UDC 519.622.2:004.032.26 **DOI: 10.20998/2079-0023.2021.02.13**

N. A. MARCHENKO, G. YU. SYDORENKO, R. O. RUDENKO

USING OF MULTILAYER NEURAL NETWORKS FOR THE SOLVING SYSTEMS OF DIFFERENTIAL EQUATIONS

The article considers the study of methods for numerical solution of systems of differential equations using neural networks. To achieve this goal, the following interdependent tasks were solved: an overview of industries that need to solve systems of differential equations, as well as implemented a method of solving systems of differential equations using neural networks. It is shown that different types of systems of differential equations can be solved by a single method, which requires only the problem of loss function for optimization, which is directly created from differential equations and does not require solving equations for the highest derivative. The solution of differential equations' system using a multilayer neural networks is the functions given in analytical form, which can be differentiated or integrated analytically. In the course of this work, an improved form of construction of a test solution of systems of differential equations was found, which satisfies the initial conditions for construction, but has less impact on the solution error at a distance from the initial conditions compared to the form of such solution. The way has also been found to modify the calculation of the loss function for cases when the solution process stops at the local minimum, which will be caused by the high dependence of the subsequent values of the functions on the accuracy of finding the previous values. Among the results, it can be noted that the solution of differential equations' system using artificial neural networks may be more accurate than classical numerical methods for solving differential equations, but usually takes much longer to achieve similar results on small problems. The main advantage of using neural networks to solve differential equations` system is that the solution is in analytical form and can be found not only for individual values of parameters of equations, but also for all values of parameters in a limited range of values.

Keywords: systems of differential equations, artificial neural networks, multilayer neural network, numerical methods, gradient descent method, solution's error function

Н. А. МАРЧЕНКО, Г. Ю. СИДОРЕНКО, Р. О. РУДЕНКО

ВИКОРИСТАННЯ БАГАТОШАРОВОЇ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ РОЗВ'ЯЗАННЯ СИСТЕМ ДИФЕРЕНЦІАЛЬНИХ РІВНЯНЬ

В статті розглядається дослідження методів чисельного розв'язку систем диференціальних рівнянь з використанням нейронних мереж. Для досягнення поставленої мети були вирішені наступні взаємозалежні задачі: проведений огляд галузей, що потребують розв'язання систем диференціальних рівнянь, а також реалізований метод розв'язання систем диференціальних рівнянь за допомогою багатошарових нейронних мереж. В роботі показано, що різні типи систем диференціальних рівнянь можуть бути розв'язані єдиним методом, який потребує лише завдання функції втрат для оптимізації, що цілком створюється з диференціальних рівнянь та не потребує розв'язання рівнянь відносно найвищої похідної. Розв'язок систем диференціальних рівнянь за допомогою нейронних мереж є функції задані у аналітичній формі, що можуть бути диференційовані або інтегровані також аналітично. В ході виконання даної роботи була знайдена покращена форма побудови пробного розв'язку систем диференціальних рівнянь, що задовольняє початковим умовам за будовою, але має менший вплив на помилку розв'язку на відстані від початкових умов у порівнянні з формою побудови такого розв'язку. Також було знайдено спосіб модифікації розрахунку функції втрат для випадків, коли процес розв'язання зупиняється в локальному мінімумі, що спричиняться великою залежністю наступних значень функцій від точності знаходження попередніх значень. Серед результатів можна зазначити, що розв'язання систем диференціальних рівнянь за допомогою штучних нейронних мереж може мати точність порівняну з класичними чисельними методами розв'язання диференціальних рівнянь, але зазвичай потребує значно більшого часу для досягнення близьких результатів на задачах малих розмірностей. Основною перевагою використання нейронних мереж для розв'язання систем диференціальних рівнянь є те, що розв'язок знаходиться в аналітичній формі та може бути знайдений не тільки для окремих значень параметрів системи рівнянь, але й для всіх значень параметрів в обмеженій області значень.

Ключові слова: системи диференціальних рівнянь, штучні нейронні мережі, багатошарова нейронна мережа, чисельні методи, метод градієнтного спуску, функція похибки розв'язку

Н. А. МАРЧЕНКО, А. Ю. СИДОРЕНКО, Р. А. РУДЕНКО

ИСПОЛЬЗОВАНИЕ МНОГОСЛОЙНОЙ НЕЙРОННОЙ СЕТИ ДЛЯ РЕШЕНИЯ СИСТЕМ ДИФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

В статье рассматривается исследование методов численного решения систем дифференциальных уравнений с использованием нейронных сетей. Для достижения поставленной цели были решены следующие взаимосвязанные задачи: проведен обзор отраслей, требующих решения систем дифференциальных уравнений, а также реализован метод решения систем дифференциальных уравнений с помощью многослойных нейронных сетей. В работе показано, что различные типы систем дифференциальных уравнений могут быть решены единственным способом, который требует только задания функции потерь для оптимизации, создаваемого напрямую из систем дифференциальных уравнений и не требует решения систем уравнений относительно наивысшей производной. Решением систем дифференциальных уравнений с помощью нейронных сетей являются функции, заданные в аналитической форме, и могут быть дифференцированы или интегрированы также аналитически. В ходе выполнения данной работы была найдена улучшенная форма построения пробного решения систем дифференциальных уравнений, что удовлетворяет начальным условиям построения, но имеет меньшее влияние на ошибку решения на расстоянии от начальных условий по сравнению с формой построения такого решения. Также был найден способ модификации расчета функции потерь для случаев, когда процесс решения останавливается в локальном минимуме, что приводит к большой зависимости следующих значений функций от точности нахождения предыдущих значений. Среди результатов можно отметить, что решение систем дифференциальных уравнений с помощью искусственных нейронных сетей может иметь точность сравнимую с классическими численными методами решения дифференциальных уравнений, но обычно требует значительно большего времени для достижения более точных результатов на задачах малых размерностей. Основным преимуществом использования нейронных сетей для решения дифференциальных уравнений является то, что решение находится в аналитической форме и может быть найдено не только для отдельных значений параметров уравнений, но и для всех параметров в ограниченной области значений.

