vector, creating a mutated vector. This process is mathematically expressed in Eq. (2.2).

The mutation vector mathematical expression in Eq. (2.2), transformed using alpha-cut intervals (from initialization and subsequently), has the following form:

$$m_{i,G+1}^{\alpha} = A_{r_1,G}^{\alpha_{r_1}} + F * (A_{r_2,G}^{\alpha_{r_2}} - A_{r_3,G}^{\alpha_{r_3}}) \quad r_1 \neq r_2 \neq r_3 \quad (2.24)$$

Utilizing fuzzy interval arithmetic properties for addition and subtraction (Bojadziev and Bojadziev, 1995),

$$A + B = (a_1^{\alpha}, a_2^{\alpha}) + (b_1^{\alpha}, b_2^{\alpha}) = (a_1^{\alpha} + b_1^{\alpha}, (a_2^{\alpha} + b_2^{\alpha})) \quad (2.25)$$

$$A - B = (a_1^{\alpha}, a_2^{\alpha}) - (b_1^{\alpha}, b_2^{\alpha}) = (a_1^{\alpha} - b_2^{\alpha}, (a_2^{\alpha} - b_1^{\alpha}))$$

and substituting for Eq.(2.24) yields the lower and upper mutation vector interval bounds:

$$m_{i,G+1}^{L,\alpha} = A_{r_1,G}^{\alpha,L} + F * (A_{r_2,G}^{\alpha,L} - A_{r_3,G}^{\alpha,U}) \quad r_1 \neq r_2 \neq r_3 \quad (2.26)$$

$$m_{i,G+1}^{U,\alpha} = A_{r_1,G}^{\alpha,U} + F * (A_{r_2,G}^{\alpha,U} - A_{r_3,G}^{\alpha,L}) \quad r_1 \neq r_2 \neq r_3$$

Where $i = 1, 2, \dots, NP$. The alpha-cut population vector interval $A_{i,G}^\alpha$, is represented by discrete endpoints $(A_{i,G}^{\alpha,L}, A_{i,G}^{\alpha,U})$ for levels $\alpha_{r1}, \alpha_{r2}, \alpha_{r3}$. These levels may equal to each other or they may be different. However, as seen in Eq. (2.25) the alpha-cut level α must be the same throughout in order to proceed with interval arithmetic. This is likely not the case in the initialization stage where unique alpha-cut intervals are generated.

Each of the alpha-cuts for the purpose of the FDE algorithm represents a unique fuzzy number. These fuzzy numbers are incomplete, because they are defined by a single alpha-cut level (Bojadziev and Bojadziev, 1995). In order to perform interval arithmetic at the same alpha-cut level, redefining of incomplete fuzzy numbers is required. Redefining allows incorporating levels not given initially (Bojadziev and Bojadziev, 1995).

The mutated alpha-cut intervals vector can be expressed in the traditional singular value form:

$$m_{i,G+1}^\alpha = \frac{1}{2} * (m_{i,G}^{L\alpha} + m_{i,G}^{U\alpha}) \quad (2.27)$$

2.5.3 Illustrative Example of FDE Algorithm

The same simple numerical example that was used to illustrate the original DE algorithm is presented here to illustrate the FDE algorithm. Let us consider the following objective function for optimization:

$$\text{Minimize } f(x) = x_1 + x_2 + x_3 \quad (2.28)$$

The initial population is chosen by taking NP (defined as 6) random alpha-cuts of a fuzzy membership function for each decision variable; in this case x_1, x_2 and x_3 are defined by the same triangular fuzzy membership function triplet (0, 1, 3). Therefore, the initial parameter range is $\epsilon[0,3]$ while the target or focus is 1.

A sample calculation for initialization of x_1 , "Individual 1" is shown using Eq. (2.22), where the alpha-cut is randomly selected at 0.6.

$$A_{1,0}^{0.6} = (A_{1,0}^{L0.6}, A_{1,0}^{U0.6}) \quad (2.29)$$

$$A_{1,0}^{L0.6} = 0 + 0.6 * (1 - 0) = 0.6$$

$$A_{1,0}^{U0.6} = 3 - 0.6 * (3 - 1) = 1.8$$

The fuzzy interval in Eq. (2.29) is transformed to a singleton using Eq. (2.23).

$$x_{1,0} = \frac{1}{2} * (A_{1,0}^{U0.6} + A_{1,0}^{L0.6}) = \frac{1}{2} * (1.8 + 0.6) = 1.2 \quad (2.30)$$

The population along with its respective objective function values is shown in Table 2.9. The first member of the population “Individual 1” is set as the target vector.

In order to generate the mutated vector, three individuals (“Individual 3”, “Individual 5” and “Individual 6”) from the population size are selected randomly (ignoring “Individual 1”, since it is set as the target vector). The weighted difference between “Individual 3” and “Individual 5” is added to the third randomly chosen vector “Individual 6” to generate the mutated vector. This procedure in FDE is different than in classical DE, in that the weighted difference is done on the alpha-cut fuzzy intervals before conversion into a single value the algorithm can utilize. The weighting factor F is chosen as 0.80, the weighted difference vector is obtained in Table 2.10 and the mutated vector in Table 2.11.

Table 2.9. An illustrative example

Population Size $NP = 6$ (user defined), $D = 3$												
	Individual 1		Individual 2		Individual 3		Individual 4		Individual 5		Individual 6	
	Fuzzy Interval		Fuzzy Interval		Fuzzy Interval		Fuzzy Interval		Fuzzy Interval		Fuzzy Interval	
x_1	1.20	0.60,1.80	1.11	0.77,1.46	1.05	0.90,1.20	1.19	0.61,1.77	1.28	0.45,2.11	1.14	0.72,1.67
x_2	1.11	0.79,1.42	1.26	0.48,2.04	1.14	0.73,1.55	1.49	0.03,2.94	1.18	0.63,1.73	1.06	0.89,1.22
x_3	1.44	0.12,2.76	1.12	0.76,1.48	1.02	0.97,1.07	1.30	0.39,2.22	1.09	0.82,1.37	1.16	0.68,1.63
$f(x)$	3.74		3.49		3.20		3.98		3.55		3.36	

Table 2.10. Calculation of the weighted difference vector for the illustrative example

	Individual 3		Individual 5		Difference Vector	Weighted Difference Vector	
	Fuzzy Interval	Lower	Upper	Lower			Upper
x_1	Lower	0.90			2.11	= -1.21	= -0.97
	Upper		1.20		0.45	= 0.75	= 0.60
x_2	Lower	0.73		-	1.73	= -1.00	$\times F$ = -0.80
	Upper		1.55		0.63	= 0.92	($F = 0.80$) = 0.74
x_3	Lower	0.97			1.37	= -0.40	= -0.32
	Upper		1.07		0.82	= 0.25	= 0.20

Table 2.11. Calculation of the mutated vector for the illustrative example

	Fuzzy Interval	Weighted Difference Vector	+	Individual 6		Mutated Vector
				Lower	Upper	
x_1	Lower	-0.97		0.72		= -0.25
	Upper	0.60			1.67	= 2.27
x_2	Lower	-0.80		0.89		= 0.09
	Upper	0.74			1.22	= 1.96
x_3	Lower	-0.32		0.68		= 0.36
	Upper	0.20			1.63	= 1.83

The mutated vector fuzzy intervals can be expressed in traditional single value form using Eq. (2.27). The mutated vector in single value form is given in Table 2.12.

Table 2.12. Interval to single value mutated vector calculation

	Upper Fuzzy Interval Bound		Lower Fuzzy Interval Bound	Sum		Mutated Vector
x_1	2.27		-0.25	= 2.22		= 1.11
x_2	1.96	+	0.09	= 2.05	×0.5	= 1.03
x_3	1.83		0.36	= 2.19		= 1.10

The mutated vector does a crossover with the target vector to generate the trial vector as shown in Table 2.13. This is carried out by (1) generating random numbers equal to the dimension of the problem (2) for each of the dimensions: if random number $> CR$; copy the value from the target vector, else copy the value from the mutated vector into the trail vector. In this example, the crossover constant CR is chosen as 0.60.

Table 2.13. Generation of the trial vector for the illustrative example

	Target Vector		Mutated Vector	Trail Vector
x_1	1.20		1.11	= 1.11
x_2	1.11	Crossover	1.03	= 1.03
x_3	1.44	($CR = 0.60$)	1.10	= 1.44
$f(x)$	3.74		3.24	3.58

The objective function of the trial vector is compared with that of the target vector and the vector with the lowest value of the two (minimization problem) becomes “Individual 1” for the next generation. To evolve “Individual 2” for the next generation, the second member of the population is set as target vector and the above process is repeated. This process is repeated NP times until the new population set array is filled which completes one generation. Once the termination criterion is met, the algorithm ends.

Table 2.14. New population for next generation for the illustrative example

Population Size $NP = 6$ (user defined), $D = 3$

	Individual 1	Individual 2	Individual 3	Individual 4	Individual 5	Individual 6
	Fuzzy Interval	Fuzzy Interval	Fuzzy Interval	Fuzzy Interval	Fuzzy Interval	Fuzzy Interval
x_1	1.11	-0.25,2.27				
x_2	1.03	0.09,1.96				
x_3	1.44	0.12,2.76				
$f(x)$	3.58					

Chapter 3

3.1 Decision Support System Software Package

The DE algorithm has been implemented in the form of a convenient decision support system (DSS) called the Differential Evolution Optimizer (DEO). The decision support system integrates, alongside the classical algorithm, key differential evolution features discussed in the methodology, such as fuzzy differential evolution and the ability to deal with constraints. DSS is developed to provide a convenient optimization software package with a friendly graphical user interface for the MS Windows operating system. DSS provides easy access and all the practical benefits to an efficient optimization algorithm for less technical individuals. DEO was programmed in C# and the code, as well as the installation files, have been provided electronically with the thesis. Brief overviews of the supplementary files included with this thesis are in Appendix C.

In this chapter, a helpful user guide of DEO is presented to review the key features and the process involved in inputting and reading the results from a defined optimization problem. In addition to the user guide, an illustrative example problem is used as a step-by-step guide of the typical procedure towards finding an optimal solution using the DSS.

3.2 Differential Evolution Optimizer Overview

Once the Differential Evolution Optimizer DSS is run, an execution window like the one shown in Fig. 3.1 should be displayed. Upon starting the DEO decision support system the user is greeted with the “**Algorithm Inputs**” window tab open. As the user fills in the appropriate input fields he/she is able to proceed to the “**Optimization Inputs**” window and finally the “**Optimization Results**” window. These will be reviewed in the subsequent sections.

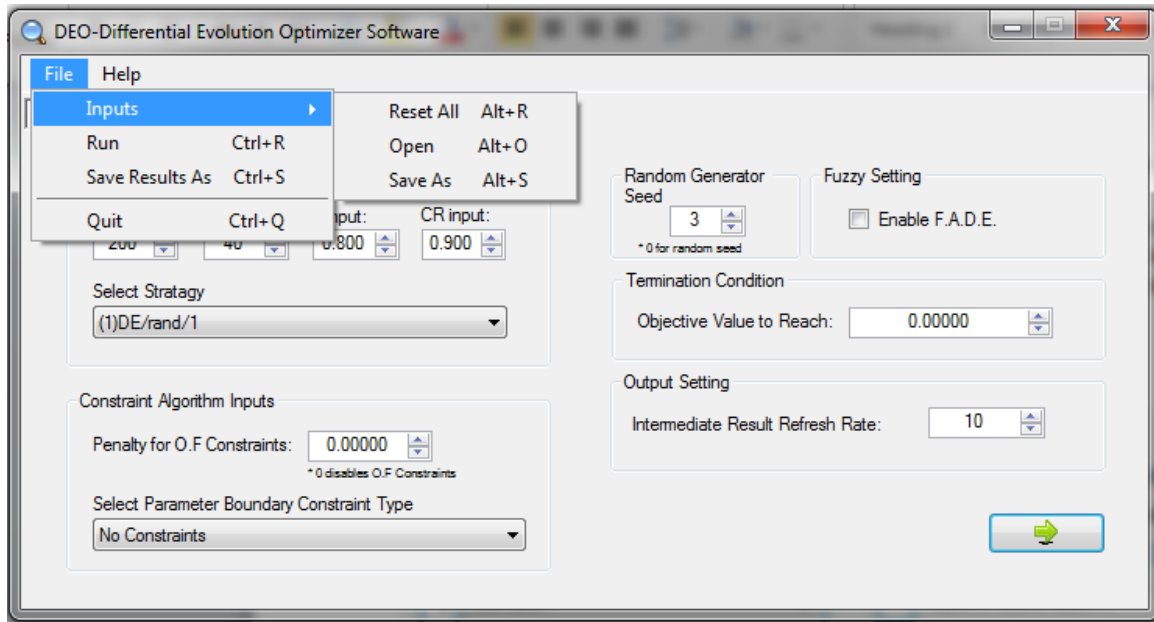


Figure 3.1. Interface of DEO menu.

Fig 3.1. shows the interface of the menu strip in the top left corner of the program window with two options “**File**” and “**Help**” (the documentation you are now reading). Upon clicking “**File**”, the user is presented with the option for “**Inputs**”, to “**Run**” the optimization, “**Save Results**” of the optimization once a problem has been optimized and the option to “**Quit**”, i.e. to close the program.

Selecting “**Inputs**” will further open additional menu options: “**Reset All**” reverts all input parameters to default; “**Open**” automatically fills the input requirements by prompting the user to select past saved (.deo extension) input files; and “**Save As**” saves the current inputs and prompts the user to name the file and the file will be saved with a .deo extension.

Selecting “**Save Results**” will prompt the user to name the file; the file will be saved with a .csv extension and may be accessed later in Microsoft Excel for post processing and review.

3.2.1 Algorithm Inputs

The main body of the algorithm inputs window contains multiple interface inputs pertaining to setting up the differential evolution algorithm. Fig. 3.2 shows the algorithm inputs window. The inputs are labeled numerically for reference within this section.

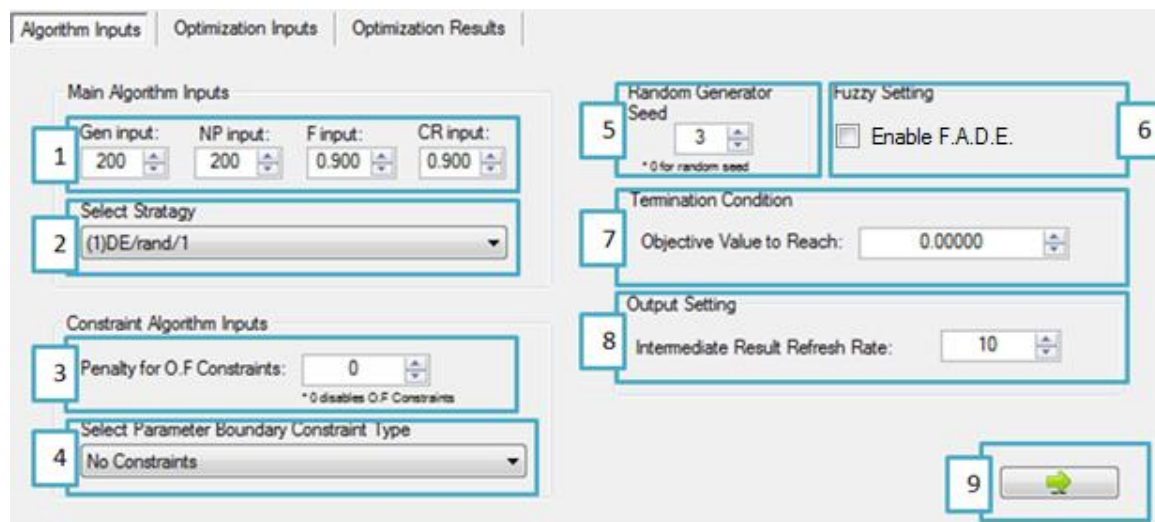


Figure 3.2. Algorithm inputs window.

