Solving differential equations with constructed neural networks

Ioannis G. Tsoulos[(1)], Dimitris Gavrilis[(2)], Euripidis Glavas[(3)]

[(1)]Department of Computer Science,
University of Ioannina, GREECE 45110
[(2)]Digital Curation Unit Athena Research Centre
Artemidos 6 & Epidavrou 15125 Maroussi Greece
[(3)]Department of Communications, Informatics and Management,
Technological Educational Institute of Epirus, Greece

Abstract

A novel hybrid method for the solution of ordinary and partial differential equations is presented here. The method creates trial solutions in neural network form using a scheme based on grammatical evolution. The trial solutions are enhanced periodically using a local optimization procedure. The proposed method is tested on a series of ordinary differential equations, systems of them as well as on partial differential equations with Dirichlet boundary conditions and the results are reported.

Keywords: Differential equations, Neural networks, genetic programming, grammatical evolution.

1 Introduction

A series of problems in many scientific fields can be modelled with the use of differential equations such as problems in physics [1, 2, 3, 4, 5], chemistry [6, 7, 8], biology [9, 10], economics [11] etc. Due to importance of differential equations many methods have been proposed in the relevant literature for their solution such as Runge Kutta methods [12, 13, 14], Predictor - Corrector [15, 16, 17], radial basis functions [18, 19], artificial neural networks [20, 21, 22, 23, 24, 25, 26, 27], models based on genetic programming [28, 29] etc. In this article a hybrid method utilizing constructed feed - forward neural networks by grammatical evolution and a local optimization procedure is used in order to solve ordinary differential equations (ODE'S), systems of ordinary differential 's). The constructed neural networks with grammatical evolution have been recently introduced by Tsoulos et al [30] and it utilizes the well - established Grammatical Evolution

technique [31] to evolve the neural network topology along with the network parameters. The method has been tested with success on a series of data - fitting and classifications problems. In this article the constructed neural network methodology is applied on a series of differential equations preserving the initial or boundary conditions using penalization. The proposed method does not require the user to enter any information regarding the topology of the network. Also, the new method can be used to solve either ODE's ore PDE's and it can be easily parallelized. This idea is similar to the Cascade Correlation neural networks introduced by [32] in which the user is not required to enter any topology information. However the method for selecting the network topology differs since the proposed algorithm is a stochastic one. In the proposed method, the advantage of using an evolutionary algorithm is that the penalty function (used for initial or boundary conditions) can be incorporated easily into the training process.

The rest of this article is organized as follows: in section 2 a brief description of the Grammatical Evolution algorithm is given followed by an analytical description of the proposed, in section 3 the test functions used in the experiments followed by the experimental results are outlined and in section 4 some conclusions are derived.

2 Method description

In this section a brief description of the Grammatical Evolution algorithm is given, the main steps of the proposed algorithm are outlined with the steps for the fitness evaluation for the cases of ODE's, SODE's and PDE's.

2.1 Grammatical Evolution

Grammatical evolution is an evolutionary technique that can produce code in any programming language requiring the grammar of the target language in BNF syntax and some proper fitness function and it has been used with success in many scientific fields such as symbolic regression [34], discovery of trigonometric identities [35], robot control [36], caching algorithms [37], financial prediction [38] etc. Chromosomes in grammatical evolution, in contrast to classical genetic programming [33], are not expressed as parse trees, but as vectors of integers. Each integer denotes a production rule from the given 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:

- Read an element from the chromosome (with value V).
- Select the rule according to the scheme

$$\text{RULE} = V \bmod \mathcal{R} \tag{1}$$

where $\mathcal{R}$ 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. If the limit of the wrapping events is reached the chromosome is rejected by assigning to it a large fitness value, which prevents the chromosome to be used in the crossover procedure. In the proposed algorithm the limit of wrapping events was set to 2. As an example of the mapping procedure of the Grammatical Evolution consider the BNF grammar shown in figure 1. The number in parentheses denote the sequence number of the corresponding production rule to be used in the mapping procedure. Consider the chromosome $x = [9, 8, 6, 4, 16, 10, 17, 23, 8, 14]$. The steps of the mapping procedure are listed in table 1. The final outcome of these steps is the expression $x_2 + \cos(x_3)$.

