UNIVERSITY of OULU
OULUN YLIOPISTO

CONTROL ENGINEERING LABORATORY

Evolutionary algorithms in nonlinear model identification

Aki Sorsa, Anssi Koskenniemi and Kauko Leiviskä

Report A No 44, September 2010

University of Oulu
Control Engineering Laboratory
Report A No 44, April 2008

Evolutionary algorithms in nonlinear model identification

Aki Sorsa, Anssi Koskenniemi and Kauko Leiviskä

University of Oulu, Control Engineering Laboratory

Abstract: Evolutionary algorithms are optimization methods which basic idea lies in biological evolution. They suit well for large and complex optimization problems. In this study, genetic algorithms and differential evolution are used for identifying the parameters of the nonlinear fuel cell model. Different versions of the algorithms are used to compare the methods and their available operators. The problem with the studied algorithms is the parameters that regulate the development of the population. In this report, some suitable methodology is proposed for defining appropriate tuning parameters for the used algorithms. The results show that the used methods suit well for nonlinear parameter identification but that differential evolution performs a bit better on average. The results also show that the studied identification problem has a lot of local minima that are very close to each other and thus the optimization problem is very challenging.

Keywords: genetic algorithms, differential evolution, PEM fuel cell

ISBN 978-951-42-6332-3 (pdf)
ISSN 1238-9390

University of Oulu
Control Engineering Laboratory
P.O. Box 4300
FIN-90014 University of Oulu

TABLE OF CONTENTS

1 INTRODUCTION

Evolutionary algorithms are an interesting subgroup of search and optimization methods which has become more popular with improved computer technology. The basic idea of evolutionary algorithms lies in biological evolution which has shown its competence during millions of years. Evolutionary algorithms suit particularly well for large and complex optimization problems. They are typically based on the population of possible solutions which is then evolved to better solutions to find the optimal one. The evolution of the population is regulated by operators derived from the nature. Typical operators are selection, crossover and mutation. The possible solutions are assessed through an objective function. The better solutions have higher probability to reproduce and thus their properties are enriched in the later generations. (Sarker et al. 2003)

Numerical methods, such as gradient methods, are typically used in parameter identification. These methods typically search near the initial guess and thus are prone to get trapped into local optima. In such a case, the optimal parameter values are not found and the model performance deteriorates. It is possible to try multiple initial guesses and then select the best solution or to use other methods such as evolutionary algorithms. (Ikonen and Najim 2002)

Evolutionary algorithms are more likely to find the global optimum than the traditional algorithms. That is because the evolutionary algorithms search the optimum from multiple directions simultaneously and they also allow (in some cases) the search to proceed to a direction leading to worse solutions. Thus evolutionary algorithms are likely to escape from local optima. Another benefit of evolutionary algorithms is that they do not require prior knowledge about the optimization problem. (Storn and Price 1997; Chipperfield 1997)

The downside of evolutionary algorithms is that it is never certain that the search converges to the global optimum. Thus, one possibility is to use evolutionary algorithms in finding promising regions in the search space and then continue with traditional methods. For such tasks, different kinds of hybrid algorithms have been proposed. For example, Wolf and Moros (1997) used genetic algorithms to find the initial guesses for a Nelder-Mead optimization routine. Katare et al. (2004) used genetic algorithms similarly except that they used a modified Levenberg-Marquardt optimization method. A third type of hybrid algorithm is presented in Mo et al. (2006). They use the Nelder-Mead search as a part of their objective function. In other words, each population member serves as an initial guess for the Nelder-Mead search.

In this study, two popular methods, genetic algorithms and differential evolution, are used for identifying a nonlinear process model. The used model is built for a PEM fuel cell. In a PEM fuel cell, electrodes are separated by a solid polymer-electrolyte-membrane structure through which only protons can permeate. The behaviour of the fuel cell is influenced by the prevailing conditions such as temperature and pressure.

2 FUEL CELLS

Clean energy production has become very topical due to environmental problems such as acid rains, CO_2 emissions and reduction of air quality. One of the most promising alternatives to overcome these problems is fuel cells which convert chemical energy to electricity. PEM fuel cells (polymer electrolyte membrane) include an anode and a cathode together with a membrane separating them. The structure of a PEM fuel shell is shown in Figure 1. The membrane allows only protons to permeate it. In its simplest form, the fuel cell uses only hydrogen and oxygen even though the latter one can also be substituted with air. Hydrogen gas is fed to the anode where it is oxidized according to (1). The released electrons are transported to the cathode through a conducting element as shown in Figure 1. H^+-ions migrate through the membrane and also end up in the cathode where they are reduced according to (2). For the reduction reaction, oxygen is provided to the cathode. When hydrogen and oxygen react only water is produced. The oxidation and reduction reactions together with the overall reaction are given in Table 1. (Mo et al. 2006)

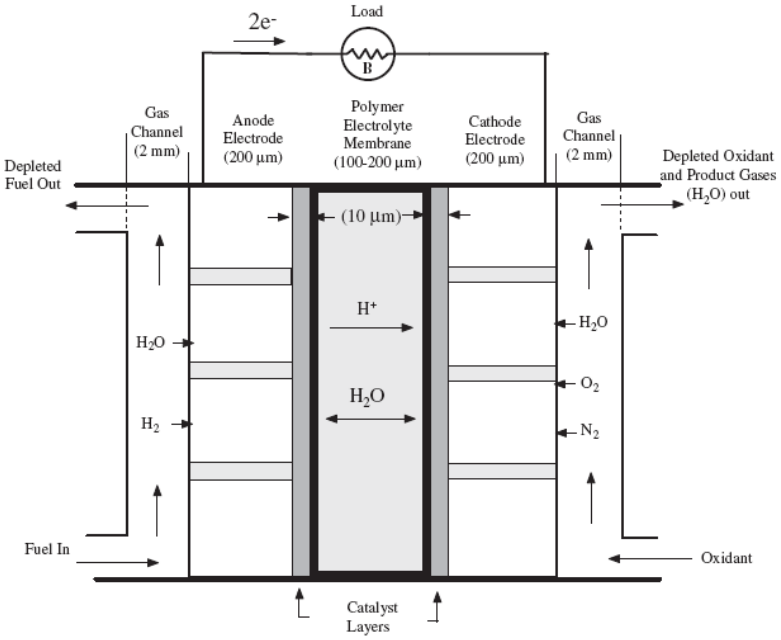

Figure 1. The structure of a PEM fuel cell. (Mo et al. 2006)

Table 1. Chemical reactions occurring in a PEM fuel cell.

Oxidation of H2 (anode)	$2H_2 \rightarrow 4H^+ + 4e^-$	(1)
Reduction of H+ (cathode)	$O_2 + 4H^+ + 4e^- \rightarrow 2H_2O$	(2)
Overall reaction	$2H_2 + O_2 \rightarrow 2H_2O$	(3)

The models proposed for PEM fuel cells typically include mass and energy balances combined with the electrochemical part describing the relation between the outlet voltage

and current. Mass and energy balances are not described in more detail because only the electrochemical part is studied here. The same has been done earlier in Ohenoja & Leiviskä (2010).

The internal potential of the fuel cell (E_{cell} [V]) is the theoretical maximum outlet voltage obtained from the fuel cell in thermodynamic equilibrium and with no external load. It can be calculated from the Nernst equation given by (Corrêa et al. 2004)

$$E_{cell} = E_{Nernst} = \frac{\Delta G}{2F} + \frac{\Delta S(T - T_{ref})}{2F} + \frac{RT}{2F}\left[ln(p_{H_2}) + \frac{1}{2} ln(p_{O_2}) \right]. \quad (4)$$

Above, ΔG and ΔS [J/mol] are the changes in the free Gibb's energy and entropy, T and T_{ref} [K] are the operating and reference temperatures, p_{H_2} and p_{O_2} [atm] are the fugacities (effective partial pressures) of hydrogen and oxygen, F (= 96487 C/mol) is the Faraday constant and R (= 8.314 J/mol K) is the ideal gas constant. Applying the numerical values for ΔG, ΔS and T_{ref} in the standard temperature and pressure (4) becomes (Corrêa et al. 2004)

$$E_{cell} = 1.229 - 0.85 \cdot 10^{-3}(T - 298.15) + 4.31 \cdot 10^{-5} T\left[ln(p_{H_2}) + \frac{1}{2} ln(p_{O_2}) \right]. \quad (5)$$

When pure hydrogen and oxygen are utilized in the fuel cell, the fugacities are (Mo et al. 2006)

$$p_{O_2} = \frac{p_c - RH_c\, p_{H_2O}^{sat}}{1 + 3.7619\, exp(0.291(i/A)T^{-0.832})} \text{ and} \quad (6)$$

$$p_{H_2} = 0.5 RH_a\, p_{H_2O}^{sat}\left[\left(\frac{RH_a\, p_{H_2O}^{sat}}{p_a} exp(1.635(i/A)T^{-1.334}) \right)^{-1} - 1 \right]. \quad (7)$$

In (6) and (7), RH_a and RH_c are the relative humidities of vapour and p_a and p_c [atm] are the inlet pressures in the anode and cathode. T [K] is the cell temperature, i [A] is the cell current and A is the effective electrode area [cm^2]. The saturation pressure of water vapour ($p_{H_2O}^{sat}$) depends on the cell temperature and is given by (Mo et al. 2006)

$$log_{10}(p_{H_2O}^{sat}) = 2.95 \cdot 10^{-2}(T - 273.15) - 9.18 \cdot 10^{-5}(T - 273.15)^2 + 1.44 \cdot 10^{-7}(T - 273.15)^3 - 2.18 . \quad (8)$$

The theoretical maximum outlet voltage is not obtained from the fuel cell due to the voltage losses of various natures. In Wang et al. (2005), three components describing the voltage losses are used. Namely these are the activation overpotential V_{act} [V], the ohmic voltage drop V_{ohmic} [V] and the concentration overpotential due to the mass transfer limitations V_{con} [V]. Now, the outlet voltage of the fuel cell is

$$V_{cell} = E_{cell} - V_{act} - V_{ohmuic} - V_{con}. \quad (9)$$

The activation overpotential is caused by the phenomena occurring in the electrodes, especially in the cathode. It is significant especially with small currents. The activation overpotential can be calculated from (Mann et al. 2000)

$$V_{act} = -\left[\xi_1 + \xi_2 T + \xi_3 T\, ln(i) + \xi_4 T\, ln\left(c^*_{O_2}\right)\right]. \quad (10)$$

Above, ξ_j are empirical coefficients and $c^*_{O_2}$ [mol/cm^2] is the dissolved oxygen concentration at the cathode/membrane interface. The dissolved oxygen concentration can be assessed based on the Henry's law to be (Mo et al. 2006)

$$c^*_{O_2} = \frac{p_{O_2}}{5.08 \cdot 10^6} exp\left(\frac{498}{T}\right). \quad (11)$$