Each number in Figure 3.2 corresponds to a detailed explanation given below.

1. Included under the main algorithm inputs heading are four user defined control parameters for the differential evolution algorithm. They are detailed below.
 - **Generation input** is the number of iterations (generations) the algorithm will go through to find the optimal solution before termination. The more generations given, the greater the accuracy of the final result may be to the true optimal solution at the expense of more computation time.
 - **NP input** is the number of parents which, as a guideline, may be selected to be 10 times the number of parameters of the objective function. Increasing the number of parents increases the search space, thereby

speeding up convergence. Empirical evidence suggests that increasing NP above 40 does not significantly influence the convergence rate.

- **F input**, the weighing factor F [0, 2] controls the amplification of differential variation; to begin with, a value of 0.8 is suggested.
- **CR input** The crossover weight CR [0,1] probabilistically controls the amount of recombination; initially a value of 0.9 is suggested.

These parameters are of significance for the accuracy and convergence time required. Therefore, a proper selection is very important. Adequate selection of each of the parameters may differ from problem to problem and may require some trial and error in selection. The user can choose to enter the values directly within the textbox or increment the number by clicking either the up or down arrow beside the textbox.

2. Differential evolution has a specialized nomenclature that describes the **selected strategy** for optimization. The nomenclature and the methodology for the variants included within DEO were discussed in detail in Chapter 2. DEO has 4 available strategies that are accessed through a dropdown menu:

- i. **DE/rand/1**: The classical version of DE.
- ii. **DE/best/1**: Tailored for small population sizes and fast convergence. Dimensionality should not be too high.
- iii. **DE/local-to-best/1**: A version which has been used by numerous scientists. Attempts a balance between robustness and fast convergence.
- iv. **FDE/rand/1**: The classical version of DE transformed into a novel fuzzy differential evolution strategy. The parameter initialization is in the form of fuzzy triangular membership functions utilizing alpha cuts to carry out the mutation and crossover on subsequent generations. This strategy mimics the performance of classical DE/rand/1 with the addition of knowledge for inputs supplied by the decision maker.

3. This DSS uses the **penalty function method** (discussed in Chapter 2) in order to deal with constraints on the feasible space for the objective function. The penalty function method is comprised of the optimization of the objective function with the addition of the constraint violation function (the sum of the violation of all constraint functions). The main challenge of the penalty function method lies in the difficulty of selecting an appropriate value for the penalty coefficient that adjusts the strength of the penalty. The user is required to provide the penalty function coefficient to be used for all the objective function constraints; if the user does not wish to use any constraints on the objective function, a penalty coefficient of zero should be used. When dealing with a minimization problem, the penalty coefficient must be a positive value; conversely, when it is a maximization problem, the coefficient must be a negative value. This assures that any constraint violations will indeed penalize the optimization solution and not make it better.

The user can choose to enter the values directly within the textbox or increment the number by clicking either the up or down arrow beside the textbox.

4. One of two boundary (**random reinitialization** or **bounce-back**) search space constraint methods (discussed in Chapter 2) can be selected for the algorithm from the dropdown menu. Random reinitialization occurs if any trial parameter exceeds a bound placed on a parameter. The out of bounds parameters values are reset into allowed values by randomly choosing a value from within the allowed range. Because it radically changes a parameter's value, reinitialization can disrupt the progress towards solutions that lie near the bounds. Random reinitialization, similar to the bounce-back method, replaces a vector that exceeded one or more of its bounds by a valid vector that satisfies all boundary constraints. In contrast to random reinitialization, the bounce-back strategy takes the progress toward the optimum into account. The user may also choose **no boundary constraints** to be used. In such a case there is no guarantee that the optimal solution will be within the search space bounds.

5. The user is given the option of selecting the **seed value** used by the random generator. This makes it possible to achieve the same results through multiple optimization runs, given that the same seed is used. Furthermore, an easier comparison between different control inputs can be achieved. If the user wishes for the seed to be random, a value of zero should be placed in the input field. The user can choose to enter the values directly within the textbox or increment the number by clicking either the up or down arrow beside the textbox.
6. The fuzzy settings allow the user to enable the **FADE settings** by clicking the checkbox. FADE stands for fuzzy adaptive differential evolution, as discussed in detail in Chapter 2. FADE optimizes the control parameters CR and F for each generation to increase accuracy and convergence speed by referencing a database corresponding to fuzzy rules based on empirical findings. As a result, the user does not need to spend time selecting appropriate values for CR and F , as these values are only used for the initialization of FADE.
7. The termination condition input, **VTR**, is the value that will terminate the algorithm upon achieving. This feature is particularly useful for benchmark functions where the optimal objective function solution is known.
8. The output setting allows the user to select how the intermediate results should be displayed. For example if the input here is 10, the intermediate result outputs will be displayed every 10th iteration (generation).
9. After all the inputs are complete and assured to be accurate, the user should click the **next arrow button** on the interface to proceed to the optimization inputs window.

3.2.2 Optimization Inputs

After clicking the next button on the **algorithm inputs** window the **optimization inputs** tab will open. Here, multiple interface inputs allow for the objective function to be defined—the boundary range for each parameter (used for initialization) and objective constraints (if any)—before finally proceeding with the optimization. Fig. 3.3 shows the optimization inputs window. Inputs are labeled numerically for reference within this section. At any point the user may choose to hit the green arrow to go back to the previous algorithm inputs window.

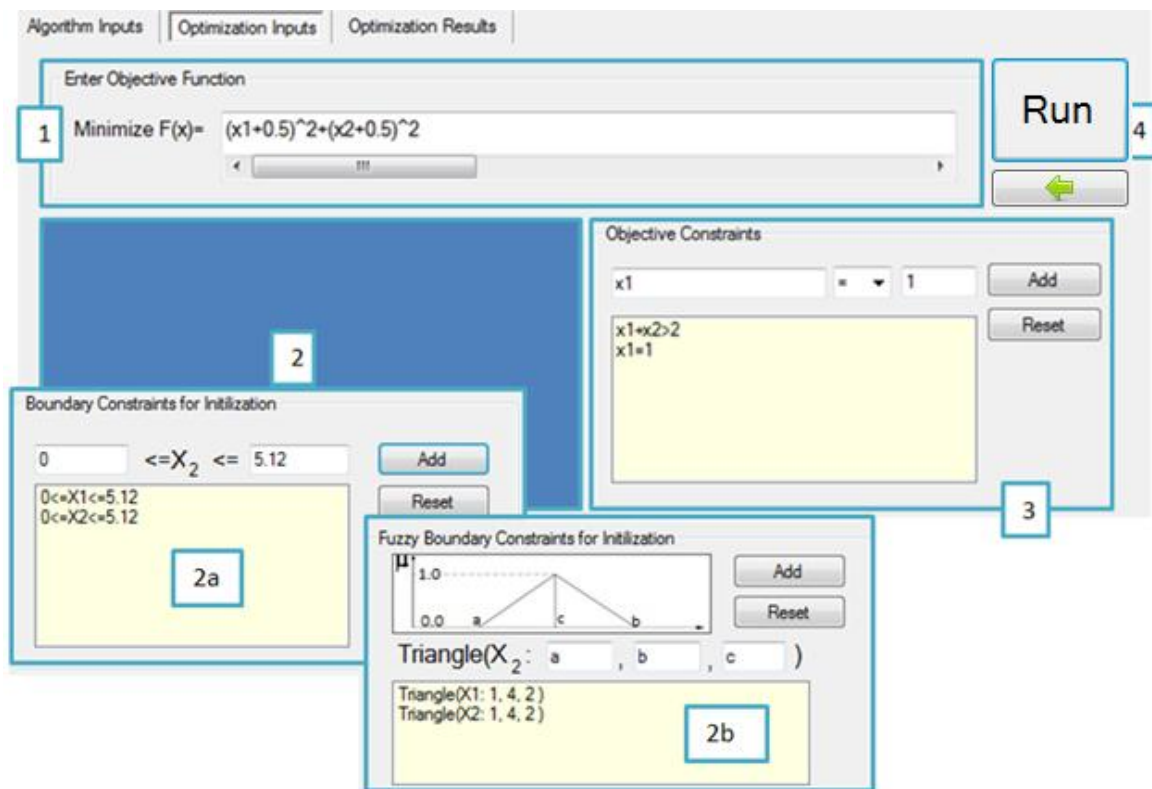


Figure 3.3. Optimization inputs window.

Each number in Figure 3.3 corresponds to a detailed explanation given below.

1. The user is required to provide an **objective function** in the textbox alongside “Minimize F(x) =”. By default, DEO deals with the minimization of the objective function; for maximization problems a simple transformation is needed (i.e. multiply the whole function by negative one, -1). The input textbox accepts up to

30 parameters and they must be defined as $x1$, $x2$ and so on. The operations and prebuilt functions which are recognizable by DEO are listed in Table 3.1.

Table 3.1. List of available functions

	Symbol	Description
Operator	+ - * /	Four arithmetic operations
	^	Power function
Functions	sqrt()	Square root function
	pi	π (3.14159...)
	abs()	Absolute function
	sin()	Sine function
	cos()	Cosine function
	tan()	Tangent function

2. Once the objective function is provided, the user is required to define the search space used for initialization of the differential evolution algorithm.
 - a. If the user selected a traditional DE strategy, the user will be presented with this interface and will be required to give upper and lower bounds for each parameter defined in the objective function.
 - b. If the FDE strategy is selected, then the user will be presented with this interface and will need to define the triangular membership function for the boundary constraint of each parameter in the objective function.

Once each parameter is defined, the user needs to click the Add button, this process is repeated until all have been defined. If at any point a mistake is made, the reset button can be clicked which will restart the search space definition process.

3. In this interface the user may input the constraints (if any) on the objective function itself, using the penalty constraint method. The leftmost textbox allows for the user to write the appropriate constraint equation, whereas the middle

dropdown box enables the user to choose between inequalities to be used for the constraint. Available inequalities to choose between are: less than [$<$], greater than [$>$] or equal [$=$]. The right textbox accepts only crisp numerical values corresponding to the right-hand size of the constraints. Once each constraint is defined, the user needs to click the Add button, this process is repeated until all have been defined. If at any point a mistake is made the reset button can be clicked which will erase all the constraints.

4. Finally, once all the inputs have been provided, clicking the **Run** button will initiate the optimization. When Run is selected, the program may take some time to complete the optimization, depending on the complexity of the problem. Once the optimization is complete, the user will be presented with the optimization results window.

3.2.3 Optimization Results

Fig.3.4 details the optimization results window; the outputs are labeled numerically for detailed explanation below.

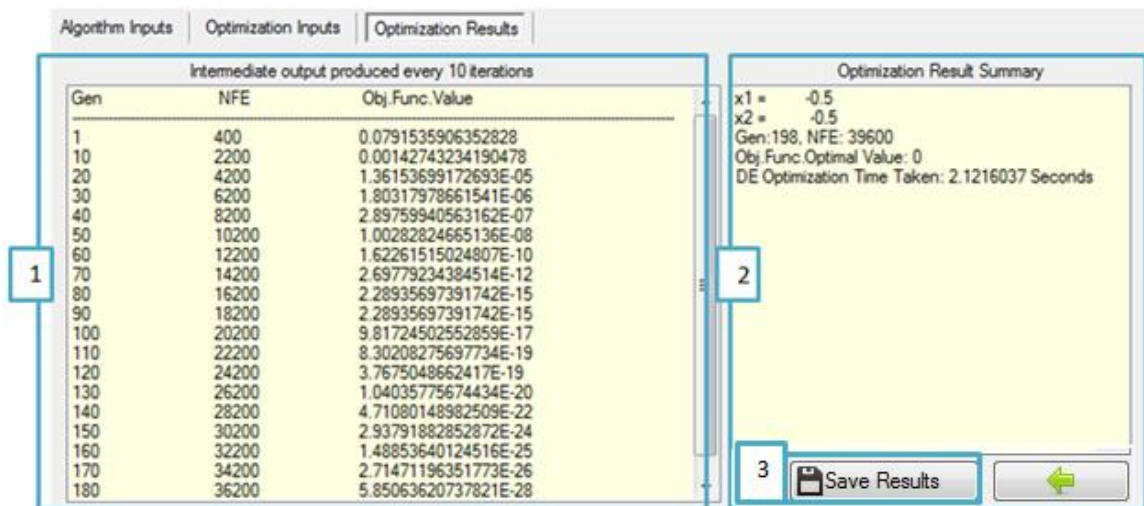


Figure 3.4. Optimization results window.

1. The **intermediate output** is produced at the user defined interval, showing (from left to right) the current generation, the NFE (number of function evaluations) and the corresponding objective function value.
2. The **optimization results are summarized** here. These include the optimal parameter values, the optimal objective value and the generation and number of objective function evaluations it took to achieve the optimal results. The computational time needed for optimization is also given in seconds.
3. The user may choose to save these results by clicking this button. Alternatively results may be saved through the menu strip as previously mentioned. The saved file includes all the outputs seen in the interface, in addition to intermediate parameter values to go along with the intermediate objective function values.

3.3 Illustrative Example

The goal of the example below is to familiarize the user with the basic functionality of the decision support system by means of a numerical example.

Consider a minimization problem where D , the number of parameters, is equal to 5 and the objective function is given in Eq. (3.1). The additional inputs to be used are bounds of $[-5.24, 5.24]$ for each x_i , 400 iterations, NP of 100, F of 0.8, CR of 0.9. The remaining inputs should be application defaults. In addition, it is reasonable to believe that the optimal result of each parameter (i.e. x_i) lies at about -0.5.

Given the above information, use the *FDE/rand/1* strategy and confirm that the true solution occurs at a global optimum of 0 for each parameter being equal to -0.5.

$$f_1(x) = \sum_{i=1}^D (x_i + 0.5)^2 \quad (3.1)$$

Solution:

First, the algorithm inputs were entered from the givens, as shown in Figure 3.5. A value of 3 was chosen for the random generator seed and no feasible space or search space constraints were activated. The objective function value to reach was chosen as 0 and the intermediate results were chosen to be at increments of 10.

