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. For example, differential equations can be applied to physics problems, chemistry problems, economics, modeling etc. This manuscript presents a number of global optimization techniques that have been successfully applied to train machine learning models to approximate differential equation solutions. More specifically, two modified versions of genetic algorithms and particle swarm optimization methods are proposed here. 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 of ordinary differential equations (ODEs), systems of differential equations (SODEs) and partial differential equations (PDEs) is commonly used in many scientific fields, like physics [?, ?], chemistry [?, ?, ?], economics [?, ?], biology [?, ?] etc. It is obvious that the numerical solution of differential equations has positive effects in many scientific areas and, as a result, many techniques have been published in the modern literature. These methods can include the well - known Runge Kutta methods [?, ?, ?], wavelet transformations [?, ?, ?], predictor - corrector variations [?, ?], methods that incorporate artificial neural networks [?, ?] to solve differential equations [?, ?, ?], finite element methods [?, ?] etc. Similar, semi - implicit composition methods have been

*Corresponding author. Email: itsoulos@uoi.gr

proposed to solve differential equations like the work of Chen and Shen, used to solve the Ginzburg—Landau equation and the Cahn—Hilliard equation [?]. Likewise, Fedoseev et al proposed a novel algorithm for Semi-Implicit Composition ODE Solvers [?]. Also, semi-explicit composition methods have been proposed for the solution of differential equations [?]. Similar, semi-implicit predictor–corrector methods have been used for practical problems modeled as differential equations [?, ?]. Recently, another technique based on Grammatical Evolution [?] has been proposed to tackle differential equations in closed analytical form by Tsoulos and Lagaris [?].

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 continuous and differentiable function $f : S \to R, S \subset R^n$ and it is defined as:

$$x^* = \arg\min_{x \in S} f(x) \tag{1}$$

with S:

$$S = [L_1, R_1] \otimes [L_2, R_2] \otimes \ldots [L_n, R_n]$$

The value of n is the number 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 left margin of the parameter x, and the vector R as the right margin. Machine learning models have been used a lot in solving differential equations. For example, the use of SVM techniques [?, ?], the incorporation of deep learning methods [?, ?], constructed neural networks [?] etc. In current work, two machine learning models were tested: artificial neural networks and Radial Basis Function networks (RBF) [?]. The parameters in these machine learning models were trained using modified versions of two well-known global optimization techniques: Genetic Algorithms [?, ?, ?] and the PSO method [?, ?]. In the present work, these two global optimization techniques were chosen as they have been used in dozens of practical problems and are easily adaptable to the problem. Furthermore, both methods can be easily parallelized [?, ?, ?] to take advantage of modern parallel architectures and are tolerant of numerical errors and noise that may be present in the data. The proposed method has been successfully used to solve ordinary differential equations, systems of differential equations as well as partial differential equations with Dirichlet boundary conditions.

In the proposed methodology, the problem of numerically solving differential equations has been reduced to a machine learning model parameterization problem using global optimization techniques. Consequently, any global optimization method can be used to solve differential equations, even techniques with parallelization capabilities, such as Genetic Algorithms for example. This means that the proposed technique could also be used to solve difficult problems of differential equations, such as for example large systems of ordinary differential equations.

Recently , Effati et al [?] used artificial neural networks for solving differential equations. Also, Pakdaman et al [?] used artificial neural networks to solve

differential equations of fractional order. Furthermore, Admon et al proposed an algorithm based on artificial neural networks for solving differential equations of fractional order.

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

2 Method description

The proposed technique minimizes the error of machine learning models, which are used as solvers of differential equations. Therefore, the section starts with the definition of the error for each differential equation case and continues with the definition of the machine learning models used and ends with the global optimization techniques used to minimize the error of the differential equations.

