axioms

MDPI

Article

RbfDeSolver: A Software Tool to Approximate Differential Equations Using Radial Basis Functions

Ioannis G. Tsoulos *,†, Alexandros Tzallas † and Evangelos Karvounis †

Department of Informatics and Telecommunications, University of Ioannina, 47150 Ioannina, Greece; tzallas@uoi.gr (A.T.); ekarvounis@uoi.gr (E.K.)
* Correspondence: itsoulos@uoi.gr
† These authors contributed equally to this work.

Abstract: A new method for solving differential equations is presented in this work. The solution of the differential equations is done by adapting an artificial neural network, RBF, to the function under study. The adaptation of the parameters of the network is done with a hybrid genetic algorithm. In addition, this text presents in detail the software developed for the above method in ANSI C++. The user can code the underlying differential equation either in C++ or in Fortran format. The method was applied to a wide range of test functions of different types and the results are presented and analyzed in detail.

Keywords: differential equations; neural networks; genetic algorithms

MSC: 65K05; 65L05

Citation: Tsoulos, I.G.; Tzallas, A.; Karvounis, E. RbfDeSolver: A Software Tool to Approximate Differential Equations Using Radial Basis Functions. *Axioms* **2022**, *11*, 294. https://doi.org/10.3390/axioms11060294

Academic Editor: Luigi Brugnano

Received: 17 May 2022
Accepted: 13 June 2022
Published: 16 June 2022

1. Introduction

A variety of problems in areas such as physics [1,2], chemistry [3–5], economics [6,7], biology [8,9], etc., can be modeled using ordinary differential equations (ODEs), systems of differential equations (SYSODEs) and partial differential equations (PDEs). Due to the importance of differential equations, several methods have appeared in the relevant literature, such as Runge–Kutta methods [10–12] or Predictor–Corrector methods [13,14]. Moreover, many methods based on machine learning models have appeared, such as methods that utilize neural networks [15–17], methods based on differential evolution techniques [18,19], genetic algorithms [20,21], etc. Furthermore, in recent years, a variety of methods that take advantage of modern GPU architectures have been published for the solution of differential equations [22–24]. In addition, a method based on Grammatical Evolution [25] has been introduced to solve differential equations in analytical form by Tsoulos and Lagaris [26], that creates solutions of differential equations in closed analytical form. Le et al. recently proposed [27] a Radial Basis Neural Network Approximation with extended precision for solving partial differential equations, and Wei et al. presented [28] a MATLAB code to solve differential equations with a conjunction of finite elements and Radial Basis Function network (RBF) neural networks. Additionally, a recent work based on quintic B-splines is proposed [29] for solving second-order coupled nonlinear Schrödinger equations. The current work proposes the incorporation of a modified genetic algorithm [30–32] and utilizes an RBF [33] to tackle the problem of solving differential equations. RBFs are usually expressed as:

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

where the vector $\overrightarrow{x}$ is considered the input vector and the vector $\overrightarrow{w}$ is denoted as the weight vector. In many cases the function $\phi(x)$ is a Gaussian function such as:

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

where the value $\phi(x)$ depends on the distance between the vectors $\overrightarrow{x}$, $\overrightarrow{c}$. The vector x is considered the input to the artificial neural network and the vector c is called the centroid for the corresponding function. The centroid is often calculated from the input vectors using clustering techniques such as the KMeans [34] algorithm.

RBF networks have been used in many practical problems in various areas, such as physics [35–38], chemistry [39–41], medicine [42–44], economics [45–47], etc. In the current work, the RBF network is used as an estimator of the differential equations for the cases of ODEs, SYSODEs, and PDEs. The enforcement of the initial and boundary conditions is done through penalization. The parameters of the network are adapted through a hybrid genetic algorithm. The proposed study aims to present an innovative methodology for solving differential equations using RBF artificial neural networks, which are distinguished for their ability to learn and adapt to complex computational problems. However, in order to better adapt the parameters of these networks, their training is performed using an extremely reliable global optimization method, such as genetic algorithms. However, although network training with a genetic algorithms can achieve more accurate results, it is a rather time-consuming technique and is demanding of computational resources. This means that the use of modern parallel processing techniques is required, which will make the most of modern computing structures such as those of multiple cores. In the case of the proposed algorithm and the accompanying computing tool, the OpenMP programming library [48] was chosen.

In addition, the used software tool is illustrated in detail and some examples of usage are presented. The tool is designed for UNIX systems equipped with the GNU C++ and Fortran 77 (g77) compilers. Furthermore, the software utilizes the qmake installation utility of the QT software library, freely available from https://qt.io (accessed on 10 May 2022).

The rest of this article is organized as follows: in Section 2 the proposed method is fully described; in Section 3 the software details are presented; in Section 4 the experimental results for some differential equations are presented; and finally in Section 5 some conclusions and guidelines for future improvements of the method and the accompanied software are given.

2. Detailed Description

In the proposed method, an artificial RBF network with n weights is used as a function estimator that solves a differential equation. The initial and boundary conditions are imposed by the use of punitive factors. The network parameters are estimated using a hybrid genetic algorithm. The genetic algorithms are biologically inspired programming tools that maintain and evolve a pool of candidate solutions to an optimization problem. The members of this pool are usually called chromosomes or genetic population. The evolution of the population is done through the operations of mutation and crossover. Among other advantages, genetic algorithms are distinguished for their simplicity in implementation, for the ease of their parallelization, their tolerance for errors, etc. The size of each chromosome in the used genetic algorithm is calculated as: $d \times n + n + n$, where the value d is 1 for ODEs and system of ODEs and 2 for PDEs. The first $d \times n$ values are used for the centroid vectors c_i of the Equation (1), the next n values are used for the σ values of every Gaussian unit and the remaining n values of the chromosome are used for the weights w_i of Equation (1). In addition, in the proposed implementation, a local optimization method is periodically applied to some randomly selected chromosomes of the population. This approach is performed in order to improve the accuracy of the solution produced by the genetic algorithm, but also to speed up the solution of the differential equation. The used local search procedure for this work was a BFGS variant of Powell [49].

In the following subsections, the proposed method is outlined in detail as well as the fitness calculation for every case of differential equation.

