Artificial Neural Networks and Wiener-Hopf Factorization

Elena Alymova and Oleg Kudryavtsev, *Member, IAENG*

***Abstract*—The paper suggests a hybrid numerical method to price barrier options under Lévy processes. As the main ingredient of our approach, we model the values of the Wiener-Hopf factors using artificial neural networks in the exact formula for the solution. The numerical Wiener-Hopf factorization typically reduces the problem to the factorization of the polynomial of $\exp(i\xi)$, which is interpreted as the characteristic function of the random variable that approximates the Lévy process at the exponentially distributed time moment. We design and train a feedforward neural network with one hidden layer that approximates the coefficients of factors based on the input vector of the factorized polynomial coefficients. We implemented in the software a training data generator and a generalized loss function to factorize a polynomial of arbitrary degree. We demonstrate the performance of our approach using examples of factorization of second-, sixth- and 254th-degree polynomials. It takes a fraction of a second for our trained artificial neural networks to calculate the factors.**

***Index Terms*—Wiener-Hopf factorization, numerical methods, option pricing, Lévy processes, artificial neural networks**

I. Introduction

COMPUTATIONAL finance serves as the driver for developing numerical methods. In this paper, we consider the problem of numerical pricing barrier options under Lévy processes, which are popular jump models in a broad range of applications, including finance (see, e.g., [1]–[3] the main approaches to jump model calibration). For an introduction to these models, we refer to [1].

The option pricing problem under stochastic processes is crucial for computational finance. Recall that a barrier option pays the specified amount on its expiration date T, provided that during the option's lifetime, the price of the underlying asset does not cross a fixed constant barrier H (*knockout barrier options*). If the underlying price crosses the barrier, the option becomes worthless.

From a probabilistic perspective, the price of the barrier option in the Lévy model can be expressed in terms of the conditional expectation of the payoff function, which depends on the Lévy process and its supremum (or infimum). In financial practice, the fast speed and algorithmizability of computational methods are of great importance. Unfortunately, in this sense, pricing problems in Lévy models are still a challenge for researchers.

The main groups of "traditional" numerical methods for pricing options in Lévy financial models include:

Manuscript received January 23, 2024; revised June 30, 2024. This work was supported by the Russian Science Foundation grant No 23-21-00474.

O. Kudryavtsev is Head of the Department of Computer Science and Customs Technologies at the Rostov Branch of the Russian Customs Academy, Rostov-on-Don, Budennovskiy 20, 344002, Russia (e-mail: koe@donrta.ru)

E. Alymova is Researcher at InWise Systems, LLC, Rostov-on-Don, Eremenko 58/11, 344015, Russia (e-mail: langnbsp@gmail.com)

- Monte Carlo methods (classical Monte Carlo methods, multilevel Monte Carlo algorithms, Monte Carlo algorithms combined with Wiener-Hopf factorization, see, e.g., [4]–[6]);
- Numerical methods for computing the correspondent expectation (backward induction and Fourier transform, projection methods, Wiener-Hopf factorization method in combination with the Laplace transform, see, e.g., [7]–[11])
- Numerical methods for solving integrodifferential partial differential equations (method of multinomial trees, finite difference method, Wiener-Hopf factorization method, see, e.g., [12]–[19]).

A detailed review can be found in a recent article [20].

Recently, research in the field of computational finance has made great strides through the use of machine learning techniques. Typically, machine learning methods have been used to analyze and model financial markets. Currently, there is a trend towards the development of hybrid methods of computational mathematics using machine learning methods and classical numerical methods. In the field of computational finance, these methods have been used to solve problems of pricing European options in Lévy models in relatively simple cases. Hybrid methods, combining "traditional" numerical methods with machine learning algorithms, can help solve this problem and significantly influence the development of computational finance.

The ability to price options using machine learning methods follows from Cybenko's theorem presented in [21], which states that an artificial feedforward neural network without cycles, with one hidden layer, can approximate any continuous function of many variables with any accuracy. Unfortunately, direct training of such networks can cause a number of practical difficulties, both due to the complexity of collecting and analyzing statistical data necessary to construct a training set and of a computational nature (the required training time to achieve satisfactory accuracy and the amount of memory occupied). In [22], to speed up and refine modeling, initial assumptions about the dynamics of the process are used, expressed in the form of a "universal differential equation," which is defined in full or part by a universal approximator (a neural network, a random forest or another model).

An approach to solving multidimensional option pricing problems in models with stochastic volatility and/or jumps described in [23] is based on the use of deep neural networks. In the paper mentioned, the neural network is trained to handle the values of the emerging differential operator and the initial-boundary conditions of the problem. The algorithm used (called the "Deep Galerkin method" by the authors of the article for its ideological similarity to the Galerkin method) is trained on data sets randomly located in time and space meshes and is capable, in some cases, of solving equations of up to 200 variables. In addition, as shown in [24], it is also possible in some cases to obtain irregular solutions using deep neural networks. This property can be useful for pricing barrier options using operator techniques, where functions whose derivatives are discontinuous may appear during the calculations. Finally, for the case of Lévy models, in [25] the neural network is applied to exponential Lévy processes (e.g., Kou [26] and CGMY [27] models) for pricing European options.