The ohmic voltage drop includes the resistances due to the fuel cell structure. Such resistances are, for example, the resistance to electron and proton transfer in the electrodes and the membrane. The ohmic voltage drop is almost linearly dependent to the current and is given by (Corrêa et al. 2004)

$$V_{ohm} = i\left(R_m + R_c\right). \quad (12)$$

Above, R_m [Ω] is the resistance of the membrane and R_c [Ω] is the overall resistance to electron transfer in the fuel cell. The resistance of the membrane is a function of the membrane thickness (l_m [cm]) and the cell active area given by (Mann et al. 2000)

$$R_m = \frac{r_m l_m}{A}, \quad (13)$$

where r_m [Ωcm] is the resistivity of the membrane. Furthermore, Mann et al. (2000) define the membrane resistivity as

$$r_m = \frac{181.6\left[1 + \frac{0.03i}{A} + 0.062\left(\frac{T}{303}\right)^2\left(\frac{i}{A}\right)^{2.5}\right]}{\left[\lambda - 0.0634 - 3\left(\frac{i}{A}\right)\right] exp\left[4.18\left(\frac{T-303}{T}\right)\right]}, \quad (14)$$

where λ is an empirical coefficient. Mass transfer limits the rate at which hydrogen and oxygen are supplied to the electrodes and thus defines the maximum outlet current density (I_{max} [A/cm^2]). Typical values for I_{max} in PEM fuel cells range from 500 to 1500 mA/cm^2 (Corrêa et al. 2004). The voltage drop due to the mass transfer is defined by (Mo et al. 2006)

$$V_{con} = -b\,ln\left(1 - \frac{I}{I_{max}}\right). \tag{15}$$

Above, b [V] is an empirical coefficient and I [A/cm^2] is the cell current density. Equations (4)-(15) can be used to express the relationship between the outlet current and voltage of the fuel cell. Figure 2 presents a typical current-voltage curve (polarization curve) of a fuel cell operating at 70 °C and standard pressure. The figure shows that even with no external load the theoretical maximum voltage is not reached. Furthermore, very low current densities lead to the cell voltage experiencing a significant voltage drop which then settles to a gentler and almost linear drop with increasing current densities. The voltage drop then increases again with higher values of current density. (Larminie and Dicks 2003)

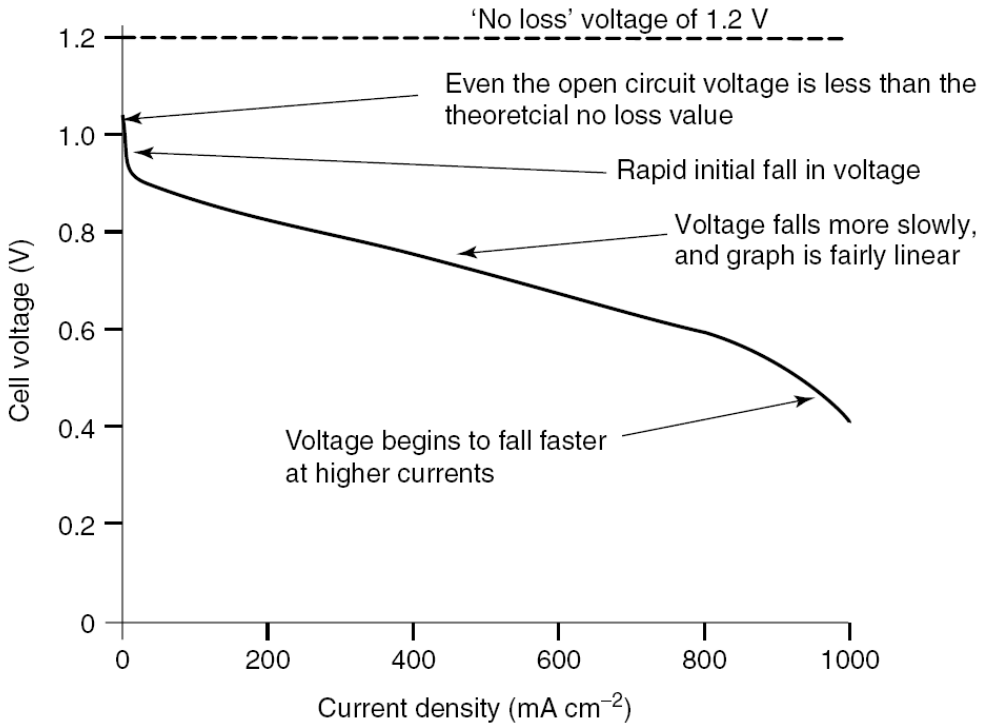

Figure 2. A typical current-voltage curve for a fuel cell operating at low temperature and standard pressure. (Larminie and Dicks 2003)

3 GENETIC ALGORTIHMS

Genetic algorithms are a class of robust evolutionary optimization methods suitable for various types of optimization problems. The basic idea is derived from biological evolution. During the development of genetic algorithms, new features inspired by the nature are added even though the significance of the features is not clear. It has been thought that if evolution has favoured certain feature in nature that feature must be advantageous. For example, crossing methods have been generated and selected based on this. (Goldberg 1989)

Genetic algorithms are rather easy to use but still their problem is the great number of parameters influencing the performance of the algorithm. Such parameters are, for example, crossing and mutation probabilities, rate of elitism and population size. In order to make the algorithm work properly, all these parameters must be set to appropriate values. (Katare et al. 2004)

Typically, genetic algorithms are based on three operations: selection, crossover and mutation. A flow chart of a typical genetic algorithm is given in Figure 3. As shown in the figure, the algorithm is iterative generating a population after population as long as the termination criterion is satisfied.

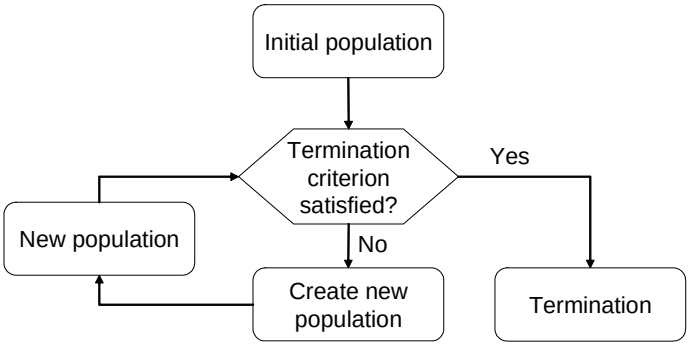

Figure 3. A flow chart of a genetic algorithm.

3.1 Binary vs. real-valued coding

The early applications of genetic algorithms used binary coded chromosomes. The binary strings were decoded and then converted to real-valued parameters in order to evaluate the fitness of the chromosomes. The conversion from binary to real values obviously increases the computational cost and thus it becomes computationally very expensive when the optimization problem is large and requires high accuracy (Michalewicz 1996). Binary coding also limits the accuracy of the solution depending on the number of bits used for representing each parameter. Problems with binary coding may also arise because changing one bit may lead to great changes in the parameter value due to the conversion from binary to real value. Generally, real-valued genetic algorithms suit better for real engineering problems than binary algorithms (Chang 2007).

3.2 Selection

In selection, chromosomes are selected from the population to act as parents in crossing. The principle in genetic algorithms is to give higher probability to better chromosomes to act as parents and thus produce offspring. If the roulette wheel selection is applied, the probability to be selected is directly proportional to the fitness of the chromosome. Even the worst chromosome may be selected even though the probability of this is quite low. In roulette wheel selection, the cumulative fitness of the chromosomes is calculated and a random number between zero and the sum of fitnesses is created. The chromosome corresponding to the cumulative fitness exceeding the random number is selected. (Davis 1991)

In tournament selection, a certain number of candidate chromosomes is selected randomly from the population. Among these, the one having the highest fitness is selected. The same is repeated as long as the needed a number of parents is selected. The tournament selection allows weaker chromosomes to be selected only if they attend a tournament with even weaker other candidates. The very worst chromosome, however, can never be selected. (Fogel 2000)

The selection method affects the overall efficiency of the algorithm. Selecting mainly the better chromosomes makes the algorithm converge faster. Sometimes the convergence is even too fast. When weaker chromosomes are also selected the rate of convergence is slower but the search space is better covered. (Michalewicz 1996)

3.3 Crossover

Crossover is an essential operator in genetic algorithms which combines some desired properties of chromosomes. It is possible that moderate chromosomes include very good values for some parameters. Thus combining these good values of several chromosomes leads to better chromosomes. Crossover rate is typically regulated by defining a crossover probability. This means that not all the selected parents are crossed but some are placed directly into the new population. (Goldberg 1989)

In the following, some common crossover techniques are presented. Besides the ones presented here even more techniques have been proposed in the literature. More methods can be found, for example, in Deep and Thakur (2007a) and Sorsa et al. (2008).

3.3.1 Point and uniform crossover

In one-point crossover, a random point is selected beyond which the chromosome segments are switched to produce two offspring. The closer the good segments are to each other the easier they end up in the same offspring. One-point crossover is not able to combine good parameter combinations if they are located further away from each other in chromosomes. In such cases, two-point crossover can be used. (Davis 1991)

In uniform crossover, two parents are used to create two offspring. Offspring are created in such a way that each of the parameters are taken randomly from the parents. With uniform crossover, the orientation of the parameters does not affect the offspring which was the case with one-point crossover. However, uniform crossover may destroy previously found good combinations. (Davis 1991)

3.3.2 Arithmetic crossover

The crossover methods presented previously move the parameters unchanged from parents to offspring. Such methods are not necessarily the most suitable for real-value coded genetic algorithms because they can not create new solutions but only combine the existing solutions in different ways.

In arithmetic crossover, two parents are arithmetically combined. The offspring are created somewhere between the parents depending on the random number α. The offspring according to arithmetic crossing are (Michalewicz 1994)

$$\begin{aligned} X_{1,G+1} &= \alpha X_{1,G} + (1-\alpha) X_{2,G} \\ X_{2,G+1} &= \alpha X_{2,G} + (1-\alpha) X_{1,G} \end{aligned}, \tag{16}$$

where α is a uniform random number between 0 and 1, $X_{1,G}$ and $X_{2,G}$ are the parents and $X_{1,G+1}$ and $X_{2,G+1}$ are the offspring.

3.3.3 Linear crossover

In linear crossover, two parents produce three offspring. The fittest two of these are chosen to next population. The offspring are created in such a way that they are on a line crossing each of the parents. The offspring are (Herrera et al. 1998)

$$\begin{aligned} X_{1,G+1} &= \frac{1}{2} X_{1,G} + \frac{1}{2} X_{2,G} \\ X_{2,G+1} &= \frac{3}{2} X_{1,G} - \frac{1}{2} X_{2,G} \\ X_{3,G+1} &= -\frac{1}{2} X_{1,G} + \frac{3}{2} X_{2,G} \end{aligned} . \tag{17}$$

3.3.4 Heuristic crossover

Heuristic crossover considers the fitness values of the parents in determining the direction of the crossover. Furthermore, heuristic crossover produces only one offspring at maximum from two parents. The offspring according to heuristic crossover is (Michalewicz 1994)

$$X_{1,G+1} = \alpha(X_{2,G} - X_{1,G}) + X_{2,G} . \tag{18}$$

Above, the parent $X_{2,G}$ is fitter than $X_{1,G}$ and thus the offspring is created near the fitter parent. If the offspring is beyond the feasible area, a new offspring with different value of α is created. If a feasible offspring is not created after several attempts, no offspring at all is created. (Michalewicz 1994)

3.4 Mutation

In mutation, random changes are produced to chromosomes to create new information to the population (Davis 1991). Mutation is also useful in changing duplicate chromosomes existing in the population (Michalewicz 1994). While crossover combines the information in chromosomes and makes the population converge, mutation produces new information and maintains diversity of the population (Deep 2007a). The mutation rate is regulated by the predefined mutation probability which typically is small. For each parameter, a random number is generated and compared to the mutation probability. If the random number is lower than the mutation probability, mutation occurs (Davis 1991)

The most usual mutation operator in genetic algorithms is uniform mutation where the parameter to be mutated is simply replaced by a feasible random value. Uniform mutation is essential in early generations to maintain enough diversity and to guarantee that the search space is adequately covered. Another mutation operator is boundary mutation which can be applied when solving constraint optimization problems and the solution is near the constraints. In boundary mutation, the mutated parameter is replaced by the boundary value of the search space. (Michalewicz 1994)

Among the above presented mutation operators, many others have been proposed. These are not presented here but can be found, for example, in Deep & Thakur (2007b) and Sorsa et al. (2008).

