Genet Program Evolvable Mach (2006) 7:33–54
DOI 10.1007/s10710-006-7009-y

Solving differential equations with genetic programming

I. G. Tsoulos · I. E. Lagaris

Received: 7 February 2005 / Revised: 7 November 2005

Abstract A novel method for solving ordinary and partial differential equations, based on grammatical evolution is presented. The method forms generations of trial solutions expressed in an analytical closed form. Several examples are worked out and in most cases the exact solution is recovered. When the solution cannot be expressed in a closed analytical form then our method produces an approximation with a controlled level of accuracy. We report results on several problems to illustrate the potential of this approach.

Keywords Grammatical evolution · Genetic programming · Differential equations · Evolutionary modeling

1. Introduction

A lot of problems in the fields of physics, chemistry, economics etc. can be expressed in terms of ordinary differential equations (ODE's) and partial differential equations (PDE's). Weather forecasting, quantum mechanics, wave propagation and stock market dynamics are some examples. For that reason many methods have been proposed for solving ODE's and PDE's such as Runge Kutta, Predictor–Corrector [1], radial basis functions [4] and feedforward neural networks [5]. Recently, methods based on genetic programming have also been proposed [6], as well as methods that induce the underlying differential equation from experimental data [7, 8]. The technique of genetic programming [2], is an optimization process based on the evolution of a large number of candidate solutions through genetic operations such as replication, crossover and mutation [14]. In this article we propose a novel method based on genetic programming. Our method attempts to solve ODE's and PDE's by generating solutions in a closed analytical form. Offering closed form analytical solutions is very helpful and highly desirable. Methods that offer such type of solutions have appeared in the past. We mention the Galerkin type of methods, methods based on neural networks [5],

Communicated by: Hitoshi Iba

I. G. Tsoulos · I. E. Lagaris (✉)
Department of Computer Science, University of Ioannina, P.O. Box 1186, Ioannina 45110, Greece
e-mail: lagaris@cs.voi.gr

etc. These methods choose a basis set of functions with adjustable parameters and proceed approximating the solution by varying these parameters. Our method offers closed form solutions, however the variety of the basis functions involved is not a priori determined, rather is constructed dynamically as the solution procedure proceeds and can be of high complexity if required. This last feature is the one that distinguishes our method from others. We have not dealt with the problem of differential equation induction from data. The generation is achieved with the help of grammatical evolution. We used grammatical evolution instead the "classic" tree based genetic programming, because grammatical evolution can produce programs in an arbitrary language, the genetic operations such as crossover and mutation are faster and also because it is far more convenient to symbolically differentiate mathematical expressions. The code production is performed using a mapping process governed by a grammar expressed in Backus Naur Form. Grammatical evolution has been applied successfully to problems such as symbolic regression [11], discovery of trigonometric identities [12], robot control [15], caching algorithms [16], financial prediction [17] etc. The rest of this article is organized as follows: in Section 2 we give a brief presentation of grammatical evolution, in Section 3 we describe in detail the new algorithm, in Section 4 we present several experiments and in Section 5 we present our conclusions and ideas for further work.

2. Grammatical evolution

Grammatical evolution is an evolutionary algorithm that can produce code in any programming language. The algorithm requires as inputs the BNF grammar definition of the target language and the appropriate fitness function. Chromosomes in grammatical evolution, in contrast to classical genetic programming [2], are not expressed as parse trees, but as vectors of integers. Each integer denotes a production rule from the BNF grammar. The algorithm starts from the start symbol of the grammar and gradually creates the program string, by replacing non terminal symbols with the right hand of the selected production rule. The selection is performed in two steps:

- We read an element from the chromosome (with value V).
- We select the rule according to the scheme

$$\text{Rule} = V \bmod \text{NR} \tag{1}$$

where NR is the number of rules for the specific non-terminal symbol. The process of replacing non terminal symbols with the right hand of production rules is continued until either a full program has been generated or the end of chromosome has been reached. In the latter case we can reject the entire chromosome or we can start over (wrapping event) from the first element of the chromosome. In our approach we allow at most two wrapping events to occur.

In our method we used a small part of the C programming language grammar as we can see in Fig. 1. The numbers in parentheses denote the sequence number of the corresponding production rule to be used in the selection procedure described above.

The symbol S in the grammar denotes the start symbol of the grammar. For example, suppose we have the chromosome $x = [16, 3, 7, 4, 10, 28, 24, 1, 2, 4]$. In Table 1 we show how a valid function is produced from x. The resulting function in the above example is $f(x) = \log(x^2)$. Further details about grammatical evolution can be found in [9, 10, 11, 13].

Fig. 1 The grammar of the proposed method

```
S::=<expr> (0)
<expr> ::=  <expr> <op> <expr>  (0)
          | ( <expr> )          (1)
          | <func> ( <expr> )   (2)
          |<digit>              (3)
          |x                    (4)
          |y                    (5)
          |z                    (6)
<op> ::=     +      (0)
          | -       (1)
          | *       (2)
          | /       (3)
<func> ::=   sin  (0)
          |cos  (1)
          |exp  (2)
          |log  (3)
<digit>  ::= 0  (0)
          | 1 (1)
          | 2 (2)
          | 3 (3)
          | 4 (4)
          | 5 (5)
          | 6 (6)
          | 7 (7)
          | 8 (8)
          | 9 (9)
```

Table 1 Example of program construction

String	Chromosome	Operation
<expr>	16, 3, 7, 4, 10, 28, 24, 1,2,4	16 mod 7 = 2
<func>(<expr>)	3, 7, 4, 10, 28, 24, 1, 2, 4	3 mod 4 = 3
log(<expr>)	7, 4, 10, 28, 24, 1, 2, 4	7 mod 7 = 0
log(<expr><op><expr>)	4, 10, 28, 24, 1, 2, 4	4 mod 7 = 4
log(x<op><expr>)	10, 28, 24, 1, 2, 4	10 mod 4 = 2
log(x*<expr>)	28, 24, 1, 2, 4	28 mod 7 = 0
log(x*<expr><op><expr>)	24, 1, 2, 4	24 mod 7 = 3
log(x*<digit><op><expr>)	1, 2, 4	1 mod 10 = 1
log(x*1<op><expr>)	2, 4	2 mod 4 = 2
log(x*1*<expr>)	4	4 mod 7 = 4
log(x*1*x)		

3. Description of the algorithm

To solve a given differential equation the proper boundary/initial conditions must be stated. The algorithm has the following phases:

1. Initialization.
2. Fitness evaluation.
3. Genetic operations.
4. Termination control.

3.1. Initialization

In the initialization phase the values for mutation rate and replication rate are set. The replication rate denotes the fraction of the number of chromosomes that will go through unchanged to the next generation (replication). That means that the probability for crossover is set to 1-replication rate. The mutation rate controls the average number of changes inside a chromosome.

