Filomat 38:32 (2024), 11553–11582
https://doi.org/10.2298/FIL2432553Y

Published by Faculty of Sciences and Mathematics,
University of Niš, Serbia
Available at: http://www.pmf.ni.ac.rs/filomat

Comparative analysis in function approximation with neural networks and some basic polynomials

Kostadin Yotov[a], Emil Hadzhikolev[a,*], Stanka Hadzhikoleva[a], Margarita Terziyska[b]

[a] *Faculty of Mathematics and Informatics, University of Plovdiv Paisii Hilendarski, Bulgaria*
[b] *Department of Mathematics, Physics and Information Technologies, University of Food Technologies, Bulgaria*

Abstract. Choosing the appropriate approximation method is a crucial step in solving a broad class of problems. It affects both the quality of the solutions obtained and the efficiency of the computational process. The article presents a study of various approximation cases with a specific group of polynomials and feedforward artificial neural networks. A comparative analysis of the results obtained has been conducted.

1. Introduction

The present study is part of a comparative analysis conducted on the approximation efficiency of classical approaches and different types of artificial neural networks, which covers the following stages:

1. Comparative analysis in approximation through polynomial approximations and artificial neural networks with different architecture and functionality;
2. Comparative analysis in the approximation of functions with the method of least squares and artificial neural networks;
3. Comparative analysis in approximation of functions through different types of splines and artificial neural networks.

This part describes a portion of the first stage of the research done and the results in different cases of approximation from a certain group of polynomials and feedforward neural networks. The focus is both on a comparison of the interpolation performance and juxtaposition of the errors of the different methods in subsequent prediction of function values outside of the training samples. The first part of the article discusses the basic principles of the most commonly used approximation polynomials. The basic principles and characteristics of neural networks are described, including their architectures and some of the popular learning algorithms. In the next part, a comparative analysis of the efficiency and applicability of neural networks and polynomial approximation approaches is performed. Their advantages and disadvantages are discussed, considering various aspects such as in-sample modeling accuracy and predictive ability.

2020 *Mathematics Subject Classification*. Primary 68T07, 65D05.
Keywords. approximation; approximation method; approximation cases; neural network approximations; polynomial approximations
Received: 21 March 2024; Revised: 05 April 2024; Accepted: 03 May 2024
Communicated by Miodrag Spalević
* Corresponding author: Emil Hadzhikolev
Email addresses: kostadin_yotov@uni-plovdiv.bg (Kostadin Yotov), hadjikolev@uni-plovdiv.bg (Emil Hadzhikolev), stankah@uni-plovdiv.bg (Stanka Hadzhikoleva), mterziyska@uft-plovdiv.bg (Margarita Terziyska)

The purpose of the study is to provide an objective comparison between neural networks and classical approximation polynomials, which will help those working on similar issues to choose the appropriate method for their tasks oriented towards specific data modeling and approximation. In contrast to previous studies that focused on comparing polynomial approximations and artificial neural networks, this study adds a differential consideration of small and large-volume samples. It also examines the influence of noise on the two methods and measures and compares the execution times of the different approaches.

2. Classical polynomial approximations and artificial neural networks used for data approximation

A polynomial approximation is a data approximation method that uses polynomials of different types and aims at representing the typically hidden functional dependence which the previously made measurements depend upon [1], [2], [3]. This method is based on the idea that we can approximate complex functions with simpler polynomial structures by using different optimization methods. Approximation by polynomials is widely used in various fields of mathematics, physics, engineering, and computer sciences. For example, in [4] a com-pressed sensing approach for polynomial approximation of multidimensional complex functions is proposed and analyzed, [5] consider the problem of approximate reconstruction of a function defined on the surface of a sphere in Euclidean space, and [6] use polynomial approximations to approximate the nonlinear relationship between the parameters and the state of a power system. In addition to interpolation, very often polynomial approximations can also be used to predict the future development of the observed quantities and extrapolate the available samples [7], [8], [9]. It should also be noted that polynomial approximation can also be used for multivariate data, in which case polynomials with more than one variable are used [10]-[12]. All these qualities make polynomial approximations a useful tool for data analysis and function modeling in various fields of science and engineering. This study focuses on some of the most commonly used approximation polynomials.

2.1. Lagrange polynomial

Let the given sample be

$$D = \{x_i \in R, i = 1, 2, \dots n; y_i = f(x_i), i = 1, 2, \dots, n\} \tag{1}$$

from observed values of the function $f(x)$. A standard approach to data approximation is to use the Lagrange polynomial [13]-[14]:

$$L_n(x) = \sum_{j=0}^{n} y_j F_j(x), \tag{2}$$

where y_j are the values of the function whose interpolation we seek for the given sample

$$F_j(x) = \prod_{k=0, k\neq j}^{n} = \frac{x - x_k}{x_j - x_k} = \frac{(x - x_0)(x - x_1)\dots(x - x_{j-1})(x - x_{j+1})\dots(x - x_n)}{(x_j - x_0)(x_j - x_1)\dots(x_j - x_{j-1})(x_j - x_{j+1})\dots(x_j - x_n)} \tag{3}$$

It is obvious that $F_j(x)$ are polynomials of degree n, and in the interpolation nodes we have:

$$F_j(x_i) = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if } i \neq j, \end{cases} \tag{4}$$

Therefore, the Lagrange polynomial degree is also of degree n, and at $x = x_i$, all addends in the sum $L_n(x)$ become zero, except for addend number $j = i$, equal to y_i.

So the Lagrange polynomial is

$$L_n(x_i) = y_i \tag{5}$$

i.e. for all values of the independent variable, the values of their dependent variable are returned, hence it is indeed an interpolation polynomial. In Fig. 1 a Lagrange polynomial (the red graph) is presented interpolation $\sin(x)$ in the segment $[-3, 3]$, using a sample of the following points $x = [-3, -1.5, 0, 1.5, 3]$ and the corresponding functional values.

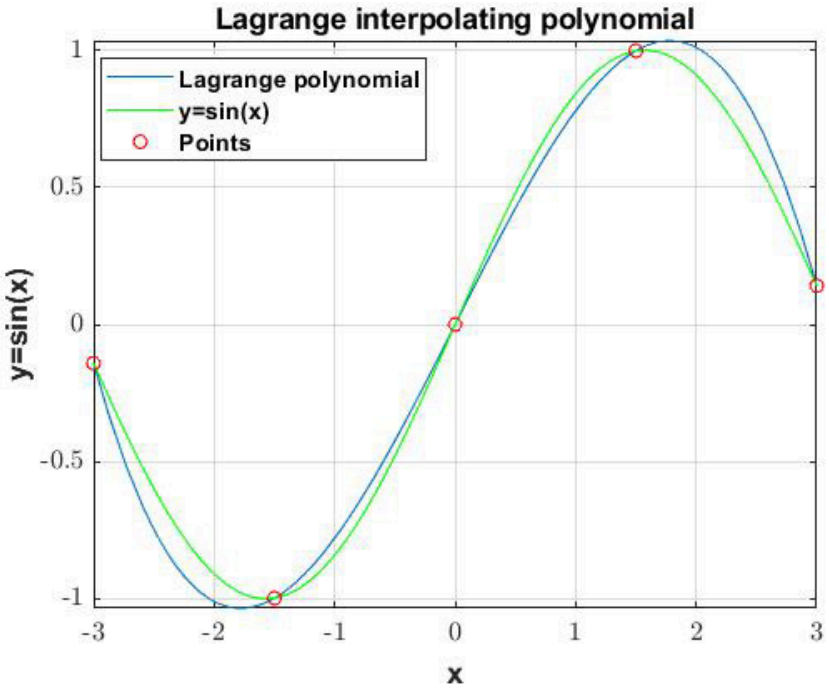


Figure 1: Lagrange interpolation polynomial for the function $\sin(x)$.

Often this type of polynomials is also used for predicting, as for example in [15]-[16]. The Lagrange polynomial approximation has several important advantages:

1. **Ease of use:** The approximation method with Lagrange polynomials is easy to understand and apply, especially compared to other similar approaches, such as the least squares method or the Chebyshev method.
2. **Flexibility:** Lagrange polynomials can be used to approximate functions in different domains and intervals. They can be adapted to different types and shapes of functions, which means that they are the right choice for a variety of tasks.
3. **Accuracy:** Lagrange polynomials can achieve high precision when approximating functions, especially if a large number of interpolation points are used. Given enough points, the Lagrange polynomial can approximate the true function with good accuracy. However, it should be noted that working with this polynomial requires increased attention with an excessively large number of points or complex functions because in such conditions the degree of the polynomial becomes very large and it may show a tendency to increase errors even within the limits of the samples with which it was formed.

It is worth noting here that when using many of the polynomial approximations, including the Lagrange polynomial, it is important to keep in mind that, in the general case, they can be sensitive to the location of the data and may cause problems such as excessive distortion of the function outside of the range of available data (the Runge phenomenon/Runge) [17]-[20]. For this reason, when choosing a polynomial for approximation, some factors need to be considered, such as the polynomial degree, the data location and distribution, and the application for which the approximation is being used.

2.2. Newton's polynomial

Newton's polynomial is an interpolation method that uses an interpolation polynomial of degree n, based on the set sample of points [20]-[21]. Newton's polynomial is based on the idea of divided differences.

For the given sample (1), the first-order divided differences have the representation

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{6}$$

as the divided differences of a higher order being inductively defined, for example from order $(k + 1)$:

$$f[x_i, x_{i+1}, \ldots, x_{i+k+1}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k+1}] - f[x_i, x_{i+1}, \ldots, x_{i+k}]}{x_{i+k+1} - x_i} \tag{7}$$

Newton's interpolation polynomial is based on the Lagrange polynomial and is introduced through the proof of the following theorem:

The Lagrange interpolation polynomial $L_n(x)$ can be represented like this:

$$L_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \ldots$$
$$+ f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_n). \tag{8}$$

The representation 8 of the Lagrange polynomial is called Newton's divided difference interpolation formula. Newton's polynomial has several advantages and is preferred in several cases:

1. **Easy calculation:** The process of generating Newton's polynomial and its coefficients is relatively simple when using software algorithms.
2. **Effectiveness:** When the sample points are slightly changed to obtain a new interpolation polynomial, the already calculated divided differences can be reused, reducing the computational effort.
3. **Universality:** Newton's polynomial can be used to interpolate arbitrary functions, including nonlinear ones and boundary conditions.
4. **Adaptability:** Newton's polynomial can be easily modified to add new points or remove existing points from the sample.

Despite its advantages, however, just like the case of Lagrange polynomial, Newton's polynomial has its limitations expressed in its increasing complexity as the samples become larger and its sensitivity to errors in the input data. In some cases where accuracy and efficiency are critical, other interpolation methods may be preferred, e.g. the Hermite polynomial or the Chebyshev polynomial.

