Solving differential equations with global optimization techniques

Ioannis G. Tsoulos$^{(1)}$[*], Alexandros Tzallas$^{(1)}$, Dimitrios Tsalikakis$^{(2)}$

$^{(1)}$Department of Informatics and Telecommunications, University of Ioannina, 47100 Arta, Greece
$^{(2)}$University of Western Macedonia, Department of Engineering Informatics and Telecommunications, Greece

Abstract

The solution of differential equations finds many applications in a huge range of problems, and many techniques have been developed to approximate their solutions. This manuscript presents a number of global optimization techniques that have been successfully applied to train machine learning models to approximate differential equation solutions. These methods have been successfully applied to solving ordinary differential equations and systems of differential equations as well as partial differential equations with Dirichlet boundary conditions.

Keywords: Differential equations; global optimization; stochastic methods; machine learning

1 Introduction

The solution ordinary differential equations (ODEs), systems of differential equations (SODEs) and partial differential equations (PDEs) is commonly used in many scientific fields such as physics [1, 2], chemistry [3, 4, 5], economics [6, 7], biology [8, 9] etc. It is obvious that the numerical solution of differential equations has positive effects in many scientific areas and for this reason many techniques have been proposed in the modern literature. These methods can inclue the well - known Runge Kutta methods [10, 11, 12], wavelet transformations [13, 14, 15], predictor - corrector variations [16, 17], methods that incorporate artificial neural networks [18, 19] to solve differential equations [20, 21, 22], finite element methods [23, 24] etc. Also, recently a novell method based on Grammatical Evolution [25] has been proposed to tackle differential equations in closed analytical form by Tsoulos and Lagaris [26].

[*]Corresponding author. Email: itsoulos@uoi.gr

In this work, the differential equations to be solved are presented as machine learning models and the solution of the corresponding equation is reduced to a global optimization problem, where the task is to locate the global minimum of a multidimensional function $f(x):\ S \subset R^n \to R$ with $x_i\ \in [L_i, R_i]\,,\ i = 1..n$. The value of n is the amount of parameters in the corresponding machine learning model, for example the number of weights and biases in some artificial neural network. The vector L is considered as the lower bound of the parameter x and and the vector R as the upper bound. Machine learning models have been used a lot in solving differential equations, for example the use of SVM techniques [27, 28], the incorporation of deep learning methods [29, 30], constructed neural networks [31] etc. In the current work two machine learning models was tested: artificial neural networks and Radial Basis Function networks (RBF) [32]. The parameters in these machine learning models were trained using modified versions of two well-known global optimization techniques: Genetic Algorithms [33, 34, 35] and the PSO method [36, 37]. These models have been successfully used to solve ordinary differential equation, systems of differential equations as well as partial differential equations with Dirichlet boundary conditions.

The rest of this article is organized as follows: in section 2 the used models and the proposed modified global optimization techniques are outlined in detail, in section 3 the differential equations used in the experiments are listed as well as the experimental results and finally in section 4 some conclusions about the used methods are presented.

2 Method description

2.1 The neural network model

Artificial neural networks are parametric mathematical models used in many scientfic areas with great importance such as physics [38, 39, 40], chemistry [41, 42, 43], medicine [44, 45] etc. A neural network can be modelled as a function $N(\overrightarrow{x}, \overrightarrow{w})$, where the vector $\overrightarrow{x}$ is called the the input vector or pattern and $\overrightarrow{w}$ is called the weight vector. A method that trains a neural network should be used to estimate the vector $\overrightarrow{w}$ for a certain problem. The optimization technique minimizes the following quantity:

$$E\left(N\left(\overrightarrow{x}, \overrightarrow{w}\right)\right) = \sum_{i=1}^{M} \left(N\left(\overrightarrow{x}_i, \overrightarrow{w}\right) - y_i\right)^2 \tag{1}$$