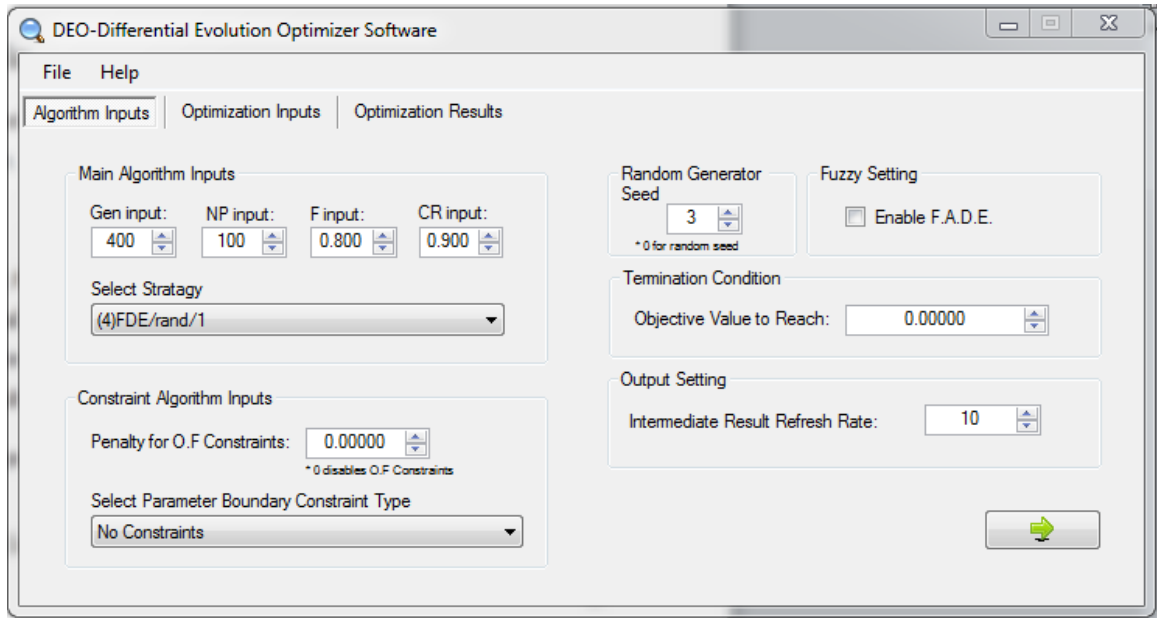


Figure 3.5. Algorithm inputs for illustrative example.

After clicking the next arrow in Figure 3.5, the optimization inputs window is opened. The objective function is written in addition to the fuzzy triangle membership function definition for initialization, as shown in Figure 3.6.

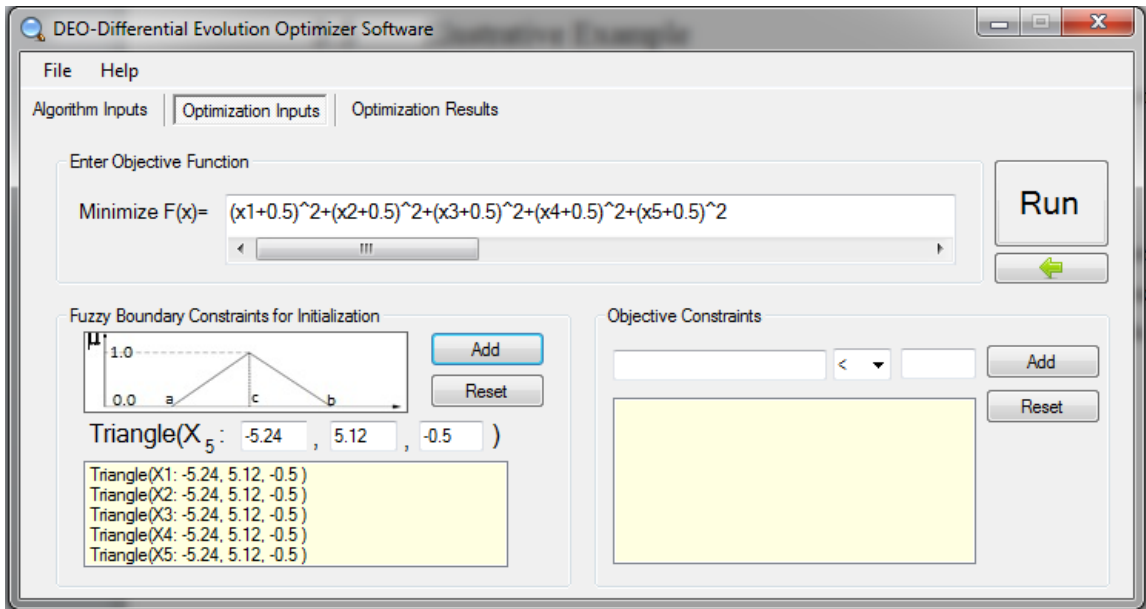


Figure 3.6. Optimization inputs for illustrative example.

Finally, the run button is clicked in the optimization inputs window and subsequently the optimization results are presented, as seen in Figure 3.7.

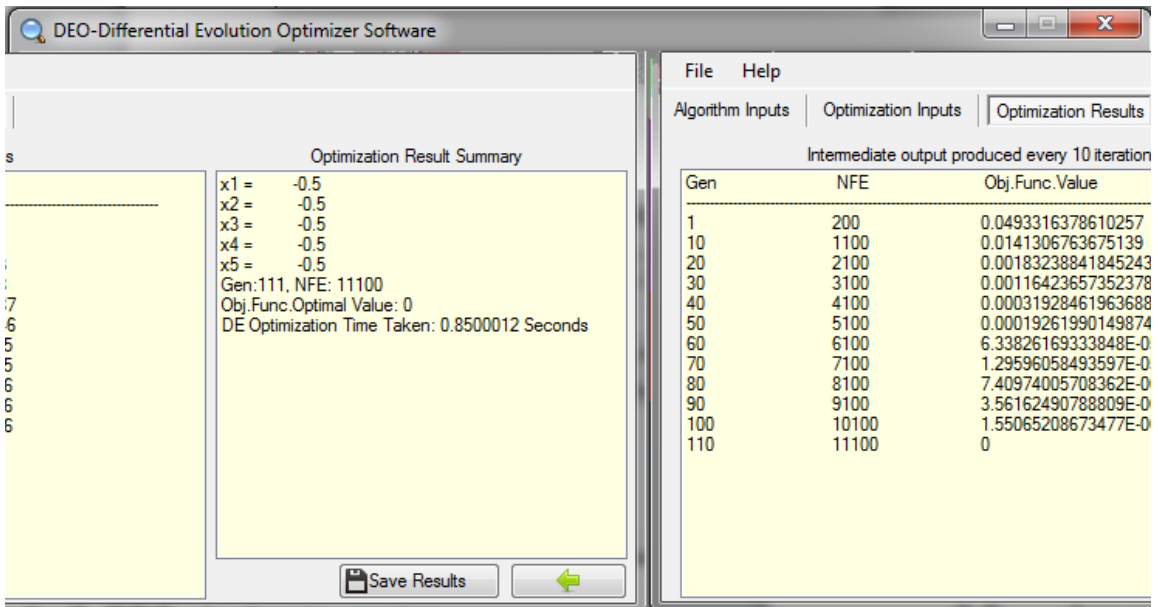


Figure 3.7. Optimization results for illustrative example.

From the results in Figure 3.7 it can be confirmed that the global optimal solution is 0 for $x \in -0.5$ which was found after only 111 generations. In addition we can conveniently observe the convergence progress in the intermediate output interface.

Chapter 4

4.1 Application

This chapter on application of the novel fuzzy differential evolution algorithm explores two topics. One is on the theoretical application using benchmark functions and the other on practical application using a water resource management case study. The objective of benchmark function applications is to evaluate the performance of FDE compared to the classic DE strategy. On the other hand, the objective of the case study is to demonstrate a real-world example of how FDE can better utilize knowledge previously disregarded in other DE strategies due to its “fuzzy” characteristic to achieve more efficient optimization.

4.2 Benchmark Functions

Benchmark functions, or test functions, are commonly used in order to test optimization procedures (Molga and Smutnicki, 2005). The quality of the proposed FDE algorithm is evaluated by comparing it with the original DE algorithm variant - DE/rand/1/bin (simply referred to as DE from this point on) by utilizing well-known benchmark functions from the literature.

The function testbed contains four functions: (i) first De Jong, (ii) Rosenbrock’s Valley, (iii) modified third De Jong, and (iv) Rastrigin’s function (Black, 1996; Molga and Smutnicki, 2005). These functions exhibit distinctive difficulties for a global optimization algorithm. For all functions, an initial parameter range, IPR, and focus value were defined. At the beginning of the optimization, initial parameter values are drawn using traditional methodology or FDE initialization.

IPR for FDE and DE is kept consistent for all functions $x \in [-5.12, 5.12]$, while for the case of DE with smaller bounds the IPR is changed to $x \in [-1, 1]$.

The algorithm settings for each test function are given in Table 4.1; the FDE strategy is compared to DE; user-given controls are kept consistent for a fair comparison.

Table 4.1. Algorithm settings

Method, Parameters	Settings for Benchmark Tests	
Strategy	DE/rand/1/bin	FDE/rand/1/bin
Test Problems	Min f(X)	Min f(X)
Generations	2000	2000
Mutation Factor	0.8	0.8
Crossover Factor	0.9	0.9
Number of Individuals	10×D	10×D
Random Generator Seed	3	3

First De Jong Function

De Jong is one of the pioneers in evolutionary computation. De Jong's function was originally introduced to evaluate genetic algorithms and subsequently has been well accepted by the evolutionary optimization community. The First or Sphere De Jong function is one of simplest problems for optimization algorithms because it does not contain local optima and provides a smooth gradient towards a global optimal solution:

$$f_1(x) = \sum_{i=1}^D x_i^2 \quad (4.1)$$

The global minimum is $f_1(0) = 0$. The graph of the function can be seen in Fig. 4.1.

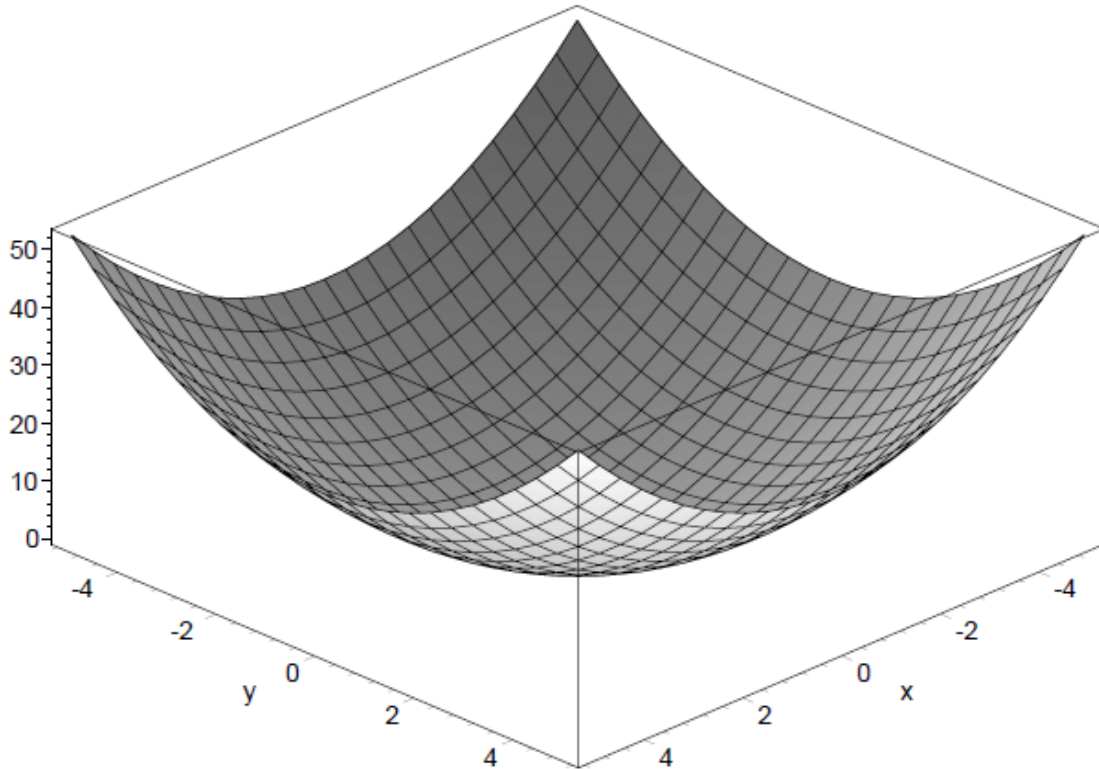


Figure 4.1. First De Jong's function in 2 dimensions (Molga and Smutnicki, 2005).

Rosenbrock's Valley Function

Rosenbrock's function is a classical optimization problem used as a performance test for optimization algorithms. The function may be referred to as the second function of De Jong, or Banana function due to its shape as shown in Fig. 4.2.

$$f_2(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \quad (4.2)$$

Although $f_2(x)$ has just two parameters, it has the reputation of being a difficult minimization problem. The global minimum is $f_2(1)=0$.

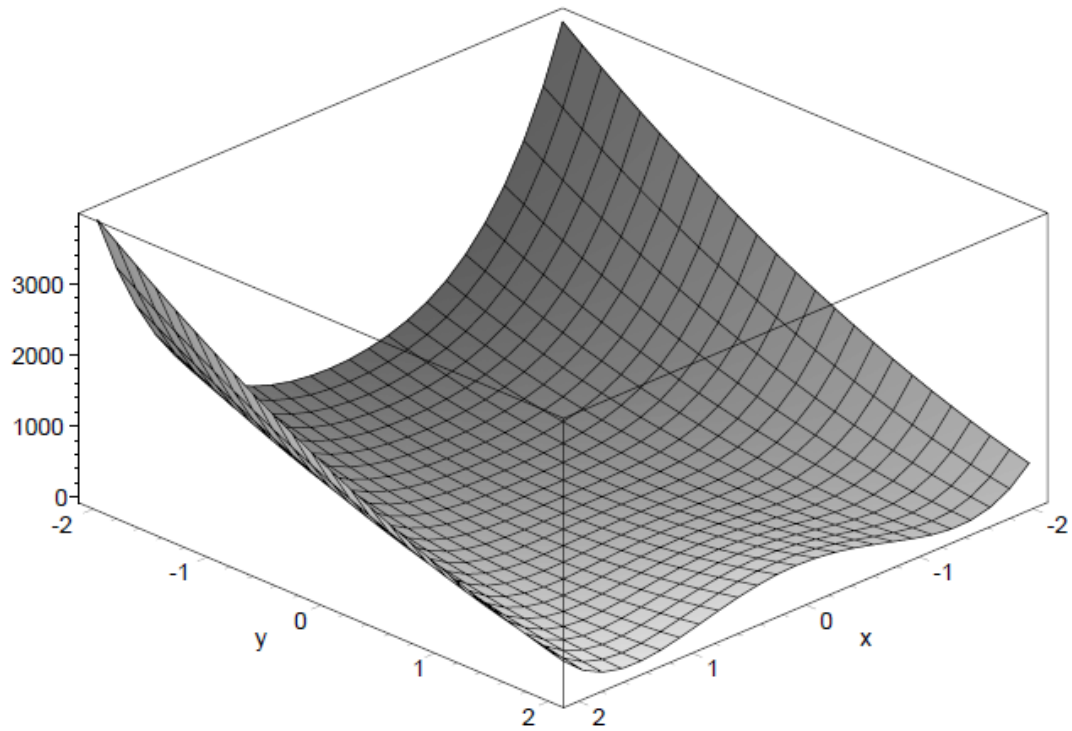


Figure 4.2. Rosenbrock's function in 2 dimensions (Molga and Smutnicki, 2005).