2.1. Main Algorithm

The steps of main the algorithm are described below:

1. **Initialization step**
   (a) **Set** *iter* = 0, as the current number of generations.
   (b) **Set** N_c, as the total number of chromosomes.
   (c) **Set** n, the number of weights in the RBF network.
   (d) **Initialize** randomly the chromosomes $X_i, i = 1 \ldots N_c$.
   (e) **Set** ITERMAX as the maximum number of generations.
   (f) **Set** p_s as the selection rate and p_m the mutation rate.
   (g) **Set** $f_l = \infty$, the best fitness in the population.
   (h) **Set** L_I, the number of generations to run before applying the local optimization method
   (i) **Set** L_c, the number of chromosomes that will involved in the local search procedure.
2. **Termination check. If** *iter* >ITERMAX **OR** $f_l \leq \epsilon$ terminate.
3. **Calculate** the fitness f_i for every chromosome x_i. The calculation procedure is described in Section 2.2.
4. **Genetic Operators**
   (a) **Selection** procedure: During selection, the chromosomes are classified according to their suitability. The best $(1 - p_s) \times N_c$ chromosomes are transferred 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** procedure: During this process, $p_s \times N_c$ chromosomes will be created. Firstly, for every pair of produced offspring, two distinct chromosomes (parents) are selected from the current population using tournament selection: First, a subset of $K > 1$ randomly selected chromosomes is created and the chromosome with the best fitness value is selected as parent. For every pair (z, w) of parents, two new offsprings $\tilde{z}$ and $\tilde{w}$ are created through the following:

$$\begin{aligned} \tilde{z}_i &= a_i z_i + (1 - a_i) w_i \\ \tilde{w}_i &= a_i w_i + (1 - a_i) z_i \end{aligned} \tag{3}$$

   where a_i is a random number with the property $a_i \in [-0.5, 1.5]$ [50].
   (c) **Mutation** procedure: For every element of each chromosome, select a random number $r \in [0, 1]$ and alter the corresponding chromosome if $r \leq p_m$.
5. **Set** *iter* = *iter*+1
6. **Local Search Step**
   (a) **If** $iters \mod L_I = 0$ **Then**
      i. **Select** a subset of L_C randomly chosen chromosomes from the genetic population. Denote this subset with L_S.
      ii. **For** every chromosome X_i **in** L_S
         A. **Start** a local search procedure $y = L(x_i)$
         B. **Set** $f_i = y$
   (b) **Endif**
7. **Denote** with f_l the best fitness value for the corresponding chromosome x_l
8. **Goto** step 2.

2.2. Fitness Evaluation

The evaluation of the fitness is different for every case of differential equations, although in every case penalization is used to enforce the initial or the boundary conditions of every case.

2.3. Ode Case

Consider an ODE in the following format:

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

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

$$h_i\left(x, y, y^{(1)}, \ldots, y^{(n)}\right)_{|x=t_i}, i = 1, \ldots, n \tag{5}$$

where t_i could be a or b. The steps for the calculation of the fitness $f(g)$ of a chromosome g are the following for the ODE case:

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

$$P_r = \lambda \sum_{k=1}^{n} h_k^2\left(x, r(x), r^{(1)}(x), \ldots, r^{(n)}(x)\right)_{|x=t_k} \tag{6}$$

where $\lambda > 0$.

5. **Return** $f(g) = E_r + P_r$

2.4. Systems of ODEs Case

The system of ODEs that should be solved is in the form:

$$\begin{pmatrix} \psi_1\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \\ \psi_2\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \\ \vdots & \vdots & \vdots \\ \psi_k\left(x, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \end{pmatrix} \tag{7}$$

with $x \in [a, b]$ and the initial conditions are expressed as:

$$\begin{pmatrix} y_1(a) & = & y_{1a} \\ y_2(a) & = & y_{2a} \\ \vdots & \vdots & \vdots \\ y_k(a) & = & y_{ka} \end{pmatrix} \tag{8}$$

The fitness calculation $f(g)$ for a given chromosome g has as follows:

1. **Create** $T = \{x_1 = a, x_2, x_3, \ldots, x_N = b\}$ a set of equidistant points.
2. **Split** the chromosome g into k parts and create the corresponding RBF networks $r_i = r(g_i)$
3. **Calculate** the errors: $E_{r_i} = \sum_{j=1}^{N}\left(\psi_i\left(x_j, r_1, r_1^{(1)}, r_2, r_2^{(1)}, \ldots, r_k, r_k^{(1)}\right)\right)^2$
4. **Calculate** the penalty values: $P_{r_i} = \lambda(r_i(a) - y_{ia})^2$
5. **Calculate** the total fitness value: $f(g) = \sum_{i=1}^{k}(E_{r_i} + P_{r_i})$

2.5. Pde Case

Consider a Pde in the following form:

$$h\left(x, y, \Psi(x, y), \frac{\partial}{\partial x}\Psi(x, y), \frac{\partial}{\partial y}\Psi(x, y), \frac{\partial^2}{\partial x^2}\Psi(x, y), \frac{\partial^2}{\partial y^2}\Psi(x, y)\right) = 0 \tag{9}$$

with $x \in [a,b]$, $y \in [c,d]$. For Dirichlet boundary conditions we have the following condition functions:

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. **Construct** the set $T = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$ uniformly sampled points in $[a,b] \times [c,d]$.
2. **Construct** the set $x_B = \{x_{b1}, x_{b2}, \ldots, x_{bM}\}$ equidistant points in $[a,b]$.
3. **Construct** the set $y_B = \{y_{b1}, y_{b2}, \ldots, y_{bM}\}$ equidistant points in $[c,d]$.
4. **Set** $r = r(g)$ the RBF network for the chromosome g.
5. **Calculate** the quantity E_r as

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

6. **Calculate** the following penalty values:

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

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

3. Software Details

3.1. Installation

The package is distributed in a zip file from the relevant GitHub URL https://github.com/itsoulos/RbfDeSolver (accessed on 10 May 2022) named RbfDeSolver-master.zip and under UNIX systems the user must execute the following commands to compile the software:

1. unzip RbfDeSolver-master.zip.
2. cd RbfDeSolver.
3. qmake.
4. make clean.
5. make.

The final outcome of the compilation is the software *RbfDeSolver*. The differential equations should be compiled separately: every differential equation is a different file to be compiled as a shared object using the qmake utility. For example, in order to compile the ODE of the file *ode1.so* located under examples subdirectory, the user should create a *ode1.pro* file with the following contents:

```
TEMPLATE=lib
SOURCES=ode1.cc
```

Afterwards, the compilation of the ode is done using the following commands:

1. qmake ode1.pro.
2. make.

The outcome of the compilation is the shared library *ode1.so*

3.2. Command Line Options

The software *RbfDeSolver* has the following command line options:

1. −−help. Prints a help screen and terminates.
2. −−kind = DE_KIND. The string value DE_KIND determines the kind of differential equation to be used and the accepted values are: ode, sysode, pde.
3. −−problem = FILE, the string parameter file determines the path to the differential equation to be solved.
4. −−count = K, set as K, the number of chromosomes in the genetic population. The default value is 500.
5. −−random = R, set as R, the seed for the random number generation.
6. −−generations = G, set as G, the maximum number of generations allowed. The default value is 2000.
7. −−epsilon = E, set as E, a small positive value used in the comparisons as well as the termination criterion of the genetic algorithm. The default value is 10^{-7}
8. −−weights = W, set as W, the number of weights for the RBF network. The default value is 1.
9. −−srate = S, set as S, the selection rate (parameter p_s) of the genetic algorithm. The default value is 0.1
10. −−mrate = M, set as M, the mutation rate (parameter p_m) of the genetic algorithm. The default value is 0.05
11. −−lg = G, set as G, the number of generations that should be passed in the genetic algorithm before the local search method is applied. The default value is 100.
12. −−li = I, set as I, the number of chromosomes that will participate in the local search procedure. The default value is 20.
13. −−threads = T, set as T, the number of OpenMp threads. The default value is 1.

