Solving differential equations with grammatical evolution

I. G. Tsoulos, I. E. Lagaris*

Department of Computer Science, University of Ioannina
P.O. Box 1186, Ioannina 45110 - Greece

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, 7]. Genetic programming [2], that is based on genetic algorithms 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. The generation is achieved with the help of grammatical evolution. Grammatical evolution is an evolutionary process that can produce programs in an arbitrary language. The production is performed using

*Corresponding author

a mapping process governed by a grammar 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 the grammar of the target language in BNF syntax and the proper 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

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

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 figure 1. The numbers in parentheses denote the sequence number of the corresponding production rule to be used in the selection procedure described above.

Figure 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)
```

The symbol S in the grammar denotes the start symbol of the grammar. For example, suppose we have the chromosome $x = [16, 3, 4, 7, 10, 28, 24, 1, 2, 4]$. 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 [8, 9, 10, 11, 13]

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

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 selection rate are set. The selection rate denotes the fraction of the number of chromosomes that will go through unchanged to the next generation(replication). The mutation rate controls the average number of changes inside a chromosome. Every chromosome in the population is initialized at random. The initialization of every chromosome is performed by randomly selecting an integer for every element of the corresponding vector.

3.2 Fitness evaluation

3.2.1 ODE case

We express the ODE's in the following form:

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

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

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

where t_i is one of the two endpoints a or b. The steps for the fitness evaluation of the population 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) Construct the corresponding model $M_i(x)$, expressed in the grammar described earlier.
   (b) Calculate the quantity

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

   (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:

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

where λ is a positive number.

3.2.2 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:

$$f\left(x, y, \Psi(x,y), \frac{\partial}{\partial x}\Psi(x,y), \frac{\partial}{\partial y}\Psi(x,y), \frac{\partial^2}{\partial x^2}\Psi(x,y), \frac{\partial^2}{\partial y^2}\Psi(x,y)\right) = 0 \tag{6}$$

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

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

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

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

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

   - Calculate the fitness of the chromosome as:

$$v_i = E(M_i) + P_1(M_i) + P_2(M_i) + P_3(M_i) + P_4(M_i) \qquad (7)$$

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, 19, 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 $(a_0, a_1, ..., a_n)$ 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.

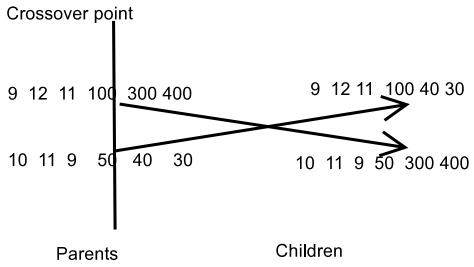
3.4 Genetic operations

At first, we perform a sorting of chromosomes 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. After that, we produce $c = (1-s)*g$ new chromosomes, where s is the selection rate of the model and g is the total number of individuals in the population. The new individuals will replace the worst in the population at the end of the crossover procedure. For every couple of children we select two individuals from the current population with the method of tournament selection, i.e.:

- A group of $K \geq 2$ random individuals is created.
- The individual with the best fitness in the group is selected, the others are discarded.

Having selected two individuals, we proceed with the one - point crossover. In that procedure we cut the chromosomes at a randomly chosen point and we exchange the right-hand-side sub-chromosomes, as shown in figure 2:

Figure 2: One - Point crossover



After the crossover we proceed with mutation. For every chromosome in the population (excluding those which have been selected for replication to the next generation) and for every element in the chromosome we choose a random number in the range $[0, 1]$. If this number is less than or equal to the mutation rate we change randomly the corresponding element in the chromosome, otherwise we leave it intact.

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 with . 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 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 selection rate and 5% for the mutation rate. The population size was set to 1000 and the length of each chromosome to 50. 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. The maximum number of generations allowed was set to 2000 and the preset fitness target for the termination criteria was 10^{-7}. The value of N for ODE's was between 10 and 20 depending on the problem. For PDE's N was set to 5 and $N_x = N_y = 50$. Also, for some problems we present graphs of the intermediate trial solutions. The evaluation of the derived functions was performed using the FunctionParser programming library [3].

