Chapter 9
On Synthesis and Solutions of Nonlinear Differential Equations—A Bio-Inspired Approach

Ivan Zelinka

Abstract This chapter discusses an alternative approach for mathematical-physical problems solution by means of bio-inspired methods, especially by evolutionary algorithms. Two different approaches are demonstrated here. The first one is the use of evolutionary algorithms on design, parameter estimation and control of the chemical reactor that is represented by 5 nonlinear and mutually joined differential equations, the second one is the use of analytic programming (method of the same class as genetic programming or grammatical evolution) to solve two different differential equations (4th and 2nd order), that represent problems from civil engineering by appropriate function synthesis. Theoretical background as well as applications are discusses here.

9.1 Introduction

Evolutionary computation is a sub-discipline of computer science belonging to the 'bio-inspired' computing area. The main ideas of evolutionary computation have been published for example in [1] and widely introduced to the scientific community [6, 8]. The most well known evolutionary techniques are Genetic Algorithms (GA) introduced by J. Holland [6, 8] based on ideas of A.M Turing [7] and based on first computer experiments by Barricelli in [2], Evolutionary Strategies (ES) by Schwefel [21] and Rechenberg [18] and Evolutionary Programming (EP) by Fogel [5] for example.

The main idea is that every individual of a species can be characterized by its features and abilities that help it to cope with its environment in terms of survival and reproduction. These features and abilities can be termed by its fitness and are inheritable via its genome. In the genome the features/abilities are encoded. The

I. Zelinka (✉)
Faculty of Electrical Engineering and Computer Science, Department of Computer Science, VSB-TU, 17. Listopadu 15, Ostrava-Poruba, Czech Republic
e-mail: ivan.zelinka@vsb.cz
URL:http://www.ivanzelinka.eu

A. Adamatzky (ed.), *Advances in Unconventional Computing*,
Emergence, Complexity and Computation 23,
DOI 10.1007/978-3-319-33921-4_9

code in the genome can be viewed as a kind of description that allows to store, process and transmit the information needed to build the individual. So, the fitness coded in the parent's genome can be handed over to new descendants and support the descendants in performing in the environment. The evolutionary principles are transferred into computational methods in a simplified form that will be outlined now. If the evolutionary principles are used for the purposes of calculations, the following procedure is used, as reported in [26]:

1. Specification of the EA parameters: For each algorithm, parameters that control the run of algorithm or terminate it regularly must be defined, if the termination criterion defined in advance are fulfilled (for example, the number of cycles—generations). Part of this point is the definition of the cost function (objective function) or, as the case may be, what is called fitness—a modified return value of the objective function). The objective function is usually a mathematical model of the problem, whose minimization or maximization leads to the solution of the problem.
2. Generation of the initial population (generally $N \times M$ matrix, where N is the number of parameters of an individual—D. Depending on the number of optimized arguments of the objective function and the user's criterions, the initial population of individuals is generated. An individual is a vector of numbers having such a number of components as the number of optimized parameters of the objective function. These components are set randomly and each individual thus represents one possible specific solution of the problem. The set of individuals is called population.
3. All the individuals are evaluated through a defined objective function and to each one of them fitness value is assigned, which is a modified (usually normalized) value of the objective function, or directly the value of the objective function.
4. Now parents are selected according to their quality (fitness, value of the objective function) or, as the case may be, also according to other criterions.
5. Descendants are created by crossbreeding the parents. The process of crossbreeding is different for each algorithm. Parts of parents are changed in classic genetic algorithms, in a differential evolution, crossbreeding is a certain vector operation, etc.
6. Every descendant is mutated by presence of randomness. In other words, a new individual is changed by means of a suitable random process. This step is equivalent to the biological mutation of the genes of an individual.
7. Every new individual is evaluated in the same manner as in step 3.
8. The best individuals are selected.
9. The selected individuals fill a new population.
10. The old population is eliminated and is replaced by a new population; step 4 represents further continuation.

Steps 4–10 are repeated until the number of evolution cycles (generations etc.) specified before by the user is reached or if the required quality of the solution is not achieved. The principle of the evolutionary algorithm outlined above is general

and might more or less differ in specific cases. For more detailed overview about evolutionary algorithms we strongly recommend to read [26].

Together with, say classical evolutionary algorithms, an alternative approach how to use evolution has been developed (since 90's) so that instead of numerical solutions (i.e. unknown parameters estimation) of given problem, symbolic solutions have been derived in the form of mathematical formulas, electronic circuits etc. Generally this approach can be called as "symbolic regression" and the most popular and classic approaches are for example Genetic Programming (GP) [11] or Grammatical Evolution (GE) [15]. Another interesting research was carried out by Artificial Immune Systems (AIS) or/and systems, which do not use tree structures like linear GP and other similar algorithm like Multi Expression Programming (MEP), etc.

In this chapter, a different method called Analytic Programming (AP) [27], is used. AP is a grammar free algorithm—structure, which can be used by any programming language and also by any arbitrary evolutionary algorithm (EA) or another class of numerical optimization method.

The term *symbolic regression* represents a process during which measured data sets are fitted, thereby a corresponding mathematical formula is obtained in an analytical way. An output of the symbolic expression could be, for example, $\sqrt[N]{x^2 + \frac{y^3}{k}}$, and the like. For a long time, symbolic regression was a domain of human calculations but in the last few decades it involves computers for symbolic computation as well.

For closer description of the theoretical principles of AP it is recommended to study [27]. Comparative studies with selected well known case examples from GP as well as applications on synthesis of: controller, systems of deterministic chaos, electronics circuits, etc. are described there. For simulation purposes, AP has been co-joined with EA's like Differential Evolution (DE) [22], Self-Organizing Migrating Algorithm (SOMA) [3], Genetic Algorithms (GA) [6] and Simulated Annealing (SA) [4, 10]. All case studies are described, mentioned and referenced there.

The initial idea of symbolic regression by means of a computer program was proposed in GP [11]. The other approach of GE was developed in [15] and AP in [27]. Another interesting investigation using symbolic regression was carried out in [20] on AIS and Probabilistic Incremental Program Evolution (PIPE), which generates functional programs from an adaptive probability distribution over all possible programs. Yet another new technique is the so called *Transplant Evolution*, see [23] or [24] which is closely associated with the conceptual paradigm of AP, and modified for GE. GE was also extended to include DE by [14]. Symbolic regression is schematically depicted in Fig. 9.1. Generally speaking, it is a process which combines, evaluates and creates more complex structures based on some elementary and non-complex objects, in an evolutionary way. Such elementary objects are usually simple mathematical operators $(+, -, \times, \ldots)$, simple functions $(sin, cos, And, Not, \ldots)$, user-defined functions (simple commands for robots—MoveLeft, TurnRight, …), etc. An output of symbolic regression is a more complex "object" (formula, function, command, …), solving a given problem like data fitting of the so-called Sextic and Quintic problem described by Eq. (9.1) [12], randomly synthesized

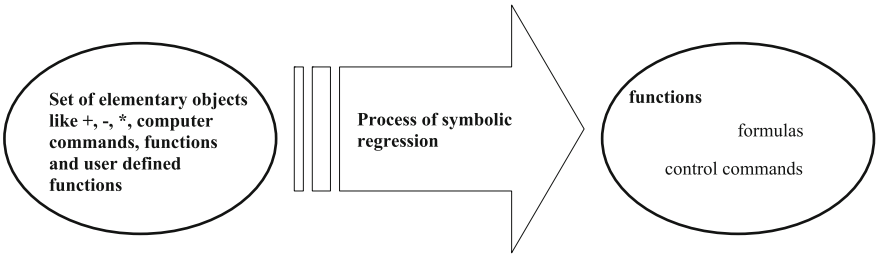

Fig. 9.1 Symbolic regression—schematic view

function by Eq. (9.2), Boolean problems of parity and symmetry solution (basically logical circuits synthesis) by Eq. (9.3) [11, 29], or synthesis of quite complex robot control command by Eq. (9.4) [12, 16]. Equations (9.1)–(9.4) mentioned here are just a few samples from numerous repeated experiments done by AP, which are used to demonstrate how complex structures can be produced by symbolic regression in general for different problems.

