Iranian Journal of Numerical Analysis and Optimization
Vol. 14, No. 3, 2024, pp 681–707
https://doi.org/10.22067/ijnao.2024.86692.1384
https://ijnao.um.ac.ir/

CC BY NC

Research Article

An improved imperialist competitive algorithm for solving an inverse form of the Huxley equation

H. Dana Mazraeh, K. Parand*, H. Farahani and S.R. Kheradpisheh

*Corresponding author

Received 04 February 2024; revised 28 April 2024; accepted 01 May 2024

Hassan Dana Mazraeh
Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, G.C. Tehran, Iran. e-mail: h_danamazraeh@sbu.ac.ir

Kourosh Parand
Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, G.C. Tehran, Iran.
Department of Cognitive Modeling, Institute for Cognitive and Brain Sciences, Shahid Beheshti University, G.C. Tehran, Iran. e-mail: k_parand@sbu.ac.ir

Hadi Farahani
Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, G.C. Tehran, Iran. e-mail: h_farahani@sbu.ac.ir

Saeed Reza Kheradpisheh
Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, G.C. Tehran, Iran. e-mail: s_kheradpisheh@sbu.ac.ir

How to cite this article
Dana Mazraeh, H., Parand, K., Farahani, H. and Kheradpisheh, S.R., An improved imperialist competitive algorithm for solving an inverse form of the Huxley equation. *Iran. J. Numer. Anal. Optim.*, 2024; 14(3): 681-707. https://doi.org/10.22067/ijnao.2024.86692.1384

Abstract

In this paper, we present an improved imperialist competitive algorithm for solving an inverse form of the Huxley equation, which is a nonlinear partial differential equation. To show the effectiveness of our proposed algorithm, we conduct a comparative analysis with the original imperialist competitive algorithm and a genetic algorithm. The improvement suggested in this study makes the original imperialist competitive algorithm a more powerful method for function approximation. The numerical results show that the improved imperialist competitive algorithm is an efficient algorithm for determining the unknown boundary conditions of the Huxley equation and solving the inverse form of nonlinear partial differential equations.

AMS subject classifications (2020): Primary 68W50; Secondary 35A25, 35R30.

Keywords: Huxley equation; Imperialist competitive algorithm; Partial differential equations; Meta-heuristic algorithms; Genetic algorithm.

1 Introduction

The Huxley equation, classified as a nonlinear partial differential equation (NPDE), has the capacity to model a diverse range of phenomena, including biological population dynamics [9] and the propagation of nerves [30]. Its significance lies in its ability to capture the intricate dynamics and interrelationships within these systems, providing valuable insights into their behavior and characteristics. The choice of the Huxley equation as our focus has a dual rationale. First, as previously mentioned, this equation finds numerous practical applications. Second, the selection of this equation, being an NPDE, serves to demonstrate the capability of our proposed algorithm in handling a wide range of inverse forms of NPDEs. Within the realm of partial differential equations (PDEs), an equation is considered "inverse" when one or more of the initial or boundary conditions are missed. In solving inverse forms of PDEs, we utilize data collected from sensors, often referred to as "over-specified conditions," to compensate for the missing condition(s). The primary challenge in solving inverse forms of PDEs lies in the identification

of the missing condition(s). In this paper, we specifically address a scenario in which one of the boundary conditions, denoted as $q(t)$, is missing.

The main purpose of this paper is to present an improved imperialistic competitive algorithm (IICA) for determining the missing boundary condition, $q(t)$. The reason why the imperialistic competitive algorithm (ICA) was chosen is that this algorithm has demonstrated a remarkable capability for solving equations [2, 19, 12]. In this paper, improvements are made to the original ICA to enhance its suitability for estimating a function. Since the ICA is a meta-heuristic algorithm, the results of the IICA and the original ICA are compared to a genetic algorithm (GA), which is one of the well-known and leading algorithms in the realm of meta-heuristic algorithms. In recent years, meta-heuristic algorithms have shown a significant capability in solving inverse forms of linear and nonlinear PDEs and other challenging problems. Also, the convergence of these algorithms has been studied well [6, 3, 21, 20, 29, 17, 22, 28, 10, 24]. Therefore, investigating the capabilities of the new methods and improved algorithms might yield valuable advancements in this field.

The rest of this paper is organized as follows. To calculate the fitness function (cost function) of the algorithms, we need to solve the direct form of the Huxley equation. In section 2, we present the main form of the Huxley equation and the discretization of the Huxley equation using the Crank–Nicolson method [26], which is a finite difference method. This discretization is employed to solve the direct form of the Huxley equation and evaluate the fitness value of a candidate solution accordingly. In section 3, we present the improved ICA in detail, explaining how our improvement makes the original ICA a more powerful method for estimating a function. Since the GA has been widely used and is famous, section 4 provides a brief description of a real-valued GA. In section 5, we present the numerical experiments of the IICA, ICA, and GA and discuss the results. Finally, in section 6, we conclude the paper and state its main outcomes.

2 The Huxley equation and its discretization

In this section, we first present the formulation of the Huxley equation in subsection 2.1, followed by the discretization of this equation in subsection 2.2, which is utilized to construct the fitness function.

2.1 The Huxley equation

The general form of the Huxley equation is as follows:

$$\frac{\partial U}{\partial t} = \frac{\partial^2 U}{\partial x^2} + U(1 - U^\delta)(U^\delta - \gamma), \tag{1a}$$

with initial and boundary conditions:

$$U(x, 0) = f(x), \tag{1b}$$

$$U(0, t) = p(t), \tag{1c}$$

$$U(1, t) = q(t), \tag{1d}$$

where δ is a positive integer, and $\gamma \in (0, 1)$. In this paper, we consider $0 \leq t \leq 1$ and $0 \leq x \leq 1$.

Additionally, the over-specified condition (data coming from a sensor) is as follows:

$$U(a, t) = s(t_j), \quad t_j = k \times j, \quad j = 1, 2, 3, \ldots, M. \tag{2}$$

Here, a represents the location of the sensor, k is the discretization step size of time, $s(t_j)$ is the value measured by the sensor at time t_j, and $x = a$.