2.2 Algorithm description

The proposed method is based on an evolutionary algorithm, a stochastic process whose basis lies in the biological evolution. In order to measure the efficiency of the algorithm, a neural network capable of solving differential equations is employed. The neural network'As efficiency is used as the fitness of the evolutionary algorithm along with a penalty function which is used in order to represent the boundary or initial conditions of the differential equations. The idea of combining a neural network with an evolutionary algorithm is a well established approach that has been used numerous times both in the bibliography and in real-world applications. The main steps of the algorithm have as follows:

1. **Set** the number of chromosomes S, the number of maximum generations allowed K, the crossover rate p_c, the mutation rate p_m, a small positive number ϵ the integer parameter G and the integer parameter M. The parameter G determines how frequent the local search procedure will be applied and the parameter M determines in how many chromosomes the local optimization procedure will be applied.
2. **Set** iters=0.
3. **Initialize** the S chromosomes. Each chromosome will be mapped to a neural network using a procedure described subsequently in subsection 2.3.
4. **Calculate** ss
   is described in the following subsection.
5. **Apply** the genetic operations of crossover and mutation to the population.
6. **Set** iters=iters+1.
7. **If** iters mod G=0 **then**

(a) **For** i=1..M **do**

i. **Select** randomly a chromosome R_i from the genetic population.

ii. **Construct** with the Grammatical Evolution procedure the corresponding neural network $N(R_i)$.

iii. **Train** the neural network $N(R_i)$ with a local optimization procedure.

iv. **Put** the modified chromosome back to the genetic population.

(b) **End for**

8. **Endif**

9. **If** iters$\geq K$ **or** the best chromosomes has fitness value below the predefined threshold ϵ **terminate**, **else goto** step 4.

2.3 Neural Network Construction

Every chromosome in the genetic population is a vector of integers, which is mapped through the mapping procedure of Grammatical Evolution into a feed - forward artificial neural network with one hidden level and one output. The output of the constructed neural network is a summation of different sigmoidal units and it can be formulated as:

$$N(x, \overrightarrow{p}) = \sum_{i=1}^{H} p_{(d+2)i-(d+1)} \sigma \left(\sum_{j=1}^{d} p_{(d+2)i-(d+1)+j} x_j + p_{(d+2)i} \right) \quad (2)$$

The constant d denotes the dimension of the input vector $\overrightarrow{x}$, the parameter H denotes the number of the processing units (hidden nodes) of the neural network and the function (x) is the sigmoidal function expressed by the equation:

$$(x) = \frac{1}{1 + \exp(-x)} \quad (3)$$

The BNF grammar that controls the mapping procedure is shown in figure 2. The sigmoidal functions used in the neural network can be replaced by other functions including radial basis functions. By introducing dynamic activation functions in the neural network, a new parameter is introduced in the system. This new parameter can be either encoded in the evolutionary algorithm or it can be wrapped into another algorithm that firstly "selects" the optimal activation function. In such cases it is usually required to use cross validation to measure how well the selected activation functions work. For further information regarding this matter please refer to [30].

2.4 Fitness evaluation

a similar way as in [29] using penalization. The proposed penalty function is used to force the neural

network to train on the boundary conditions (PDEs) or the initial conditions (ODEs). The error function represents the neural network'As misclassification rate and is necessary in order to measure its efficiency.

2.4.1 ODE case

The method considers ODE's given in the following form:

$$f\left(x, y, y^{(1)}, \ldots, y^{(n-1)}, y^{(n)}\right) = 0,\ x \in [a, b] \tag{4}$$

where $y^{(n)}$ denotes the n-order derivative of y. The initial conditions are expressed in the following form:

$$\Psi_i\left(x, y, y^{(1)}, ..., y^{(n-1)}\right)_{|x=t_i} = 0,\ i = 1..n \tag{5}$$

where t_i is either a or b. The steps for the fitness evaluation of any given chromosome g are:

1. **Choose** T equidistant points in $[a, b]$ denoted by $[x_0, x_1, ..., x_{T\ 1}]$.
2. **Construct** the neural network $N(x, g)$ using Grammatical Evolution.
3. **Calculate** the train error of the network using the equation:

$$E\left(N(g)\right) = \sum_{i=0}^{T-1}\ f\left(x_i, N\left(x_i, g\right), N^{(1)}\left(x_i, g\right), ..., N^{(n)}\left(x_i, g\right)\right)\Big)^2 \tag{6}$$

4. **Calculate** the penalty value $P\left(N(g)\right)$using the following equation:

$$P\left(N(g)\right) = \lambda \sum_{k=1}^{n} \Psi_k^2\left(x, N(x, g), N^{(1)}(x, g), ..., N^{(n-1)}(x, g)\right)_{|x=t_k} \tag{7}$$

   where λ is a positive number.