Modified Third De Jong Function (step)

The step function introduces small plateaus to the topology of an underlying continuous function (Back, 1996). Instead of the original linear step function proposed by De Jong, shown in Fig.4.3 is the discretization of a sphere model.

$$f_3(x) = \sum_{i=1}^D (x_i + 0.5)^2 \quad (4.3)$$

The modified step function in Eq.4.3 exhibits many plateaus which pose a considerable problem for many optimization algorithms as they do not contribute any information on the favorable search direction. The global minimum is $f_3(-0.5) = 0$.

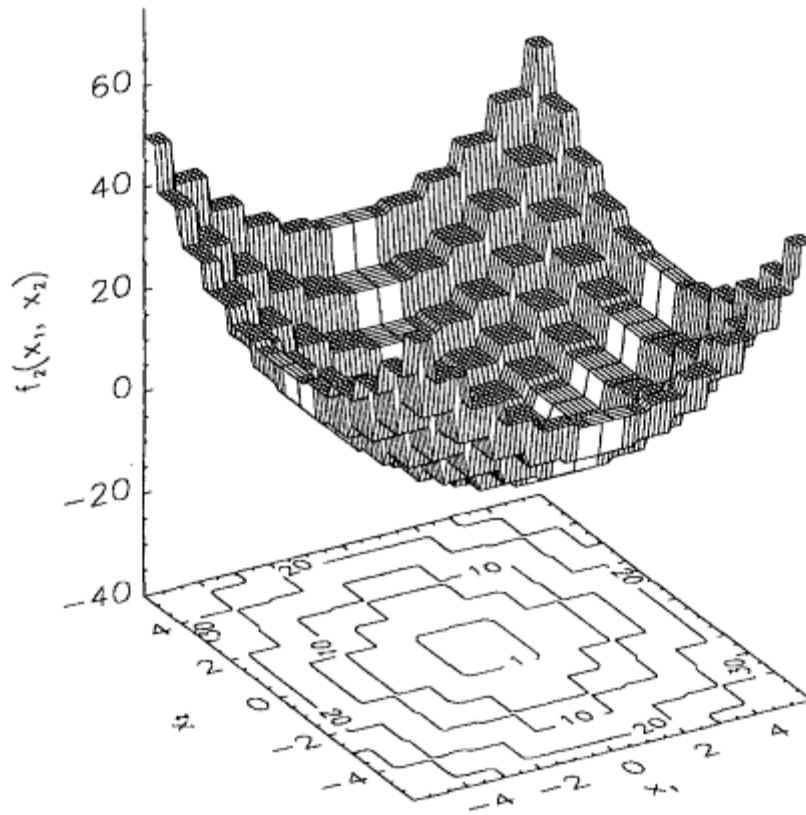


Figure 4.3. Modified Third De Jong Function in 2 dimensions (Black, 1996).

Rastrigin's Function

Rastrigin's function, as shown in Fig. 4.4, is a highly multimodal test function. This function is fairly difficult to optimize due to its large search space and its large number of local minima produced by the cosine modulation. For those reasons, it is frequently selected for testing the performance of various optimization algorithms:

$$f_4(x) = 10D + \sum_{i=1}^D [(x_i^2 - 10 \cos(2\pi x_i))] \quad (4.4)$$

The global minimum is $f_4(0)=0$.

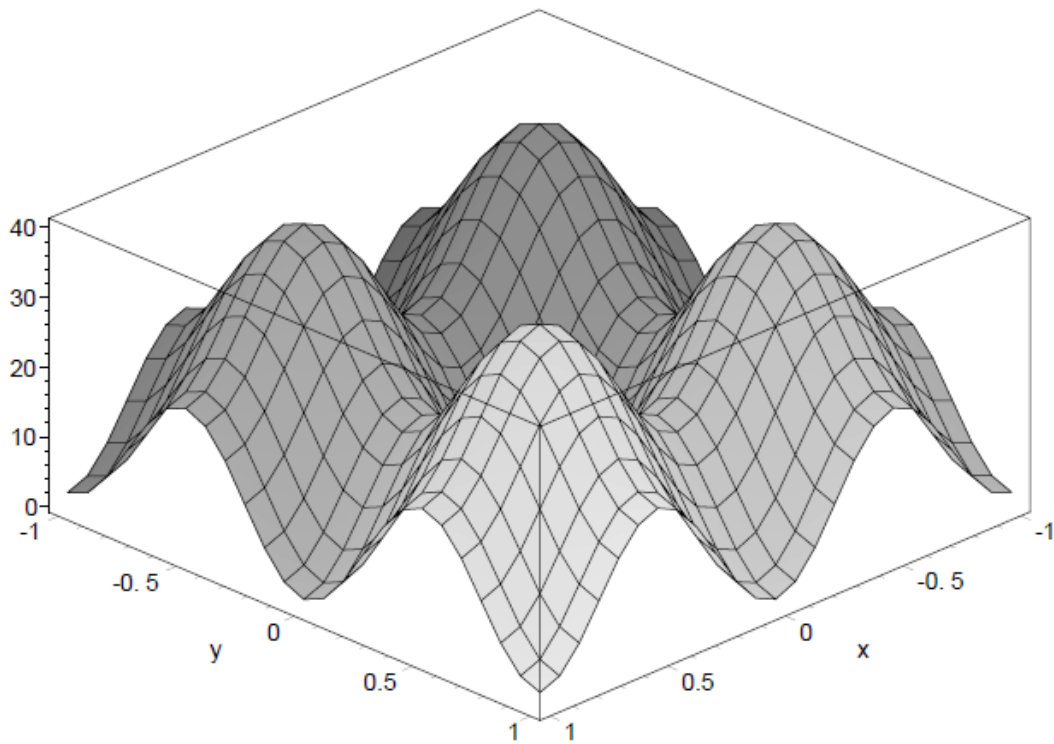


Figure 4.4. Rastrigin's function in 2 dimensions (Molga and Smutnicki, 2005).

4.2.1 Benchmark Function Results and Discussions

There are several conclusions reached after the comparison of FDE to the original DE strategy using the benchmark functions. First, a comparison of DE and FDE is made by selecting three arbitrary focusing targets. The focus targets are selected as 1, 3 and 5 for the sphere and Rastrigin function while for the step and Rosenbrock function they are selected as -1, 1 and 3. The results are shown in Table 4.2. In Table 4.2 the values column lists the dimensions of the problem, D and objective function optimal solution. Where $f(X^*)$ is the known exact solution while f_{DE} and $f_{FDE}^{(focus)}$ are the optimal solutions found through the use of the DE and FDE algorithm respectively. The results indicate that FDE performs better than classic DE in terms of convergence speed, independently of the selected target initialization value. This can be seen especially in the first 400 to 500 generations. This is attributed mostly to the more focused initialization strategy of FDE. Furthermore, the results in Table 4.2 show that the quality of optimal solution improves based on the proximity of the initial focus target value to the true solution. For example, the convergence speed incrementally improves for the First De Jong function (see Table 4.2) as the subjective focus value approaches the true optimal solution of zero. The magnitude of the optimal solution differences between the varying targets does not directly correlate with the magnitude of the target differences themselves.

Table 4.2. Performance comparison of FDE and DE algorithms at various focus targets

Functions	Comparison of DE and FDE	
	Values	Curves of best solutions
<p><i>First De</i></p> <p><i>Jong</i></p> <p><i>(Sphere)</i></p>	<p>$f(X^*):$</p> <p>$f(0) = 0$</p> <p>$D = 20$</p> <p>$f_{DE} = 5.686e - 2$</p> <p>$f_{FDE}^1 = 0$</p> <p>$f_{FDE}^3 = 0$</p> <p>$f_{FDE}^5 = 0$</p>	
<p><i>Rosenbrock</i></p>	<p>$f(X^*):$</p> <p>$f(0) = 0$</p> <p>$D = 10$</p> <p>$f_{DE} = 2.099e - 8$</p> <p>$f_{FDE}^{-1} = 4.311$</p> <p>$f_{FDE}^1 = 3.638$</p> <p>$f_{FDE}^3 = 6.86$</p>	

Modified
Third De
Jong(Step)

$$f(X^*):$$

$$f(X^*) = 0$$

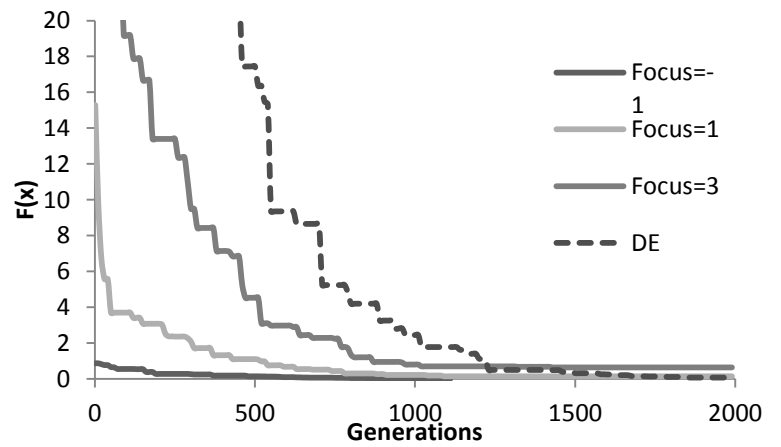
$$D = 20$$

$$f_{DE} = 6.141e - 2$$

$$f_{FDE}^{-1} = 0$$

$$f_{FDE}^1 = 1.445e - 1$$

$$f_{FDE}^3 = 6.406e - 1$$



Rastrigin

$$f(X^*):$$

$$f(0) = 0$$

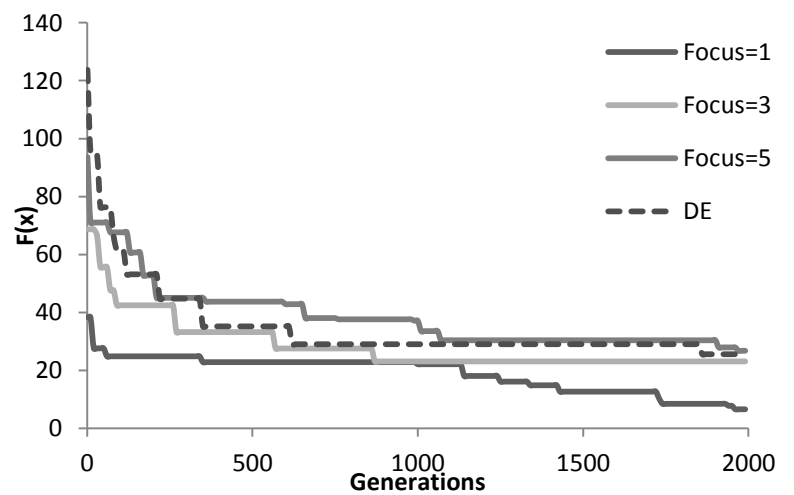
$$D = 10$$

$$f_{DE} = 25.557$$

$$f_{FDE}^1 = 6.543$$

$$f_{FDE}^3 = 23.074$$

$$f_{FDE}^5 = 26.745$$

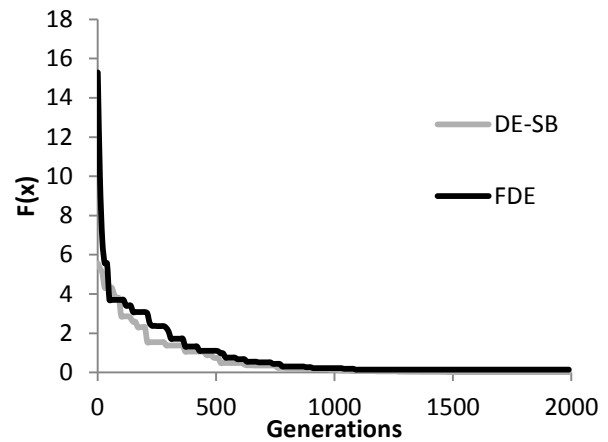


In Table 4.3, the benchmark function optimization comparison is made between DE with drastically smaller bounds (DE-SB) and FDE with focusing target of 1. The values column in Table 4.3 lists the initial (f_{DE}^i, f_{FDE}^i) and final (f_{DE}^f, f_{FDE}^f) objective function values through using the DE and FDE algorithm respectively. Decreasing the initialization bounds in DE-SB and keeping FDE bounds wider shows that FDE performs similarly to or better than traditional strategies, without limiting the search space by imposing more certainty than is available. Additionally, the outcomes shown in Table 4.3 indicate that the improved results using FDE over DE are not just attributed to the better initialization values (due to the more focused smaller initial parameter range) but are affected by the novel mutation strategy as well.

Table 4.3. Performance comparison between the original DE algorithm with smaller bounds and FDE with a focus equal to one

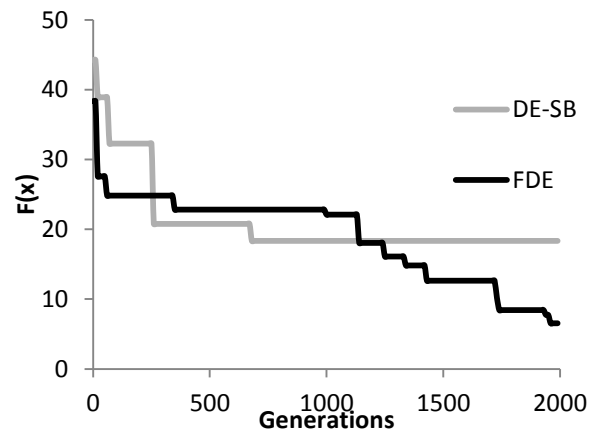
Function	Comparison of FDE and DE-SB	
	Values	Curves of best solution
First De Jong (Sphere)	$f_{FDE}^i = 3.838$ $f_{DE-SB}^i = 3.484$ $f_{FDE}^F = 0$ $f_{DE-SB}^F = 2.169e - 3$	
Rosenbrock	$f_{FDE}^i = 50.169$ $f_{DE-SB}^i = 82.599$ $f_{FDE}^F = 3.638$ $f_{DE-SB}^F = 4.07e - 10$	

Modified $f_{FDE}^i = 15.292$
 Third De $f_{DE-SB}^i = 5.562$
 Jong (Step) $f_{FDE}^F = 1.445e - 1$
 $f_{DE-SB}^F = 4.001e - 3$



Rastrigen

$f_{FDE}^i = 38.358$
 $f_{DE-SB}^i = 44.253$
 $f_{FDE}^F = 6.543$
 $f_{DE-SB}^F = 18.345$



The Rosenbrock function in particular appears to perform worse using the FDE algorithm than the traditional DE algorithm. It can be seen in Table 4.2 that the Rosenbrock function using the FDE algorithm appears to stall due to misconvergence while the traditional DE algorithm continues to converge towards the optimal solution. This may be a result of the particular control parameters selected (CR and F) not being adequate for the FDE algorithm when it comes to this particular function, or it could be that the algorithm itself does not cater as well as DE to such a function. In Table 4.3 at first glance it appears that the performance of the Rosenbrock function using the FDE algorithm is again worse, this may be attributed to the DE algorithm giving a significant

head start advantage due to the smaller initialization bounds used (based on the initial objective values). Thus the performance of FDE for the Rosenbrock function after considering the aforementioned is comparable to DE with smaller initialization bounds as shown in Table 4.3.