4.1 Linear ODE's

First order ODE's

In table 2 we list first order ODE's that were solved exactly with the proposed method. The columns ODE, x_0, x_1, $y(x_0)$, G and $y(x)$ contain the equation to be solved, the left and the right endpoints of the domain, the boundary condition at $x = x_0$, the number of the required generations for recovering the exact solution and the exact solution.

Table 2: First order ODEs

ODE	x_0	x_{N-1}	$y(x_0)$	G	$y(x)$
$y' = \frac{2x-y}{x}$	0.1	1.0	20.1	241	$x + \frac{2}{x}$
$y' = \frac{1-y\cos(x)}{\sin(x)}$	0.1	1.0	$\frac{2.1}{\sin(0.1)}$	35	$\frac{x+2}{\sin(x)}$
$y' = -\frac{1}{5}y + \exp\left(-\frac{x}{5}\right)\cos(x)$	0.0	1.0	0.0	108	$\exp\left(-\frac{x}{5}\right)\sin(x)$

Second order ODE's

In table 3 we present second order ODE's, with boundary conditions on the left endpoint, that were solved with the proposed method. The name and the meaning for the columns is the same as in table 2 with the addition of the column $y'(x_0)$, which contains the value of first derivative on the left boundary. Similarly, in table 4 we list second order ODE's, with boundary conditions on both of the endpoints. The column $y(x_1)$ contains the boundary condition on the right endpoint.

Table 3: Second order ODE's

ODE	x_0	x_{N-1}	$y(x_0)$	$y'(x_0)$	G	$y(x)$
$y'' = -100y$	0.0	1.0	0.0	10.0	59	$\sin(10x)$
$y'' = 6y' - 9y$	0.0	1.0	0.0	2.0	258	$2x\exp(3x)$
$y'' = -\frac{1}{5}y' - y - \frac{1}{5}\exp(-\frac{x}{5})\cos(x)$	0.0	2.0	0.0	1.0	137	$\exp(-\frac{x}{5})\sin(x)$

Table 4: Second order ODEs

ODE	x_0	x_{N-1}	$y(x_0)$	$y(x_1)$	G	$y(x)$
$y'' = -100y$	0.0	1.0	0.0	$\sin(10.0)$	76	$\sin(10x)$
$xy'' + (1-x)y' + y = 0$	0.0	1.0	1.0	0.0	15	$1 - x$
$y'' = -\frac{y'}{5} - y - \frac{1}{5}\exp(-\frac{x}{5})\cos(x)$	0.0	1.0	0.0	$\frac{\sin(1)}{exp(0.2)}$	110	$\exp(-\frac{x}{5})\sin(x)$

In figure 3 we can see plots of the evolving trial corresponding solution for the first problem of table 3. At generation 22 the fitness value was 4200.5 and the intermediate solution was:

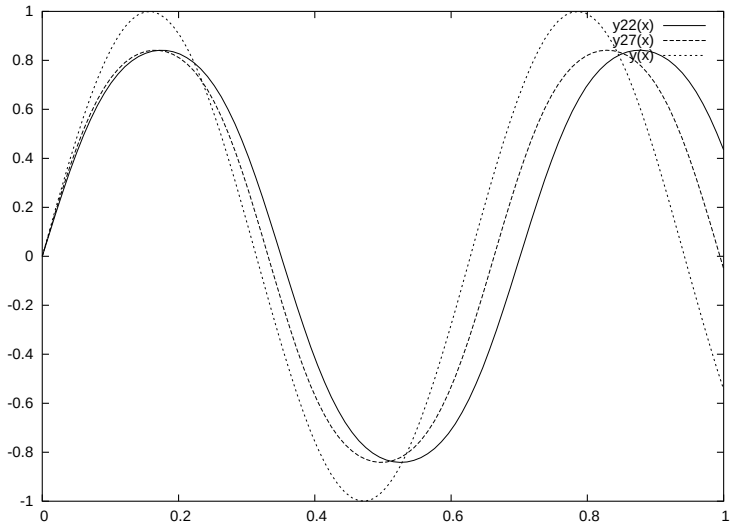
$$y_{22}(x) = \sin((\sin(-\log(4)x((-\cos(\cos(\exp(7)))\exp(\cos(6))) - 5))))$$

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.