The use of machine learning methods to price barrier options and other path-dependent derivatives is a case that is currently not examined in detail in existing publications. The most promising approach can be to use neural networks to handle typical subproblems in the framework of some classical numerical methods.

A Wiener-Hopf method is a universal tool for solving two-dimensional initial boundary value problems for pricing path-dependent options under Lévy processes. However, in the case of general Lévy models, the Wiener-Hopf factors are not available in a closed form and should be approximated by using special numerical tricks. In the current paper, we study the possibility to approximate Wiener-Hopf factors with artificial neural networks as the main ingredient of the fast, accurate and universal numerical method for pricing barrier options under Lévy models.

Artificial neural networks (ANN) have already shown their practical effectiveness and are very popular in solving problems using machine learning methods. ANNs are used in modern software applications for pattern recognition [28], [29], medical data analysis [30], [31], stock rate prediction [32], and the solution of computational finance problems [25]. Universal approximation theorems have been proven for neural networks [21], [33]–[35] to show the effectiveness of a neural network in approximating functions of various classes.

The work [36] examines the optimal Kolmogorov approximation of a function using deep neural networks. Deep networks are shown to provide exponential accuracy in the approximation of the multiplication operation, polynomials, sinusoidal functions, and some smooth functions. The research in [37] is devoted to studying the precision, computational efficiency, and complexity of neural network training. The potential to develop more effective solutions is identified by carefully designing network architectures and training strategies.

Fast numerical algorithms in combination with machine learning methods to price options can become the basis for the implementation of risk management trading systems in the financial market. A combined approach using machine learning methods to select an effective trading strategy on the exchange is presented in [38]. The mentioned approach uses the calculation of the linear regression slope coefficient using logarithmic return indicators and the determination of the trend of quotes of the BTC/USD currency pair in the next period based on the calculated sign of the coefficient.

The goal of the current paper is to suggest a hybrid numerical method to price barrier options under Lévy processes. The main advantage of the method is to apply an efficient approximation of the Wiener-Hopf factors with artificial neural networks in the exact formula for the solution. In the present work, we use a fully connected feedforward neural network with one hidden layer to find an approximate solution to the Winner-Hopf factorization problem.

II. Theoretical background

A. Pricing barrier options under Lévy processes

In short, Lévy processes are stochastically continuous processes with stationary independent increments (further details can be found in [39]). According the Lévy-Khintchine formula, the characteristic exponent ψ of a one-dimensional Lévy process X_t is defined by:

$$\psi(\xi) = \frac{\sigma^2}{2}\xi^2 - i\gamma\xi + \int_{-\infty}^{+\infty} (1 - e^{i\xi y} + i\xi y\mathbf{1}_{[-1;1]}(y))F(dy), \tag{1}$$

where $\sigma^2 \geq 0$ and μ are the variance and the drift of the Gaussian component, $\mathbf{1}_{[-1;1]}$ is the indicator function of the interval $[-1;1]$, and the Lévy measure $\Pi(dy)$ satisfies

$$\int_{\mathbf{R}\setminus\{0\}} \min\{1, y^2\}\Pi(dy) < +\infty.$$

Recall that X_t is completely specified by its characteristic exponent by the equality $E[e^{i\xi X(t)}] = e^{-t\psi(\xi)}$.

In applications to finance, the following non-Gaussian Lévy models are among the popular ones.

Example 1. [CGMY] The characteristic exponent of a pure jump CGMY process [27] is given by

$$\begin{aligned} \psi(\xi) &= -i\gamma\xi + C\Gamma(-Y)[G^Y - (G+i\xi)^Y] \\ &+ C\Gamma(-Y)[M^Y - (M-i\xi)^Y], \end{aligned} \tag{2}$$

where $C > 0$, $\mu \in \mathbf{R}$, $G > 0$, $M > 1$, and $Y \in (0,2), Y \neq 1$.

Example 2. [Kou model] The characteristic exponent of Kou model introduced in [26] is of the form

$$\psi(\xi) = \frac{\sigma^2}{2}\xi^2 - i\gamma\xi + \frac{ic_+\xi}{\lambda_+ + i\xi} + \frac{ic_-\xi}{\lambda_- + i\xi}, \tag{3}$$

where $c_\pm \geq 0$ and $\lambda_- < -1 < 0 < \lambda_+$.

Example 3. [Compound Poisson process with binomial distribution of jumps] The characteristic exponent of the compound Poisson process with binomial distribution of jumps has the following form

$$\psi(\xi) = \lambda \cdot (1 - p_r e^{id\xi} - p_l e^{-id\xi}), \tag{4}$$

where $\lambda, d > 0$ and $0 < p_r, p_l < 1$, $p_r + p_l = 1$.

