Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2015, Article ID 910316, 11 pages
http://dx.doi.org/10.1155/2015/910316

Research Article

Closed-Form Solutions to Differential Equations via Differential Evolution

L. Mex,[1] Carlos A. Cruz-Villar,[2] and F. Peñuñuri[1]

[1]*Facultad de Ingeniería, Universidad Autónoma de Yucatán, Apartado Postal 150, Cordemex, 97310 Mérida, YUC, Mexico*
[2]*Departamento de Ingeniería Eléctrica, Cinvestav-IPN, Avenida IPN 2508, Apartado Postal 14-740, 07300 México, DF, Mexico*

Correspondence should be addressed to Carlos A. Cruz-Villar; cacruz@cinvestav.mx

Received 4 December 2014; Accepted 23 February 2015

Academic Editor: Garyfalos Papashinopoulos

We focus on solving ordinary differential equations using the evolutionary algorithm known as differential evolution (DE). The main purpose is to obtain a closed-form solution to differential equations. To solve the problem at hand, three steps are proposed. First, the problem is stated as an optimization problem where the independent variables are *elementary* functions. Second, as the domain of DE is real numbers, we propose a grammar that assigns numbers to functions. Third, to avoid truncation and subtractive cancellation errors, to increase the efficiency of the calculation of derivatives, the dual numbers are used to obtain derivatives of functions. Some examples validating the effectiveness and efficiency of our method are presented.

1. Introduction

Most of the problems in engineering and physics can be modeled as ordinary differential equations (ODEs). For this reason there are many studies addressing their solution. Regarding the deterministic arena, the most used methods are the Runge-Kutta methods [1–4], predictor-corrector methods [5–7], and radial basis functions methods [8–10]. Recently, some studies dedicated to solve differential equations using nondeterministic methods have been published. In [11], genetic algorithms are used to solve some differential equations appearing in economic sciences. In [12] a variational approach has been used in order to solve elliptic partial differential equations, and a genetic algorithm is used as the optimization method. In all the previously referenced articles—deterministic or not—the solution is given in a numerical approximated form. There are very few studies reporting closed-form solutions to differential equations. For example, using the evolutionary method known as grammatical evolution, [13] reports a method that produces closed-form solutions. Another approach that produces closed-form solutions to differential equations is [14], and the method used there is a hybrid method combining grammatical evolution and neural networks.

In this paper we propose a method based on the DE algorithm that obtains solutions to second-order ODEs as closed-form expressions. When the exact solution is not reached, the algorithm we propose converges to a solution that minimizes the objective functional of an optimization problem considering both, the differential equation and the boundary conditions. As DE was proposed to minimize real valued functions (i.e., not functionals), we propose a one-to-one grammar that assigns integer numbers to elementary functions; in this way, we are able to use DE to minimize the objective functional, which measures if a candidate function (a candidate function is the result of the evolution process obeying the DE algorithm) satisfies or not the ODE we want to solve.

In order to compute the first and second derivatives of the candidate function, we use the dual number approach. In this way, the derivatives are directly obtained without the use of a limit process, thus avoiding truncation and subtractive cancellation errors. All the programming functions required to solve an ODE are coded in Fortran language and they are included in a folder, which is provided as additional material to this paper, whose download link is as follows: http://www.meca.cinvestav.mx/personal/cacruz/archivos-ccv/.

The rest of the paper is organized as follows. Section 2 states the optimization problem and describes the classical

DE algorithm. Section 3 presents the proposal of the one-to-one grammar which allows using DE for minimization of functionals. Section 4 presents the dual number approach to obtain the first and second derivatives of the candidate functions. Section 5 works out several examples and applications of our method. Conclusions are presented in Section 6. Finally, two appendixes close the paper. Appendix A presents the way in which the candidate function is generated and evaluated. Appendix B presents graphs of the behavior of the DE algorithm for each worked-out example.

2. Statement of the Problem

Let us consider a second-order ordinary differential equation (1) defined on the real interval $[a, b]$, with boundary conditions (2) and (3), where $x_1 < x_2$ and $y \in C^2$. Note that it is not required that the functions $f : \mathbb{R}^4 \to \mathbb{R}$, $\mathbf{h}_1 : \mathbb{R}^3 \to \mathbb{R}^2$, and $\mathbf{h}_2 : \mathbb{R}^3 \to \mathbb{R}^2$ be differentiable:

$$f\left(x, y(x), y'(x), y''(x)\right) = 0, \tag{1}$$

$$\mathbf{h}_1\left(x_1, y(x_1), y'(x_1)\right) = \mathbf{0}, \tag{2}$$

$$\mathbf{h}_2\left(x_2, y(x_2), y'(x_2)\right) = \mathbf{0}. \tag{3}$$

The problem that we address in this paper is to find a closed-form expression for $y(x)$ satisfying (1), (2), and (3). Therefore, we rewrite the problem as that of minimizing the functional (4) under $y_s(x)$, where $\lambda_1 > 0$ and $\lambda_2 > 0$ are weighting factors (chosen by the user):

$$\begin{aligned} R = {} & \lambda_1 \left\| \mathbf{h}_1\left(x_1, y_s(x_1), y_s'(x_1)\right)\right\|^2 \\ & + \lambda_2 \left\| \mathbf{h}_2\left(x_2, y_s(x_2), y_s'(x_2)\right)\right\|^2 \\ & + \int_{x_1}^{x_2} f^2\left(x, y_s(x), y_s'(x), y_s''(x)\right) dx. \end{aligned} \tag{4}$$

If there exists a function $y_s(x)$ for which $R = 0$, then $y_s(x)$ will be the solution $y(x)$ to (1), satisfying (2) and (3). As the approach we follow to minimize (4) is evolutive, we use the differential evolution algorithm which is a simple yet powerful evolutionary algorithm for global optimization introduced by Storn and Price [15], which is presented below.

2.1. Differential Evolution. The DE algorithm has gradually become more popular and has been used in many practical cases. It only requires information about the objective function itself, which does not need to be a differentiable function, and the state space of possible solutions can be disjoint and can encompass infeasible regions [16]. Below, the original version of the method—known as DE/rand/1/bin—is outlined [17].

(1) The population is described by

$$\begin{aligned} \mathbf{P}_{\mathbf{x},g} &= \left(\mathbf{x}_{i,g}\right), \quad i = 1, \ldots, m;\ g = 0, \ldots, g_{\max}, \\ \mathbf{x}_{i;g} &= \left(x_{i;g}^j\right), \quad j = 1, \ldots, D, \end{aligned} \tag{5}$$

where D, m, and $g_{\max}$ represent the dimensionality of $\mathbf{x}$, the number of individuals, and the number of generations, respectively.