2.1 Calculation of error for the machine learning models

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

2.1.1 Calculation for ODEs

An ordinary differential equation can be defined as:

$$\psi\left(t, y, y^{(1)}, \ldots, y^{(n)}\right) = 0, \ t \in [a, b] \tag{2}$$

where the quantity $y^{(i)}$ stands for the ith-order derivative of $y(x)$. Also, the initial conditions for the equation are given by:

$$h_i\left(t_i, y, y^{(1)}, \ldots, y^{(n)}\right), i = 1, \ldots, n \tag{3}$$

where t_i can be a or b. The calculation of the model error $f(r)$ of a given machine learning model r are the following:

1. **Construct** $T = \{t_1 = a, t_2, t_3, \ldots, t_N = b\}$. This is a set of equally spaced points.
2. **Compute** the quantity for the error $E_r = \sum_{i=1}^{N} \psi\left(t_i, r\left(t_i\right), r^{(1)}\left(t_i\right), \ldots, r^{(n)}\left(t_i\right)\right)^2$
3. **Compute** the associated penalty value, based on the initial conditions:

$$P_r = \lambda \sum_{k=1}^{n} h_k^2\left(t, r(t), r^{(1)}(t), \ldots, r^{(n)}(t)\right)_{|t=t_k} \tag{4}$$

   , where $\lambda > 0$.
4. **Return** $f(r) = E_r + P_r$

2.1.2 Calculation for SODEs

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

$$\begin{pmatrix} \psi_1\left(t, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \\ \psi_2\left(t, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \\ \vdots & \vdots & \vdots \\ \psi_k\left(t, y_1, y_1^{(1)}, y_2, y_2^{(1)}, \ldots, y_k, y_k^{(1)}\right) & = & 0 \end{pmatrix} \tag{5}$$

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

$$\begin{pmatrix} y_1(a) & = & y_{1a} \\ y_2(a) & = & y_{2a} \\ \vdots & \vdots & \vdots \\ y_k(a) & = & y_{ka} \end{pmatrix} \tag{6}$$

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

1. **Construct** a set $T = \{t_1 = a, t_2, t_3, \ldots, t_N = b\}$ of equally spaced points.
2. **For every model** $r_i,\ i = 1, ..k$ **do**
   (a) **Calculate** the error value: $E_{r_i} = \sum_{j=1}^{N} \left(\psi_i\left(t_j, r_1, r_1^{(1)}, r_2, r_2^{(1)}, \ldots, r_k, r_k^{(1)}\right)\right)^2$
   (b) **Calculate** the corresponding penalty value: $P_{r_i} = \lambda \left(r_i\left(a\right) - y_{ia}\right)^2$
3. **EndFor**
4. **Compute** the total error value: $f(r) = \sum_{i=1}^{k} \left(E_{r_i} + P_{r_i}\right)$

2.1.3 Calculation for PDEs

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

$$h\left(t, y, \Psi(t, y), \frac{\partial}{\partial t}\Psi(t, y), \frac{\partial}{\partial y}\Psi(t, y), \frac{\partial^2}{\partial t^2}\Psi(t, y), \frac{\partial^2}{\partial y^2}\Psi(t, y)\right) = 0 \tag{7}$$

with $t \in [a, b],\ y \in [c, d]$. The boundary conditions consider in this manuscript are the Dirichlet boundary conditions provided by:

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

The error value of any chromosome g is calculated as follows:

1. **Define** the set $T = \{(t_1, y_1), (t_1, y_2), \ldots, (t_N, y_N)\}$. This is a grid of $N \times N$ points in $[a, b] \times [c, d]$.
2. **Define** the set $t_B = \{t_{b1}, t_{b2}, \ldots, t_{bN}\}$, equally spaced points in $[a, b]$.
3. **Define** the set $y_B = \{y_{b1}, y_{b2}, \ldots, y_{bN}\}$, equally spaced points in $[c, d]$.
4. **Calculate** the error value E_r as
$$E_r = \sum_{i=1}^{N} h\left(t_i, y_i, r\left(t_i, y_i\right), \frac{\partial}{\partial x} r\left(t_i, y_i\right), \frac{\partial}{\partial y} r\left(t_i, y_i\right)\right)^2$$
5. **Calculate** the associated penalty values:
$$\begin{aligned} P_{1r} &= \lambda \sum_{i=1}^{N} \left(r\left(a, y_{bi}\right) - f_0\left(y_{bi}\right)\right)^2 \\ P_{2r} &= \lambda \sum_{i=1}^{N} \left(r\left(b, y_{bi}\right) - f_1\left(y_{bi}\right)\right)^2 \\ P_{3r} &= \lambda \sum_{i=1}^{N} \left(r\left(t_{bi}, c\right) - g_0\left(t_{bi}\right)\right)^2 \\ P_{4r} &= \lambda \sum_{i=1}^{N} \left(r\left(t_{bi}, d\right) - g_1\left(t_{bi}\right)\right)^2 \end{aligned}$$
6. **Compute** the total error: $f(r) = E_r + P_{1r} + P_{2r} + P_{3r} + P_{4r}$