In equation 1 (commonly addressed also as error function), the value y_i denotes actual output for the point $\overrightarrow{x_i}$. The form for the neural network is the same as in the case of [46]. Let us have a neural network with one processing layer and the output of each processing unit is given by:

$$o_i(x) = \sigma\left(p_i^T x + \theta_i\right), \tag{2}$$

with p_i the weight vector and θ_i is the bias for the processeing unit i. The function $\sigma(x)$ is well - known sigmoid function and it is given by:

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

If the neural network has H hidden nodes, then the output of the entire network is given by:

$$N(x) = \sum_{i=1}^{H} v_i o_i(x), \tag{4}$$

with v_i being the output weight for the processing node i. Therefore, if a single vector is used for both weights and biases we will have the following form for the artificial neural network:

$$N\left(\overrightarrow{x}, \overrightarrow{w}\right) = \sum_{i=1}^{H} w_{(d+2)i-(d+1)} \sigma \left(\sum_{j=1}^{d} x_j w_{(d+2)i-(d+1)+j} + w_{(d+2)i} \right) \tag{5}$$

where d is the dimension of vector $\overrightarrow{x}$.

2.2 The RBF model

An RBF network has also a lot of applications [47, 48, 49] and it is denoted as a function $r(x)$:

$$r(x) = \sum_{i=1}^{k} w_i \phi\left(\|x - c_i\|\right) \tag{6}$$

with $\overrightarrow{x}$ being the input vector and the vector $\overrightarrow{w}$ is considered as the weight vector. The function $\phi(x)$ used here is the following Gaussian function:

$$\phi(x) = \exp\left(-\frac{(x-c)^2}{\sigma^2}\right) \tag{7}$$

The function $\phi(x)$ has the property that its valued depends on the distance between the vectors $\overrightarrow{x}$, $\overrightarrow{c}$.

2.3 Calculation of error

This subsection details the calculation of the error of the machine learning models for each differential equation case. In each equation case the initial or boundary conditions are imposed using penalty factors.

2.3.1 Calculation for ODEs

The ODEs are defined as:

$$\psi\left(x, y, y^{(1)}, \ldots, y^{(n)}\right) = 0,\ x \in [a, b] \tag{8}$$

with $y^{(i)}$ the ith-order derivative of $y(x)$. The corresponding conditions are:

$$h_i\left(x,y,y^{(1)},\ldots,y^{(n)}\right)_{|x=t_i},i=1,\ldots,n \tag{9}$$

where t_i is either a or b. The calculation of the model error $f(r)$ of a model r are the following:

1. **Create** $T=\{x_1=a,x_2,x_3,\ldots,x_N=b\}$ a set of equidistant points.
2. **Calculate** the error value $E_r=\sum_{i=1}^{N}\psi\left(x_i,r\left(x_i\right),r^{(1)}\left(x_i\right),\ldots,r^{(n)}\left(x_i\right)\right)^2$
3. **Calculate** the penalty factor for the initial conditions:

$$P_r=\lambda\sum_{k=1}^{n}h_k^2\left(x,r(x),r^{(1)}(x),\ldots,r^{(n)}(x)\right)_{|x=t_k} \tag{10}$$

   , where $\lambda>0$.

4. **Return** $f(r)=E_r+P_r$

2.3.2 Calculation for SODEs

The system of odes used in the current work is in in the form:

$$\left(\begin{array}{ccc}\psi_1\left(x,y_1,y_1^{(1)},y_2,y_2^{(1)},\ldots,y_k,y_k^{(1)}\right) & = & 0\\ \psi_2\left(x,y_1,y_1^{(1)},y_2,y_2^{(1)},\ldots,y_k,y_k^{(1)}\right) & = & 0\\ \vdots & \vdots & \vdots\\ \psi_k\left(x,y_1,y_1^{(1)},y_2,y_2^{(1)},\ldots,y_k,y_k^{(1)}\right) & = & 0\end{array}\right) \tag{11}$$

with $x\in[a,b]$. The initial conditions of the system are defined as the following vector:

$$\left(\begin{array}{ccc}y_1(a) & = & y_{1a}\\ y_2(a) & = & y_{2a}\\ \vdots & \vdots & \vdots\\ y_k(a) & = & y_{ka}\end{array}\right) \tag{12}$$