3.2. Fitness evaluation

3.2.1. ODE case

We express the ODE's in the following form:

$$f\left(x, y, y^{(1)}, ..., y^{(n-1)}, y^{(n)}\right) = 0, \quad x \in [a, b] \tag{2}$$

where $y^{(n)}$ denotes the n-order derivative of y. Let the boundary or initial conditions be given by:

$$g_i\left(x, y, y^{(1)}, \ldots, y^{(n-1)}\right)\Big|_{x=t_i} = 0, \quad i = 1, \ldots, n$$

where t_i is one of the two endpoints a or bThe steps for the fitness evaluation of the population are the following:

1. Choose N equidistant points $(x_0, x_1, \ldots, x_{N-1})$ in the relevant range.
2. For every chromosome i

   (a) Construct the corresponding model $M_i(x)$, expressed in the grammar described earlier.
   (b) Calculate the quantity

$$E(M_i) = \sum_{j=0}^{N-1} \left(f\left(x_j, M_i^0(x_j), \ldots, M_i^{(n)}(x_j)\right)\right)^2 \tag{3}$$

   (c) Calculate an associated penalty $P(M_i)$ as shown below.
   (d) Calculate the fitness value of the chromosome as:

$$v_i = E(M_i) + P(M_i) \tag{4}$$

The penalty function P depends on the boundary conditions and it has the form:

$$P(M_i) = \lambda \sum_{k=1}^{n} g_k^2\left(x, M_i, M_i^{(1)}, \ldots, M_i^{(n-1)}\right)\Big|_{x=t_k} \tag{5}$$

where λ is a positive number.

3.2.2. SODE case

The proposed method can solve systems of ordinary differential equations that are expressed in the form:

$$\begin{pmatrix} f_1\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) = 0 \\ f_2\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) = 0 \\ \vdots \qquad\qquad \vdots\ \vdots \\ f_k\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) = 0 \end{pmatrix}, \quad x \in [a, b] \tag{6}$$

with initial conditions:

$$\begin{pmatrix} y_1(a) = y_{1a} \\ y_2(a) = y_{2a} \\ \vdots\ \ \vdots\ \ \vdots \\ y_k(a) = y_{ka} \end{pmatrix} \tag{7}$$

The steps for the fitness evaluation are the following:

1. Choose N equidistant points $(x_0, x_1, ..., x_{N-1})$ in the relevant range.
2. For every chromosome i

(a) Split the chromosome uniformly in k parts, where k is the number of equations in the system.
(b) Construct the k models M_{ij}, $j = 1,\ldots, k$
(c) Calculate the quantities

$$E(M_{ij}) = \sum_{l=0}^{N-1} \left(f_j(x_l, M_{i1}(x_l)), M_{i1}^{(1)}(x_l), M_{i2}(x_l), M_{i2}^{(1)}(x_l), \ldots, M_{ik}(x_l), M_{ik}^{(1)}(x_l) \right)^2,$$
$$\forall j = 1, \ldots, k$$

(d) Calculate the quantity

$$E(M_i) = \sum_{j=1}^{k} \left(E(M_{ij}) \right) \tag{8}$$

(e) Calculate the associated penalties

$$P(M_{ij}) = \lambda \left(M_{ij}(a) - y_{ja} \right)^2, \quad \forall j = 1, \ldots, k \tag{9}$$

where λ is a positive number.
(f) Calculate the total penalty value

$$P(M_i) = \sum_{j=1}^{k} (P(M_{ij})) \tag{10}$$

(g) Finally, the fitness of the chromosome i is given by:

$$u_i = E(M_i) + P(M_i) \tag{11}$$

3.2.3. PDE case

We only consider here elliptic PDE's in two and three variables with Dirichlet boundary conditions. The generalization of the process to other types of boundary conditions and higher dimensions is straightforward. The PDE is expressed in the form:

$$f\left(x, y, \Psi(x, y), \frac{\partial}{\partial x}\Psi(x, y), \frac{\partial}{\partial y}\Psi(x, y), \frac{\partial^2}{\partial x^2}\Psi(x, y), \frac{\partial^2}{\partial y^2}\Psi(x, y)\right) = 0 \tag{12}$$

with $x \in [x_0, x_1]$ and $y \in [y_0, y_1]$. The associated Dirichlet boundary conditions are expressed as: $\Psi(x_0, y) = f_0(y)$, $\Psi(x_1, y) = f_1(y)$, $\Psi(x, y_0) = g_0(x)$, $\Psi(x, y_1) = g_1(x)$.

The steps for the fitness evaluation of the population are given below:

1. Choose N^2 equidistant points in the box $[x_0,x_1] \times [y_0,y_1]$, N_x equidistant points on the boundary at $x = x_0$ and at $x = x_1$, N_y equidistant points on the boundary at $y = y_0$ and at $y = y_1$.
2. For every chromosome i
   - Construct a trial solution $M_i(x, y)$ expressed in the grammar described earlier.
   - Calculate the quantity

$$E(M_i) = \sum_{j=1}^{N^2} f\left(x_j, y_j, M_i(x_j, y_j), \frac{\partial}{\partial x}M_i(x_j, y_j), \frac{\partial}{\partial y}M_i(x_j, y_j), \frac{\partial^2}{\partial x^2}M_i(x_j, y_j), \frac{\partial^2}{\partial y^2}M_i(x_j, y_j)\right)^2$$

   - Calculate the quantities

$$P_1(M_i) = \sum_{j=1}^{N_x} \big(M_i(x_0, y_j) - f_0(y_j)\big)^2$$

$$P_2(M_i) = \sum_{j=1}^{N_x} \big(M_i(x_1, y_j) - f_1(y_j)\big)^2$$

$$P_3(M_i) = \sum_{j=1}^{N_y} \big(M_i(x_j, y_0) - g_0(x_j)\big)^2$$

$$P_4(M_i) = \sum_{j=1}^{N_y} \big(M_i(x_j, y_1) - g_1(x_j)\big)^2$$

   - Calculate the fitness of the chromosome as:

$$v_i = E(M_i) + \lambda(P_1(M_i) + P_2(M_i) + P_3(M_i) + P_4(M_i)) \tag{13}$$

3.2.4. A complete illustrative example

Consider the ODE

$$y'' + 100y = 0, \quad x \in [0, 1]$$

with the boundary conditions $y(0) = 0$ and $y'(0) = 10$. We take in the range [0, 1] $N = 10$ equidistant points $x_0,\ldots, x_9$. Suppose that we have chromosomes with length 10 and one chromosome is the array: $g = [7, 2, 10, 4, 4, 2, 11, 20, 30, 5]$. The function which corresponds to the chromosome g is $M_g(x) = \exp(x) + \sin(x)$. The first order derivative is $M_g^{(1)(x)} = \exp(x) + \cos(x)$ and the second order derivative is $M_g^{(2)(x)} = \exp(x) - \sin(x)$. The symbolic computation of the above quantities is described in detail in the Section 2. Using the Eq. (3) we have:

$$\begin{aligned} E(M_g) &= \sum_{i=0}^{9} \left(M_g^{(2)}(x_i) + 100M_g(x_i)\right)^2 \\ &= \sum_{i=0}^{9} (101\exp(x_i) + 99\sin(x_i))^2 \\ &= 4849332.4 \end{aligned}$$