(2) Initialization of population is as follows:

$$x_{i;0}^j = \text{rand}^j(0,1) \cdot \left(b_U^j - b_L^j\right) + b_L^j. \tag{6}$$

Vectors $\mathbf{b}_U$ and $\mathbf{b}_L$ are the parameter limits and $\text{rand}^j(0,1)$ is a random number in $[0,1)$ generated for each parameter.

(3) Mutation is as follows:

$$\mathbf{v}_{i;g} = \mathbf{x}_{r_0;g} + F \cdot \left(\mathbf{x}_{r_1;g} - \mathbf{x}_{r_2;g}\right). \tag{7}$$

$\mathbf{x}_{r_0;g}$ is called the base vector which is perturbed by the difference of two other vectors.

$r_0, r_1, r_2 \in \{1, 2, \ldots, m\}$, $r_1 \neq r_2 \neq r_3 \neq i$. F is a scale factor greater than zero.

(4) Crossover is as follows.

A dual recombination of vectors is used to generate the trial vector:

$$\mathbf{u}_{i;g} = u_{i;g}^j = \begin{cases} v_{i;g}^j & \text{if } \text{rand}^j(0,1) \leqslant \text{Cr or } j = j_{\text{rand}} \\ x_{i;g}^j & \text{otherwise.} \end{cases} \tag{8}$$

The crossover probability, $\text{Cr} \in [0,1]$, is a user-defined value, $j_{\text{rand}} \in [1, D]$.

(5) Selection is as follows.

The selection is made according to

$$\mathbf{x}_{i;g+1} = \begin{cases} \mathbf{u}_{i;g} & \text{if } f\left(\mathbf{u}_{i;g}\right) \leqslant f\left(\mathbf{x}_{i;g}\right) \\ \mathbf{x}_{i;g} & \text{otherwise.} \end{cases} \tag{9}$$

In our study we use the DE/rand/1/bin method with the *dither* variant, which means that the parameter F is taken to be a random number—in our case $F \in (0,1)$.

3. Construction of Functions

As we can see, the DE method was designed to seek the optimum individual $\mathbf{x}$ on a real continuum domain. Since we are interested in solving differential equations (i.e., minimizing a functional), our individuals are functions. Therefore, we associate a function with a real vector; once this is done, the DE method can be applied as usual.

In order to assign a function to a real vector we propose the grammar shown in Table 1. The relation between a set of numbers and a function can be done by using a parse tree. This parse tree should be read from top to down and from left to right. Table 2 shows an example of a function construction and Figure 1 shows the corresponding parse tree. We can get an easy understanding of the function construction by conceptualizing the operators $+, *, /$, as functions of two arguments. For example, the expression 1 + 2 can be seen as

Table 1: Proposed grammar.

String	Associated number
Integer from one to ten	1 : 10
x	11
+	12
*	13
/	14
pow()	15
−()	16
sin()	17
cos()	18
exp()	19
log()	20
$\sqrt{()}$	21
tan()	22
arcsin()	23
arccos()	24
arctan()	25
erf()	26
sinh()	27
cosh()	28
tanh()	29

Table 2: Construction of the function $f(x) = \sin x + \cos x$.

Chromosome	String
12:	+
12, 17:	sin()+
12, 17, 11:	sin(x) +
12, 17, 11, 18:	sin(x) + cos()
12, 17, 11, 18, 11:	sin(x) + cos(x)

plus(1, 2) where the plus function is defined as plus(x, y) = $x + y$.

The set of integers related to a mathematical function can be manipulated by the DE method but the mutation operator will produce a set of real numbers that will not necessarily be a set of integers. This is addressed by taking the integer part of the numbers or by using the floor (ceiling) function.

4. Evaluation of the Constructed Function and Its Derivatives

For the evaluation of the constructed function we have written a Fortran parse function that receives the integer vector generated by the proposed grammar and produces a candidate function and its derivatives (first and second) evaluated at some specified point. The construction of this programming function is explained in Appendix A.

Traditional methods for calculating numerical derivatives (finite-difference) are subject to both truncation and subtractive cancellation errors. These problems are avoided by using dual functions. The approach to obtain first order derivatives by using dual functions is well known [18–20]. However, in order to make the paper self-contained this section presents the essential ideas as follows [20].

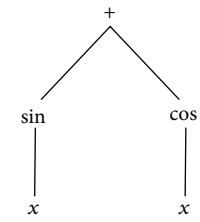

Figure 1: Parse tree for the function $f(x) = \sin x + \cos x$.

A dual number is a number of the form $\widehat{x} = \alpha + \epsilon\beta$ where $\alpha, \beta \in \mathbb{R}$, the field of the real numbers, and $\epsilon^2 = 0$. From the Taylor theorem if a function $y(x)$ is analytic, then

$$y(x+h) = y(x) + y'(x)\,h + \frac{y''(x)}{2}h^2 + \cdots; \tag{10}$$

a substitution of $x + h \rightarrow x + \epsilon$ in the above formula will give

$$y(x+\epsilon) = y(x) + y'(x)\,\epsilon. \tag{11}$$

The function $y(x + \epsilon)$ is called a dual function $\widehat{y}$ of the dual variable $\widehat{x} = x+\epsilon$. So if we substitute all of our real numbers by dual numbers and make the coefficient β of the dual variable equal to one, we end up with a dual function whose real term is the original function and the dual term is its derivative. Another convenient notation for the function $\widehat{y}$ is

$$\widehat{y}(\widehat{x}) = \{y_0, y_1\}, \tag{12}$$

where $y_0 = y(x)$ and $y_1 = y'(x)$.

Applying the chain rule we can dualize the composition of $y(x)$ with the function $u(x)$:

$$\widehat{y}(\widehat{u}) = \{y_0(u_0), y_1(u_0)\,u_1\}. \tag{13}$$

From this, a generalization to second derivatives is straightforward. Using a tilde to denote such a generalization we have

$$\begin{aligned} \widetilde{x} &= \{x, 1, 0\}, \\ \widetilde{y}(\widetilde{x}) &= \{y_0, y_1, y_2\}, \end{aligned} \tag{14}$$

where $y_0 = y(x)$, $y_1 = y'(x)$, and $y_2 = y''(x)$. Similarly, for the composition $y(u(x))$, we will have

$$\widetilde{y}(\widetilde{u}) = \left\{y_0(u_0), y_1(u_0)\,u_1, y_2(u_0)\,u_1^2 + y_1(u_0)\,u_2\right\}. \tag{15}$$

5. Experimental Results

This section presents the results that are obtained when we apply our proposal to obtain closed-form solutions to the case studies considered in [13]. All the cases with closed-form solutions were reproduced. Below, we present our results for some interesting cases.