The error value for a set of models $r_1,r_2,...,r_k$ is calulated using the following:

1. **Create** $T=\{x_1=a,x_2,x_3,\ldots,x_N=b\}$ a set of equidistant points.
2. **For every model** $r_i,\ i=1,..k$ **do**
   - (a) **Calculate** the error value: $E_{r_i}=\sum_{j=1}^{N}\left(\psi_i\left(x_j,r_1,r_1^{(1)},r_2,r_2^{(1)},\ldots,r_k,r_k^{(1)}\right)\right)^2$
   - (b) **Calculate** the corresponding penalty value: $P_{r_i}=\lambda\left(r_i\left(a\right)-y_{ia}\right)^2$
3. **EndFor**
4. **Calculate** the total error value: $f(r)=\sum_{i=1}^{k}\left(E_{r_i}+P_{r_i}\right)$

2.3.3 Calculation for PDEs

The partial differential equations solved and test in the curren work have the following form:

$$h\left(x, y, \Psi(x,y), \frac{\partial}{\partial x}\Psi(x,y), \frac{\partial}{\partial y}\Psi(x,y), \frac{\partial^2}{\partial x^2}\Psi(x,y), \frac{\partial^2}{\partial y^2}\Psi(x,y)\right) = 0 \qquad (13)$$

with $x \in [a,b],\ y \in [c,d]$. The boundary conditions considere in this manuscript are the Dirichlet boundary conditions with the following form:

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

The steps to calculate the fitness $f(g)$ for any given chromosome are the following:

1. **Define** the set $T = \{(x_1,y_1),(x_1,y_2),\ldots,(x_N,y_N)\}$ of $N \times N$ points in $[a,b] \times [c,d]$.
2. **Define** the set $x_B = \{x_{b1}, x_{b2}, \ldots, x_{bN}\}$ equidistant points in $[a,b]$.
3. **Define** the set $y_B = \{y_{b1}, y_{b2}, \ldots, y_{bN}\}$equidistant points in $[c,d]$.
4. **Calculate** the error value E_r as

$$E_r = \sum_{i=1}^{N} h\left(x_i, y_i, r\left(x_i,y_i\right), \frac{\partial}{\partial x} r\left(x_i,y_i\right), \frac{\partial}{\partial y} r\left(x_i,y_i\right)\right)^2$$

5. **Calculate** the associated penalty values:

$$\begin{array}{rcl} P_{1r} & = & \lambda \sum_{i=1}^{M}\left(r\left(a,y_{bi}\right) - f_0\left(y_{bi}\right)\right)^2 \\ P_{2r} & = & \lambda \sum_{i=1}^{M}\left(r\left(b,y_{bi}\right) - f_1\left(y_{bi}\right)\right)^2 \\ P_{3r} & = & \lambda \sum_{i=1}^{M}\left(r\left(x_{bi},c\right) - g_0\left(x_{bi}\right)\right)^2 \\ P_{4r} & = & \lambda \sum_{i=1}^{M}\left(r\left(x_{bi},d\right) - g_1\left(x_{bi}\right)\right)^2 \end{array}$$

6. **Calculate** the total fitness as $f(r) = E_r + P_{1r} + P_{2r} + P_{3r} + P_{4r}$

2.4 The used genetic algorithm

The Genetic Algorithm is an global optimization technique inspired by biology and it includes a series of genetic operations such as selection, crossover and mutation. This method initiates by creating a population of candidate solutions, called also chromosomes and subsquently these solutions are evolved through the application of the genetic opeations. The method has been applied with success in a variety of research problems such as electromangnetics [50], combinatorial problems [51], design of water distribution networks [52] etc In this work, a hybrid form of genetic algorithm is presented, where a local minimization method is periodically applied to a subset of the genetic algorithm's chromosomes. Furthermore, the termination rule has taken into account the nature of the problem.