Ключевые слова: системы дифференциальных уравнений, искусственные нейронные сети, многослойная нейронная сеть, многочисленные методы, метод градиентного спуска, функция погрешности решения

Introduction. Differential equations and their systems are widely used in mathematical modeling to describe a variety of real processes: physical, environmental, biological, and other. Solving some equations in partial derivatives in cases that allow the separation of variables is also reduced to problems for ordinary differential equations. These are, as a rule, boundary value problems (problems of natural oscillations of elastic beams and plates, determination of the spectrum of natural values of particle energy in spherically symmetric fields, etc.). In addition, higher-order differential equations lead to the solution of systems of differential equations. It is known that solutions of differential equations and their systems can be found analytically or numerically. Finding analytical solutions is a very time consuming process, and in most cases impossible. Therefore, at present, traditional numerical methods are widely used to solve differential equations and their systems, among which the most well-known are Runge – Kutta methods, finite-difference methods, prediction and correction methods [1, 2].

The general problem of classical numerical methods is the need to choose their parameters to ensure a compromise between computational costs and the accuracy of the result. Therefore, in this work it is forbidden to use artificial multilayer neural networks, where, in contrast to classical methods, the solution is presented in analytical form, from which you can repeatedly take derivatives [3, 4]. Solutions are stored as neural network parameters, which requires much less memory than storing a solution as a discrete array in traditional numerical methods [5, 6]. The method is also universal and can therefore be used to solve different types of differential equations and their systems, both ordinary and partial derivatives [7-8].

The main advantage of using neural networks to solve differential equations' systems is that the solution is in analytical form and can be found not only for individual values of parameters of equations, but also for all values of parameters in a limited range of values.

A review of the literature showed the relevance of the problem and the feasibility of creating software. Therefore, the aim of this article is to solve systems of differential equations using a multilayer neural network.

The article's objective is to study the methods of numerical solution of ordinary differential equations` systems and to develop software for their solution using multilayer neural networks.

The mathematical formulation of the problem.

Suppose we need to solve a system of differential equations in the form '[1, 2]:

$$F(x, Y(x), Y'(x)) = 0, \quad (1)$$

with initial conditions

$$Y(x_0) = A,$$

where x is the vector of variable values;

$Y(x) = (y_1(x), y_2(x), \dots, y_N(x))$ – required function;

$Y(x_0) = y_1(x_0), y_2(x_0), \dots, y_N(x_0)$ – coordinates of initial conditions;

$A = a_0, a_1, a_2$ – their matching values.

To solve this problem, the solution is presented in the form [3, 5]:

$$Y^*(x) = N(x, p), \quad (2)$$

where N – neural network function with p parameters and input values x.

In this case, the initial conditions are not satisfied by the creation and therefore are studied gradually during the learning of the neural network.

The construction of the solution of differential equations' systems can be written in a form that satisfies the initial conditions from the beginning:

$$Y^*(x) = A(x) + \mathrm{Z}(x) \cdot N(x, p), \quad (3)$$

where $A(x)$ is a function that satisfies the initial conditions in advance;

$\mathrm{Z}(x)$ – function what construct as the points corresponding which are equal to zero to the coordinates of the initial conditions

$N(x, p)$ – output of backforward neural network with input x the weights p.

The task of a building function $A(x)$ is reduced to the task of the function that takes a certain values in the given points, and can take any value at all other points. To find the function, for example, an interpolation polynomic of Lagrange can be used in this case that looks like:

$$Pol(x) = \sum_{i=1}^{n} y_i l_i(x),$$

where $l_i(x)$ - basic polynomials are determined by the formula:

$$l_i(x) = \frac{x - x_0}{x_i - x_0} \dots \frac{x - x_{i-1}}{x_i - x_{i-1}} \dots \frac{x - x_{i+1}}{x_i - x_{i+1}} \dots \frac{x - x_n}{x_i - x_n}$$

To reduce the influence of the shape of the error of the approximation of the solution, we write the expression for $\mathrm{Z}(x)$ in the form:

$$Q(x) = \prod_{i=1}^{S} \mathrm{th}(x - x_i)^{d_i + 1} \quad (4)$$

A multilayer neural network of direct propagation is chosen as the structure of the neural network for solving differential equations' systems. The number of layers and the number of neurons in each layer are chosen based on the structure of the problem and the complexity of the form of the solution. These parameters are chosen after the experiments, because it is impossible to know in advance the optimal parameters of the neural network structure for each task.

The description of a multilayer neural network.

An artificial neural network is a structure that consists of a large number of processor elements, each of which has local memory and can interact with other elements [3, 4, 6, 9, 12]. This interaction takes place through communication channels in order to transmit data that can be interpreted in any way. Processor elements independently in time process the local data arriving to them through input channels. Changing the parameters of the algorithms of such processing depends only on the characteristics of the data. If we consider an artificial

neural network as an environment for information processing, then it can be set by defining the elements of this environment and the rules of their interaction.