3.5 Elitism

In genetic algorithms, the offspring replace the parents in the population. It is possible that the parents are fitter and thus it is possible to lose some good solutions from the population. To guarantee that the very best solutions never disappear from the population, elitism is applied. In elitism, the predefined number of the fittest chromosomes is moved directly to the new population. This may increase the dominance of the fittest chromosomes and thus decrease diversity. On the other hand, the convergence rate is increased. (Davis 1991)

4 DIFFERENTIAL EVOLUTION

Differential evolution was introduced in Storn and Price (1995). The algorithm is simple, easy to use and converges well (Zaharie 2002). Among evolutionary algorithms, it is closest to genetic algorithms. The fundamental difference is the basic principle of the mutation operator. While random changes are produced in genetic algorithms, the differences between chromosomes are utilized when arithmetically combining them in differential evolution (Feoktistov and Janaqi 2004). When compared to genetic algorithms, differential evolution only has a few tuneable parameters. For example, the simplest form of differential evolution has only three parameters: mutation coefficient *F*, crossing coefficient *CR* and population size *NP* (Storn and Price 1995).

In differential evolution, simple arithmetic operators are combined to traditional genetic operations (crossover, mutation and selection). Because the size of the step to be taken towards the optimum changes as the algorithm proceeds, the mutation operator must be adaptive. The differences between the chromosomes are a good indicator of an appropriate step size. When the variance among the population members increases or decreases so does the step size in differential evolution. (Bergey and Ragsdale 2005)

The fundamental idea in differential evolution is the technique for creating the trial vector which is obtained by combining a weighted difference vector to a base vector (Storn & Price 1995). The difference vector is obtained as a difference between two chromosomes while the base vector is a selected chromosome. To avoid premature convergence, there must be enough diversity in the population (Zaharie 2002). Figure 4 shows a flow chart of a basic differential evolution.

Because of the coding and the evolutionary operators, differential evolution suits very well for real-valued optimization problems. The algorithm is also quite easily convertible to suit for integer and binary problems. (Feoktistov and Janaqi 2004)

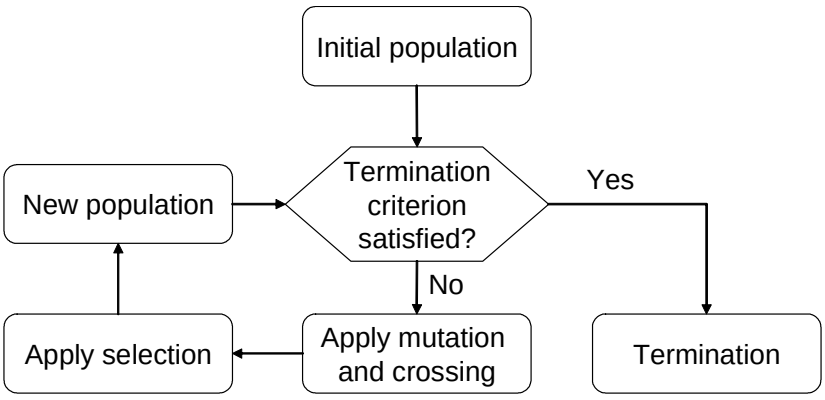

Figure 4. A Flow chart of a basic differential evolution.

4.1 Structure and principles

As mentioned above, differential evolution utilizes the differences between the population members. The mutant vector is created by selecting at least three chromosomes from the population. Two of these (at least) are used to create the difference vector which is then multiplied by the mutation coefficient *F* and then added to the base vector of mutation (the third chromosome). The mutant vector is given by (Price et al. 2005)

$$V_{i,G} = X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}) . \tag{19}$$

Above, $V_{i,G}$ is the mutant vector, $X_{r_1,G}$ is the base vector for mutation and $X_{r_2,G}$ and $X_{r_3,G}$ are the random vectors for creating the difference vector. The next step in differential evolution is crossover the mutant and the target ($X_{i,G}$) vectors. Each of the chromosomes acts as a target vector at a time. Through crossover, the trial vector ($U_{i,G}$) is obtained which is then compared to the target vector in the selection step. The selection method is the tournament method with two candidates. In other words, the fitter vector is moved to the new population according to (Price et al. 2005)

$$U_{i,G+1} = \begin{cases} U_{i,G}, \textit{if } J(U_{i,G}) \le J(X_{i,G}) \\ X_{i,G}, \textit{otherwise} \end{cases} . \tag{20}$$

The selection given in (20) guarantees that all the chromosomes in the new population are equal or better compared to the previous population. Thus differential evolution is an elitistic algorithm (Pant et al. 2009).

The overall algorithm can be summarized in the following steps.

1. Create the initial population and evaluate their fitnesses.
2. Apply mutation, crossing and selection. Each chromosome act as a target vector $X_{i,G}$ at a time. For each target vector,
3. Select the base vector for mutation and the vectors needed to calculate the difference vectors,
4. Calculate the mutant vector, $V_{i,G}$,
5. Create the trial vector $U_{i,G}$ through crossing and
6. Select $X_{i,G}$ or $U_{i,G}$ to be moved to the new population.
7. Evaluate the fitness of the population

If the termination criterion is satisfied, terminate, otherwise, go to step 2.

4.2 Different variations of differential evolution

Different kinds of variations of differential evolution have been developed. They differ in methods used for crossover and the selection of the base vector for mutation. Also the number of difference vectors may change. Typically, different variations are written as (Feoktistov and Janaqi 2004)

$$DE/a/b/c, \tag{21}$$

where *a* is the method for selecting the base vector for mutation, *b* is the number of difference vectors and *c* is the method used for crossing. The base vector for mutation is typically selected randomly (*rand*) or the best chromosome of the population (*best*) is used. Also the target vector can be used as the base vector for mutation in some cases (*current*). Among these three options, their combinations can also be used (*rand-to-best*, *current-to-rand* and *current-to-best*). The number of difference vectors is typically 1 or 2. Crossover methods are typically either binomial crossing (*bin*) or exponential crossing (*exp*) but also arithmetic crossing can be used (*arith*). (Bergey and Ragsdale 2005)

4.3 Initial population

An essential part of any evolutionary search is the generation of the initial population. When it is generated successfully, a good solution is found and the search convergences faster. Thus the computational time required to find the good enough solution is directly proportional to the quality of the initial population. (Rahnamayan et al. 2008)

If there is no prior knowledge about the optimum, it is typical to generate the initial population randomly between the lower and upper bounds defined for each parameter. The initial population is then (Storn et al. 2005)

$$P_0 = X_{i,0} = x_{i,j,0} = rand_j[0,1](u_j - l_j) + l_j, \tag{22}$$

where $X_{i,0}$ is the *i*:th chromosome of the initial population, $x_{j,i,0}$ is the *j*:th parameter of the *i*:th chromosome and l_j and u_j are the lower and upper bounds for the *j*:th parameter. Rahnamayan et al. (2008) propose the use of the opposite population and they report that about 10 % decrease in convergence time is obtained. In their approach, the random initial population is complemented with an opposite population. The opposite population is given by Rahnamayan et al. (2008)

$$O_0 = Z_{i,0} = z_{i,j,0} = l_j + u_j - x_{i,j,0}, \tag{23}$$

where $Z_{i,0}$ is the *i*:th chromosome of the opposite population and $z_{i,j,0}$ is the *j*:th parameter of the *i*:th chromosome. The original initial population P_0 and the opposite population O_0 are then combined and the fittest chromosomes are selected for the actual initial population. The idea of the method is that the random initial population and the opposite population each have a 50 % probability to be closer to the optimum. When the fitnesses of these populations are evaluated before proceeding, it is assumed that the resulting actual initial population is closer to the optimum than the original or the opposite populations. (Rahnamayan et al. 2008)

If there is prior knowledge about the optimum, it is worth using the information. The obvious selection then is to create the initial population in the neighbourhood of the

assumed optimum. The coverage of the initial population should be, however, large enough to guarantee that the optimum does not lie outside the initial population. When the coverage is large enough also enough variance is created for efficient search. (Storn and Price 1997)

The initial population in the neighbourhood of the assumed optimum can be created according to the uniform or Gaussian distributions depending on the accuracy of the assumed optimum. If the optimum is known quite accurately, the use of Gaussian distribution may lead to fast search. However, the premature convergence to a local optimum is also more probable. Overall, the uniformly distributed initial population is favoured because it represents the uncertainty about the exact optimum. (Price et al. 2005)

4.4 Mutation

When applying mutation two things have to be considered: the direction and size of the change. In differential evolution, these depend on the difference vector and especially the vectors used to create the difference vector. Thus increasing the population size or the number of vectors used for building the difference vector leads to more alternative mutation directions. (Feoktistov and Janaqi 2004)

Almost exclusively one or two difference vectors are used. According to Price (1996), two difference vectors make it easier to find an appropriate value for F. When one difference vector is used, the mutant vector is obtained with (19) and with two difference vectors, the mutant vector is (Storn and Price 1996)

$$V_{i,G} = X_{r_1,G} + F\left(X_{r_{21},G} - X_{r_3,G} + X_{r_4,G} - X_{r_5,G}\right). \tag{24}$$

The selection of the base vector $X_{r_1,G}$ for mutation is also essential considering the mutant vector. Thus there exist different approaches for selecting it. (Mezura-Montes et al. 2006) In the following, the most typical methods are presented.

4.4.1 Rand

The base vector for mutation can be selected randomly. Then each chromosome has equal probability to be chosen. The problem with purely random selection is that same chromosomes can be selected multiple times. Thus a better random approach is to use universal stochastic sampling where multiple chromosomes are selected at a same time. A method where each chromosome is selected once is permutation selection where the indices of chromosomes are scrambled and the base vectors selected according to the new order. An even simpler approach is to select a random starting point for the selection. When random mutation is applied, the mutant vector is given by (19). (Price et al. 2005)

4.4.2 Best

The best chromosome ($X_{best,G}$) can be also used as the base vector for mutation. It increases the rate of convergence but may, however, lead to premature convergence (Price 1996). In such a case, the found solution may be poor. Using the best chromosome as a base vector is useful when the global optimum is easy to find (Storn and Price 1995). The mutant vector when the best chromosome is used as the base vector is

$$V_{i,G} = X_{best,G} + F(X_{r_2,G} - X_{r_3,G}). \tag{25}$$

4.4.3 Rand-to-best

Rand-to-best mutation is a combination of the previous two methods for selecting the base vector for mutation. The base vector is neither a random nor the best chromosome of the population but a combination of these. The base vector in rand-to-best mutation is given by (Storn 1996)

$$V_{i,G} = X_{r_1,G} + \lambda_w\left(X_{best,G} - X_{r_1,G}\right) + F\left(X_{r_2,G} - X_{r_3,G}\right). \tag{26}$$

Above, λ_w is a weighting coefficient ranging from 0 to 1 determining how close the base vector is to the best chromosome. With high values of λ_w the mutant vector is close to the best chromosome and with small values the mutant vector is close to the random chromosome. To simplify the method, λ_w may be defined to equal F (Storn 1996).