2.2 Discretization of the Huxley equation

In this study, we use an implicit finite difference approximation (Crank–Nicolson) method, to discretize (1). As a result, we obtain the following discretized representation for the Huxley equation:

$$
\begin{aligned}
&-r_1U_{i-1,j+1}+(2+2r_1)U_{i,j+1}-r_1U_{i+1,j+1}\\
&\quad=r_1U_{i-1,j}+(2-r_2-2r_1)U_{i,j}+r_1U_{i+1,j}+2r_2{U_{i,j}}^2-r_2{U_{i,j}}^3,\\
&\qquad\qquad i=1,\ldots,N-1,\ j=0,\ldots,N-1, \qquad (3a)\\
&U_{i,0}=f(ih),\ j=0,\ i=1,\ldots,N-1, \qquad (3b)\\
&U_{0,j}=p(jk),\ i=0,\ j=0,1,\ldots,N-1, \qquad (3c)\\
&U_{N,j}=q(jk),\ Nh=1,\ j=0,1,\ldots,N-1, \qquad (3d)
\end{aligned}
$$

where $x=ih,\quad i=0,1,\ldots,N-1$ and h is the step size of the discretization of x, $t=jk,\quad j=0,1,\ldots,N-1$, and k is the step size of the discretization of t, $r_1=k/h^2$ and $r_2=2k$.

Using (3), we obtain the following linear algebraic system of equations:

$$
\begin{pmatrix}
2+2r_1 & -r_1 & 0 & 0 & 0 & 0 & 0\\
-r_1 & 2+2r_1 & -r_1 & 0 & 0 & 0 & 0\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
0 & 0 & 0 & 0 & -r_1 & 2+2r_1 & -r_1\\
0 & 0 & 0 & 0 & 0 & -r_1 & 2+2r_1
\end{pmatrix}
\begin{pmatrix}
U_{1,j+1}\\ U_{2,j+1}\\ \vdots\\ U_{N-2,j+1}\\ U_{N-1,j+1}
\end{pmatrix}
$$

$$
=\begin{pmatrix}
2-r_2-2r_1 & r_1 & 0 & 0 & 0 & 0 & 0\\
r_1 & 2-r_2-2r_1 & r_1 & 0 & 0 & 0 & 0\\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots\\
0 & 0 & 0 & 0 & r_1 & 2-r_2-2r_1 & r_1\\
0 & 0 & 0 & 0 & 0 & r_1 & 2-r_2-2r_1
\end{pmatrix}
\begin{pmatrix}
U_{1,j}\\ U_{2,j}\\ \vdots\\ U_{N-2,j}\\ U_{N-1,j}
\end{pmatrix}
$$

$$
+r_1\begin{pmatrix}
U_{0,j}+U_{0,j+1}\\ 0\\ \vdots\\ 0\\ U_{N,j}+U_{N,j+1}
\end{pmatrix}
+\begin{pmatrix}
2r_2{U_{1,j}}^2-r_2{U_{1,j}}^3\\
2r_2{U_{2,j}}^2-r_2{U_{2,j}}^3\\
\vdots\\
2r_2{U_{N-2,j}}^2-r_2{U_{N-2,j}}^3\\
2r_2{U_{N-1,j}}^2-r_2{U_{N-1,j}}^3
\end{pmatrix}
$$

where $x=ih,\quad i=0,1,\ldots,N-1$ and h is the step size of the discretization of x, $t=jk,\quad j=0,1,\ldots,N-1$, and k is the step size of the discretization of t, $r_1=k/h^2$ and $r_2=2k$.

In this study, the IICA, ICA, and GA are used to approximate the unknown function $q(t)$ in (1). Specifically, $q(t)$ is treated as a candidate solu-

tion represented as a real-valued vector (coefficients of a polynomial), which is then input into the fitness function for assessment. To evaluate the fitness of a candidate solution, system (3) is solved, and the numerical values of $U(x_i, t_j)$ are computed. Subsequently, the vector $\hat{s}(t_j) = U(x = a, t_j)$ is compared to the vector $s(t_j)$ as described in (2). To perform this comparison, the mean squared error is calculated. Smaller values of the mean squared error between $\hat{s}(t_j)$ and $s(t_j)$ indicate a better approximation of the unknown function $q(t)$. The pseudo-code of the fitness function in this study is as follows:

Algorithm 1: Pseudo-code of the fitness function

Data: Input values: Coefficients of a polynomial approximating $q(t)$

Result: Fitness value of the input values

Function `Fitness(`*Coefficients of a polynomial approximating* $q(t)$`):`

Calculate $U(x_i, t_j)$ using System (3)

return $\frac{1}{\sum_{j=1}^{m}(U(a,t_j)-s_{t_j})^2}$;

In Algorithm 1, as the approximation of $q(t)$ converges towards the exact $q(t)$, the denominator decreases. Consequently, the value of the fitness function increases.

3 Improved imperialistic competitive algorithm (IICA)

The ICA is a powerful meta-heuristic algorithm that has been successfully applied to a wide range of problems in science and engineering. Additionally, in recent years, several authors have attempted to present improved versions of the algorithm to enhance its effectiveness for optimization problems [7, 8, 35, 1, 34, 31, 33, 25, 32, 18, 13, 23]. This research paper represents the first attempt to enhance the original ICA, transforming it into a powerful method for function estimation in differential equations. In this section, we first present the original ICA briefly in Subsection 3.1. Then, in Subsection 3.2, we present the improved version of the ICA, which is a powerful method for function approximation in solving differential equations.

3.1 Original ICA

The ICA is a robust and versatile meta-heuristic optimization technique that has gained great attention in the fields of science and engineering. This algorithm was developed to tackle a wide range of complex problems. The ICA was inspired by the dynamics of imperialistic competition in societies. This algorithm emulates the concept of countries competing for dominance and resources, where each candidate solution to an optimization problem is treated as an independent "country." These countries try to spread their dominance through various interactions, such as assimilation and colonization [4]. The main steps of the original ICA are as follows:

1. initialization: First, initialize a population of candidate solutions (countries) randomly. Each country is considered as follows:

$$\text{country}_i = \{a_1, a_2, \ldots, a_m\}.$$

Here, country_i is the ith candidate solution with size m. In fact, $a_j, 1 \leq j \leq m$ indicate the coefficients of a polynomial as follows:

$$y(x) = a_m x^{m-1} + a_{m-1} x^{m-2} + \cdots + a_2 x^1 + a1.$$

Next, evaluate the fitness of each candidate solution. Then, select the top $N_{impires}$ countries as the imperialists. Finally, form the empires by dividing the remaining countries (colonies) among the imperialists in proportion to the fitness of the imperialists.

2. Assimilation: In every empire, the colonies move towards their imperialist using a randomly adjusted vector, which is scaled by a proximity factor. This stage emulates the impact of imperialism on the colonies and attempts to improve the fitness of each colony. The assimilation operator works to bring the colonies of an empire closer to the characteristics of the imperialist state within the search space. It like guides the colonies to adopt the traits of the imperialist, somewhat similar to how cultural assimilation happens where colonies start to resemble the imperialist in certain ways.