5. **Calculate** the final fitness value as:

$$V(g) = E(N(x, g)) + P(N(x, g)) \tag{8}$$

2.4.2 SODE case

The proposed method deals with systems of ordinary differential equations expressed in the form:

$$\left(\begin{array}{lcl} f_1\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, ..., y_k, y_k^{(1)}\right) & = & 0 \\ f_2\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, ..., y_k, y_k^{(1)}\right) & = & 0 \\ \vdots & \vdots & \vdots \\ f_k\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, ..., y_k, y_k^{(1)}\right. & = & 0 \end{array}\right),\ x \in [a, b] \tag{9}$$

with the following initial conditions:

$$\left(\begin{array}{ccc} y_1(a) & = & y_{1a} \\ y_2(a) & = & y_{2a} \\ \vdots & \vdots & \vdots \\ y_k(a) & = & y_{ka} \end{array} \right) \tag{10}$$

The steps for the fitness evaluation of any given chromosome g are the following:

1. **Choose** T equidistant points in $[a, b]$ denoted by $[x_0, x_1, ..., x_{T\ 1}]$.
2. **Split** the chromosome g into k parts and **construct** using Grammatical Evolution the neural networks $N_1(x, g),\ N_2(x, g),\ ...,\ N_k(x, g)$.
3. **Calculate** the train errors $E\left(N_i(g)\right),\ i = 1..k$ for every neural network using the equation:

$$E\left(N_i(x,g)\right) = \sum_{j=0}^{T\ \ 1} f_i\left(x_j, N_1\left(x_j,g\right), N_1^{(1)}\left(x_j,g\right), ..., N_k\left(x_j,g\right), N_k^{(1)}\left(x_j,g\right)\right)\Big)^2 \tag{11}$$

4. **Calculate** the penalty value for every neural network $N_i(x, g),\ i = 1..k$:

$$P\left(N_i(g)\right) = \lambda\left(N_i\left(a,g\right) - y_{ia}\right)^2 \tag{12}$$

5. **Calculate** the total fitness value

$$V(g) = \sum_{i=1}^{k} E\left(N_i(x,g)\right) + P\left(N_i(x,g)\right) \tag{13}$$

2.4.3 PDE case

The proposed method deals with PDE's of two variables with Dirichlet boundary conditions, without loss in generality. The PDE's must be expressed in the form:

$$f\left(x, y, \Psi(x,y), \frac{\partial}{\partial x}\Psi(x,y), \frac{\partial}{\partial y}\Psi(x,y), \frac{\partial^2}{\partial x^2}\Psi(x,y), \frac{\partial^2}{\partial y^2}\Psi(x,y)\right) = 0 \tag{14}$$

with $x \in [a, b]$ and $y \in [c, d]$. The boundary conditions are given by:

1. $a, y) = f_0(y)$
2. $\Psi\left(b, y\right) = f_1(y)$
3. $x, c) = g_0(x)$
4. $\Psi\left(x, d\right) = g_1(x)$

e g are:

1. **Choose** T uniformly distributed points in the grid $[a,b] \times [c,d]$ denoted by $[x_i, y_i]\,,\ i = 0..T-1$.
2. **Choose** B equidistant points in $[a,b]$ denoted $x_{bi},\ i = 0..B-1$.
3. **Choose** B equidistant points in $[c,d]$ denotes by $y_{bi}, i = 0..B-1$.
4. **Construct** using the Grammatical Evolution procedure the neural network $N(x,y,g)$.
5. **Calculate** the train error of the neural network $N(x,y,g)$:

$$E(N(x,y,g)) = \sum_{i=0}^{T-1} f\left(x, y, N(x,y,g), \frac{\partial}{\partial x}N(x,y,g), \frac{\partial}{\partial y}N(x,y,g), \frac{\partial^2}{\partial x^2}N(x,y,g), \frac{\partial^2}{\partial y^2}N(x,y,g)\right)^2 \tag{15}$$