The experiments were performed on an Intel Core i5-3230M @ 2.60 GHz processor running Debian GNU/Linux and using the Intel Fortran Compiler. In all the cases we used 100 equally spaced points to evaluate (4), $\lambda_1 = \lambda_2 = 10$, and a crossover probability of 0.2.

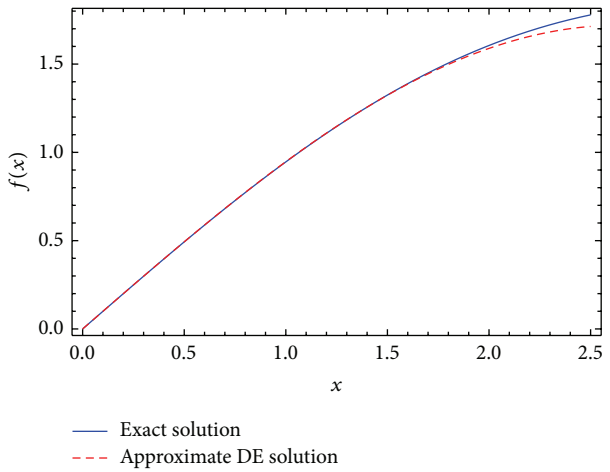

Figure 2: Exact and approximate solution for $y'' + y'/x - (1/x)\cos x = 0$ with $y(0) = 0$ and $y'(0) = 1$.

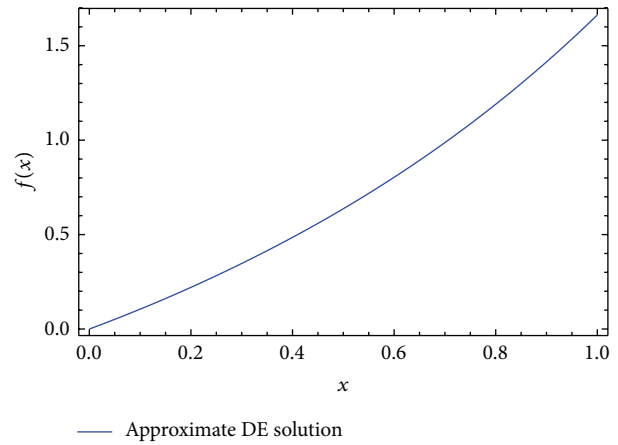

Figure 3: Approximate solution for $(x^2 + 1)y'' - 2xy - x^2 - 1 = 0$ with $y(0) = 0$ and $y'(0) = 1$.

Case 1. In this case, we want to find a closed-form solution to the differential equation (16), subject to boundary conditions $y(0) = 0$ and $y'(0) = 1$, with $x \in [0, 1]$:

$$y'' + \frac{y'}{x} - \frac{1}{x}\cos x = 0. \tag{16}$$

Since for the present case

$$\lim_{x \to 0}\left(\frac{y'}{x} - \frac{1}{x}\cos x\right) = 0, \tag{17}$$

it is not difficult to prove that (16) is equivalent to

$$xy'' + y' - \cos x = 0. \tag{18}$$

Even when both equations are equivalent, (18) is more adequate to the minimization of (4). The exact solution to (16), (18) is given by (19) and the approximated closed-form solution we have found is given by (20):

$$y(x) = \int_0^x \frac{\sin t}{t} dt, \tag{19}$$

$$y_s(x) = x\cos\left[\frac{x}{5\cos(\cos 9)}\right]. \tag{20}$$

Since the exact solution is known we can quantify the error in the interval $x \in [0, 1]$ as (21), which for this case resulted as $E^2 = 2.83 \times 10^{-7}$:

$$E^2 = \int_a^b \left[y(x) - y_s(x)\right]^2 dx. \tag{21}$$

In Figure 2, we show the exact and approximated solutions in the interval $[0, 2.5]$, since the error in the interval $[0, 1]$ is not enough to recognize any difference between both solutions. For this case we used a population of 100 individuals of dimension 20 and 5 000 generations.

Case 2. Let us consider the ODE

$$\left(x^2 + 1\right)y'' - 2xy - x^2 - 1 = 0 \tag{22}$$

with $x \in [0, 1]$ and initial conditions $y(0) = 0$ and $y'(0) = 1$. For this case a closed-form solution is not known. The approximate solution we found is

$$y_s(x) = e^x - e^{\sin^4[x/(x+\sin 8)]}, \tag{23}$$

giving a value of 1.76×10^{-4} for (4). In order to obtain (23) we used a population of 80 individuals of dimension 20 and 15 000 generations. In Figure 3 we show the approximate solution we found.

Case 3. Let us consider the ODE studied in [21]:

$$y' + y^2 = 2 + x^2 + 2x\tanh(x) \tag{24}$$

with $x \in [0, 1]$ and initial condition $y(0) = 0$. The proposed method is able to find the exact solution $y(x) = x + \tanh x$, when the population size is 100 individuals of dimension 30 and using 100 generations. Clearly the final value for objective functional (4) is zero.

Case 4. Let us consider the nonlinear differential equation [22]:

$$y'' - yy'' - 0.5y'^2 - 0.5 = 0 \tag{25}$$

with $x \in [0, 1]$ and boundary conditions $y(0) = 0$ and $y(1) = -0.5$. The approximate solution we obtained with a value of 3×10^{-3} for (4) using 100 individuals of dimension 30 and 10 000 generations was

$$y_s(x) = 0.125xe^{-\sinh(\mathrm{erf}(\cosh(x)))} - \mathrm{erf}\left(0.999978 - 0.479426^x\right). \tag{26}$$

In Figure 4 we show the approximate solution we found.

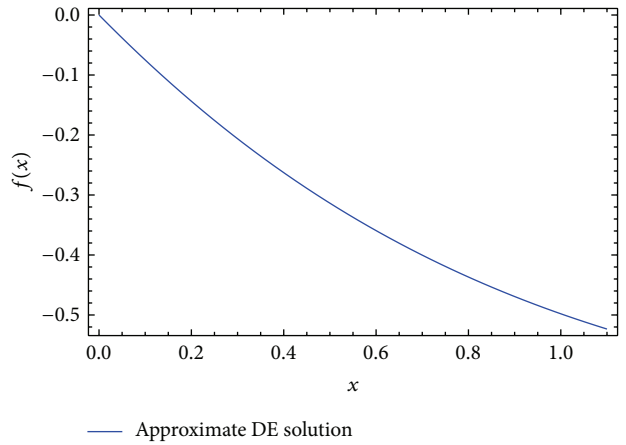

FIGURE 4: Approximate solution for $y'' - yy'' - 0.5y'^2 - 0.5 = 0$ with $y(0) = 0$ and $y(1) = -0.5$.