Multilayer artificial neural networks can be considered as a serial connection of single-layer artificial neural networks of direct propagation. The structure of weights in these networks is organized in such a way that more complex classes are processed on layers of high-level neurons by combining and intersecting simple classes, which are formed at lower levels of artificial neural networks. There is strong evidence that two-layer artificial neural networks are able to recognize any class of convex shape, provided that it is possible to use a sufficient number of hidden layer neurons, and the weights are adjusted accordingly [8-9].

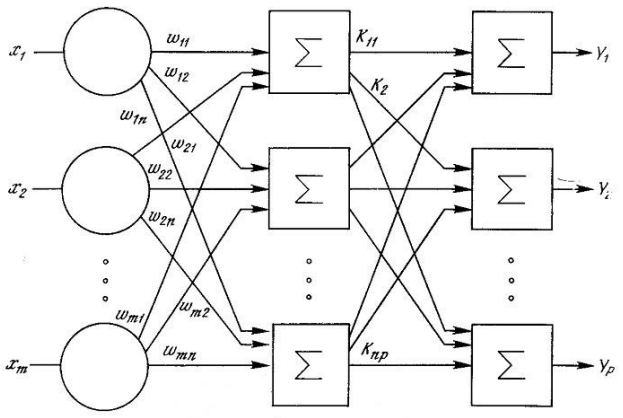

Fig. 1. The example of a scheme of a multilayer neural network of direct propagation

Artificial neural networks of direct propagation with several hidden layers are potentially capable of recognizing classes of arbitrary shape. Therefore, setting the problem on artificial neural networks of direct propagation includes determining the minimum possible number of neurons in the hidden layer and choosing an effective method of adjusting the weights. To date, both of these problems are not trivial. To explain the basic principles of building teaching methods with the teacher we will consider a two-layer artificial neural network. The zero layer of this network performs the auxiliary function of signal branching and does not contain neurons. For this reason, his work does not lead to modification of the input vector. The last layer of artificial neural networks is called the source layer. All layers located between zero and source are hidden layers with nonlinear activation function of neurons. In this example, we will consider one hidden layer with m neurons that use the hyperbolic tangent as an activation function.

It consists of m neurons that are simultaneously able to receive the input vector of signals $X = (x_1, \dots, x_i, \dots, x_n)$. To reproduce the elements of this vector use special devices, which are shown to the left of the neurons. These devices do not perform information processing, so they are not considered a layer of the neural network. According to the model of a formal neuron, each of its input signals is multiplied by a weighting factor w_{ij}, where i – the current vector element number X, a j – the current neuron number. All weights of a single-layer neural network form a matrix of weights

$$\boldsymbol{W} = \begin{bmatrix} w_{11} & \dots & w_{1j} & \dots & w_{1m} \\ . & \dots & . & \dots & . \\ w_{i1} & \dots & w_{ij} & \dots & w_{im} \\ . & \dots & . & \dots & . \\ w_{n1} & \dots & w_{nj} & \dots & w_{nm} \end{bmatrix}$$

Then the vector of arguments is defined as the product of $Y = WX$ and the vector of output signals is the vector of values of activation functions:

$$Y = F(V) = \begin{cases} f_1(v_1), \\ \dots \\ f_j(v_j), \\ \dots \\ f_m(v_m). \end{cases}$$

The name of the networks indicates that they have a dedicated direction of propagation of signals that move from the input through one or more hidden layers to the output layer. It is easy to see that a multilayer neural network can be obtained by cascading single-layer networks with matrices of weights $W^1, W^2, \dots, W^p$, where p is the number of layers of the neural network. If the multilayer neural network is linear, then for activation functions it can be reduced to the equivalent single-layer with a matrix of weights $W = W^1 * W^2 * \dots * W^p$. This means that the formation of such structures makes sense if nonlinear activation functions in neurons are used.

The gradient descent method for artificial neural networks.

The idea of the gradient descent method is to sequentially change the parameters of the artificial neural network in a direction that reduces the target function E [5]. Since the function E is differentiated by each of the parameters, it is possible to calculate the gradient vector. Moving in the direction of the negative gradient for each of the parameters, we find the local minima of the objective function. The change in the parameter is expressed by the formula:

$$\Delta W = -\eta \frac{dE}{dW} = \eta \sum_{i=1}^{N} \left(y^{(i)} - \tilde{y}^{(i)}\right) X^{(i)} = \qquad (5)$$

$$= \eta \sum_{i=1}^{N} (y^{(i)} - W^T X^{(i)})\, X^{(i)}.$$

This algorithm is called a batch-type algorithm, because to determine the magnitude of the step of changing the parameter, it is necessary to process the entire training sample.

The training sample $\Psi = \{(X^{(n)}, y^{(n)})\}_{n=1}^{N}$ containing N pairs: $x^{(n)}, y^{(n)}$ respectively, the input and output vectors and the set of parameters $W = \{w, v\}$, which consists of the parameters of the neurons of the hidden layer w and the parameters of the output layer v. The method of inverse propagation [13] is to minimize the objective function:

$$E = \frac{1}{2N}\sum_{n=1}^{N}[y^{(n)} - \tilde{y}^{(n)}(x)]^2 -> \min \quad (6)$$

The parameter v_j is searched as (7):

$$\frac{dE}{dv_j} = \frac{-1}{N}\sum_{n=1}^{N}(y^{(n)} - \tilde{y}^{(n)})\frac{d\tilde{y}^{(n)}}{dv_j} =$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)} - \tilde{y}^{(n)}\right)\varphi'\left(s^{(n)}\right)\frac{ds^n}{dv_j} = \quad (7)$$

$$= -\frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)} - \tilde{y}^{(n)}\right)\varphi'\left(s^{(n)}\right)h_j.$$