2.2 The neural network model

One of the machine learning models used for the approximate solution of differential equations is artificial neural networks. Artificial neural networks are parametric mathematical models used in many scientific areas with great importance, such as physics [**?**, **?**, **?**], chemistry [**?**, **?**, **?**], medicine [**?**, **?**] etc. A neural network can be modeled as a function $N(\overrightarrow{x}, \overrightarrow{w})$, where the vector $\overrightarrow{x}$ is considered the input pattern and $\overrightarrow{w}$ is considered as the weight vector. The methods used to train neural network, usually estimate the weight vector $\overrightarrow{w}$ for a given dataset. The optimization technique minimizes the following quantity:

$$E\left(N\left(\overrightarrow{x}, \overrightarrow{w}\right)\right) = \sum_{i=1}^{M} \left(N\left(\overrightarrow{x}_i, \overrightarrow{w}\right) - y_i\right)^2 \tag{8}$$

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

$$o_i(x) = \sigma\left(p_i^T x + \theta_i\right), \tag{9}$$

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

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{10}$$

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

$$N(x) = \sum_{i=1}^{H} v_i o_i(x), \tag{11}$$

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:

$$N\left(\overrightarrow{x}, \overrightarrow{w}\right) = \sum_{i=1}^{H} w_{(d+2)i-(d+1)} \sigma \left(\sum_{j=1}^{d} x_j w_{(d+2)i-(d+1)+j} + w_{(d+2)i} \right) \tag{12}$$

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

2.3 The RBF model

The second machine learning model used to estimate a solution for differential equations is the Radial Basis Function (RBF) network. An RBF network also has a lot of applications [?, ?, ?] and it is denoted as a function $r(x)$:

$$r(x) = \sum_{i=1}^{H} w_i \phi\left(\|x - c_i\|\right) \tag{13}$$

with $\overrightarrow{x}$ denoting the input pattern and the vector $\overrightarrow{w}$ stands for the weight vector. The function $\phi(x)$ used here is the following Gaussian function:

$$\phi(x) = \exp\left(-\frac{(x-c)^2}{\sigma^2}\right) \tag{14}$$

The function $\phi(x)$ has the property that its value is calculated based on the distance between the vectors $\overrightarrow{x}$, $\overrightarrow{c}$. The vector $\overrightarrow{c}$ is also called the centroid vector.

2.4 The used genetic algorithm

The Genetic Algorithm is a well - known global optimization technique based on biological observations and it incorporates a series of operations such as selection, crossover and mutation. The Genetic algorithm initiates by creating a population of candidate solutions, called also chromosomes, and subsequently these solutions are evolved through the application of genetic operations. The method has been used on a variety of research problems such as electromagnetic [?], combinatorial problems [?], design of water distribution networks [?] 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 used genetic algorithm is composed by the following steps:

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** the chromosomes $g_i,\ i = 1, ..., N_c$ using random numbers.
   (h) **Set** iter=0
2. **Check** for termination.
   (a) **Obtain** the best fitness
   $$f^* = \min_{i \in [0...N_c]} f_i$$
   (b) **Terminate** if iter $\geq N_g$ OR $f^* \leq \epsilon$
3. **Calculate** fitness.
   (a) **For** $i = 1, \ldots, N_c$ **do**
      i. **Construct** a neural network or an RBF network using as parameters the chromosome g_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 **??**.
      iii. **Draw** a random number $r \in [0, 1]$. **If** $r \leq P_l$, apply a local search procedure to the model m_i and locate a better value f_i for fitness. The used local search procedure is a BFGS method [**?**].
   (b) **EndFor**
4. **Application** of genetic operators.
   (a) **Selection** operation. The procedure starts by sorting the chromosomes according to their fitness. The first $(1 - P_s) \times N_c$ are copied intact to the next generation of the population. The remaining chromosomes will be substituted by chromosomes that will be computed by crossover and mutation.