3. Exchanging positions of the imperialist and a colony: A colony may find a better position than the imperialist as it moves closer to it. In this situation, the imperialist and the colony swap their positions and the algorithm continues. The exchange process involves the transfer of colonies between imperialists based on the fitness value. This exchange aims to improve the overall quality of both imperialists and colonies.

4. Imperialistic competition:

   At first, the total power of an empire is evaluated using the following equation:

$$\text{T.C. }_n = \text{Fitness}\left(\text{imperialist}_n\right) + \xi \text{mean}\left\{\text{Fitness}\left(\text{colonies of empire}_n\right)\right\} \tag{4}$$

   Here $T.C_n$ is the total fitness of the nth empire, and ξ is a positive number that is considered to be less than 1. The strength of an empire mostly depends on how strong its imperialist country is. However, the colonies within that empire also have some influence, although it is not very significant. The author suggests that by adjusting a factor called ξ, we can change how much the colonies contribute to the empire's overall power. They recommend setting ξ at 0.1 for a balanced approach.

   Then, the imperialistic competition begins. All empires attempt to acquire colonies belonging to other empires and take control of them. This competitive, imperialistic process results in the gradual weakening of less powerful empires and the strengthening of more dominant ones. This competition is modeled by assigning one of the weakest colonies from the weakest empires to a dominant empire. The dominant empire is the one that wins the competition, which is based on the $T.C$ values of the empires.

5. Eliminating the powerless empires: An empire that has no power will be destroyed in the imperialistic competition and its colonies will be distributed among other empires. Different criteria can be used to model the destruction mechanism and determine when an empire has no power. In the original ICA, an empire collapses when it loses all of its colonies.

Steps 3.1 through 3.1 are repeated until the stop criterion is met. These steps are illustrated in Figure 1. Furthermore, Figure 2 illustrates the flowchart of the original ICA.

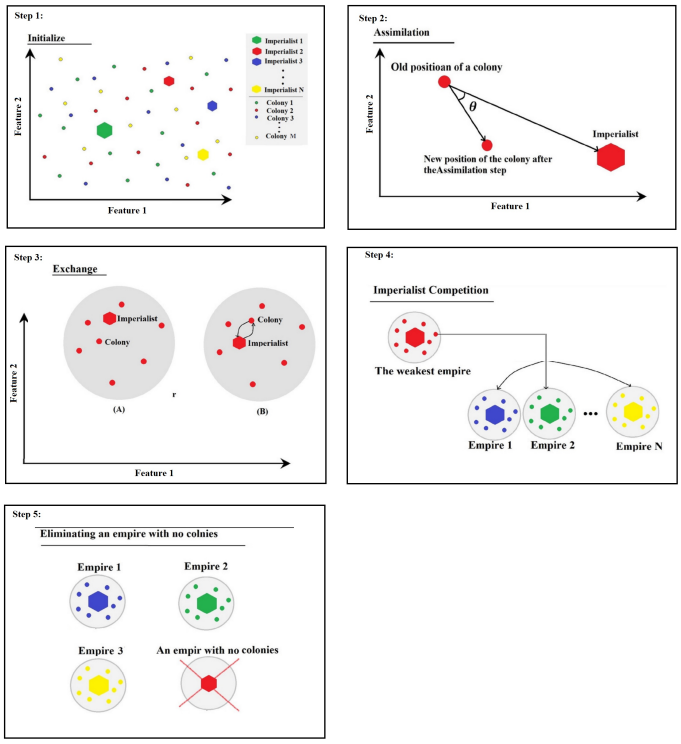

Figure 1: The main steps of the original ICA.

3.2 Improved imperialistic competitive algorithm (IICA)

Our improvements to the original ICA are as follows:

1. Smoothness: This improvement arises from the fact that in many methods, the unknown function that needs to be found is assumed to be smooth [14]. In our paper, we assume that the unknown function $q(t)$ is a polynomial of degree n. The IICA, ICA, and GA attempt to approx-

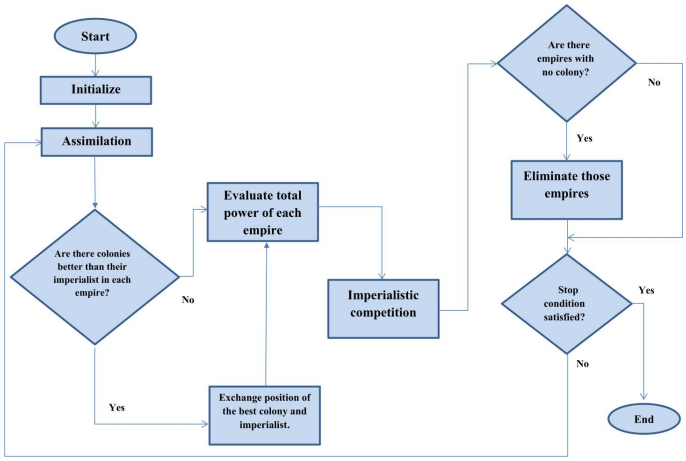

Figure 2: The flowchart of the original ICA.

imate the coefficients of the unknown function $q(t)$ such that the fitness value of the approximated $q(t)$ is maximized. During the middle iterations of the algorithm execution, the values of the coefficients of $q(t)$ may vary too much between two successive values, which might cause a big jump between $q(t_k)$ and $q(t_{k+1})$. Consequently, we introduce a step in which a procedure is executed to make the successive values of the unknown vector $q(t_j)$ much smoother. To apply this smoothness procedure to any elements of the coefficients vector $q(t)$, the following steps are performed:

- Calculate the mean of c_k and c_{k+2} as follows:

$$m = \frac{c_k + c_{k+2}}{2}.$$

- Then, move the value of c_{k+1} toward m as follows:

$$c_{k+1} = \frac{c_{k+1} + \alpha \times m}{1 + \alpha}, \tag{5}$$

where c_j is a coefficient of the candidate solution approximating the unknown $q(t)$ and $\alpha \in \mathbb{R}$ is a hyper-parameter, which should be tuned efficiently. Note that, at the beginning of the algorithm, we have the value $q(t_0) = q(0) = f(0)$ because the initial condition is known. Additionally, for the last element of the candidate solution, we use the

preceding value of m. The effect of the smoothness step is demonstrated through an example in Appendix A.