The step of changing the weights of the source layer is equal to (8):

$$\Delta v_j = -\eta\frac{dE}{dv_j} = \frac{\eta}{N}\sum_{n=1}^{N}(y^{(n)} - \tilde{y}^{(n)})\varphi'(s^{(n)})h_j. \quad (8)$$

The step of changing the weights of the hidden layer:

$$\Delta w_{ij} = -\eta\frac{dE}{dw_{ij}} =$$

$$= \frac{\eta}{N}\sum_{n=1}^{N}(y^{(n)} - \tilde{y}^{(n)})\varphi'(s^{(n)})v_j f'(t_j^{(n)})x_i^{(n)}.$$

But in fact, the correct calculation of values at points closer to the initial conditions is much more important than the calculation at points further away. To correct the optimization to take into account the influence of the values of the functions in the previous points on the values of the functions in the following points, the calculation of the loss function was modified to give the greatest weight to points closer to the initial conditions, keeping the sum of the loss function.

$$loss^*(x) = loss(x) * e^{-\frac{kx}{x_{max}}}, \quad (9)$$

$$loss^{**}(x) = loss^*(x) * \frac{\sum_{i=1}^{p} loss(x_i)}{\sum_{i=1}^{p} loss^*(x_i)}, \quad (10)$$

where $loss(x)$ – the loss function;

x – argument of the required function;

x_i – points at which optimization is performed;

p – the number of points at which optimization is performed.

The main results of the work.

The Python and R programming languages were used for perform this work, the TensorFlow library was chosen as the machine learning library with neural network learning support, and the PyCharm environment was used as the integrated development environment.

1. System of differential equations with constant coefficients

Consider the problem of solving a system consisting of three differential equations:

$$\begin{cases} y_1' = -y_1 + y_2, & y_1(0) = a_1, \\ y_2' = y_1 - 2y_2 + y_3, & y_2(0) = a_2, \\ y_3' = y_2 - y_3, & y_3(0) = a_3. \end{cases} \quad (11)$$

Solve the problem at once for many values of the initial conditions a_1, a_2, a_3 in the value ranges:

$$a_1 \in [1,4], a_2 \in [0,2], a_3 \in [0,2].$$

The loss function of the differential part of the equation and the cost of the initial conditions:

```
  def loss_pred(y_true, y_pred):
  y1,y2,y3=u[:,0:1],u[:,1:2],u[:,2:3]
  t = x
  dy1_dt = tf.gradients(y1, t)[0]
   dy2_dt = tf.gradients(y2, t)[0]
  dy3_dt = tf.gradients(y3, t)[0]
  eq1_r = - y1 +   y2
  eq2_r =   y1 - 2*y2 + y3
  eq3_r =   y2 -   y3
  eq1 = dy1_dt - grad_k(eq1_r, 0.1)
  eq2 = dy2_dt - grad_k(eq2_r, 0.1)
  eq3 = dy3_dt - grad_k(eq3_r, 0.1)
loss_diff=tf.reduce_mean(eq1**2+eq2**2+
eq3**2)
  initial_loc = (t - 0) ** 2 < 1e-7
  loss_initial=tf.reduce_sum(
(y1[initial_loc] - a1) ** 2 +
(y2[initial_loc] - a2) ** 2 +
(y3[initial_loc] - a3) ** 2) /
reduce_sum (tf.cast(initial_loc,
y1.dtype)) + 1e-10)
  return loss_diff + loss_initial
```

Neural network transformation function to meet the initial conditions for construction:

```
  def initial condition(x, u):
   def fn(v):
    x = v[0]
    u = v[1]
   def ths(z):
    return tf.tanh(z)
   return ths(x) * u + tf.constant([a1,
a2, a3])
   u = Lambda(fn)([x,u])
  return u
```

When using the model with the satisfaction of the initial conditions for construction, the cost function is simplified to:

```
  def loss_pred(y_true, y_pred):
```

```
    y1,y2,y3=u[:,0:1],u[:,1:2],u[:,2:3]
    t = x
    dy1_dt = tf.gradients(y1, t)[0]
    dy2_dt = tf.gradients(y2, t)[0]
    dy3_dt = tf.gradients(y3, t)[0]
    eq1_r = - y1 +   y2
    eq2_r =   y1 - 2*y2 + y3
    eq3_r =   y2 -   y3
    eq1 = dy1_dt - grad_k(eq1_r, 0.1)
    eq2 = dy2_dt - grad_k(eq2_r, 0.1)
    eq3 = dy3_dt - grad_k(eq3_r, 0.1)
loss_diff=tf.reduce_mean(eq1**2+eq2**2+
eq3**2)
  return loss_diff
```

The results of solving the problem are shown in fig. 2–5.

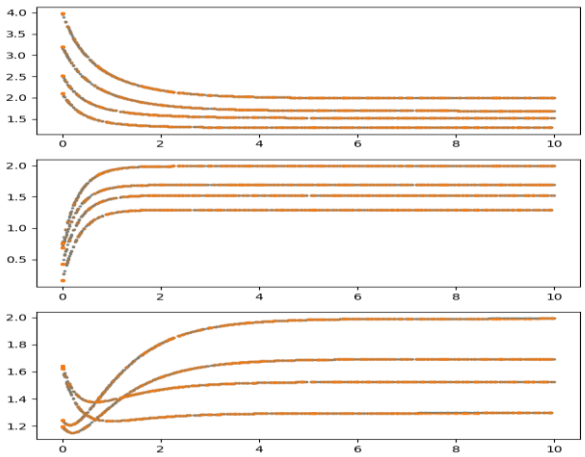

Fig. 2 The solving of the differential equations` system (11) as (2)