6. **Calculate** the penalty quantities

$$\begin{aligned} P_1(N(x,y,g)) &= \lambda \sum_{i=0}^{B-1} \left(N\left(a, y_{bi}, g\right) - f_0\left(y_{bi}\right)\right)^2 \\ P_2(N(x,y,g)) &= \lambda \sum_{i=0}^{B-1} \left(N\left(b, y_{bi}, g\right) - f_1\left(y_{bi}\right)\right)^2 \\ P_3(N(x,y,g)) &= \lambda \sum_{i=0}^{B-1} \left(N\left(x_{bi}, c, g\right) - g_0\left(x_{bi}\right)\right)^2 \\ P_4(N(x,y,g)) &= \lambda \sum_{i=0}^{B-1} \left(N\left(x_{bi}, d, g\right) - g_1\left(x_{bi}\right)\right)^2 \end{aligned} \tag{16}$$

   where λ a positive real number.
7. **Calculate** the total fitness with

$$V(g) = E(N(x,y,g)) + P_1(N(x,y,g)) + P_2(N(x,y,g)) + P_3(N(x,y,g)) + P_4(N(x,y,g)) \tag{17}$$

3 Experiments

The proposed method was tested on a series of ODE's, non - linear ODE's, sysiables and Dirichlet boundary conditions. Theses test functions are listed subsequently and they have been used in the experiments performed in [20] and in [29]. In all experiments, the ODE's were sampled using a uniform distribution and only 1000 samples were extracted in the intervals of x that for each case. In the following sub-sections, the ODE's are presented for each one of the four series.

3.1 Linear ODE's

ODE1

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

with $y(1) = 3$ and $x \in [1, 2]$. The analytical solution is $y(x) = x + \frac{2}{x}$

ODE2

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

with $y(1) = \frac{3}{\sin(1)}$ and $x \in [1, 2]$. The analytical solution is $y(x) = \frac{x+2}{\sin(x)}$.

ODE3

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

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

ODE4

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

with $y(0) = 0$ and $y(1) = \frac{\sin(0.1)}{\exp(0.2)}$ and $x \in [0, 1]$. The analytical solution is $y(x) = \exp\left(-\frac{x}{5}\right)\sin(x)$.

ODE5

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

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

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

ODE6

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

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

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

ODE7

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

with $y(0)=0,\ y'(0)=1$ and $x\in[0,1]$. The analytical solution is $y(x)=\left(x^2+1\right)\arctan(x)$.

3.2 Non linear ODE's

NLODE1

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

with $y(1)=1$ and $x\in[1,4]$. The analytical solution is given by $y(x)=\sqrt{x}$

NLODE2

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

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

NLODE3

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

with $y(1)=0$ and $x\in[1,2]$. The analytical solution is given by $y(x)=\log\ \left(x^2\right)$.

NLODE4

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

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

3.3 Systems of ODE's

SODE1

$$\begin{aligned} y_1^{'} &= \cos(x)+y_1^2+y_2-\left(x^2+\sin^2(x)\right)\\ y_2^{'} &= 2x\quad x^2\sin(x)+y_1y_2 \end{aligned}$$

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

SODE2

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

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

SODE3

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

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 analytical solution is given by: $y_1(x) = \sin(x),\ y_2(x) = \cos(x),\ y_3(x) = \sin(x),\ y_4(x) = \cos(x),\ y_5(x) = \sin(x)$.

SODE4

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

with $y_1(0) = \cos(1.0),\ y_2(0) = 1.0$ and $x \in [0, 1]$.The analytical solution is given by: $y_1(x) = \cos(\exp(x)),\ y_2(x) = \exp(\ \ x)$.

3.4 PDE's

PDE1

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

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

PDE2

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

with $x \in [0, 1],\ y \in [0, 1]$. The associated boundary conditions are: $\Psi(0, y) = 0,\ \Psi(1, y) = \sin(1)\cos(y),\quad x, 0) = \sin(x),\ \Psi(x, 1) = \sin(x)\cos(1)$. The analytical solution is given by: $\quad x, y) = \sin(x)\cos(y)$.