4.4.4 Current

If the base vector for mutation and the target vectors are the same, the search becomes isolated. Each chromosome performs its own search without changing genetic material with others. This may be advantageous if it is known that the objective function has multiple global optima. The probability to find as many optimum as possible increases when the chromosomes search for the optimum independently. The mutant vector when current mutation is applied is given by (Price et al. 2005)

$$V_{i,G} = X_{i,G} + F(X_{r_2,G} - X_{r_3,G}). \tag{27}$$

4.4.5 Current-to-best and current-to-rand

In current-to-best and current-to-rand methods, the base vector is somewhere between two chromosomes similarly as already presented with the rand-to-best method. In current-to-best method, the base vector is located between the target and the best chromosomes. The mutant vector is (Zaharie 2009)

$$V_{i,G} = X_{i,G} + \lambda_w\left(X_{best,G} - X_{i,G}\right) + F\left(X_{r_2,G} - X_{r_3,G}\right). \tag{28}$$

The base vector is a combination of the target and random chromosomes in current-to-rand mutation. The mutant vector is thus (Zaharie 2009)

$$V_{i,G} = X_{i,G} + \lambda_w \left(X_{r_1,G} - X_{i,G} \right) + F \left(X_{r_2,G} - X_{r_3,G} \right). \tag{29}$$

4.4.6 Trigonometric mutation

The mutation operations presented above do not take into account the fitness values of the chromosomes and thus the step direction is random. This may lead to increased computational time. Thus trigonometric mutation has been proposed in Fan and Lampinen (2003). It utilizes three chromosomes selected randomly from the population. These chromosomes form a triangle whose centre point is used as the base vector for mutation. To the base vector, three weighted difference vectors are added according to Fan and Lampinen (2003)

$$\begin{aligned} V_{i,G} = &(X_{r_1,G} + X_{r_2,G} + X_{r_3,G}) / 3 + (p_2 - p_1)(X_{r_1,G} - X_{r_2,G}) \\ &+ (p_3 - p_2)(X_{r_2,G} - X_{r_3,G}) + (p_1 - p_3)(X_{r_3,G} - X_{r_1,G}). \end{aligned} \tag{30}$$

Above,

$$p_1 = \left| J\left(X_{r_1,G} \right) \right| / p', \tag{31}$$

$$p_2 = \left| J\left(X_{r_2,G} \right) \right| / p', \tag{32}$$

$$p_3 = \left| J\left(X_{r_3,G} \right) \right| / p' \text{ and} \tag{33}$$

$$p' = \left| J\left(X_{r_1,G} \right) \right| + \left| J\left(X_{r_2,G} \right) \right| + \left| J\left(X_{r_3,G} \right) \right|. \tag{34}$$

Fan and Lampinen (2003) also proposed an algorithm that uses the trigonometric mutation with the probability p_{mt} and otherwise the traditional rand mutation. According to their results the modified version of differential evolution converged significantly faster than the original algorithm.

4.4.7 Degeneration of mutation

When selecting chromosomes for mutation, it is possible to select the same chromosome multiple times. This decreases the efficiency of mutation. The possible schemes for multiple selections are: $r_2 = r_3$, $r_2 = r_1$ or $r_3 = r_1$, $r_1 = i$ and $r_2 = i$ or $r_3 = i$. (Price et al. 2005)

If $r_2 = r_3$, no mutation occurs because the difference vector becomes a zero vector. Hence, the mutant vector is the selected base vector for mutation. If no restrictions exist, this occurs about once in a generation. Thus the probability for $r_2 = r_3$ is $1/NP$. If $r_2 = r_1$ or $r_3 = r_1$, mutation becomes arithmetic crossing presented in Section 3.3.2. Both cases occur about once a generation and thus the probability for them is $1/NP$. The possible mutant vectors are (Price et al. 2005)

$$V_{i,G} = X_{r_1,G} + F\left(X_{r_1,G} - X_{r_3,G}\right), r_2 = r_1 \text{ and} \quad (35)$$

$$V_{i,G} = X_{r_1,G} + F\left(X_{r_2,G} - X_{r_1,G}\right), r_3 = r_1 . \quad (36)$$

If $r_1 = i$, the influence of crossing is decreased and only mutation occurs at certain elements of the target vector. That is because the base vector for mutation and the target vector are the same. Thus the deviation between the trial and target vectors depends on the crossing coefficient *CR*. The last case, where $r_2 = i$ or $r_3 = i$, is actually a normal case because two separate vectors are used for generating the difference vector. (Price et al. 2005)

Avoiding the cases given above is rather easy. The condition $i \neq r_1$ can be satisfied with a simple loop where the random selection of r_1 is repeated as many times as needed. Similar loops can also be used to force $i \neq r_1 \neq r_2 \neq r_3$. The pseudo code for the loops is given below. (Storn et al. 2005)

```
do
{
r1 = floor(randi(0,1)*NP)
} while (r1 == i);

do
{
r2 = floor(randi(0,1)*NP)
} while (r1 == i || r2 == r1); // "||" denotes "if"

do
{
r3 = floor(randi(0,1)*NP)
} while (r1 == i || r3 == r1 || r3 == r2);
```

4.5 Crossover

After mutation, crossover is applied in differential evolution. In crossover, a trial vector is generated by combining the mutant and the target vectors. The used crossover method and the crossing coefficient CR both have influence on how close to the mutant vector the trial vector is. The closer the trial vector is to the mutant vector the bigger step size is applied and the algorithm proceeds faster. Typical crossover methods are binomial and exponential crossover but also arithmetic crossing is sometimes used. (Zaharie 2009)

4.5.1 Binomial crossover

In binomial crossover, the elements are selected to the trial vector from the mutant vector with the probability *CR* and otherwise they are taken from the target vector. The selection is made independently for each element. Because it is desired that the trial vector is not a duplicate of the target vector, one element is forced to be taken from the mutant vector. The trial vector according to the binomial crossing is (Storn and Price 1997)

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, if\ \ randb(j) \le CR\ or\ \ j = rnbr(i) \\ x_{i,j,G}, if\ \ randb(j) > CR\ or\ \ j \ne rnbr(i) \end{cases}. \tag{37}$$

Above, *randb*(*j*) is a uniformly distributed random number between 0 and 1, *rnbr*(*i*) is a random integer between 1 and *D*, where *D* is the number of optimized parameters and thus the length of one chromosome. Because one of the elements is forced to betaken from the mutant vector, the probability that a parameter is taken from the mutant vector does not equal *CR* ($p_m \ne CR$). The probability p_m depends on the population size. When crossing, the probability p_m for *D*-1 elements is *CR* and for the *rnbr*(*i*):th element it is 1. Thus for one parameter the probability is (Zaharie 2009)

$$p_m = CR\left(1 - \frac{1}{D}\right) + \frac{1}{D} = \frac{CR(D-1)+1}{D}. \tag{38}$$

The expected number of parameters taken from the mutant vector is given by (Zaharie 2009)

$$E(L) = NP \cdot p_m = (NP-1)CR + 1. \tag{39}$$

4.5.2 Exponential crossover

Exponential crossover is similar to the one-point- and two-point crossover operators in genetic algorithms presented earlier in Section 3.3.1. L elements starting from a random point are taken from the mutant vector and the rest of the trial vector is taken from the target vector. Exponential crossover is presented by (Storn and Price 1995)

$$u_{i,j,G} = \begin{cases} v_{i,j,G}, if\ \ j = \langle n \rangle_D, \langle n+1 \rangle_D, ..., \langle n+L-1 \rangle_D \\ x_{i,j,G}, otherwise \end{cases}. \tag{40}$$

Above, *n* is a random integer between 1 and *D* and $\langle n \rangle_D$ is the remainder of the division *n*/*D*. The elements are taken from the mutant vector as long as a generated random number is lower than *CR*. The pseudo code for defining *L* is given below. (Storn and Price 1995)

```
L = 0;
do
{
L = L + 1;
} while (rand() < CR & L < D);
```

The probability of taking an element from the mutant vector and also the expectation for the overall number of elements taken from the mutant vector can be calculated. They are given, respectively, by (Zaharie2009)

$$p_m = \frac{1 - CR^D}{D(1 - CR)} \text{ and} \tag{41}$$

$$E(L) = \frac{1 - CR^{NP}}{(1 - CR)}. \tag{42}$$

Compared to binomial, exponential crossover requires a lot higher *CR* to obtain the same expectation *E*(*L*). Practically, only in cases where *CR* is close to 1, the majority of the elements in trial vector are taken from the mutant vector. Thus if the problem is such that mutation is critical in finding the optimum, binomial crossing is to be used. With exponential crossing, defining an appropriate *CR* is also harder because the correlation between *CR* and p_m is nonlinear while in binomial crossing it is linear. Thus the majority of the applications nowadays uses binomial crossing. (Zaharie 2009)

4.5.3 Arithmetic crossover

Two crossing methods presented above took the elements from either the mutant or the target vector. It is also possible to arithmetically combine the information in these two vectors. The trial vector according to arithmetic crossing is (Zaharie 2008)

$$u_{i,j,G} = (1 - q)x_{i,j,G} + qv_{i,j,G}, \tag{43}$$

where *q* is the weighting coefficient regulating the balance between the mutant and the target vectors. Arithmetic crossing is practically the same in differential evolution and genetic algorithms.

4.6 Maintaining the population inside the search space

If the population is allowed to drift outside the search space during the differential evolution run, it may take more time to find the optimal solution due to the increasing variance of the population. If needed, this can be avoided by checking that all the parameters are within the search space. If a parameter is outside the search space, a random new value can be taken according to (Rönkkönen et al. 2005)

$$u_{i,j,G} = rand_j[0,1](u_j - l_j) + l_j. \tag{44}$$

Also, the value of the parameter can be bounced back to the feasible range. The feasible value is then (Rönkkönen et al. 2005)

$$u_{i,j,G} = \begin{cases} 2l_j - u_{i,j,G}, if\ u_{i,j,G} < l_j \\ 2u_j - u_{i,j,G}, if\ u_{i,j,G} > u_j \end{cases}. \tag{45}$$

4.7 Selection

In evolutionary algorithms, selection can be applied in two ways. The parents are selected for crossover in genetic algorithms while in differential evolution selection is applied to distinguish which chromosomes are placed into the new population. (Price et al. 2005) Selection in differential evolution is a tournament with two candidates: the trial and the target vectors. The selection for a minimization problem is mathematically presented in (20).

When the selection is made between the target vector and its offspring, losing genetic material is avoided. Also, when only one of the chromosomes including similar genetic material survives, its genetic material does not become dominant in the population very quickly. (Bergey and Ragsdale 2005)

4.8 Selection of the tuning parameters

There are only a few tuning parameters in differential evolution and thus those should be defined carefully. Differential evolution is sensitive especially to the mutation coefficient F and the crossing coefficient CR (Tvrdík 2009). The appropriate values of the parameters depend on the problem. The influence of all three parameters (F, CR and NP) is similar. Small values increase the rate of convergence but may lead to premature convergence to a local optimum. (Kukkonen and Lampinen 2005)

Some suggestions for the tuning parameters can be found in the literature. Price et al. (2005) suggest that a good initial guess for separable functions are $CR = 0.2$ and $F = 0.5$. However, if the parameters to be solved depend on each other, Price et al. (2005) state that efficient optimization is obtained with $CR = 0.9$ and $F \geq 0.8$. Some guidance for the population size is also found in Price et al. (2005). They suggest that $NP = 5 \cdot D \cdot CR$ is an appropriate lower limit but even $NP \geq 10 \cdot D$ may be required in complex problems.