2.3. Hermite polynomial

The Hermite polynomial is a special kind of polynomial which is also often used for interpolating and approximating functions. It is based on the idea of expanding the Lagrange or Newton's polynomials by including additional components that also include derivatives of the function at the interpolation points [22]-[23]. The Hermite polynomial of degree n can be defined by the following expression:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{x^n} e^{-x^2} \tag{9}$$

The first few Hermite polynomials are:

$$T_0(x) = 1$$
$$T_1(x) = 2x$$
$$T_2(x) = 4x^2 - 2$$
$$T_3(x) = 8x^3 - 12x$$
$$T_4(x) = 16x^4 - 48x^2 + 12$$
$$T_5(x) = 32x^5 - 160x^3 + 120x$$
$$T_6(x) = 64x^6 - 480x^4 + 720x^2 - 120$$
$$T_7(x) = 128x^7 - 1344x^5 + 3360x^3 - 1680x$$
$$T_8(x) = 256x^8 - 3584x^6 + 13440x^4 - 32x^2 + 1680$$

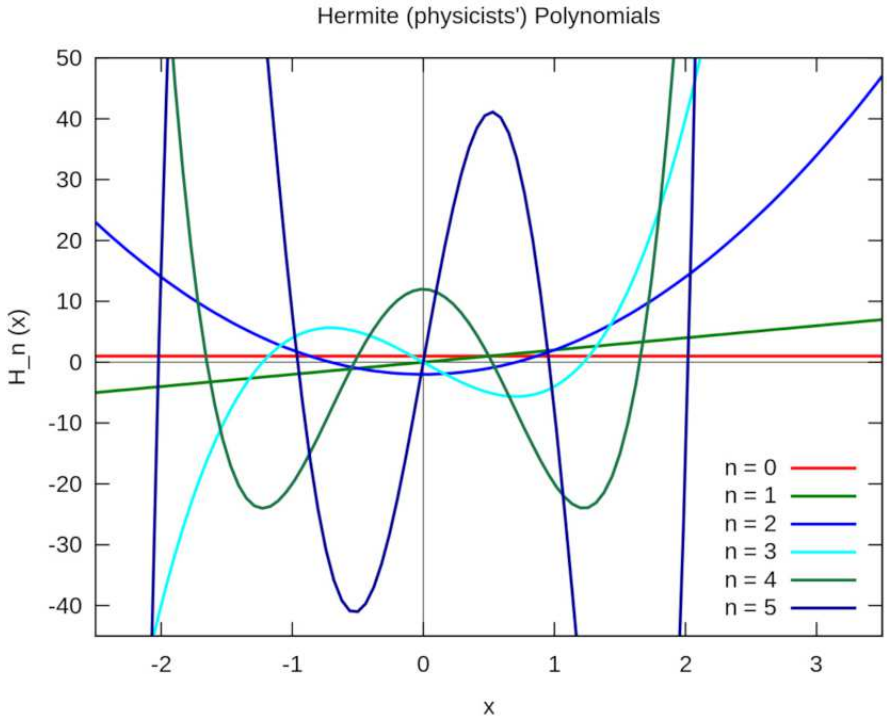


Figure 2: The first five Hermite polynomials

The approximation with the Hermite polynomial is done by polynomials based on the ones known from the predefined sample, values of the function and its derivatives in certain points. Similar approaches can be used when we have information about the values of the function and its derivatives at specific points and we want to construct the approximating polynomial to represent the function with predetermined accuracy.

The advantages of the Hermite polynomial approximation are:

1. **The possibility of interpolation and approximation of the function and its derivatives at specific points.** This means that we can represent not only the values of the function at the points, but also the information about its change and the rate of this change (the acceleration).
2. **The Hermite polynomial can represent complex forms of functions and capture details in the function's behavior.** It can approximate the curvature, the convexity and other characteristics of the function.
3. **It is not to be overlooked that Hermite polynomials can be constructed and used easily with suitable algorithms and software tools.** There are various libraries and programs that provide ready-made functions for creating and manipulating Hermite polynomials. In our study, for example, we use the built-in functions *hermite*, *hermitepoly*, and *hermiteinterp* in the Matlab environment. But similar tools can be found in Python, Maple, etc.

As a disadvantage of the approaches using the Hermite polynomial, it should be noted that in many cases the polynomial can turn out to be complex and require a large number of points for the correct representation of the function and its derivatives. This may lead to a higher computational load and larger data volume. In turn, the high degree of the polynomial can cause additional difficulties in a subsequent analysis of the results.

2.4. Chebyshev approximation

The Chebyshev polynomials are a certain group of polynomials that are used to approximate functions in a certain interval. They are based on the Chebyshev points and have specific properties that make them the preferred method in many cases for approximating functions [24]-[25]. Recurring formulas for Chebyshev polynomials of the first kind have the representation:

$$\begin{aligned} &T_0(x) = 1 \\ &T_1(x) = x \\ &\dots \\ &T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \end{aligned} \tag{10}$$

and those for second-kind polynomials –

$$\begin{aligned} &U_0(x) = 1 \\ &U_1(x) = 2x \\ &\dots \\ &U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x) \end{aligned} \tag{11}$$

In our study, we use Chebyshev polynomials of the first kind.
The first few polynomials of this recurrent series are:

$$\begin{aligned} &T_0(x) = 1 \\ &T_1(x) = x \\ &T_2(x) = 2x^2 - 1 \\ &T_3(x) = 4x^3 - 3x \\ &T_4(x) = 8x^4 - 8x^2 + 1 \\ &T_5(x) = 16x^5 - 20x^3 + 5x \\ &T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 \\ &T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x \\ &T_8(x) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ &T_9(x) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \end{aligned}$$

In the experiments we performed, for the distribution of the data (Chebyshev nodes) we use the function:

$$x = sin\Big(\frac{k\pi}{2n}\Big), \tag{12}$$

where n is the degree of the polynomial, and $k = -n : 2 : n$.

The major advantages of approximations with Chebyshev polynomials are:

1. **Evenness:** Chebyshev polynomials are formed through evenly scattered and distributed points in a given interval, which can lead to better results when approximating functions with a complex structure.
2. **Minimal errors at the interval boundaries:** Chebyshev polynomials have the property of minimizing the error at the boundaries of the interval. This means that the approximation with Chebyshev polynomials can be more accurate around the boundaries compared to other methods.

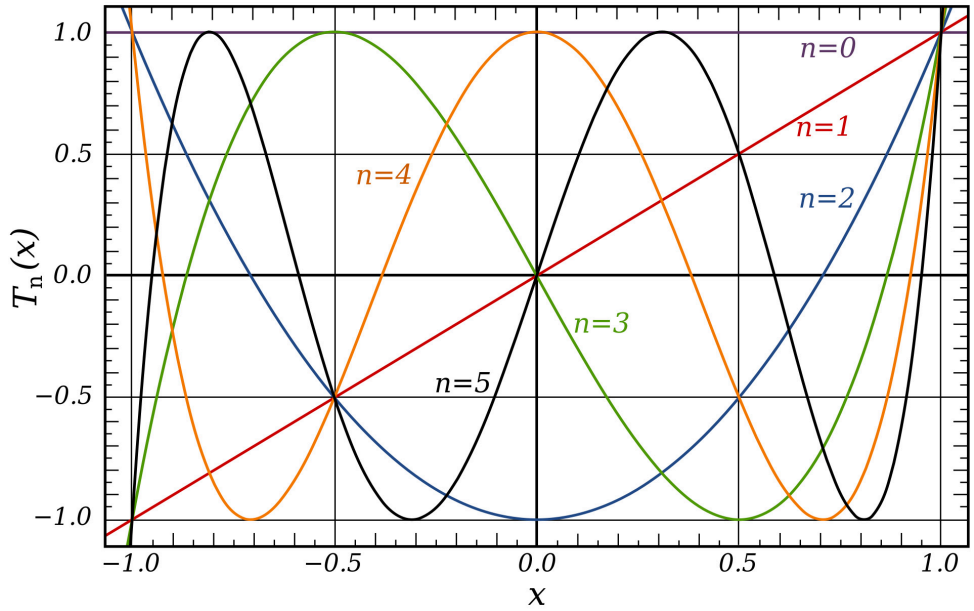


Figure 3: The first five Chebyshev polynomials of the first kind

3. **Stability:** Approximation with Chebyshev polynomials is robust to changes in data or the interval. This means that even if there is some variation in the input data, Chebyshev polynomials can provide a good approximation.

The main disadvantage of this approach worth noting is the computational complexity of the approximation – with Chebyshev polynomials, it can be high, especially with large degrees of the polynomials, which in turn requires more time and resources for the calculations. In addition, approximation with Chebyshev polynomials requires a preliminary setting of the interval and the number of points, which means that the method is not very adaptable and may face challenges when approximating functions with a variable structure or ones with more complex behavior.

2.5. Approximation with artificial neural networks

Artificial neural networks (ANNs) are computer models based on the structure and functioning of biological neural networks – a major part of the human brain [26]-[28]. The biological neuron is the basic unit of the nervous system in living organisms which has a key role in the transmission of bio-electrical signals and in-formation processing.

Anatomically and functionally, the biological neuron can be divided into several main components (Fig. 4):

1. **Soma (body):** The soma of a neuron is the body of the cell, it contains a nucleus and organelles necessary for the metabolic functions and the maintenance of cellular life.
2. **Dendrites:** Dendrites are elongated and branched structures – the input part of the neuron that is responsible for receiving signals and information from other neurons, receptors, glands or muscle groups.
3. **Axon:** The axon is a long structure, typically insulated with a myelin sheath that transmits electrical signals from the soma to other cells – the postsynaptic neurons. At the end of the axon are the axon terminals which connect to the dendrites of the next cell through structures called synapses.
4. **Synapses:** Synapses are anatomical structural formations through which the contact between neurons is made. They can be chemical, electrical or mixed. In chemical synapses, neurons communicate

through chemical substances called neurotransmitters, while electrical synapses allow direct transmission of electrical signals from one neuron to another.

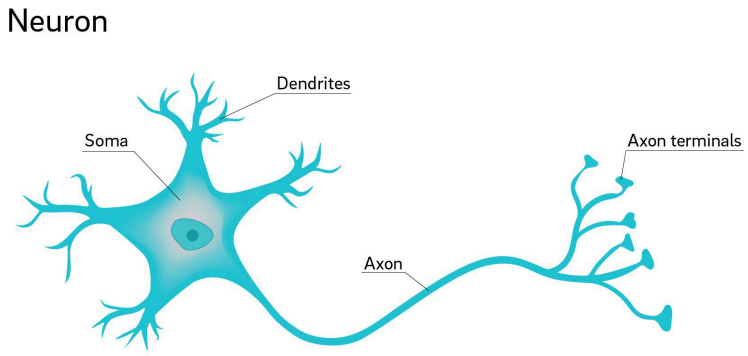


Figure 4: Structure of the biological neuron.

Adhering to the evolutionarily created model, the structure of the artificial neuron is presented schematically according to Fig. 5. Here:
- $\vec{X}(x_1, x_2, x_3, \ldots, x_n)$ is the input stimuli vector;
- $\vec{W}(w_1, w_2, \ldots, w_n)$ is the weight vector;
- b – threshold of the neuron;
- S – function modeling the summation of input stimuli;
- g – activation/transfer function modeling the output signal along the axon.

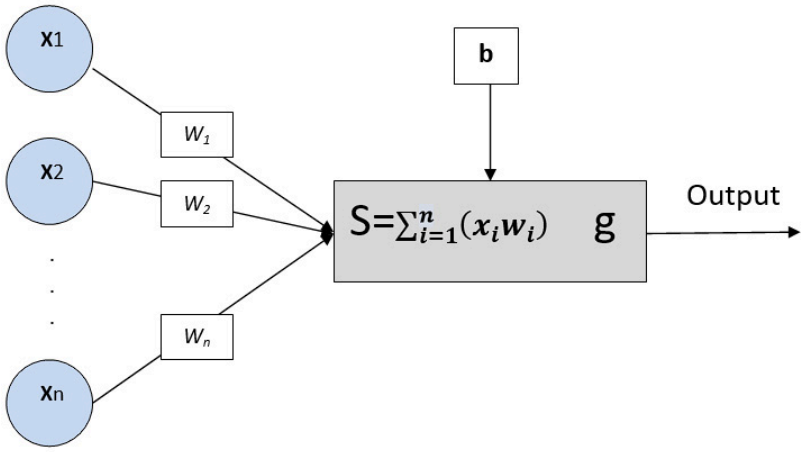


Figure 5: Scheme of an artificial neuron

Generally, artificial neural networks can be defined as computer models that are structured from multiple artificial neurons designed to model biological neural networks. They are characterized by:

1. **Layered structure:** ANNs consist of several layers of neurons that are arranged sequentially (Fig. 6). Typically, ANNs include an input layer, hidden layers and an output layer, where the input layer

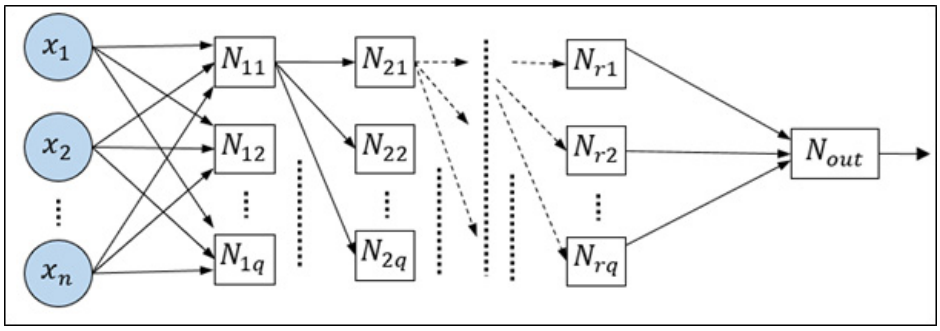


Figure 6: An artificial neural network with r-hidden layers, each of which has q neurons

receives the data from the environment external to the neuron, the hidden layers perform processing and pass the information forward, and the output layer generates the final result.