2. Correction: In many applications of meta-heuristic algorithms, there exists a search space for the values of the unknown vector, typically within the range $[LB, UB]$, where LB and UB stand for lower bound and upper bound respectively. In our algorithm, following the smoothness and assimilation steps, some values may surpass the interval $[LB, UB]$. Consequently, it becomes essential to reposition these values within the valid interval. This procedure is executed as follows:

$$c_k = c_k - \beta,$$

where β represents the amount by which c_j has exceeded $[LB, UB]$. For example, let us consider $UB = 20$, and suppose that c_j has reached a value of 25 after the smoothness and assimilation steps. Following the correction step, c_j will be adjusted to 15.

The whole procedure of the IICA is as follows:

1. Initialization: Generate randomly an initial population and create empires.

2. Assimilation: In every empire, the colonies move towards their imperialism using a randomly adjusted vector.

3. Smoothness: The smoothness procedure is applied to the candidate solutions.

4. Correction: The correction step is applied to the candidate solutions to keep them inside the valid interval.

5. Evaluation: Evaluate the fitness of the candidate solutions.

6. Exchanging position: The imperialist and the colony swap their positions if the colony is better than the imperialist.

7. Evaluation of empires' total power: The total power of the empires is evaluated.

8. Imperialistic competition: The imperialistic competition is done.

9. Collapse: Empires without colonies collapse.

10. Repeat Step 3.2 to Step 3.2, until the predefined number of iterations is not satisfied.

In fact, these two additional steps apply regularization to the coefficients of a polynomial that approximates the unknown $q(t)$, similar to the regularization that is done in the machine learning realm [14]. Figure 3 illustrates the steps of the improved ICA. We repeat steps 2 through 7 until the stop criterion is met. Furthermore, Figure 4 illustrates the flowchart of the improved ICA.

Table 1 presents the parameters of a real-valued GA used in this paper.

Table 1: Parameters of the IICA

Representation	Real valued vectors
Length of countries	Degree of a polynomial
Range of entries	$[-1, 1]$
Initialization	Random
Number of population	50 and 200
Number of empires	5 and 20
α	0.1
Collapse criteria	Having no colony
Termination condition	Number of generation

4 GA for the solution to inverse forms of Huxley equation

The GA, which was mainly developed by Holland [15], is a search method based on the Darwinian principles of biological evolution. This algorithm has been successfully used for various optimization problems. The GA is a stochastic optimization method that uses a population of chromosomes, each representing a possible solution. By applying a genetic operator, each chromosome improves gradually and becomes the basis for the next generation.

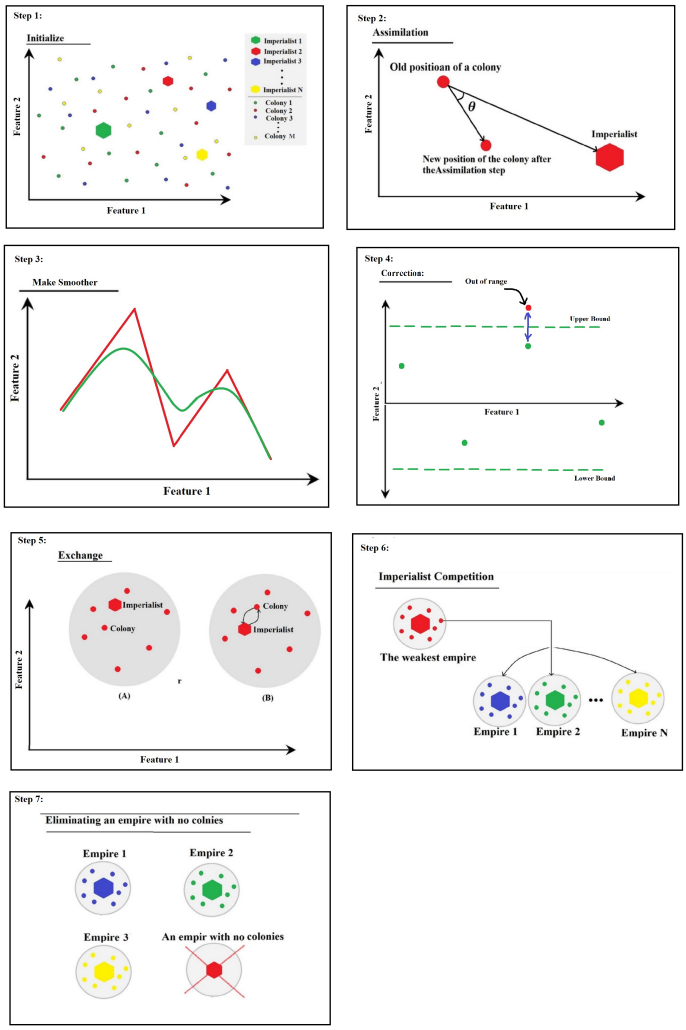

Figure 3: The steps of the improved ICA.

The process continues until the desired number of generations is reached or the predefined fitness value is achieved.

The procedure of a GA is as follows:

1. Generate at random an initial population of chromosomes.
2. Evaluate the fitness of each chromosome in the population.
3. Select some chromosomes as parents.

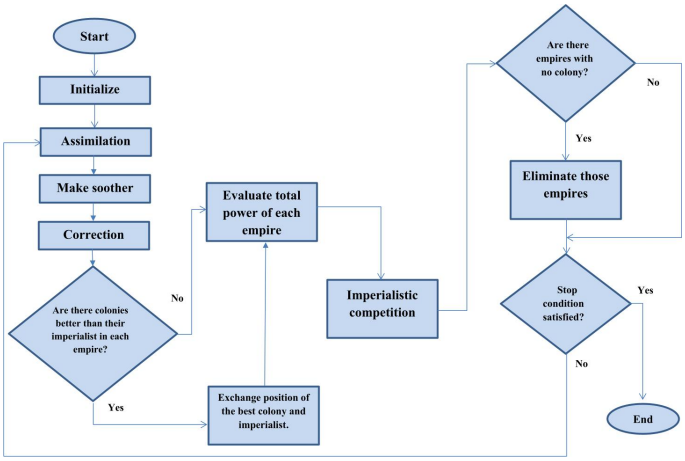

Figure 4: The flowchart of the improved ICA.

4. Apply recombination operation on parents.
5. Apply mutation operation on offspring.
6. Evaluate the fitness of offspring.
7. Update the population.
8. Repeat Step 4 to Step 4, until the predefined number of iterations is not satisfied.

Figure 5 presents the flowchart of the GA used in this paper.