Figure 3: Evolving candidate solutions of $y' = -100y$ with boundary conditions on the left



Hermite Polynomials

The Hermite polynomials are solutions of the ordinary differential equation:

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

where n is an integer with the attribute $n \geq 0$. In table 5 we see some of the Hermite polynomials that solved with the proposed method. The meaning of the columns is the same as the previous tables.

Table 5: Hermite Polynomials

ODE	x_0	x_{N-1}	$y(x_0)$	$y'(x_0)$	G	$y(x)$
$y'' - 2xy' = 0$	0.0	1.0	1.0	0.0	10	1
$y'' - 2xy' + 2y = 0$	0.0	1.0	0.0	2.0	35	$2x$
$y'' - 2xy' + 4y = 0$	0.0	1.0	-2.0	0.0	105	$4x^2 - 2$
$y'' - 2xy' + 6y = 0$	0.0	1.0	0.0	-12.0	350	$8x^3 - 12x$

4.2 Non - linear ordinary differential equations

Example 1

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

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. However the resulting solution $y = \exp\left(\frac{\log(x)}{2}\right)$ is identical to $\sqrt{x}$ and was recovered in 122 generations.

Example 2

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

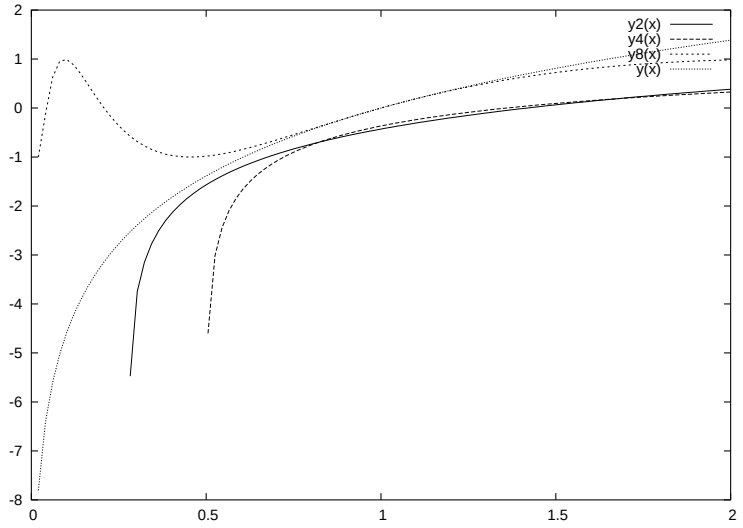
with $y(1) = 1 + \sin(1)$ and $x \in [1, 2]$ The exact solution is $y = x + \sin(x)$, recovered after 4 generations.

Example 3

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

with $y(1) = 0$ and $x \in [1, 2]$. The exact solution is $y = \log(x^2)$. In figure 4 we plot intermediate trial solutions.

Figure 4: Candidate solutions of third non-linear equation



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

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

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 8^{th} 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 9^{th} generation.

Example 4

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

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 30^{th} generation.

4.3 Systems of ODE's

Example 1

$$\begin{aligned} y_1' &= \cos(x) + y_1^2 + y_2 - (x^2 + \sin^2(x)) \\ y_2' &= 2x - x^2\sin(x) + y_1 y_2 \end{aligned}$$

with $y_1(0) = 0$, $y_2(0) = 0$ and $x \in [0, 1]$. The exact solution is given by: $y_1 = \sin(x)$, $y_2 = x^2$ recovered at the 45^{th} generation.