Case 5. Let us consider the temperature distribution equation on a uniformly thick rectangular fin radiation to free space studied in [23, 24]:

$$y'' - 2y^4 = 0 \tag{27}$$

with $x \in [0, 1]$ and boundary conditions $y(1) = 1$ and $y'(0) = 0$.

For this case, the approximate solution that the proposed methodology has obtained is

$$y_s(x) = \sin(x) \tan^{-1}(0.25 \tan(x)) + 0.693147, \tag{28}$$

when we use 300 individuals of dimension 30 and 10 000 generations. A final value of 9×10^{-4} was obtained for the objective functional (4). In Figure 5 we show the approximate solution we found.

Case 6. Consider the differential equation modeling the cooling process of a lumped system by combined convection and radiation [25]:

$$y' + y^4 + y = 0 \tag{29}$$

with $x \in [0, 1]$ and initial condition $y(0) = 1$.

Applying the proposed methodology we obtained the approximate solution

$$y_s(x) = \frac{1}{[\operatorname{erf}(x)]^{e^x} + e^x}. \tag{30}$$

For this case the value of (4) was 2×10^{-4}, when using 500 individuals of dimension 30 and 5 000 generations.

In Figure 6 we show the approximate solution we found.

Case 7. Consider the Duffing equation studied in [26]:

$$y''(x) + 0.4y'(x) + 1.1y(x) + y^3(x) = 2.1 \cos(1.8x), \tag{31}$$

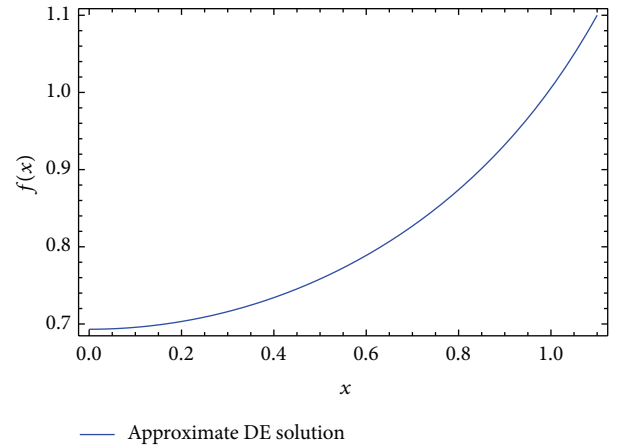

FIGURE 5: Approximate solution for $y'' - 2y^4 = 0$ with $y(1) = 1$ and $y'(0) = 0$.

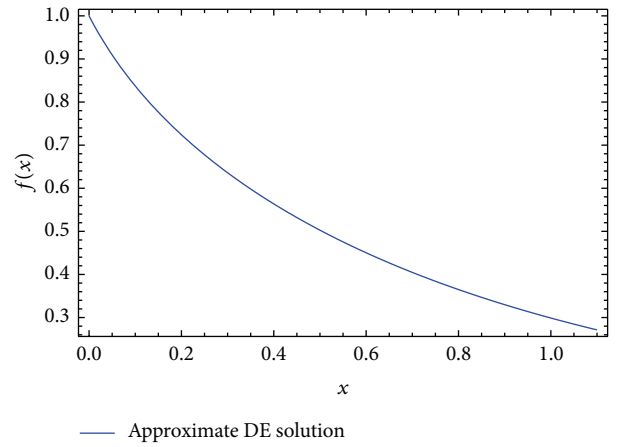

FIGURE 6: Approximate solution for $y' + y^4 + y = 0$ with $y(0) = 1$.

but with boundary conditions $y(0.5) = 0$, $y(1) = 1$. The approximate solution we found is

$$y_s(x) = \arctan\left(\ln\left(x^{\tanh(\cos^5(\tan(9+3+x)))}\right)\right) + \ln\left(\sin\left(\arctan\left(3^x + \sqrt{4}\right)\right)\right) + x + e^{-4}, \tag{32}$$

giving a value of 5.2×10^{-4} for (4). In order to obtain (32) we used a population of 500 individuals of dimension 30 and 15 000 generations. In Figure 7 we show the approximate solution we found.

Case 8. We close this section considering three examples of the Van der Pol equation:

$$\varepsilon y'' + \left(\delta + \beta y^2\right) y' - \mu y = F \sin(\omega t) \tag{33}$$

studied in [27].

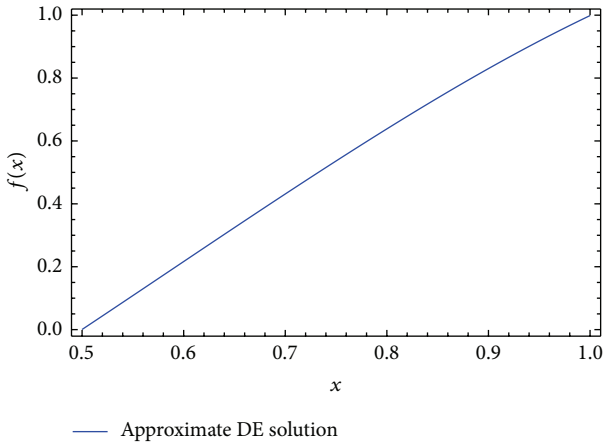

FIGURE 7: Approximate solution for $y''(x) + 0.4y'(x) + 1.1y(x) + y^3(x) = 2.1\cos(1.8x)$, with $y(0.5) = 0$ and $y(1) = 1$.

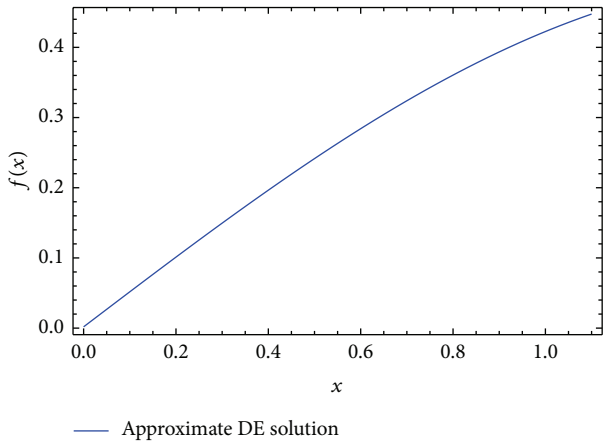

FIGURE 8: Approximate solution for $y'' - 0.05(1 - y^2)y' + y = 0$ with $y(0) = 0$ and $y'(0) = 0.5$.