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

$$\begin{aligned} \tilde{z}_i &= a_i z_i + (1 - a_i) w_i \\ \tilde{w}_i &= a_i w_i + (1 - a_i) z_i \end{aligned} \tag{15}$$

with a_i a random number and $a_i \in [-0.5, 1.5]$ [**?**].

(c) **Mutation** operation. Alter every element of each chromosome randomly with probability P_m.

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

5. **Goto** step **??**.

2.5 The used PSO method

The method PSO is a stochastic global optimization method and it constructs a swarm of candidate solutions (particles) that are moving towards 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 used in many applications [**?**, **?**, **?**, **?**]. This manuscript proposes a hybrid PSO method with a periodical application of a local search procedure. 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) **Initialize** the positions of the particles $p_1, p_2, ..., p_{N_c}$ randomly. The size of each particle is defined as n.

   (g) **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 $b_i = p_i$, where b_i is the best located position for particle i.

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

2. **Check** for termination.

   (a) **Obtain** the best fitness

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

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

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

   (a) **Calculate** the velocity u_i

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

   (b) **Set** $p_i = p_i + u_i$

   (c) **Evaluate** the fitness f_i of the particle p_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 p_i.

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

4. **End For**

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

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

7. **Goto** Step **??**

3 Experiments

The differential equations used in the experiments are presented below. These equations have been used in experiments and other related publications[**?**, **?**].

3.1 Linear ode equations

ODE1

$$y' = \frac{2t - y}{t}$$

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

ODE2

$$y' = \frac{1 - y\cos(t)}{\sin(t)}$$

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

ODE3

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

with $t \in [0, 1]$. The initial conditions are $y(0) = 0,\ y'(0) = 2$. The solution is $y(t) = 2t\exp(3t)$

ODE4

$$y'' = -\frac{1}{5}y' - y - \frac{1}{5}\exp\left(-\frac{t}{5}\right)\cos(t)$$

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

ODE5

$$y'' = -100y$$

with $t \in [0, 1]$. The initial conditions are $y(0) = 0,\ y'(0) = 10$. The solution is given by

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

3.2 Non-linear ODEs

NLODE1

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

with $t \in [1, 4]$ and the initial condition $y(1) = 1$. The solution is given by $y(t) = \sqrt{t}$

NLODE2

$$(y')^2 + \log(y) - \cos^2(t) - 2\cos(t) - 1 - \log(t + \sin(t)) = 0$$

with $t \in [1, 2]$ and the initial condition $y(1) = 1 + \sin(1)$. The solution is given by $y(t) = t + \sin(t)$

NLODE3

$$y''y' = -\frac{4}{t^3}$$

with $t \in [1, 2]$ and the initial conditions are $y(1) = 0,\ y(2) = \log(4)$. The solution is $y(t) = \log\left(t^2\right)$

NLODE4

$$t^2y'' + (ty')^2 + \frac{1}{\log(t)} = 0$$

with $t \in [e, 2e]$ and the initial conditions are $y(e) = 0,\ y'(e) = \frac{1}{e}$. The solution is $y(t) = \log(\log(t))$

3.3 Odes without analytic solution

UNSOLODE1

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

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

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

UNSOLODE2

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

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

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

3.4 Systems of ode cases

SODE1

$$\begin{aligned} y_1' &= \cos(t) + y_1^2 + y_2 - \left(t^2 + \sin^2(t)\right) \\ y_2' &= 2t - t^2\sin(t) + y_1y_2 \end{aligned}$$

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

SODE2

$$\begin{aligned} y_1' &= \frac{\cos(t) - \sin(t)}{y_2} \\ y_2' &= y_1 y_2 + \exp(t) - \sin(t) \end{aligned}$$

with $t \in [0,1]$ and the initial conditions are $y_1(0) = 0,\ y_2(0) = 1$. The exact solutions are: $y_1(x) = \frac{\sin(t)}{\exp(t)},\ y_2 = \exp(t)$

SODE3

$$\begin{aligned} y_1' &= \cos(t) \\ y_2' &= -y_1 \\ y_3' &= y_2 \\ y_4' &= -y_3 \\ y_5' &= y_4 \end{aligned}$$

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

SODE4

$$\begin{aligned} y_1' &= -\frac{1}{y_2} \sin\left(\exp(t)\right) \\ y_2' &= -y_2 \end{aligned}$$