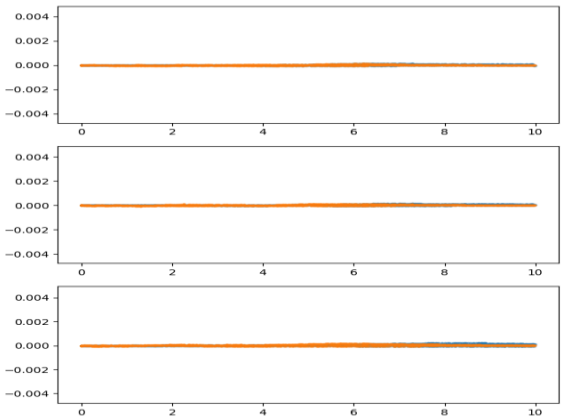

Fig. 3 The error function of the solving (2) for the differential equations` system (11)

The obtained optimization result in the basic form:

```
10000/10000-8s-loss:0.9901-rmse:0.0017-
val_loss:1.0000-val_rmse:7.7629e-04
```

The root mean square error is $7.7629 \cdot 10^{-4}$ compared to the implicit solution of the 4-th order Runge – Kutta method in steps of 10^{-3}. The error in half of the points closer to the boundary conditions is 4.16 times greater than the error in half of the points at a distance from the boundary conditions.

The obtained optimization result in a form that satisfies the initial conditions for construction:

```
10000/10000 - 9s - loss: 4.3334e-07 -
rmse: 1.5741e-04 - val_loss: 2.5469e-07
- val_rmse: 1.6917e-04.
```

The root mean square error is $1.6917 \cdot 10^{-4}$ compared to the implicit solution of the 4-th order Runge – Kutta method with step 10^{-3}. The error in half of the points closer to the boundary conditions is 1.06 times less than the error in half of the points at a distance from the boundary conditions.

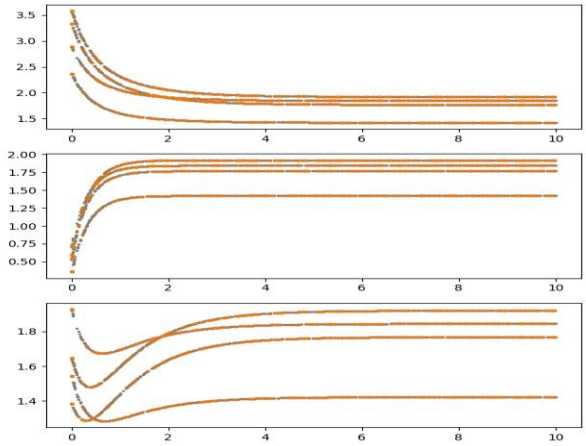

Fig. 4 The solving of the differential equations` system (11) as (3)

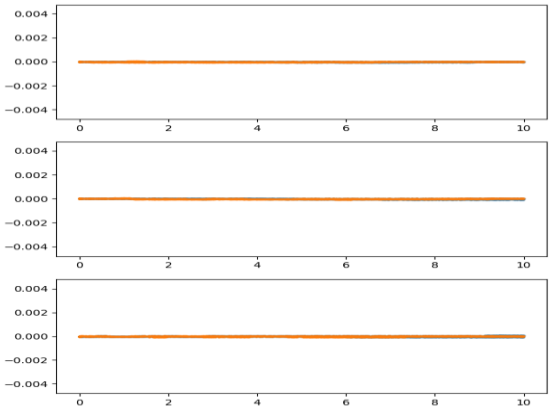

Fig. 5 The error function the differential equations` system (11) of the solving (3)

2. The system of nonlinear differential equations

Consider the problem of solving a system consisting of two differential equations:

$$\begin{cases} y_1' = 2(y_1 - y_1 * y_2), & y_1(0) = 1, \\ y_2' = -y_2 + y_1 * y_2, & y_2(0) = 3. \end{cases} \quad (12)$$

The loss function of the differential part of the equation and the loss of the initial conditions:

```
def loss_pred(y_true, y_pred):
```

```
  y1 = u[:, 0:1]
  y2 = u[:, 1:2]
  t = x
  dy1_dt = tf.gradients(y1, t)[0]
  dy2_dt = tf.gradients(y2, t)[0]
  eq1_r = 2* ( y1 - y1*y2 )
  eq2_r =    - y2 + y1*y2
  eq1 = dy1_dt - grad_k(eq1_r, 0.02)
  eq2 = dy2 dt - grad k(eq2 r, 0.02)

loss_diff=tf.reduce_mean(eq1**2+eq2**2)
    initial_loc = (t - 0) ** 2 < 1e-7
    loss_initial=tf.reduce_sum(
(y1[initial_loc] - 1) ** 2 +
(y2[initial_loc] - 3) ** 2) /
(tf.reduce_sum(tf.cast(initial_loc,
y1.dtype)) + 1e-10)
    return loss_diff + loss_initial
```

The neural network transformation function to satisfy the initial conditions of construction:

```
  def initial_condition(x, u):
    def fn(v):
      x = v[0]
      u = v[1]
      def ths(z):
        return tf.tanh(z*4)/4
 return ths(x)*u+tf.constant([1.0,3.0])
    u = Lambda(fn)([x,u])
  return u
```

When a model with satisfying conditions uses to construct the initial loss function extends to:

```
  def loss_pred(y_true, y_pred):
     y1 = u[:, 0:1]
     y2 = u[:, 1:2]
     t = x
    dy1_dt = tf.gradients(y1, t)[0]
    dy2_dt = tf.gradients(y2, t)[0]
    eq1_r = 2* ( y1 - y1*y2 )
    eq2_r =    - y2 + y1*y2
    eq1 = dy1_dt - grad_k(eq1_r, 0.02)
    eq2 = dy2_dt - grad_k(eq2_r, 0.02)
loss_diff=tf.reduce_mean(eq1**2+eq2**2)
al_loc, y1.dtype)) + 1e-10)
  return loss_diff
```

The obtained optimization result in the basic form:

```
10000/10000 - 1s - loss: 0.0055 - rmse:
1.1408 - val_loss: 0.0064 - val_rmse:
1.1348
```

Medium-square error is 1.1348.