$$x\left(K_1+\frac{\left(x^2K_3\right)}{K_4\left(K_5+K_6\right)}\right)*\left(-1+K_2+2x\left(-x-K_7\right)\right) \tag{9.1}$$

$$\sqrt{t}\left(\frac{1}{\log\left(t\right)}\right)^{\sec^{-1}(1.28)}\log^{\sec^{-1}(1.28)}\left(\sinh\left(\sec\left(\cos\left(1\right)\right)\right)\right) \tag{9.2}$$

$$\begin{aligned}&Nor[(Nand[Nand[B\ ||\ B,\ B\ \&\&\ A],\ B])\ \&\&\ C\ \&\&\ A\ \&\&\ B,\\ &Nor[(\ !C\ \&\&\ B\ \&\&\ A\ ||\ !A\ \&\&\ C\ \&\&\ B\ ||\ !C\ \&\&\ !B\ \&\&\ !A)\ \&\&\\ &(\ !C\ \&\&\ B\ \&\&\ A\ ||\ !A\ \&\&\ C\ \&\&\ B\ ||\ !C\ \&\&\ !B\ \&\&\ !A)\ ||\\ &A\ \&\&\ (\ !C\ \&\&\ B\ \&\&\ A\ ||\ !A\ \&\&\ C\ \&\&\ B\ ||\ !C\ \&\&\ !B\ \&\&\ !A),\\ &(C\ ||\ !C\ \&\&\ B\ \&\&\ A\ ||\ !A\ \&\&\ C\ \&\&\ B\ ||\ !C\ \&\&\ !B\ \&\&\ !A)\ \&\&\ A]]\end{aligned} \tag{9.3}$$

$$\begin{aligned}&\text{Prog2[Prog3[Move, Right, IfFoodAhead[Left, Right]],}\\ &\text{IfFoodAhead[IfFoodAhead[Left, Right], Prog2[IfFoodAhead[}\\ &\text{IfFoodAhead[IfFoodAhead[Left, Right], Right], Right],}\\ &\text{IfFoodAhead[Prog2[Move, Move], Right]]]]}\end{aligned} \tag{9.4}$$

Lets briefly discuss the main ideas of GP, GE and AP to give better overview of SR background in order to understand how it has been used on examples here.

9.1.1 Genetic Programming

GP was the first tool for symbolic regression carried out by means of computers instead of humans. The main idea comes from GA, which was used in GP [11, 12]. Its ability to solve very difficult problems is well proven; for example, GP performs so well that it can be applied to synthesize highly sophisticated electronic circuits, etc.

The main principle of GP is based on GA, which is working with populations of individuals represented in the LISP programming language. Individuals in a canonical form of GP are not binary strings, different from GA, but consist of LISP symbolic objects (commands, functions, …), etc. These objects come from LISP, or they are simply user-defined functions. Symbolic objects are usually divided into two classes: functions and terminals. Functions were previously explained and terminals represent a set of independent variables like x, y, and constants like π, 3.56, etc.

The main principle of GP is usually demonstrated by means of the so-called trees (basically graphs with nodes and edges, as shown in Figs. 9.2 and 9.3, representing individuals in LISP symbolic syntax). Individuals in the shape of a tree, or formula like $0.234\ Z\ +\ X\ -\ 0.789$, are called programs. Because GP is based on GA,

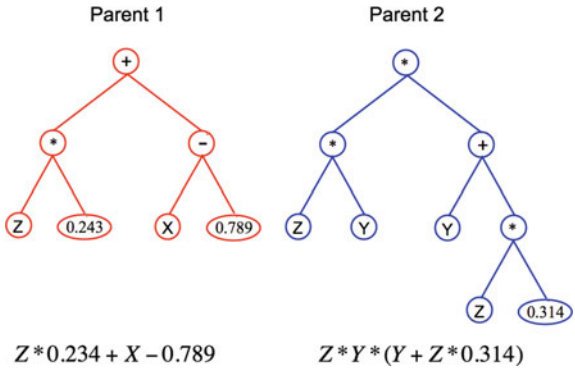

Fig. 9.2 Parental trees

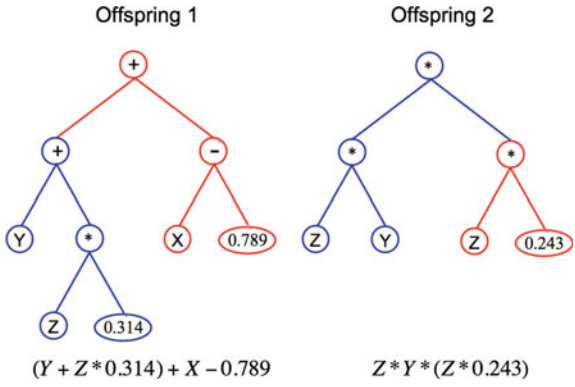

Fig. 9.3 Offsprings

evolutionary steps (mutation, crossover, …) in GP are in principle the same as GA. As an example, GP can serve two artificial parents—trees on Figs. 9.2 and 9.3, representing programs $0.234\ Z\ +\ X\ -\ 0.789$ and $ZY(Y + 0.314Z)$. When crossover is applied, for example, subsets of trees are exchanged. Resulting offsprings of this example are shown on Fig. 9.3.

Subsequently, the offspring fitness is calculated, such that the behavior of the just-synthesized and evaluated individual-tree should be as similar as possible to the desired behavior. The desired behavior can be regarded as a measured data set from some process (a program that should fit them as well as possible) or like an optimal robot trajectory, i.e., when the program is evaluating a sequence of robot commands (TurnLeft, Stop, MoveForward,...) leading as close as possible to the final position. This is basically the same for GE.

For detailed description of GP, see for example classical books [11, 12].

9.1.2 Grammatical Evolution

GE is another program developed in [15] which performs a similar task to that of GP. GE has one advantage over GP, which is the ability to use any arbitrary programming language, not only LISP as in the case of the canonical version of GP. In contrast to other EA's, GE was used only with a few search strategies, and with a binary representation of the populations. The last successful experiment with DE applied on GE was reported in [14]. GE in its canonical form is based on GA, thanks to few important changes it has in comparison with GP. The main difference is in the individual coding.