3.3. Format for ODEs

In Figures 1 and 2 we present the formulation for ODEs in the languages C++ and Fortran correspondingly. The listed functions have the following meaning:

1. `getx0()`: Returns the lower boundary point, x_0.
2. `getx1()`: Returns the upper boundary point, x_1.
3. `getkind()`: Returns 1, 2 or 3:
   (a) If the value is 1 then the ODE is of first order and the boundary condition is of the form: $y(x_0) = y_0$.
   (b) If the value is 2 then the ODE is of second order with boundary conditions of the form: $y(x_0) = y_0$, $y'(x_0) = y'_0$.
   (c) Code 3 indicates that the ODE is of second order with boundary conditions of the form: $y(x_0) = y_0$, $y(x_1) = y_1$.
4. `getnpoints()`: Returns the number of equidistant training points (value N in Section 2.3)
5. `getf0()`: Returns the boundary condition on the left, y_0.
6. `getf1()`: Returns the boundary condition on the right, y_1.
7. `getff0()`: Returns the left boundary condition for second order ODEs y'_0.
8. `ode1ff(x,y,yy)`: If the ODE is of first order, then the purpose of the tool is to minimize the function ode1ff$(x, r(x), r^{(1)}(x))$, for different values of x in the range $[x_0, x_1]$. The parameter y represents $r(x)$ and the parameter yy represents $r^{(1)}(x)$.
9. `ode2ff(x,y,yy,yyy)`: If the ODE is of second order, then the `tool` tries to minimize the function ode2ff$(x, r(x), r^{(1)}(x), r^{(2)}(x))$, for different values of x in the range $[x_0, x_1]$. The parameter y represents $r(x)$, the parameter yy represents $r^{(1)}(x)$ and the parameter yyy represents $r^{(2)}(x)$.

3.4. Format for System of ODEs

In Figures 3 and 4 we demonstrate the formulation of System of ODES in C++ and in Fortran programming languages correspondingly. The functions used in those formulations have the following meanings:

1. `getx0()`: returns the left boundary, x_0.
2. `getx1()`: returns the right boundary, x_1.
3. `getnode()`: returns the number of ODEs in the system (parameter k in Section 2.4).
4. `getnpoints()`: R returns the number of equidistant training points (value N in Section 2.4)
5. `systemfun(k,x,y,yy)`: For the SYSODE case, the aim of the `RbfDeSolver` is to minimize the function systemfun$(k, x, Y, Y^{(1)})$ for values of x in the range $[x_0, x_1]$, where k is the total number of equations in the system, Y is the vector of Rbf networks $r_i(x)$, $i = 1 \ldots k$ and Y' is a vector with elements the first derivative of these k equations evaluated at x. The double precision array y stands for the vector Y and similar the double precision array yy represents the vector Y'.
6. **`systemf0(node,f0)`:** the argument node stands for the number of differential equations in the system and the double precision array f0 with node elements represents the vector holding the boundary conditions for each equation in the system (vector of Equation (8)).

```
extern "C"
{
double getx0()
{}

double getx1()
{}

int getkind()
{}

int getnpoints()
{}

double getf0()
{}

double getf1()
{}

double getff0()
{}

double ode1ff(double x,double y,double yy)
{}

double ode2ff(double x,double y,double yy,double yyy)
{}
}
```

Figure 1. Ode format in C++.

```
double precision function getx0()
end

double precision function getx1()
end

integer function getkind()
end

integer function getnpoints()
end

double precision function getf0()
end

double precision function getf1()
end

double precision function getff0()
end

double precision function ode1ff(x,y,yy)
double precision x,y,yy
end

double precision function ode2ff(x,y,yy,yyy)
double precision x,y,yy,yyy
end
```

Figure 2. Ode format in Fortran.

```
extern "C" {
double getx0()
{}

double getx1()
{}

int getnode()
{}

int getnpoints()
{}

double systemfun(int node, double x, double *y, double *yy)
{}

void systemf0(int node,double *f0)
{}
}
```

Figure 3. Format for SYSODEs in C++.

```
double precision function getx0()
end

double precision function getx1()
end

integer function getnode()
end

integer function getnpoints()
end

double precision function systemfun(node,x,y,yy)
integer node
double precision x
double precision y(node)
double precision yy(node)
end

integer function systemf0(node,f0)
integer node
double precision f0(node)
end
```

Figure 4. Format for SYSODEs in Fortran.

3.5. PDE Format

The system is capable of solving elliptic PDEs in two dimensions in a box $[x_0, x_1] \times [y_0, y_1]$ with the Dirichlet boundary conditions $\Psi(x_0, y) = f_0(y), \Psi(x_1, y) = f_1(y), \Psi(x, y_0) = g_0(x)$ and $\Psi(x, y_1) = g_1(x)$. In Figures 5 and 6 we can see the formulation of PDE's in C++ and Fortran programming languages. The presented functions have the following representation:

1. `getx0()`: returns the left boundary x_0.
2. `getx1()`: returns the right boundary x_1.
3. `gety0()`: returns the left boundary y_0.
4. `gety1()`: returns the right boundary y_1.
5. `getnpoints()`: returns the amount of interior training points for the PDE.
6. `getbpoints()`: returns the amount of training points across each boundary of the PDE.
7. `f0(y)`: returns the boundary condition $f_0(y)$ across $x = x_0$.
8. `f1(y)`: returns the boundary condition $f_1(y)$ across $x = x_1$.
9. `g0(x)`: returns the boundary condition $g_0(x)$ across $y = y_0$.
10. `g1(x)`: the function returns the boundary condition $g_1(x)$ across $y = y_1$.
11. `pde(x,y,v,x1,y1,x2,y2)`: For the PDE case `RbfDeSolver` minimizes the function $\text{pde}\left(x, y, r(x,y), \frac{\partial r(x,y)}{\partial x}, \frac{\partial r(x,y)}{\partial y}, \frac{\partial^2 r(x,y)}{\partial x^2}, \frac{\partial^2 r(x,y)}{\partial y^2}\right)$, where $x \in [x_0, x_1]$ and $y \in [y_0, y_1]$. The argument v corresponds to $r(x,y)$. The argument x1 corresponds to the first derivative of $r(x,y)$ with respect to x, y1 corresponds to the first derivative of $r(x,y)$ with respect to y, x2 corresponds to the second derivative of $r(x,y)$ with respect to x and the y2 corresponds to the second derivative of $r(x,y)$ with respect to y.

```
extern "C" {
double getx0()
{}

double getx1()
{}

double gety0()
{}

double gety1()
{
}

int getnpoints()
{}
int getbpoints()
{
}

double f0(double y)
{}

double f1(double y)
{}
double g0(double x)
{}

double g1(double x)
{}

double pde(double x, double y, double v, double x1, double y1,
double x2, double y2)
{}
}
```