Let $T, H, G(S)$ be the maturity, strike, and pay-off function. Assume that the riskless rate r is constant and, under a risk neutral measure chosen by the market, the underlying asset dynamics is modeled with the exponential Lévy process $S_t = \exp(X_t)$. The characteristic exponent $\psi(\xi)$ then admits the analytic continuation in the strip $\Im\xi \in (-1,0)$ and the continuous continuation in the closed strip $\Im\xi \in [-1,0]$, since $E[\exp(X_t)] < +\infty$.

Introduce the supremum process $\overline{X}_t = \sup_{0\leq s\leq t} X_s$ and the infimum process $\underline{X}_t = \inf_{0\leq s\leq t} X_s$, which are of great importance in pricing barrier options.

Then the no-arbitrage down-and-out option price at $t = 0$ is defined as follows (see, e.g. [9]):

$$V_{do}(x,T) = E\left[e^{-rT}G(\exp(x+X_T))\mathbf{1}_{\exp(x+\underline{X}_T)>H}\right], \tag{5}$$

where $G(S)$ is the payoff at time T, and $\mathbf{1}_{Y>H}$ is equal to 1 if the condition $Y > H$ is satisfied, and 0 – otherwise.

The no-arbitrage price of up-and-out option at $t = 0$ writes analogously, but in the terms of the supremum process:

$$V_{uo}(x,T) = E\big[e^{-rT}G(\exp(x+X_T))\mathbf{1}_{\exp(x+\overline{X}_T)<H}\big], \quad (6)$$

The universal tool (see, e.g., [9], [11], [18], [40]) that allows calculating expectations (5) and (6) is the Wiener-Hopf method that leads to factorization of the characteristic function $E[e^{i\xi X_{T_q}}] = q/(q+\psi(\xi))$, where T_q is an exponentially distributed random variable with intensity parameter $q > 0$. Carr's randomization or the Laplace transform reduces the pricing problem to the calculation of the appropriate sequence of expectations of the following kind

$$V_q(x) = E\big[G\left(\exp(x + X_{T_q})\right)\mathbf{1}_{x+\underline{X}_{T_q}>\ln H}\big]. \quad (7)$$

The expectation of the form (7) can be easily computed using the Wiener-Hopf factorization method and the Fast Fourier Transform algorithm when the factors are known. According to the state-of-the-art implementation of the Wiener-Hopf method (see, e.g., [10]), we have

$$V_q(x) = \mathcal{F}^{-1}_{\xi\to x}\phi^-_q(\xi)\mathcal{F}_{x\to\xi}\mathbf{1}_{x>\ln H}\mathcal{F}^{-1}_{\xi\to x}\phi^+_q(\xi)\mathcal{F}_{x\to\xi}G(x), \quad (8)$$

where $\mathcal{F}_{x\to\xi}$ is the Fourier transform of the function from spatial domain (x) to frequency domain (ξ), $\mathcal{F}^{-1}_{\xi\to x}$ is the inverse Fourier transform of the function from frequency domain to spatial domain, $\phi^+_q(\xi)$, $\phi^-_q(\xi)$ are the characteristic functions of the random variables $\overline{X}_{T_q}, \underline{X}_{T_q}$, respectively, that factorize the characteristic function $\frac{q}{q+\psi(\xi)}$ of X_{T_q}.

However, when considering general Lévy models, it is not possible to obtain explicit formulas for the characteristic functions $\phi^{\pm}_q(\xi)$ in the Wiener-Hopf factorization identity

$$\frac{q}{q+\psi(\xi)} = \phi^+_q(\xi)\phi^-_q(\xi), \quad \forall\ \xi \in \mathbf{R}. \quad (9)$$

Evaluating the Wiener-Hopf factors $\phi^{\pm}_q(\xi)$ numerically requires advanced numerical methods, which complicates the implementation of the Wiener-Hopf method for practitioners. In the paper, we suggest a new approach that simplifies the factorization technique by using artificial neural networks for the approximation of the Wiener-Hopf factors in the exact formula (8) for expectation (7). In the next subsection, we briefly give the main definitions and facts on feedforward neural networks.