Example 2

$$\begin{aligned} y_1'' - y_1' - y_2' &= -1 \\ y_2' + 2y_1' - y_2 &= 1 - x + 2\exp(x) \end{aligned}$$

with $y_1(0) = 1$, $y_1'(0) = 1$, $y_2(0) = 0$ and $x \in [0, 1]$. The exact solution is $y_1 = \exp(x)$, $y_2 = x$, recovered at the 23^{th} generation.

Example 3

$$\begin{aligned} y_1' &= \tfrac{\cos(x) - \sin(x)}{y_2} \\ y_2' &= y_1 y_2 + \exp(x) - \sin(x) \end{aligned}$$

with $y_1(0) = 0$, $y_2(0) = 1$ and $x \in [0, 1]$. The exact solution is $y_1 = \frac{\sin(x)}{\exp(x)}$, $y_2 = \exp(x)$, recovered at the 89^{th} generation.

Example 4

$$\begin{aligned} y_1' &= y_1 \\ y_2' &= \tfrac{1}{2\exp(x)}(y_3 - y_1) \\ y_3' &= 3y_1 \end{aligned}$$

with $y_1(0) = 1$, $y_2(0) = 0$, $y_3(0) = 3$ and $x \in [0, 1]$. The exact solution is $y_1 = \exp(x)$, $y_2 = x$, $y_3 = 3\exp(x)$, recovered at the 167^{th} generation.

Example 5

$$\begin{aligned} y_1' &= \cos(x) \\ y_2' &= -y_1 \\ y_3' &= y_2 \\ y_4' &= -y_3 \\ y_5' &= y4 \end{aligned}$$

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 $y_1 = \sin(x)$, $y_2 = \cos(x)$, $y_3 = \sin(x)$, $y_4 = \cos(x)$, $y_5 = \sin(x)$, recovered at the 825^{th} generation.

Example 6

$$\begin{aligned} y_1' &= -\tfrac{1}{y_2}\sin(\exp(x)) \\ y_2' &= -y_2 \end{aligned}$$

with $y_1(0) = \cos(1.0)$, $y_2(0) = 1.0$ and $x \in [0,1]$. The exact solution is $y_1 = \cos(\exp(x))$, $y_2 = \exp(-x)$, recovered at the 119^{th}generation.

4.4 ODE's 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:

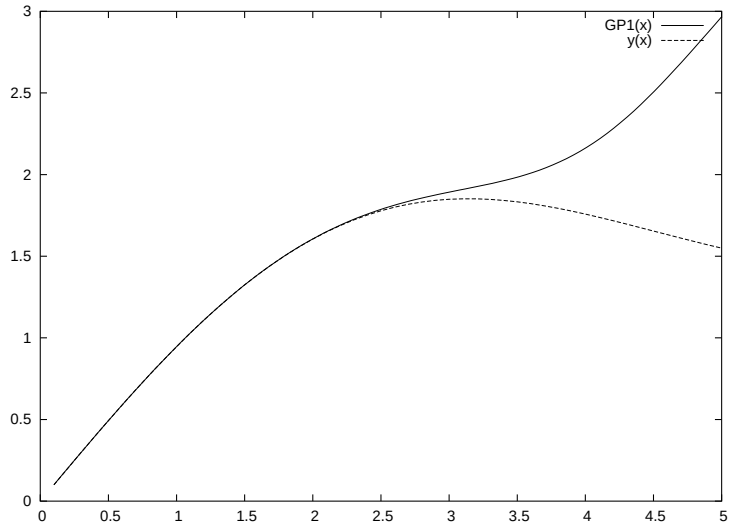
$$\text{GP1}(x) = x(\cos(-\sin(x/3 + \exp(-5 + x - \exp(\cos(x)))))))$$

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

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