The main steps of the used genetic algorithm has as follows:

1. **Initialization** step.
    (a) **Set** as N_c the number of chromosomes that will participate.
    (b) **Set** as N_g,the maximum number of allowed generations.
    (c) **Set** as p_m, the mutation rate.
    (d) **Set** as p_s, the selection rate.
    (e) **Set** as p_l, the local search rate.
    (f) **Set** ϵ a small positive number, i.e $\epsilon = 10^{-8}$.
    (g) Initialize randomly the chromosomes $x_i,\ i = 1, ..., N_c$
    (h) **Set** iter=0
2. **Check** for termination.
    (a) **Obtain** the best fitness
    $$f^* = \min_{i\in[0\ldots N_c]} f_i$$
    (b) **Terminate** if iter $\geq N_g$ OR $f^* \leq \epsilon$
3. **Calculate** the fitness.
    (a) **For** $i = 1, \ldots, N_c$ **do**
        i. **Create** a neural network or a RBF network using as parameters the chromosome x_i and denote this model as m_i. For the case of SODEs the chromosome is divided into k parts (the number of ODEs in the system) and a model $m_{ij},\ j = 1, .., k$ is constructed for every equation.
        ii. **Calculate** the fitness value $f_i = f(m_i)$ using the error equations of subsection 2.3.
        iii. **If** $r \leq p_l$, where $r \in [0, 1]$ a random number, apply a local search procedure to the model m_i and locate a better value f_i for the fitness. The used local search procedure is a BFGS method [57].
    (b) **EndFor**
4. **Application** of genetic operators.

(a) **Selection** operation. During selection, the chromosomes are classified according to their fitness. The first $(1-p_s)\times N_c$ are copied without changes to the next generation of the population. The rest will be replaced by chromosomes that will be produced at the crossover.

(b) **Crossover** operation. In the crossover operation $p_s \times N_c$ chromosomes are produced. For every couple of produced offsrpings two parents (z,w) are selected using the well - known procedure of tournament selection. For every pair (z,w) of parents, two offsprings $\tilde{z}$ and $\tilde{w}$ are produced according to the following equations:

$$\begin{aligned}\tilde{z_i} &= a_i z_i + (1-a_i)\, w_i \\ \tilde{w_i} &= a_i w_i + (1-a_i)\, z_i \end{aligned} \tag{14}$$

where a_i is a random number with the property $a_i \in [-0.5, 1.5]$ [58].

(c) **Mutation** operation. For each element of every chromosome a random number $r \in [0,1]$ is produced. Subsequently we change randomly the corresponding element if $r \le p_m$

(d) **Set** iter=iter+1

5. **Goto** step 2.

2.5 The used PSO method

The method PSO is a stochastic global optimization method and is based on a population of candidate solutions (particles) that move to search for the global minimum at some speed that is constantly changing. The speed of each solution is affected by the best position in which the specific particle has been found so far, but also by the overall best position of the swarm of particles.The PSO method has been applied on a wide range of applications [53, 54, 55, 56]. The steps of the modified PSO method are the following:

1. **Initialization**.

   (a) **Set** iter $= 0$.

   (b) **Set** as N_c the number of particles.

   (c) **Set** as N_g the maximum number of generations.

   (d) **Set** p_l the local search rate.

   (e) **Set** ϵ a small positive number, i.e $\epsilon = 10^{-8}$.

   (f) **Randomly** initialize the positions of the particles $x_1, x_2, ..., x_{N_c}$. The size of each particle is defined as n.

   (g) **Randomly** initialize the velocities of the particles $u_1, u_2, ..., u_{N_c}$ using the scheme

$$u_{ij} = L_j + r \times \frac{R_j - L_j}{20},\ i = 1, ..., N_c,\ j = 1, .., n$$

where r is a random number with $r \in [0,1]$, R_j is the lower bound for parameter j and R_j is the upper bound for parameter j.

(h) **For** $i = 1..N_c$ do $p_i = x_i$, where p_i is the best located position for particle i.

(i) **Set** $p_{\text{best}} = \arg\min_{i \in 1..N_c} f(x_i)$

2. **Termination Check**.