PDE3

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

with $x \in [0,1],\ y \in [0,1]$. The boundary conditions are: $\Psi(0,y) = y^2 + y + 1,\ \Psi(1,y) = y^2 + y + 3,\quad x,0) = x^2 + x + 1,\ \Psi(x,1) = x^2 + x + 3$. The analytical solution is given by: $\quad x,y) = x^2 + y^2 + x + y + 1$.

PDE4

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

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

3.5 Experimental results

The method was performed 30 times, using different seed for the random number generator each time, on every differential equation described previously and averages were taken. In table 2 the numerical values for the parameters of the algorithm are listed. The numerical values for the majority of these parameters were taken from the papers of Lagaris et al[20] and Tsoulos et al [30] and some of them have been found experimentally. The local optimization procedure used in the experiments was a BFGS variant due to Powell [39]. In table 3 the results from the application of the proposed method to the test functions are listed. The column TRAIN ERROR denotes the average per point error of the proposed method to the T points of the training set, the column TEST ERROR denotes the average per point error of the proposed method to the points belonging to the test set and the column GENERATIONS denote the average number of the required generations of the genetic algorithm. For the cases of ODE's and SODE's the test set had 1000 equidistant points and for the case of PDE's the test set had 10000 uniformly distributed points. In figure 3 we can observe the progress of solution for a example run for ODE1. As we can notice the proposed method managed in 60 generations to solve the objective problem. Also in figure 4 the application of the final solution in range [1:3] is plotted against the true solution $f(x) = x + \frac{2}{x}$. As we can see the final solution maintains its equality even outside the training domain. Furthermore, in table 4 we list the experimental results for the same problem using different values of the parameter λ. As it can be noticed, the method succeeds in solving the problem even though for small values of the critical parameter . As we can notice from the column GENERATIONS of table 3 the proposed method managed to solve all ODE's in very little time, if time is expressed as the number of required generations. The proposed method is a combination of two types of algorithms: a stochastic (genetic algorithm) and a deterministic. This fact raises some stability issues so, in order to overcome this, each experiment ran for

30 times with a different seed in order to diminish any random variations in the experimental results. The stability issues mostly refer to the stochastic nature of the proposed algorithm. This means that it does not always converge to the same local minimum. That is why in the experimental setup each experiment ran for 30 times with different seed.

4 Conclusions

In conclusion, a novel method for solving ODE's, PDE's and SODE's is presented. This method utilizes neural networks that are constructed using artificial neural networks that are constructed using grammatical evolution. This novel method for simultaneously constructing and training neural networks, has been used successfully in other domains. Concerning the differential equations problem, a series of experiments in 19 well known problems, showed that the proposed method managed to solve all problems. Although, a number of methods for differential equations solving exists, the proposed one has a very little execution time and does not require the user to enter any parameters. The main advantages of the proposed method are the following:

1. The user is only required to sample the differential equations in order to create the train/test files.
2. The method is general enough to be applied to ODE's SODE's and PDE's.
3. The final solution is expressed in a closed analytical form (neural network form) which is a differentiable form.
4. The final solution provides good generalization abilities, even outside the domain of the differential equation.
5. The proposed method is based on genetic algorithms, which means that it can be easily parallelized.
6. The proposed method can be extended by using different BNF grammar for the constructed neural networks with different topologies (such as recurrent neural networks) or different activations functions.
7. The method can be extended to solve higher order differential equations, a task that is currently being researched.

Acknowledgements

All the experiments of this paper were conducted at the Research Center of
is composed of 200
computing nodes with dual cpus (AMD OPTERON 2.2GHZ 64bit) running Redhat Enterprise Linux.

References

[1] A. R. Its, A. G. Izergin, V. E. Korepin, N. A. Slavnov, Differential equations for quantum correlation functions, International Journal of Modern Physics B **4**, pp. 1003-1037, 1990.

[2] A. V. Kotikov, Differential equations method: the calculation of vertex-type Feynman diagrams, Physics Letters B **259**, pp. 314-322, 1991.

[3] H. Gang, H. Kaifen, Controlling chaos in systems described by partial differential equations, Phys. Rev. Lett. **71**, pp. 3794 - 3797, 1993.