2. **Links and weights:** Neurons in the different layers of the ANN are connected through links that have associated weights. These weights determine the importance of input signals and control the influence that one neuron has on others.
3. **Learning process:** ANNs are trained by providing input data and adjusting the weights of the links between neurons. The training can be done in a variety of ways and is closely related to the type of transfer function used. During training, the ANN goes through an iterative error correction algorithm. Through the iterative process, the weights are changed to reduce the error at each subsequent cycle [29]-[31].

Based on their structure and principles of functionality, along with the training option, artificial neural networks become a powerful tool for processing information and solving complex tasks. They are capable of learning from data, discovering complex patterns and even making predictions. They are used for image recognition [32], speech recognition [33], natural language processing [34], prediction [35], robotics [36], medicine [37], finances [38] and other. In the present study, ANNs were used to approximate functions in parallel with polynomials, and a comparative analysis of the obtained results was carried out.

2.6. Other contemporary approaches to approximation of functions

There are numerous modern approaches that are used to approximate an unknown functional dependency, based on experimentally obtained data on the predictors and the target value. Some of the most commonly encountered methods in the literature include:

1. **Support Vector Machines (SVM):** SVM is a classification and regression method used to find an optimal hyperplane that partitions the data in space based on various features. SVM can be used to approximate functions and predict their future behavior. For example, Martin offers an online method for using SVM to approximate time series and numerical functions [39]. On the other hand, Hammer and Gersmann show that an SVM with a polynomial kernel of degree $p-1$ trained with a training set of size p can approximate p training points to any desired accuracy [40].
2. **Decision Trees and Random Forests:** These methods use tree-like structures to divide the data into sequential steps. Random forests are an ensemble of multiple decision trees that work together to make predictions more efficient. In their work, Balaskas and colleagues use the approximating capabilities of Decision Trees to solve problems related to technologies used in manufacturing printed circuits in electronics [41]. An example of how to design decision rules using Decision Trees and approximations is presented in [42].
3. **Genetic Algorithms:** Genetic algorithms are used to solve optimization problems, offering an alternative approach to finding optimal solutions in the problem space by creating, mutating, and combining genetic codes that represent potential solutions. In function approximation, genetic algorithms can find approximate solutions to functional problems, where the population of solutions is represented

by genetic codes that are transformed and optimized through evolutionary operators such as selection and mutation. The process iteratively evolves over several generations, aiming to find increasingly efficient solutions. Examples of the use of genetic approximation algorithms can be seen in [43]-[44].

In the present study, artificial neural networks are used to conduct a meaningful comparative analysis of their approximation efficiency relative to that of polynomial approximations.

3. Comparative analysis of the efficiency of polynomial approximations and artificial neural networks in function approximation

Approximation of functions is of great importance in various fields of science, engineering, business and sometimes even in public life. It allows complex functions to be represented with simpler models that can be analyzed and used more easily, to make predictions, extract information and find solutions that would otherwise be difficult or even impossible. There are many different approaches to approximation – polynomials, splines, regression models and others, and it should be noted that each method has its advantages and disadvantages in terms of accuracy, computational complexity or applicability in specific cases. This means that choosing the most suitable approximation method is essential when we have to decide which one to use depending on the characteristics of the data, the presence of noise, and the desired accuracy. That is why conducting a comparative analysis between different approximation methods is extremely important. Such an analysis helps us compare the effectiveness, accuracy, and applicability of the different methods in specific scenarios. Through the comparative analysis, we can select the most suitable method for our needs and achieve optimal results in the approximation of functions.

In the present study, a comparative analysis was conducted between some classical polynomial approximations and artificial neural networks, and for the purposes of the analysis, applying the same computer configuration, the following criteria for the effectiveness of the respective approach were used:

1. **Accuracy in interpolation:** How accurately the method approximates the real function or the data it is trying to approximate. To estimate the in-sample precision, we use mean absolute error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |\overline{y_i} - y_i|, \tag{13}$$

where $\overline{y_i}$ are the values interpolated by the respective method, y_i are the real values, and n is the volume of the sample.
Unlike the research of other authors, which focused on comparing polynomial and neural network methods, we have set several additional new tasks for ourselves, namely, to compare the following indicators:
2. **Speed:** How fast the respective algorithm performs the approximation. In general, speed can be measured by indicating the execution time or the number of steps that need to be completed. In this study, we have chosen to consider the execution time.
3. **Memory and computer efficiency:** Requiring a large amount of memory or computing power can also limit the effectiveness of the chosen approximation method, especially when dealing with larger or more complex data. Therefore, as an additional criterion considering the efficacy of an approach, we have also added the efficiency when using the memory or computing capabilities of the computer.
4. **Predicting near and away from the boundaries of the given sample:** The data and functions we want to approximate can often have significant variations both near the sample boundaries and far beyond. This means that if the approximation method is not good in one of these two areas, the results may be inaccurate or not applicable in real conditions. Thus, if an approximation method is only good near the boundaries of the sample, its usefulness and reliability decrease when applied to a prediction or approximation far beyond those boundaries.

5. **Scalability:** The ability of the selected approximation method to handle large data sets or complex functions. It is important to verify whether the algorithm can be applied efficiently on different data sizes or function complexity. In our case, we check scalability through re-considering criteria (1)-(4) with an increased sample size.
6. **Impact of noise on the results:** To what extent the compared methods are able to handle the noise in the input data?

3.1. *Comparative analysis of Lagrange polynomial approximation and artificial neural networks*

To make a comparative analysis of the efficiency in approximation of Lagrange polynomials and artificial neural networks, we used different types of functions. The process of experimentation covered several different, relatively long stages, and the results in all cases were uniform. Therefore, in the present study, the analysis described when comparing an approximation done by the Lagrange polynomial and by artificial neural networks, one of the most representative examples is chosen – the approximation of the function:

$$y = \log(x) \tag{14}$$

3.1.1. *Small number of nodes* ($n = 10$)

The sample we used at this stage is as follows:

$$D = \{x_i = 1 : 10,\ y_i = \log(x_i)\}; \tag{15}$$

The interpolating Lagrange polynomial was implemented with the Matlab script shown below.

```
% Sample
 x =1:10; y = log(x);
% Lagrangeinterpolationpolynomial
symst;
L = 0;
fori = 1:length(x)
% Lagrangepolynomialformula
    Li = 1;
for j = 1:length(x)
if j ~= i
            Li = Li * (t - x(j)) / (x(i) - x(j));
end
end
    L = L + Li * y(i);
end
% Calculatingthepolynomialerror
error = subs(L, t, x) - y;
mean_error = mean(abs(error));
% Algorithmspeed
tic
interpolated_y = sum(y .* L);
execution_time = toc;
```

Once the input-output sample vectors (x, y) are defined, the variable t for the implementation of the Lagrange polynomial L is also declared. After the creation of L , all differences (**errors**) between the values of the polynomial and the objective function in the sample nodes are reported. Subsequently, the mean absolute error during interpolation (**mean_error**) is formed. During the implementation of the code, the measurement of the time of completion (speed) of the algorithm is also started (**tic**).

To specify a suitable neural network, we use the following formula for the number of neurons in one hidden layer [45]-[46]:

$$q \leq \frac{m-1}{n+2} \tag{16}$$

where m is the number of training samples and n is the number of input stimuli. At this sample size, the neural network we use consists of one hidden layer that contains 3 neurons. All training samples from D were divided into three groups: 70% (6 samples) – training; 15% (2) – validation; 15% (2) – network test. The hidden neurons in the network are used for transfer function

$$TanH(x) = \frac{2}{1+e^{-2x}} - 1 \tag{17}$$

and the training was implemented with the Levenberg-Marquardt algorithm.

```
% preparing the data for the neural network
input = x;output = y;
% Creating a neural network
net=fitnet(3);
net.divideParam.trainRatio = 75/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;
net = train(net, input, output);% Training
% Interpolation with the neural network
interpolated_y_nn = net(input);
% Calculation of the mean absolute error (MAE)
mae_nn = mean(abs(interpolated_y_nn - output));
```

The comparison between this neural network with the Lagrange polynomial constructed for the sample showed the following results (Fig. 7).

From the interpolation graph it is immediately noticeable how strongly expressed Runge's phenomenon is, for the Lagrange polynomial – outside the boundaries of the sample, it sharply begins to deviate from the real function, something that the neural network copes with. However, it can be said that under these conditions of a small-size sample and a neural network that has distributed the insufficient training samples into three groups, an ANN cannot do very well and the advantage is in favor of the Lagrange polynomial. The polynomial interpolation (within the sample bounds) is better, the time for executing the approximation with the polynomial is shorter, and the error (MAE) is also smaller even when predicting outside the sample but near its boundaries. Only when predicting far outside the sample boundaries, the absolute error of the neural network is negligible compared to that of the polynomial, a fact that is a consequence of the strong dependence of Lagrange polynomials on Runge's phenomenon.

Table 1 describes the numerical results of the comparison between the two methods.

Method	Approximation execution time (seconds)	Interpolation error	Close prediction error (at x=11)	Distant prediction error (at x=20)
Lagrange polynomial	0.0064	0	0.0354	1.829e+3
Artificial neural network	0.059	0.0387	0.0696	0.0501

Table 1: Comparison between Lagrange polynomial and an artificial neural network with three neurons in the hidden layer and sampling distribution training/test/validation (70%/15%/15%)

It is noticeable that, unlike the polynomial whose mathematical model obliges it to pass correctly through all nodes of the sample since it is distributed into training samples, test, and validation samples, the neural

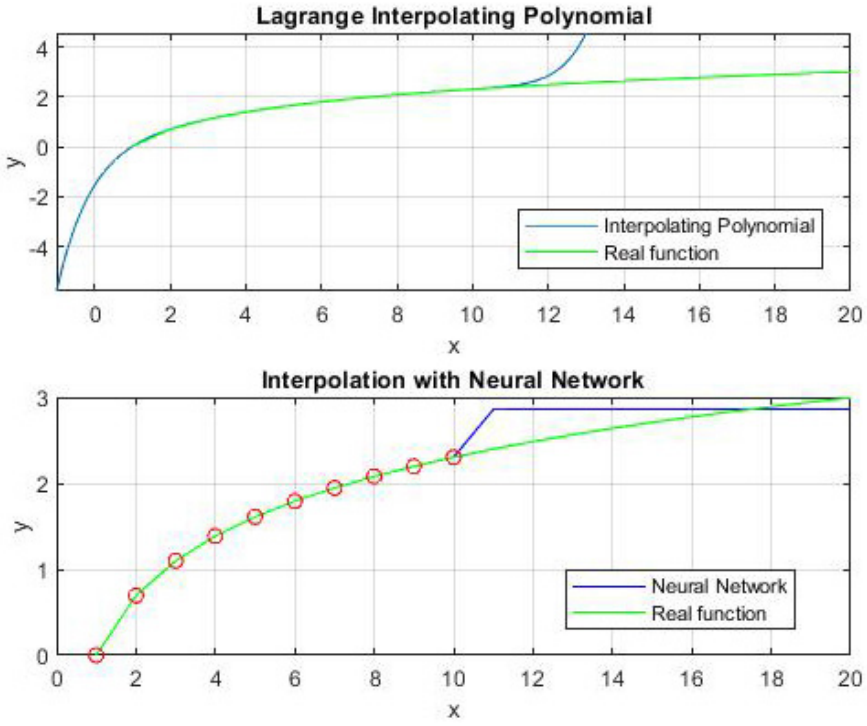


Figure 7: Comparison of the Lagrange polynomial with three-neuron artificial neural network and distributed samples for training, test and validation (70%; 15%; 15%).