The penalty function $P(M_g)$ is calculated following the Eq. (5) as:

$$\begin{aligned} P(M_g) &= \lambda\left((M_g(0) - y(0))^2 + \left(M_g^{(1)}(0) - \right)^2\right) \\ &= \lambda((\exp(0) + \sin(0) - 0)^2 + (\exp(0) - \sin(0) - 10)^2) \\ &= \lambda((1 + 0 - 0)^2 + (1 - 0 - 10)^2) \\ &= 82\lambda \end{aligned}$$

So, the fitness value u_g of the chromosome is given by:

$$\begin{aligned} u_g &= E(M_g) + P(M_g) \\ &= 4849332.4 + 82\lambda \end{aligned}$$

We perform the above procedure to all chromosomes in the population and we sort them in ascending order according to their fitness value. In consequence, we apply the genetic operators, the new population is created and the process is repeated until the termination criteria are met.

3.3. Evaluation of derivatives

Derivatives are evaluated together with the corresponding functions using an additional stack and the following differentiation elementary rules, adopted by the various Automatic Differentiation Methods [21] and used in corresponding tools [18–20]:

1. $(f(x) + g(x))' = f'(x) + g'(x)$

2. $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$
3. $\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x)-g'(x)f(x)}{g^2(x)}$
4. $f(g(x))' = g'(x)f'(g(x))$

To find the first derivative of a function we use two different stacks, the first is used for the function value and the second for the derivative value. For instance consider that we want to estimate the derivative of the function $f(x) = \sin(x) + \log(x + 1)$. Suppose that S_0 is the stack for the function's value and S_1 is the stack for the derivative. The function $f(x)$ in postfix order is written as "x sin x 1 + log + ". We begin to read from left to right, until we reach the end of the string. The following calculations are performed in the stacks S_0 and S_1. We denote with (a_0,a_1, $\cdots$, a_n) the elements in a stack, a_n being the element at the top.

1. $S_0 = (x)$, $S_1 = (1)$
2. $S_0 = (\sin(x))$, $S_1 = (1 \cos(x))$
3. $S_0 = (\sin(x), x)$, $S_1 = (1 \cos(x), 1)$
4. $S_0 = (\sin(x), x, 1)$, $S_1 = (1 \cos(x), 1, 0)$
5. $S_0 = (\sin(x), x + 1)$, $S_1 = (1 \cos(x), 1 + 0)$
6. $S_0 = (\sin(x), \log(x + 1))$, $S_1 = (1 \cos(x), \frac{1+0}{x+1})$
7. $S_0 = (\sin(x) + \log(x + 1))$, $S_1 = (1 \cos(x) + \frac{1+0}{x+1})$

The S_1 stack contains the first derivative of $f(x)$. To extend the above calculations for the second order derivative, a third stack must be employed.

3.4. Genetic operations

3.4.1. Genetic operators

The genetic operators that are applied to the genetic population are the initialization, the crossover and the mutation.

The initialization is applied only once on the first generation. For every element of each chromosome a random integer in the range [0..255] is selected.

The crossover is applied every generation in order to create new chromosomes from the old ones, that will replace the worst individuals in the population. In that operation for each couple of new chromosomes two parents are selected, we cut these parent-chromosomes at a randomly chosen point and we exchange the right-hand-side sub-chromosomes, as shown in Fig. 2.

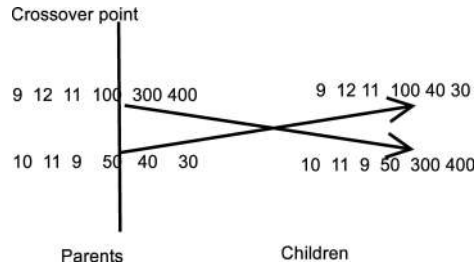

Fig. 2 One-point crossover

The parents are selected via tournament selection, i.e.:

- First, we create a group of $K \geq 2$ randomly selected individuals from the current population.
- The individual with the best fitness in the group is selected, the others are discarded.

The final genetic operator used is the mutation, where for every element in a chromosome a random number in the range [0,1] is chosen. If this number is less than or equal to the mutation rate the corresponding element is changed randomly, otherwise it remains intact.

3.4.2. Application of genetic operators

In every generation the following steps are performed:

1. The chromosomes are sorted with respect to their fitness value, in a way that the best chromosome is placed at the beginning of the population and the worst at the end.
2. $c = (1 - s) * g$ new chromosomes are produced by the crossover operation, where s is the replication rate of the model and g is the total number of individuals in the population. The new individuals will replace the worst ones in the population at the end of the crossover.
3. The mutation operation is applied to every chromosome excluding those which have been selected for replication in the next generation.

3.5. Termination control

The genetic operators are applied to the population creating new generations, until a maximum number of generations is reached or the best chromosome in the population has fitness better than a preset threshold.

4. Experimental results

We describe several experiments performed on linear and non linear first and second order ODE's and systems and PDE's in two and three dimensions. In addition we applied our method to ODE's that do not posses an analytical closed form solution and hence can not be represented exactly by the grammar. For the case of systems of ODE's, each chromosome is split uniformly in M parts, where M is the number of equations in the system. Each part of the chromosome represents the solution of the corresponding ODE. We used 10% for the replication rate (hence the crossover probability is set to 90%) and 5% for the mutation rate. We investigated the importance of these two parameters by performing experiments using sets of different values. Each experiment was performed 30 times and we plot the average number of generations for the ODE7 problem in Fig. 3. As one can see the performance is somewhat dependent on these parameters, but not critically. The population size was set to 1000 and the length of each chromosome to 50. The size of the population is a critical parameter. Too small a size weakens the method's effectiveness. Too big a size renders the method slow. Hence since there is no first principals estimation for the the population size, we resorded to an experimental approach to obtain a realistic determination of its range. It turns out that values in the interval [200, 1000] are proper. We used fixed-length chromosomes instead of variable-length to avoid creation of unnecessary large chromosomes which will render the method inefficient. The length of the chromosomes is usually depended on the problem to be solved. For the case of simple ODE's a length between 20 and 50 is usually sufficient, while for the case of SODE's and PDE's where the chromosome must be