Let us consider $\varepsilon = 1$, $\delta = -0.05$, $\beta = 0.05$, $\mu = -1$, and $F = 0$, so we have

$$y'' - 0.05\left(1 - y^2\right)y' + y = 0. \tag{34}$$

By applying the proposed methodology to this equation and considering boundary conditions $y(0) = 0$ and $y'(0) = 0.5$, we obtain

$$y_s(x) = 0.00182376 + 0.5\sin x. \tag{35}$$

For this case the final value of the objective functional (4) is 3×10^{-4}, when using 300 individuals of dimension 30 and 5 000 generations. In Figure 8 we show the approximate solution we have found.

Case 9. If we now consider $\varepsilon = 0.1$, $\delta = 1$, $\beta = -1$, $\mu = -1$, and $F = 0$ for (33), we obtain

$$0.1y'' + \left(1 - y^2\right)y' + y = 0. \tag{36}$$

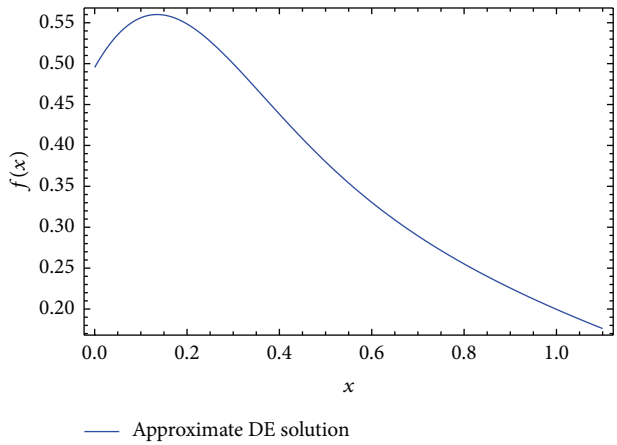

FIGURE 9: Approximate solution for $0.1y'' + (1 - y^2)y' + y = 0$ with $y(0) = 0.5$ and $y'(0) = 1$.

Considering boundary conditions $y(0) = 0.5$ and $y'(0) = 1$ and applying the proposed methodology, we obtain the approximate solution

$$y_s(x) = \tanh(\arctan(x)) + \arctan\left(\cos\left(\cosh^{\cosh^{0.205087}(x)}\cdot(\arctan(3.49384x))\right)\right). \tag{37}$$

For this case, the final value for the objective functional (4) is 7×10^{-3}, when using a population of 150 individuals of dimension 15 and 70 000 generations. In Figure 9 we show the approximate solution we have found.

Case 10. Taking $\varepsilon = 1000$, for (33) and considering the same boundary conditions and values for the other parameters as in the previous case, we have

$$1000y'' + \left(1 - y^2\right)y' + y = 0. \tag{38}$$

By applying the proposed methodology to solve this equation, we obtain the approximate solution

$$y_s(x) = (x + 0.498241)^{0.99918}. \tag{39}$$

For this case the value of (4) is 2×10^{-2}, when using 300 individuals of dimension 30 and 20 000 generations. Figure 10 shows the found approximate solution.

As a final remark, we want to point out that all the approximate solutions shown in this section were practically the same to those obtained by applying the Runge-Kutta (RK) algorithm. It is well known that the RK algorithm is by far the most used and in our opinion the best choice, for numerically solving a differential equation. Nevertheless, the proposed evolutionary approach is useful for cases when a closed-form solution is needed. Moreover, the introduced algorithms with dual numbers open a door for many applications.

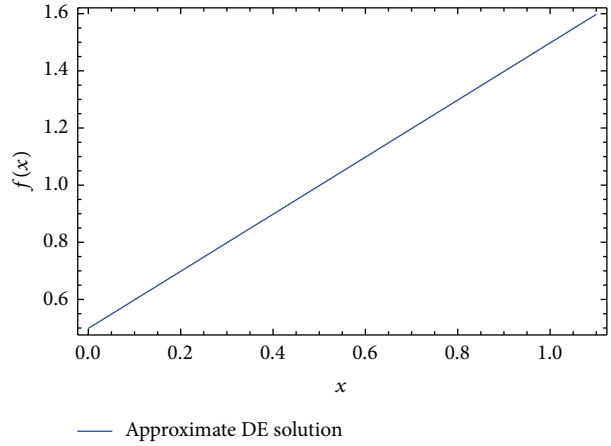

FIGURE 10: Approximate solution for $1000y'' + (1 - y^2)y' + y = 0$ with $y(0) = 0.5$ and $y'(0) = 1$.

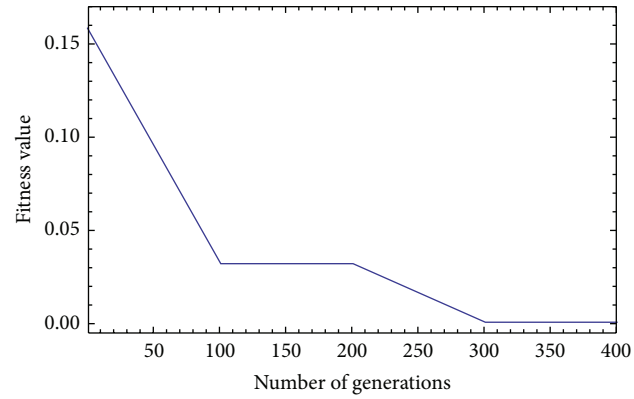

FIGURE 11: Performance graph for Case 1.

6. Conclusions

We have demonstrated the use of the differential evolution algorithm in order to obtain closed-form solutions to second-order differential equations. The main drawback for the application of the differential evolution algorithm to find exact closed-form solutions to differential equations was handled by constructing a function associated with a real vector. This function was constructed using a simple grammar and the concept of parse tree. This evolutionary algorithm along with the use of dual numbers in order to obtain the derivatives of a function produces an approximate solution expressed as a closed-form expression, even if the exact solution cannot be found (or it is not known as a closed-form expression). Thus, the proposed method could be efficient and useful for practical applications.

All the programming functions needed for the solution of the differential equations were coded in the Fortran language and provided as additional material to this paper.

Appendices

A. Parse Function

The function that maps the integer vector `x` associated with a function and evaluates it in the dual number `xval` is the function `parsv(x,xval)` and is coded in the module `parsef_mod.f90` of the additional material. Below we describe the construction of this function.