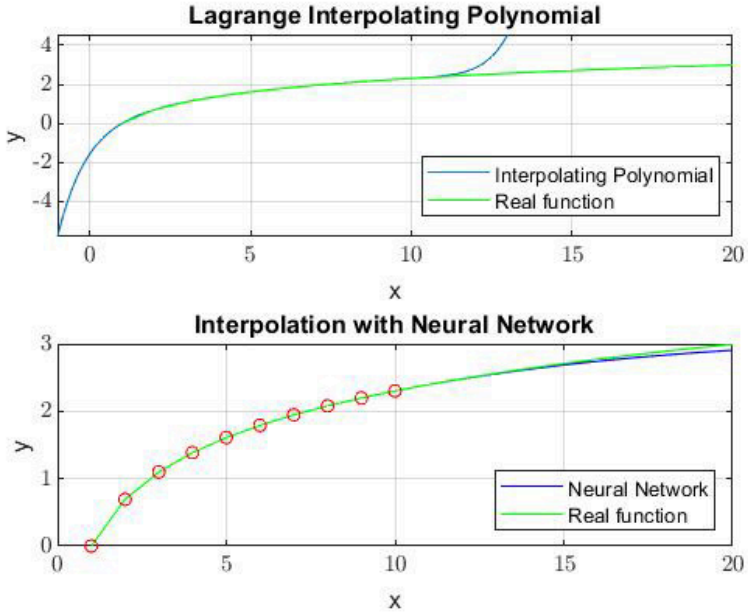


Figure 8: Comparison of Lagrange polynomial with an artificial neural network with three neurons and distributed samples for training, test and validation (100%; 0%; 0%).

Method	Approximation execution time (seconds)	Interpolation error	Close prediction error (at x=11)	Distant prediction error (at x=20)
Lagrange polynomial	0.0064	0	0.0354	1.829e+3
Artificial neural network	0.1261	5.0592e-5	8.7503e-3	0.508

Table 2: Comparison between Lagrange polynomial and an artificial neural network with three neurons in the hidden layer and sampling distribution training/test/validation (100%/0%/0%).

Method	Approximation execution time (seconds)	Interpolation error	Close prediction error (at x=11)	Distant prediction error (at x=20)
Lagrange polynomial	0.6634	0	$3.7211e+12$	$1.5536e+25$
Artificial neural network	0.2621	0.033	0.0055	0.0042

Table 3: Comparison between Lagrange polynomial and artificial neural network with 33 neurons in the hidden layer and sample volume $n = 100$

network fails to pass successfully and precisely through these same nodes. However, if we force the neural network to use all the samples only for training (100%/0%/0%), things change in several aspects (Fig. 8).

The change in the neural network's ability to interpolate the given sample nodes is immediately noticeable. Even though it cannot be compared to the zero error of the polynomial at these nodes, the artificial network error has become much smaller than in the previous case. The increased ability of the network to make predictions near the end of the sample is also impressive – now its error is smaller than that of the Lagrange polynomial. There is also a slight degradation of the neural network's ability to predict far beyond the sample boundaries, which is explained by the fact that there are no separate samples for validation and independent tests during the execution of the training algorithm.

Even so, the artificial neural network still predicts much more successfully than the polynomial in these distant regions (Table 2).

3.1.2. Increased number of nodes ($n = 100$)

As the sample size increases, the performance of the Lagrange polynomial begins to lag behind that of the ANN. The increased number of nodes leads to an increase in the degree of the polynomial, which is related to the used computing power and memory, the implementation of the algorithm slows down, and the errors, compared to those of the ANN, become larger and larger (Fig. 9).

In the case of 100 training samples and one input neuron, according to formula (1) we used a neural network with 33 neurons in the hidden layer. In the experiment, the increasingly unstable behavior of the polynomial at its boundaries is immediately noticeable as the sample volume increases. In contrast, the neural network approximation maintains stability, and it is clear from the look of the graph that its prediction errors will be much smaller. The blue graph of the approximation made by the neural network almost completely coincides with the green one – that of the real function. The numerical values of the comparative analysis made under these circumstances can be viewed in Table 3.

Due to the principles on which the mathematical model of the Lagrange polynomial is built, it is forced to pass through all nodes of the sample, and this is its only advantage over the artificial neural network. The performance of the neural network in the approximation is better than that of the polynomial, starting from the execution time of the algorithm and reaching the prediction close and far from the sample boundaries. Here we have only described the comparative analysis between the Lagrange polynomial and an artificial neural network. In the course of the study, similar comparisons were sought for different types of functions:

$$Linear: \ y = \sum_{i=1}^{n} a_i x_i \tag{18}$$

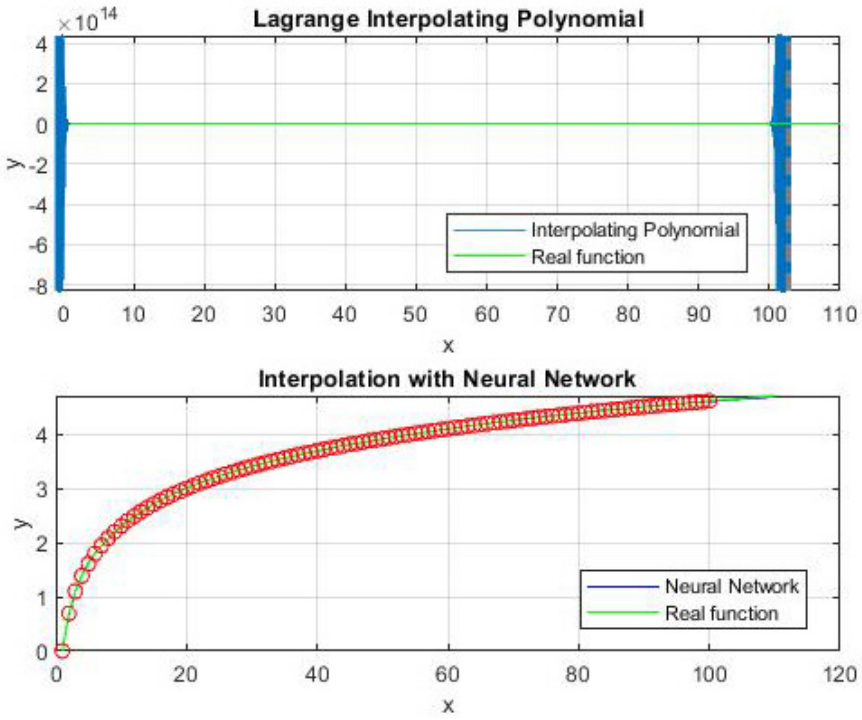


Figure 9: Comparison of a Lagrange polynomial with an artificial neural network with three neurons and sample size $n = 100$

$$Power: \ y = \sum_{i=1}^{n} a_i x_i^i \tag{19}$$

$$Trigonometric: \ y = \sum_{i=1}^{n} a_i \sin x_i, \ y = \sum_{i=1}^{n} a_i \cos x_i \tag{20}$$

$$y = \sum_{i=1}^{n} a_i tg x_i, \ y = \sum_{i=1}^{n} a_i cotg x_i \tag{21}$$

$$Logarithmic: \ y = \sum_{i=1}^{n} a_i \log x_i \tag{22}$$

$$Exponential: \ y = \sum_{i=1}^{n} a_i e^{x_i} \tag{23}$$

3.1.3. Conclusions from the comparative analysis of function approximation with Lagrange polynomials and with artificial neural networks

The comparison of the approximation capabilities of the Lagrange polynomial and the artificial neural networks leads to several important conclusions:

1. **Interpolation:** Due to the mathematical principles on which the Lagrange polynomial is built, it always passes through the sample nodes, making it more convenient when we are only looking for a small interpolation error. However, it should be noted that with an appropriate choice of transfer functions and type of training, artificial neural networks have relatively small errors.

2. **Speed:** For small samples, the speed of the function approximation algorithm using the Lagrange polynomial is higher than that of the artificial neural network. However, with an increased sample volume, due to an increase in the degree of the polynomial, the computational resources increase and so does the execution time. Thus, for larger volumes of data, the artificial neural network has apparent superiority in terms of approximation speed.
3. **Memory and computing resources. Scalability:** As with the speed of the algorithm, it should be noted here that an increase in the volume of samples is accompanied by an increase in used memory and computational resources, which is especially noticeable in the case of Lagrange polynomials. When we add this fact to the speed decrease with large samples as well as Runge's phenomenon, it becomes clear that Lagrange polynomials, as an approximation algorithm, have poorer scalability than artificial neural networks whose behavior remains stable under the appropriate form of training.
4. **Prediction:** In the general case, Lagrange polynomials deteriorate sharply in their predictive qualities beyond the boundaries of the samples (Runge). In both near-range and farther-range predictions, artificial neural networks have an apparent advantage.

3.2. Comparison of the efficiency of approximation using Newton's polynomials and artificial neural networks.

Experimenting with Newton's polynomials and artificial neural networks was similar to comparing La-grange polynomials with artificial neurons. The results are quite similar. Newton's polynomials can also be affected by Runge's phenomenon when approximating functions. Large oscillations and "warps" of the approximating polynomial near the boundaries of the interpolation interval were observed. This means that when used for prediction, polynomials expand beyond the boundaries of the samples with large deviations from the real functions. Furthermore, since Newton's polynomial is expressed in terms of basic polynomials associated with the interpolation points, and since when interpolating samples of volume n, each interpolation point is used as a node of the polynomial and provides its own basic polynomial when all basic polynomials are combined, we obtain Newton's polynomial of degree n-1 that passes through all interpolation points. This high degree of Newton's polynomial in large samples, just like with the Lagrange polynomial, is accompanied by an increase in the speed of the approximation and the required computing resources, which means very poor scalability of the method. After experimenting with many different functions, the results we obtained point to the superiority of ANNs in most of the selected criteria for approximation efficiency.

3.3. Comparison of the efficiency of approximation using Hermite polynomials and artificial neural networks

Experimenting with a Hermite polynomial was a bit different from that associated with the Lagrange and Newton polynomials. The differences arose from the application of the Hermite polynomial, which proves useful in the following situations:

1. **Interpolation with smooth curves:** The Hermite polynomial can be used to interpolate data when we want the resulting curve to be smooth and without sharp changes. This is achieved through the inclusion of information about the derivatives of the function in the interpolation process.
2. **Approximation of complex functions:** The Hermite polynomial can be used to approximate functions that are complex and difficult to represent with simple analytical formulas. By including information about the derivatives of the function, we can approximate its behavior to the real function at various points.
3. **Solving optimization tasks:** The Hermite polynomial is also used in optimization tasks where we want to find the function's extrema. The use of information on the derivatives helps us trace the behavior and identify critical points where the respective extrema can occur.

In the present study, we conducted a number of experiments related to each of these trends in the application of Hermite polynomials. Here we will present some of the examples that were used to compare the approximation of these polynomials with artificial neural networks.