In figure 5, we plot the two functions in the range [0,5]

Figure 5: Plot of GP1(x) and $y(x) = \int_0^x \frac{\sin(t)}{t} dt$



Example 2

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

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

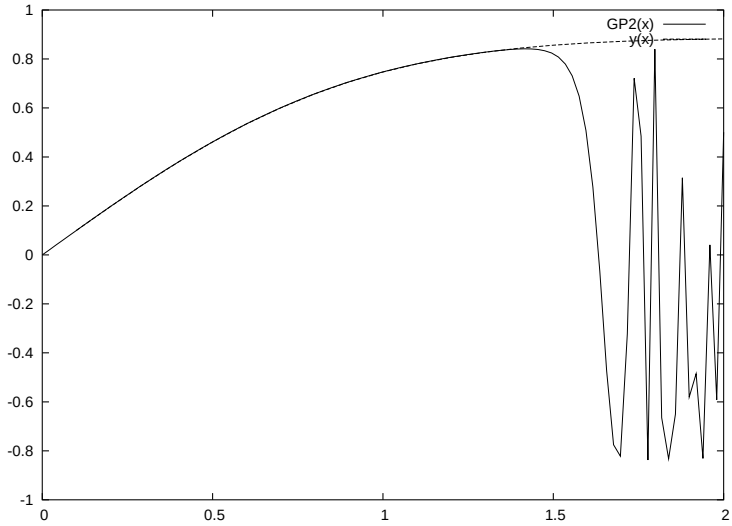
$$y(x) = \int_0^x \exp(-t^2)dt$$

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

Figure 6: Plot of GP2(x) and $y(x) = \int_0^x \exp(-t^2)dt$



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

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

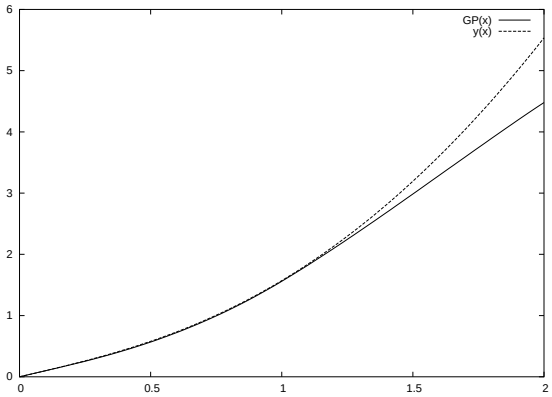
in the range $[0, 1]$ and with initial conditions $y(0) = 0$ and $y'(0) = 1$. The analytical solution is $y(x) = (x^2+1)\text{atan}(x)$. Note that atan($x$) does not belong

to the function repertoire of the method and this make the case special. The solution reached is not exact but approximate given by:

$$\begin{aligned} \mathrm{GP}(x) \quad = \quad & x/\sin(\exp(\cos(5/4/\exp(x))- \\ & \exp((-\exp(((-((-\exp(\cos(\sin(2x))))))))))) \end{aligned}$$

plotted in figure 7 with fitness 0.0059168.

Figure 7: GP(x) and $y(x) = (x^2 + 1)\mathrm{atan}(x)$



4.6 PDE's

Example 1

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

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 $\Psi(x,y) = (x+y^3)\exp(-x)$, recovered at the 821^{th} generation.

Example 2

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

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 $\Psi(x,y) = \sin(x)\cos(y)$, recovered at the 58^{th} generation. 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 figure 8. At the 10^{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 figure 9.