The result of the optimization for the viewer (4) is shown in fig. 6–7, the result for the viewer (3) is similar. It can be seen that both options stopped at the local optimization minimum, this is due to the fact that the optimization of the neural network in the region $t \in [2.5, 3.5]$ significantly increases the error in the region $t \in [3.5, 10]$, although the value the loss function is calculated for all points of equal weight.

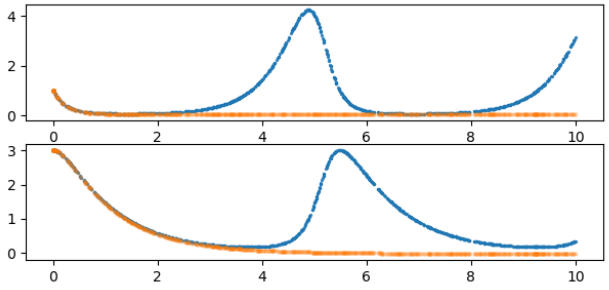

Fig. 6 The solving of the differential equations` system (12) as (2)

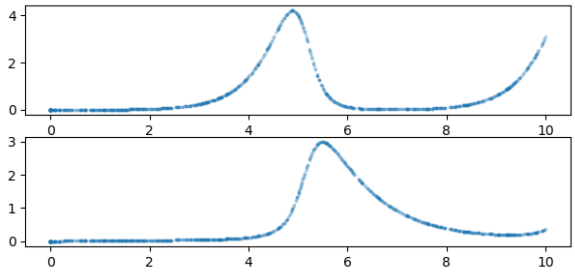

Fig. 7 The error function the differential equations` system (12) of the solving (2)

With the addition of the redistribution of the error function, the loss function is written as follows:

```
def loss_pred(y_true, y_pred):
    y1 = u[:, 0:1]
    y2 = u[:, 1:2]
    t = x
    dy1_dt = tf.gradients(y1, t)[0]
    dy2_dt = tf.gradients(y2, t)[0]
    eq1_r = 2* ( y1 - y1*y2 )
    eq2_r =    - y2 + y1*y2
    eq1 = dy1_dt - grad_k(eq1_r, 0.02)
    eq2 = dy2_dt - grad_k(eq2_r, 0.02)
    eq_err = eq1 ** 2 + eq2 ** 2
    eq_err_norm= q_err*K.exp(-
10*(t/K.max(t)))
    eq_err_norm = eq_err_norm *
K.stop_gradient(K.sum(eq_err) /
K.sum(eq_err_norm))
    loss_diff =
tf.reduce_mean(eq_err_norm)
    return loss_diff
```

After modifying the loss function, the problem was successfully solved. An example of the redistribution of the error function is shown in fig. 8–9.

The final result of solving the problem is shown in fig. 10–11. The obtained optimization result in a form that satisfies the initial conditions for construction:

`10000/10000-18s-loss:` $3.3917 \cdot 10^{-07}$ `-rmse:` $1.2223 \cdot 10^{-04}$ `- val_loss:` $2.3906 \cdot 10^{-07}$ `- val_rmse:` $1.3370 \cdot 10^{-04}$.

The solution has a root mean square error $1.3370 \cdot 10^{-04}$ in comparison with the solution of the implicit Runge – Kutta's method of the 4th order with the step 10^{-03}.

Conclusions. The system (12) is difficult to solve with neural networks and could not be solved without additional changes to the loss function, regardless of the form of solution. The applied modification can be used in other cases, when the solution of differential equations by optimization methods coincides to the local minimum.

When solving the system (11), the accuracy of the reproduction of the initial conditions had a significant effect on the whole solution. The error of the solution in the basic form was 4.59 times higher than the error of the solution in the form with satisfaction of the initial conditions for construction.

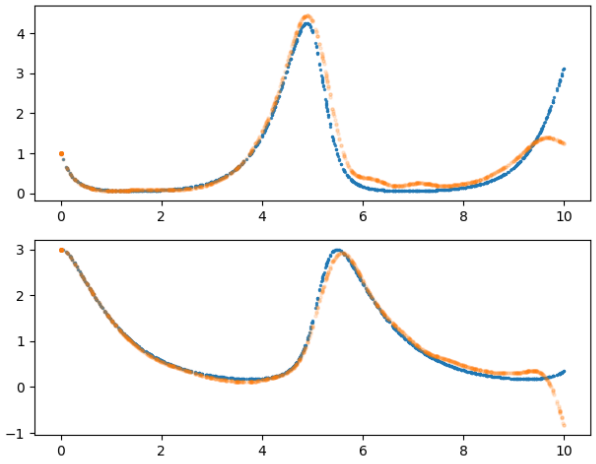

Fig. 8 The intermediate solution of the differential equations` system (12)