Often, the Hermite polynomial interpolation uses data on both the behavior of the objective function that is known up to a point and its speed and acceleration. One example is the following. *Observations were made of the temperature (in degrees Celsius), its rate of change, and the acceleration with which it was changing (Table 4).*

Moment in time t	Temperature $T = T(t)$	Temperature speed dT/dt	Temperature acceleration dT^2/dt^2
0	10.0	0.0	1.00
2	9.5	2.50	2.50
4	8	7.0	2.75
6	6.5	9.5	1.5
8	5	9.5	0.5

Table 4: Measurements of temperature, speed, and acceleration as it changes

The task is to build a Hermite polynomial that approximates the temperature.
The main part of the script with which the task was accomplished consists of building the polynomial itself.

```
% A sample of temperature, speed and acceleration values
T = [0.0, 2.0, 4.0, 6.0, 8.0];
temperature = [10.0, 9.5, 8.0, 6.5, 5.0];
dT_dt = [0.00, 2.50, 7.00, 9.50, 9.50];
d2T_dt2 = [1.00, 2.50, 2.75, 1.50, 0.50];
% Calculation of the Hermite polynomial
symsx;
H = hermite_interpolation_rate(T, temperature, dT_dt, d2T_dt2, x);
```

The function **hermite_interpolation_rate** building the polynomial model itself is as follows.

```
function H = hermite_interpolation_skorost(T, temperature, dT_dt, d2T_dt2, x)
    n = length(T);
    H = sym(0); % Zero Hermite polynomial
for i = 1:n
term = temperature(i);
for j = 1:n
if j ~= i
term = term * ((x - T(j)) / (T(i) - T(j)));
end
end
        H = H + term * (1 - 2 * (x - T(i)) * diff(dT_dt(i), x)) + term * (x - T(i))
* (x - T(i)) * d2T_dt2(i);
end
end
```

To build the Hermite polynomial, both the temperature values and the parameters of its change (such as speed and acceleration) are used. The degree of the constructed polynomial is 9 ($2n - 1$, where n is the sample volume), and part of its graph is shown in Fig. 10.a.

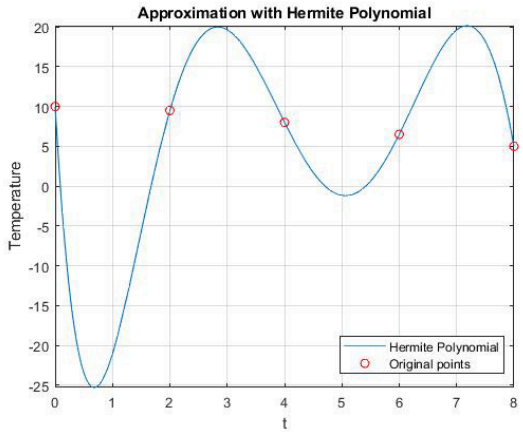
The mean absolute error of the Hermite polynomial interpolation is:

$$MAE = 0$$

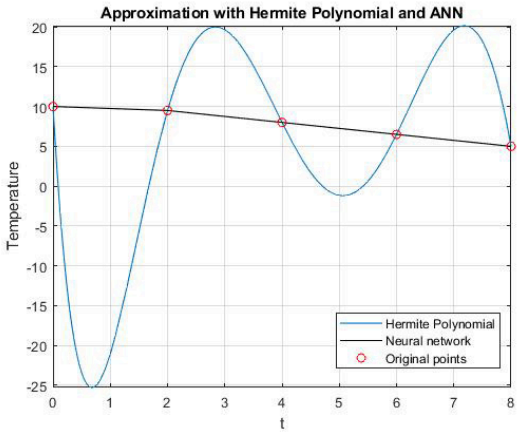
To compare the interpolation capability of this polynomial with an artificial neural network, a single-layer network is built, with 10 neurons in the hidden layer. The transfer function in the bodies of the hidden neurons was a hyperbolic tangent and the network was trained with the Levenberg-Marquardt algorithm. It should be noted that due to the small amount of data in the sample, training was performed using all samples. Since the goal is to approximate the behavior of the temperature, the moment in time, speed,

and acceleration of change in temperature fluctuations were used as training input data. The results of the interpolation of the network show the mean absolute error in the sampled nodes (Fig. 10.b):

$$MAE = 3.65E - 15$$



(a) Figure 10a. Approximation of the sample from Table 6 with a Hermite polynomial.



(b) Figure 10b. Approximation of the sample from Table 6 with a Hermite polynomial and an ANN.

It must be admitted that the error of the artificial neural network during interpolation of the sample is larger than that of the polynomial, yet it should be noted that as a representative of artificial intelligence, the neural network was trained using 5 samples only. When trying to increase the sample volume, an increased delay was noticed in the performance of the Hermite polynomial due to an increase in degree, which in turn is associated with additional difficulties in the calculation algorithms.

For sets consisting of several hundred samples, in addition to the large delay, the interpolation errors started to become comparable to those of artificial neural structures.

For a more in-depth comparative analysis, we again used samples of different sizes to approximate different types of functions: linear, power functions, logarithmic, exponential, trigonometric and mixed.

We will summarize the results of all conducted experiments with different functions into two groups:

1. **Experimentation with small volume samples** ($n \leq 100$)**.** For these small data samples, the Hermite polynomial definitely outperforms the ANN approximation. The method is faster, and the interpolation error is zero. Although this type of polynomials also suffers from the effect caused by Runge's phenomenon, the model they produce for small samples is excellently adapted for prediction both near and far from the boundaries of the samples.
2. **For large volume samples** ($n \geq 500$)**.** As the sample size increases, the approximation qualities of the polynomial deteriorate and the efficiency of the neural network improves. For samples with volume $n > 100$, deviations from the decent indicators for the polynomials gradually begin. The gradual increase of the sample is first reflected in the implementation time of the algorithm, followed by deterioration in the near and far prediction. For samples with volume $n \geq 500$, the polynomial errors observed in the interpolation nodes are larger than those of artificial neurons.

3.3.1. Conclusions from the comparative analysis of approximation of functions using Hermite polynomials and artificial neural networks.

The comparative analysis of function approximation using Hermite polynomials and artificial neural networks leads to several important conclusions:

.

1. **Interpolation:** For small samples, the Hermite polynomial always passes through the sample nodes with zero error. Just like the other types of polynomials, this makes it more convenient when we are only looking for a small error in the interpolation of the given data sample.
2. **Speed:** For small samples, the speed of the algorithms for approximation of functions with the Hermite polynomial is higher than that of the artificial neural network. However, with an increased sample size, due to an increase in the polynomial degree, the computational resources increase and so does the execution time. Thus, with larger data, the artificial neural network has superiority in terms of approximation speed.
3. **Memory and computing resources. Scalability:** An increase in the volume of samples is accompanied by an increase in the used memory and computer resources, which is particularly noticeable for all polynomials, including the Hermite one. It should be noted that as an approximation algorithm, Hermite polynomials have a poorer scalability than artificial neural networks whose behavior remains stable under the appropriate form of training.
4. **Predicting:** For small sample volumes, Hermite polynomials have better predictive qualities than neural networks, both near and far from the sample boundary. This is somewhat related both to the low degree of the polynomial (resp. leading to smaller errors) and to the lack of sufficient samples to properly train the neural structures. However, with larger volumes of data samples trained with larger amounts of specimens and competing with a slow, imprecise polynomial of a very high degree, which is also affected by Runge's phenomenon, especially with highly oscillating objective functions, neural networks outperform in terms of prediction.

3.4. Comparison of the efficiency of approximation using Chebyshev polynomials and artificial neural networks

Chebyshev polynomials are used in several different situations. Below are some of the tasks in which its use is preferred:

1. **Approximation of functions with large variations in input data:** The Chebyshev polynomial can be effective when approximating functions that have large variance or contain different modes of behavior. This includes tasks such as signal approximation, financial data, kinetic models, and more.
2. **Minimizing the maximum approximation error:** The Chebyshev polynomial has the ability to minimize the maximum approximation error in a given interval. It is therefore used in tasks where it is important to ensure that the approximation error is as small as possible over the entire interval.
3. **Solving tasks having samples with randomly distributed data:** The Chebyshev polynomial is useful when approximating functions that have randomly distributed data or when robustness of the approximation to data deviations is required.
4. **Approximation of nonlinear functions:** The Chebyshev polynomial can be used to approximate non-linear functions with a more complex structure, where standard polynomials are not suitable or have a high error.

In the present study, a comparative analysis was carried out of the approximation capabilities of Chebyshev polynomials and artificial neural networks, both in the indicated areas and for the approximation of individual functions of different characters.

The example that we will point out here is related to the approximation of the non-linear function with a more complex character, for which the other studied polynomials are associated with large errors:

$$f(x) = 0,3x\cos(1,3\pi x + 0,3) + \frac{1}{1+e^{-2x}} \tag{24}$$

An approximation of the function with a Chebyshev polynomial and artificial neural networks in the segment $[-1,1]$ was tracked.

First-order Chebyshev polynomials and different degrees (3,5,6,12 and 20) were used in the experiments.

```
clear; clc;
%Polynomial order
order = 20; %Low = 3, Med = 5, Med_1=6 high_1 = 12 high_2=20;
%Points used
nData = 5000;
%Region
d = [-1,1];
%Target function
testFun = @(x) 0.3*x.*cos(1.3*pi*x + 0.3) + 1./(1+exp(-2*x));
noiseAmpl = 0.05;% Noise value
t = linspace(d(1),d(2),nData); % arguments of the objective function
% functional values of the objective function
y = testFun(t);
ySample = y + noiseAmpl*randn(size(y));
%Chebyshev nodes
chebPts = chebyshevPoints(order+1,d);
%Approximation
f = chebFitLs(t,ySample,d,order);
yFit = chebEval(f,t,d);
```

The **chebyshevPoints** function is used to create the Chebyshev points, and the nodes are built using the function:

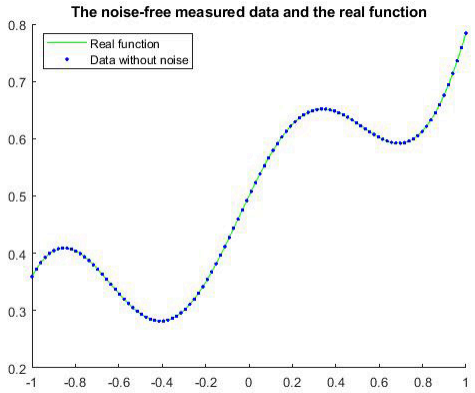
$$x = \sin\left(\frac{k\pi}{2n}\right) \tag{25}$$

where n is the polynomial degree, and $k = -n : 2 : n$.

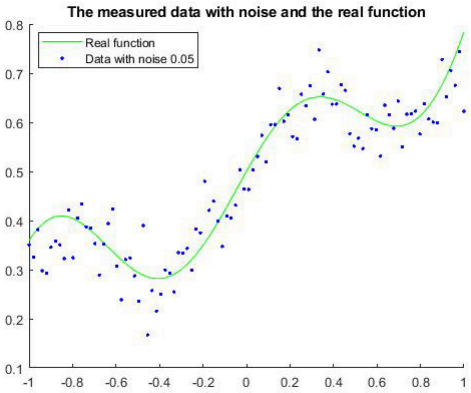
```
function [x, w] = chebyshevPoints(n,d)

if n == 1, x = 0; return, end% Special case at n=1, x=0
% Calculation of Chebyshev points in the segment[-1,1]:
m = n-1;
x = sin(pi*(-(m+1)):2:m+1)/(2*m));
ifnargin~=1
    x = (diff(d)*x + sum(d))/2;
end
ifnargout==2
    w = weights(n)*diff(d)/2;
end
end
function w = weights(n)
if n == 1
    w = 2;
else

    n = n-1;
    u0 = 1/(n^2-1+mod(n,2));
    L = 0:n-1; r = 2./(1-4*min(L,n-L).^2);
    w = [ifft(r-u0) u0];
end
end
```

As for the number of neurons in the hidden layers of the artificial neural networks – it was determined depending on the number and size of the input-output training samples whose training was carried out using the Levenberg-Marquardt algorithm, and a hyperbolic tangent was used for the transfer function in the hidden layers.