Table 2 presents the parameters of a real-valued GA used in this paper.

To solve an inverse form of the Huxley equation using the GA presented in this section, we consider each candidate solution (chromosome) a real-valued vector as follows:

$$\text{Chromosome}_i = \{a_1, a_2, \ldots, a_m\},$$

where Chromosome_i is the ith candidate solution with size m in the population. Entries $a_j, 1 \leq j \leq m$ indicate the coefficients of a polynomial as follows:

$$y(x) = a_m x^{m-1} + a_{m-1} x^{m-2} + \cdots + a_2 x^1 + a1. \tag{6}$$

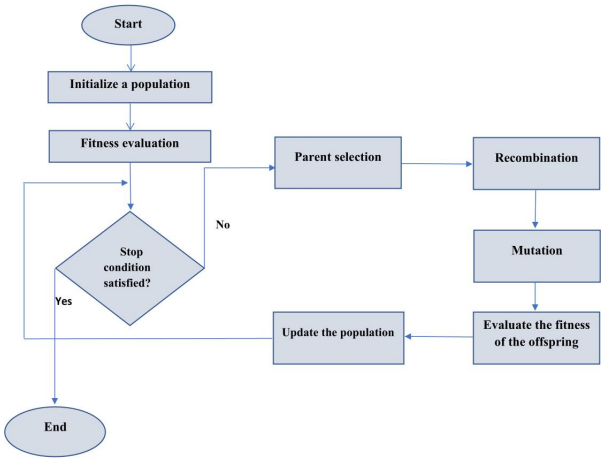

Figure 5: The flowchart of the GA used in this paper.

Table 2: Parameters of the GA

Representation	Real valued vectors
Length of chromosomes	Degree of a polynomial
Recombination	One point crossover
Recombination probability	100%
Mutation	Adding a random value
Mutation probability	$1/n$
Parent selection	Roulette wheel
Survivor selection	Replace the worst
Number of offspring	1
Initialization	Random
Termination condition	Number of generation

Each candidate solution such as (6) is considered as the missing condition $q(t)$ of the Huxley equation.

In our GA, we initially generate a random population of a specific size and evaluate their fitness. Then, as indicated in Figure 5, the main loop iterates the number of iterations times. In each iteration, two candidate solutions are selected as parents based on the roulette wheel selection method [11]. The recombination step, using one-point crossover, is applied to the selected

parents, and a new offspring is created. Subsequently, the mutation step is applied to the offspring by adding a small random value to a randomly chosen entry. The population is then updated by replacing the worst individual with the offspring.

5 Numerical examples

An inverse form of the Huxley equation, when $0 < x < 1, \quad 0 < t < t_M$, is as follows [5]:

$$U_t(x,t) = U_{xx}(x,t) + U(x,t)(1-U(x,t))(U(x,t)-1), \quad 0 < x < 1, \quad 0 < t < T, \tag{7a}$$

$$U(x,0) = \frac{1}{2} + \frac{1}{2}\tanh(\frac{1}{2\sqrt{2}}x), \tag{7b}$$

$$U(0,t) = \frac{1}{2} + \frac{1}{2}\tanh(\frac{-t}{4}), \tag{7c}$$

$$U(1,t) = q(t), \tag{7d}$$

and the over-specified condition

$$s(t_j) = U(0.5, t_j), \quad t_j = k \times j, \quad j = 1, 2, \ldots, N, \tag{7e}$$

where k is the discretization step size time (t), and the function $q(t)$ is missing. In this equation, the exact $U(x,t)$ and $q(t)$ are $\frac{1}{2} + \frac{1}{2}tanh(\frac{1}{2\sqrt{2}}(x - \frac{t}{\sqrt{2}}))$ and $\frac{1}{2} + \frac{1}{2}tanh(\frac{1}{2\sqrt{2}}(1 - \frac{t}{\sqrt{2}}))$, respectively.

The primary goal of this paper is to solve an inverse form of the Huxley equation by estimating its missing condition, denoted as $q(t)$. To assess the accuracy of the estimated function $\hat{q}(t)$, we employ the mean absolute error (MAE) criterion. We calculated the MAE value over the interval $[0, 1]$ with a step size of $h = 0.01$ for each algorithm to demonstrate its precision. Table 3 presents the MAE values comparing the estimated $\hat{q}(t)$ and the exact $q(t)$ for the implementations of IICA, ICA, and GA. The population size is set at 50 for all algorithms, α (the parameter for the smoothness step in Eq. (5)) is fixed at 0.1, and the initial number of empires for both IICA and ICA is 5. It is important to note that, as meta-heuristic algorithms are part of the

stochastic algorithm class, we ran the algorithms three times and reported the best result out of all the outcomes in this paper.

Table 3: The MAE in $[0,1]$ with the step size 0.01 between the exact $q(t)$ and the estimated function $\hat{q}(t)$ found by the IICA, ICA, and GA algorithms. These calculations were based on a population size of 50, an α value of 0.1, and 5 empires

$Num.of.Iter.$	MAE of the GA	MAE of the ICA	MAE of the IICA
100	0.05801	0.01227	0.00636
150	0.02166	0.01171	0.00444
200	0.01402	0.00608	0.00367
250	0.03842	0.00656	0.00417
300	0.04420	0.00392	0.00301
350	0.02635	0.00600	0.00221
400	0.01275	0.00351	0.00175
450	0.00825	0.00301	0.00186
500	0.00337	0.00255	0.00172

Figure 6 displays both the exact function $q(t)$ and the numerically approximated $\hat{q}(t)$ as determined by the IICA, which used 500 iterations, a population size of 50, an α value of 0.1, and 5 empires. The figure shows that the MAE across the interval $[0,1]$, with a step size of $h = 0.01$, is 0.00172. Furthermore, in this section, we will expand our examination to see how increasing the population size and the initial number of empires affects the performance of the IICA.

Figure 7 shows the discrepancy between the exact function $q(t)$ and its numerical approximation $\hat{q}(t)$, as derived by the IICA. This was achieved after 500 iterations, with a population size of 50, an α value of 0.1, and 5 initial empires.

We present the convergence patterns of the IICA, ICA, and GA as depicted in Figure 8, which is derived from Table 3. The figure clearly shows that the IICA converges more rapidly than the ICA, and the ICA, in turn, converges quicker than the GA. The figure also indicates that while the GAs performance varies within the interval, the ICA and IICA demonstrate a consistent improvement as the number of iterations increases.