B. A short introduction to artificial neural networks

A feedforward artificial neural network (ANN in short) consists of three types of computational block called layers (the input layer, hidden layers, and the output layer) and determines a nonlinear function:

$$f : R^d \to R^M, \quad (10)$$

where d is the number of neurons at the input (first) layer and M is the number of neurons at the output (last) layer. Let L be the number of hidden layers in the network, N_l be the number of neurons in the lth layer, the vector $\mathbf{x} = \left(z^0_1, \ldots, z^0_d\right)$ be the input layer 0, and let z^l_j denote the output of the j -th neuron in the layer l, $w^l_{i,j}$ be the weight of the arc that connects the neuron i of the lth layer and the neuron j of the layer $l+1$, let b^l_j denote the bias of the neuron j in the layer l, and $\mathbf{y} = \left(z^{L+1}_1, \ldots, z^{L+1}_m\right)$ be the output $(L+1)$th layer. Then the outputs z^{l+1}_j between the current $(l+1)$th and previous lth layers in the feedforward network are determined by the following relation.

$$\begin{aligned} z^{l+1}_j &= \sigma_l\left(\sum_{i=1}^{N_l} w^l_{i,j} z^l_i + b^{l+1}_j\right), \\ &\quad 0 < l < L,\ 1 \le j \le N_{l+1}. \end{aligned} \quad (11)$$

where where $\sigma_l(\cdot)$ is the activation function. In this work, we consider only two types of activation function:

- linear activation function: $\sigma_l(z) = z$,
- sigmoid activation function: $\sigma_l(z) = \frac{1}{1+e^{-z}}$.

Using the relations (11), one can define a non-linear function (10) that expresses the output layer $\mathbf{y}$ via the input layer $\mathbf{x}$.

The number of layers in a neural network determines its depth, and the number of neurons in the network determines its size.

The universal approximation theorem, proven by George Cybenko (Cybenko's theorem), states that a feedforward artificial neural network with a single hidden layer can approximate any continuous function of many variables on a compact with any accuracy [21]. Thus, it is important to define a sufficient number of neurons in the hidden layer, weights between the input neurons and neurons of the hidden layer, weights between connections from neurons in the hidden layer and the output neuron, and biases.

In Figure 1 we present the architecture of a simple feedforward neural network with four neurons in the input layer, three neurons in the only single hidden layer, and two output neurons.

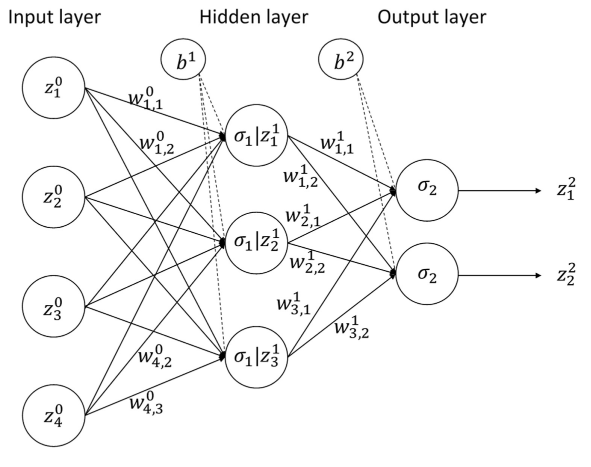

Figure 1. The architecture of a feedforward neural network with one hidden layer

In this article, we will design a feedforward network with one hidden layer to obtain an approximation of the factorization of the polynomial function. The number of neurons in the hidden layer will be chosen empirically to obtain a practically significant precision.

When training a neural network, the loss function is of great importance. It is used to calculate the error between the ground-truth labels (values that we want to predict) and the predicted values. The main goal of ANN training is to minimize this error.

The mean absolute error (MAE) and the mean square error (MSE) are the most common loss functions. To calculate MAE, one needs to take the absolute difference between the predicted values y_i and the ground truth labels $\tilde{y}_i$ and average them over the entire data set:

$$\text{MAE } = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{M} \left| y_i^k - \tilde{y}_i^k \right|,$$

where N is the number of samples in the train or test datasets, and M is the dimension of the output layer. MAE evaluates how close the predictions of the ANN are to the actual values. This metric is less sensitive to outliers and can give a generalized estimate of the quality of the model.

To calculate MSE, you need to take the difference between the predicted values and the ground truth labels, square it, and average it over the entire data set:

$$\text{MSE } = \frac{1}{N} \sum_{k=1}^{N} \sum_{i=1}^{M} \left(y_i^k - \tilde{y}_i^k \right)^2. \tag{12}$$

The metric (12) is useful for identifying anomalies.

In this paper, we do not use standard loss functions when training a neural network to find polynomial factors for a given polynomial. In formulas such as (8), we are not interested in the proximity of the actual and predicted values of the coefficients of the polynomial factors, but in the identity of the original and factorized polynomial. Therefore, we will propose using our custom loss function that takes this specifics into account using the forward and inverse discrete Fourier transforms to evaluate factorization accuracy.