   (a) **Obtain** the best fitness

$$f^* = \min_{i \in [0 \ldots N_c]} f_i$$

   (b) **Terminate** if iter $\geq N_g$ OR $f^* \leq \epsilon$

3. **For** $i = 1..N_c$ **Do**

   (a) Update the velocity u_i

$$u_{ij} = u_{ij} + r_1 \times (p_{ij} - x_{ij}) + r_2 \times \left(p_{\text{best},j} - x_{ij}\right), \ j = 1,..,n$$

   (b) Update the position x_i as $x_i = x_i + u_i$

   (c) Evaluate the fitness of the particle x_i using the same scheme as for the genetic algorithm.

   (d) **If** $r \leq p_l$, where $r \in [0,1]$ a random number, apply a local search procedure to the model m_i and locate a better value f_i for the fitness of particle x_i.

   (e) If $f(x_i) \leq f(p_i)$ then $p_i = x_i$

4. **End For**

5. **Set** $p_{\text{best}} = \arg\min_{i \in 1..m} f(x_i)$

6. **Set** iter = iter + 1.

7. **Goto** Step 2

3 Experiments

A series of test functions used in various research papers [20, 26] have been used here for testing purposes.

3.1 Linear ode cases

ODE1

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

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

ODE2

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

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

ODE3

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

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

ODE4

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

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

ODE5

$$y'' = -100y$$

with $y(0) = 0,\ y'(0) = 10,\ x \in [0,1]$ and the solution is

$$y(x) = \sin(10x)$$

3.2 Non-linear ODEs

NLODE1

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

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

NLODE2

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

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

NLODE3

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

with $y(1) = 0,\ y(2) = \log(4),\ x \in [1, 2]$ and solution $y(x) = \log\left(x^2\right)$

NLODE4

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

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

3.3 Odes without analytic solution

UNSOLODE1

$$xy'' + y' - \cos(x) = 0$$

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

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

UNSOLODE2

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

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

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

3.4 Systems of ode cases

SODE1

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

with $y_1(0) = 0,\ y_2(0) = 0,\ x \in [0, 1]$. The analytical solutions are $y_1(x) = \sin(x),\ y_2(x) = x^2$.

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 $y_1(0) = 0,\ y_2(0) = 1,\ x \in [0,1]$ and solutions $y_1(x) = \frac{\sin(x)}{\exp(x)},\ y_2 = \exp(x)$

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

SODE4

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

with $y_1(0) = \cos(1.0),\ y_2(0) = 1.0,\ x \in [0,1]$ and solutions $y_1(x) = \cos\left(\exp(x)\right),\ y_2(x) = \exp(-x)$.

3.5 Pde cases

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 boundary conditions: $\Psi(0,y) = y^3,\ \Psi(1,y) = \left(1 + y^3\right)\exp(-1),\ \Psi(x,0) = x\exp(-x),\ \Psi(x,1) = (x+1)\exp(-x)$ The solution is given by: $\Psi(x,y) = \left(x + y^3\right)\exp(-x)$

PDE2

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

with $x \in [0,1],\ y \in [0,1]$ 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 analytical solution is $\Psi(x,y) = \sin(x)\cos(y)$.

Table 1: Parameter settings for the genetic algorithm.

PARAMETER	VALUE
N_c	500
N_g	2000
p_s	0.10
p_m	0.05
p_l	0.05
λ	100.0
N	20

PDE3

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

with $x \in [0,1]$, $y \in [0,1]$ 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 solution is: $\Psi(x,y) = x^2 + y^2 + x + y + 1$.

PDE4

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

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

3.6 Experimental results

Anafora sto OPTIMUS edo

4 Conclusions

References

[1] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, Journal of Computational Physics **357**, pp. 125-141, 2018.

[2] T. Lelièvre, G. Stoltz, Partial differential equations and stochastic methods in molecular dynamics. Acta Numerica **25**, pp. 681-880, 2016.

[3] G. Scholz, F. Scholz, First-order differential equations in chemistry, ChemTexts **1**, 2015.