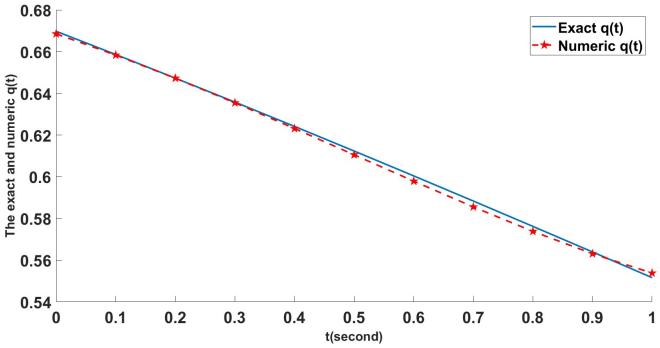

Figure 6: The exact $q(t)$ and the approximated (numeric) $\hat{q}(t)$ found by the IICA with iterations 500, population size 50, $\alpha = 0.1$, and the number of empires 5.

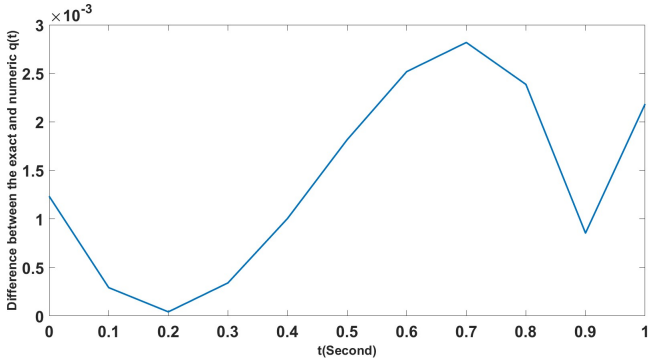

Figure 7: The difference between the exact $q(t)$ and approximated (numeric) $\hat{q}(t)$ found by the IICA with iterations 500, population size 50, $\alpha = 0.1$, and the number of empires 5.

Table 4 displays the MAE in $[0, 1]$ with the step size 0.01 comparisons for the exact function $q(t)$ against the approximated $\hat{q}(t)$ found by the IICA, ICA, and GA algorithms. These results were obtained with a population size of 200, an α value of 0.1, and an initial empire count of 20 for both the IICA and ICA. The table clearly indicates that the precision of the IICA and ICA improves with larger populations and more empires, whereas these changes do not significantly impact the performance of the GA.

Figure 9 showcases the exact $q(t)$ alongside the numerically approximated $\hat{q}(t)$ found by the IICA through 500 iterations, a population size of 200, an α of 0.1, and 20 empires. The figure indicates an MAE within the interval

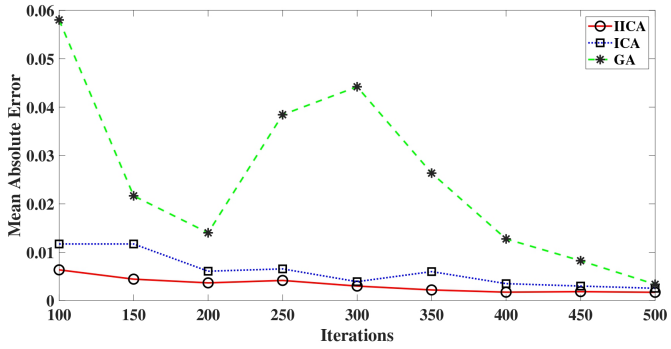

Figure 8: The convergence of the IICA, ICA, and GA extracted from Table 3.

Table 4: The MAE between the approximated $\hat{q}(t)$ and the exact $q(t)$ by the implementation of the IICA, ICA and GA for population size 200, $\alpha = 0.1$, and the number of empires 20

Num.of.Iter.	MAE of the GA	MAE of the ICA	MAE of the IICA
100	0.05280	0.00511	0.00302
150	0.06424	0.00320	0.00282
200	0.02392	0.00309	0.00139
250	0.03457	0.00320	0.00107
300	0.02326	0.00231	0.00104
350	0.03192	0.00203	0.00149
400	0.03222	0.00209	0.00094
450	0.03447	0.00147	0.00110
500	0.02410	0.00156	0.00083

$[0, 1]$, at a step size of 0.01, of 0.00083, which signifies a precise solution in the domain of inverse form of NPDEs.

Figure 10 presents the comparison between the exact $q(t)$ and its numerical approximation $\hat{q}(t)$ as produced by the IICA, following 500 iterations, with a population size of 200, an α of 0.1, and 20 empires. The figure demonstrates that the absolute error is generally on the order of $O(10^{-4})$ across most of the interval. Between approximately 0.2 and 0.5, the error increases to the order of $O(10^{-3})$. While there are fluctuations throughout the interval, the MAE consistently remains at the order of $O(10^{-4})$.

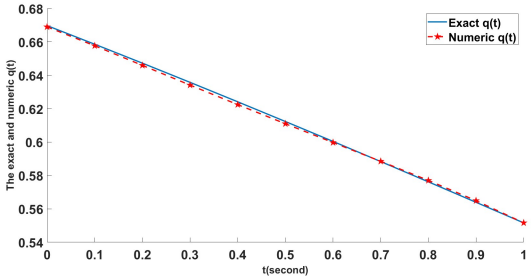

Figure 9: The exact $q(t)$ and approximated (numeric) $\hat{q}(t)$ found by the IICA with iterations 500, population size 200, $\alpha = 0.1$, and the number of empires 20.

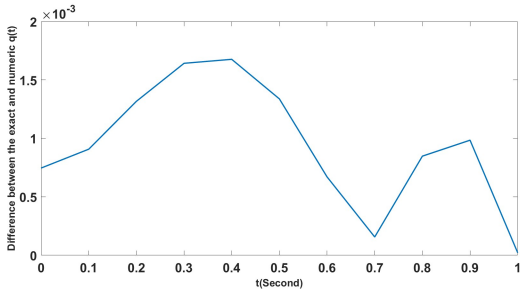

Figure 10: The difference between the exact $q(t)$ and approximated (numeric) $q(t)$ found by the IICA with iterations 500, population size 200, $\alpha = 0.1$, and the number of empires 20.

For illustrative purposes, we present the convergence of the IICA, ICA, and GA extracted from Table 4 in Figure 11. As evident from the figure, the convergence rate of the IICA surpasses that of the ICA and the GA. The error value curve of the IICA consistently lies below those of the original ICA and the GA. The performance of the GA fluctuates between 200 and 500 iterations, while both the ICA and the IICA steadily improve with an increasing number of iterations.