III. Factorization of polynomials using neural networks

A. The problem setup of polynomial factorization using a neural network

According to [5], [9], for a given positive d and a large $M(= 2^N,\ N \in \mathbb{N})$ we can approximate $\phi^+(\xi)$, $\phi^-(\xi)$ and $q(q+\psi(\xi))^{-1}$ in (9) with the following kind of Fourier series.

$$q(q+\psi(\xi))^{-1} \approx \sum_{k=-M/2+1}^{M/2-1} p_k e^{i\xi kd}, \tag{13}$$

$$\sum_{k=-M/2}^{M/2-1} p_k = 1, p_k \geq 0; \tag{14}$$

$$\phi_q^+(\xi) \approx \sum_{k=0}^{M/2-1} p_k^+ e^{i\xi kd}, \tag{15}$$

$$\sum_{k=0}^{M/2-1} p_k^+ = 1, p_k^+ \geq 0; \tag{16}$$

$$\phi_q^-(\xi) \approx \sum_{k=0}^{M/2-1} p_k^- e^{-i\xi kd}, \tag{17}$$

$$\sum_{k=0}^{M/2-1} p_k^- = 1, p_k^- \geq 0; \tag{18}$$

Since $\{p_k\}$, $\{p_k^-\}$, and $\{p_k^+\}$ approximate the probability distributions of X_{T_q}, $\underline{X}_{T_q}$ and $\overline{X}_{T_q}$, respectively, we assume that

i both sequences $\{p_k^-\}_{k=0}^{k=M/2-1}$ and $\{p_k^+\}_{k=0}^{k=M/2-1}$ are strictly decreasing.

ii the numbers p_k do not decrease as k increases from $-\frac{M}{2}+1$ to 0, and do not increase as k increases from 0 to $\frac{M}{2}-1$.

If the number M is not large, we can rewrite the formula (8) in explicit form on the grid $x_n = nd, n \in \mathbf{Z}$:

$$V_q(nd) = \sum_{k=0}^{M/2-1} W_q((n-k)d)p_k^-, nd > \ln H, \tag{19}$$

$$W_q(nd) = \sum_{k=0}^{M/2-1} G((n+k)d)p_k^+, nd > \ln H, \tag{20}$$

$$W_q(nd) = 0, nd \leq \ln H \tag{21}$$

Notice that the decomposition (13) can be found using the following formulas. Set for each $k = -M/2+1, -M/2+2, \ldots, 0, \ldots, M/2-1:$

$$p_k = \frac{d}{2\pi} \int_{-\pi/d}^{\pi/d} \Big(q(q+\psi(\xi))^{-1} \Big) e^{-i\xi kd} d\xi. \tag{22}$$

Thus, we can reduce our factorization problem (9) to the following. For a given set of coefficients p_k, $k = -M/2+1, ..., M/2-1$, such that (14) holds, we want to find to set of coefficients p_k^+, $k = 0, ..., M/2-1$, and p_k^-, $k = 0, ..., M/2-1$ such that (16), (18) are valid, and satisfy the following factorization identity:

$$\sum_{k=-M/2-1}^{M/2-1} p_k e^{i\xi kd} = \sum_{k=0}^{M/2-1} p_k^+ e^{i\xi kd} \cdot \sum_{k=0}^{M/2-1} p_k^- e^{-i\xi kd}. \tag{23}$$

Note that the formula (23) can be interpreted as the factorization of characteristic functions. In fact, consider a discrete random variable X that takes the following positive, negative, and zero values. Assume that this random variable can be represented in the following form

$$X = X^- + X^+, \tag{24}$$

where $X^- \leq 0$ and $X^+ \geq 0$ are independent discrete random variables with the following unknown probability distributions

$$\mathbf{P}(X^- = -kd) = p_k^-, k = 0, ..., M/2-1, \tag{25}$$

and

$$\mathbf{P}(X^+ = kd) = p_k^+, k = 0, ..., M/2-1, \tag{26}$$

where $\{p_k^-\}$, and $\{p_k^+\}$ satisfy (i). Then the characteristic exponent of X is defined by (9), and (ii) is valid for $\{p_k\}$.

Thus, the factorization problem (9) can be reformulated as follows. For a given distribution of X:

$$\mathbf{P}(X = kd) = p_k, k = -\frac{M}{2}+1, -\frac{M}{2}+2, \ldots, 0, \ldots, \frac{M}{2}-1, \tag{27}$$

with $\{p_k\}$ that satisfies (ii), we need to find the distributions of X^- and X^+.

In the following subsection, we reduce our factorization problem to factorization of polynomials.

B. Factorization of polynomials

Let the following sequence of $M-1$ numbers be given:

$$p_0, p_1, \dots, p_{M-2},$$

where

$$p_k \geq 0, k = \overline{0, M-2}; \sum_{k=0}^{M-2} p_k = 1, M = 2^m, m \in \mathbb{N}. \quad (28)$$

According to similar considerations as in Section III-A, $\phi_X(\xi) = \sum_{k=0}^{M-2} p_k e^{i\xi(k-\frac{M}{2}+1)}$ is the characteristic function of a discrete random variable X, taking values at points $x_k = k - \frac{M}{2} + 1$ with probabilities p_k, $k = 0, \dots, M-2$.