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

3.5 Pde cases

PDE1

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

with $t \in [0,1],\ y \in [0,1]$. The boundary conditions are given by: $\Psi(0,y) = y^3,\ \Psi(1,y) = \left(1 + y^3\right)\exp(-1),\ \Psi(x,0) = t\exp(-t),\ \Psi(t,1) = (t+1)\exp(-t)$ The solution is given by: $\Psi(t,y) = \left(t + y^3\right)\exp(-t)$

PDE2

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

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

PDE3

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

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

PDE4

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

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

3.6 Experimental results

A set of experiments were conducted to determine the efficiency and the validity of the proposed methods. The equations as well as the proposed methods were coded in ANSI C++ using the OPTIMUS optimization library, freely available from `https://github.com/itsoulos/OPTIMUS/`. The parameters of the methods are shown in the table **??**. The weights for the neural network as well as for the RBF case were set to 10 and all the experiments were conducted 30 times and averages were reported. The experiments were conducted using the drand48() function of the C language as the random number generator. In order to check the reliability of the generated solutions, they were applied to randomly selected points in the definition fields of the differential equations, which were twice as many as the points used to train the models. The model's error at these points was called the generalization error. The results are listed in the table **??**. The meaning of the columns are:

1. The column EQUATION denotes the title of the equation to be solved. The names of the equations were presented in the previous subsection.
2. The column MLP-GEN shows the generalization error of a neural network, trained using the proposed genetic algorithm. The number of weights for the neural network was set to 10.
3. The column RBF-GEN shows the generalization errors of an RBF model. The weights as well as the centroids of the RBF network were trained using the proposed genetic algorithm. The number of weights for the RBF model was set to 10.
4. The column MLP-PSO stands for the application of a neural network with 10 hidden nodes that was adapted to the given equation using the proposed PSO algorithm.

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
H	10

5. The column MLP-RBF denotes the application of an RBF to the equation and the network was trained using the proposed PSO variant.

In this table, in all cases the proposed methodology had very low generalization errors even in the case of partial differential equations which is considered a more difficult problem than those presented previously. In fact, in several cases, the generalization error is extremely low, and in particular, RBF networks seem to show a lower generalization error in a range of functions. Also, it should be noted that there is no clear distinction between genetic algorithms and particle swarm optimization in terms of average generalization error.

Of course, the proposed method may be slower than other Numerical Analysis techniques, especially in simple differential equation problems. However, this issue can be partially addressed by using parallel Genetic Algorithms, for example. In figure **??** the average execution time of the proposed method for the test cases of ODEs and SODEs is plotted. In this series of experiments, an increasing number of threads is used at every experiment. The number of threads varies from 1 to 8. The programming library used for the threads execution was the OpenMP library [**?**]. The execution was performed on an Intel i7-10700T running at 2.00GHz with 16GB of ram and the operating system was Debian Linux. The required training time is significantly reduced as the number of processing threads increases. The use of parallel techniques will certainly be of great benefit in cases with significant computational cost such as partial differential equations. Of course, the reduction in time will be even greater if more processing threads are available, but the reduction is not directly proportional to the number of threads, since communication between threads takes time.

Furthermore, the quality of the generated solutions is shown in the two additional figures, **??** and **??**. The first one shows the absolute error for the nlode4 function and the second the absolute errors for the second system of differential equations (sode2). In both cases, an artificial neural network trained with the proposed genetic algorithm was used. In both cases the error is extremely low. The average generalization error was extremely low and this is reflected in the error per point.

Table 2: Experimental results using the hybrid genetic algorithm. The numbers in the cells represent the average generalization error for each run.

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}

Figure 1: Average execution times for the proposed technique using the Genetic algorithm and an increasing number of OpenMP threads.