Figure 5. Format for PDEs in C++.

```
double precision function getx0()
end

double precision function getx1()
end

double precision function gety0()
end

double precision function gety1()
end

integer function getnpoints()
end

integer function getbpoints()
end

double precision function f0(y)
double precision y
end

double precision function f1(y)
double precision y
end

double precision function g0(x)
double precision x
end

double precision function g1(x)
double precision x
end

double precision function pde(x,y,v,x1,y1,x2,y2)
double precision x,y,v,x1,y1,x2,y2
end
```

Figure 6. Format for PDEs in Fortran.

4. Experiments

A series of test functions used in various research papers [15,26] have been used here for testing purposes. All the problems have been coded in ANSI C++ and the execution was performed on a Intel i7-10700T running at 2.00 GHz with 16 GB of RAM, and the operating system was Debian Linux.

4.1. Linear ODEs

1. ODE1

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

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

2. ODE2

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

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

3. ODE3

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

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

4. 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 $y(x) = \exp\left(-\frac{x}{5}\right)\sin(x)$

5. ODE5

$$y'' = -100y$$

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

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

4.2. Non-Linear ODEs

1. LODE1

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

with $y(1) = 1$, $y(4) = 2$, $x \in [1, 4]$. The solution is $y(x) = \sqrt{x}$

2. 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 $y(x) = x + \sin(x)$

3. NLODE3

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

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

4. NLODE4

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

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

4.3. Systems of ODEs

1. SYSODE1

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

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

2. SYSODE2

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

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

3. SYSODE3

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

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 $y_1(x) = \sin(x)$, $y_2(x) = \cos(x)$, $y_3(x) = \sin(x)$, $y_4(x) = \cos(x)$, $y_5(x) = \sin(x)$.

4. SYSODE4

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

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

4.4. PDEs

1. PDE1

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

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

2. 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 $\Psi(x,y) = \sin(x)\cos(y)$.

3. 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: $\Psi(x,y) = x^2 + y^2 + x + y + 1$.

4. 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: $\Psi(x,y) = \sin(xy)$.

4.5. Experimental Results

To validate the ability of the proposed method to tackle differential equations, a series of experiments were made using the following values for the weight number of the Rbf network: $w = 5$, $w = 10$, $w = 15$. The values for the parameters of the experiments

are listed in Table 1. All the experiments were conducted 30 times using different seeds for the random number generator and the average error was measured. The random number generator used was the drand48() function of the C programming language. The experimental results are listed in Table 2.

Table 1. Experimental parameters.

Parameter	Value
N_c	1000
ITERMAX	5000
p_s	0.1
p_m	0.05
ϵ	10^{-7}
L_I	100
L_C	20
λ	100

Table 2. Experimental results.

Equation	w = 5	w = 10	w = 15
ODE1	3.9×10^{-6}	2.2×10^{-6}	3.4×10^{-6}
ODE2	2.1×10^{-5}	1.4×10^{-5}	1.5×10^{-5}
ODE3	6.6×10^{-2}	7.7×10^{-2}	9.4×10^{-2}
ODE4	8.8×10^{-7}	3.8×10^{-6}	8.7×10^{-7}
ODE5	9.4×10^{-1}	1.1×10^{-1}	5.9×10^{-2}
NLODE1	7.2×10^{-4}	1.6×10^{-5}	1.9×10^{-4}
NLODE2	4.6×10^{-4}	5.3×10^{-4}	1.1×10^{-4}
NLODE3	5.7×10^{-6}	7.9×10^{-6}	5.4×10^{-6}
NLODE4	1.2×10^{-4}	7.9×10^{-6}	2.6×10^{-5}
SYSODE1	2.8×10^{-6}	1.9×10^{-6}	3.2×10^{-6}
SYSODE2	1.4×10^{-5}	3.2×10^{-6}	3.9×10^{-6}
SYSODE3	3.1×10^{-5}	8.1×10^{-5}	1.1×10^{-5}
SYSODE4	1.4×10^{-4}	4.4×10^{-6}	6.4×10^{-6}
PDE1	3.7×10^{-2}	6.8×10^{-3}	1.1×10^{-2}
PDE2	3.2×10^{-3}	2.1×10^{-5}	2.2×10^{-5}
PDE3	7.9×10^{-2}	4.9×10^{-4}	7.4×10^{-4}
PDE4	8.0×10^{-2}	7.8×10^{-3}	3.4×10^{-3}

As the experimental results show, the proposed method solves the vast majority of differential equations even when the number of weights is relatively small. However, adding weights seems to have more positive effects on difficult problems especially in the case of partial differential equations. Of course, increasing the number of weights implies increased execution times and, for this reason, the use of parallel processing techniques is necessary. In the application, there is the possibility of using more processing threads through the OpenMP library.

In addition, the graphical representation of the produced solutions as well as the absolute error between the estimated RBF networks is also plotted. For example, in Figure 7 the solution of ODE1 and the estimated RBF network are plotted and in Figure 8 the absolute difference of these functions is plotted. Additionally, the absolute error between the solutions $y1(x) = \sin(x)$, $y(2) = x^2$ and the estimated RBF networks for the SYSODE1 case is also plotted in Figure 9. Moreover, the absolute error between the solution of the PDE1 case and the estimated RBF network is plotted in Figure 10. All graphs show the ability of the proposed method to approach to a large extent the solution of the differential equation which is under study.

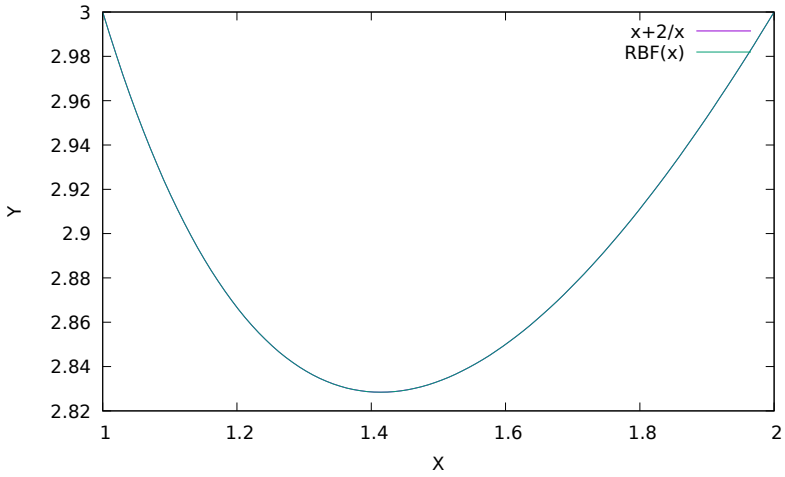

Figure 7. Graphical comparison between the solution of ODE1 and the estimated neural network.