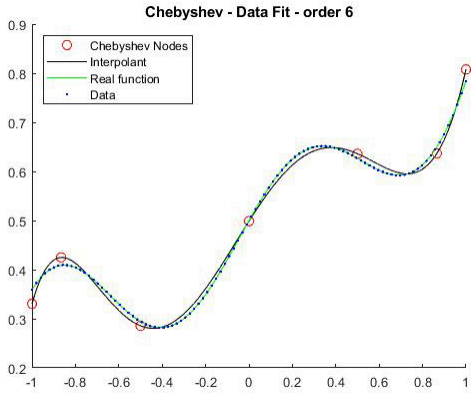
The experiments carried out included checking the stability of the two approximation approaches in the presence of noise α in the data (e.g. random errors in data collection or measurement) leading to the wrong



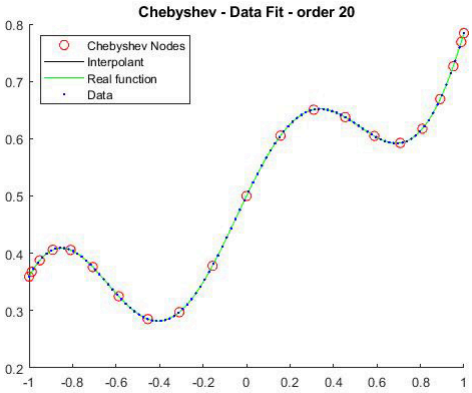
(a) Figure 11a. Data measured with no noise and the real function.



(b) Figure 11b. Example of data measured with noise ($\alpha = 0.05$) and the real function.



(a) Figure 12a.Comparison of interpolation using ANN and Chebyshev polynomial of degree 6.



(b) Figure 12b. Comparison of interpolation using ANN and Chebyshev polynomial of degree 20.

idea about the objective function (24):

$$f(x) = f(x) + \alpha p, \tag{26}$$

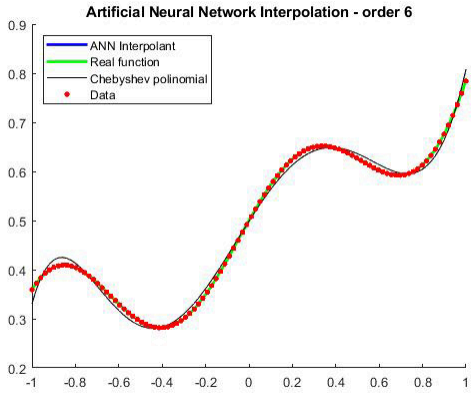
where p is a random number in the interval $[0, m]$,m-the sample volume.

The noise values experimented with are $\alpha = 0$; 0.001; 0.005; 0, 01 and 0.05, and the sample volume is $m = 100$.

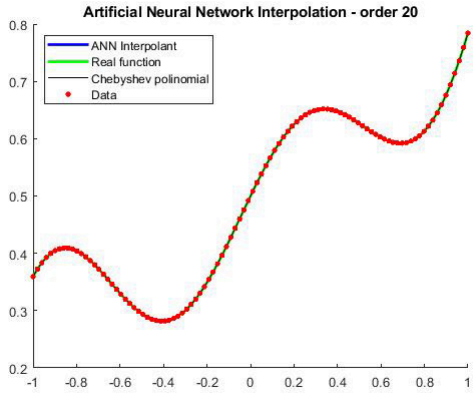
3.4.1. Comparative analysis of the approximation in the absence of noise ($\alpha = 0$).

In the absence of noise in the observed data that provides the sample, the higher the Chebyshev polynomial degree, the better it performs (Fig. 12).

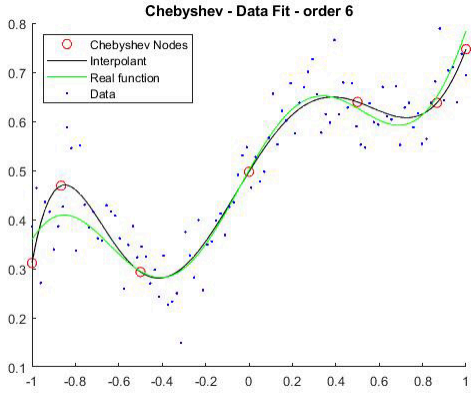
At a higher degree of the polynomial (n), the number of Chebyshev nodes (3) is larger, due to the larger range of the interval for $k = -n : 2 : n$. The respective mean absolute interpolation error also decreases, and from 0.0086 for a polynomial of degree 6, becomes 4.22e-8 – for a polynomial of degree 20. However, it is interesting to note that there is almost no change in the abilities of the two polynomials to predict correctly near and far from the sample boundaries. For both degrees of the polynomial, the absolute error when predicting the functional target value at a distance of 0.1 from the sample boundary is about 0.14, and for a distant prediction (at 40 units beyond the sample boundary) the error is about 11.68.



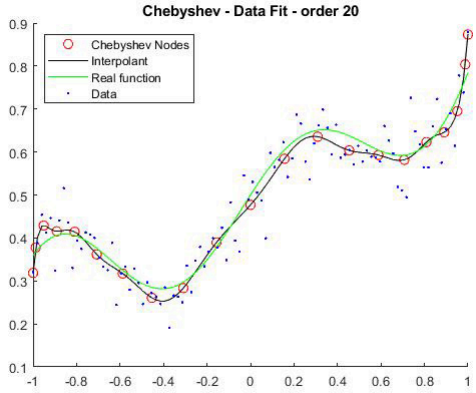
(a) Figure 13a. Comparison of interpolation using ANN and Chebyshev polynomial of degree 6.



(b) Figure 13b.Comparison of interpolation using ANN and Chebyshev polynomial of degree 20.



(a) Figure 14a. Interpolation of the objective function with a Chebyshev polynomial of degree 6.



(b) Figure 14b. Interpolation of the objective function with a Chebyshev polynomial of degree 20.

For the ANN, a network with 33 neurons in the hidden layer is selected. The number of neurons is chosen based on the number of training samples ($n = 100$) and the number of input neurons, according to [38]. The comparison checking the network's ability to approximate the sample was performed with Chebyshev polynomials of different degrees. Fig. 13 shows the graph for data interpolation using polynomials of 6^th and 20^th degree and the artificial neural network.

The mean absolute error of the neural network with interpolation is 2.74e-6 and its graph has almost merged with the graph of the objective function, whereas the graph of the polynomial (in black) deviates. The network error is smaller than the error of the degree-6 polynomial but greater than the Chebyshev degree-20 polynomial. The network prediction near and far from the sample boundaries at the same points – pt. $x = 1.1$ and pt.$x = 40$ the errors are respectively: 0.11 and 11.64 – almost matching those of the degree-20 polynomial.

3.4.2. Comparative analysis of the approximation in the presence of noise ($\alpha = 0.005$)

In this case, experiments show that in the presence of noise, the Chebyshev polynomial of higher degree does a slightly better job of interpolating the objective function than the one of lower degree (Fig. 14).

The mean interpolation error (MAE) of the polynomial of degree 6 is 0.0149, whereas that of the polynomial of order 20 has increased from 4.22e-8 (in the absence of noise) to 0.0136, in the presence of noise $\alpha = 0.005$. **To summarize: the presence of noise in the input data leads to an increase in the errors of**

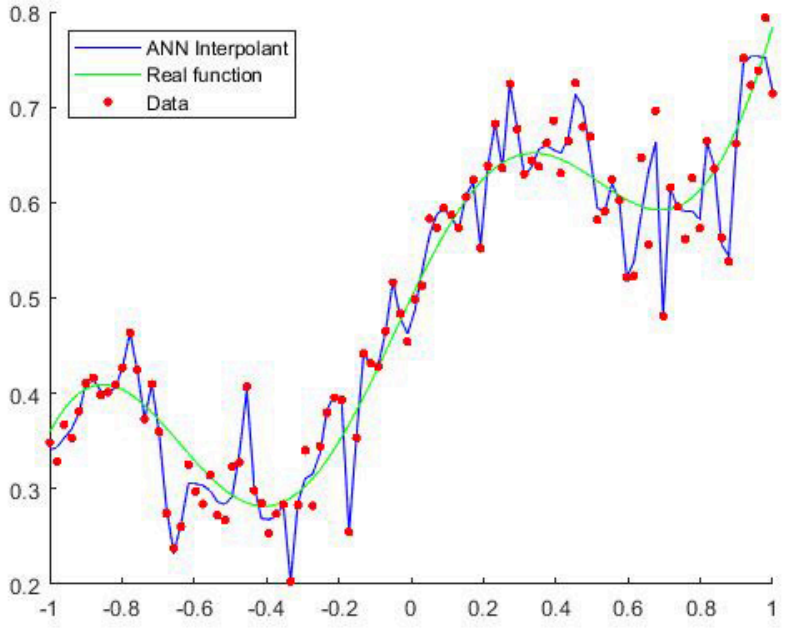


Figure 15: Interpolation of the sample with ANN in the presence of noise $\alpha = 0.005$.

the Chebyshev polynomials in the interpolation. Regarding prediction, it should be noted that with this level of noise, the absolute errors of the polynomial of degree 6 for near and far prediction are respectively: 0.1798 and 11.72. For the polynomial of degree 20 these values are: 0.1293 and 11.67, i.e. very slightly better predictive ability.

However, it is interesting what happens to the approximating properties of the neural network. It turns out that the presence of noise significantly worsens its interpolation and prediction (Fig. 15). Unlike the Chebyshev polynomial for which the Chebyshev nodes are key, the artificial neural network seeks to traverse all the sample data, which, however, is associated with errors in the measurements or the collection of data itself, with noise $\alpha = 0.005$. The network interpolation error is larger than that of the two polynomials:

$$MAE = 0.0318.$$

The absolute errors of the artificial neural network in near and far prediction (for $x = 1.1$ and $x = 40$ respectively) are respectively: 2.5424 and 14.0798. In both cases, the neural network gives poorer predictions than the Chebyshev polynomials. This means that with incorrectly measured data, errors in data collection, or other circumstances that worsen the input reliability, the artificial neural network performs significantly worse than Chebyshev polynomials. Similar experiments were again done with different types of objective functions: linear, power functions, logarithmic, exponential, trigonometric, and mixed. The results were similar. In the absence of noise, neural networks are comparable to high-degree Chebyshev polynomials in their accuracy for interpolation and prediction. The presence of noise, however, turn artificial neural structures into a more unreliable approximation mechanism. Experiments have shown that these conclusions hold as the sample size increases.

Method	Interpolation error	Close prediction error (at $x = 1.1$)	Distant prediction error (at $x = 40$)
Chebyshev polynomial of order 20	0.0022	0.1413	11.6787
Artificial neural network	0.0036	0.1364	11.6738

Table 5: Chebyshev polynomials and artificial neural network interpolation and prediction errors for a sample size of 5000

3.4.3. Conclusions from the comparative analysis of approximation of functions using Chebyshev polynomials and artificial neural networks

The comparative analysis of function approximation using first-order Chebyshev polynomials and artificial neural networks leads to the following conclusions:

1. **Interpolation:** The experiments done indicate that in the absence of noise, the sample interpolation errors of both the Chebyshev polynomials and artificial neural networks are almost the same. However, when the data is accompanied by a certain amount of noise, it becomes clear that neural networks become an unreliable tool for interpolating the real function, as they try to track all nodes, including the noisy ones. In contrast, the Chebyshev polynomial, relying primarily on Chebyshev nodes, performs significantly better.
2. **Speed, memory, and computing resources:** The speed, memory occupied and computational resources needed for approximating functions using either of the two methods are comparable, both for small samples and for a larger volume of data.
3. **Scalability:** As the sample size increases, a minimal increase in errors related to interpolation and prediction is observed in both approaches. It is interesting to note that as the number of input-output training samples increases, the errors hardly change, with a slight advantage of neural networks in predictions. Thus, for a sample of 5000 specimens, the absolute mean errors for interpolation and prediction are presented in Table 5.
4. **Prediction:** The Chebyshev polynomial prediction errors are comparable with those of the ANN. The presence of noise is associated with worse results by the ANNs. Still, in these conditions (presence of noise) the increase in the volume of the sample leads to a decrease in the errors of the ANNs. For example, even if the reliability of the input data remains at level $\alpha = 0.005$, the error of the neural network when predicting far from the sample boundaries goes from 14.0798 for a sample size of 100 elements to 11.6738 for a sample size of 5000. Interpolation errors in noisy conditions go from 2.5424 for a sample size of 100 to 0.0036 for a sample size of 500. The conclusion that can be made here is that for small samples and the presence of noise, the Chebyshev polynomial has an advantage in interpolation and prediction. However, if large data is processed, the preferred means of approximation should remain the ANN, regardless of the presence of noise.