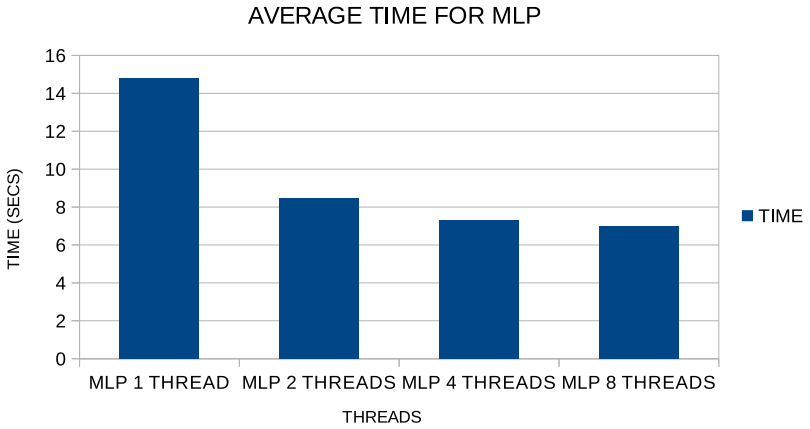


Figure 2: Absolute error for the NLODE4 case.

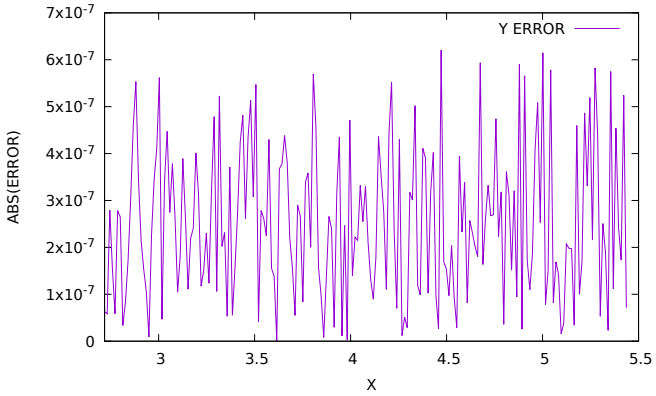
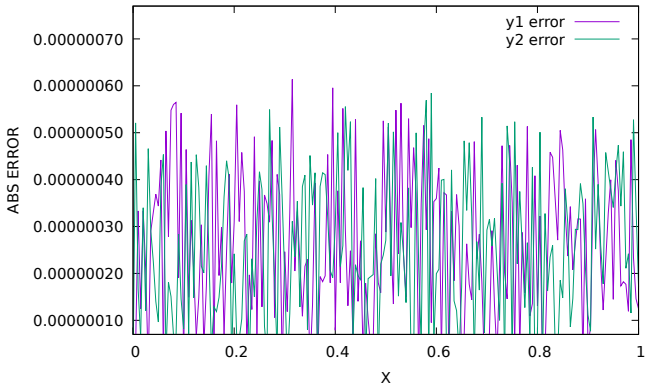


Figure 3: Absolute error for the SODE2 test case.



3.7 Comparison with other methods

In order to be able to understand the dynamics of the proposed methodology, it was applied to a series of practical problems which are available from the relevant website `https://archimede.uniba.it/~testset/`. The obtained problems are the Hires problem, the Rober problem and the Orego problem.

Hires problem

The Hires problem was suggested by Schäfer in 1975 [?] and it describes how light is involved in morphogenesis. The problem is defined as

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

where $t \in [0, 321.8122]$ and the function $f(y)$ is the following array of equations:

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

The initial conditions of the system are: $y_0 = (1, 0, 0, 0, 0, 0, 0, 0.0057)$

Rober problem

The Rober problem [?] describes the kinetics of an autocatalytic reaction and the system of equations is:

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

with $t \in [0, 10]$. The initial conditions are $y_0 = (1, 0, 0)$.

Orego problem

The Orego problem [?] describes 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 :

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

Table 3: Comparison with MEMDB method on some real world problems.

PROBLEM	MEBDF	MLP GEN	MLP PSO
HIRES	6.1×10^{-1}	7.6×10^{-6}	8.7×10^{-6}
OREGO	1.2×10^{-3}	5.0×10^{-11}	1.5×10^{-10}
ROBER	2.5×10^{-2}	2.66×10^{-7}	2.9×10^{-7}

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

The proposed method with the artificial neural network as model was applied on the above practical problems. The experimental results are compared against MEBDF [?] and are listed in the Table ??.