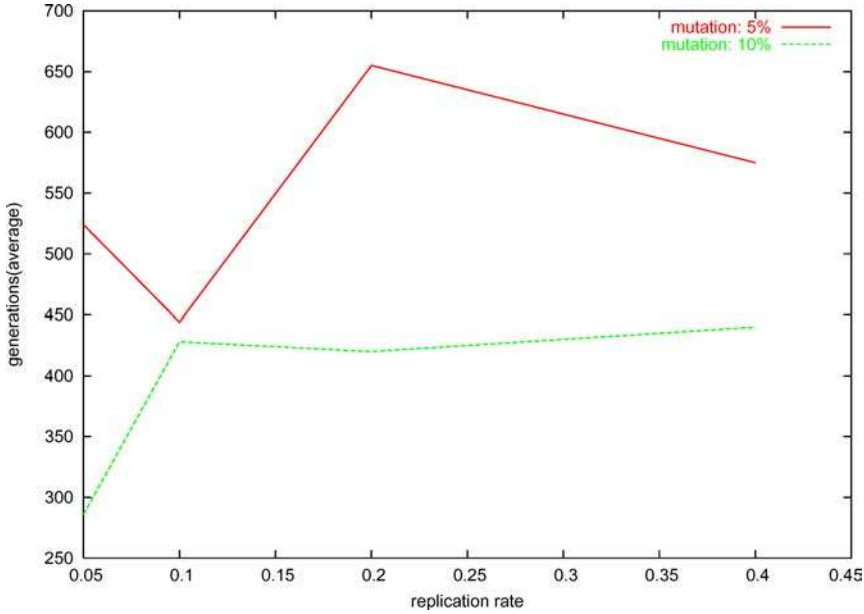

Fig. 3 Average number of generations versus the replication rate for two values of the mutation rate (problem ODE7)

split into parts, the length must be increased accordingly. The experiments were performed on an AMD ATHLON 2400+ running Slackware Linux 9.1 The penalty parameter λ in the penalty function was set to 100 in all runs, except for PDE cases where the value $\lambda = 1$ was sufficient. The maximum number of generations allowed was set to 2000 and the preset fitness target for the termination criteria was 10^{-7}. From the conducted experiments, we have observed that the maximum number of generations allowed must be greater for difficult problems such as SODE's and PDE's than for simpler problems of ODE's. The value of N for ODE's was between 10 and 20 depending on the problem. For PDE's N was set to 5 (i.e. $5^2 = 25$ points) and $N_x = N_y = 50$. We tried to select the value for N and N_x so as to minimize the computational effort without sacrificing quality. The results were compared against the results from the DSolve procedure of the Mathematica program version 5.0 running on Mandrake Linux v10.0. According to the manual of Mathematica the DSolve procedure can solve linear ordinary differential equations with constant coefficients. Also, it can solve many linear equations up to second order with non-constant coefficients and it includes general procedures that handle a large fraction of the nonlinear ODE's whose solutions are given in standard reference books such as Kamke [22]. The evaluation of the derived functions was performed using the FunctionParser programming library [3].

4.1. Linear ODE's

In the present subsection we present the results from first and second order linear ODE's. The proposed was applied in each Eq. (30) times and in every experiment the analytical solution was found.

ODE1

$$y' = \frac{2x - y}{x}$$

with $y(0) = 20.1$ and $x \in [0.1, 1.0]$. The analytical solution is $y(x) = x + \frac{2}{x}$.

ODE2

$$y' = \frac{1 - y\cos(x)}{\sin(x)}$$

with $y(0.1) = \frac{2.1}{\sin(0.1)}$ and $x \in [0.1,1]$. The analytical solution is $y(x) = \frac{x+2}{\sin(x)}$.

ODE3

$$y' = -\frac{1}{5}y + \exp(-\frac{x}{5})\cos(x)$$

with $y(0) = 0$ and $x \in [0, 1]$. The analytical solution is $y(x) = \exp(-\frac{x}{5})\sin(x)$.

ODE4

$$y'' = -100y$$

with $y(0) = 0$ and $y'(0) = 10$ and $x \in [0, 1]$. The analytical solution is $y(x) = \sin(10x)$.

ODE5

$$y'' = 6y' - 9y$$

with $y(0) = 0$ and $y'(0) = 2$ and $x \in [0, 1]$. The analytical solution is $y(x) = 2x\exp(3x)$.

ODE6

$$y'' = -\frac{1}{5}y' - y - \frac{1}{5}\exp\left(-\frac{x}{5}\right)\cos(x)$$

with $y(0) = 0$ and $y'(0) = 1$ and $x \in [0, 2]$. The analytical solution is $y(x) = \exp(-\frac{x}{5})\sin(x)$.

ODE7

$$y'' = -100y$$

with $y(0) = 0$ and $y(1) = \sin(10)$ and $x \in [0, 1]$. The analytical solution is $y(x) = \sin(10x)$.

ODE8

$$xy'' + (1 - x)y' + y = 0$$

with $y(0) = 1$ and $y(1) = 0$ and $x \in [0, 1]$. The analytical solution is $y(x) = 1 - x$.

ODE9

$$y'' = -\frac{1}{5}y' - y - \frac{1}{5}\exp\left(-\frac{x}{5}\right)\cos(x)$$

Table 2 Method results for linear ODE's

ODE	MIN	MAX	AVG
ODE1	8	1453	653
ODE2	52	1816	742
ODE3	23	1598	705
ODE4	14	1158	714
ODE5	89	1189	441
ODE6	37	1806	451
ODE7	42	1242	444
ODE8	3	702	66
ODE9	59	1050	411

with $y(0) = 0$ and $y(1) = \frac{\sin(1)}{\exp(0.2)}$ and $x \in [0, 1]$. The analytical solution is $y(x) = \exp(-\frac{x}{5})\sin(x)$.

In Table 2 we list the results from the proposed method for the equations above. Under the ODE heading the equation label is listed. Under the headings MIN, MAX, AVG we list the minimum, maximum and average number of generations (in the set of 30 experiments) needed to recover the exact solution. The Mathematica subroutine DSolve has managed to find the analytical solution in all cases.

In Fig. 4 we plot the evolution of a trial solution for the fourth problem of Table 2. At generation 22 the fitness value was 4200.5 and the intermediate solution was:

$$y_{22}(x) = \sin((\sin(-\log(4)x((-\cos(\cos(\exp(7)))\exp(\cos(6))) - 5))))$$

At generation 27 the fitness value was 517.17 and the corresponding candidate solution was:

$$y_{27}(x) = \sin((\sin(-\log(4)x((-\cos(\cos(\sin(7)))\exp(\cos(6))) - 5))))$$

Finally, at generation 59 the problem was solved exactly.