[4] C. J. Budd, A. Iserles, Geometric Integration: Numerical Solution of Differential Equations on Manifolds, Philosophical Transactions: Mathematical, Physical and Engineering Sciences **357**, pp. 945-956, 1999.

[5] Y.Z. Peng, Exact solutions for some nonlinear partial differential equations, Physics Letters A **314**, pp. 401-408, 2003.

[6] J. G. Verwer, J. G. Blom, M. van Loon, E. J. Spee, A comparison of stiff ODE solvers for atmospheric chemistry problems, Atmospheric Environment **30**, pp. 49-58, 1996.

[7] J. Behlke, O. Ristau, A new approximate whole boundary solution of the Lamm differential equation for the analysis of sedimentation velocity experiments, Biophysical Chemistry **95**, pp. 59-68, 2002.

[8] U. Salzner, P. Otto, J. Ladik, Numerical solution of a partial differential equation system describing chemical kinetics and diffusion in a cell with the aid of compartmentalization, Journal of Computational Chemistry **11**, pp. 194-204, 1990.

[9] R.V. Culshaw, S. Ruan, A delay-differential equation model of HIV infection of CD4+ T-cells, Mathematical Biosciences **165**, pp. 27-39, 2000.

[10] G.A. Bocharov, F.A. Rihan, Numerical modelling in biosciences using delay differential equations, Journal of Computational and Applied Mathematics **125**, pp. 183-199, 2000.

[11] R. Norberg, Differential equations for moments of present values in life insurance, Insurance: Mathematics and Economics **17**, pp. 171-180, 1995.

[12] J.C. Butcher, The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods,Wiley-Interscience New York, NY, USA, 1987.

[13] J. G. Verwer, Explicit Runge-Kutta methods for parabolic partial differential equations, Applied Numerical Mathematics **22**, pp. 359-379, 1996.

[14] A. Wambecq, Rational Runge-Kutta methods for solving systems of ordi- **20**, pp. 333-342, 1978.

[15] J. Douglas, B. F. Jones, On Predictor-Corrector Methods for Nonlinear Parabolic Differential Equations, Journal of the Society for Industrial and Applied Mathematics **11**, pp. 195-204, 1963.

[16] R. W. Hamming, Stable Predictor-Corrector Methods for Ordinary Differential Equations, Journal of the ACM **6**, pp. 37-47, 1959.

[17] J.D. Lambert, Numerical methods for Ordinary Differential Systems: The Initial Value Problem, John Wiley & Sons: Chichester, England, 1991.

[18] G.E. Fasshauer, Solving differential equations with radial basis functions: Multilevel methods and smoothing, Advances in Computational Mathematics **11**, pp. 139-159, 1999.

[19] C. Franke, R. Schaback, Solving partial differential equations by collocation using radial basis functions, Applied Mathematics and Computation **93**, pp. 73-82, 1998.

[20] I.E. Lagaris, A. Likas, D.I.Fotiadis, Artificial neural networks for solving ordinary and partialdifferential equations, IEEE Transactions on Neural Networks **9**, pp. 987-1000, 1998.

[21] P. Ramuhalli, L. Udpa, S.S. Udpa, Finite - element neural networks for solving differential equations, IEEE Transactions on Neural Networks **16**, pp. 1381-1392, 2005.

[22] D.R. Parisi, M.C. Mariani, M.A. Laborde, Solving differential equations with unsupervised neural networks, Chemical Engineering and Processing: Process Intensification **42**, pp. 715-721, 2003.

[23] L. Jianyu, L. Siwei, Q. Yingjian, H. Yaping, Numerical solution of differential equations by radial basis function neural networks, Proceedings of the International Joint Conference on Neural Networks **1**, pp. 773-777, 2002.

[24] S. He, K. Reif, R. Unbehauen, Multilayer neural networks for solving a class of partial differential equations, Neural Networks **13**, pp. 385-396, 2000.

[25] F. Puffer, R. Tetzlaff, D. Wolf, Learning algorithm for cellular neural networks (CNN) solving nonlinear partial differential equations, Conference Proceedings of the International Symposium on Signals, Systems and Electronics, pp. 501-504, 1995.