Next, we impose an additional condition on the sequence $\{p_k\}$. For $k \leq \frac{M-2}{2}$ the sequence p_k increases and for $k \geq \frac{M-2}{2}$ it decreases. We want to represent the characteristic function of X as the following product of the characteristic functions of two independent discrete random variables taking non-negative and non-positive values:

$$\phi_X(\xi) = \sum_{k=0}^{M/2-1} \beta_k e^{i\xi k} \cdot \sum_{k=0}^{M/2-1} \alpha_k e^{-i\xi k},$$

where

$$\alpha_i > 0, i = \overline{0, M/2-1}; \qquad \sum_{i=0}^{\frac{M}{2}-1} \alpha_i = 1, \quad (29)$$

$$0 < \alpha_i < \alpha_{i+1} < 1, \qquad i = \overline{0, M/2-1}; \quad (30)$$

$$\beta_i > 0, i = \overline{0, M/2-1}; \qquad \sum_{i=0}^{\frac{M}{2}-1} \beta_i = 1 \quad (31)$$

$$1 > \beta_i > \beta_{i+1} > 0, \qquad i = \overline{0, M/2-1}. \quad (32)$$

The indicated task reduces to finding the following factorization

$$p(x) = q(x) \cdot r(x), \quad (33)$$

where

$$\begin{aligned} q(x) &= \alpha_0 + \alpha_1 x + \dots + \alpha_{\frac{M}{2}-1} x^{\frac{M}{2}-1}; \\ r(x) &= \beta_0 + \beta_1 x + \dots + \beta_{\frac{M}{2}-1} x^{\frac{M}{2}-1}; \\ p(x) &= p_0 + p_1 x + \dots + p_{M-2} x^{M-2}. \end{aligned}$$

To solve the problem (33), we train a neural network to find the coefficients of the polynomials q and r, the product of which is equal to the given polynomial p.

The desired neural network should approximate the function

$$F : (p_i) \to (\alpha_j, \beta_k),$$

where $M = 2^m, m \in \mathbb{N}$, $i = \overline{0, M-1}$; $j, k = \overline{0, M/2-1}$. Thus, in this paper, we consider the factorization of polynomials of degree $2^m - 2$, $m \in \mathbb{N}, m > 1$.

In this work, the discrete Fourier transform is used to establish the identity (33) of the coefficients found using a neural network.

For a given natural number M being the power of two, the discrete Fourier transform of the vector $(p_0, p_1, \dots, p_{M-1})$ can be interpreted as computing the values of the polynomial

$$p(x) = p_0 + p_1 x + \dots + p_{M-1} x^{M-1}$$

at the roots of unity $x_k = e^{2\pi i k/M}$, $k = \overline{0,\ M-1}$. Note that in the factorization problem (33) the polynomial $p(x)$ is of degree $M-2$, that is, $p_{M-1} = 0$.

It is well known that the values of a n-th degree polynomial at $n+1$ different points uniquely determine the polynomial itself. Furthermore, if $q(x_0) = a$ and $r(x_0) = b$, then $r(x_0) \cdot q(x_0) = ab$. Thus, due to the values of the polynomials $r(x)$ and $q(x)$ at every M-th root of unity, we can determine the values of the polynomial $p = r \cdot q$ at the same points and recover its coefficients p_k using the discrete inverse Fourier transform.

C. Generating training data

In machine methods framework, the training dataset is of great importance to fit the model. In the problem considered, a set of train data for our neural network is a set of pairs of polynomials of a given degree $M/2-1$ as input data and their products (polynomials of degree $M-2$) as output data under the conditions described in Section III-B.

The initial data of the function for generating polynomial coefficients for the training sample for a given M are the sequences α and β defined in Section III-B. These sequences define factor polynomials, the product of which gives us the original polynomial suitable for factorization using a neural network.

As the most simple case, we consider a trinomial factorization that corresponds to $M = 4$. It is easy to see that for $M = 4$ the equality (33) takes the following form:

$$p_0 + p_1 x + p_2 x^2 = (\alpha_0 + \alpha_1 x)(\beta_0 + \beta_1 x). \quad (34)$$

From (34) we derive the formulas for computing coefficients p_0, p_1, p_2 for given $\alpha_i, \beta_i (i = 0, 1)$:

$$\begin{aligned} p_0 &= \alpha_0 \cdot \beta_0; \\ p_1 &= \alpha_1 \cdot \beta_0 + \alpha_0 \cdot \beta_1; \quad (35) \\ p_2 &= \alpha_1 \cdot \beta_1. \end{aligned}$$

For $M = 8$ the identity (33) reads as follows:

$$\begin{aligned} \sum_{i=0}^{6} p_i x^i &= (\alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3) \times \quad (36) \\ & \quad (\beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3). \end{aligned}$$