Therefore, with some functions FDE may not perform better than the original DE scheme. This is due to misconvergence or stalling of the algorithm based on the objective function itself and the control parameters selected. FDE shares this robustness problem with many other DE scheme variants. Therefore, care needs to be taken when selecting FDE alongside the control parameters for an objective function to ensure that it is the correct choice in achieving the best convergence efficiency. Currently, as with most other variants, validation of selection may only be confirmed through trial and error procedure. Future research may be directed into sensitivity analysis of FDE to a multitude of benchmark functions, with the purpose of determining the general set of best handled function types. However, the potential reduction in application capacity does not lessen the undeniable value of the FDE algorithm in being included in the optimization toolbox.

4.3 Case Study

The reservoir operation case study presented in this section demonstrates the practical application of the novel fuzzy differential evolution algorithm for optimization in the field of water resource management.

4.3.1 Study Area Background

This study is focused on the optimization of the operation of the Wildwood reservoir in the Upper Thames River basin. The basin is located in the Great Lakes Region, between Lake Erie and Lake Huron in Southwestern Ontario, Canada (see Fig. 4.5). The watershed encompasses an area of 3,482 square kilometers, with a total population of 485,000 (UTRCA, 1993). Most of the basin area is rural except for the larger urban centers of London, Stratford and Woodstock.

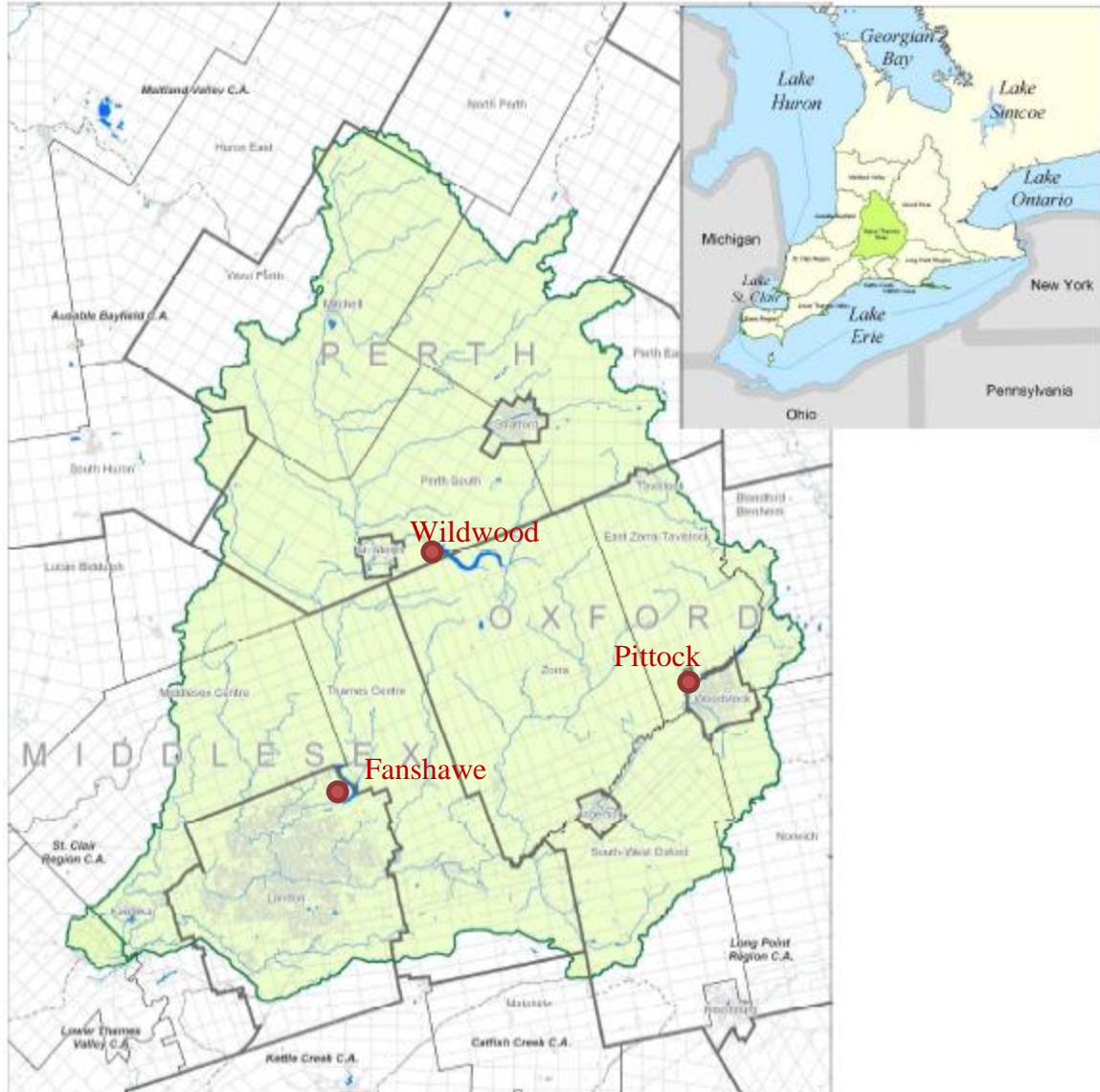


Figure 4.5. Location of the Upper Thames basin.

Seasonal flooding has historically been a major hazard for the Upper Thames River basin. Typically, flooding occurs in early March during snowmelt and in the summer seasons as a result of extreme rainfall events (UTRCA, 1993). In 1937, the city of London experienced a massive flooding event. As a result, this sparked the creation of the Upper Thames River Conservation Authority (UTRCA) in 1947. Since the creation of the UTRCA three major water management reservoirs were created: Pittock, Wildwood and Fanshawe (see Fig. 4.5).

Among the aforementioned reservoirs Wildwood was the first major project commissioned by UTRCA in 1948 and finally constructed in 1965. The Wildwood reservoir is located on Trout Creek, upstream of the Town of St. Mary's. The reservoir is designed to control downstream flooding and to increase summer stream flows. The reservoirs also provide a range of recreational opportunities for thousands of people each year. The primary goals of the reservoir include flood control during the snowmelt period and summer storm season, low flow augmentation during the drier summer months from May to October and recreational uses during the summer season. Among these goals, the most important one is flood control. Floods in the basin result from a combination of snowmelt and intense precipitation during December to April. In addition to the primary goals of the reservoir, it is also used for recreational purposes, hydro power generation and by local fisheries.

Wildwood is operated by the Upper Thames Conservation Authority in a coordinated manner with reservoirs at Fanshawe (London) and at Pittock (Woodstock) (UTRCA, 2012). This optimizes flood control and low flow augmentation efforts for the North Thames River in St. Mary's and for the Thames River watershed in general. Operating the reservoir involves control of one or more of the three outflow structures. The outflow components include: four large sluice gates, three small vales and concrete baffle walls. The sluice gates are used to provide coarse control of flows from the dam during peak runoff periods. This may include the spring runoff period (March-April) and during the fall and early winter when the soil may be frozen or saturated and thus susceptible to runoff. Otherwise, the valves provide fine control of outflow during the summer and periods of low flows. The valves are located in the core of the dam. As such, they allow for maintenance and discharge of cooler water from the bottom of the reservoir. Concrete baffle walls above the gates provide some automatic control during the early summer months when the reservoir level is at or close to its highest level. Water can spill over the walls when the reservoir rises following summer storms.

4.3.2 Problem Definition

A release strategy for the optimal operation of the Wildwood reservoir is required for the year 2010. The year 2010 in this study represents the future so that the available historical 2010 inflow data can be used for problem formulation. The operation of the reservoir must be optimized in order to ensure that the reservoir meets the primary requirements of flood control and low flow augmentation. In addition to the primary goals, the reservoir must be operated keeping in mind constraints put forth by the fisheries industry and recreational reservoir use. A simplified schematic of the reservoir is given in Fig. 4.6, showing the allocation of storage (maximum reservoir capacity, C ; active storage, S_t ; minimum storage allowable, S_{\min}) and reservoir flows (inflow, i_t ; release, R_t); the notations are consistent with the mathematical formulation.

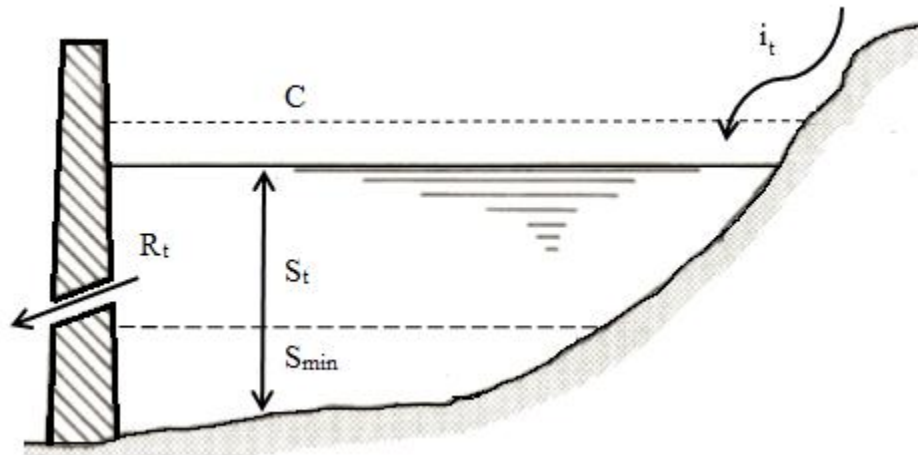


Figure 4.6. Wildwood reservoir schematic.

4.3.3 Mathematical Formulation

Optimization can be defined as a process searching for an optimal solution that provides a maximum or minimum value of an objective function (Rao, 1996). Therefore, formulation of the objective function is the most important step in solving an optimization problem.

The objective function is formulated as shown in Eq. (4.5) based on primary flood control operation goals and based on some additional constraint descriptions in Table 4.4.

$$\text{Minimize } f = \frac{(\sum_{t=1}^3 S_t^{min} + \sum_{t=5}^{12} S_t^{min})}{\sum_{t=4} S_t^{max}} \quad (4.5)$$

The above is a minimization optimization objective concerning reservoir storage S_t and a $t = 12$ month time horizon. Where $t = 1$ corresponds to January and $t = 12$ to December. It can be seen that the objective function, though globally a minimization problem, has a dual objective for both minimization and maximization. The months requiring minimization of storage (S^{min}) are for the purpose of flood control and furthermore preventing damage as a result of flood inundation to upstream properties. The maximization of storage (S^{max}) is required by fisheries and hydro power, based on the description given in Table 4.4. This occurs for the month of April or $t = 4$.

Table 4.4. Constraints of the Wildwood reservoir (UTRCA, 1993)

Categories	Constraint Description
Physical Constraints	Reservoir maximum capacity $18,470 \times 10^3 \text{ m}^3$ and minimum capacity $2,430 \times 10^3 \text{ m}^3$
Flood Control	The release from reservoir should not exceed $10 \text{ m}^3/\text{s}$ to avoid significant flooding. Release should be less than $3 \text{ m}^3/\text{s}$ to avoid nuisance flooding at St. Mary's golf course.
Low flow augmentation	In the months of May to October the release from reservoir should target at least $1.13 \text{ m}^3/\text{s}$
Recreation	Wide fluctuations should be avoided

	particularly in the summer time.
Fisheries	Peak storage should be achieved by the first week of April and then subsequently reduced during spring. The reservoir storage level should remain stable at summer levels until late fall.
Hydro Power	Peak storage should be achieved by the first week of April.

In order to perform the optimization of the proposed objective function, additional equations are required to properly model the Wildwood reservoir system. These equations and their variables are simplifications of the complex real-world system and as such can only approximate the true behavior. The model is defined in the form of constraints of which the continuity constraint is the most important one in that it ensures that the reservoir system is balanced with inflow and release, properly accounting for changes in reservoir storage.

Continuity constraint:

$$S_{t-1} + i_t - R_t = S_t \quad t = 1, 2, \dots, 12 \quad (4.6)$$

where R_t is the release at the current time step, i_t is the inflow at the current time step, similarly S_t represents the storage at the current time step, while S_{t-1} is the storage in the previous time step. Therefore, in order to utilize the above equation for a 12 month time horizon, the initial reservoir storage S_0 must be given.

In addition to the continuity constraint, there are release and storage constraints that are governed by the physical capacities of the reservoir given in Table 4.4.

Subject to release constraints:

$$0 \leq R_t \leq R_{max} \quad t = 1, 2, 3, 4, 11, 12 \quad (4.7)$$

In addition to the reservoir physical release capacity constraints, there is a minimum release constraint for low flow augmentation in the summer months, as detailed in Table 4.4.

(4.8)

$$R_{augmented} \leq R_t \leq R_{max} \quad t = 5, 6, \dots, 10$$

Where R_{max} is the maximum physical capacity for the outflow structure (sluice gates, etc.) and $R_{augmented}$ is the minimum target release for low flow augmentation.

Subject to storage constraints:

$$S_0 \leq S_{12} \quad (4.9)$$

This storage constraint is to ensure that the released storage does not exceed the initially available one.

Storage capacity constraint:

$$S_{min} \leq S_t \leq C \quad t = 1, 2, \dots, 12 \quad (4.10)$$

where S_{min} is the physical minimum capacity of the reservoir (for structural and mechanical integrity of the dam components) and C is the maximum physical capacity of the reservoir beyond which significant flooding will occur.

The final constraint that appropriately models the reservoir is intended to ensure that the fisheries industry has a stable reservoir level for fishing from late summer to late fall. In other words, the August storage levels (S_8) are maintained.

Fisheries stability constraint:

$$S_8 = S_t \quad t = 9, 10, 11 \quad (4.11)$$

4.3.4 Algorithm and Optimization Inputs

Having formulated the Wildwood reservoir optimization problem, the fuzzy differential evolution algorithm inputs must be assigned. Given in Table 4.5 are the control parameter inputs for the FDE algorithm itself. These values were subjectively chosen using trial and error, as they produce best results for the problem formulation. In addition to FDE, the classical DE/rand/1/bin strategy is also used with the same inputs for comparison.

Table 4.5. DE algorithm inputs

Number of Generations	1500
Mutation Factor, F	0.8
Crossover Factor, CR	0.9
Random Seed	5
Penalty Constant	0.0001
Strategy	DE/rand/1 & FDE/rand/1

There are 24 decision variables in the mathematical formulation, divided evenly between variables for release and storage. In order for the optimization algorithm to proceed, these decision variables/parameters must be initialized. In order for initialization to take place, the parameter range and target (focus) values must be established. This may be done by utilizing a decision maker's inherent knowledge to establish the parameter bounds. In this case the knowledge was extracted from historical data provided by UTRCA for the period of 1985–2011. The parameter range, or the upper and lower bounds for each parameter, were determined by analyzing the monthly historical data and selecting the maximum and minimum values within the data set. Thus, the feasible range for release and storage is established without the need for subjective decision maker inputs. In practice, however, the process is not so easy for the selection of the target or focus for each parameter. The goal of our optimization problem is, in essence, to find future operation optimal release and subsequent storage strategies. To do this, we therefore must establish a subjective target for the release and storage that is believed to be an adequate representation of where the optimum would be. To establish such a target for each parameter, subjective (and likely vague) decision maker knowledge is required.