While GP manipulates in LISP symbolic expressions, GE uses individuals based on binary strings. These are transformed into integer sequences and then mapped into a final program in the Backus-Naur Form (BNF) [15], as explained by the following artificial example. Let $T = \{+, -, \times, /, x, y\}$ be a set of operators and terminals and let F = {epr, op, var} be the so-called nonterminals. In this case, the grammar used for final program synthesis is given in Table 9.1. The rule used for individuals transforming into a program is based on Eq. (9.5) below. GE is based on binary chromosome with a variable length, divided into the so-called codons (range of integer values, 0–255), which is then transformed into an integer domain according to Table 9.2.

$$\begin{aligned} &\text{unfolding} \ = \ \text{codon mod rules} \\ &\text{where rules is number of rules for given nonterminal} \end{aligned} \tag{9.5}$$

Synthesis of an actual program can be described by the following. Start with a nonterminal object expr. Because the integer value of Codon 1 (see Table 9.2) is 40, according to Eq. (9.5), one has an unfolding of *expr* = *op expr expr* (40 mod 2, 2 rules for *expr*, i.e., 0 and 1). Consequently, Codon 2 is used for the unfolding of *op* by * (162 mod 4), which is the terminal and thus the unfolding for this part of program is

Table 9.1 Grammatical evolution—rules

Nonterminals	Unfolding	Index
expr	::= op expr expr	0
	var	1
op	::= +	0’
	-	1’
	*	2’
	/	3’
var	:: X	0”
	Y	(1”)

Table 9.2 Grammatical evolution—codon

Chromozone	Binary	Integer	BNF index
Codon 1	101000	40	0
Codon 2	11000011	162	2’
Codon 3	1100	67	1
Codon 4	10100010	12	0”
Codon 5	1111101	125	1
Codon 6	11100111	231	1”
Codon 7	10010010	146	Unused
Codon 8	10001011	139	Unused

closed. Then, it continues in unfolding of the remaining nonterminals (*expr expr*) till the final program is fully closed by terminals. If the program is closed before the end of the chromosome is reached, then the remaining codons are ignored; otherwise, it continues again from the beginning of the chromosome. The final program based on the just-described example is in this case $x \cdot y$ (see Fig. 9.4). For a fully detailed description of GE principles, see [15].

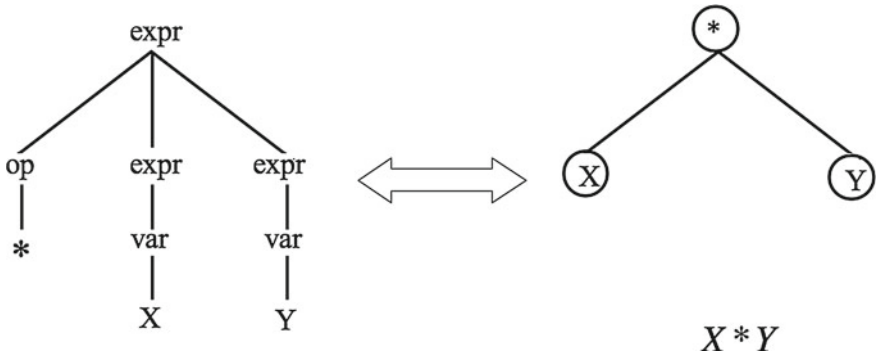

Fig. 9.4 Final program by GE

9.1.3 Analytic Programming

The final method described here and used for experiments in this chapter is called AP, which has been compared to GP with very good results (see, for example, [16, 27, 29]).

The basic principles of AP were developed in 2001 [27] and it is also based on the set of functions, operators and terminals, which are usually constants or independent variables alike, for example:

- **functions**: $sin, tan, tanh, And, Or, \ldots$
- **operators**: +, -, $\times$, /, $dt, \ldots$
- **terminals**: 2.73, 3.14, $t, \ldots$

All these objects create a set, from which AP tries to synthesize an appropriate solution. Because of the variability of the content of this set, it is called a general functional set (GFS). The structure of GFS is nested, i.e., it is created by subsets of functions according to the number of their arguments (Fig. 9.5). The content of GFS is dependent only on the user. Various functions and terminals can be mixed together. For example, GFS_{all} is a set of all functions, operators and terminals, GFS_{3arg} is a subset containing functions with maximally three arguments, GFS_{0arg} represents only terminals, etc. (Fig. 9.5).

AP, as further described later, is a mapping from a set of individuals into a set of possible programs. Individuals in population and used by AP consist of non-numerical expressions (operators, functions, …), as described above, which are represented by their integer position indexes in the evolutionary process (Figs. 9.6 and 9.7, see also Chap. 2). This index then serves as a pointer into the set of

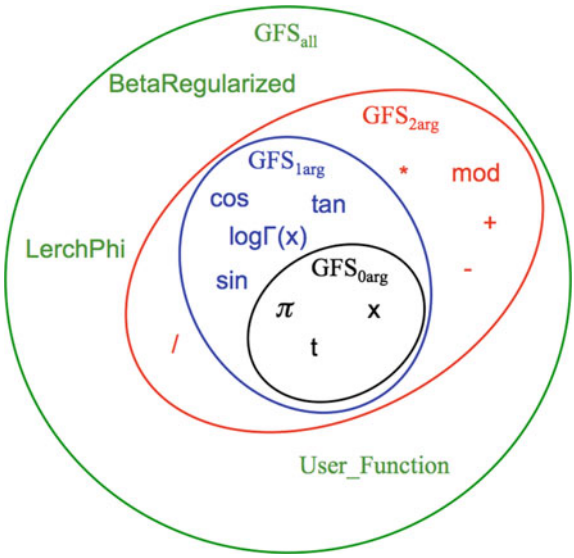

Fig. 9.5 Hierarchy in GFS

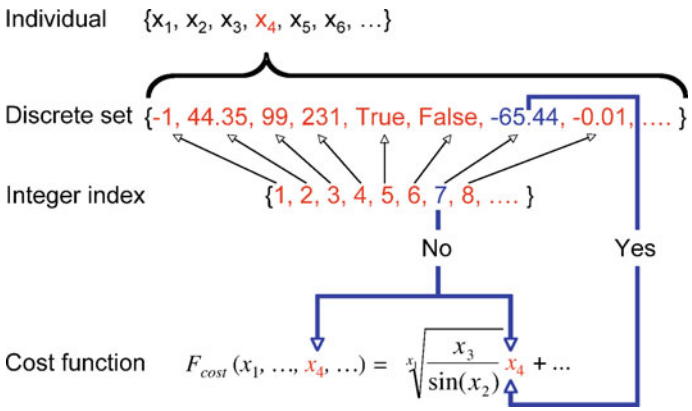

Fig. 9.6 DSH-Integer index, see Chap. 2

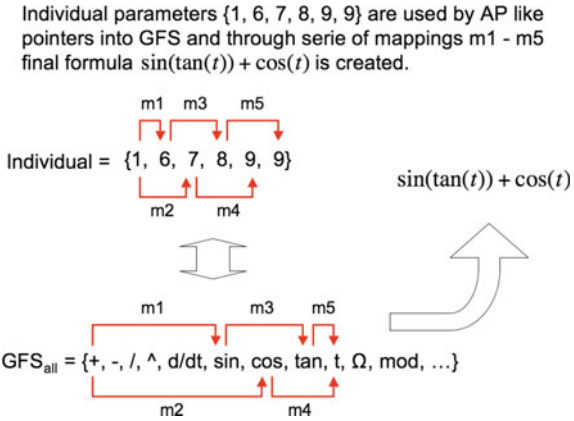

Fig. 9.7 Principle of mapping from GFS to programs

expressions and AP uses it to synthesize the resulting function-program for cost function evaluation.

Figure 9.7 demonstrates an artificial example how a final function is created from an integer individual via Discrete Set Handling (DSH). Number 1 in the position of the first parameter means that the operator + from GFS_{all} is used (the end of the individual is far enough). Because the operator + must have at least two arguments, the next two index pointers 6 (sin from GFS) and 7 (cos from GFS) are dedicated to this operator as its arguments. The two functions, sin and cos, are one-argument functions, so the next unused pointers 8 (tan from GFS) and 9 (t from GFS) are dedicated to the sin and cos functions. As an argument of cos, the variable t is used, so this part of the resulting function is closed (t is zero-argument) in its AP development. The one-argument function tan remains, and because there is one unused pointer 9, tan is mapped on t which is on the 9th position in GFS.