Figure 8: Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and GP1(x, y)

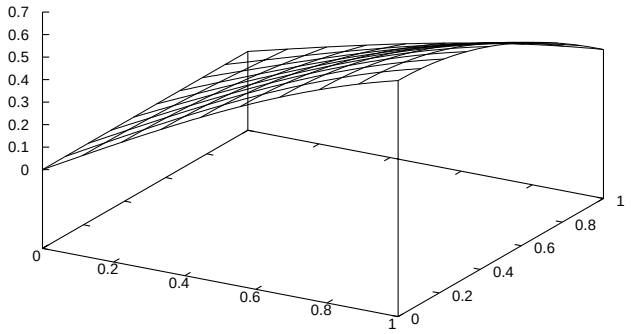
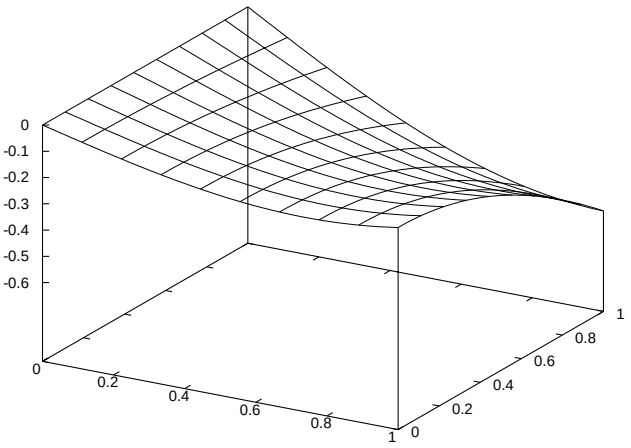


Figure 9: Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and GP10(x, y)

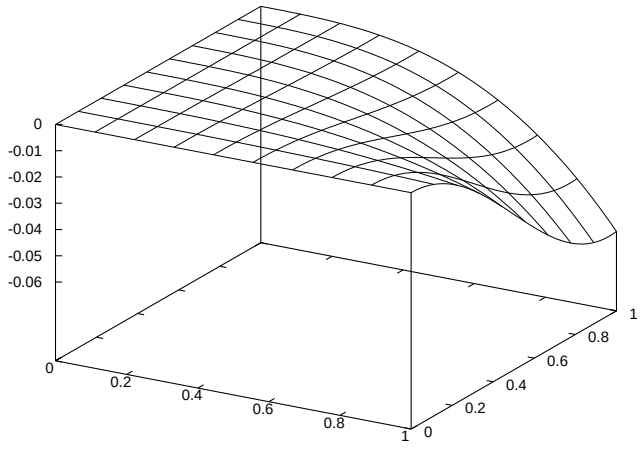


At the 40^{th} generation the trial solution was

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

with fitness value 0.59. The difference between the trial solution $\text{GP40}(x, y)$ and the exact solution $\Psi(x, y)$ is shown in figure 10.

Figure 10: Difference between $\Psi(x, y) = \sin(x)\cos(y)$ and $\text{GP40}(x, y)$



Example 3

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

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 $\Psi(x, y) = x^2 + y^2 + x + y + 1$, recovered at the 124^{th} generation.

Example 4

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

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 $\Psi(x, y) = \sin(xy)$, recovered at the 10^{th} generation.

Example 5

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

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 $\Psi(x,y) = x(\exp(-x)+\exp(-y))$, recovered at the 119^{th} generation.

Example 6

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: $f_0(y) = \log(1+y^2)$, $f_1(y) = \log(2+y^2)$, $g_0(x) = \log(1+x^2)$ and $g_1(x) = \log(2+x^2)$. The exact solution is $\Psi(x,y) = \log(1+x^2+y^2)$, recovered at the 236^{th} generation.

Example 7

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

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 $\Psi(x,y,z) = x^2+y^2+z^2+1$, recovered at the 1298^{th} generation.

Example 8

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

with $x \in [0,1]$ and $y \in [0,1]$ and $z \in [0,1]$ and boundary conditions: $\Psi(0,y,z) = y^2\cos(z)$, $\Psi(1,y,z) = z\sin(1) + y^2\cos(z)$, $\Psi(x,0,z) = z\sin(x)$, $\Psi(x,1,z) = z\sin(x)+\cos(z)$, $\Psi(x,y,0) = y^2$ and $\Psi(x,y,1) = \sin(x)+y^2\cos(1)$. The exact solution is $\Psi(x,y,z) = z\sin(x) + y^2\cos(z)$, recovered at the 280^{th} generation.