5.1 Discussion

According to our experiments, the best result is achieved when α is set to 0.1 (the parameter for the smoothness step in (5)). Additionally, based on

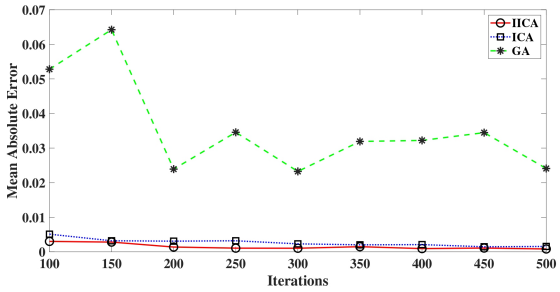

Figure 11: The convergence of the IICA, ICA, and GA extracted from Table 4.

the experiments, when the population size is less than 50, the accuracy of the results is low, and increasing the population size beyond 200 does not considerably improve accuracy. Therefore, we have reported these two values for the population size. Furthermore, according to the results, in general, the performance of the original ICA is better than that of the GA, and the performance of the IICA is better than both of them. Therefore, we focused on the IICA to plot figures and analyze its performance. A good solution using the IICA is obtained when the population size is 200, the size of the initial empires is 20, and the algorithm is iterated 500 times. In this case, the accuracy of the result is on the order of $O(10^{-4})$, which is good accuracy in the realm of the inverse form of NPDEs. Additionally, $[LB, UB]$ is $[-1, 1]$ for all experiments. Due to the nature of the IICA algorithm, the population size has a significant impact on its accuracy. This is the reason why we focused on this parameter. Figures 6 and 9 show the exact $q(t)$ and the numeric $\hat{q}(t)$ found by the IICA for population sizes 50 and 200, respectively. In these figures, it is evident that generally, at the beginning and end of the interval, the accuracy is better than in the middle of the interval. Moreover, for the population size 200 in Figure 9, the figures exactly match, indicating that the proposed improvement has reached a high accuracy. Figures 7 and 10 present the error study found by our proposed algorithm (IICA) for the population sizes 50 and 200, respectively. As can be seen from these figures, the overall accuracy is better for the population size 200 than for the population size 50. Figures 8 and 11 present the convergence of the IICA, the ICA, and the GA with iterations from 100 to 500 for the population sizes 50 and 200,

respectively. It is clear from the figures that our proposed algorithm (IICA) converges better than other algorithms with different population sizes. In fact, the error curve of the IICA is almost always below that of the ICA and the GA. The slope of the convergence curve for the IICA and ICA does not change significantly around the number of iterations of 500. Furthermore, these figures show that the original algorithm converges faster than the GA for a function approximation problem when the unknown function is considered a polynomial. As can be seen from Tables 3 and 4, increasing the size of the population does not help the GA reach a better solution, and this algorithm does not converge to a high accuracy when the number of iterations is increased to the population size of 200. On the other hand, the convergence of the IICA and ICA improves when the size of the population is increased from 50 to 200.

6 Conclusion

The improvements presented in this paper make the original ICA a much more powerful method for solving differential equations and function approximation. Since, in general, in real-world applications, the unknown functions are smooth, the smoothness procedure introduced in this paper helps the algorithm reach a high accuracy in function approximation tasks faster. Furthermore, this paper presents the application of meta-heuristic algorithms in solving inverse forms of NPDEs, which are categorized as ill-posed and challenging problems. The numerical results demonstrate that the IICA can effectively and proficiently solve inverse forms of nonlinear PDEs. Given the prevalence of inverse problems in applied fields, this method holds the potential for solving real-world challenges, which could lead to reduced execution times and enhanced accuracy. In this paper, we considered polynomials as basis functions to approximate the unknown function. In future research, another set of functions, such as orthogonal functions (e.g., Jacobi polynomials, Legendre polynomials, Chebyshev polynomials, and Gegenbauer polynomials), could be considered as the basis functions. Additionally, future research may involve the parallel implementation of the IICA. Moreover, other meta-heuristic algorithms could be employed to tackle this class of problems, al-

lowing for comparative analyses of their outcomes in relation to the results obtained in this study.

Appendix A

Figure 12 illustrates the impact of the smoothness procedure on the vector $x = [3, -7, 15, -6, -17, 18, -19, 9, 13, -4]$ when $\alpha = 0.5$. After applying the smoothness procedure, the resulting vector is denoted as $x' = [0.94, -2, 8.7, -5.4, -9.2, 7.3, -9.95, 6.5, 9.1, -2.2]$.

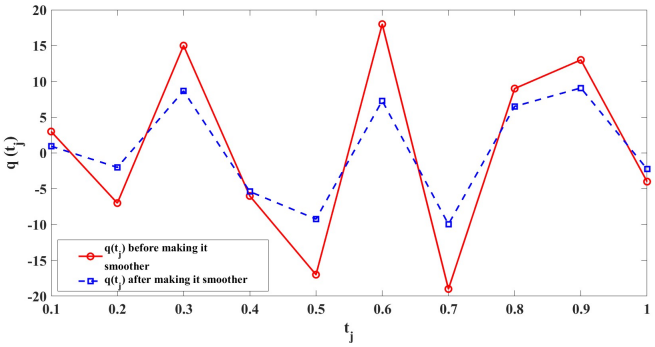

Figure 12: The impact of the smoothness procedure on a vector.

References

[1] Abbasi Molai, A. and Dana Mazraeh, H., *A modified imperialist competitive algorithm for solving nonlinear programming problems subject to mixed fuzzy relation equations*, Int. J. Nonlinear Anal. Appl. 14(3) (2023), 19–32.

[2] Abdollahi, M., Isazadeh, A. and Abdollahi, D. *Imperialist competitive algorithm for solving systems of nonlinear equations*, Comput. Math. Appl. 65(12) (2013), 1894–1908.

[3] Aliyari Boroujeni, A., Pourgholi, R. and Tabasi, S.H. *A new improved teaching–learning-based optimization (ITLBO) algorithm for solv-*

ing nonlinear inverse partial differential equation problems, Comput. Appl. Math. 42(2) (2023), 99.

[4] Atashpaz-Gargari, E. and Lucas, C. *Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition*, In 2007 IEEE, congress on evolutionary computation (pp. 4661-4667). IEEE, 2007.

[5] Babolian, E. and Saeidian, J. *Analytic approximate solutions to Burgers, Fisher, Huxley equations and two combined forms of these equations*, Commun. Nonlinear Sci. Numer. Simul. 14(5) (2009), 1984–1992.