To avoid synthesis of pathological functions, a few security *tricks* are used in AP. The first one is that GFS consists of subsets containing functions with the same or a smaller number of arguments. The nested structure (see also Fig. 9.5) is used in the special security subroutine, which measures how far the end of an individual is and, according to this, mathematical elements from different subsets are selected to avoid pathological functions synthesis. More precisely, if more arguments are desired then a possible function (the end of the individual is near) will be replaced by another function with the same index pointer from the subset with a smaller number of arguments. For example, it may happen that the last argument for one function will not be a terminal (zero-argument function). If the pointer is longer than the length of subset, e.g., a pointer is 5 and is used GFS_0, then the element is selected according to the rule: element = pointer_value mod number_of_elements_in_GFS_0. In this example, the selected element would be the variable t (see GFS_0 in Fig. 9.5).

GFS need not be constructed only from clear mathematical functions as demonstrated above, but may also be constructed from other user-defined functions, e.g., logical functions, functions which represent elements of electrical circuits or robot movement commands, linguistic terms, etc.

9.1.3.1 Versions

AP was evaluated in three versions. All three versions utilize the same set of functions for program synthesis, terminals, etc., as in GP [11, 12]). The second version labelled as AP_{meta} (the first version, AP_{basic}) is modified in the sense of constant estimation. For example, the so-called sextic problem was used in [12] to randomly generate constants, whereas AP uses only one, called K, which is inserted into the formula (9.6) below at various places by the evolutionary process. When a program is synthesized, all K's are indexed as K_1, K_2, …, K_n to obtain (9.7) the formula, and then all K_n are estimated by using a second EA, the result of which can be, for example, (9.8). Because EA (slave) "works under" EA (master), i.e., $EA_{master} \rightarrow$ program $\rightarrow K$ indexing $\rightarrow EA_{slave} \rightarrow$ estimation of K_n, this version is called AP with metaevolution, denoted as AP_{meta}.

$$\frac{x^2 + K}{\pi^K} \tag{9.6}$$

$$\frac{x^2 + K_1}{\pi^{K_2}} \tag{9.7}$$

$$\frac{x^2 + 3.56}{\pi^{-229}} \tag{9.8}$$

Due to this version being quite time-consuming, AP_{meta} was further modified to the third version, which differs from the second one in the estimation of K. This is

accomplished by using a suitable method for nonlinear fitting (denoted AP_{nf}). This method has shown the most promising performance when unknown constants are present. Results of some comparative simulations can be found in [27]. AP_{nf} was the method chosen for the simulations described in this chapter.

9.1.3.2 Analytic Programming Subroutines

AP described above is in full detail explained in [27]. Special procedures, that ensure that AP is stable, fast and efficient algorithm are described there. Reader who is interested in this topic can get an explanation what structure is created by basic building block elements, how is individual (i.e. the vector of the real number representation) mapped into space of possible programs/solutions (i.e. math formulas, electronic circuits etc.), how crossover and mutation are done, what is reinforced AP evolution as well as how, so called security procedures, are used to ensure that AP will generate "non-pathological" solutions (i.e. closed solutions—no divide by 0, synthesized function has all arguments as needed etc.). Also similarities and differences between AP, GP and GE are discussed in [27].

9.2 Selected Applications

This section briefly describes some selected applications of classical EA and AP use, which have been conducted during the past few years. The most representative from our own research are

1. Evolutionary control of the chemical reactor control and synthesis of its geometrical structure and parameters estimation.
2. Civil engineering problem solution.

The first case has been solved by classical evolutionary approach and the second one by means of SR with AP and SOMA use.

9.2.1 Chemical Reactor—Predictive Control and Design

Chemical process control requires intelligent monitoring due to the dynamic nature of the chemical reactions and the non-linear functional relationship between the input and output variables involved. Chemical reactors are one of the major processing units in many chemical, pharmaceutical and petroleum industries as well as in environmental and waste management engineering. In spite of continuing advances in optimal solution techniques for optimization and control problems, many of such problems remain too complex to be solved by the known techniques. In chemical engineering, evolutionary optimization has been applied by the author and others to

system identification a model of a process is built and its numerical parameters are found by error minimization against experimental data. Evolutionary optimization has been widely applied to the evolution of neural networks models for use in control applications. The area of reactor network synthesis currently enjoys a proliferation of contributions in which researchers from various perspectives are making efforts to develop systematic optimization tools to improve the performance of chemical reactors. The contributions reflect on the increasing awareness that textbook knowledge and heuristics [13], commonly employed in the development of chemical reactors, are now deemed responsible for the lack of innovation, quality, and efficiency that characterizes many industrial designs, [13]. The main aim of this participation is to show that evolutionary algorithms (EAs) are capable of optimization on chemical engineering processes. The ability of EAs to successfully work with at investigation on optimization and predictive control of chemical reactors. Firstly, a nonlinear mathematical model is required to describe the dynamic behavior of batch process; this justifies the use of evolutionary method to deal with this process, for static optimization of a chemical continuous stirred tank reactor. Consequently, it is used to design geometry technique equipment for chemical reaction. In the next part, we have used EAs to predictive control of chemical process of reactor, too. The optimized reactor and predictive control were used in a simulation with optimization by evolutionary algorithms and the results are presented in graphs. The use of evolutionary algorithms in optimization and control of chemical technologies is very important today and many researchers are working in that field. They are using classical as well as evolutionary algorithms for those purposes, lets mention for example [13, 17] for more classical approach to solve problems of chemical technologies, [9]. Surprisingly, many problems can be defined as optimization problems, e.g. the optimal trajectory of robot arms; the optimal thickness of steel in pressure vessels; the optimal set of parameters for controllers; optimal relations or fuzzy sets in fuzzy models; and so on. Solutions to such problems are usually more or less hard to arrive at, their parameters usually including variables of different types, such as real or integer variables. Evolutionary algorithms are quite popular because they allow the solution of almost any problem in a simplified manner, because they are able to handle optimizing tasks with mixed variables—including the appropriate constraints, as and when required. This paper explains SOMA's use on design and predictive control of given chemical reactor.

9.2.1.1 Reactor Description

Model of the reactor as depicted in Fig. 9.8 inside which above-mentioned reactions can be realized was given by the system of equations (9.9). This is set of 5 nonlinear differential equations that are mutually joined and coupled. Exact solution of such kind of equation system is in an analytic way usually very hard or impossible. In the system (9.9) there are also many free, i.e. adjustable parameters. The set of adjustable parameters is in Table 9.3. Some of them are given by kind of chemical reactions running inside reactor, some of them can be set 'arbitrary' based on expert knowledge

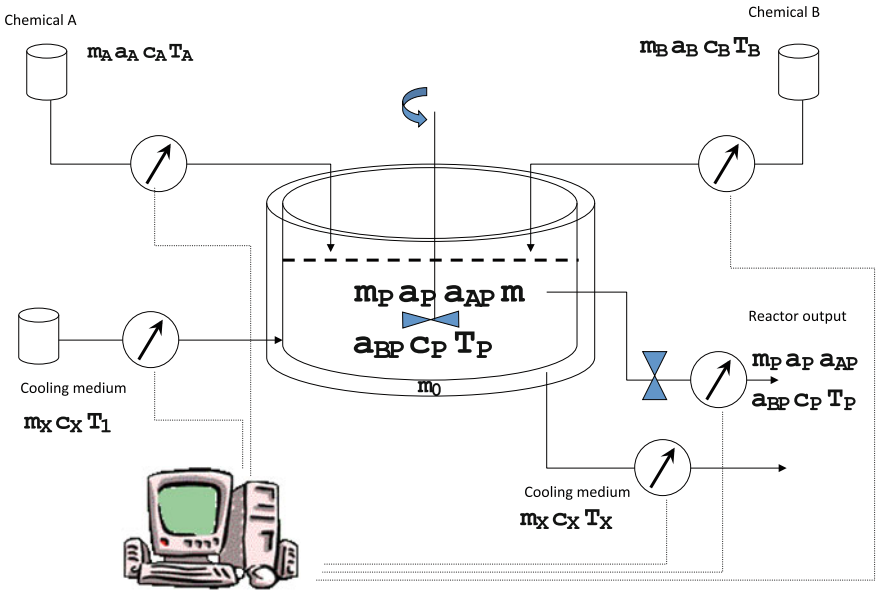

Fig. 9.8 Reactor scheme

Table 9.3 Chemical reactor adjustable parameters

Value	Units in SI	Description
m	Kg	Chemical mass inside reactor
m_A	Kgs^{-1}	Chemical input A
m_B	Kgs^{-1}	Chemical input B
a_A	%	Concentration of chemical o input A
a_B	%	Concentration of chemical on input B
T_A	K	Input temperature of chemical on input A
T_B	K	Input temperature of chemical on input B
T_1	K	Input temperature of cooling medium
m_P	Kgs^{-1}	Output of chemical product
S	m^2	Cooling surface
m_0	Kgs^{-1}	I/O cooling medium
m_X	Kg	Cooling medium inside reactor