4 Conclusions

In this text, the numerical solution of differential equations using machine learning models was presented. These models were artificial neural networks and RBF neural networks. The adjustment of the parameters of these models to the initial conditions of the equations was done using penalty factors. Finding the optimal parameters for these models was done using two modified versions of well - known global optimization techniques. One global optimization method used was the genetic algorithm and the other modification was the particle swarm optimization. In all experimental cases, the generalization error was extremely low and, indeed, in some cases, the RBF network was shown to outperform the artificial neural network.

Future extensions of the proposed methodology may include:

1. Incorporation of additional machine learning models, such as SVM models.
2. Usage of more efficient stopping rules.
3. Parallelization of the whole process using modern programming techniques such as MPI [?] or OpenMP [?].

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.

[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] L.Q.Chen, J. Shen, Applications of semi-implicit Fourier-spectral method to phase field equations, Computer Physics Communications 108, pp. 147-158, 1998.

[26] P. Fedoseev, D. Pesterev, A. Karimov, D. Butusov, New Step Size Control Algorithm for Semi-Implicit Composition ODE Solvers, Algorithms **15**, article number 275, 2022.

[27] A. Tutueva, T. Karimov, D. Butusov, Semi-Implicit and Semi-Explicit Adams-Bashforth-Moulton Methods, Mathematics 8, Article number 780, 2020.

[28] C. Clancy, J.A. Pudykiewicz, A class of semi-implicit predictor–corrector schemes for the time integration of atmospheric models, Journal of Computational Physics **250**, pp. 665-684, 2013.

[29] A. Tutueva, D. Butusov, Stability Analysis and Optimization of Semi-Explicit Predictor–Corrector Methods, Mathematics **9**, Article number 2463, 2021.

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

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

[32] 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.

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

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

[35] 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.

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

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

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

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

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

[41] 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.

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

[43] E.Cantú-Paz, D.E.Goldberg, Efficient parallel genetic algorithms: theory and practice, Computer Methods in Applied Mechanics and Engineering **186**, pp. 221-238, 2000.

[44] K. Wang,Z. Shen, A GPU-Based Parallel Genetic Algorithm for Generating Daily Activity Plans, IEEE Transactions on Intelligent Transportation Systems **13**, pp. 1474-1480, Sept. 2012.

[45] D. Kečo, A. Subasi, J. Kevric, Cloud computing-based parallel genetic algorithm for gene selection in cancer classification, Neural Comput & Applic **30**, pp. 1601–1610, 2018.

[46] S. Effati, M. Pakdaman, Artificial neural network approach for solving fuzzy differential equations, Information Sciences **180,** pp. 1434-1457, 2010.

[47] M. Pakdaman, A. Ahmadian, S. Effati, S. Salahshour, D. Baleanu, Solving differential equations of fractional order using an optimization technique based on training artificial neural network, Applied Mathematics and Computation **293,** pp. 81-95, 2017.

[48] M. R. Admon, N. Senu, A. Ahmadian, Z. A. Majid, S. Salahshour, A new efficient algorithm based on feedforward neural network for solving differential equations of fractional order, Communications in Nonlinear Science and Numerical Simulation **117**, 106968, 2023.

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

[50] 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

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

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

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

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

[59] 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.

[60] 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.

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

[62] 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.

[63] 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.

[64] 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.

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

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

[67] 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.

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

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

[70] E. Schäfer, A new approach to explain the "high irradiance responses" of photomorphogenesis on the basis of phytochrome. J. Math. Biology **2**, pp. 41–56, 1975.

[71] H.H. Robertson, H.H. The Solution of a Set of Reaction Rate Equations. In: Walsh, J., Ed., Numerical Analysis: An introduction, Academic Press, Cambridge, Massachusetts, pp. 178-182, 1967.

[72] E. Hairer, G. Wanner, Solving Ordinary Di erential Equations II: Sti and Di erential- algebraic Problems. Springer-Verlag, second revised edition, 1996.

[73] .R. Cash, S. Considine, An MEBDF code for stiff initial value problems, ACM Trans. Math. Software **18**, pp. 142-158, 1992.

[74] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca and A. Lumsdaine, "Open MPI: A High-Performance, Heterogeneous MPI," 2006 IEEE International Conference on Cluster Computing, 2006, pp. 1-9.

[75] L. Dagum, R. Menon, OpenMP: an industry standard API for shared-memory programming, IEEE Computational Science and Engineering **5**, pp. 46-55, 1998.