4. Comparison of approximation in more complex tasks

For the needs of the study, additional tests were conducted on the approximation of polynomials and artificial neural networks, focusing on more complex functions. We considered multidimensional functions and discontinuous functions, as well as the possibility of approximating curves in three-dimensional Euclidean space. Here, we present the results from attempts to approximate the trajectories of bodies that describe a helical line in three-dimensional Euclidean space.

Let's consider the trajectory of a material point moving with a uniform linear velocity d, 1 along a generator of a right circular cylinder with radius a, which rotates around its axis with angular velocity w.

$$c: \; x_1 = a\cos wt, \; x_2 = a\sin wt, \; x_3 = d\,t, \; t \in R^+ \tag{27}$$

Lines of this type are called helical (Fig. 16). They have many applications in various fields of science

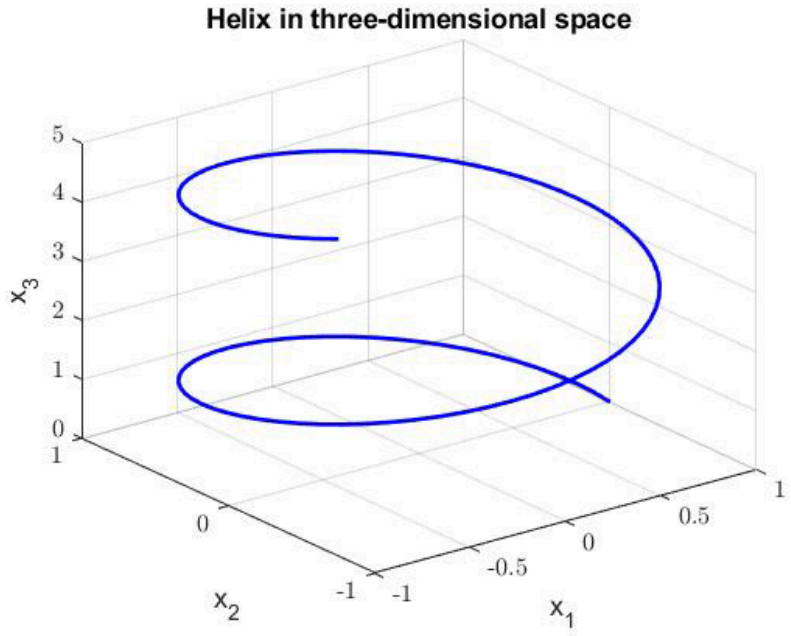


Figure 16: Helical line: $a = 1; w = 1; d = 0.5$

and engineering. They are used in astronomy to describe the orbital motions of the planets around the sun, in the aviation industry to model the trajectories of airplanes, and in electronics to describe the motion of charge carriers in semiconductor devices.

Helix lines can be mathematically described through various methods and theories—differential equations, the theory of functions of complex variables, and analytic geometry. In general, they are fundamental to understanding the motion and dynamics of objects in various contexts, and to extracting information about their trajectory and direction of motion.

We experimented with the approximation of the helix line for parameters $a = 1$, $d = 0.5$, and $w = 1$ in the three spatial directions. The best polynomial result was obtained through approximation with the Lagrange polynomial (Fig. 17), with a mean squared error of 13.123.

As expected, due to their ability to handle highly non-linear approximations, artificial neural networks achieved an approximation result with a root mean square error of 6E-14 (Fig. 19). The network has a two-layer architecture with 10 neurons in each layer. It is trained using the Levenberg-Marquardt algorithm. In the body of neurons in the hidden layer, the transfer function is a hyperbolic tangent. For the transfer functions in the three output neurons, a linear one, $f(x) = x$, is used. Remarkably, unlike the need for three separate polynomials to approximate the line in the three directions, the artificial neural network utilized just one model equipped with three output neurons, each indicating the development of the line along the respective directions.

With respect to dependencies that are discontinuous in nature, ANNs again outperform polynomials. One of the tasks we set, is the approximation of the distribution law of a binomially distributed discrete random variable ξ, with the function:

$$P(\xi = k) = P_n(k) = C_n^k p^k q^{n-k},\ q = 1 - p, k = 0, 1, \ldots, n \tag{28}$$

Both methods – polynomials and neural networks – attempt to represent the distribution function in a continuous form, which, in a sense, is a shortcoming. However, unlike the most efficient polynomial (Hermite) which has a squared error of 145.28 (for a specific Bernoulli scheme with $n = 1000$, $p = 1/6$), ANNs show results with squared errors smaller than 34.672.

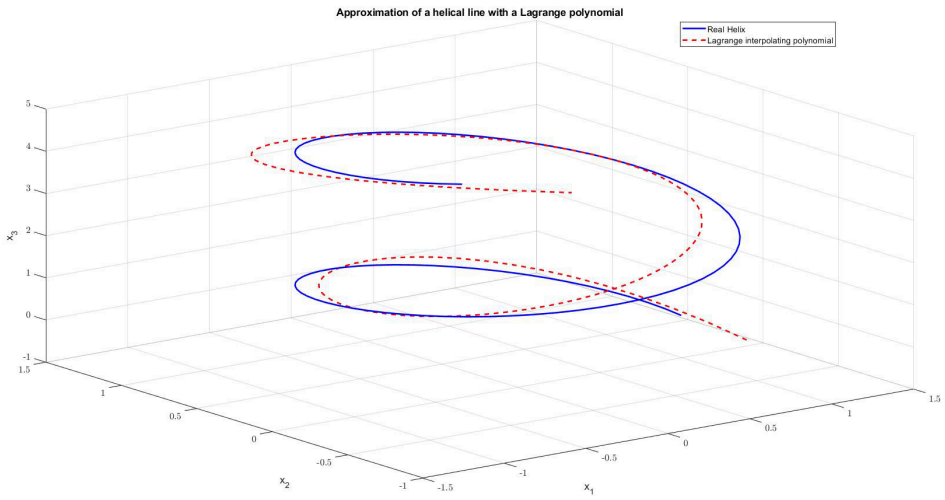


Figure 17: Interpolation of the helix with three Lagrange polynomials for each of the axes

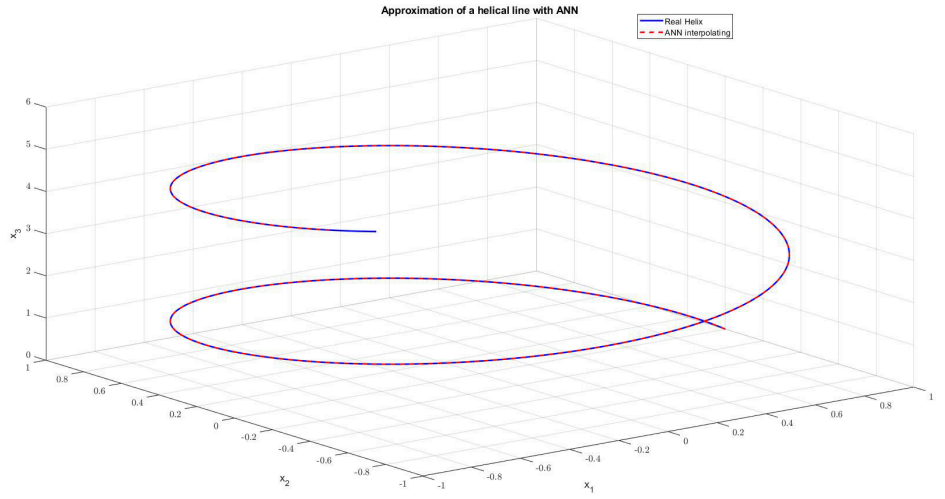


Figure 18: Interpolation with an artificial neural network

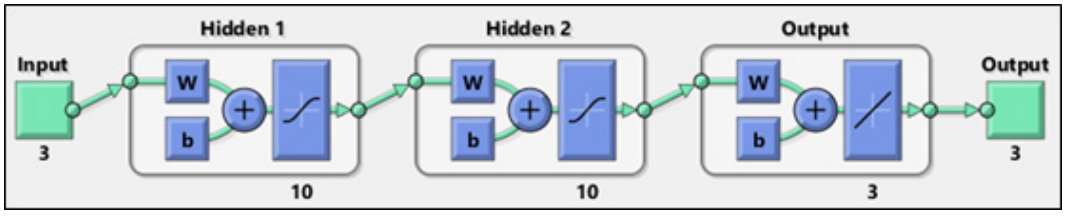


Figure 19: Architecture of the used artificial neural network

Neural networks are increasingly used to solve various approximation tasks, but the appropriate method must be carefully selected. For example, the task of researching the discretely monitored Barrier option is interesting. In [47], a Multivariate Normal Distribution method was proposed, which outperforms the Monte Carlo method—one of the most frequently used quantitative methods in finance—with higher convergence. Comparisons with other numerical methods provide the basis for a motivated choice [48]. On the other hand, the use of neural networks to solve this task must take into consideration the fact that the choice of network architecture is of great importance to accuracy and the training time of the machine learning algorithm [49]. Thus, in-depth research, analysis, and comparisons need to be done between the different approximation possibilities.

5. Conclusions

The conducted research has shown the importance of selecting an appropriate approximation method when we have to choose between some polynomials and artificial neural networks. For small sample volumes, due to the mathematical models on which the building of a polynomial is based, and due to the fact that in suitable situations there are not enough samples for training neural networks, polynomials should remain the preferred approach, especially when it comes to sample interpolation. But when it's about prediction, near and far from the boundaries of the available sample, we need to be particularly careful about the type of polynomial. Due to Runge's phenomenon which makes Lagrange, Newton, and Hermite types of polynomials deviate sharply from the real functions or have many oscillations, the preferred approach should be polynomials that are not strongly affected by this phenomenon. In such situations, a preferred solution is to choose the Chebyshev polynomial.

When working with large samples, the Lagrange, Newton, and Hermite polynomials lead to larger errors both in interpolation and prediction. These polynomials exhibit poorer scalability in terms of the speed of the algorithms and computational resources used, due to the increased degree of the polynomial. Thus, in the cases of larger samples, neural networks should remain the preferred approach. The case with the Chebyshev polynomial is interesting. When predicting functions, the Chebyshev polynomial is more stable and effective that the Lagrange, Newton, and Hermite polynomials, especially for more distant predictions. Regarding their comparison with artificial neural networks, it turns out that when interpolating functions in the absence of noise, the errors of those two approximation approaches are almost equal. In the presence of noise in the input data, however, the Chebyshev polynomials perform better than ANNs because they prefer to use Chebyshev nodes and do not attempt to take the noise into account. In contrast, the neural network aims to get trained using the available data and tries to keep track of the noise which leads to larger errors. On the other hand, the neural network stands out with its speed and scalability, which offers advantages when working with large volumes of data. So, in the end, it should be noted that the choice between the artificial neural network and the Chebyshev polynomial depends on the specific task and the conditions of the research. For small samples and the presence of noise, the Chebyshev polynomial is more suitable for interpolation and prediction. For large volumes of data and high speed and scalability requirements, an artificial neural network is the preferred means of approximation. In summary:

1. **For small samples:** The mathematical models of polynomials allow them to go through all the sample nodes without error. Their degree is small and the speed of execution is high. On the other hand, neural networks do not have enough training samples and the interpolation and prediction errors are either larger or comparable to those of polynomials. If the data is unreliable or noisy, the errors become larger. Therefore, in cases of small sample sizes, polynomial approximations are the more appropriate choice.
2. **For large samples, regardless of the presence or absence of noise in the data:** For very large samples, the degrees of the polynomials become very large. The speed of the algorithm shows down, more computing resources are used, and errors become larger. In situations with large samples, on the other hand, neural networks have enough training samples, and based on their adaptive nature, their errors start to decrease regardless of the presence of noise in the input data. Thus, in large sample settings, artificial neural networks are the natural choice for approximating the desired functional dependence.