or/and by selected numerical methods of optimization. For our purposes the set of adjustable parameters is in Table 9.3. The chemical reactions in this reactor belong to the class of exothermic reactions (releasing heat) so reactor needs to be cooled, as mentioned on the Fig. 9.8 (double reactor wall for cooling medium). For more about chemical reactors and its control it is recommended to see for example [13, 17].

$$
\begin{aligned}
\dot{a}_{AP} &= \frac{a_A m_A}{m} - \frac{a_{AP}(t) m_P}{m} - Ae^{-\frac{E}{RT_P(t)}} a_{AP}(t) a_{BP}(t) \\
\dot{a}_{BP} &= \frac{a_B m_B}{m} - \frac{a_{BP}(t) m_P}{m} - Ae^{-\frac{E}{RT_P(t)}} a_{AP}(t) a_{BP}(t) \\
\dot{a}_P &= Ae^{-\frac{E}{RT_P(t)}} a_{AP}(t) a_{BP}(t) - \frac{a_P(t) m_P}{m} \\
\dot{T}_P &= \frac{T_A m_A c_A}{m c_P} - \frac{T_B m_B c_B}{m c_P} + \frac{Ae^{-\frac{E}{RT_P(t)}} a_{AP}(t) a_{BP}(t) \Delta \dot{H}_r}{c_P} - \frac{Skp_1 T_P(t)}{m c_P} - \frac{m_P T_P(t)}{m} + \frac{Skp_1 T_x(t)}{m c_P} \\
\dot{T}_x &= \frac{T_1 m_x}{m_0} + \frac{Skp_1 T_P(t)}{m_0 c_X} - \frac{T_P(t) m_X}{m_0} - \frac{Skp_1 T_X(t)}{m_0 c_X}
\end{aligned}
\tag{9.9}
$$

Certain class of chemical reactions like enzymatic dechromation technology, etc. can be realized inside this kind of reactor. The advantage of the enzymatic reaction is the production of protein hydrolyzates of relatively good quality and chrome sludge. Using organic bases to form alkaline reaction mixture increases the quality of both its products. A partial regeneration of organic base when diluted protein hydrolyzates undergo concentration cuts the operating costs of enzymatic hydrolysis. In commercial application, the greatest volume of protein hydrolyzate is channelled into agriculture. Hydrolyzate, as an organic nitrogenous fertilizer, not only equals the combined urea-ammonium nitrate fertilizer in crop yield, but also surpasses its manifolds in the foodstuff value of consumer's greens. The content of nitrates is as much as 200 times lower on average. Hydrolyzate is also used in the manufacture of biodegradable foil, especially for producing sowing tape. Application of protein hydrolyzates appears to have good potential in the building industry and in manufacture of modified amino-plastic for adhesive compounds of zero, or at least very low, free formaldehyde content. A serious problem, and at present one lacking a completely satisfactory solution, is the development of recycling technology for chrome from so called filter cake. The main obstacle for the utilization of the chrome sludge is a relatively high content of proteins in the dry substance of cake. In closing, it may be said that enzymatic hydrolysis has a place in the treatment of chromium containing tannery waste and the funds expended on this field of research have brought satisfactory results.

9.2.1.2 Experiment Design and Used Algorithms

For our experiments, focused on predictive control of reactor (9.9), described here, reactor has been tested for its behavior under expert setting and evolutionary estimated and set parameters. Before predictive control has been tested on this reactor, behavior of reactor was investigated in order to select the best configuration for predictive control experiments. This was done in [28] and is only mentioned here. Expert parameters were used for initial—original setting. They come from experiences of experts and literature and were partly obtained during visit in laboratory, Resine and Composite for Forest Products' in Sainte-Foy, Canada. This set of parameters consisted of two kind of parameters i.e. parameters of chemical materials and physical parameters of reactor under consideration, see system of (9.9). An initial conditions (a_{AP0}, a_{BP0}, a_{P0}, T_{P0}, T_{X0},) used in following simulations are also described in Eq. (9.9). As the result shows [28] the reactor under expert parameters produce

unsatisfactory behavior [28]. Reactor production stabilizes itself (i.e. without control procedure) after 27 h on 15 % concentration of output chemical. From that point of view above-mentioned parameters were regarded as unsatisfactory. Used evolutionary algorithm was SOMA [3, 25]. It is a stochastic optimization algorithm that is modeled based on the social behavior of competitive—cooperating individuals. It was chosen because it has been proven that this algorithm has the ability to converge towards the global optimum. Before predictive control a few static optimizations by SOMA algorithm were consequently done in [28]. They were done in following steps:

1. Optimization without restrictions.
2. Optimization with restrictions applied exactly in given time.
3. Optimization with penalty applied during time interval.
4. Optimization with penalty applied during time interval and sub optimization of cooling surface.

The last static optimization is the important one. Each of four optimization cases was 10× repeated and consequently depicted [28]. From all 10 simulations the best reactor was finally chosen. In all four cases evolutionary optimization was focused on parameters (Table 9.3) estimation. According to reactor setting based on the Table 9.3 a global extreme was found in 13th dimensional configuration space. Last 13th dimension was cost value of cost function. In case of the last optimization (optimization with penalty applied during time interval and sub-optimization of cooling surface) searching for global extreme had run in 11th dimensional space because of relations among some parameters. Based on this, we decided that for predictive control (again done by evolutionary algorithms) will be used reactor whose structure was optimized by evolutionary algorithms. As mentioned in [28], reactor under expert parameters produce unsatisfactory behavior. Reactor production stabilizes itself (i.e. without control procedure) after 27 h on 15 % concentration of output chemical. Optimization of the parameters and reactor structure gives configuration, in which reactor production stabilizes itself (i.e. without control procedure) after 8 min on 50 % concentration of output chemical. It is 0.009 of the original time from expert set reactor. Thus improvement of the reactor is more than significant.

Differences between the two reactors are best seen in Table 9.5. Parameters expertly designed reactor and parameters obtained by static optimization are reported there. What is clear from both sets of parameters, the size of the reactor, which was the most successful optimization tied to the volume and surface cooling reactor is not clear from the Table 9.5. The terms r_S radius r_m and Δr (for the optimal case, the reactor is completely filled) are reported there. The parameter r_s is the radius of the reactor in the event that this relates to the cooling surface S and the radius r_m related to the total reactor volume derived from the m. In the original reactor there is a visible the gap between the two radii, which means that the reactor would need either additional auxiliary cooling surface or part of the cooling surface should not to be used. This is not the case of the optimized reactor. The parameter Δr is the difference between the outer and inner radius because it is a double-walled reactor. The

original reactor is the distance between outer and inner casing $\Delta r = 0.08\,\text{m}$ while the optimized $\Delta r = 0.14\,\text{m}$. Remaining parameters are clearly seen from Tables.