[26] A.J. Meade Jr., A.A. Fernandez, Solution of nonlinear ordinary differential equations by feedforward neural networks, Mathematical and Computer Modelling **20**, pp. 19-44, 1994.

[27] A.J. Meade Jr., A.A. Fernandez, The numerical solution of linear ordinary 94, Mathematical and Computer Modelling **19**, pp. 1-25, 1994.

[28] G. Burgess, Find approximate analytic solutions to differential equations using genetic programming, Surveillance Systems Division, Electronics and Surveillance Research Laboratory, Department of Defense, Australia, 1999.

[29] I.G. Tsoulos, I.E. Lagaris, Solving differential equations with genetic programming, Genetic Programming and Evolvable Machines **7**, pp. 33-54, 2006.

[30] I.G. Tsoulos, D. Gavrilis, E. Glavas, Neural Network Construction and Training using Grammatical Evolution, accepted for publication in Neurocomputing.

[31] M. O'Neill, C. Ryan, Grammatical Evolution, IEEE Trans. Evolutionary Computation **5**, pp. 349-358, 2001.

[32] S.E. Fahlman, C. Lebiere, The cascade-correlation learning architecture, Advances in Neural Information Processing Systems 2, pp. 524-532, 1990.

[33] J. R. Koza, Genetic Programming: On the programming of Computer by Means of Natural Selection. MIT Press: Cambridge, MA, 1992.

[34] M. O'Neill and C. Ryan, Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language, volume 4 of Genetic programming. Kluwer Academic Publishers, 2003.

[35] 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.

[36] Collins J. and Ryan C., Automatic Generation of Robot Behaviors using Grammatical Evolution, In Proc. of AROB 2000, the Fifth International Symposium on Artificial Life and Robotics.

[37] 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

[38] 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 26 June 2003, pp. 492-498, 2003.

[39] M.J.D Powell, A Tolerant Algorithm for Linearly Constrained Optimization Calculations, Mathematical Programming **45**, pp. 547-566, 1989.

Figure 1: An example grammar.

```
S::=<expr> (0)
<expr> ::= ( <expr> <op> <expr> ) (0)
            | <func> ( <expr> ) (1)
            |<terminal> (2)
 <op> ::= + (0)
            | - (1)
            | * (2)
            | / (3)
<func> ::= sin (0)
            |cos (1)
            |exp (2)
            |log (3)
<terminal>::=<xlist> (0)
            |<digitlist>.<digitlist> (1)
<xlist>::=x1 (0)
            | x2 (1)
            |x3 (2)
<digitlist>::=<digit> (0)
            | <digit><digitlist> (1)
<digit> ::= 0 (0) | 1 (1) | 2 (2) .... | 9 (9)
```

Table 1: An example of the mapping procedure.

String	Chromosome	Operation
<expr>	9,8,6,4,16,10,17,23,8,14	9 mod 3=0
(<expr><op><expr>)	8,6,4,16,10,17,23,8,14	8 mod 3=2
(<terminal><op><expr>)	6,4,16,10,17,23,8,14	6 mod 2=0
(<xlist><op><expr>)	4,16,10,17,23,8,14	4 mod 3=1
(x2<op><expr>)	16,10,17,23,8,14	16 mod 4=0
(x2+<expr>)	10,17,23,8,14	10 mod 3=1
(x2+<func>(<expr>))	17,23,8,14	17 mod 4=1
(x2+cos(<expr>))	23,8,14	23 mod 3=2
(x2+cos(<terminal>))	8,14	8 mod 2=0
(x2+cos(<xlist>))	14	14 mod 3=2
(x2+cos(x3))		

Figure 2: The proposed grammar

```
<S>::=<sigexpr> (0)
<sigexpr>::=<number>*sig(<sum>+<number>) (0)
            |<number>*sig(<sum>+<number>) <sigexpr> (1)
<sum>::=<number>*<xxlist> (0)
        |<sum>+<sum>      (1)
<xxlist>::=x1 (0) | x2 (1) | ... | x_d (d-1)
<number>::=(<digitlist>.<digitlist>) (0)
          |(-<digitlist>.<digitlist>) (1)
<digitlist>::=<digit>             (0)
             |<digit><digitlist>  (1)
<digit>::=0 (0) | 1 (1) | 2 (2) | 3 (3) | 4 (4)
        | 5 (5) |6  (6) | 7 (7) | 8 (8)  |9 (9)
```