Table 2: Experimental results using the hybrid genetic algorithm.

EQUATION	MLP-GEN	RBF-GEN	MLP-PSO	RBF-PSO
ODE1	1.6×10^{-3}	1.7×10^{-8}	1.6×10^{-3}	2.1×10^{-8}
ODE2	1.3×10^{-3}	1.3×10^{-7}	2.2×10^{-3}	2.0×10^{-8}
ODE3	1.6×10^{-15}	1.3×10^{-15}	6.5×10^{-12}	2.9×10^{-10}
ODE4	1.8×10^{-9}	1.7×10^{-10}	4.4×10^{-10}	5.7×10^{-9}
ODE5	3.9×10^{-5}	7.8×10^{-2}	3.8×10^{-10}	5.4×10^{-2}
NLODE1	1.3×10^{-15}	7.3×10^{-10}	1.9×10^{-9}	7.7×10^{-9}
NLODE2	2.7×10^{-8}	1.9×10^{-10}	7.8×10^{-12}	2.5×10^{-9}
NLODE3	3.5×10^{-6}	1.7×10^{-8}	1.7×10^{-8}	3.4×10^{-8}
NLODE4	4.5×10^{-8}	4.6×10^{-9}	3.4×10^{-8}	4.7×10^{-8}
UNSOLODE1	2.1×10^{-10}	5.8×10^{-11}	3.4×10^{-12}	1.7×10^{-8}
UNSOLODE2	1.2×10^{-15}	1.8×10^{-8}	1.3×10^{-12}	5.9×10^{-8}
SODE1	1.6×10^{-8}	1.6×10^{-8}	1.1×10^{-3}	1.4×10^{-8}
SODE2	1.3×10^{-9}	1.8×10^{-8}	4.1×10^{-7}	5.3×10^{-9}
SODE3	1.4×10^{-9}	6.7×10^{-9}	1.63×10^{-9}	7.7×10^{-9}
SODE4	3.1×10^{-3}	5.9×10^{-9}	1.5×10^{-4}	1.1×10^{-8}
PDE1	8.1×10^{-3}	5.5×10^{-2}	6.7×10^{-4}	5.9×10^{-3}
PDE2	7.6×10^{-5}	9.7×10^{-3}	3.5×10^{-6}	4.1×10^{-4}
PDE3	2.1×10^{-4}	2.1×10^{-10}	2.1×10^{-4}	1.8×10^{-8}
PDE4	5.6×10^{-4}	3.6×10^{-3}	1.7×10^{-4}	1.9×10^{-4}

[4] J.L. Padgett, Y. Geldiyev, S. Gautam, W. Peng, Y. Mechref, A. Ibraguimov, Object classification in analytical chemistry via data-driven discovery of partial differential equations. Comp and Math Methods, 2021.

[5] O. Owoyele, P. Pal, ChemNODE: A neural ordinary differential equations framework for efficient chemical kinetic solvers, Energy and AI **7**, 2022.

[6] Z. Wang, X. Huang, H. Shen, Control of an uncertain fractional order economic system via adaptive sliding mode, Neurocomputing **83**, pp. 83-88, 2012.

[7] Y. Achdou, F.J. Buera, J.M. Lasry, P.L. Lions, B. Moll, Partial differential equation models in macroeconomicsPhil. Trans. R. Soc. A. **372**, 2012.

[8] K. Hattaf, N. Yousfi, Global stability for reaction–diffusion equations in biology, Computers & Mathematics with Applications **66**, pp. 1488-1497, 2013.

[9] P. Getto, M. Waurick, A differential equation with state-dependent delay from cell population biology, Journal of Differential Equations **260**, pp. 6176-6200, 2016.

[10] C.A. Kennedy, M.H. Carpenter, Higher-order additive Runge–Kutta schemes for ordinary differential equations, Applied Numerical Mathematics **136**, pp. 183-205, 2019.

[11] H. Ranocha, D.I. Ketcheson, Relaxation Runge–Kutta Methods for Hamiltonian Problems, J Sci Comput **84**, 17 2020.