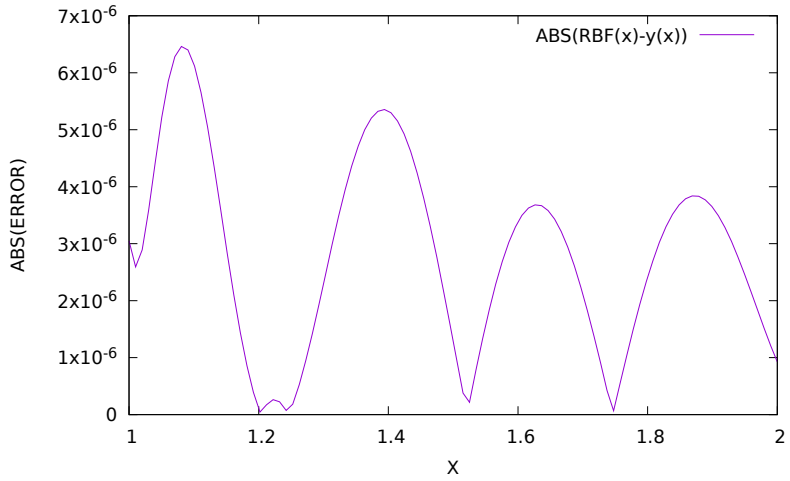

Figure 8. The absolute difference between the real solution of ODE1 and the produced RBF network.

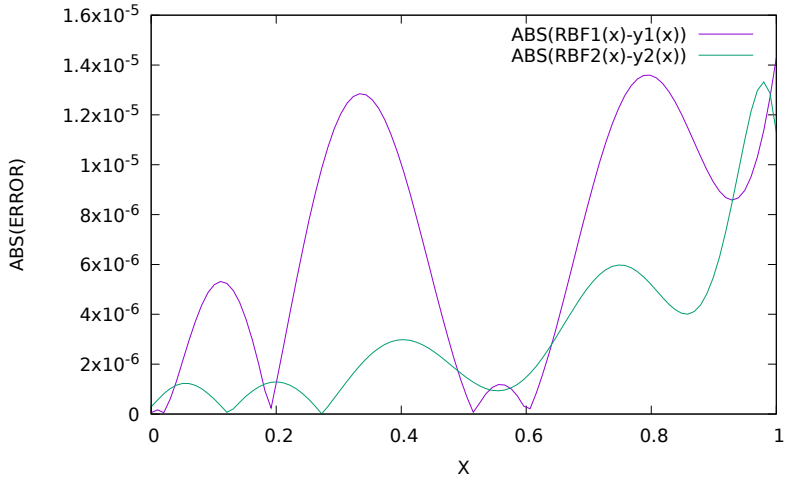

Figure 9. The difference between the functions $y1(x) = \sin(x)$, $y(2) = x^2$ and the estimated RBF solutions for the case of System of ODE's SYSODE1.

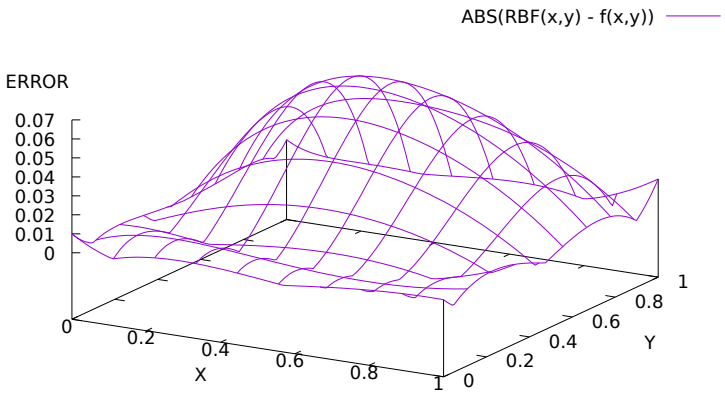

Figure 10. Absolute error between $f(x,y) = \left(x + y^3\right) \exp(-x)$ and the estimated solution of PDE1.

Additionally, a plot was made to demonstrate the speed of the algorithm and the effectiveness of using more threads. In this test, the method was tested in ODE1 function for different values for the number of the threads and the desired accuracy (the value $-\log 10(\epsilon)$ for the parameter ϵ in Table 2). The plot is shown in Figure 11. Execution times are significantly reduced as the number of threads increases and this makes the method able to be applied to more complex problems if there is enough computing power and several computing units available, which is possible in modern multi-core computers.

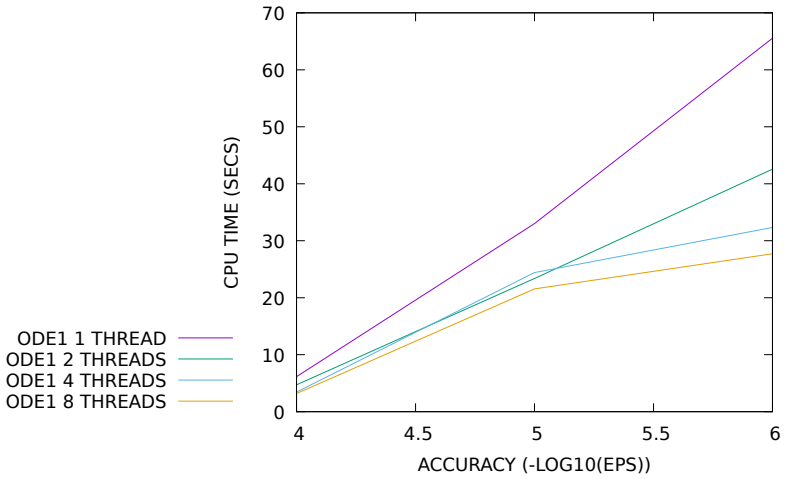

Figure 11. Computational cost and desired accuracy for different number of threads.

4.6. Some Practical Problems

The proposed method was also applied on a series of systems of ODEs available from https://archimede.uniba.it/~testset/ (accessed on 10 May 2022) and more specific the Hires problem, the Rober problem, and the Orego problem.