Typically, forecasting information from several sources is used to operate the Wildwood Dam. Computer models of floods, operating tables, weather data and water level information from above and below the dam enable staff to assess and respond to flood potential. In practice, combining these existing methods for operation could establish the subjective target values required for initialization of the optimization algorithm. In this case study we had available historical data of storage and release; based on these values we could choose an appropriate target. Conveniently, since we already had operational data for the year 2010, we could use these values as the basis for our targets.

Table 4.6 and Table 4.7 show the storage and release initialization inputs including parameter range and target values for the year 2010. However, the target values do not initially satisfy some of the reservoir constraints. Consequently, a calibration on the target values for release and storage was performed. This adjustment assured that the optimization started in the feasible space. Table 4.8 shows the constraint-satisfying target

values for release and storage initialization. When using the classical DE algorithm, the same initialization parameter range was used as for FDE.

Table 4.6. Storage initialization inputs for the year 2010 [10^3 m^3]

Month	1	2	3	4	5	6	7	8	9	10	11	12
Lower Bound	2417	2930	5278	12856	13939	13426	12561	11038	8764	4986	2790	3081
Upper Bound	9626	10399	15618	17685	18354	18300	17499	16434	14463	13420	10836	9492
Target	6908	6610	10090	15359	17516	17860	17194	15523	11660	8449	4222	4039

Table 4.7. Release initialization inputs for the year 2010 [10^3 m^3]

Month	1	2	3	4	5	6	7	8	9	10	11	12
Lower Bound	2605	10147	1426	1743	1714	1607	2000	2426	2766	2807	2677	1423
Upper Bound	16364	1987	12961	12514	11204	8328	14530	8057	8615	13491	19336	16382
Target	4345	3463	2387	1763	1966	4596	3180	4719	5786	8511	6153	4940

Table 4.8. Constraint satisfying release and storage target initialization inputs for the year 2010 [10^3 m^3]

Month	1	2	3	4	5	6	7	8	9	10	11	12
Storage Target	7000	6000	10000	15000	17000	17000	17000	8000	8000	8000	8000	7000
Release Target	4000	3500	2500	1500	4000	5000	4000	5000	6000	8000	5000	5000

In addition to the initialization inputs given, feasible space constraints and inflow inputs were required. The release constraints given in Table 4.4 are converted to corresponding monthly equivalent values for convenience in Table 4.9. The monthly inflow data for the Wildwood reservoir was provided by UTRCA and is given in Table 4.10.

Table 4.9. Release constraints [10^3 m^3]

Month	1	2	3	4	5	6	7	8	9	10	11	12
Max	26784	24192	26784	25920	26784	25920	26784	26784	25920	26784	25920	26784
Min	0	0	0	0	3027	2929	3027	3027	2929	3027	0	0

Storage constraints corresponding to physical reservoir capacity and minimum storage:

$$S_{min} = 2,430 \times 10^3 \text{ m}^3$$

$$C = 18,470 \times 10^3 \text{ m}^3$$

Initial Storage (Storage amount in last month of previous year, 2009), $S_0 = 6,564 \times 10^3 \text{ m}^3$

Table 4.10. Monthly inflows for the Wildwood reservoir [10^3 m^3]

Month	1	2	3	4	5	6	7	8	9	10	11	12
Inflow	4187	2656	9419	4901	3350	4433	2421	1413	2617	4819	4341	4859

4.3.5 Study Results and Discussions

The optimization results of combining the mathematical formulation with the algorithm and optimization inputs are presented in this section. Three optimization trials were performed; one using the classic DE strategy and two trials using the novel FDE strategy. The parameter ranges for initialization were kept constant throughout all the trials. The

two FDE trials were used to analyze results from variation in initialization inputs. The notation of FDE_1 was used when the initialization target or focus was outside of the feasible space and FDE_2 was used when the target was within the feasible space.

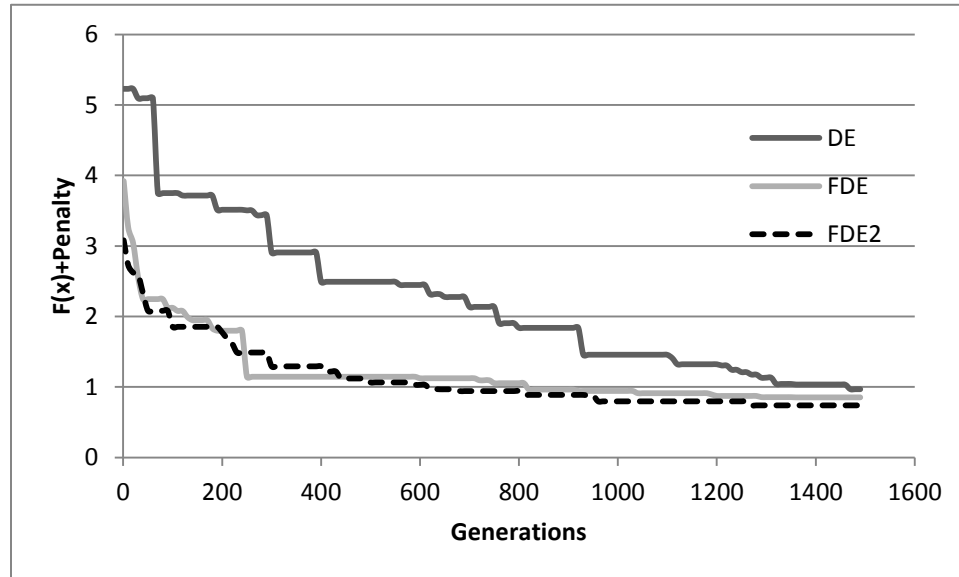


Figure 4.7. Wildwood reservoir optimization progress.

Figure 4.7 shows the convergence speed of the objective function combined with the sum of all the penalty functions for each of the trials. As expected, FDE performed much better than traditional DE. Furthermore, it can be seen that FDE_2 outperformed FDE_1 . The difference in performance between the two FDE trials depended primarily on the subjective inputs of the additional information provided by the decision maker. If the decision maker provides target inputs that do not satisfy the constraints from the outset, the algorithm will focus on a search space outside of the feasible region and may thus fail to converge as quickly as possible, as is the case with FDE_1 . If the subjective values provided for initialization satisfy the constraints initially, as shown in the case of FDE_2 , then the optimization will result with a more optimal solution.

Table 4.11. Wildwood reservoir objective functions and error after optimization

	Optimization Result		
	DE	FDE₁	FDE₂
Error	4880	3084	3024
Objective Function	0.4797	0.5425	0.4361

The exact values of the objective function separated from the penalty function are given in Table 4.11. In this Table, error is the sum of all the constraint violations; based on these results, it can be established that constraint violations were prevalent. This demonstrates the general difficulty with using the penalty function method for constraint handling in a complex problem, because the optimal solution may be one that does not satisfy all the constraints as is clearly the case here. This problem may be addressed through a very detailed sensitivity analysis of various penalty constants for each constraint, a very time consuming trial-and-error process. Here, however, for the sake of convenience, just one penalty constant was used for all the constraints, resulting in more relaxed but still adequate constraint representation.

The penalty constant selected had to produce a penalty function of similar magnitude as the objective function. With too small a penalty constant, significant constraint violations would not be detected by the algorithm, as they would be overshadowed by the objective function values. However, if the penalty constant were too large, the objective function information provided to the algorithm would be overshadowed and the search would not be adequate. Table 4.12 illustrates the importance of selecting the appropriate penalty constant value so that the fitness function conveys the objective function and constraint violation information to the algorithm.

Table 4.12. Penalty constant selection

Penalty Constant	Σ Constraint Violations (Error in Table 1)	Penalty Function Value	Objective Function Value	Fitness Function
0.0001		0.488		0.968
1	$\times 4880$	$= 4880$	$+ 0.480$	$= 4880.48$
0.000001		0.00488		0.48488

It can be concluded from the results in Table 4.11 that FDE₂ had the best objective function solution while still maintaining the least amount of constraint violation when compared to the other two trials. DE did have a better objective function solution than FDE₁; however, when considering the amount of constraint violations, the performance of DE is easily eclipsed by the one of FDE₁.

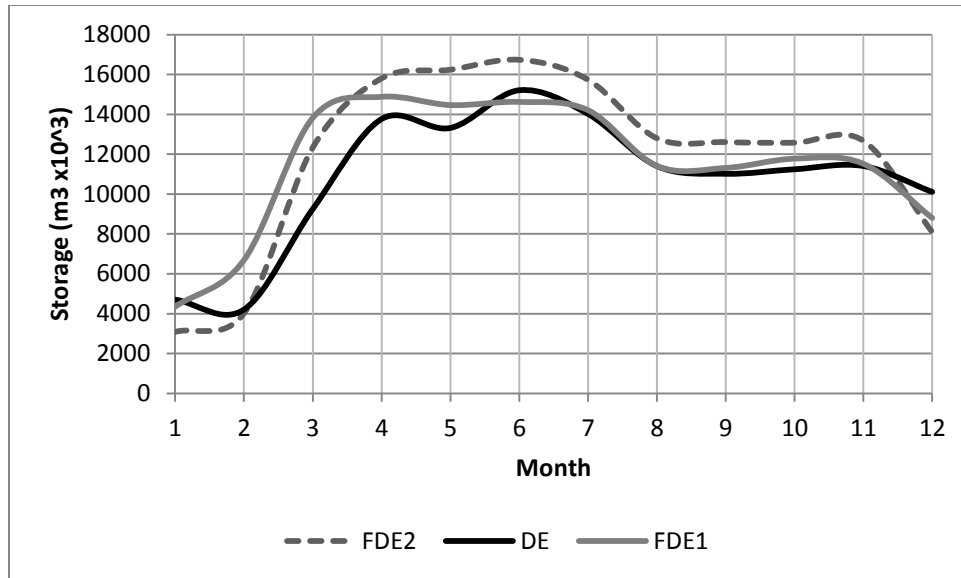


Figure 4.8. Wildwood reservoir storage for a twelve-month time horizon.

The Wildwood optimal reservoir storage and release policy for the year 2010 is shown in Figures 4.8 and 4.9, respectively. Exact optimal values for each decision variable can be found in Table 6.1 of Appendix D. The results follow the problem formulation closely. The storage for the month of April is indeed maximized, while the late summer to fall storage is indeed kept consistent. Similarly, the release policy meets the minimum release requirement for low flow augmentation. Thus, the optimization can be deemed satisfactory.

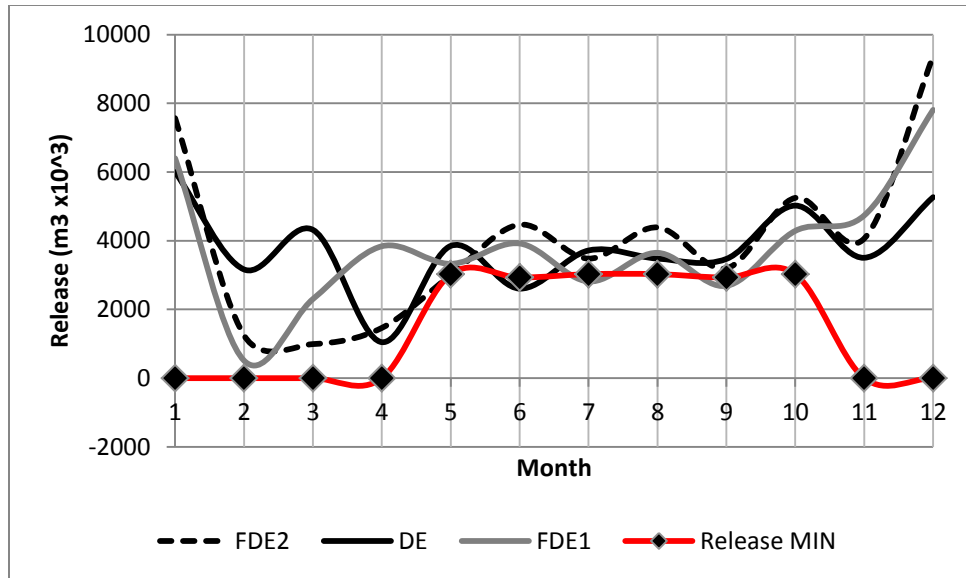


Figure 4.9. Wildwood reservoir release for a twelve-month time horizon.

Chapter 5

5.1 Summary

This thesis proposes a novel method, fuzzy differential evolution algorithm, which utilizes fuzzy triangular membership functions for initialization, combined with random alpha-cuts to create alpha-cut intervals to be perturbed through mutation by fuzzy interval arithmetic. This approach, through the utilization of fuzzy theory concepts, takes advantage of all the available domain knowledge. The FDE algorithm has flexibility in being used for a wide range of linear and non-linear optimization problems. The novel algorithm with fuzzy set theory elements allows the decision makers to provide supplementary knowledge needed to define a more focused search space and, consequently, a more efficient optimization.

A decision support system, named the “Differential Evolution Optimizer” (DEO), was created to assemble the fundamental tools for optimization using the differential evolution algorithm including FDE in a convenient Windows interface. A detailed review of the decision support system has been discussed in Chapter 3. All the optimization results in this paper have been obtained through the use of DEO.

As concluded from the experimental results obtained using the benchmark functions, the addition of the decision maker’s supplied domain knowledge guided the algorithm in a superior way, resulting in faster convergence towards an optimal solution when compared with the traditional DE scheme. This was the main benefit of FDE. Alternatively, the decision maker can reduce the initialization bounds in the traditional algorithm in an attempt to mimic the focusing achieved by FDE. This method incorrectly implies certainty that the solution is indeed within such bounds, whereas the FDE strategy allows for the benefit of focusing on a certain region, while still searching a wider search space to account for uncertainty.

In addition to the main benefits, the benchmark functions results show that even when compared with decreased initial parameter bounds of DE, FDE was still able to

outperform DE, or perform comparatively to it. While the benchmark function results show instances where the initial search bounds were equal, FDE appeared to outperform DE regardless of the focusing target in the search space. It is clear that focusing was of major importance in the search for an optimal solution. However, even in circumstances where the focusing target is highly inaccurate, the algorithm still performed better than DE.

Emphasis is placed on the fact that the FDE algorithms, like all evolutionary algorithms, make no guarantee that an optimal solution is ever found. Furthermore, misconvergence may result using FDE in certain instances. Therefore, FDE may not be better than DE in the absolute sense, but it does provide an alternative to be used where more domain knowledge is available to provide a more efficient convergence. The use of FDE provides more freedom in expressing available domain knowledge without incorrectly claiming full certainty or uncertainty because of the limitations of the algorithm itself.

The FDE algorithm was shown through the Wildwood reservoir case study to be applicable in the water resource management field. The addition of subjective targets for initialization with FDE led to a focused search, ultimately resulting in FDE outperforming the traditional DE algorithm in the convergence towards the optimal solution. The case study also demonstrated the use of constraints within the DE and FDE algorithm and the associated challenge with setting appropriate penalty constants.