To guarantee that the search is efficient and mutation produces big enough step sizes, the variance of the population compared to the state of the search should be high enough. The expectation of the population variance after mutation and crossover is described by (Zaharie 2002)

$$E\left(Var(U_G)\right) = \left(2F^2 p_m - \frac{2p_m}{NP} + \frac{p_m^2}{NP} + 1\right) Var(X_G) . \tag{46}$$

Because selection favours the solutions near the optimum, it decreases the variance of the population. Thus it is desired that mutation and crossover slightly increases the variance. On the other hand, too high variance may decrease the rate of convergence significantly. Thus the parameters for differential evolution are to be defined so that (Zaharie 2002)

$$c = \sqrt{2F^2 p_m - \frac{2p_m}{NP} + \frac{p_m^2}{NP} + 1} \ , \tag{47}$$

where $1 \le c \le 1.5$.

5 APPLIED ALGORITHMS

Different variations of genetic algorithms and differential evolution are used for identifying the fuel cell model presented in Section 2. With genetic algorithms, different coding strategies are studied independently. Also, different crossover methods are used. The number of difference vectors, the selection of the base chromosome for mutation and different crossover operators are studied with differential evolution.

5.1 Used data sets and the objective function

For identifying the model parameters, experimental data is available in Mo et al. (2006). That data is obtained from a 250 W PEM fuel cell with properties given in Table 2. The used data includes four sets of current and voltage measurements. Each set is obtained in different operating conditions and includes 15 data points. The operating conditions are given in Table 3. The data sets are visualized in Figure 5. The same data sets have been earlier used in Mo et al. (2006) and Ohenoja and Leiviskä (2010). Two of the data sets (1 and 2) are used for the model identification while the remaining two sets (3 and 4) are used for model validation as has been earlier done in Mo et al. (2006) and Ohenoja and Leiviskä (2010). The objective function is the sum of the squared error of prediction (SSEP). For one data set, the objective function is given by

$$J = \sum_{i=1}^{N} \left(y - \hat{y}\right)^2 , \tag{48}$$

where N is the number of data points in the set. The overall objective function is obtained by summing the SSEPs of the training data sets.

Table 2. The properties of the studied fuel cell. (Mo et al. 2006)

Property	Symbol	Value
The number of cells	n	24
Effective area [cm^2]	A	27
Membrane thickness [μm]	l_m	127
Maximum current density [mA/cm^2]	I_{max}	860
Rated power [W]	V_{cell}	250
Relative humidity in anode	RH_a	1
Relative humidity in cathode	RH_c	1

Table 3. The operating conditions in different data sets.

	Data set 1	Data set 2	Data set 3	Data set 4
p_a [bar]	3.0	1.0	2.5	1.5
p_c [bar]	5.0	1.0	3.0	1.5
T [°C]	80	70	70	70

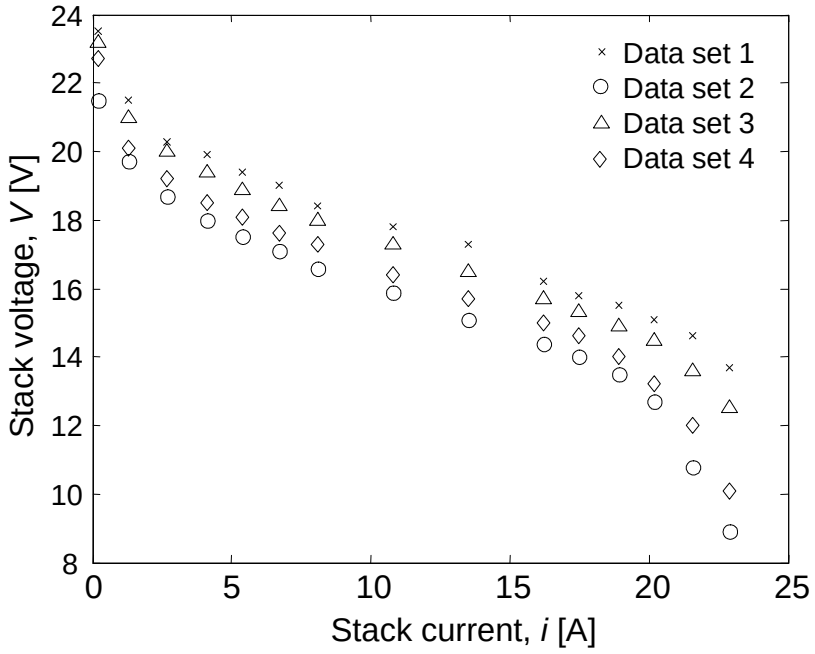

Figure 5. The used data sets.

5.2 Identified model

The identified parameters are the seven parameters in the PEM fuel cell model described in Section 2. The parameters are the empirical coefficients ξ_1, ξ_2, ξ_3 and ξ_4, b and λ and the overall resistance R_c. The model is simplified by assuming the maximum current density (I_{max}) and the membrane resistance (R_m) constant in the operating conditions given in Table 3 even though they are known to depend on the conditions (Mo et al. 2006). Through the assumptions, a small prediction error is introduced. Further simplifications are made by assuming ξ_2 constant even though Mann et al. (2000) state that ξ_2 should be considered as a function of the electrode effective area and the dissolved hydrogen concentration. The functional form of ξ_2 is (Mann et al. (2000)

$$\xi_2 = 0.00286 + 0.0002 \cdot ln(A) + 4.3 \cdot 10^{-5} \cdot ln\left(c^*_{H_2}\right). \tag{49}$$

The same assumption is made in Mo et al. (2006) and Ohenoja and Leiviskä (2010). The search space for the parameters is given in Table 4.

Table 4. The search space for the parameters. Mo et al. (2006)

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	b	λ	R_c [$\cdot 10^{-3}$]
Lower limit	-1	0	-2	7	0.016	9	0
Upper limit	0	5	-1	13	0.5	23	1

5.3 Binary coded genetic algorithms

The binary coded genetic algorithms used in this study differ in the used crossover operator. The first genetic approach uses one-point crossover while the second uses uniform crossover. The crossover probability is defined to regulate the reproduction rate. Tournament selection with the number of candidates set to 2 as suggested in Michalewicz (1996) is used. In mutation, each bit is browsed and subjected to a mutation if the random number is smaller than the defined mutation probability. After the new population is generated through crossing and mutation, elitism is applied in order to prevent the very best solution from disappearing from the population. In this study, the worst chromosome of the new population is replaced with the best chromosome of the previous population.

5.4 Real-value coded genetic algorithms

Arithmetic and heuristic crossover methods are used with real-value coded genetic algorithms. The mutation operator is similar to binary coded algorithms but the mutated value is taken randomly from the feasible range of the corresponding parameter. Elitism is applied as described in the previous subsection.

5.5 Differential evolution

Five different DE algorithms are studied in order to gain knowledge about different versions of the algorithm. The studied versions are

- DE/rand/1/bin,
- DE/rand/2/bin,
- DE/best/1/bin,
- DE/rand-to-best/1/bin and
- DE/rand/1/exp.

Two first ones are used to determine if it is favourable to use more than one difference vectors. The effects of different base vectors for mutation are studied with DE/rand/1/bin, DE/best/1/bin and DE/rand-to-best/1/bin. Finally, binomial and exponential crossover operators are compared with DE/rand/1/bin and DE/rand/1/exp.

5.6 Defining appropriate population size

There are multiple parameters regulating the evolution of the population. The population size is one of them and is essential because it should be

- great enough so that the initial population has enough diversity (i.e. the initial population covers the whole search space) and
- as small as possible to decrease the computational load.

In this study, entropy is used as the measure of diversity. The higher the entropy the more diverse the population is. The entropy of a random variable is given by

$$H(X) = -\sum p(x) log\ p(x), \tag{50}$$

where $p(x)$ is the probability mass function of the random variable X. The probability mass function satisfies

$$\sum p(x) = 1. \tag{51}$$

The base of the logarithm in (51) is 10 in this study. For the case that $p(x)$ contains zero probability components, it is defined that

$$0 log\ 0 = 0. \tag{52}$$

The entropy is at its maximum when the random variable is uniformly distributed. Typically and also in this study, the initial population is taken from the uniform distribution. When the initial population is too small it does not follow the uniform distribution leading to lower entropy. However, with increasing population size the initial population follows the uniform distribution better and the entropy is closer to the maximum. In this study, the entropy is calculated for each parameter independently and those entropies are then summed. Figure 6 shows the entropy as the function of the population size. For each population size, 10 different initial populations are created and their entropies are calculated. The average of those is then plotted in Figure 6. With 7 parameters and using the 10 based logarithm, the maximum entropy is 7. Table 5 shows entropies with selected population sizes. From Figure 6 and Table 5, it is clearly seen that the initial steep rise of the entropy reaches a plateau somewhere around the population size of 160 where about 99 % of the maximum entropy is reached. With the population size of 100, almost 98 % of the maximum entropy is reached while almost 96 % is reached with the population size of 50. In this study, it is assumed that reaching 95 % of the maximum entropy is enough and thus the population size of 50 is used with all algorithms.

Table 5. Entropies with some population sizes.

Population size	10	20	50	100	200
Entropy	5.31	6.24	6.70	6.85	6.93

5.7 The tuning parameters for genetic algorithms

Finding appropriate tuning parameters for the algorithms is sometimes a challenging task. In this study, entropy is used as a measure of diversity when defining the appropriate population size as described in the previous subsection and also as a measure of convergence. To find appropriate tuning parameters, entropy of the last population is used. If that entropy is zero, the population has converged to a single solution. This is not

desired in genetic algorithms because of mutation that constantly generates new possible solutions. Thus having zero entropy for the last population indicates that mutation is not efficient and there is a chance that only a local optimum has been found. Figure 7 shows the entropy of the last population as a function of the mutation probability. Binary coded genetic algorithm with one-point crossover and crossover probability of 0.9 is used. The algorithm is run 10 times with each mutation probability. The figure shows that the population is almost uniformly distributed if the mutation probability is higher than 0.1. With lower mutation probabilities, some convergence is achieved (entropy of the initial population is about 6.7 as shown in Figure 6 and Table 5). Also it is noticed from Figure 7 that mutation probabilities 0.001 and 0.0316 (= $10^{-2.5}$) leads to the situation where the population has converged almost to a single solution. Thus it can be said that the appropriate mutation probability is somewhere between 0.01 and 0.1. In this study, it is assumed that the appropriate entropy of the last population is about 1 and thus, in this case, the mutation probability p_m is set to 0.01. The same kind of reasoning is carried out with all the studied genetic algorithms. Table 6 presents the used mutation probabilities for genetic algorithms.

Table 6. The used mutation probabilities for genetic algorithms.