[12] J. Niegemann, R. Diehl, K. Busch, Efficient low-storage Runge–Kutta schemes with optimized stability regions, Journal of Computational Physics 231, pp. 364-372, 2012.

[13] Y. Wang. Q. Fan, The second kind Chebyshev wavelet method for solving fractional differential equations, Applied Mathematics and Computation **218**, pp. 8592-8601, 2012.

[14] M.H.Heydari, M.R.Hooshmandasl, F.Mohammadi, Legendre wavelets method for solving fractional partial differential equations with Dirichlet boundary conditions, Applied Mathematics and Computation **234**, pp. 267-276, 2014.

[15] B. Yuttanan, M. Razzaghi, Legendre wavelets approach for numerical solutions of distributed order fractional differential equations, Applied Mathematical Modelling **70**, pp. 350-364, 2019.

[16] H. Kim, R. Sakthivel, Numerical solution of hybrid fuzzy differential equations using improved predictor–corrector method, Communications in Nonlinear Science and Numerical Simulation **17**, pp. 3788-3794, 2012.

[17] V. Daftardar-Gejji, Y. Sukale, S. Bhalekar, A new predictor–corrector method for fractional differential equations, Applied Mathematics and Computation **244**, pp. 158-182, 2014.

[18] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.

[19] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control Signals and Systems 2, pp. 303-314, 1989.

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

[21] A. Malek, R.Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network—Optimization method, Applied Mathematics and Computation **183**, pp. 260-271, 2006.

[22] W. Zhang, W. Cai, FBSDE based neural network algorithms for high-dimensional quasilinear parabolic PDEs, Journal of Computational Physics **470**, 2022.

[23] P. Frauenfelder, C. Schwab, R.A. Todor, Finite elements for elliptic problems with stochastic coefficients, Computer Methods in Applied Mechanics and Engineering **194**, pp. 205-228, 2005.

[24] P. Houston, E. Süli, A note on the design of hp-adaptive finite element methods for elliptic partial differential equations, Computer Methods in Applied Mechanics and Engineering **194**, pp. 229-243, 2005.

[25] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation **5**, pp. 349-358, 2001.

[26] I.G. Tsoulos, I.E. Lagaris, Solving differential equations with genetic programming. Genet Program Evolvable Mach **7**, pp. 33–54, 2006.

[27] S. Mehrkanoon, T. Falck and J. A. K. Suykens, Approximate Solutions to Ordinary Differential Equations Using Least Squares Support Vector Machines, IEEE Transactions on Neural Networks and Learning Systems **23**, pp. 1356-1367, 2012.

[28] S. Mehrkanoon, J.A.K.Suykens, Learning solutions to partial differential equations using LS-SVM, Neurocomputing **159**, pp. 105-116, 2015.

[29] J. Sirignano, K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics **375**, pp. 1339-1364, 2018.

[30] M.Raissi, P.Perdikaris, G.E.Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics **378**, pp. 686-707, 2019.

[31] I.G. Tsoulos, D. Gavrilis, E. Glavas, Solving differential equations with constructed neural networks **72**, pp. 2385-2391, 2009.

[32] J. Park and I. W. Sandberg, Universal Approximation Using Radial-Basis-Function Networks, Neural Computation 3, pp. 246-257, 1991.

[33] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Reading, Massachussets, 1989.

[34] Z. Michaelewicz, Genetic Algorithms + Data Structures = Evolution Programs. Springer - Verlag, Berlin, 1996.

[35] S.A. Grady, M.Y. Hussaini, M.M. Abdullah, Placement of wind turbines using genetic algorithms, Renewable Energy **30**, pp. 259-270, 2005.

[36] J. Kennedy, R.C. Eberhart, The particle swarm: social adaptation in information processing systems, in: D. Corne, M. Dorigo and F. Glover (eds.), New ideas in Optimization, McGraw-Hill, Cambridge, UK, pp. 11-32, 1999.

[37] F. Marini, B. Walczak, Particle swarm optimization (PSO). A tutorial, Chemometrics and Intelligent Laboratory Systems **149**, pp. 153-165, 2015.