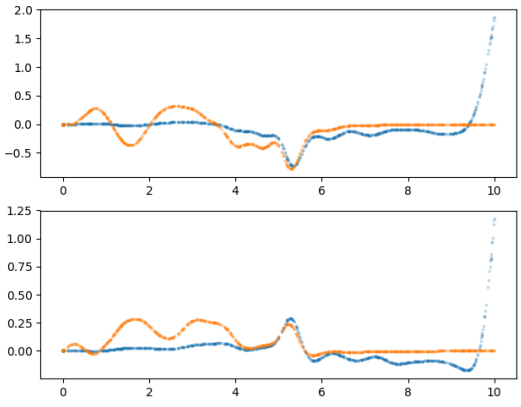

Fig. 9. The solution error`s function and the redistributed solution error function

Based on the results, we can say that the choice of the form of the solution and the construction of the loss function depends on the differential equations system and the needs of the problem to be solved. Some differential equations require special forms of construction of the loss function to be solved.

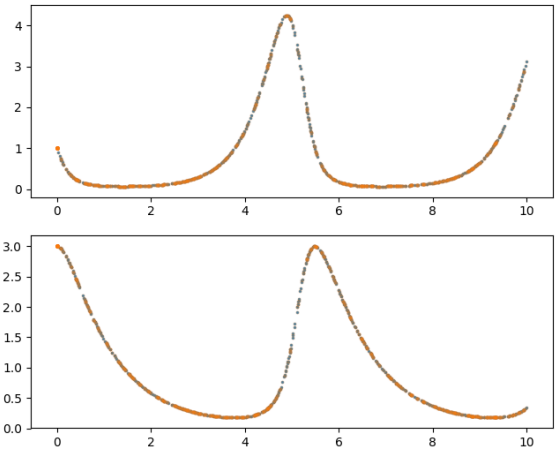

Fig. 10 The final solution of the differential equations system (12)

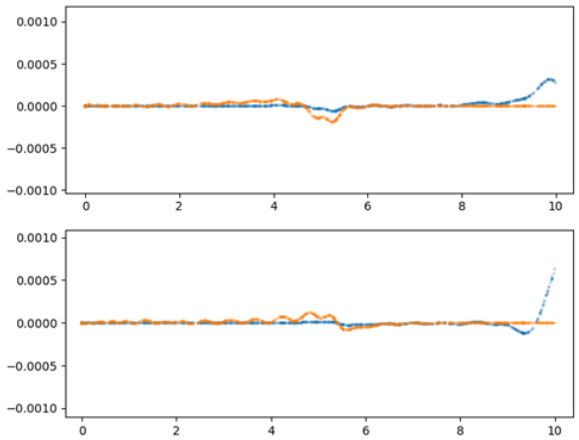

Fig. 11. The final solution error`s function and the redistributed solution error function of the differential equations system (12)

References

1. Задачин В. М. Конюшенко І. Г. *Чисельні методи: навчальний посібник*. Харків: Вид. ХНЕУ ім. С. Кузнеця, 2014. 180 с.
2. Хайер Э., Нессерт С., Ванер Г. *Решение обыкновенных дифференциальных уравнений. Нежесткие задачи*. Москва: Мир, 1999. 685 с.
3. Lagaris I. E., Likas A., Fotiadis D. I. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*. 1998. Vol. 9. No. 5. P. 987–1000.
4. Devipriya R., Selvi S. Modelling and Solving Differential Equations using Neural Networks: A Study. *International Journal of Computational Intelligence and Informatics*. 2020. Vol. 10. No. 1. P. 18–23.
5. Okereke R. N., Maliki O. S, Oruh B. I. A novel method for solving ordinary differential equations with artificial neural networks. *Applied Mathematics.* 2021. No. 12. P. 900–918. DOI: 10.4236/am.2021.1210059.
6. Tsoulos I. G., Gavrilis D., Glavas E. Solving differential equations with constructed neural networks. *Neurocomputing*. 2009. Vol. 72. No. 10–12. P. 2385–2391.
7. Корокая Л. И. Использование нейронных сетей при численном решении некоторых дифференциальных уравнений. *Eastern European Journal of Enterprise Technologies.* 2011. Vol. 3. No. 4(51). P. 24–27. ISSN 1729-3774.
8. Денисюк О.Р. Определение рациональных параметров численного решения систем дифференциальных уравнений некоторых классов. Вестник Херсонского национального

технического университета. № 3(58), 2016. С. 208–212. ISSN 2078-4481.
9. Aarts L. P., Van Der Veer P. Neural network method for solving partial differential equations. *Neural Processing Letters.* 2001. Vol. 14. No. 3. P. 261–271.
10. Sirignano J, Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*. 2018. Vol. 375. P. 1339–1364.
11. Baymani M., Kerayechian A., Effati S. Artificial Neural Networks Approach for Solving Stokes Problem. *Applied Mathematics*. 2010. Vol. 01(04). P. 288–292. DOI:10.4236/am.2010.14037.
12. Marchenko N. A, Sydorenko G. Yu, Rudenko R. O. Using neural networks to solve the differential equation. *Інформаційні системи та технології: праці 10-ї Міжнародної науково-технічної конференції, /* наук. ред. А.Д. Тевяшев, Л.Б. Петришин, В.В. Безкоровайний, В.Г. Кобзєв. Харків: ХНУРЕ, 2021. Р. 125–129.