The cost value which minimization leads to the optimal control setting was done by Eq. (9.10) which consist of two parts. The first one was output product concentration a_P and the second output product temperature T_P. In fact during whole study has been used different cost functions (as mentioned in [28]) like Eq. (9.10) (basic optimization of the reactor structure), (9.11) (basic control on fixed output concentration of chemical and temperature a_p), (9.12) and (9.13) advanced cost functions for predictive control without and with penalization of output quality.

$$f_{cost} = 0.6 - \sum_{t=500}^{1000} a_P(t) \tag{9.10}$$

$$f_{cost} = \sum_{i=0}^{3000} a_p(i) + T_p(i) \quad i = \{0, 10, 20, \ldots, 3000\} \tag{9.11}$$

$$\begin{aligned}
&f_{cost} = |0.6 - a_P(t)| + |a_{AP}(\tau)| + |a_{BP}(\tau)| + |T_P(\tau)| + |T_X(\tau)| \\
&where \quad \begin{matrix} t = 1200 \\ \tau = 100 \end{matrix} \\
&under\ conditions \\
&a_{AP}(\tau) = \begin{cases} 0 & if \quad a_{AP}(\tau) \in < 0,\ 1 > \\ 100\, a_{AP}(\tau) & else \end{cases} \\
&a_{BP}(\tau) = \begin{cases} 0 & if i \quad a_{BP}(\tau) \in < 0,\ 1 > \\ 100\, a_{BP}(\tau) & else \end{cases} \\
&T_P(\tau) = \begin{cases} 0 & if \quad T_P(\tau) \in < 273.15,\ 273.15 + 150 > \\ T_P(\tau) & else \end{cases} \\
&T_X(\tau) = \begin{cases} 0 & if \quad T_X(\tau) \in < 273.15,\ 273.15 + 500 > \\ T_X(\tau) & else \end{cases}
\end{aligned} \tag{9.12}$$

$$\begin{aligned}
&f_{cost} = 0.6 - \sum_{t=500}^{1000} a_P(t) + |a_{AP}(\tau)| + |a_{BP}(\tau)| + |T_P(\tau)| + |T_X(\tau)| \\
&under\ conditions\ for\ \tau \in < 500,\ 1000 > \\
&a_{AP}(\tau) = \begin{cases} 0 & if \quad Max(a_{AP}(\tau))\ \&\ Min(a_{AP}(\tau)) \in < 0,\ 1 > \\ 100\, Max(a_{AP}(\tau)) & else \end{cases} \\
&a_{BP}(\tau) = \begin{cases} 0 & if \quad Max(a_{BP}(\tau))\ \&\ Min(a_{BP}(\tau)) \in < 0,\ 1 > \\ 100\, Max(a_{BP}(\tau)) & else \end{cases} \\
&T_P(\tau) = \begin{cases} 0 & if \quad Max(T_P(\tau))\ \&\ Min(T_P(\tau)) \in < 273.15,\ 273.15 + 150 > \\ Max(T_P(\tau)) & else \end{cases} \\
&T_X(\tau) = \begin{cases} 0 & if \quad Max(T_X(\tau))\ \&\ Min(T_X(\tau)) \in < 273.15,\ 273.15 + 150 > \\ Max(T_X(\tau)) & else \end{cases}
\end{aligned} \tag{9.13}$$

9.2.1.3 Predictive MIMO Evolutionary Control

The main aim of this experiment was to provide predictive control of chemical reactor (9.9), see also [13, 17]. In this case control was considered as the MIMO (multiple input–multiple output) control. Control process was done by evolutionary algorithms use instead of classical controller. Selected algorithm was SOMA [3] and was used like predictive controller estimating control parameters for reactor in order to reach desired values a_{AP} in the shortest time for different values of $a_{AP} = 0.35, 0.25, 0.1,$ 0.45(output concentration) and constant T_P (product temperature). The cost function (the predictive control functional) that has to be minimized by SOMA, in order to follow a_{AP} and T_P was given by Eq. (9.14).

$$J(N_1, N_2, N_u) = \sum_{i=1}^{L} \sum_{j=1}^{N_2} \eta_i(j)[y(k+j) - w(k+j)]^2 + \sum_{i=1}^{M} \sum_{j=1}^{N_u} \lambda_i(j)[\Delta u(k+j-1)]^2 \tag{9.14}$$

In this functional there is variable penalty of overshooting (controlled trajectory can overshoot desired value w and thus needs to be penalized) and in fact there are 2 of them ($L = 2$, penalization of two outputs). The dimension of inputs was 5: $\lambda = \Delta m_A, \Delta m_B, \Delta T_A, \Delta T_B, \Delta T_1 = [400, 200, 20, 20, 20]$, i.e. MIMO 5:2. Controlled variables ap and Tp were penalized when controlled values overstepped bigger value than $\eta = a_p, T_P = [400, 20]$. These parameters were chosen so that the effects of the two controlled variables on the resulting value of the functional approximately equal. Penalization of a_p was selected in the bigger range than of T_p. Thanks to this the impact of penalization on the final value of the functional was at least approximately the same level. Optimization would run of course even for the badly set penalization or also without it. Used algorithm would then put more emphasis on changing the values on which it was appropriate functional more sensitive, i.e. those that contribute more to change its cost value. The above values have been set approximate based on trial and error approach. Those that are numerically shown above, this seemed to be the most suitable for control of the proposed reactor above (9.9). Compared to classical approach of the predictive control, special feature of these simulations was that it did not carry out the calculation of the new controller output in each k_{th} step, but only if system behavior was changed by some external error or by change in the desired values. Given that, control actions were calculated based on the knowledge of future behavior (according to model reactor), it is still predictive control despite modification of the standard conventions for calculating control actions at each step k. This predictive control has been taken for the variable of mass flow rates and temperatures of both input products including coolant temperature. There were done two kinds of predictive control, without and with penalization of the output-controlled values [28], see also Figs. 9.9 and 9.10.

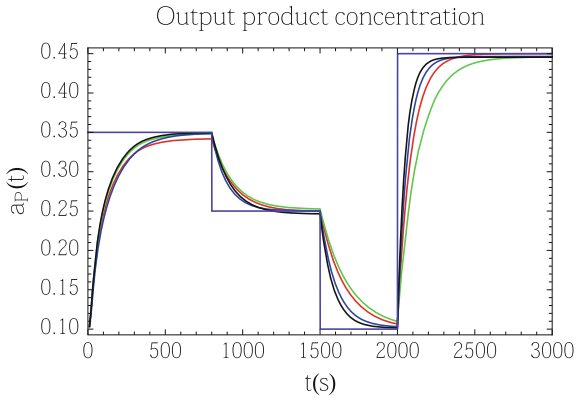

Fig. 9.9 Controlled concentration

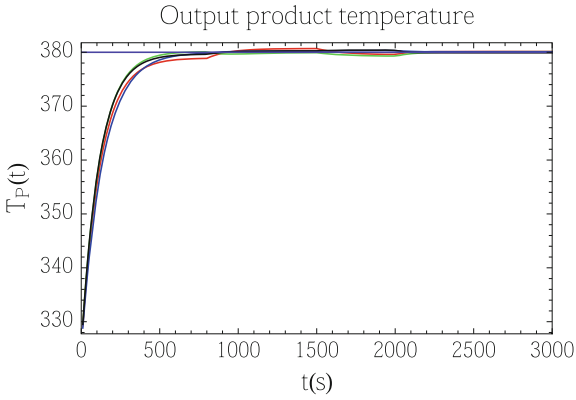

Fig. 9.10 Controlled temperature

9.2.1.4 Results

As discussed in the previous section, MIMO 5:2 was used in chemical reactor to control two kinds of predictive control, without and with penalization of the output-controlled values. Selected Figs. 9.9 and 9.10 from our experiments show both kinds of simulations that have been 4 times repeated and all 4 simulations were depicted in one figure to show its quality and diversibility. Figures 9.9 and 9.10 show how output variable a_p followed changes of desired value and how T_p stand on desired value for the control with penalization. Results are also reported in the Tables 9.4 and 9.5 and are calculated according to Eqs. (9.10)–(9.13). As seen from figures (Figs. 9.9 and 9.10) and tables, penalization plays important role in such kind of optimization and control.