5.2 Recommendations for Future Work

The FDE methodology discussed in this paper demonstrates fundamental principles for initialization and mutation within a hybrid fuzzy and probabilistic domain. The methodological background however can be modified and expanded. Future work is recommended to explore additional membership functions for the initialization of FDE. Instead of the triangular membership function, other membership functions should be investigated, such as the trapezoidal and Gaussian membership functions. These membership functions may prove to be more suited for the solution of other types of

problems. That is to say, different membership functions may be better representative of the vague parameter knowledge.

An additional recommendation for further research is to investigate how the alpha-cut intervals are transformed into singular values. This transformation is essential in order to evaluate the objective function or rather the fitness functions, which is indispensable for the differential evolution algorithm to proceed. Currently, this is done by taking the centroid value of the alpha-cut, this being the preference-neutral way (i.e. given to neither extreme).

Another suggestion is to allow the selection of the singular value from the interval, based on a decision maker supplied preference. This would add another control parameter the decision makers can interact with and establish their preference favoring parameters, greater or smaller than the alpha-cut interval.

Lastly, the FDE mutation strategy presented in this paper shows the modification of DE/rand/1/bin or the classic DE strategy. Similarly, other common DE strategies, such as DE/best/1/bin may be modified to work with alpha-cut interval arithmetic. The application of other such strategies incorporating the fundamental FDE concepts could prove to have similar benefits as the ones shown in this paper, in addition to the benefits incorporated by using the new strategy.

References

- Arunachalam, V. (2008). Optimization Using Differential Evolution. Water Resources Research Report No. 060. London, Canada, Facility for Intelligent Decision Support: 42.
- Bazaraa, S. M., D. S. Hanif, et al. (2006). Nonlinear Programming: Theory and Algorithms. New Jersey, John Wiley & Sons, Inc.
- Back, T. (1996). Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. New York, Oxford University Press.
- Bojadziev, G. and M. Bojadziev (1995). Fuzzy Sets, Fuzzy Logic, Applications. Singapore, World Scientific.
- Brest, J., B. Greiner, et al. (2006). "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems." IEEE Trans. Evolut. Comput. 10: 646-657.
- Chakraborty, U. K., Ed. (2008). Advances in Differential Evolution. Studies in Computational Intelligence. Berlin, Springer.
- Courant, R. (1943). "Variational methods for the solution of problems of equilibrium and vibrations." Bulletin of the American Mathematical Society 49: 1-23.
- Dantzig, G. B. (1963). Linear Programming and Extension. Princeton, NJ, Princeton University Press.
- Fogel, D. B. (2006). Evolutionary computation: toward a new philosophy of machine intelligence, John Wiley and Sons.
- Fogel, L. J., A. J. Owens, et al. (1966). Artificial Intelligence Through Simulated Evolution. Chichester, John Wiley.

- Friendman, R., C. Ansell, et al. (1984). "The Use of Models for Water Resources Management, Planning and Policy." *Water Resources Research* 20(7): 793-802.
- Gamperle, R., S. Muller, et al., Eds. (2002). *A Parameter Study for Differential Evolution. Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, WSEAS Press.
- Gen, M. and R. Chen (1997). *Genetic Algorithms and Engineering Design*. New York, John Wiley & Sons, Inc.
- Hall, W. A. and J. A. Dracup (1970). *Water Resources Systems Engineering*. New York, McGraw-Hill.
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI, University of Michigan Press.
- Iba, H. and N. Noman (2012). *New Frontier in Evolutionary Algorithms Theory and Applications*. London, Imperial College Press.
- Ilonen, J., J. K. Kamarainen, et al. (2003). "Differential evolution training algorithm for feed-forward neural networks." *Natural Processing Letters* 17: 95-105.
- Jiménez, F., J. M. Cadenas, et al. (2003). "Solving fuzzy optimization problems by evolutionary algorithms." *Information Sciences* 152: 303-311.
- Joshi, R. and A. C. Sanderson (1999). "Minimal representation multi sensor fusion using differential evolution." *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 29: 63-76.
- Kirkpatrick, S., C. D. Gelatt, et al. (1983). "Optimization by Simulated Annealing." *Science* 220: 671-680.
- Kisi, O. (2004). "Daily suspended sediment modeling using a fuzzy-differential evolution approach." *Hydrol. Sci. J* 49(1): 183-197.

- Liu, J. and J. Lampinen (2002). On setting the control parameters of the differential evolution method. Mendel 8th International Conference on Soft Computing, Brno, Czech Republic, pp. 11-18.
- Liu, J. and J. Lampinen (2004). "A fuzzy adaptive differential evolution algorithm." *Soft Comput. Fusion Found. Methodol.* 9(6): 448-462.
- Lockwood, C. and T. Moore (1993). "Harvesting scheduling with spatial constraints: a simulated annealing approach." *Canadian Journal of Forest Research* 23: 468-478.
- Loucks, D. P. and J. R. da Costa (1991). *Decision Support Systems: Water Resources Planning*, Springer.
- Loucks, D. P., J. R. Stedinger, et al. (1981). *Water resource systems planning and analysis*, Prentice-Hall.
- Mamdani, E.H. and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, Vol. 7, No. 1, pp. 1-13, 1975.
- Molga, M. and C. Smutnicki (2005) "Test functions for optimization needs." Available at <http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf>.
- Onwubolu, G. C. (2008). "Design of hybrid differential evolution and group method of data handling networks for modeling and prediction." *Information Sciences* 178: 3616-3634.
- Pan, Q. K., L. Wang, et al. (2009). "A novel differential evolution algorithm for bi criteria no-wait flow shop scheduling problems." *Computers and Operations Research* 36: 2498–2511.
- Price, K. (1996). *Differential Evolution: A Fast and Simple Numerical Optimizer*. NAFIPS'96. Berkeley: 524-527.

- Price, K., R. Storn, et al. (2005). *Differential evolution: A practical approach to global optimization*. New York, Springer.
- Qin, A. K. and P. N. Suganthan (2005). Self-adaptive differential evolution algorithm for numerical optimization. 2005 IEEE Congress Evolutionary Computation, September 2-5, Edinburgh, UK, pp. 1785-1791.
- Rao, S. S. (1996). *Engineering Optimization: Theory and Practise*. New York, John Wiley and Sons.
- Rogalsky, T., R. W. Derksen, et al. (1999). Differential evolution in aerodynamic optimization. Proceedings of 46th annual conference of Canadian Aeronautics and Space Institute, Montreal, Quebec, pp. 29-36.
- Rogers, P. P. and M. B. Fiering (1986). "Use of Systems Analysis in Water Management." *Water Resources Research* 22(9): 146s-158s.
- Ross, T. J. (2004). *Fuzzy logic with engineering applications*. UK, John Wiley and Sons.
- Russell, D. and C. J. Kim (1993). "Automatic generation of membership function and fuzzy rule using inductive reasoning." *IEEE Trans paper* 0-7803-1485-9/93.
- Simonovic, S. P. (2009). *Managing Water Resources: Methods and Tools for a Systems Approach*. London, UNESCO, Paris and Earthscan James & James.
- Sniedovich, M. (2011). *Dynamic programming: Foundations and Principles*. Boca Raton, CRC Press.
- Storn, R. (1996). On the usage of differential evolution for function optimization. Biennial conference of the North American Fuzzy Information Processing Society (NAFIPS). Berkeley: 519-523.
- Storn, R. and K. Price (1995). *Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces*. Technical Report TR-95 012, Berkeley, CA.

- Storn, R. and K. Price (1997). "Differential evolution-A simple and efficient heuristic for global optimization over continuous spaces." *Journal of Global Optimization* 11(4): 341-359.
- UTRCA. (January 12, 2012). "Wildwood Dam." Retrieved January 22, 2012, 2012, from http://www.thamesriver.on.ca/Water_Management/Wildwood_Dam.htm.
- UTRCA (1993). *Low Flow Hydrology and Operations Optimization Study. Wildwood and Pittock Reservoirs. Final Report.*
- Wolpert, D. H. and W. G. Macready (1997). "No free lunch theorems for optimization." *IEEE Transactions on evolutionary computation* 1(1): 67-82.
- Wurbs, R. A. (1998). "Dissemination of Generalized Water Resources Models in the United States." *Water International* 23(3): 190-198.
- Yeh, W. W. (1985). "Reservoir Management and Operations Models: A State-of-the-Art Review." *Water Resources Research* 21(12): 1797-1818.
- Yu, X. and M. Gen (2010). *Introduction to Evolutionary Algorithms*, Springer.
- Zadeh, L. A. (1965). "Fuzzy sets." *Information and control* 8: 338-358.
- Zaharie, D. (2003). Control of population diversity and adaptation in differential evolution algorithms. *Proc. of Mendel 2003, 9th Internat. Conference on Soft Computing, Brno*, pp. 41-46.
- Zhang, J. and A. Sanderson (2009). *Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization*. Berlin, Springer.

Appendices

Appendix A: Fuzzy Set Theory

The Fuzzy set theory was intentionally developed to try to capture judgmental belief, i.e. the uncertainty caused by the lack of knowledge or by ambiguity. The concept of a fuzzy set can be described as a “class” (set) with a continuum of grades of membership (Zadeh, 1965). Each object within a fuzzy set is graded in the interval $[0, 1]$. For example, in the class of animals, rocks may be said to have 0 degree of membership in the set of animals that is they do not belong, while cats may have full membership and belong. These definitions are common to traditional ordinary sets, where the values are crisp either belonging or not with no partial degree of belonging (Zadeh, 1965). Fuzzy sets extend the ordinary sets; consider that in the set of animals starfish have an ambiguous status and thus hold a degree of membership in the interval $[0, 1]$, i.e. partial membership. Therefore, starfish can be properly represented without the need to classify them as either belonging or not to the set (class). Fuzziness thus measures the degree to which an event occurs, not whether it occurs, a contrast to probability theory.

A fuzzy set (class) is characterized by a membership (characteristic) function which associates each member of the fuzzy set with a real number in the interval $[0, 1]$ (Zadeh, 1965; Ross, 2004). The membership function essentially embodies all fuzziness for a particular fuzzy set; its description is the essence of a fuzzy property or operation. There are numerous ways to assign membership values or functions to fuzzy variables; more ways than there are to assign probability density functions to random variables. In the following sections a sample of the available methods for assigning membership values or functions are summarized. For further details the reader is directed to the textbook by Ross (2004).

Intuition

This method is derived simply from the capacity of humans to develop membership functions through their own innate intelligence and understanding (Ross, 2004). In order to utilize intuition, contextual and semantic knowledge about an issue is essential. Thus,

the membership function development is dependent on the subjectivity of the individual or individuals consulted in its development. A single fuzzy variable may have more than one membership function, that is, there may be many partitions. An important characteristic for the purposes of use in fuzzy operations is that these partitions overlap.

Inference

The inference method comes from our ability to perform deductive reasoning. When given a body of facts or knowledge, we are able to deduce or infer a conclusion. The inference method can take many forms; consider an example of identifying a triangle when we possess a formal knowledge of geometry and geometric shapes, Ross (2004). In identifying a triangle, let A , B and C be the inner angles of a triangle in the order $A \geq B \geq C \geq 0$ and let U be the universe of triangles, such that

$$U = \{(A, B, C) \mid A \geq B \geq C \geq 0; A + B + C = 180^\circ\} \quad (6.1)$$

We can infer membership of different triangle types, because we possess knowledge of geometry. We can determine if a triangle is approximately isosceles by developing an algorithm for the membership. Meeting the constraints of Eq. (6.1) we have:

$$\mu_1(A, B, C) = 1 - \frac{1}{60} \min(A - B, B - C) \quad (6.2)$$

For example, if $A = B$ or $B = C$, the membership value of the isosceles triangle is $\mu_1 = 1$. However, if $A = 120^\circ$, $B = 60^\circ$, $C = 0^\circ$, then $\mu_1 = 0$. In the first case we thus have full membership or belonging of the fuzzy variable in the fuzzy set for an approximate isosceles triangle, while the second case is a total contrast.

Rank Ordering

The approach arises from assessing preferences by a single individual, a committee, a poll and other opinion methods that can be used to assign membership values to a fuzzy variable (Ross, 2004). Preferences are determined by pairwise comparisons and these determine the ordering of the membership. This method is similar to finding relative preferences through a questionnaire and developing membership functions as a result.

Neural Networks

Neural network is a technique that seeks to build an intelligent program using models that try to recreate the working of neurons in the human brain. Neurons are believed to be responsible for the humans' ability to learn; thus, the goal is to implement this to machine language as a tool to generate membership functions. The use of neural networks in membership function generation is centered on a training process (learning based on available data for input) and an unsupervised clustering process (Ross, 2004). After training, the degree of a membership function for a given input value may be estimated through network computation. That is to say, each input value has a certain estimated degree of belonging to a cluster which is equivalent to the degree of the membership function represented by the cluster.

Genetic Algorithms

Genetic algorithms use the concept of Darwin's theory of evolution in searching for the best solution of a given set based on the principle of "survival of the fittest" (Ross, 2004). Among all possible solutions, a fraction of the good solutions is selected while others are eliminated. The selected solutions undergo a process of reproduction, crossover, and mutation to create a new generation of possible solution. The process continues until there is a convergence within a generation. The genetic algorithms can be used in the derivation of membership functions. The process starts by assuming some functional mapping for a system (membership functions and their shapes for fuzzy variable/s). The membership functions are then converted to a code familiar to the algorithm, bit strings (zeros and ones) which can then be connected together to make a longer chain of code for

manipulation in the genetic algorithm (i.e. crossover, elimination, reproduction). An evaluation function is used to evaluate the fitness of each set of membership functions (parameters that define the functional mapping). Based on the fitness value, unsatisfactory strings are eliminated and reproduction of satisfactory strings proceeds for the next generation. This process of generating and evaluating strings is continued until the membership functions with the best fitness value are obtained.

Inductive Reasoning

This approach utilizes the inductive reasoning to generate the membership functions by deriving a general consensus from the particular (Ross, 2004). Inductive reasoning assumes availability of no information other than a set of data (Russell & Kim, 1993). The approach is to partition a set of data into classes based on minimizing entropy. The entropy, S , where only one outcome is true, is the expected value of the information contained in the data set and is given by

$$S = -k \sum_{i=1}^N [p_i \ln p_i + (1 - p_i) \ln(1 - p_i)] \quad (6.3)$$

where the probability of the i th sample to be true is p_i and N is the number of samples. The minus sign in front of the parameter k in Eq. (6.3) ensures that entropy will be a positive value, greater than or equal to zero. Through iterative partitioning, the segmented data calculation of an estimate for entropy is possible. The result is a solution of points in the region of the data interval, used to define the membership function. The choice of shape for membership functions is arbitrary, as long as some overlap is present between membership functions; therefore simple shapes, like triangles, which exhibit some degree of overlap, are often a sensible choice.

Appendix B: Mamdani Fuzzy Inference

The fuzzy approach used for simulation is derived from utilizing the fuzzy inference method, based on the representation of human knowledge in IF-THEN rule-based form, such that it becomes possible to infer a conclusion or fact (consequent) given an initial known fact (premise, hypothesis, antecedent) (Ross, 2004).

A typical form of the IF-THEN rule-based form, also referred to as the deductive form, is shown in the expression below:

$$\text{IF } \textit{premise (antecedent)}, \text{ THEN } \textit{conclusion (consequent)} \quad (6.4)$$

The fuzzy simulation (rule-based system) is most useful in modeling complex systems that can be observed by humans. Linguistic variables are used as antecedents and consequents. These linguistic variables can be naturally represented by fuzzy sets and logical connectives of these sets.