In what follows, NaN stands for not a number, $\widetilde{\text{nan}}$ stands for a dual number with, at least, its real component a NaN. The dual version of a number a is written as $\widetilde{a}$; for instance, the dual version of the number 2 is written as $\widetilde{2}$ and in our work is treated as the vector $[2, 0, 0]$.

The `parsv` function requires the following functions:

(i) `mynan()`

A NaN implementation.

(ii) `narg(x)`

Function that calculates the number of arguments for the elements of the grammar. For example, `narg(13)` = 2.

(iii) `operf(x,xval)`

Function that evaluates the functions of one argument in the dual point `xval` and then changes the real component of `xval` to NaN. For instance, `operf(17,xval)` will return `sindual(xval)` and the `xval` number is changed to $\widetilde{\text{nan}}$.

(iv) `operf2(x,vecval)`

Function that evaluates the functions of two arguments, that is, +, ∗, /, `power`. `vecval` is an array with dual numbers, possibly with NaN components. This function operates on the first two dual no-$\widetilde{\text{nan}}$ components of the vecval array and then changes its real components to NaN components. For instance, `operf2(12,[`$\widetilde{\text{nan}}$`,sindual(`$\widetilde{z}$`),` $\widetilde{\text{nan}}$`,`$\widetilde{1}$`])` will return `sindual(`$\widetilde{z}$`) +` $\widetilde{1}$ and then `vecval = [`$\widetilde{\text{nan}}$`, sindual(`$\widetilde{z}$`),`$\widetilde{\text{nan}}$`,`$\widetilde{1}$`]` is changed to [$\widetilde{\text{nan}}$, $\widetilde{\text{nan}}$, $\widetilde{\text{nan}}$, $\widetilde{\text{nan}}$].

The Fortran code for the `parsv` function is shown in Algorithm 1.

Let us analyze the above code for the case when `x = [12,17,13,2,11,20,18,14,11,5]` and `xval` = [1.1, 1, 0]. In this particular case, the dimension `length` of the vector `x` is `length` = 10 and `x` represents the function $f(x) = \sin(2x) + \log(\cos(x/5))$.

For the sake of concreteness, we will exemplify taking into account only the real component. The dual components will be omitted; notice however that the parsv function coded above takes into account such dual components.

The instruction `auxv = mynan()` sets `auxv` to `auxv = [N,N,N,N,N,N,N,N,N,N]` where `N` stands for NaN. In fact `auxv` is a 3×10 matrix but as we said before, we are ignoring the dual components.

The first loop evaluates the functions of zero arguments changing `auxv` to `auxv = [N,N,N,2,1.1,N,N,N,1.1,5]` as shown in Algorithm 2.

```
(1)  function parsv(x,xval) result(f_result)
(2)  implicit none
(3)  integer, intent(in):: x(:)
(4)  integer:: length, k
(5)  real(8), intent(in), dimension(3):: xval
(6)  real(8), dimension(3):: f_result
(7)  real(8), dimension(3,size(x)):: auxv
(8)
(9)   length = size(x)
(10)
(11) auxv = mynan()
(12) f_result = auxv(:,1)
(13)
(14) do k=1,length
(15)         if(x(k).ge.1.and. x(k).le. 10) auxv(:,k) = [1.d0*x(k), 0.d0, 0.d0]
(16)         if(x(k).eq.11) auxv(:,k) = xval
(17) end do
(18)
(19) f_result = auxv(:,1)
(20)
(21) do k=length-1,1,-1
(22)         if(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq. 0 ) then
(23)                 cycle
(24)
(25)         elseif(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq. 1 ) then
(26)                 auxv(:,k) = operf(x(k),auxv(:,k+1))
(27)                 if( isnan(auxv(1,k)) ) return
(28)
(29)         elseif(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq.2)then
(30)                 auxv(:,k) = operf2(x(k),auxv(:,k+1:length))
(31)
(32)                 if( isnan(auxv(1,k)) ) return
(33)         end if
(34) end do
(35)
(36)         f_result = auxv(:,1)
(37)
(38) end function parsv
```

Algorithm 1

```
do k=1,length
  if(x(k).ge.1.and. x(k).le. 10) auxv(:,k) = [1.d0*x(k), 0.d0, 0.d0]
  if(x(k).eq.11) auxv(:,k) = xval
end do
```

Algorithm 2

The second loop is a little more complicated but essentially evaluates the functions of one argument and two arguments. Let us analyze it in detail as shown in Algorithm 3.

For $k = 9$ the `if` condition is satisfied; thus the auxv vector is not changed.

For $k = 8$ the second `elseif` condition is satisfied and the `auxv` vector turns out to be

`auxv = [N,N,N,2,1.1,N,N,1.1/5,N,N]`, and so forth.

Finally for $k = 1$ we have

`auxv = [sin(2 * 1.1) + log(cos(1.1/5)),N,N,N,N,N,N,N,N,N]`.

For the case of taking into account the dual components, `auxv(1,1)` $= f(1.1)$, `auxv(2,1)` $= f'(1.1)$, and `auxv(3,1)` $= f''(1.1)$ which means that `parsv(1.1, xval)` $= [f(1.1), f'(1.1), f''(1.1)]$.

Notice that if the vector `x` does not represent a valid function, the function parsv will look from right to left for a

```
do k=length-1,1,-1
    if(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq. 0 ) then
        cycle
    elseif(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq. 1 ) then
        auxv(:,k) = operf(x(k),auxv(:,k+1))
        if( isnan(auxv(1,k)) ) return
    elseif(.not. isnan( auxv(1,k+1) ).and. narg(x(k)).eq.2)then
        auxv(:,k) = operf2(x(k),auxv(:,k+1:length))
        if( isnan(auxv(1,k)) ) return
    end if
end do
```

Algorithm 3

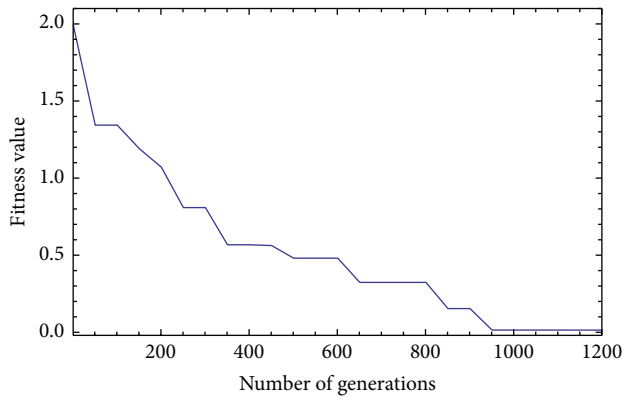

Figure 12: Performance graph for Case 2.