4.2. Non-linear ordinary differential equations

In this subsection we present results from the application of the method to non-linear ordinary differential equations. In all the equations the method was applied 30 times and in every application the exact solution was found.

NLODE1

$$y' = \frac{1}{2y}$$

with $y(1) = 1$ and $x \in [1, 4]$. The exact solution is $y = \sqrt{x}$. Note that $\sqrt{x}$ does not belong to the basis set.

NLODE2

$$(y')^2 + \log(y) - \cos^2(x) - 2\cos(x) - 1 - \log(x + \sin(x)) = 0$$

with $y(1) = 1 + \sin(1)$ and $x \in [1, 2]$ The exact solution is $y = x + \sin(x)$.

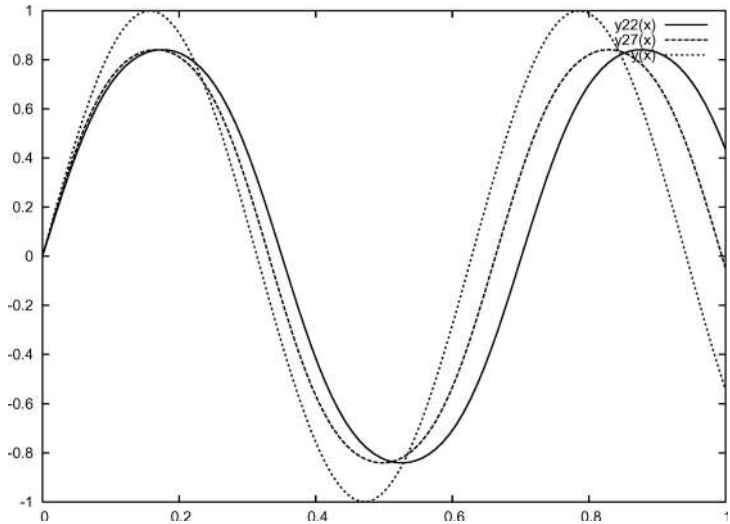

Fig. 4 Evolving candidate solutions of $y' = -100y$ with boundary conditions on the left

NLODE3

$$y''y' = -\frac{4}{x^3}$$

with $y(1) = 0$ and $x \in [1,2]$. The exact solution is $y = \log(x^2)$.

NLODE4

$$x^2y'' + (xy')^2 + \frac{1}{\log(x)} = 0$$

with $y(e) = 0,\ y'(e) = \frac{1}{e}$ and $x \in [e, 2e]$. The exact solution is $y(x) = \log(\log(x))$ and it was recovered at the 30th generation. In Table 3 we list results from the application of the method to the equations above. The meaning of the columns is the same as 2. The Mathematica subroutine DSolve has managed to find the exact solution only for NLODE1.

In Fig. 5 we plot intermediate trial solutions of the NLODE3.

At the second generation the trial solution had a fitness value of 73.512 and it assumed the form:

$$y_2(x) = \log(x - \exp(-x - 1)) - \cos(5)$$

Table 3 Method results for non-linear ODE's

NLODE	MIN	MAX	AVG
NLODE1	6	945	182
NLODE2	3	692	86
NLODE3	4	1564	191
NLODE4	6	954	161

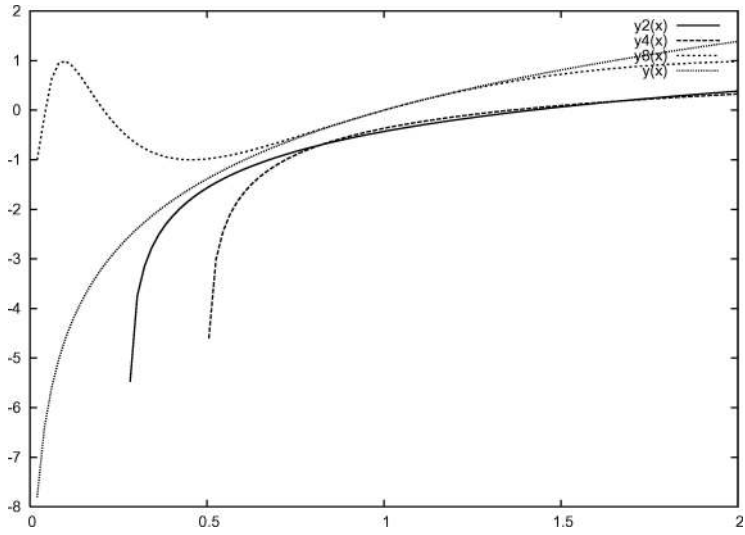

Fig. 5 Candidate solutions of third non-linear equation

while at the fourth generation it had a fitness value of 48.96 and it became:

$$y_4(x) = \log(\log(x + x))$$

Similarly at the 8th generation the fitness of the intermediate solution was 4.61 and its functional form was:

$$y_8(x) = \sin(\log(x * x))$$

The exact solution was obtained at the 9th generation.

4.3. Systems of ODE's

SODE1

$$y_1' = \cos(x) + y_1^2 + y_2 - (x^2 + \sin^2(x))$$
$$y_2' = 2x - x^2 \sin(x) + y_1 y_2$$

with $y_1(0) = 0, y_2(0) = 0$ and $x \in [0, 1]$. The exact solution is given by: $y_1 = \sin(x), y_2 = x^2$.

SODE2

$$y_1' = \frac{\cos(x) - \sin(x)}{y_2}$$
$$y_2' = y_1 y_2 + \exp(x) - \sin(x)$$

with $y_1(0) = 0, y_2(0) = 1$ and $x \in [0, 1]$. The exact solution is $y_1 = \frac{\sin(x)}{\exp(x)}$, $y_2 = \exp(x)$.

Table 4 Method results for non-linear ODE's

SODE	MIN	MAX	AVG
SODE1	6	1211	201
SODE2	15	1108	234
SODE3	30	1205	244
SODE4	5	630	75

SODE3

$$y_1' = \cos(x)$$
$$y_2' = -y_1$$
$$y_3' = y_2$$
$$y_4' = -y_3$$
$$y_5' = y4$$

with $y_1(0) = 0$, $y_2(0) = 1$, $y_3(0) = 0$,$y_4(0) = 1$, $y_5(0) = 0$ and $x \in [0, 1]$. The exact solutions is $y_1 = \sin(x)$, $y_2 = \cos(x)$, $y_3 = \sin(x)$, $y_4 = \cos(x)$, $y_5 = \sin(x)$.