Coding	Binary	Binary	Real-valued	Real-valued
Crossover	One-point	Uniform	Arithmetic	Heuristic
p_c	0.95	0.85	0.85	0.85
p_m	0.01	0.01	0.0316	0.0316

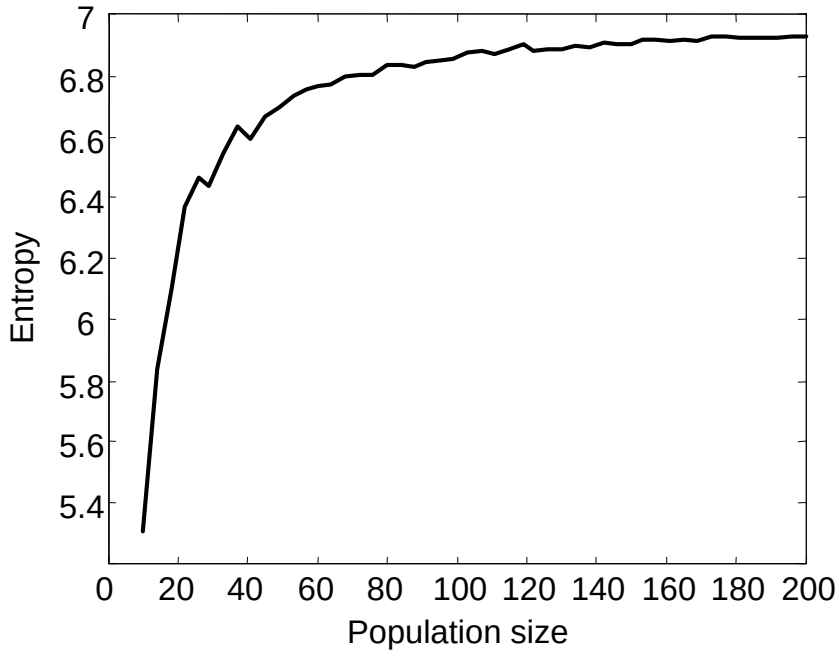

Figure 6. Entropy as the function of the population size.

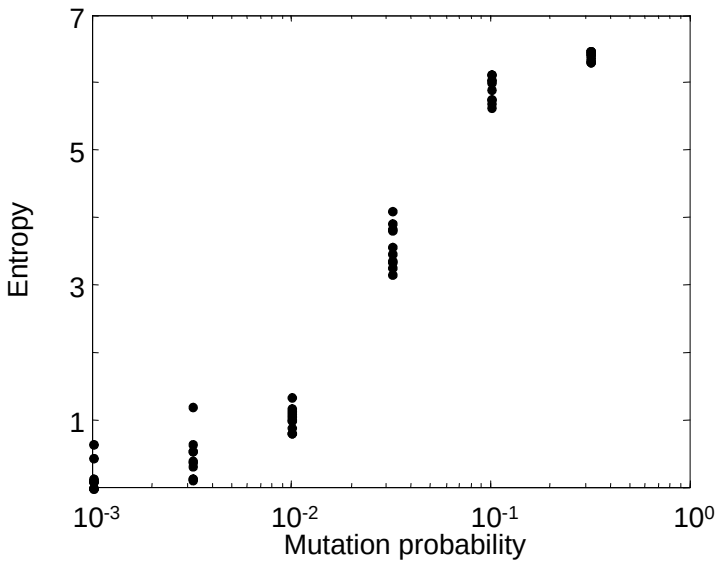

Figure 7. The entropy of the last population as a function of the mutation probability, p_m when using binary coded genetic algorithms with one-point crossover.

Before defining the appropriate crossover probability, the mutation probabilities are defined and kept fixed for all the genetic algorithms (Table 6). The crossover probability is defined by studying the best solution found by the algorithm and also the entropy of the last population. The genetic search is repeated 10 times for each studied probability and the mean and standard deviation of the best solutions and entropies are calculated. From Table 7, it can be seen that best solutions are obtained with p_c = 0.95 and thus that is chosen as the crossover probability. However, when studying the entropies it is seen that p_c = 0.95 leads to entropies higher than 1 and that the deviation of the entropies is quite high. This indicates that the evolution of the population is not very stable but the outcome varies quite a lot. Thus it would be better if 0.9 is selected as the crossover probability. In this case, however, the solutions with p_c = 0.95 are so much better that it is chosen. When defining crossover probabilities for other genetic algorithms the same kind of reasoning as described above is carried out. Table 6 presents the used crossover probabilities. It should be noticed that with other genetic algorithms the mean and standard deviation of both the best solutions and entropies indicated that the selected probabilities are better than the others.

Table 7. The means and standard deviations of the best solution and entropy of the last population with the genetic algorithm using binary coding and one-point crossover.

	Best solution		Entropy	
p_c	Mean	St. dev.	Mean	St. dev.
0.75	6.30	0.95	0.94	0.33
0.80	5.99	0.71	1.21	0.28
0.85	5.91	0.76	1.24	0.28
0.90	5.96	0.90	0.95	0.18
0.95	5.65	0.42	1.40	0.47

5.8 The tuning parameters for differential evolution

The tuning parameters for differential evolution are the mutation and crossover coefficients *F* and *CR*, respectively. The coefficients should be such that population diversity compared to the state of the optimization is appropriate. As mentioned in Section 4.8, population variance allows mutation to produce big enough step sizes for convergence but too high variance may decrease the convergence rate. In this study, appropriate tuning parameters are defined by utilizing (47) which gives the limits for *F* and *CR* so that the appropriate variance remains in the population. The tuning parameters are defined through the following steps.

1. Select the levels for *CR* and *c* to be tested.
2. Select values for *CR* and c systematically.
3. Calculate the mutation probability p_m with (38) or (41) depending on the crossover operator.
4. Use (47) to solve F for each pair of *CR* and *c*.
5. Evaluate results.

It is obvious that experimental designs can be utilized in step 2 of the procedure. Using experimental designs also decreases the number of runs needed for finding the appropriate tuning parameters and also helps in evaluating results. Even though experimental designs can be useful, they are not used in this study. The studied values for *CR* are [0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9] and for *c* they are [1.1 1.25 1.4]. For each pair of CR and c, the optimization is repeated 10 times. Thus the procedure used in this study requires 270 optimizations for each of the used types of the DE algorithm. Figure 8 shows the results obtained with DE/rand/1/bin. The figure shows that the best results are obtained when *CR* = 0.9 and *c* = 1.1. Thus *CR* = 0.9 is used and *F* is calculated from (47) with *c* = 1.1. It should be noticed, however, that Figure 8 indicates that even higher CR values and lower c values should be tested. In this study, this is not done but the tuning coefficients are selected as given in Table 8. Table 8 also shows the studied DE algorithms and their performance indices with the tuning coefficients defined as described above.

Table 8. The tuning coefficients of the studied DE algorithms.

	c	*CR*	*F*	mean	st. dev.	min
DE/rand/1/bin	1.1	0.35	0.9	5.14	0.04	5.09
DE/rand/2/bin	1.1	0.35	0.9	5.18	0.04	5.11
DE/best/1/bin	1.1	0.39	0.7	5.10	0.05	5.07
DE/rand-to-best/1/bin	1.25	0.56	0.9	5.08	0.03	5.07
DE/rand/1/exp	1.1	0.35	0.9	5.14	0.05	5.08

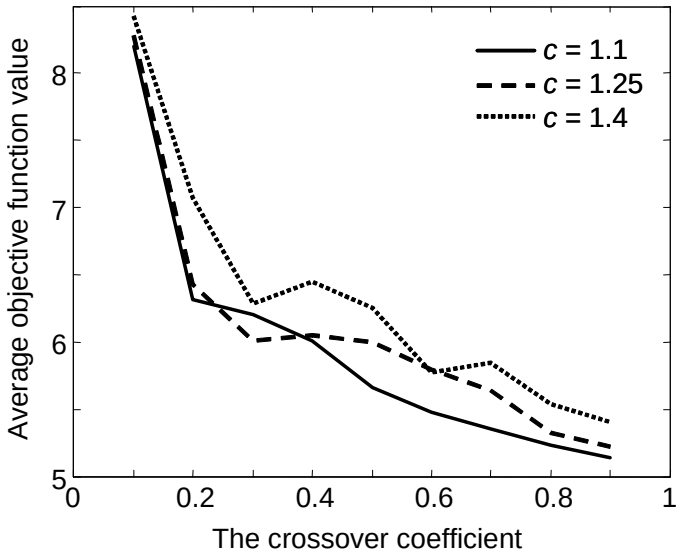

Figure 8. The average objective function values as a function of *CR* when using DE/rand/1/bin.

6 RESULTS AND DISCUSSION

The optimizations are repeated 500 times for each studied version of the algorithms. From the repetitions, statistical information about the performance of the algorithms is obtained. Thus the histograms of the best solutions are plotted and studied. Statistical values are calculated for the best parameter values to gain information about the convergence and the significance of the parameters. Among the evaluation of the best parameter values, the corresponding model accuracy is studied. Thus the average, standard deviation and minimum of the objective function values are calculated. The accuracy of the best models is evaluated through the objective function given in (48). The value of the objective function is calculated for all the data sets given in Figure 5. Especially the sets dedicated for validation are examined carefully. Among the SSEP value, the average and standard deviation of the prediction error are calculated.

6.1 Genetic algorithms

The statistical values of the best models in 500 repetitions of the genetic algorithms are given in Table 9. It can be noticed that the algorithm utilizing real-valued coding and arithmetic crossover gives the worst results. The minimum SSEP value is reasonable but the repeatability of the algorithm is poor as can be seen from the high average and standard deviation. About the rest of the genetic algorithm versions, binary coded with uniform crossover and real-value coded with heuristic crossover give the best and almost equal results. The minimum SSEP is obtained with the latter one while the prior one seems to perform better on average. The significance of the parameters can be evaluated from Table 10 which shows the entropies of the parameters in 500 repetitions. Entropy is calculated with (50)-(52) as described in subsection 5.6. Table 10 shows that the best solutions are spread widely to the search space despite the parameter b which has almost zero entropy and thus its values are almost the same despite the optimization run and the algorithm. This indicates that the search space includes a lot of local optima which are very close to the global optimum and leads to an almost equal prediction accuracy. Tables 11 and 12 present the best parameter values of the genetic algorithms using binary coding with uniform crossover and real-valued coding with heuristic crossover. It can be seen that only ξ_3, b and λ have almost equal values but others vary greatly. This also indicates that two different optima are found but that the objective function value of these optima are very close to each other as can be seen from Table 9. The difference can be seen from Table 13 where the prediction accuracy of the best models is studied in more detail. From the table, it can be seen that the solution found by the binary coded genetic algorithm with uniform crossover follows data set 2 more closely than the solution found by the real-valued genetic algorithm with heuristic crossover. As a consequence of this, the latter model works better for the validation data sets (data sets 3 and 4) and thus is considered better.

The results obtained with the genetic algorithms show that with different versions of the algorithms similar results can be obtained. In this case, the objective function includes a lot of local optima and thus finding the global optimum is hard which can be seen from

Tables 10-12. The real-value coded algorithm using heuristic crossover found the best solution in this study especially when the prediction accuracy for the validation data sets are emphasized. The real-coded algorithm is also computationally less expensive than the binary coded algorithm and thus should be favoured in this case (Chang 2007). The best solutions found by the algorithms are influenced by the initial population and thus using bigger population size could have led to more consistent results. Based on Figure 6, it would have been justified to use the population size of 100 or even a bit bigger. However, high entropy was obtained already with the population size of 50 and thus it was used. The limited population size also leads to the situation where the efficiency of the used operators has more influence on the results and thus makes the comparison of the algorithms more reasonable.

Table 9. Statistical values of the SSEPs in 500 repetitions of the GAs.

Coding	Crossover	Minimum	Average	St. dev
binary	one-point	5.093	6.022	1.041
binary	uniform	5.084	5.806	0.659
real-valued	arithmetic	5.276	11.656	7.515
real-valued	heuristic	5.074	5.832	0.830

Table 10. Entropies of the best parameter values in 500 repetitions.