Example 9

$$\nabla^2\Psi(x,y) = \left(\frac{1}{x} + \frac{x}{y^2}\right)\sin(xy) + \frac{2}{y^3}\left(\cos(xy) - 1\right)$$

with $x \in [0,1]$ and $y \in [0,1]$ and the appropriate Dirichlet boundary conditions. The exact solution is

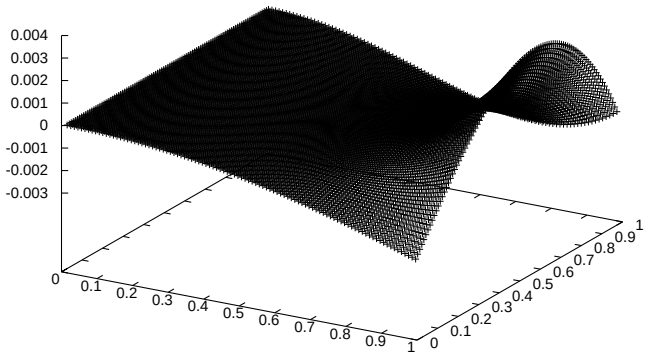
$$\Psi(x,y) = \int_0^y \left(\frac{1-\cos(xz)}{z^2}\right) dz$$

The following approximate solution was found with fitness value 9.65×10^{-4}:

$$\text{GP}(x,y) = x\sin(x/2)(log(exp(y-5)+1)+y))$$

The difference between $\Psi(x,y)$ and $\mathrm{GP}(x,y)$ is shown in figure 11.

Figure 11: Difference between $\Psi(x,y) = \int_0^y \left(\frac{1-\cos(xz)}{z^2}\right) dz$ and $\mathrm{GP}(x,y)$



5 Conclusions and further work

We presented a novel approach for solving ODE's and PDE's. The method is based on grammatical evolution. This approach, similarly with others, creates trial solutions and seeks to minimize an associated error. The advantage is that grammatical evolution can produce trial solutions of high complexity and of a very 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 depository or even to allow for additional operations.

We applied the method to a set of problems to obtain an assessment for its potential. Although the effort can be reduced from the number of generations and the number of the training points, we quote CPU times for three examples, in table 6.

Table 6: CPU times

Section	Problem	Time
4.1	Table 2 - Problem 3	8.6
4.3	Example 5	170
4.6	Example 6	94

Non - linearity does not seem to require either special handling or extra effort. Note that PDE's require significantly higher an effort to be solved. This is to be expected since to cover higher dimension domains one needs correspondingly more training points, a fact which in turn weighs on the computing time. 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, No. 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. C. Hongqing, K. Lishan, and C. Yuping, "Evolutionary Modeling of Systems of Ordinary Differential Equations with Genetic Programming," Genetic Programming and Evolvable Machines, vol. 1, pp. 309-337, 2000.

8. M. O'Neill and C. Ryan, "Genetic code degeneracy: Implications for grammatical evolution and beyond," In D. Floreano, J.-D. Nicoud, and F. Mondada (eds.), Advances in Artificial Life, volume 1674 of LNAI, Lausanne, 13-17 September 1999, Springer Verlag, page 149, 1999.

9. M. O'Neill and C. Ryan, "Under the hood of grammatical evolution," In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon Vasant Honavar, Mark Jakiela, and Robert E. Smith (eds.), Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, Orlando, Florida, USA, 13-17 July 1999, Morgan Kaufmann, pp. 1143-1148, 1999.

10. M. O'Neill and C. Ryan, "Evolving Multi-Line Compilable C Programs," In Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty (eds.), Proceedings of EuroGP'99, volume 1598 of LNCS, Goteborg, Sweden, 26-27 May 1999. Springer-Verlag, pp. 83-92, 1999.

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 and C. Ryan, "Automatic generation of caching algorithms," In Kaisa Miettinen, Marko M. Mkel, Pekka Neittaanmki, and Jacques Periaux (eds.), Evolutionary Algorithms in Engineering and Computer Science, 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 H. R. Arabnia et al., editor, Proceedings of the International conference on Artificial Intelligence, volume II, 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 M. Iri and K. Tanabe (eds.), Mathematical Programming: Recent Developments and Applications, Kluwer Academic Publishers, Amsterdam, pp. 83-108, 1989.