4.6.1. Hires Problem

This is a system of eight non-linear ODEs proposed by Schäfer in 1975 [51] and it describes how light is involved in morphogenesis. The problem is defined as

$$\frac{dy}{dt} = f(y),\ y(0) = y_0$$

with $t \in [0, 321.8122]$ and the function $f(y)$ defined as

$$f(y) = \begin{pmatrix} -1.71y_1 + 0.43y_2 + 8.32y_3 + 0.0007 \\ 1.71y_1 - 8.75y_2 \\ -10.03y_3 + 0.43y_4 + 0.035y_5 \\ 8.32y_2 + 1.71y_3 - 1.12y_4 \\ -1.745y_5 + 0.43y_6 + 0.43y_7 \\ -280y_6y_8 + 0.69y_4 + 1.71y_5 - 0.43y_6 + 0.69y_7 \\ 280y_6y_8 - 1.81y_7 \\ -280y_6y_8 + 1.81y_7 \end{pmatrix}$$

and the initial conditions $y_0 = (1, 0, 0, 0, 0, 0, 0, 0.0057)$

4.6.2. Rober Problem

The Rober problem [52] describes the kinetics of an autocatalytic reaction and it is defined a system of three non-linear ordinary differential equations. The problem is defined as

$$\frac{dy}{dt} = f(y),\ y(0) = y_0$$

with $t \in [0, T]$ and the function $f(y)$ is

$$f(y) = \begin{pmatrix} -0.04y_1 + 10^4 y_2 y_3 \\ 0.04y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2 \\ 3 \times 10^7 y_2^2 \end{pmatrix}$$

and the initial conditions $y_0 = (1, 0, 0)$ and the value of T was set 10.

4.6.3. Orego Problem

The Orego problem [53] is a system of three non-linear ordinary differential equations and refers to the Oregonator model. The problem is defined as

$$\frac{dy}{dt} = f(y),\ y(0) = y_0$$

with $t \in [0, 360]$ The function $f(y)$ is given by:

$$f(y) = \begin{pmatrix} s\left(y_2 - y_1y_2 + y_1 - qy_1^2\right) \\ \frac{1}{s}\left(-y_2 - y_1y_2 + y_3\right) \\ w(y_1 - y_3) \end{pmatrix}$$

with $y_0 = (1, 2, 3)$ and $s = 77.27,\ w = 0.161,\ q = 8.375 \times 10^{-6}$.

The results from the application of the proposed method with the parameters of Table 1 and $w = 10$ are listed in Table 3. The proposed method achieved low learning error even in the above examples. The column MEM denotes the application of the MEBDF [54] method to these problems. Furthermore, in Figures 12–14, the average execution time of the proposed method for different number of processing threads is plotted for the previous three test cases. In these plots logarithmic scale was used.

Table 3. Experimental results for the real life problems.

Problem	w = 10	MEM
Hires	2.8×10^{-5}	6.1×10^{-1}
Rober	2.1×10^{-6}	1.2×10^{-3}
Orego	2.7×10^{-5}	2.5×10^{-2}

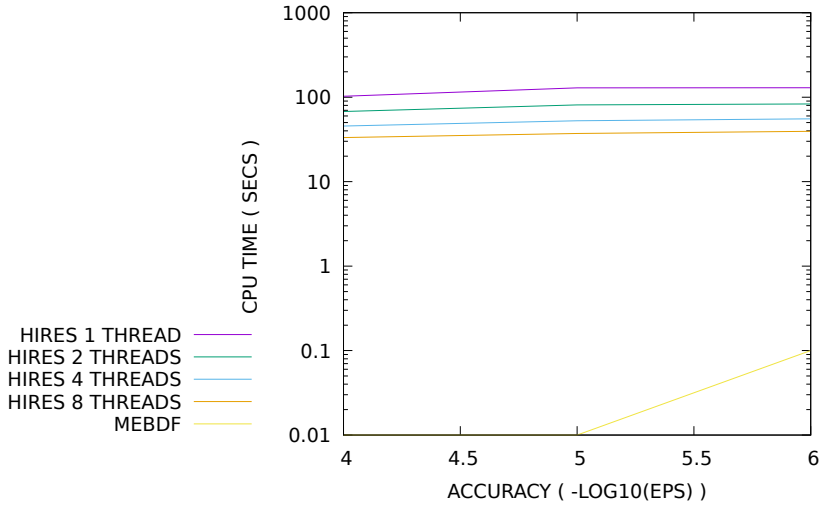

Figure 12. Computational cost and desired accuracy for different number of threads for the case of Hires problem.

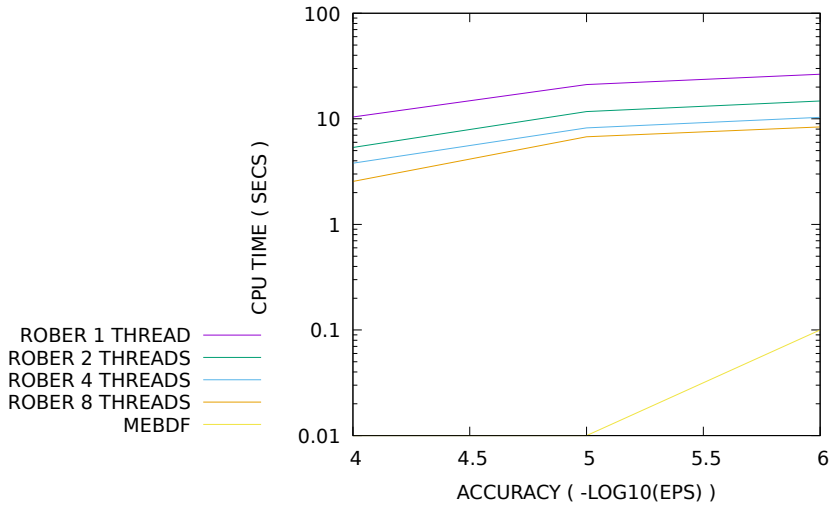

Figure 13. Computational cost and desired accuracy for different number of threads for the case of Rober problem.

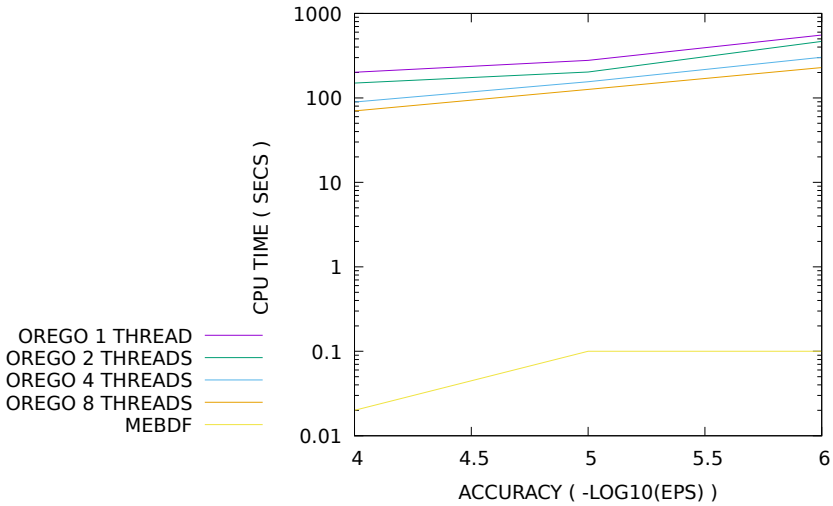

Figure 14. Computational cost and desired accuracy for different number of threads for the case of Orego problem.