Coding	Crossover	ξ_1	ξ_2	ξ_3	ξ_4	b	λ	R_c
binary	one-point	0.95	0.73	0.56	0.93	0.01	0.91	0.88
binary	uniform	0.95	0.73	0.53	0.92	0.00	0.90	0.92
real-valued	arithmetic	0.86	0.66	0.71	0.86	0.10	0.74	0.78
real-valued	heuristic	0.91	0.77	0.56	0.93	0.00	0.92	0.82

Table 11. The results with the binary coded GA with uniform crossover.

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	b	λ	R_c [$\cdot 10^{-3}$]
Best	-0.051	1.002	-1.174	11.481	0.033	11.915	0.022
Mean	-0.519	2.112	-1.226	9.922	0.038	15.814	0.392
Mode	-0.495	1.999	-1.257	10.197	0.046	16.062	0.132

Table 12. The results with the real-value coded GA with heuristic crossover.

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	b	λ	R_c [$\cdot 10^{-3}$]
Best	-0.858	2.746	-1.176	7.659	0.033	12.055	0.003
Mean	-0.466	1.900	-1.183	9.531	0.035	14.540	0.365
Mode	-1.000	0.209	-1.613	7.000	0.016	9.000	0.000

Table 13. Prediction accuracy of the best models using the binary coded GA with uniform crossover and the real-value coded GA with heuristic crossover (binary with 1-point / real-valued with heuristic).

	Data set 1	Data set 2	Data set 3	Data set 4
Error mean	-0.01 / 0.00	0.02 / -0.00	-0.25 / 0.15	-0.10 / 0.04
Error st. dev.	0.39 / 0.38	0.46 / 0.47	0.25 / 0.24	0.30 / 0.30
SSEP	2.15 / 2.03	2.93 / 3.04	1.87 / 1.15	1.40 / 1.32

6.2 Differential evolution

The results with DE algorithms are given in Table 14. It can be noticed that no big differences exist. However, in the following the results are studied in more detail to gain knowledge about the used operators and their significance.

Table 14. Statistical values of the SSEPs in 500 repetitions of the DEs.

Algorithm	minimum	average	st. dev
DE/rand/1/bin	5.070	5.152	0.075
DE/rand/2/bin	5.076	5.165	0.044
DE/best/1/bin	5.066	5.117	0.070
DE/rand-to-best/1/bin	5.066	5.080	0.033
DE/rand/1/exp	5.069	5.146	0.064

6.2.1 Influence of the number of difference vectors

The effect of the number of difference vectors can be noticed when comparing DE/rand/1/bin and DE/rand/2/bin algorithms. The best solutions found by both DE versions are presented in Table 15. It can be noticed again that even though the solutions are almost equal the parameter vectors differ. The entropies of the parameters in 500 repetitions of the algorithms are presented in Table 16 showing that only parameters ξ_3 and especially b has low entropy. Low entropy indicates that the parameter value remains almost the same in all repetitions. Also λ has a little bit lower entropy than the rest of the parameters. Above mentioned three parameters also have almost the same values in the best solutions with DE/rand/1/bin and DE/rand/2/bin as seen from Table 15. Table 17 presents the prediction accuracy of the best models. It is seen that, DE/rand/1/bin has lower SSEP for data set 1 while DE/rand/2/bin has the lower SSEP for the rest of the data sets. Thus DE/rand/2/bin has performed better in this case. From the results, it can be concluded that no significant improvement has been achieved by using 2 difference vectors instead of 1. It seems, however, then the convergence rate is a bit higher with 2 difference vectors.

Table 15. The best parameter values found by DE/rand/1/bin and DE/rand/2/bin.

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	b	λ	R_c [$\cdot 10^{-3}$]
DE/rand/1/bin	-0.117	1.108	-1.148	10.936	0.033	11.858	0.002
DE/rand/2/bin	-0.419	1.781	-1.159	9.621	0.034	12.113	0.020

Table 16. The entropies of the parameters in 500 repetitions with DE/rand/1/bin and DE/rand/2/bin.

	ξ_1	ξ_2	ξ_3	ξ_4	B	λ	R_c
DE/rand/1/bin	0.93	0.63	0.16	0.84	0	0.51	0.78
DE/rand/2/bin	0.92	0.61	0.20	0.82	0	0.42	0.79

Table 17. Prediction accuracy of the best models using DE/rand/1/bin and DE/rand/2/bin (DE/rand/1/bin / DE/rand/2/bin).

	Data set 1	Data set 2	Data set 3	Data set 4
Error mean	0.00 / -0.01	-0.01 / -0.02	-0.22 / -0.08	-0.11 / -0.06
Error st. dev.	0.39 / 0.40	0.46 / 0.45	0.25 / 0.26	0.31 / 0.30
SSEP	2.10 / 2.19	2.97 / 2.89	1.66 / 1.04	1.51 / 1.31

6.2.2 Influence of the base vector for mutation

The influence of the base vector for mutation is evaluated by comparing DE/rand/1/bin, DE/best/1/bin and DE/rand-to-best/1/bin algorithms. Table 14 shows no differences with minimum SSEPs but indicates that DE/rand-to-best/1/bin performs best on average. The best parameter values and the entropies of the parameters with these algorithms are given in Table 18 and 19, respectively. Table 18 shows that the best solutions from the two latter algorithms are almost the same while DE/rand/1/bin gives a clearly different best solution. The entropies given in Table 19 show that with DE/rand-to-best/1/bin parameters ξ_3, b, λ and R_c have almost the same values in all 500 repetitions. The prediction accuracies of the best models in Table 20 also show that DE/rand-to-best/1/bin gives the best results for the validation data sets. These results indicate that it is advantageous to use rand-to-best strategy for selecting the base vector for mutation.

Table 18. The best parameter values found by DE/rand/1/bin and DE/rand/2/bin.

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	B	λ	R_c [$\cdot 10^{-3}$]
DE/rand/1/bin	-0.117	1.108	-1.148	10.936	0.033	11.858	0.002
DE/best/1/bin	-0.765	2.542	-1.157	8.076	0.033	11.887	0.000
DE/rand-to-best/1/bin	-0.700	2.400	-1.158	8.367	0.033	11.891	0.000

Table 19. The entropies of the parameters in 500 repetitions with DE/rand/1/bin and DE/rand/2/bin.

	ξ_1	ξ_2	ξ_3	ξ_4	B	λ	R_c
DE/rand/1/bin	0.93	0.63	0.16	0.87	0	0.51	0.78
DE/best/1/bin	0.95	0.68	0.06	0.89	0	0.46	0.66
DE/rand-to-best/1/bin	0.95	0.68	0.01	0.86	0	0.27	0.29

Table 20. Prediction accuracy of the best models using DE/rand/1/bin, DE/best/1/bin and DE/ran-to-best/1/bin (DE/rand/1/bin / DE/rand/2/bin / DE/ran-to-best/1/bin).

	Data set 1	Data set 2	Data set 3	Data set 4
Error mean	0.00 / 0.00/ 0.00	-0.01 / 0.00 / 0.00	-0.22 / 0.10/ 0.07	-0.11 / 0.02 / 0.01
Error st. dev.	0.39 / 0.39 / 0.39	0.46 / 0.45 / 0.46	0.25 / 0.26 / 0.26	0.31 / 0.30 / 0.30
SSEP	2.10 / 2.17 / 2.16	2.97 / 2.90 / 2.91	1.66 / 1.10 / 1.00	1.51 / 1.27 / 1,27

6.2.3 Influence of the crossover operator

The influence of the crossover operator can be evaluated by studying DE/rand/1/bin and DE/rand/1/exp algorithms. Tables 21 and 22 give the information about the performance of the algorithms and Table 23 shows the prediction accuracy of the best models. The best solutions given in Table 21 are in this case somewhat similar except the parameter ξ_1. This is probably just a coincident because earlier results and Table 22 indicate that only parameters ξ_3, b and λ remain almost the same throughout the 500 repetitions. The prediction accuracy with the validation data sets with DE/rand/1/exp is better than with DE/rand/1/bin.

Table 21. The best parameter values found by DE/rand/1/bin and DE/rand/1/exp.

	ξ_1	ξ_2 [$\cdot 10^{-3}$]	ξ_3 [$\cdot 10^{-4}$]	ξ_4 [$\cdot 10^{-5}$]	B	λ	R_c [$\cdot 10^{-3}$]
DE/rand/1/bin	-0.117	1.108	-1.148	10.936	0.033	11.858	0.002
DE/rand/1/exp	-0.191	1.277	-1.146	10.631	0.033	11.858	0.002

Table 22. The entropies of the parameters in 500 repetitions with DE/rand/1/bin and DE/rand/1/exp.

	ξ_1	ξ_2	ξ_3	ξ_4	B	λ	R_c
DE/rand/1/bin	0.93	0.63	0.16	0.84	0	0.51	0.78
DE/rand/1/exp	0.92	0.61	0.16	0.81	0.00	0.51	0.75

Table 23. Prediction accuracy of the best models using DE/rand/1/bin and DE/rand/1/exp (DE/rand/1/bin / DE/rand/1/exp).

	Data set 1	Data set 2	Data set 3	Data set 4
Error mean	0.00 / 0.00	-0.01 / -0.01	-0.22 / -0.19	-0.11 / -0.10
Error st. dev.	0.39 / 0.40	0.46 / 0.45	0.25 / 0.27	0.31 / 0.30
SSEP	2.10 / 2.24	2.97 / 2.83	1.66 / 1.52	1.51 / 1.39

6.2.4 Comparison of the DE algorithms

In Table 24, the information already given in Tables 15-23 is collected and refined. Table 24 shows the SSEP values for the training and validation data sets and the overall entropy. From the table, it is seen that all the algorithms are able to reach almost equal value for the training data SSEP. When the SSEP of the validation data set is investigated, it is seen that DE/rand-to-best/1/bin gives the best results. Also the overall entropy shows that the solutions found by DE/rand-to-best/1/bin are close to each other throughout the 500 repetitions. Thus it seems that DE/rand-to-best/1/bin is the most suitable algorithm for the studied problem.

Table 24. The performance of the algorithms.

	rand/1/bin	rand/2/bin	best/1/bin	rand-to-best/1/bin	rand/1/exp
SSEP train	5.07	5.08	5.07	5.07	5.07
SSEP valid	3.18	2.35	2.37	2.27	2.91
Entropy	3.85	3.76	3.69	3.06	3.76

6.3 Comparison of genetic algorithms and differential evolution

The comparison between genetic algorithms and differential evolution is carried out by comparing the real-coded GA with heuristic crossover and DE/rand-to-best/1/bin algorithms. The comparison is presented in Table 25 and shows that both algorithms reach the same value for the training data set but that the DE algorithm finds better solutions considering the validation data set. Also the solutions found by the DE algorithm are more consistent which can be noticed from the lower overall entropy.

Overall, all the DE algorithms have lower entropy values. This is probably due to the elitistic nature of the differential evolution which prevents the poorer chromosomes to be placed into the new population. Thus, it is believed that the DE algorithms converge to the optimum more efficiently than the GAs. In this case, this leads to better results but generally there is also a greater risk of premature convergence to a local optimum if the tuning parameters of the DE algorithm are poorly defined.

Table 25. The performance of the real-valued GA with heuristic crossover and DE/rand-to-best/1/bin.