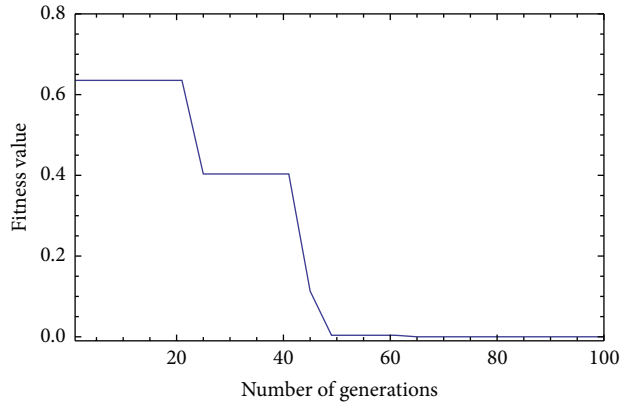

Figure 13: Performance graph for Case 3.

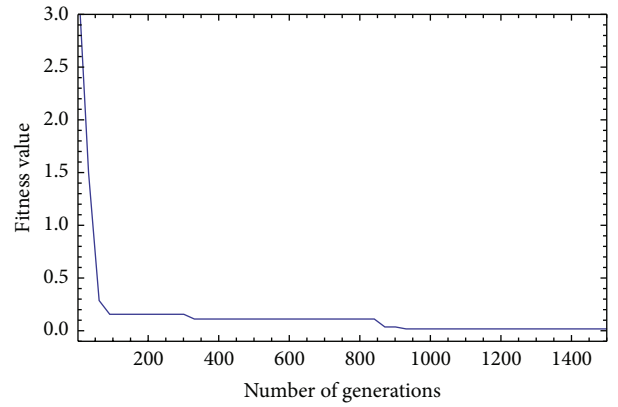

Figure 14: Performance graph for Case 4.

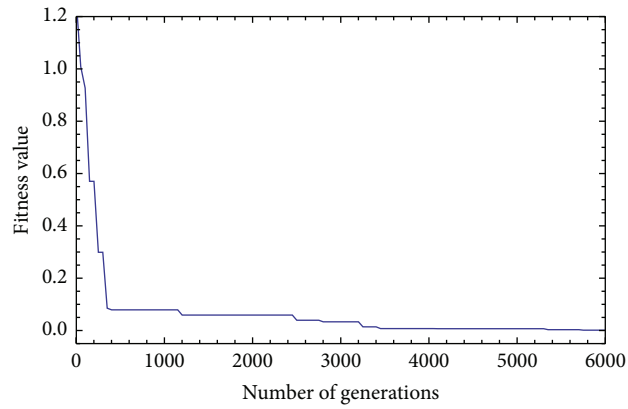

Figure 15: Performance graph for Case 5.

"subset" of x trying to construct a valid function; if it succeeds, the parsv function will return the value corresponding to that part of x corresponding to a valid function. For example, if `x = [17,13,18,2,11,16,14,19,11,5,11]`, we cannot form a valid function. However, the parsv function will return a right value corresponding to `xs =[17,13,18,2,11]`. Notice however that as long as the DE method is concern, x is always treated as `[17,13,18,2,11,16,14,19,11,5,11]`. This behavior for the function parsv could appear undesirable to first look. Nevertheless it is a nice property if we are thinking to use such a function in connection with an evolutionary algorithm. The reason is that if we discard those vectors that do not correspond to a valid function, the evolutionary method will have not enough individuals to evolve. Of course at the end we need to

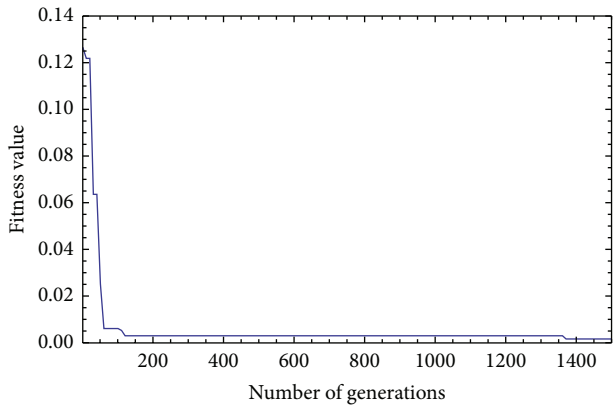

Figure 16: Performance graph for Case 6.

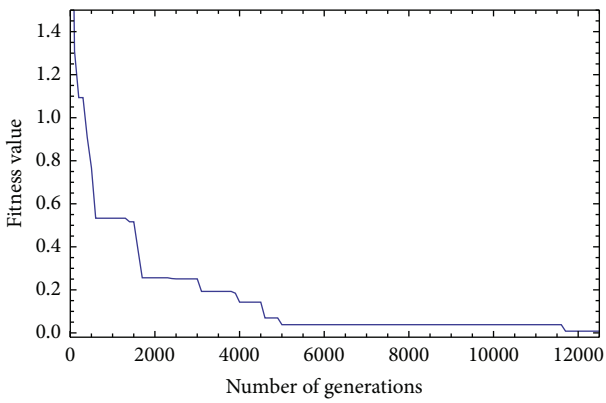

Figure 17: Performance graph for Case 7.

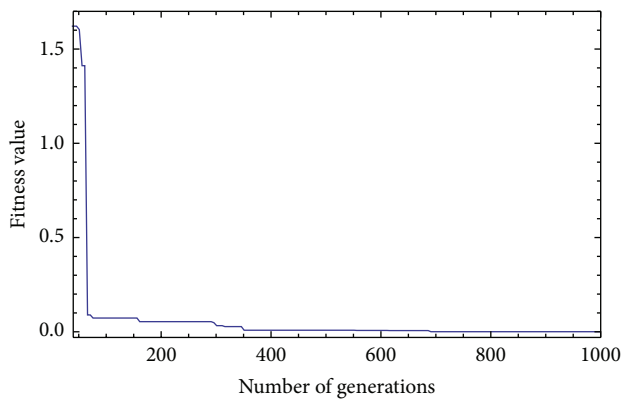

Figure 18: Performance graph for Case 8.

extract the subset of `x` which correspond to a valid function; this is done by the function `filterx` of the `parsef_mod` module.

B. Performance Graphs

This appendix shows the performance graphs of the DE method for all the studied cases as shown in Figures 11, 12, 13, 14, 15, 16, 17, 18, 19, and 20.