References (transliterated)

1. Zadachyn V. M., Konyushenko I. G. Chysel'ni metody: Navchal'nyi posibnyk [Numerical methods]. Kharkiv, KhNEU Publ., 2014. 180 p.
2. Hayrer E., Wanner G. *Reshenie obyknovennyh uravnenij*. *Nezhestkie sadachi*. [Solving ordinary differential equations. Non-rigid tasks]. Moscow, Mir Publ., 1999. 685 p.
3. Lagaris I. E., Likas A., Fotiadis D. I. Artificial Neural Networks for Solving Ordinary and Partial Differential Equations. *IEEE Transactions on Neural Networks*. 1998, vol. 9, issue 5, pp. 987–1000.
4. Devipriya R., Selvi S. Modelling and Solving Differential Equations using Neural Networks: A Study. *International Journal of Computational Intelligence and Informatics*. 2020, vol. 10, issue 1, pp. 18–23.
5. Okereke R. N., Maliki O. S, Oruh B. I. A novel method for solving ordinary differential equations with artificial neural networks. *Applied Mathematics.* 2021, issue 12, pp. 900–918. DOI: 10.4236/am.2021.1210059.
6. Tsoulos I. G., Gavrilis D., Glavas E. Solving differential equations with constructed neural networks. *Neurocomputing*. 2009, vol. 72, issue 10–12, pp. 2385–2391.
7. Korotkaya L. I. Ispol'sovanie neyronnyh setej pri chislennom reshenii differentsyal'nyh uravneniy [The use of neural networks in the numerical solution of some systems of differential equations]. *Eastern European Journal of Enterprise Technologies*. 2011, issue 3, no. 4(51), pp. 24–27. ISSN 1729-3774.
8. Denisyuk O. R. Opredelenie ratsyonal'nyh parametrov chislennogo reshenia system differencial'nyh uravneniy nekotoryh klassov [Determination of rational parameters for the numerical solution of systems of differential equations of some classes]. *Visnyk of Kherson National Technical University* [Bulletin of the Kherson National Technical University]. Kherson Publ., 2016, no. 3 (58), pp. 208–212. ISSN 2078-4481.
9. Aarts L. P., Van Der Veer P. Neural network method for solving partial differential equations. *Neural Processing Letters*. 2001, Vol. 14, no. 3, pp. 261–271.
10. Sirignano J, Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*. 2018, vol. 375, pp. 1339–1364.
11. Baymani M., Kerayechian A., Effati S. Artificial Neural Networks Approach for Solving Stokes Problem. *Applied Mathematics*. 2010, no. 01(04), pp. 288–292. DOI:10.4236/am.2010.14037..
12. Marchenko N. A, Sydorenko G. Yu, Rudenko R. O. Using neural networks to solve the differential equation. *Informaciyni systemy ta tehnologii: praci 10th Mighnarodnoii konferencii* [Information systems and technologies IST-2021 Proceedings of the 10-th International Scientific and Technical Conference]. Kharkiv, KhNURE Publ., 2021, pp. 125–129.

Received 05.09.2021

Відомості про авторів / Сведения об авторах / About the Authors

Марченко Наталя Андріївна – кандидат технічних наук, доцент, доцент кафедри системного аналізу та інформаційно-аналітичних технологій НТУ «ХПІ», м. Харків, Україна; ORCID: https://orcid.org/0000-0001-9889-3713; e-mail: natalia.marchenko@khpi.edu.ua

Сидоренко Ганна Юріївна – кандидат технічних наук, доцент, доцент кафедри системного аналізу та інформаційно-аналітичних технологій НТУ «ХПІ», доцент кафедри моделювання систем і технологій ХНУ ім. В. Н. Каразіна, м. Харків; Україна; ORCID: https://orcid.org/0000-0002-0761-2793; e-mail: ganna.sydorenko@khpi.edu.ua

Руденко Роман Олександрович – магістр, інженер-програміст; м. Харків, Україна; ORCID: https://orcid.org/0000-0002-9424-6639; e-mail: roman.rudenko.a@gmail.com

Марченко Наталья Андреевна – кандидат технических наук, доцент, доцент кафедры системного анализа и информационно-аналитических технологий НТУ «ХПИ, г. Харьков, Украина; ORCID: https://orcid.org/0000-0001-9889-3713; e-mail: natalia.marchenko@khpi.edu.ua

Сидоренко Анна Юрьевна – кандидат технических наук, доцент, доцент кафедры системного анализа и информационно-аналитических технологий НТУ «ХПИ», доцент кафедры моделирования систем и технологий ХНУ им. В.Н. Каразина, г. Харьков, Украина; ORCID: https://orcid.org/0000-0002-0761-2793; e-mail: ganna.sydorenko@khpi.edu.ua

Руденко Роман Александрович – магистр, инженер-программист; г. Харьков, Украина; ORCID: https://orcid.org/0000-0002-9424-6639; e-mail: roman.rudenko.a@gmail.com

Marchenko Natalia Andriyivna – Candidate of Technical Sciences, Docent, Associate Professor at the Department of analysis of systems and information-analytical technologies NTU "KhPI", Kharkiv, Ukraine; ORCID: https://orcid.org/0000-0001-9889-3713; e-mail: natalia.marchenko@khpi.edu.ua

Sydorenko Ganna Yurijivna – Candidate of Technical Sciences, Docent, Associate Professor at the Department of analysis of systems and information-analytical technologies NTU "KhPI", Associate Professor at the Department of Modeling of systems and technologies KhNU by V. N. Karazin, Kharkov; Ukraine; ORCID: https://orcid.org/0000-0002-0761-2793; e-mail: ganna.sydorenko@khpi.edu.ua

Rudenko Roman Oleksandrovych – Master, Software engineer; Kharkiv, Ukraine; ORCID: https://orcid.org/0000-0002-9424-6639; e-mail: roman.rudenko.a@gmail.com