	SSEP train	SSEP valid	Entropy
GA with heuristic cr.	5.07	2.48	4.92
DE/rand-to-best/1/bin	5.07	2.27	3.06

7 CONCLUSIONS

In this report, evolutionary algorithms (genetic algorithms and differential evolution) were studied and used for identifying the parameters of a fuel cell model. The fuel cell model was nonlinear having 7 parameters. Different versions of the above mentioned algorithms were used and compared. From genetic algorithms, binary coded algorithms using one-point and uniform crossover operators and real coded algorithms using arithmetic and heuristic crossover operators were used. Used differential evolution algorithms varied in the number of difference vectors, the selection of the base vector for mutation and the crossover operator. The studied DE algorithms were DE/rand/1/bin, DE/rand/2/bin, DE/best/1/bin, DE/rand-to-best/1/bin and DE/rand/1/exp.

The tuning parameters of the genetic algorithms were defined by investigating the entropy of the initial and final populations. An appropriate population size was defined based on the plot of the entropy of the initial population as a function of the population size. The mutation probability was then defined by systematically testing different probabilities (the crossover probability was set constant) and then investigating the entropy of the last population. The final tuning parameter, the crossover probability, was then defined through systematic testing.

The tuning parameters of the differential evolution algorithms were defined by using (47) that links the mutation and crossover coefficients (F and CR respectively) together. The tested values were set for c and CR while F was calculated from (47). The appropriate parameters were then selected based on the performance of the algorithms.

The results from the studies showed that the proposed procedures for defining the tuning parameters worked well. Also the results showed that both evolutionary algorithms were able to find a good solution for the problem. However, a closer comparison revealed that differential evolution performed better.

REFERENCES

Bergey P.K. and Ragsdale C. (2005) Modified differential evolution: a greedy random strategy for genetic recombination. Omega 33, 255-265.

Chang W.-D. (2007) Nonlinear system identification and control using a real-coded genetic algorithm. Applied Mathematical Modelling 31, 541-550.

Chipperdale A. (1997) Introduction to genetic algorithms. In: Zalzala A.M.S. and Fleming P.J. (eds.): Genetic algorithms in engineering systems. Stevenage, Herts, The Institution of Electrical Engineers. 1-45.

Corrêa J.M., Farret F.A., Canha L.N. and Simões M.G. (2004) An Electrochemical-Based Fuel Cell Model Suitable for Electrical Engineering Automation Approach. IEEE Transactions on Industrial Electronics 51, 1103- 1112.

Davis L (1991) Handbook of genetic algorithms. New York, Van Nostrand Reinhold. 385 s. ISBN 0-442-00173-8.

Deep K. and Thakur M. (2007a) A new crossover operator for real coded genetic algorithms. Applied Mathematics and Computation 188, 895-911.

Deep K. and Thakur M. (2007b) A new mutation operator for real coded genetic algorithms. Applied Mathematics and Computation 193, 211-230.

Fan H.-Y. and Lampinen J. (2003) A Trigonometric Mutation Operation to Differential Evolution. Journal of Global Optimization 27, 105-129.

Feoktistov V. and Janaqi S. (2004) Generalization of the strategies in differential evolution. Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), 165a.

Fogel D.B. (2000) Evolutionary computation: toward a new philosophy of machine intelligence. Piscataway, IEEE Press. 270p.

Goldberg D.E. (1989) Genetic algorithms in search, optimization, and machine learning. Reading, Addison-Wesley. 412p.

Herrera F., Lozano M. and Verdegay J.L. (1998) Tackling Real Coded Genetic Algorithms: Operators and Tools for Behavioural Analysis. Artificial Intelligence Review 12, 265-319.

Ikonen E. and Kaddour N. (2002) Advanced process identification and control. New York, Marcel Dekker. 310p.

Katare S., Bhan A., Caruthers J.M., Delgass W.N. and Venkatasubramanian V. (2004) A hybrid genetic algorithm for efficient parameter estimation of large kinetic models. Computers and Chemical Engineering, 28, 2569-2581.

Larminie J. and Dicks A. (2003) Fuel cell systems explained, 2nd edition. West Sussex, England, John Wiley & Sons Ltd, 417p.

Mann R.F., Amphlett J.C., Hooper M.A.I., Jensen H.M., Peppley B.A. and Roberge P.R. (2000) Development and application of a generalised steady-state electrochemical model for a PEM fuel cell. Journal of Power Sources 86, 173-180.

Mezura-Montes E., Velázquez-Reyes J. and Coello Coello C.A. (2006) A Comparative Study of Differential Evolution Variants for Global Optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2006), Seattle, ACM Press. 485-492.

Michalewicz Z., Logan T.D. and Swaminathan S. (1994) Evolutionary operators for continuous convex parameter spaces. In: Sebald A.V. and Fogel L.J. (eds.):

Proceedings of the 3rd Annual Conference on Evolutionary Programming. River Edge World Scientific Publishing. 84-97.

Michalewicz Z. (1996) Genetic algorithms + data structures = evolution programs, 3rd edition. Berlin, Springer. 387p.

Mo Z.-J., Zhu X.-J., Wei L.-Y. and Cao G.-Y. (2006) Parameter optimization for a PEMFC model with a hybrid genetic algorithm. International Journal of Energy Research 30, 585-597.

Ohenoja M. and Leiviskä K. (2010) Validation of genetic algorithm results in a fuel cell model. International Journal of Hydrogen Energy, In Press, Corrected Proof.

Pant M., Ali M. and Abraham A. (2009) Mixed Mutation Strategy Embedded Differential Evolution. 2009 IEEE Congress on Evolutionary Computation, Trondheim, Norway, IEEE Press, 1240-1246.

Price K.V. (1996) Differential evolution: a fast and simple numerical optimizer. In: Smith M.H. et al. (eds.): Proceedings of the 1996 biennial conference of the North American fuzzy information processing society – NAFIPS, Berkeley. 524-527.

Price K.V., Storn R.M. and Lampinen J.A. (2005) Differential evolution: a practical approach to global optimization. Berlin, Springer. 538p.

Kukkonen S. and Lampinen J. (2005) An empirical study of control parameters for generalized differential evolution. In: Proceedings of Sixth Conference on Evolutionary and Deterministic Methods for Design, Optimisation and Control with Applications to Industrial and Societal Problems (EUROGEN 2005).

Rahnamayan S., Tizhoosh H.R. and Salama M.M.A. (2008) Opposition versus randomness in soft computing techniques. Applied Soft Computing 8, 906-918.

Rönkkönen J., Kukkonen A. and Price K.V. (2005) Real-parameter optimization with differential evolution. The 2005 IEEE Congress on Evolutionary Computation, volume 1, 506-513.

Sarker R., Kamruzzaman J. and Newton C. (2003) Evolutionary Optimization (EvOpt): A Brief Review and Analysis. International Journal of Computational Intelligence and Applications 3, 311-330.

Sorsa A., Peltokangas R. and Leiviskä K. (2008) Real-coded genetic algorithms and nonlinear parameter identification. IS'08, 4th International IEEE Conference Intelligent Systems, volume 2, 10-42 - 10-47.

Storn R. and Price K. (1995) Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. International Computer Science Institute, University of California. Technical Report TR-95-012.

Storn R. and Price K. (1996) Minimizing the real functions of the ICEC'96 contest by differential evolution. In: Proceedings of the 1996 IEEE international conference on evolutionary computation, Nagoya. 842-844.

Storn R. and Price K. (1997) Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341-359.

Tvrdík J (2009) Adaptation in differential evolution: A numerical comparison. Applied Soft Computing 9, 1149-1155.

Wang C., Nehir M.H. and Shaw S.R. (2005) Dynamic models and model validation for PEM fuel cells using electrical circuits. IEEE Transactions on Energy Conversion 20, 442-451.

Wolf D. and Moros R. (1997) Estimating rate constants of heterogeneous catalytic reactions without supposition of rate determining surface steps – and application of a genetic algorithm. Chemical Engineering Science 52, 2569-2581.
Zaharie D. (2002) Critical Values for the Control Parameters of Differential Evolution Algorithms. In: Matousek R. and Osmera P. (eds.): Proceedings of Mendel 2002, 8th International Conference on Soft Computing, Brno, 62-67.
Zaharie D. (2009) Influence of crossover on the behavior of Differential Evolution Algorithms. Applied Soft Computing 9, 1126-1138.

ISBN 978-951-42-6332-3 (pdf)
ISSN 1238-9390
University of Oulu
Control Engineering Laboratory – Series A
Editor: Leena Yliniemi

28. **Mattina V & Yliniemi L**, Process control across network, 39 p. October 2005. ISBN 951-42-7875-5.
29. **Ruusunen M**, Monitoring of small-scale biomass combustion processes. 28 p. March 2006. ISBN 951-42-8027-X. ISBN 951-42-8028-8 (pdf).
30. **Gebus S, Fournier G, Vittoz C & Ruusunen M**, Knowledge extraction for optimizing monitorability and controllability on a production line. 36 p. March 2006. ISBN 951-42-9390-X.
31. **Sorsa A & Leiviskä K**, State Detection in the biological water treatment process. 53 p. November 2006. ISBN 951-42-8273-6.
32. **Mäyrä O, Ahola T & Leiviskä K**, Time delay estimation and variable grouping using genetic algorithms. 22 p. November 2006. ISBN 951-42-8297-3.
33. **Paavola M**, Wireless Technologies in Process Automation - A Review and an Application Example. 46 p. December 2007. ISBN 978-951-42-8705-3.
34. **Peltokangas R & Sorsa A**, Real-coded genetic algorithms and nonlinear parameter identification. 28 p. April 2008. ISBN 978-951-42-8785-5. ISBN 978-951-42-8786-2 (pdf).
35. **Rami-Yahyaoui O, Gebus S, Juuso E & Ruusunen M**, Failure mode identification through linguistic equations and genetic algorithms. August 2008. ISBN 978-951-42-8849-4, ISBN 978-951-42-8850-0 (pdf).
36. **Juuso E, Ahola T & Leiviskä K**, Variable selection and grouping. August 2008. ISBN 978-951-42-8851-7. ISBN 978-951-42-8852-4 (pdf).
37. **Mäyrä O & Leiviskä K**, Modelling in methanol synthesis. December 2008. ISBN 978-951-42-9014-5.
38. **Ohenoja M**, One- and two-dimensional control of paper machine: a literature review. October 2009. ISBN 978-951-42-9316-0.
39. **Paavola M & Leiviskä K**, ESNA – European Sensor Network Architecture. Final Report. 12 p. December 2009. ISBN 978-951-42-6091-9.
40. **Virtanen V & Leiviskä K**, Process Optimization for Hydrogen Production using Methane, Methanol or Ethanol. December 2009. ISBN 978-951-42-6102-2.
41. **Keskitalo J & Leiviskä K**, Mechanistic modelling of pulp and paper mill wastewater treatment plants. January 2010. ISBN 978-951-42-6110-7.
42. **Kiuttu J, Ruuska J & Yliniemi L**, Advanced and sustainable beneficiation of platinum group metals (PGM) in sulphide poor platinum (PGE) deposits – BEGBE. Final Report. May 2010. ISBN 978-951-42-6234-0.
43. **Ohenoja M, Isokangas A & Leiviskä K**, Simulation studies of paper machine basis weight control. August 2010. ISBN 978-951-42-6271-5.
44. **Sorsa A, Koskenniemi A & Leiviskä K,**, Evolutionary algorithms in nonlinear model identification. 38 p. September 2010. ISBN 978-951-42-6332-3 (pdf).