Table 2: The numerical values for the parameters of the method

NAME	VALUE
S	500
K	2000
p_c	0.9
p_m	0.05
ϵ	10^{-6}
λ	100
G	20
M	20
T	100
B	10

Table 3: Experimental results of the proposed method

PROBLEM	TRAIN ERROR	TEST ERROR	GENERATIONS
ODE1	1.6×10^{-8}	1.5×10^{-8}	111
ODE2	5.0×10^{-8}	1.5×10^{-8}	78
ODE3	4.4×10^{-9}	4.2×10^{-9}	43
ODE4	1.0×10^{-9}	1.0×10^{-9}	189
ODE5	1.9×10^{-8}	2.0×10^{-8}	119
ODE6	4.7×10^{-8}	4.7×10^{-8}	65
ODE7	2.3×10^{-8}	2.5×10^{-8}	913
NLODE1	3.2×10^{-8}	3.1×10^{-8}	69
NLODE2	1.1×10^{-8}	1.1×10^{-8}	405
NLODE3	1.2×10^{-6}	1.2×10^{-6}	842
NLODE4	1.1×10^{-7}	1.1×10^{-7}	289
SODE1	2.3×10^{-5}	2.1×10^{-5}	1422
SODE2	2.8×10^{-6}	2.7×10^{-6}	1078
SODE3	2.1×10^{-5}	2.3×10^{-5}	1136
SODE4	5.8×10^{-7}	5.9×10^{-7}	827
PDE1	3.9×10^{-7}	1.6×10^{-7}	1034
PDE2	9.6×10^{-8}	5.1×10^{-8}	996
PDE3	8.5×10^{-8}	4.7×10^{-8}	189
PDE4	1.9×10^{-6}	1.1×10^{-6}	1267

Table 4: Solving the ODE1 problem for different values of parameter λ

λ	TRAIN ERROR	TEST ERROR	GENERATIONS
10	2.4×10^{-9}	2.5×10^{-9}	100
100	1.6×10^{-8}	1.5×10^{-8}	111
1000	2.2×10^{-10}	2.3×10^{-9}	100

Figure 3: The progress of solution for an example run for ODE1

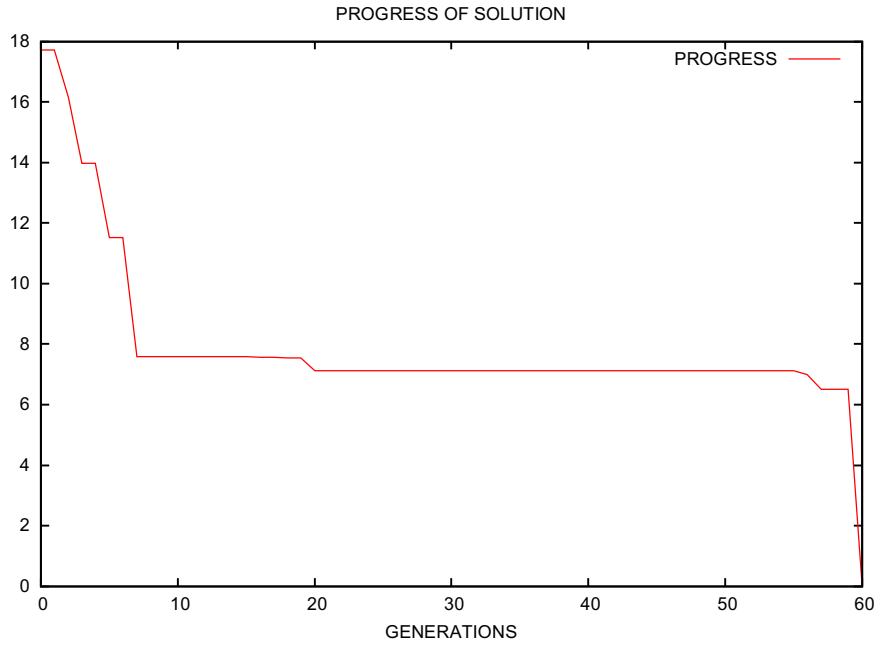

Figure 4: The quality of a trial solution for ODE1