Using (36) we derive the following explicit formulas for the coefficients $p_i (i = \overline{0,6})$ for a given $\alpha_j, \beta_j (j = \overline{0,3})$:

$$\begin{aligned} p_0 &= \alpha_0 \cdot \beta_0; \\ p_1 &= \alpha_0 \cdot \beta_1 + \alpha_1 \cdot \beta_0; \\ p_2 &= \alpha_0 \cdot \beta_2 + \alpha_1 \cdot \beta_1 + \alpha_2 \cdot \beta_0; \\ p_3 &= \alpha_0 \cdot \beta_3 + \alpha_1 \cdot \beta_2 + \alpha_2 \cdot \beta_1 + \alpha_3 \cdot \beta_0; \quad (37) \\ p_4 &= \alpha_1 \cdot \beta_3 + \alpha_2 \cdot \beta_2 + \alpha_3 \cdot \beta_1; \\ p_5 &= \alpha_2 \cdot \beta_3 + \alpha_3 \cdot \beta_2; \\ p_6 &= \alpha_3 \cdot \beta_3. \end{aligned}$$

We use formulas (35) and (37) to generate artificial training data for a neural network to factor polynomials of the second and sixth degrees, respectively.

Next, we define the generator of artificial training data for arbitrary $M = 2^N,\ N \in \mathbb{N}$, as follows.

Let us first introduce two M-dimensional vectors of the coefficients of the polynomial factors supplemented with extra zeros:

$$A = \left[\alpha_0,\ \alpha_1, \ldots, \alpha_{M/2-1}, 0, \ldots 0\right];$$

$$B = \left[\beta_0,\ \beta_1, \ldots, \beta_{M/2-1}, 0, \ldots 0\right].$$

When constructing a training sample, we generate sequences $\alpha_i, i = \overline{0, M/2-1}$ and $\beta_i, i = \overline{0, M/2-1}$ so that they meet conditions (29), (30), and (31), (32), respectively. To achieve these conditions, we introduce a sequence of increasing and non-overlapping intervals such that the upper boundary of the adjacent interval on the left is significantly less than the lower boundary of the adjacent interval on the right. Our α_i random value generator selects one value from each interval and then normalizes them. The elements of β_i are selected in a similar way, but in reverse order.

In particular, for $M = 8$, the sequence $\alpha_i, i = \overline{0, M/2-1}$ is generated as follows. First, a sequence of 4 random integers is constructed in the ranges $[0, 100], [300, 500], [1000, 1500], [5000, 10000]$. Then the resulting sequence is normalized, that is, each element is divided by the sum of all elements constructed. The sequence $\beta_i, i = \overline{0, M/2-1}$ is generated in a similar way, but with its subsequent inversion.

Then the M-dimensional vector

$$P = \left[p_0,\ p_1, \ldots, p_{M-2}, 0\right]$$

of the coefficients of the correspondent polynomial product of degree $M-2$ (with zero coefficient at degree $M-1$) for the training sample can be efficiently determined by the formula:

$$P = iDFT(DFT\,(A) \cdot DFT\,(B)),$$

where DFT is the discrete Fourier transform, iDFT is the inverse discrete Fourier transform and $\cdot$ is the element-wise multiplication of the sequences A and B.

Recall that the DFT maps the sequence of real or complex numbers $\{f_0, f_1, \ldots, f_{M-1}\}$ to the sequence of complex numbers

$$F_l = DFT[f](l) = \sum_{k=0}^{M-1} f_k e^{2\pi jkl/M}, 0 \le l \le M-1,$$

where j is the imaginary unit. Inverse DFT recovers the sequence of f_k's exactly from $\{F_0, F_1, \ldots, F_{M-1}\}$. The correspondent formula reads:

$$f_k = iDFT[F](k) = \frac{1}{M}\sum_{l=0}^{M-1} F_l e^{-2\pi jkl/M}, 0 \le k \le M-1.$$

Technical details on the implementation of the Fast Fourier Transform algorithm to efficiently calculate DFT and iDFT can be found in [41].

D. Implementing ANN to find the coefficients of polynomial factors

According to the universal approximation theorem of Cybenko [21], a feedforward neural network with a single hidden layer can approximate any continuous function of many variables with any precision. The number of neurons in the hidden layer is chosen empirically so that the factorization accuracy has practical significance.

In this paper, we implement a feedforward neural network with one hidden layer. The number of neurons in the input layer is equal to the number of coefficients of the polynomial for which factorization is performed. We trained artificial neural networks to factor polynomials of the second ($M = 2$), sixth ($M = 8$) and 254th degrees ($M = 256$). Thus, given M, the target degree of the polynomial is $M-2$, and the number of coefficients is $M-1$.

The number of neurons in the output layer corresponds to the number of coefficients of the factors. Given M, we calculate $\frac{M}{2}$ coefficients for each factor. The sum of the coefficients of the factors must be equal to one according to the conditions (29), (31) formulated in Section III-B. To strictly comply with this condition, the smallest coefficients of the polynomials $q(x), r(x)$ $(\alpha_0,\ \beta_{M/2-1})$ are calculated using the following formulas:

$$\alpha_0 = 1 - \sum_{i=1}^{M/2-1} \alpha_i,$$

$$\beta_0 = 1 - \sum_{i=0}^{\frac{M}{2}-1} \beta_i.$$

In table I we present the parameters of our neural networks for factoring polynomials of the second, sixth, and 254th degrees.