SODE4

$$y_1' = -\frac{1}{y_2}\sin(\exp(x))$$
$$y_2' = -y_2$$

with $y_1(0) = \cos(1.0)$, $y_2(0) = 1.0$ and $x \in [0,1]$. The exact solution is $y_1 = \cos(\exp(x))$, $y_2 = \exp(-x)$. In Table 4 we list results from the application of the method to the equations above. The meaning of the columns is the same as 2. The Mathematica subroutine DSolve has managed to find the analytical solution only for SODE3.

4.4. An ODE without an analytical closed form solution

Example 1

$$y'' + \frac{1}{x}y' - \frac{1}{x}\cos(x) = 0$$

with $x \in [0,1]$ and $y(0) = 0$ and $y'(0) = 1$. With 20 points in$[0,1]$ we find:

$$\mathrm{GP1}(x) = x(\cos(-\sin(x/3 + \exp(-5 + x - \exp(\cos(x)))))))$$

with fitness value $2.1 * 10^{-6}$. The exact solution is :

$$y(x) = \int_0^x \frac{\sin(t)}{t} dt$$

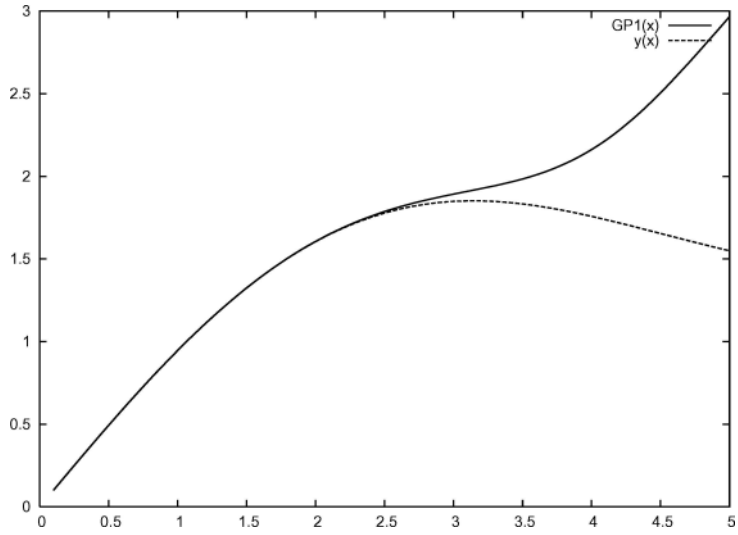

Fig. 6 Plot of GP1(x) and $y(x) = \int_0^x \frac{\sin(t)}{t} dt$

In Fig. 6, we plot the two functions in the range [0,5]

Example 2

$$y'' + 2xy = 0$$

with $x \in [0,1]$ and $y(0) = 0$ and $y'(0) = 1$. The exact solution is :

$$y(x) = \int_0^x \exp(-t^2) dt$$

With 20 points in [0,1] we find:

$$\text{GP2}(x) = \sin(\sin(x + \exp(\exp(x) \log(9)/ \exp(8 + \cos(1))/(\exp(7/ \exp(x)) + 6)))$$

with fitness $1.7 * 10^{-5}$. In Fig. 7 we the plot the two functions in the range [0, 5].

Observe, that even though the equations in the above examples were solved for $x \in [0, 1]$, the approximation maintains its quality beyond that interval, a fact that illustrates the unusual generalization ability.

4.5. A special case

Consider the ODE

$$y''(x^2 + 1) - 2xy - x^2 - 1 = 0$$

in the range [0, 1] and with initial conditions $y(0) = 0$ and $y'(0) = 1$. The analytical solution is $y(x) = (x^2 + 1)\arctan(x)$. Note that $\arctan(x)$ does not belong to the function repertoire of the method and this make the case special. The solution reached is not exact but approximate

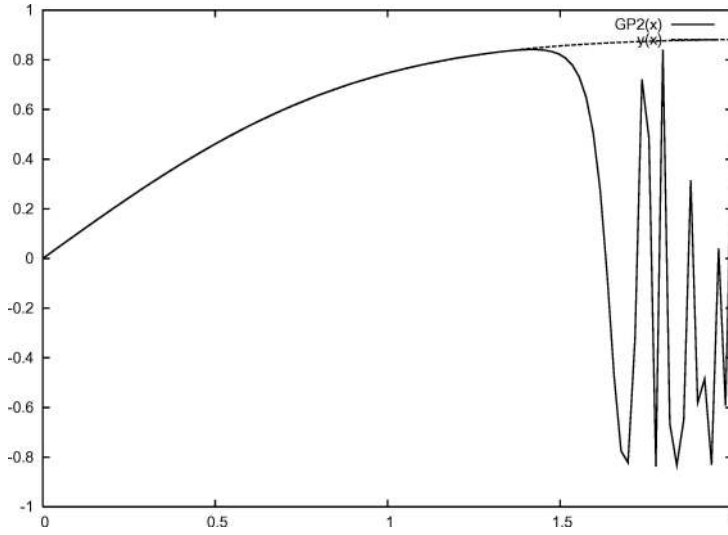

Fig. 7 Plot of GP2(x) and $y(x) = \int_0^x \exp(-t^2)dt$

given by:

$$\mathrm{GP}(x) = x/\sin(\exp(\cos(5/4/\exp(x)) - \exp((-\exp(((-((-\exp(\cos(\sin(2x))))))))))))$$

with fitness 0.0059168. Note that the subroutine DSolve of Mathematica failed to solve the above equation. In Fig. 8 we plot $z(x)$ (obtained using the method of Lagaris et al. [5]), $y(x) = (x^2 + 1)\arctan(x)$ (which is the exact solution) and the above solution GP(x). From Figs. 6–8 we observe the quality of the approximate solution even outside the training interval, hence rendering our method useful and practical.

4.6. PDE's

In this subsection we present results from the application of the method to elliptic partial differential equations. In all the equations the method was applied 30 times and in every application the exact solution was found.

PDE1

$$\nabla^2\Psi(x, y) = \exp(-x)(x - 2 + y^3 + 6y)$$

with $x \in [0, 1]$ and $y \in [0, 1]$ and boundary conditions: $\Psi(0, y) = y^3$, $\Psi(1, y) = (1 + y^3)\exp(-1)$, $\Psi(x, 0) = x\exp(-x)$, $\Psi(x, 1) = (x + 1)\exp(-x)$. The exact solution is $\Psi(x, y) = (x + y^3)\exp(-x)$.