[38] P. Baldi, K. Cranmer, T. Faucett et al, Parameterized neural networks for high-energy physics, Eur. Phys. J. C **76**, 2016.

[39] J. J. Valdas and G. Bonham-Carter, Time dependent neural network models for detecting changes of state in complex processes: Applications in earth sciences and astronomy, Neural Networks **19**, pp. 196-207, 2006

[40] G. Carleo,M. Troyer, Solving the quantum many-body problem with artificial neural networks, Science **355**, pp. 602-606, 2017.

[41] Lin Shen, Jingheng Wu, and Weitao Yang, Multiscale Quantum Mechanics/Molecular Mechanics Simulations with Neural Networks, Journal of Chemical Theory and Computation **12**, pp. 4934-4946, 2016.

[42] Sergei Manzhos, Richard Dawes, Tucker Carrington, Neural network-based approaches for building high dimensional and quantum dynamics-friendly potential energy surfaces, Int. J. Quantum Chem. **115**, pp. 1012-1020, 2015.

[43] Jennifer N. Wei, David Duvenaud, and Alán Aspuru-Guzik, Neural Networks for the Prediction of Organic Chemistry Reactions, ACS Central Science **2**, pp. 725-732, 2016.

[44] Igor I. Baskin, David Winkler and Igor V. Tetko, A renaissance of neural networks in drug discovery, Expert Opinion on Drug Discovery **11**, pp. 785-795, 2016.

[45] Ronadl Bartzatt, Prediction of Novel Anti-Ebola Virus Compounds Utilizing Artificial Neural Network (ANN), Chemistry Faculty Publications **49**, pp. 16-34, 2018.

[46] I.G. Tsoulos, D. Gavrilis, E. Glavas, Neural network construction and training using grammatical evolution, Neurocomputing **72**, pp. 269-277, 2008.

[47] P. Teng, Machine-learning quantum mechanics: Solving quantum mechanics problems using radial basis function networks, Phys. Rev. E **98**, 033305, 2018.

[48] Chuanhao Wan and Peter de B. Harrington, Self-Configuring Radial Basis Function Neural Networks for Chemical Pattern Recognition, J. Chem. Inf. Comput. Sci. **39**, 1049–1056, 1999.

[49] Y.P. Wang, J.W. Dang, Q. Li and S. Li, Multimodal medical image fusion using fuzzy radial basis function neural networks, in 2007 International Conference on Wavelet Analysis and Pattern Recognition, pp. 778-782, 2017.

[50] R.L. Haupt, An introduction to genetic algorithms for electromagnetics, Antennas and Propagation Magazine 37, pp. 7-15, 1995.

[51] J.J. Grefenstette, R. Gopal, B. J. Rosmaita, D. Van Gucht, Genetic Algorithms for the Traveling Salesman Problem, In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 160 - 168, Lawrence Erlbaum Associates, 1985.

[52] D. A. Savic, G. A. Walters, Genetic Algorithms for Least-Cost Design of Water Distribution Networks, Journal of Water Resources Planning and Management 123, pp. 67-77, 1997.

[53] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, Y. Nakanishi, A particle swarm optimization for reactive power and voltagecontrol considering voltage security assessment, IEEE Transactions on Power Systems **15**, pp. 1232-1239, 2000.

[54] J. Robinson, Y. Rahmat-Samii, Particle swarm optimization in electromagnetics, IEEE Transactions on Antennas and Propagation **52**, pp. 397- 407, 2004.

[55] M. A. Abido, Optimal power flow using particle swarm optimization, International Journal of Electrical Power & Energy Systems **24**, pp. 563-571, 2002.

[56] Z.L. Gaing, Particle swarm optimization to solving the economic dispatch considering the generator constraints, IEEE Transactions on Power Systems **18**, pp. 1187-1195, 2003.

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

[58] P. Kaelo, M.M. Ali, Integrated crossover rules in real coded genetic algorithms, European Journal of Operational Research **176**, pp. 60-76, 2007.