5. Conclusions

A method for solving differential equations is presented here, accompanied by the corresponding software. The method utilizes gra networks to solve differential equations and the enforcement of initial and boundary conditions was done using penalty factors. The network configuration was adapted using a hybrid genetic algorithm, in which a local optimization method is applied to a randomly selected set of chromosomes per distinct number of generations.

The software developed in the context of this work was also presented. The software was written in C++ using the open source library QT, so that it can be run on most operating systems. The user can encode the differential equation in either C++ or Fortran by writing a series of functions. Future extensions of the method may include more efficient methods of initializing network weights as well as more advanced methods of terminating the genetic algorithm.

Author Contributions: I.G.T., A.T. and E.K. conceived the idea and methodology and supervised the technical part regarding the software. I.G.T. conducted the experiments, employing several datasets, and provided the comparative experiments. A.T. performed the statistical analysis. E.K. and all other authors prepared the manuscript. E.K. and I.G.T. organized the research team and A.T. supervised the project. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The used software and the experimental functions and data are available from https://github.com/itsoulos/RbfDeSolver (accessed on 10 May 2022).

Acknowledgments: The experiments of this research work was performed at the high performance computing system established at Knowledge and Intelligent Computing Laboratory, Dept of Informatics and Telecommunications, University of Ioannina, acquired with the project "Educational Laboratory equipment of TEI of Epirus" with MIS 5007094 funded by the Operational Programme "Epirus" 2014–2020, by ERDF and national finds.

Conflicts of Interest: The authors declare no conflict of interest.

Sample Availability: Not applicable.

References

1. Raissi, M.; Karniadakis, G.E. Hidden physics models: Machine learning of nonlinear partial differential equations. *J. Comput. Phys.* **2018**, *357*, 125–141. [CrossRef]
2. Lelièvre, T.; Stoltz, G. Partial differential equations and stochastic methods in molecular dynamics. *Acta Numer.* **2016**, *25*, 681–880. [CrossRef]
3. Scholz, G.; Scholz, F. First-order differential equations in chemistr. *ChemTexts* **2015**, *1*, 1. [CrossRef]
4. Padgett, J.L.; Geldiyev, Y.; Gautam, S.; Peng, W.; Mechref, Y.; Ibraguimov, A. Object classification in analytical chemistry via data-driven discovery of partial differential equations. *Comp. Math. Methods* **2021**, *3*, e1164. [CrossRef]
5. Owoyele, O.; Pal, P. ChemNODE: A neural ordinary differential equations framework for efficient chemical kinetic solvers. *Energy* **2022**, *7*, 100118. [CrossRef]
6. Wang, Z.; Huang, X.; Shen, H. Control of an uncertain fractional order economic system via adaptive sliding mode. *Neurocomputing* **2012**, *83*, 83–88. [CrossRef]
7. Achdou, Y.; Buera, F.J.; Lasry, J.M.; Lions, P.L.; Moll, B. Partial differential equation models in macroeconomics. *Phil. Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.* **2012**, *372*, 20130397. [CrossRef]
8. Hattaf, K.; Yousfi, N. Global stability for reaction—Diffusion equations in biology. *Comput. Math. Appl.* **2013**, *66*, 1488–1497.
9. Getto, P.; Waurick, M. A differential equation with state-dependent delay from cell population biology. *J. Differ.* **2016**, *260*, 6176–6200. [CrossRef]
10. Tang, W.; Sun, Y. Construction of Runge—Kutta type methods for solving ordinary differential equations. *Appl. Math. Comput.* **2014**, *234*, 179–191. [CrossRef]
11. Kennedy, C.A.; Carpenter, M.H. Higher-order additive Runge–Kutta schemes for ordinary differential equations. *Appl. Numer. Math.* **2019**, *136*, 183–205. [CrossRef]