Table 9.4 Optimal reactor behavior estimated by evolution

Parameter	a_p	T_p
Max	11.46	767.36
Avg	9.05	711.24
Min	6.99	654.15

Table 9.5 Difference between selected reactor parameters

Parameters	Expert setting	SOMA setting
r_s	11.46	767.36
r_m	9.05	711.24
Δr	6.99	654.15

9.2.2 Symbolic Solution of the Nonlinear ODEs

Another use of evolution, in this case the use of the AP on ordinary differential equations (ODE) solution synthesis was used here, especially on

1. ODE solving, exactly $u''(t) = cos(t), u(0) = 1, u(\pi) = -1, u'(0) = 0, u'(\pi) = 0$, see [19], 100× repeated
2. ODE solving, exactly $((4 + x)u''(x))'' + 600u(x) = 5000(x - x2), u(0) = 0, u(1) = 0, u''(0) = 0, u''(1) = 0$, see [19], 5× repeated

In this case we used SR—AP in order to search for such functions u (of course different for each case) that satisfied equations in both cases. Cost value of cost function for these simulations were difference (see Figs. 9.14 and 9.15, size of the filed area between original and just founded function) between actually generated and expected function.

9.2.2.1 On the Hilbert Space and the Classical Approach

Hilbert space widely used in physics and applied mathematics can be viewed like special case of analytic programming (AP). Lets see it in more detailed view. Hilbert space is usually defined like functional space with following properties [19]

1. Completeness.
2. Compactness.
3. Orthogonality.
4. Orthonormality (if no, Schmidt orthonormalization can be used, see [19].
5. ...

These demands allow to solve various problems (i.e. PDE etc.) usually in a simple manner. In a geometrical point of view Hilbert space can be depicted like mutually orthogonal axes. Each axis represents one base function, usually periodic functions like sinus or cosine. Position of each point in such so-called *functional space* can be described by its co-ordinates on all axes, or on the contrary, composition of

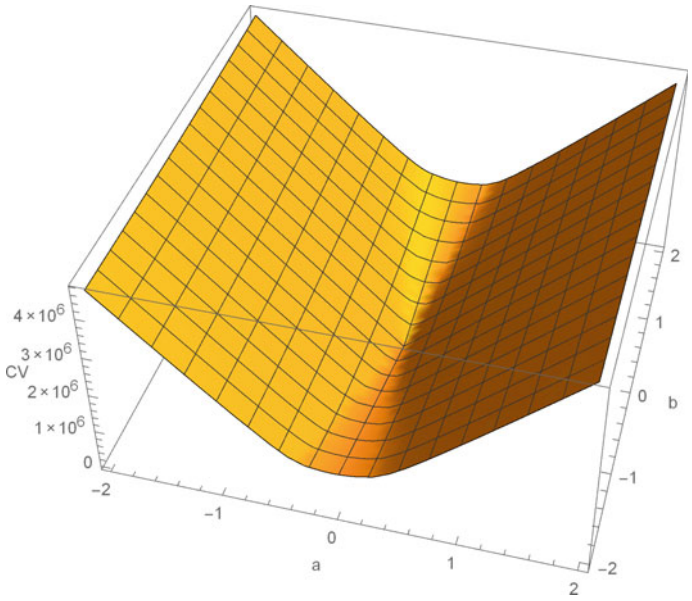

Fig. 9.11 Space of possible $u(x)$ solutions. Each point on surface is a value of the area between curves as on Fig. 9.14

all vector components destined by co-ordinates on all axes, give resulting function – vector from origin to discussed point in the Hilbert space. Solution of given problem, which is usually represented by ODE or PDE, is done in Hilbert space so that there is selected proper functional base of orthogonal functions, inside this base is build resulting *functional* and by means of suitable methods are estimated its unknown parameters. For example in [19] is solved ODE (see Case 2) whose cost function surface is depicted in Fig. 9.11 and some solutions on Fig. 9.14. Surface from Fig. 9.11 is depicted so, that z axis (each point of surface) in Fig. 9.11 is size of area between desired and actual solution depicted in Fig. 9.14. The aim of the numerical methods applied in the Hilbert space is to minimize this area, i.e. to find minimum on the cost function surface from Fig. 9.11.

9.2.2.2 Synthesis of the Case 1

This set of simulations was focused on ODE solving. In [19] it is solved by means of Ritz or Galerkin method on a priori selected functional base, which was orthogonal. Here it was solved by AP without any a priori demands on mixed functional base (sin, cos, …). The results were fully identical as with classical methods, i.e. $u(t) = -cos(t)$. This example was of course quite simple. More interesting example is the Case 2: the solution synthesis of the $((4 + x)u''(x))'' + 600u(x) = 5000(x - x2)$, $u(0) = 0, u(1) = 0, u''(0) = 0, u''(1) = 0$.

9.2.2.3 Synthesis of the Case 2

This simulation was focused on solution of quite complicated ODE (Case 2) which come from civil engineering. Original solution obtained by means of Ritz or Galerkin method [19], based on Hilbert functional space that consisted of sinus functions and original solution, obtained by means of Ritz or Galerkin method, was according to [19] $u(x) = 1.243sin(\pi x) + 0.0116sin(2\pi x) + 0.00154sin(3\pi x)$.

AP with SOMA was used in two ways. In the first one SOMA was used to estimate only parameters a, b, c of founded $u(x)$, i.e. $u(x) = a\sin(\pi x) + b\sin(2\pi x) + c\sin(3\pi x)$. In the original solution all three coefficients were calculated by means of quite complicated Ritz or Galerkin method. SOMA was able to find all three coefficients as is depicted on Fig. 9.13. This problem was basically classical optimization because functions were a priori known. This simulation was 100 times repeated and in all cases has lead to the same results. The second use of SOMA here was not focused only on parameter estimation. AP with SOMA was used on complex set of functions $(sin, cos, \ldots)$ operators $(+, -, /, \ldots)$ and constants (a, b, c) to find their best combination i.e. to build up function fitting function u as well as possible. The best result are shown on Fig. 9.12. On Figs. 9.14 and 9.15 are snapshots of the temporary solutions recorded during evolution. Through such solutions has AP/SOMA passed through all to the best one.