[6] Barrios, D., Malumbres, L. and Rios, J. *Convergence conditions of genetic algorithms*, Int. J. Comput. Math. 68(3-4) (1998), 231–241.

[7] Bilel, N., Mohamed, N., Zouhaier, A. and Lotfi, R. *An improved imperialist competitive algorithm for multi-objective optimization*, Eng. Optim. 48(11) (2016), 1823–1844.

[8] Cai, J., Yang, H., Lai, T. and Xu, K. *A new approach for optimal chiller loading using an improved imperialist competitive algorithm*, Energy and Build. 284 (2023), 112835.

[9] Dai, Y.D., Zhang, H.L., Yuan, Y.P., et al. *The traveling solution of Huxley equation (in Chinese)*, Heilongjiang Sci. Technol. Inf. 11, 57 (2016).

[10] Dana Mazraeh, H., Kalantari, M., Tabasi, S.H., Afzal Aghaei, A., Kalantari, Z. and Fahimi, F. *Solving Fredholm integral equations of the second kind using an improved cuckoo optimization algorithm*, Glob. Anal. Discret. Math. 7(1) (2022), 33–52.

[11] Eiben, A.E. and Smith, J.E. *Introduction to evolutionary computing*, Springer, 2015.

[12] Fathy, A. and Rezk, H. *Parameter estimation of photovoltaic system using imperialist competitive algorithm*, Renew. Energy, 111 (2017), 307–320.

[13] Ghasemi, M., Ghavidel, S., Ghanbarian, M.M., Massrur, H.R. and Gharibzadeh, M. *Application of imperialist competitive algorithm with its modified techniques for multi-objective optimal power flow problem: a comparative study*, Inf. Sci. 281 (2014), 225–247.

[14] Girosi, F., Jones, M. and Poggio, T. *Regularization theory and neural networks architectures*, Neural Comput. 7(2) (1995), 219–269.

[15] Holland, J.H. *Adaptation in natural and artificial systems*, year 1975, publisher: University of Michigan Press.

[16] Loyinmi, A.C. and Akinfe, T.K. *An algorithm for solving the Burgers–Huxley equation using the Elzaki transform*, SN Appl. Sci. 2(1) (2020), 7.

[17] Mazraeh, H.D. and Pourgholi, R. *An efficient hybrid algorithm based on genetic algorithm (GA) and Nelder–Mead (NM) for solving nonlinear inverse parabolic problems*, Iranian Journal of Numerical Analysis and Optimization 8(2) (2018), 119–140.

[18] Molla-Alizadeh-Zavardehi, S., Tavakkoli-Moghaddam, R. and Lotfi, F.H. *A modified imperialist competitive algorithm for scheduling single batch-processing machine with fuzzy due date*, The International Journal of Advanced Manufacturing Technology 85 (2016), 2439–2458.

[19] Nemati, K., Shamsuddin, S.M. and Darus, M. *An optimization technique based on imperialist competition algorithm to measurement of error for solving initial and boundary value problems*, Measurement 48 (2014), 96–108.

[20] Pourgholi, R., Dana, H. and Tabasi, S.H. *Solving an inverse heat conduction problem using genetic algorithm: sequential and multi-core parallelization approach*, Appl. Math. Model. 38(7-8) (2014), 1948–1958.

[21] Rarità, L. *A genetic algorithm to optimize dynamics of supply chains*, In Optimization in Artificial Intelligence and Data Sciences: ODS, First Hybrid Conference, Rome, Italy, September 14-17, 2021 (pp. 107–115). Cham: Springer International Publishing, 2022.

[22] Rarità, L., Stamova, I. and Tomasiello, S. *Numerical schemes and genetic algorithms for the optimal control of a continuous model of supply chains*, Appl. Math. Comput. 388 (2021), 125464.

[23] Razzaghpour, M. and Rusu, A. *Analog circuit optimization via a modified Imperialist Competitive Algorithm*, In 2011 IEEE International Symposium of Circuits and Systems (ISCAS) (pp. 2273–2276). IEEE, 2011.

[24] Rooholamini, F., Afzal Aghaei, A., Hasheminejad, S.M.H., Azmi, R. and Soltani, S. *Developing Chimp Optimization Algorithm for Function Estimation Tasks*, Computational Mathematics and Computer Modeling with Applications (CMCMA) (2023), 34–44.

[25] Sharifi, M. and Mojallali, H. *Multi-objective modified imperialist competitive algorithm for brushless DC motor optimization*, IETE J. Res. 65(1) (2019), 96–103.

[26] Smith, G.D. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Oxford Applied Mathematics and Computing Science Series, Third Edition, year 1986, publisher: Oxford University Press.

[27] Tang, Y. and Zhou, F. *An improved imperialist competition algorithm with adaptive differential mutation assimilation strategy for function optimization*, Expert Syst. Appl. 211 (2023), 118686.

[28] Tomasiello, S. *Numerical solutions of the Burgers–Huxley equation by the IDQ method*, Int. J. Comput. Math. 87(1) (2010), 129–140.

[29] Tomasiello, S. *DQ based methods: theory and application to engineering and physical sciences*, In Handbook of Research on Computational Science and Engineering: Theory and Practice (pp. 316–346). IGI Global. 2012.

[30] Wang, X.Y. *Nerve propagation and wall in liquid crystals*, Phys. Lett. A, 112(8) (1985), 402–406.

[31] Xu, S., Wang, Y. and Lu, P. *Improved imperialist competitive algorithm with mutation operator for continuous optimization problems*, Neural Comput. Appl. 28 (2017), 1667–1682.

[32] Yousefi, M., Yousefi, M. and Darus, A.N. *A modified imperialist competitive algorithm for constrained optimization of plate-fin heat exchangers*, Proc. Inst. Mech. Eng. A: J. Power Energy 226(8) (2012), 1050–1059.

[33] Zandieh, M., Khatami, A.R. and Rahmati, S.H.A. *Flexible job shop scheduling under condition-based maintenance: improved version of imperialist competitive algorithm*, Appl. Soft Comput. 58, (2017), 449–464.

[34] Zhang, Y., Hu, X. and Wu, C. *Improved imperialist competitive algorithms for rebalancing multi-objective two-sided assembly lines with space and resource constraints*, Int. J. Prod. Res. 58(12) (2020), 3589–3617.

[35] Zhang, Y., Wang, Y. and Peng, C. *Improved imperialist competitive algorithm for constrained optimization*, In 2009 International Forum on Computer Science-Technology and Applications (Vol. 1, pp. 204-207). IEEE, 2009.