12. Yang, X.; Shen, Y. Runge-Kutta Method for Solving Uncertain Differential Equations. *J. Uncertain. Anal. Appl.* **2015**, *3*, 17. [CrossRef]
13. Kim, H.; Sakthivel, R. Numerical solution of hybrid fuzzy differential equations using improved predictor—Corrector method. *Commun. Nonlinear Sci. Numer. Simul.* **2012**, *17*, 3788–3794. [CrossRef]
14. Daftardar-Gejji, V.; Sukale, Y.; Bhalekar, S. A new predictor–corrector method for fractional differential equations. *Appl. Math. Comput.* **2014**, *244*, 158–182. [CrossRef]
15. Lagaris, I.E.; Likas, A.; Fotiadis, D.I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **1998**, *9*, 987–1000. [CrossRef] [PubMed]
16. Mall, S.; Chakraverty, S. Application of Legendre Neural Network for solving ordinary differential equations. *Appl. Soft Comput.* **2016**, *43*, 347–356. [CrossRef]
17. Pakdaman, M.; Ahmadian, A.; Effati, S.; Salahshour, S.; Baleanue, D. Solving differential equations of fractional order using an optimization technique based on training artificial neural network. *Appl. Math. Comput.* **2017**, *293*, 81–95. [CrossRef]
18. Chang, W.D. Parameter identification of Chen and Lü systems: A differential evolution approach. *Chaos Solitons Fractals* **2007**, *32*, 1469–1476. [CrossRef]
19. Biswas, A.; Das, S.; Abraham, A.; Dasgupta, S. Design of fractional-order $PI\lambda D\mu$ controllers with an improved differential evolution. *Eng. Appl. Artif. Intell.* **2009**, *22*, 343–350. [CrossRef]
20. Arqub, O.A.; Hammour, Z.A. Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. *Inf. Sci.* **2014**, *279*, 396–415. [CrossRef]
21. Gutierrez-Navarro, D.; Lopez-Aguayo, S. Solving ordinary differential equations using genetic algorithms and the Taylor series matrix method. *J. Phys. Commun.* **2018**, *2*, 115010. [CrossRef]
22. Januszewski, M.; Kostur, M. Accelerating numerical solution of stochastic differential equations with CUDA. *Comput. Commun.* **2010**, *181*, 183–188. [CrossRef]
23. Murray, L. GPU Acceleration of Runge-Kutta Integrators. *IEEE Trans. Parallel Distrib. Syst.* **2012**, *23*, 94–101. [CrossRef]
24. Riesinger, C.; Neckel, T.; Rupp, F. Solving Random Ordinary Differential Equations on GPU Clusters using Multiple Levels of Parallelism. *Siam J. Sci. Comput.* **2016**, *38*, C372–C402. [CrossRef]
25. O'Neill, M.; Ryan, C. Grammatical evolution. *IEEE Trans. Evol. Comput.* **2001**, *5*, 349–358. [CrossRef]
26. Tsoulos, I.G.; Lagaris, I.E. Solving differential equations with genetic programming. *Genet. Program Evolvable Mach* **2006**, *7*, 33–54. [CrossRef]
27. Le, T.T.V.; Le-Cao, K.; Duc-Tran, H. A Radial Basis Neural Network Approximation with Extended Precision for Solving Partial Differential Equations. In *Soft Computing: Biomedical and Related Applications. Studies in Computational Intelligence*; Phuong, N.H., Kreinovich, V., Eds.; Springer: Berlin/Heidelberg, Germany, 2021; Volume 981.
28. Wei, P.; Li, Z.; Li, X. An 88-line MATLAB code for the parameterized level set method based topology optimization using radial basis functions. *Struct. Multidisc. Optim.* **2018**, *58*, 831–849. [CrossRef]
29. Iqbal, A.; Hamid, N.N.A.; Ismail, A.I.M.; Abbas, M. Galerkin approximation with quintic B-spline as basis and weight functions for solving second order coupled nonlinear Schrödinger equations. *Math. Comput. Simul.* **2021**, *187*, 1–16. [CrossRef]
30. Goldberg, D. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley Publishing Company: Boston, MA, USA, 1989.
31. Michaelewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs*; Springer: Berlin/Heidelberg, Germany, 1996.
32. Grady, S.A.; Hussaini, M.Y.; Abdullah, M.M. Placement of wind turbines using genetic algorithms. *Renew. Energy* **2005**, *30*, 259–270. [CrossRef]
33. Park, J.; Sandberg, I.W. Universal Approximation Using Radial-Basis-Function Networks. *Neural Comput.* **1991**, *3*, 246–257. [CrossRef]
34. MacQueen, J. Some methods for classification and analysis of multivariate observations. In Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Los Angeles, CA, USA, 1 January 1967; Volume 1, pp. 281–297.
35. Teng, P. Machine-learning quantum mechanics: Solving quantum mechanics problems using radial basis function networks. *Phys. Rev. E* **2018**, *98*, 033305. [CrossRef]
36. Jovanović, R.; Sretenovic, A. Ensemble of radial basis neural networks with K-means clustering for heating energy consumption prediction. *Fme Trans.* **2017**, *45*, 51–57. [CrossRef]
37. Alexandridis, A.; Chondrodima, E.; Efthimiou, E.; Papadakis, G.; Vallianatos, F.; Triantis, D. Large Earthquake Occurrence Estimation Based on Radial Basis Function Neural Networks. *IEEE Trans. Geosci. Remote. Sens.* **2014**, *52*, 5443–5453. [CrossRef]
38. Gorbachenko, V.I.; Zhukov, M.V. Solving boundary value problems of mathematical physics using radial basis function networks. *Comput. Math. Math. Phys.* **2017**, *57*, 145–155. [CrossRef]
39. Wan, C.; Harrington, P. Self-Configuring Radial Basis Function Neural Networks for Chemical Pattern Recognition. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 1049–1056. [CrossRef]
40. Yao, X.; Zhang, X.; Zhang, R.; Liu, M.; Hu, Z.; Fan, B. Prediction of enthalpy of alkanes by the use of radial basis function neural networks. *Comput. Chem.* **2001**, *25*, 475–482. [CrossRef]
41. Shahsavand, A.; Ahmadpour, A. Application of optimal RBF neural networks for optimization and characterization of porous materials. *Comput. Chem. Eng.* **2005**, *29*, 2134–2143. [CrossRef]

42. Wang, Y.P.; Dang, J.W.; Li, Q.; Li, S. Multimodal medical image fusion using fuzzy radial basis function neural networks. In Proceedings of the 2007 International Conference on Wavelet Analysis and Pattern Recognition, Beijing, China, 2–4 November 2007; pp. 778–782.
43. Mehrabi, S.; Maghsoudloo, M.; Arabalibeik, H.; Noormand, R.; Nozari, Y. Congestive heart failure, Chronic obstructive pulmonary disease, Clinical decision support system, Multilayer perceptron neural network and radial basis function neural network. *Expert Syst. Appl.* **2009**, *36*, 6956–6959. [CrossRef]
44. Veezhinathan, M.; Ramakrishnan, S. Detection of Obstructive Respiratory Abnormality Using Flow—Volume Spirometry and Radial Basis Function Neural Networks. *J. Med. Syst.* **2007**, *31*, 461. [CrossRef]
45. Momoh, J.A.; Reddy, S.S. Combined Economic and Emission Dispatch using Radial Basis Function. In Proceedings of the 2014 IEEE PES General Meeting | Conference & Exposition, National Harbor, MD, USA, 27–31 July 2014; pp. 1–5. [CrossRef]
46. Guo, J.-J.; Luh, P.B. Selecting input factors for clusters of Gaussian radial basis function networks to improve market clearing price prediction. *IEEE Trans. Power Syst.* **2003**, *18*, 665–672.
47. Falat, L.; Stanikova, Z.; Durisova, M.; Holkova, B.; Potkanova, T. Application of Neural Network Models in Modelling Economic Time Series with Non-constant Volatility. *Procedia Econ. Financ.* **2015**, *34*, 600–607. [CrossRef]
48. Dagum, L.; Menon, R. OpenMP: An industry standard API for shared-memory programming. *IEEE Comput. Sci. Eng.* **1998**, *5*, 46–55. [CrossRef]
49. Powell, M.J.D. A Tolerant Algorithm for Linearly Constrained Optimization Calculations. *Math. Program.* **1989**, *45*, 547. [CrossRef]
50. Kaelo, P.; Ali, M.M. Integrated crossover rules in real coded genetic algorithms. *Eur. J. Oper.* **2007**, *176*, 60–76. [CrossRef]
51. Schäfer, E. A new approach to explain the "high irradiance responses" of photomorphogenesis on the basis of phytochrome. *J. Math. Biol.* **1975**, *2*, 41–56. [CrossRef]
52. Robertson, H.H. The Solution of a Set of Reaction Rate Equations. In *Numerical Analysis: An introduction*; Walsh, J., Ed.; Academic Press: Cambridge, MA, USA, 1967; pp. 178–182.
53. Hairer, E.; Wanner, G. *Solving Ordinary Di Erential Equations II: Sti and Di Erential-Algebraic PROBLEMS*, 2nd revised ed.; Springer: Berlin/Heidelberg, Germany, 1996.
54. Cash, R.; Considine, S. An MEBDF code for stiff initial value problems. *Acm Trans. Math. Softw.* **1992**, *18*, 142–158. [CrossRef]