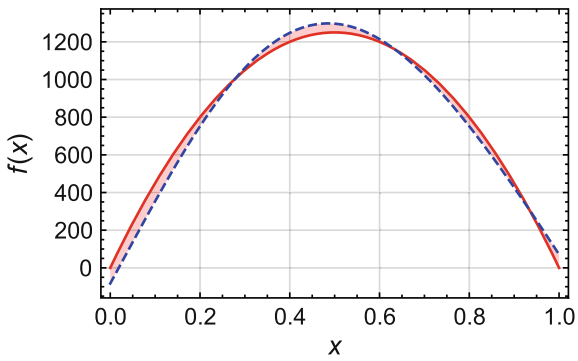

Fig. 9.12 Acceptable solution by AP (*blue dotted curve*) $u(x) = 1.243\sin(\pi x) + 0.0116\sin(2\pi x)$

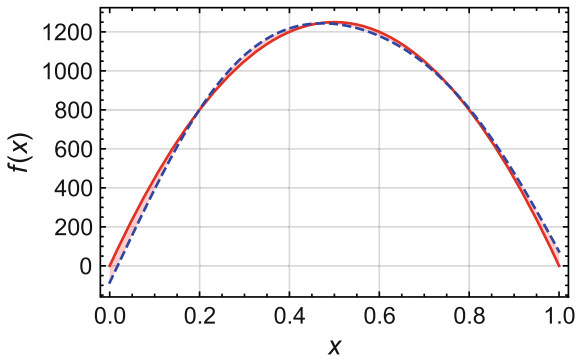

Fig. 9.13 The best solution (*blue dotted curve*) $u(x) = 1.23413\sin(\pi x) + 0.00922643\sin(2\pi x) + 0.000997829\sin(3\pi x)$ obtained during evolution

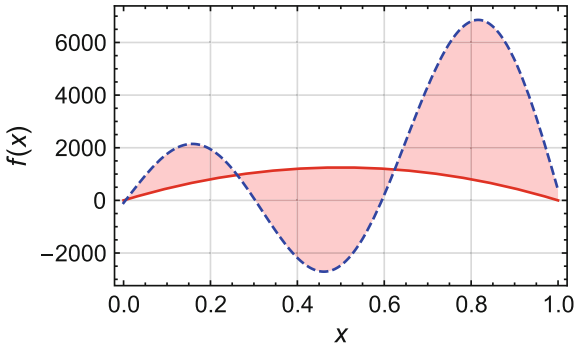

Fig. 9.14 Temporary solution (*blue dotted curve*) $u(x) = 1.243\sin(\pi x) - 0.3\sin(2\pi x) + 0.1\sin(3\pi x)$ obtained during AP evolution

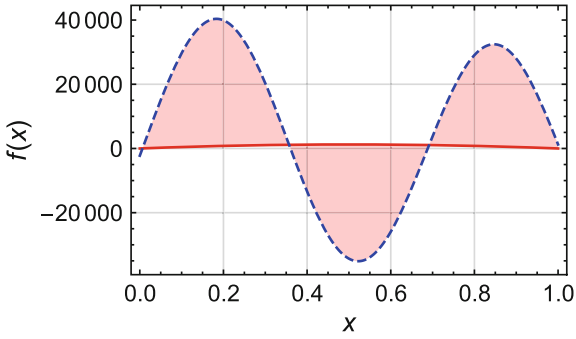

Fig. 9.15 Another temporary solution (*blue dotted curve*) $u(x) = 1.243\sin(\pi x) + 1\sin(2\pi x) + 1\sin(3\pi x)$ obtained during AP evolution

9.3 Conclusion

In this chapter we have demonstrated how different evolutionary approaches can be used in real life problems. Classical evolutionary algorithms as well as symbolic regression have been used in chemical reactor design and control. This system is represented by 5 nonlinear differential equations and evolution has been solving its geometrical design, chemical concentrations and also its control. In the second part we have introduced application of the symbolic regression on the unknown function (solution of the differential equation) synthesis and compared it with classical solution given by Ritz or Galerkin methods in functional spaces. As clearly visible from figures, all simulations show results with satisfactorily quality. This and also other results from the area of bio-inspiring algorithms clearly show that bio-inspired methods are definitely powerful and can be often used to solve very hard problems from various domains of science and/or technology.

Acknowledgments The following grants are acknowledged for the financial support provided for this research: Grant Agency of the Czech Republic–GACR P103/15/06700S, VSB-TU internal grant SGS 2016/175 and by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project "IT4Innovations excellence in science–LQ1602".

References

1. Back, T., Fogel, B., Michalewicz, Z.: Handbook of Evolutionary Computation. Institute of Physics, London (1997)
2. Barricelli, N.: Esempi numerici di processi di evoluzione. *Methodos*, pp. 45–68 (1954)
3. Davendra, D.D., Zelinka, I.: Self-Organizing Migrating Algorithm Methodology and Implementation. Springer, Heidelberg (2016)
4. Černý, V.: Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J. Opt. Theory Appl. **45**(1), 41–51 (1985)
5. Fogel, G., Corne, D.: Evolutionary Computation in Bioinformatics. Bioinformatics artificial intelligence. Morgan Kaufmann, Burlington (2003)
6. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
7. Holland, J.H.: Intelligent machinery, unpublished report for national physical laboratory (1975)
8. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology. Control and Artificial Intelligence. MIT Press, Cambridge (1992)
9. Hildebrandt, D., Hopley, F., Glasser, D.: Optimal reactor structures for exothermic reversible-reactions with complex kinetics. Chem. Eng. Sci. **51**(10), 1533–2520 (1996)
10. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
11. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
12. Koza, J.R., Andre, D., Bennett, F. H., Keane, M.A.: Genetic Programming III: Darwinian Invention and Problem Solving, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco (1999)
13. Luyben, W.L.: Chemical Reactor Design and Control. Wiley-Interscience, 1 edn, (August 2007)
14. O'Neill, M., Brabazon, A.: Grammatical differential evolution. In: Arabnia, H.R (ed.) Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006, vol. 1, pp. 231–236, CSREA Press, Las Vegas, Nevada, USA (2006)
15. O'Neill, Michael, Ryan, Conor: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Norwell (2003)
16. Oplatková, Z., Zelinka, I.: Investigation on artificial ant using analytic programming. In Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, pp. 949–950, ACM, New York, NY, USA (2006)
17. Perry, R.H., Green, D.W. (eds): Perry's Chemical Engineering Handbook. 6th edn. McGraw-Hill, New York (1984)
18. Rechenberg, I.: Evolutionsstrategie: optimierung technischer systeme nach prinzipien der biologischen evolution. Frommann-Holzboog (1973)
19. Rektorys, K.: Variational methods in Engineering Problems and Problems of Mathematical Physics, vol. 1. Academia, Prague (1999)
20. Rafal, S., Jürgen, S.: Probabilistic incremental program evolution. Evol. Comput. **5**(2), 123–141 (June 1997)
21. Schwefel, H.P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. ISR, vol. 26. Birkhaeuser, Basel/Stuttgart (1977)
22. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Opt. **11**(4), 341–359 (Dec 1997)
23. Weisser, R., Osmera, P.: Two-level tranpslant evolution. In Proceedings of the 17th Zittau Fuzzy Colloquium (2010)
24. Weisser, R., Osmera, P., Matousek, R.: Transplant evolution with modified schema of differential evolution : optimization structure of controllers. In Proceedings of the International Conference on Soft Computing, MENDEL, Brno, Czech Republic (2010)
25. Zelinka, I.: Soma-self organizing migrating algorithm. In: Onwubolu, G.C., Babu, B. (eds.) New Optimization Techniques in Engineering, Springer, New York, pp. 167–218 (2004). ISBN 3-540-20167X

26. Zelinka, I., Celikovský, S., Richter, H., Chen, G. (eds.): Evolutionary Algorithms and Chaotic Systems. Studies in Computational Intelligence, vol. 267. Springer, Heidelberg (2010)
27. Zelinka, I., Davendra, D., Senkerik, R., Jasek, R., Oplatkova, Z.: Analytical Programming-a Novel Approach for Evolutionary Synthesis of Symbolic Structures. InTech (2011)
28. Zelinka, I., Davendra, D.D., Šenkeřík, R., Pluháček, M.: Investigation on evolutionary predictive control of chemical reactor. J. Appl. Log. **13**(2 Part A):156–166, 2015
29. Zelinka, I., Oplatkova, Z., Nolle, L.: Analytic programming-symbolic regression by means of arbitrary evolutionary algorithms. Int. J. Simul. Syst. Sci. Technol. **6**(9):44–56, aug 2005. Special Issue on: Intelligent Systems