PDE2

$$\nabla^2\Psi(x, y) = -2\Psi(x, y)$$

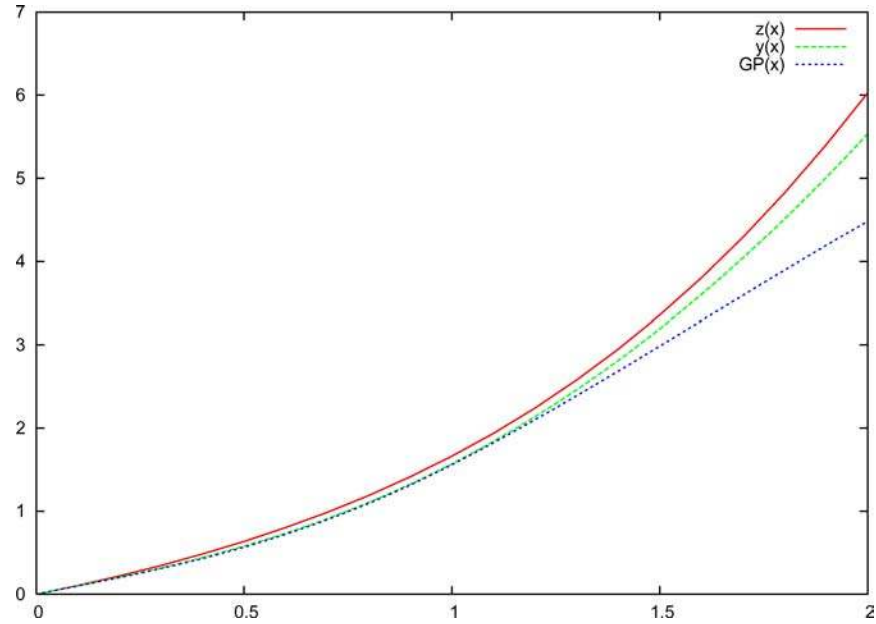

Fig. 8 $z(x), y(x) = (x^2 + 1)\arctan(x)$, GP($x$)

with $x \in [0, 1]$ and $y \in [0, 1]$ and boundary conditions: $\Psi(0, y) = 0, \Psi(1, y) = \sin(1)\cos(y)$, $\Psi(x, 0) = \sin(x), \Psi(x, 1) = \sin(x)\cos(1)$. The exact solution is $\Psi(x, y) = \sin(x)\cos(y)$.

PDE3

$$\nabla^2\Psi(x, y) = 4$$

with $x \in [0, 1]$ and $y \in [0, 1]$ and boundary conditions: $\Psi(0, y) = y^2 + y + 1$, $\Psi(1, y) = y^2 + y + 3$, $\Psi(x, 0) = x^2 + x + 1$, $\Psi(x, 1) = x^2 + x + 3$. The exact solution is $\Psi(x, y) = x^2 + y^2 + x + y + 1$.

PDE4

$$\nabla^2\Psi(x, y) = -(x^2 + y^2)\Psi(x, y)$$

with $x \in [0, 1]$ and $y \in [0, 1]$ and boundary conditions: $\Psi(x, 0) = 0$, $\Psi(x, 1) = \sin(x)$, $\Psi(0, y) = 0$, $\Psi(1, y) = \sin(y)$. The exact solution is $\Psi(x, y) = \sin(xy)$.

PDE5

$$\nabla^2\Psi(x, y) = (x - 2)\exp(-x) + x\exp(-y)$$

with $x \in [0, 1]$ and $y \in [0, 1]$ and boundary conditions: $\Psi(x, 0) = x(\exp(-x) + 1)$, $\Psi(x, 1) = x(\exp(-x) + \exp(-1))$, $\Psi(0, y) = 0$, $\Psi(1, y) = \exp(-y) + \exp(-1)$. The exact solution is $\Psi(x, y) = x(\exp(-x) + \exp(-y))$.

PDE6

Table 5 Method results for PDE's

PDE	MIN	MAX	AVG
PDE1	159	1772	966
PDE2	5	1395	203
PDE3	18	311	154
PDE4	4	1698	207
PDE5	195	945	444
PDE6	185	1579	797
PDE7	10	1122	325

The following is a highly non-linear pde:

$$\nabla^2\Psi(x, y) + \exp(\Psi(x, y)) = 1 + x^2 + y^2 + \frac{4}{(1 + x^2 + y^2)^2}$$

with $x \in [-1, 1]$ and $y \in [-1, 1]$ and boundary conditions: $f(0, y) = \log(1 + y^2)$, $f(1, y) = \log(2 + y^2)$, $g(x, 0) = \log(1 + x^2)$ and $g(x, 1) = \log(2 + x^2)$. The exact solution is $\Psi(x, y) = \log(1 + x^2 + y^2)$.

PDE7

$$\nabla^2\Psi(x, y, z) = 6$$

with $x \in [0, 1]$ and $y \in [0, 1]$ and $z \in [0, 1]$ and boundary conditions: $\Psi(0, y, z) = y^2 + z^2$, $\Psi(1, y, z) = y^2 + z^2 + 1$, $\Psi(x, 0, z) = x^2 + z^2$, $\Psi(x, 1, z) = x^2 + z^2 + 1$, $\Psi(x, y, 0) = x^2 + y^2$, $\Psi(x, y, 1) = x^2 + y^2 + 1$. The exact solution is $\Psi(x, y, z) = x^2 + y^2 + z^2 + 1$.

In Table 5 we list results from the application of the method to the equations above. The meaning of the columns is the same as 2. The Mathematica subroutine DSolve has not managed to find the exact solution for any of the examples above.

In the following we present some graphs for trial solutions of the second PDE. At generation 1 the trial solution was

$$\mathrm{GP1}(x, y) = \frac{x}{7}$$

with fitness value 8.14. The difference between the trial solution GP1(x,y) and the exact solution $\Psi(x,y)$ is shown in Fig. 9. At the 10$^{\text{th}}$generation the trial solution was

$$\mathrm{GP10}(x, y) = \sin(x/3 + x)$$

with fitness value 3.56. The difference between the trial solution GP10(x,y) and the exact solution $\Psi(x,y)$ is shown in Fig.10.
At the 40th generation the trial solution was

$$\mathrm{GP40}(x) = \sin(\cos(y)x)$$

with fitness value 0.59. The difference between the trial solution GP40(x, y) and the exact solution $\Psi(x,y)$ is shown in Fig. 11.