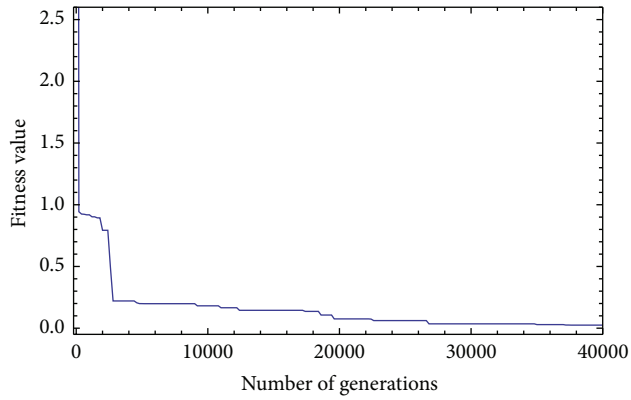

Figure 19: Performance graph for Case 9.

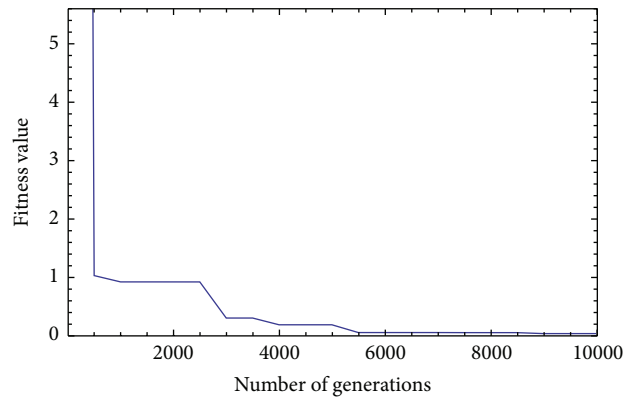

Figure 20: Performance graph for Case 10.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] A. Wambecq, "Rational Runge-Kutta methods for solving systems of ordinary differential equations," *Computing*, vol. 20, no. 4, pp. 333–342, 1978.

[2] J. Butcher, *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*, Wiley-Interscience, New York, NY, USA, 1987.

[3] P. J. van der Houwen and B. P. Sommeijer, "Parallel iteration of high-order Runge-Kutta methods with stepsize control," *Journal of Computational and Applied Mathematics*, vol. 29, no. 1, pp. 111–127, 1990.

[4] J. G. Verwer, "Explicit Runge-Kutta methods for parabolic partial differential equations," *Applied Numerical Mathematics*, vol. 22, no. 1–3, pp. 359–379, 1996.

[5] B. J. J. Douglas, "On predictorcorrector methods for nonlinear parabolic differential equations," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, pp. 195–204, 1963.

[6] K. Burrage, "Efficient block predictor-corrector methods with a small number of corrections," *Journal of Computational and Applied Mathematics*, vol. 45, no. 1-2, pp. 139–150, 1993.

[7] R. M. Thomas, T. E. Simos, and G. V. Mitsou, "A family of Numerov-type exponentially fitted predictor-corrector methods for the numerical integration of the radial Schrödinger equation," *Journal of Computational and Applied Mathematics*, vol. 67, no. 2, pp. 255–270, 1996.

[8] C. Franke and R. Schaback, "Solving partial differential equations by collocation using radial basis functions," *Applied Mathematics and Computation*, vol. 93, no. 1, pp. 73–82, 1998.

[9] 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.

[10] M. Dehghan and A. Shokri, "Numerical solution of the nonlinear Klein-Gordon equation using radial basis functions," *Journal of Computational and Applied Mathematics*, vol. 230, no. 2, pp. 400–410, 2009.

[11] G. D. Mateescu, "On the application of genetic algorithms to differential equations," *Romanian Journal of Economic Forecasting*, vol. 7, pp. 5–9, 2006.

[12] A. Sóbester, P. B. Nair, and A. J. Keane, "Genetic programming approaches for solving elliptic partial differential equations," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp. 469–478, 2008.

[13] I. G. Tsoulos and I. E. Lagaris, "Solving differential equations with genetic programming," *Genetic Programming and Evolvable Machines*, vol. 7, no. 1, pp. 33–54, 2006.

[14] I. G. Tsoulos, D. Gavrilis, and E. Glavas, "Solving differential equations with constructed neural networks," *Neurocomputing*, vol. 72, no. 10-12, pp. 2385–2391, 2009.

[15] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[16] D. B. Fogel, *Evolutionary Computation: Principles and Practice for Signal Processing*, SPIE, Bellingham, Wash, USA, 2000.

[17] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: a Practical Approach to Global Optimization*, Natural Computing Series, Springer, Berlin, Germany, 2005.

[18] Y.-L. Gu and J. Y. S. Luh, "Dual-number transformation and its applications to robotics," *IEEE Journal of Robotics and Automation*, vol. 3, no. 6, pp. 615–623, 1987.

[19] H. H. Cheng, "Programming with dual numbers and its applications in mechanisms design," *Engineering with Computers*, vol. 10, no. 4, pp. 212–229, 1994.

[20] F. Peñuñuri, R. Peon-Escalante, C. Villanueva, and C. A. Cruz-Villar, "A dual number approach for numerical calculation of derivatives and its use in the spherical 4R mechanism," http://arxiv.org/abs/1301.1409.

[21] T. A. Abassy, "Improved Adomian decomposition method (solving nonlinear non-homogenous initial value problem)," *Journal of the Franklin Institute*, vol. 348, no. 6, pp. 1035–1051, 2011.

[22] Z. Niu and C. Wang, "A one-step optimal homotopy analysis method for nonlinear differential equations," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 8, pp. 2026–2036, 2010.

[23] D. D. Ganji, "The application of He's homotopy perturbation method to nonlinear equations arising in heat transfer," *Physics Letters A*, vol. 355, no. 4-5, pp. 337–341, 2006.

[24] S. Abbasbandy, "The application of homotopy analysis method to nonlinear equations arising in heat transfer," *Physics Letters A*, vol. 360, no. 1, pp. 109–113, 2006.

[25] A. Rajabi, D. D. Ganji, and H. Taherian, "Application of homotopy perturbation method in nonlinear heat conduction and convection equations," *Physics Letters A*, vol. 360, no. 4-5, pp. 570–573, 2007.

[26] E. Yusufoğlu, "Numerical solution of duffing equation by the laplace decomposition algorithm," *Applied Mathematics and Computation*, vol. 177, no. 2, pp. 572–580, 2006.

[27] H. Kaur, R. C. Mittal, and V. Mishra, "Haar wavelet solutions of nonlinear oscillator equations," *Applied Mathematical Modelling*, vol. 38, no. 21-22, pp. 4958–4971, 2014.