Mamdani's fuzzy inference method is the most commonly seen fuzzy simulation methodology and is the methodology presented in this report (Ross, 2004). The method was originally proposed as an attempt to control a steam engine and boiler combination by synthesizing a set of linguistic control rules obtained from experienced human operators. The Mamdani inference method is a graphical technique that follows five main steps: (1) development of fuzzy sets and linguistic rules, (2) fuzzification of inputs, (3) application of fuzzy operators, (4) aggregation of all outputs, and (5) defuzzification of aggregated output.

Step 1. Development of fuzzy sets and linguistic rules

To begin, the Mamdani form rules may be described by the collection of r linguistic IF-THEN expressions. Equation (6.5) shows the expression for a fuzzy system with two non-interactive inputs x_1 and x_2 (antecedents) and a single output (consequent) y . The concept holds for any number of antecedents (inputs) and consequents (outputs).

$$\text{IF } x_1 \text{ is } A_1^k \text{ and(or)} x_2 \text{ is } A_2^k \text{ THEN } y^k \text{ is } B^k \quad \text{for } k = 1, 2, \dots, r \quad (6.5)$$

where A_1^k and A_2^k are the fuzzy sets representing the k th antecedent pairs, and B^k is the fuzzy set representing the k th consequent. The membership functions for the fuzzy sets may be generated with one of the methods discussed in Appendix A.

Step 2. Fuzzification of Inputs

The inputs to the system, x_1 and x_2 , are scalar values. In order to proceed with the inference method, the corresponding degrees to which the inputs belong to the appropriate fuzzy sets via membership functions need to be found. Fuzzification of the input thus requires the membership function of the fuzzy linguistic set to be known; the corresponding degree of membership for the scalar input belonging to the universe of discourse is found through function evaluation. Figure 6.1 outlines the procedure in a graphical form.

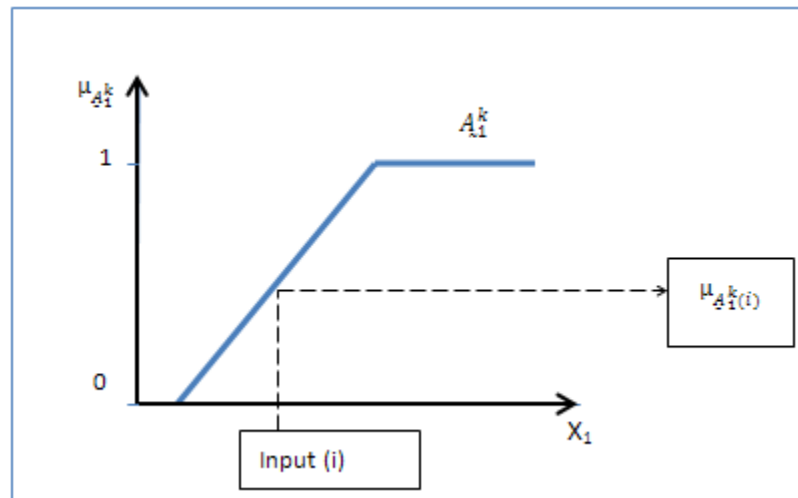


Figure 6.1. Fuzzification of scalar input from known membership function.

It should be noted that the input to any fuzzy system can be a membership function, such as gauge reading that has been fuzzified already. Either way, the methodology is the same as the one that employs fuzzy singletons (scalar values) as the input.

Step 3. Application of fuzzy operators

Once the inputs are fuzzified, the degree by which each condition of the antecedent is satisfied is known for each rule. If there are multiple antecedent conditions for each rule, as in the case of expression (6.5), then a fuzzy operator is used to obtain one number that represents the antecedent for that rule. This number is applied to the output function, producing a single truth value for the rule. The logical operators commonly employed are described.

The expression in (6.5) has conjunctive antecedents and for illustration shows disjunctive antecedents in brackets.

For conjunctive antecedents, assuming a new fuzzy subset A_s as

$$A_s^k = A_1^k \cap A_2^k \quad \text{for } k = 1, 2, \dots, r \quad (6.6)$$

expressed by means of the membership function shown in Figure 6.2:

$$\mu_{A_s^k}(x) = \min \left[\mu_{A_1^k}, \mu_{A_2^k} \right] \quad \text{for } k = 1, 2, \dots, r. \quad (6.7)$$

For disjunctive antecedents, a similar procedure follows. This time, the fuzzy set A_s is defined as

$$A_s^k = A_1^k \cup A_2^k \quad \text{for } k = 1, 2, \dots, r \quad (6.8)$$

expressed by means of the membership function shown in Figure 6.2

$$\mu_{A_s^k}(x) = \max \left[\mu_{A_1^k}, \mu_{A_2^k} \right] \quad \text{for } k = 1, 2, \dots, r. \quad (6.9)$$

Given the above, the compound rule may be rewritten as

$$\text{IF } A_{\tilde{S}}^k \text{ THEN } B_{\tilde{S}}^k \quad \text{for } k = 1, 2, \dots, r \quad (6.10)$$

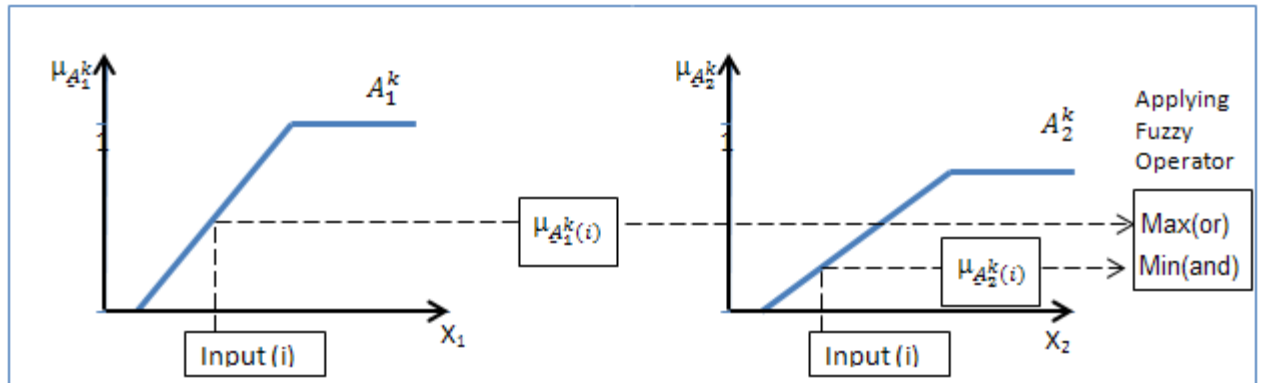


Figure 6.2. Fuzzy operator use for the generalized expression (6.5) of a rule.

Step 4. Aggregation of outputs

It is common for a rule-based system to involve more than one rule. As such, in order to reach a decision or overall conclusion, the aggregation of individual consequents or outputs contributed by each rule is required. Thus, all the outputs are combined into a single fuzzy set which may be defuzzified in the final step to obtain a scalar solution.

The aggregation of outputs may be achieved in two ways: (1) max-min truncation, or (2) max-product scaling. Only the first case will be discussed here. In the max-min case, aggregation is achieved by the minimum or maximum membership function value from the antecedents (depending on the logical operator used in the rule), propagating through to the consequent, thereby truncating the membership function for the consequent of each rule. This procedure is applied for each rule. The truncated membership functions of each rule will need to be combined. This may be achieved by disjunctive or conjunctive rules, using the same fuzzy operators as in Step 3.

If the system of rules needs to be jointly satisfied, the truncated outputs should be aggregated as a conjunctive system; the rules are connected by “and” connectives. In the case where the objective is to be satisfied for at least one rule, the aggregation of outputs may be treated as a disjunctive system, where the rules are connected by “or” connectives. Figure 6.3 illustrates the aggregation of outputs into a single fuzzy membership function. Each antecedent is treated as conjunctive and the aggregation of outputs of each rule is treated as a disjunctive system.

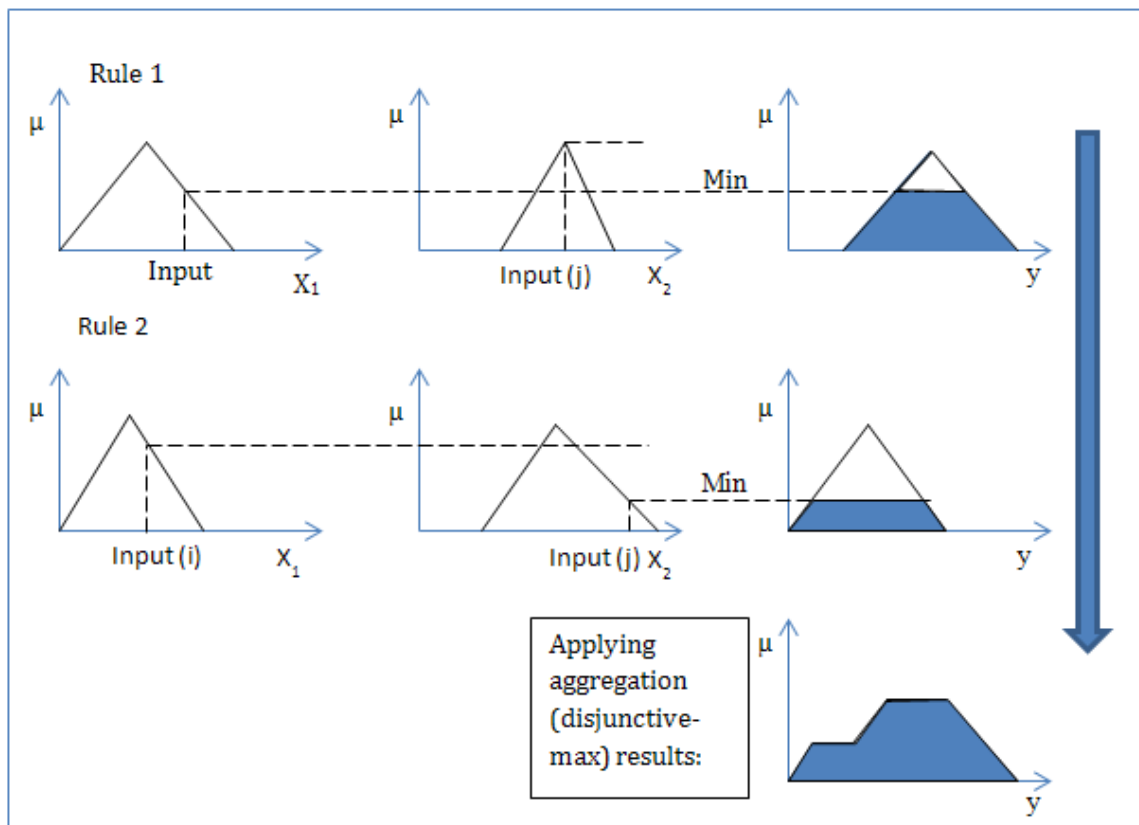


Figure 6.3. Aggregation of rule outputs into a single fuzzy membership function.

Step 5. Defuzzification of aggregated result

The final objective of the rule-based system simulation is typically a single value obtained from the defuzzification of the aggregated fuzzy set of all outputs. Many defuzzification methods are available in the literature: max membership principle,

centroid method, weighted average method and numerous others. There is no single most suitable defuzzification method. Selection of the best method for defuzzification is context or problem-dependent. For the purpose at hand, the centroid method will be used because it is well established and physically appealing among all the defuzzification methods (Ross, 2004). The centroid method shown in Figure 6.4, may also be referred to as the center of gravity or center of an area. Its expression is given as

$$y^* = \frac{\int \mu_c(y) dy}{\int \mu_c(y) dy} \quad (6.11)$$

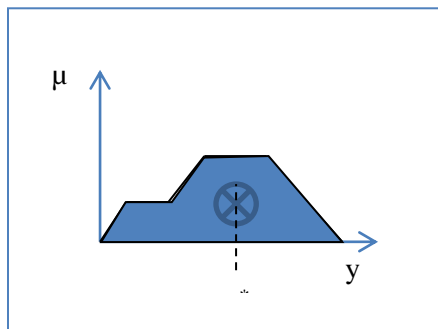


Figure 6.4. Centroid method for defuzzification.

Appendix C: Decision Support System for Implementation of DEO

Included alongside this thesis is the electronic installation file of the DSS, developed to solve optimization problems using the differential evolution algorithm. In addition to the installation file, the C# source code is also provided in electronic form due to its large size.

With the installation files provided, readers are encouraged to get familiar with the features of the DSS.

The file folder named **DEO-install**, once opened, contains:

ReadMe.txt : This file contains installation instructions and other helpful information.

Setup.exe: This is the main executable installation file that, upon launching, will install the DEO software onto the user's computer.

DEO-Examples folder: The folder contains convenient example input .deo files and documentation to familiarize a new user with the DEO software.

The file folder **DEO-Code**, once opened, contains many C# classes, one of which, titled Deopt.cs, contains most of the main algorithm code. The other classes include the interface, fuzzy inference class for FADE and various classes for parsing of the input functions. Opening the C# project file named DeOptimization using Microsoft Visual C# would be the most convenient way to access all the associated source code.

Appendix D: Wildwood optimization results

Table 6.1. Wildwood decision variables values after optimization [10^3 m^3]

Variable	Optimization Results			Target	
	DE	FDE ₁	FDE ₂	FDE ₁	FDE ₂
S_1	4706	4352	3088	6908	7000
S_2	4206	6688	4016	6610	6000
S_3	9252	13824	12336	10090	10000
S_4	13765	14880	15804	15359	15000
S_5	13318	14464	16238	17516	17000
S_6	15208	14624	16736	17860	17000
S_7	14019	14208	15736	17194	17000
S_8	11404	11408	12800	15523	8000
S_9	11018	11312	12608	11660	8000
S_{10}	11245	11776	12576	8449	8000
S_{11}	11407	11520	12672	4222	8000
S_{12}	10107	8800	8096	4039	7000
R_1	5999	6400	7568	4345	4000
R_2	3165	512	1248	3463	3500

R_3	4317	2304	992	2387	2500
R_4	1048	3840	1463	1763	1500
R_5	3850	3328	2996	1966	4000
R_6	2607	3920	4468	4596	5000
R_7	3717	2816	3480	3180	4000
R_8	3480	4352	4388	4719	5000
R_9	3472	6688	3197.5	5786	6000
R_{10}	5023	13824	5248	8511	8000
R_{11}	3504	14880	4064	6153	5000
R_{12}	5268	14464	9376	4940	5000

Curriculum Vitae

Name: Dejan Vucetic

Post-secondary Education and Degrees: The University of Western Ontario
London, Ontario, Canada
2006-2010 B.A.

Honours and Awards: Graduated with distinction
2010

Related Work Experience: Teaching Assistant
The University of Western Ontario
2010-2012

Publications:

Vucetic, D. and S.P. Simonovic (2011). Water Resources Decision Making Under Uncertainty. Water Resources Research Report No. 073. London, Canada: Facility for Intelligent Decision Support, pp. 143.

Vucetic, D. and S. P. Simonovic (2012). "Fuzzy Differential Evolution Algorithm." Fuzzy Sets and Systems. (Under review)