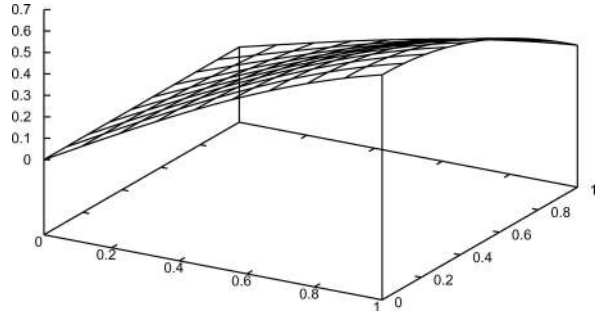

Fig. 9 Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and GP1(x,y)

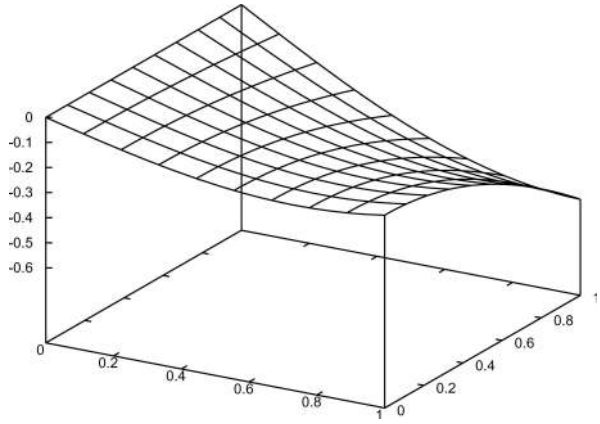

Fig. 10 Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and GP10(x,y)

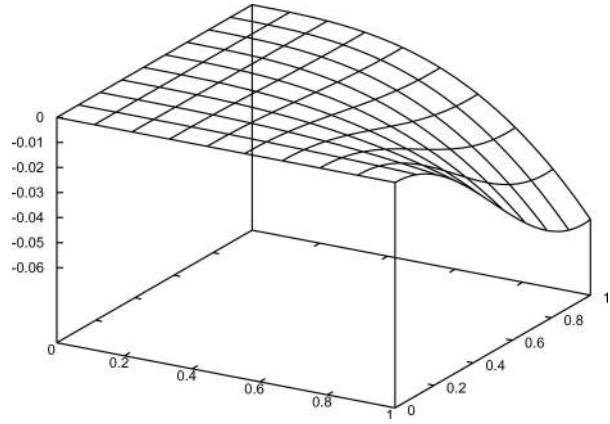

Fig. 11 Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and GP40(x,y)

5. Conclusions and further work

We presented a novel approach for solving ODE's and PDE's. The method is based on genetic programming. This approach creates trial solutions and seeks to minimize an associated error. The advantage is that our method can produce trial solutions of highly versatile functional form. Hence the trial solutions are not restricted to a rather inflexible form that is imposed frequently by basis-set methods that rely on completeness. If the grammar has a rich function

repertoire, and the differential equation has a closed form solution, it is very likely that our method will recover it. If however the exact solution can not be represented in a closed form, our method will produce a closed form approximant.

The grammar used in this article can be further developed and enhanced. For instance it is straight forward to enrich the function repertoire or even to allow for additional operations. Treating different types of PDE's with the appropriate boundary conditions is a topic of current interest and is being investigated. Our preliminary results are very encouraging.

References

1 J. D. Lambert, Numerical methods for Ordinary Differential Systems: The Initial Value Problem, John Wiley & Sons: Chichester, England, 1991.
2 J. R. Koza, Genetic Programming: On the Programming of Computer by Means of Natural Selection. MIT Press: Cambridge, MA, 1992.
3 J. Nieminen and J. Yliluoma, "Function Parser for C++, v2.7", available from http://www.students.tut.fi/˜warp/FunctionParser/.
4 G. E. Fasshauer, "Solving differential equations with radial basis functions: Multilevel methods and smoothing," Advances in Computational Mathematics, vol. 11, nos. 2–3, pp. 139–159, 1999.
5 I. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," IEEE Transactions on Neural Networks, vol. 9, no. 5, pp. 987–1000, 1998.
6 G. Burgess, "Finding approximate analytic solutions to differential equations using genetic programming," Surveillance Systems Division, Electronics and Surveillance Research Laboratory, Department of Defense, Australia, 1999.
7 H. Cao, L. Kang, Y. Chen, and J. Yu, Evolutionary modeling of systems of ordinary differential equations with genetic programming, Genetic Programming and Evolvable Machines, vol. 1, pp. 309–337, 2000.
8 H. Iba and E. Sakamoto,: "Inference of differential equation models by genetic programming," Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002), 2002, pp. 788–795.
9 M. O' Neill, Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution. PhD thesis, University Of Limerick, Ireland, August 2001.
10 C. Ryan, J. J. Collins, and M. O' Neill, Evolving programs for an arbitrary language, in Proceedings of the First European Workshop on Genetic Programming, volume 1391 of LNCS, W. Banzhaf, Ri. Poli, M. Schoenauer, and T. C. Fogarty, (eds.), Springer-Verlag, pp. 83–95, Paris, 14–15 April 1998.
11 M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers, 2003.
12 C. Ryan, M. O'Neill, and J. J. Collins, "Grammatical evolution: Solving trigonometric identities," in Proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets., Brno, Czech Republic, June 24–26 1998. Technical University of Brno, Faculty of Mechanical Engineering, pp. 111–119.
13 M. O'Neill and C. Ryan, "Grammatical evolution," IEEE Trans. Evolutionary Computation, vol. 5, pp. 349–358, 2001.
14 D. E. Goldberg, Genetic algorithms in search, Optimization and Machine Learning, Addison Wesley, 1989.
15 J. J. Collins and C. Ryan, "Automatic generation of robot behaviors using grammatical evolution," in Proc. of AROB 2000, the Fifth International Symposium on Artificial Life and Robotics.
16 M. O'Neill, J. J. Collins, and C. Ryan, "Automatic generation of caching algorithms," in Evolutionary Algorithms in Engineering and Computer Science, Kaisa Miettinen, Marko M. Mkel, Pekka Neittaanmki, and Jacques Periaux (eds.), Jyvskyl, Finland, 30 May–3 June 1999, John Wiley & Sons, pp. 127–134, 1999.
17 A. Brabazon and M. O'Neill, "A grammar model for foreign-exchange trading," in Proceedings of the International conference on Artificial Intelligence, volume II, H. R. Arabnia et al. (eds.), CSREA Press, 23–26 June 2003, pp. 492–498, 2003.
18 P. Cusdin and J. D. Muller, "Automatic differentiation and sensitivity analysis methods for CFD," QUB School of Aeronautical Engineering, 2003.
19 O. Stauning, "Flexible automatic differentiation using templates and operator overloading in C++," Talk presented at the Automatic Differentiation Workshop at Shrivenham Campus, Cranfield University, June 6, 2003.

20 C. Bischof, A. Carle, G. Corliss, and A. Griewank, "ADIFOR - generating derivative codes from fortran programs," Scientific Programming, no. 1, pp. 1–29, 1992.
21 A. Griewank, "On automatic differentiation" in Mathematical programming: Recent Developments and Applications, M. Iri and K. Tanabe (eds.), Kluwer Academic Publishers, Amsterdam, pp. 83–108, 1989.
22 E. Kamke, Differential Equations. Solution Methods and Solutions, Teubner, Stuttgart, 1983.