3. **In more complex tasks characterized by multidimensionality of the objective functions and strong non-linearity or discontinuity:** In each of these cases, artificial neural networks have an advantage over polynomials. The complexity of these tasks highlights the superior non-linear capabilities of ensembles of neurons over those of commonly used polynomials. When trying to approximate discontinuous functions, both methods tend to represent the target function as continuous; however, artificial neural networks consistently demonstrate clear superiority.

The conducted research has the following limitations: Although it covers many objective functions of different natures, the scope and volume of the study do not allow for experimentation with all possible objective functions or even classes. We have used univariate and multivariate functions, linear and highly non-linear targets, and continuous and discontinuous dependencies. Nevertheless, there are many areas that the scope of this study does not include. Such areas include, for example, fractal objective functions, chaotic and dynamic systems, approximation and pulsation models, non-linear quantum models, and functions represented as infinite series.

Future research can be conducted in various directions.

1. Bearing in mind the Chebyshev polynomials advantages when working with unreliable input data and the disadvantage of neural networks using all provided nodes for training, regardless of the noise, we set ourselves the additional task for a future study – to investigate the qualities of neural networks. The task is related to studying and experimenting with artificial neural networks which do not use noisy data in their training, but instead use a set of Chebyshev nodes previously formed on them. Under these circumstances, it will be of genuine scientific interest to re-compare the approximation qualities of Chebyshev polynomials and artificial neural networks.
2. In attempts to approximate discontinuous dependencies, both approaches strive to convey the continuous nature of the sought-after objective function. Finding an approach that preserves the discrete nature of the objective remains a subject for future research.
3. Also of interest are experiments for comparison of approximation methods in areas too complex for polynomials, such as:

- **Fractal functions:** For example, the Koch curve, the Cantor set, and the Menger sponge. Polynomial approximation becomes increasingly inaccurate as the fractal detail increases.
- **Dynamic systems:** Chaotic systems, population models, climate models. Approximations with polynomials can "mask" the chaotic behavior of the system, and here the use of other, more effective approximation methods will be necessary.
- **Functions with infinite series:** Functions with infinite expansions are very difficult to approximate using polynomials. Examples include the Dirac function and the Heaviside function. Polynomial approximations will have errors at the discontinuity points, necessitating the use of other approaches that must be compared with the effectiveness of artificial neural networks.

Acknowledgement: Kostadin Yotov and Emil Hadzhikolev are partially financed by the MUPD23-FMI-021 project of the Research Fund of the University of Plovdiv "Paisii Hilendarski". Stanka Hadzhikoleva would like to thank project № BG-RRP-2.004-0001-C01, financed by the European Union-NextGenerationEU through the National Recovery and Resilience Plan of the Republic of Bulgaria.

References

[1] N. L. Crothers, *Approximation Theory - A Short Course*. Department of Mathematics and Statistics, Bowling Green State University, 2000.
[2] I. Kidron, *Polynomial approximation of functions historical perspective and new tools*, International Journal of Computers for Mathematical Learning **8** (2004), 299–331.
[3] R. Santaeulalia-Llopis, *Function Approximation Quantitative Macroeconomics*, Barcelona: MOVE-UAB and Barcelona GSE, 2018.
[4] A. Chkifa, N. Dexter, H. Tran, C. Webster, *Polynomial approximation via compressed sensing of high-dimensional functions on lower sets*, Mathematics of Computation **87(311)** (2016), 1415-1450.

[5] F. Filbir, W. Themistoclakis, *Polynomial approximation on the sphere using scattered data*, Mathematische Nachrichten **281(5)** (2008), 650-668.
[6] Y. Zhou, H. Wu, Ch. Gu, Y. Song, *Global optimal polynomial approximation for parametric problems in power systems*, J. Mod. Power Syst. Clean Energy **7(3)** (2019), 500–511.
[7] S. Regonda, B. Rajagopalan, U. Lall, Y. Moon, M. Clark, *Local polynomial method for ensemble forecast of time series*, Nonlinear Processes in Geophysics **12(3)** (2005), 397–406.
[8] M. Lauretto, F. Nakano, C. Pereira, J. Stern, *Hierarchical Forecasting with Polynomial Nets*, New Advances in Intelligent Decision Technologies, 2009.
[9] F. Dikbas, *Forecasting Extreme Precipitations by Using Polynomial Regression*, Pamukkale University, 2022, DOI: 10.21203/rs.3.rs-1918628/v1.
[10] M. Reimer, *Multivariate Polynomial Approximation*, Birkhäuser Basel, 2003, DOI: 10.1007/978-3-0348-8095-4.
[11] L. Refethen, *Multivariate polynomial approximation in the hypercube*, Proceedings of the American Mathematical Society, ISSN 1088-6826 (2017), 837–4844.
[12] R. Millán, V. Peiris, N. Sukhorukova, J. Ugon, *Multivariate approximation by polynomial and generalised rational functions*, Optimization **71(4)** (2021), 1171-1187.
[13] M. Aanjaneya, *Polynomial, Lagrange, and Newton Interpolation* (2017), 1-18.
[14] R. L. Burden, J. D. Faires, *Interpolation and Polynomial Approximation Lagrange Interpolating Polynomials* In Numerical Analysis, by R L Burden and J D Faires. Dublin: Brooks Cole, Cengage Learning, 2011.
[15] S. Celik, H. Inci, T. Sengul, B. Sogut, A. Sengul, M. Taysi, *Interpolation method for live weight estimation based on age in Japanese quails*, Revista Brasileira de Zootecnia **45(8)** (2016), 445-450.
[16] H. Zhou, T. Wang, H. Zhao, Z. Wang, *Updated Prediction of Air Quality Based on Kalman-Attention-LSTM Network*. Sustainability **15(1)** (2023), https://doi.org/10.3390/su15010356.
[17] C. Runge, *Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten*, Zeitschrift für Mathematik und Physik **46** (1901), 224–243.
[18] G. Dahlquist, A. Bjork, *Equidistant Interpolation and the Runge Phenomeno*, Numerical Methods, ISBN 0-13-627315-7, 1974.
[19] N. Belanger, *External Fake Constraints Interpolation: the end of Runge phenomenon with high degree polynomials relying on equispaced nodes – Application to aerial robotics motion planning*, Proceedings of the 5th Institute of Mathematics and its Applications Conference on Mathematics in Defence, Camberley, UK: The Royal Military Academy Sandhurst (2017).
[20] R. Muthumalai, *Note on Newton Interpolation Formula*, International Journal of Mathematical Analysis 6(50) (2012), 2459 - 2465.
[21] M. Floater, *Newton interpolation* (2014).
[22] C. Hermite, *Sur un nouveau développement en série de fonctions*, Comptes Rendus Acad. Sci. Paris **58** (1864), 93–100.
[23] F. Qi, B-N. Guo, *Some properties of the Hermite polynomials*, Georgian Mathematical Journal **28(6)** (2021), DOI: 10.1515/gmj-2020-2088, 925–935.
[24] J. C. Mason, D. C. Handscomb, *Chebyshev polynomials*, Boca Raton, London, New York, Washington, D.C.: CHAPMAN and HALL CRC A CRC Press Company, 2003.
[25] W. Van Assche, *Chebyshev polynomials in the 16th century*, KU Leuven, Belgium, 2022.
[26] M. Hagan, H. Demuth, M. Beale, O. Jesus, *Neural Network Design*, Stillwater, Oklahoma, ISBN: 0971732116, 2014.
[27] S. Haykin, *Neural Networks And Learning Machines*, Pearson India, ISBN-10: 9789332570313, 2016.
[28] C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, Springer, SBN-10: 3319944622, 2018.
[29] B. T. Zhang, *Artificial Neural Networks*, Seoul: School of Computer Science and Engineering, Seoul National University, 2001.
[30] J. Stastny, V. Skorpil, *Neural networks learning methods comparison*, University of Technology, 2005.
[31] A. Lunardi, F. Rodrigues, L. Junior, *Comparison of artificial neural networks learning methods to evaluate supply chain performance*, Curitiba, PR, Brasil, 2020.
[32] C. Bishop, *Neural Networks for Pattern Recognition*, Oxford, Clarendon Press, 1995.
[33] O. Abdel-Hamid, A.R. Mohamed, H. Jiang, L. Deng, G. Penn, D. Yu, *Convolutional Neural Networks for Speech Recognition*, IEEE/ACM Transactions on Audio, Speech, and Language Processing **22(10)** (2014), 1533-1545.
[34] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, SYNTHESIS LECTURES ON HUMAN LANGUAGE TECHNOLOGIES, ISBN: 9781627052986, Toronto: Morgan and cLaypool publishers, 2017.
[35] X. Li, *Application of Neural Networks in Financial Time Series Forecasting Models*, Mathematical Modeling for Next-Generation Big Data Technologies (2022), Article ID 7817264, https://doi.org/10.1155/2022/7817264.
[36] D. Janglova, *Neural Networks in Mobile Robot Motion*, International Journal of Advanced Robotic Systems **1(1)** (2004), DOI: 10.5772/5615.
[37] A. Egba, O. Okonkwo, *Artificial Neural Networks for Medical Diagnosis: A Review of Recent Trends* International Journal of Computer Science and Engineering Survey **11(3)** (2020), 1-11.
[38] R. Ahmed, *NEURAL NETWORKS IN FINANCE: A DESCRIPTIVE SYSTEMATIC REVIEW*, Indian Journal of Finance and Banking **5(2)** (2021), DOI: 10.46281/ijfb.v5i2.997.
[39] M. Martin, *On-line Support Vector Machines for Function Approximation*, Technical Report LSI-02-11-R, 2002.
[40] B. Hammer, K. Gersmann, *A Note on the Universal Approximation Capability of Support Vector Machines*, Neural Processing Letters **17** (2003), 43–53.
[41] K. Balaskas, G. Zervakis, K. Siozios, M. Tahoori, J. Henkel, *Approximate Decision Trees For Machine Learning Classification on Tiny Printed Circuits*, 23rd International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA (2022), pp. 1-6, https://doi.org/10.1109/ISQED54688.2022.9806213.
[42] M. Bressan, M. Sozio, *Fully-Dynamic Approximate Decision Trees With Worst-Case Update Time Guarantees*, arXiv:2302.03994 (2023), DOI: https://doi.org/10.48550/ARXIV.2302.03994.
[43] G. Acampora, A. Chiatto, A. Vitiello, *Genetic algorithms as classical optimizer for the Quantum Approximate Optimization Algorithm*,

Applied Soft Computing **142** (2023), https://doi.org/10.1016/j.asoc.2023.110296.
[44] G. Yi, Z. Wenhao, *Optimal operation of urban tidal drainage pumping station based on genetic algorithm coupled with head-water level successive approximation*, Frontiers in Energy Research **10** (2022), https://doi.org/10.3389/fenrg.2022.1074529.
[45] K. Yotov, E. Hadzhikolev, S. Hazdhikoleva, *Determining the Number of Neurons in Artificial Neural Networks for Approximation, Trained with Algorithms Using the Jacobi Matrix*, TEM Journal **9(4)** (2020), 1320–1329.
[46] K. Yotov, E. Hadzhikolev, S. Hadzhikoleva, S. Cheresharov, *Finding the Optimal Topology of an Approximating Neural Network*, Mathematics **11(1)** (2023), https://doi.org/10.3390/math11010217.
[47] S. Kabaivanov, A. Malechkova, A. Marchev, M. Milev, V. Markovska, K. Nikolova, *A step beyond the Monte Carlo method in economics: Application of multivariate normal distribution*, AIP Conference Proceedings **1690 (1)** (2015) https://doi.org/10.1063/1.4936693.
[48] Y. Honga, S. Leeb, T. Li, *Numerical method of pricing discretely monitored barrier option*, Journal of Computational and Applied Mathematics **278** (2015), 149-161.
[49] L. Mieghem, A. Papapantoleon, J. Papazoglou-Hennig, *Machine learning for option pricing: an empirical investigation of network architectures* (2023), Papers 2307.07657, arXiv.org.