Table I
PARAMETERS OF NEURAL NETWORKS FOR FACTORIZATION OF POLYNOMIALS OF THE 2ND, 6TH AND 254TH DEGREES

DP	NI	NH	NO
2	3	8	2
6	7	1024	6
254	255	2048	254

DP is the degree of the factorizable polynomial
NI is the number of neurons in the input layer.
NH is the number of neurons in the hidden layer.
NO is the number of neurons in the output layer.

In this research, neural networks for polynomial factorization are implemented using the Tensorflow library for Python programming language.

E. Loss function

As we already mentioned, two polynomials of the nth degree are identically equal if their values coincide at $n+1$ different points. Using this property, in the problem of factoring an arbitrary polynomial of degree $M-2$, we must evaluate the values of the original and factorized polynomials at M points equal to the distinct Mth roots of unity. Considering that M is a power of two, we can effectively implement the calculation of the required values by using the Fast Fourier transform algorithm [41]. This approach appears to be more efficient than the standard MSE (mean square error) loss function.

To train the network, we propose our custom loss function for factoring a second-degree polynomial that is constructed as follows. Let

$$\begin{aligned} A &= [1-\alpha_1, \alpha_1, 0, 0]; \\ B &= [\beta_0, 1-\beta_0, 0, 0]; \\ C &= [p_0, p_1, p_2, 0], \end{aligned}$$

where p_0, p_1, p_2 are the known values of the coefficients of the quadratic trinomial, and α_1, β_0 are the predicted values of the coefficients of the linear factors.

We define the vector D as follows:

$$D = DFT\left(A\right) \cdot DFT\left(B\right) - DFT\left(C\right).$$

The loss function named CLOSS2 returns the mean sum of squares of the real and imaginary parts of all components of the vector D:

$$\text{CLOSS2} = \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{M}\left(\left(\Re D_i^k\right)^2 + \left(\Im D_i^k\right)^2\right),$$

where N is the number of samples in the training set. Figure 2 shows the values of the loss function CLOSS2 in training a neural network to factorize a second-degree polynomial.

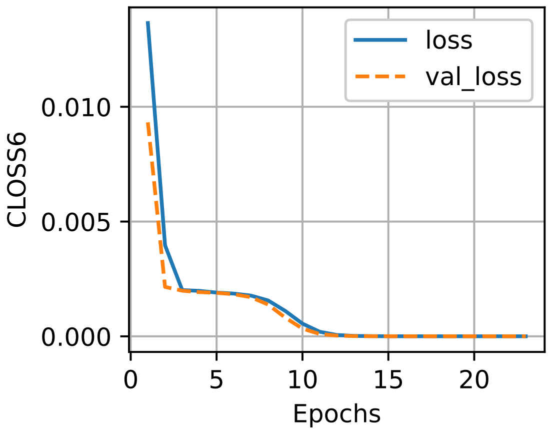

Figure 2. Values of the loss function CLOSS2 in training a neural network to factorize a polynomial of degree 2 by epochs

loss denotes values of the loss function on training data
val_loss denotes values of the loss function on validation data

When factorizing a 6th degree polynomial, the vectors A, B, C are constructed as follows:

$$A = [1-\sum_{j=1}^{3}\alpha_j, \alpha_1, \alpha_2, \alpha_3, 0, 0, 0, 0]; \quad (38)$$

$$B = [\beta_0, \beta_1, \beta_2, 1-\sum_{k=0}^{2}\beta_k, 0, 0, 0, 0]; \quad (39)$$

$$C = [p_0, p_1, p_2, p_3, p_4, p_5, p_6, 0], \quad (40)$$

where $p_i, i = \overline{0,6}$ are known values of the coefficients of a 6th degree polynomial, $\alpha_j(j = \overline{1,3}), \beta_k(= \overline{0,2})$ are predicted coefficients values for factors.

We introduce the penalties PA and PB in the CLOSS6 loss function for violating the conditions of monotonic increase of the sequence $\alpha_j(j = \overline{0,3})$ and monotonic decrease of the sequence $\beta_k(= \overline{0,3})$ as follows:

$$\begin{aligned} PA &= -\min(0, \alpha_1-\alpha_0, \alpha_2-\alpha_1, \alpha_3-\alpha_2); \\ PB &= -\min(0, \beta_0-\beta_1, \beta_1-\beta_2, \beta_2-\beta_3). \end{aligned}$$

Then the loss function for training a neural network to factor a 6th degree polynomial will have the form:

$$\begin{aligned} \text{CLOSS6} &= \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{M}\left(\left(\Re D_i^k\right)^2 + \left(\Im D_i^k\right)^2\right) + \\ &+ \ PA + PB. \end{aligned}$$

Figure 3 shows the dynamics of the CLOSS6 values during neural network training to factorize a sixth-degree polynomial.

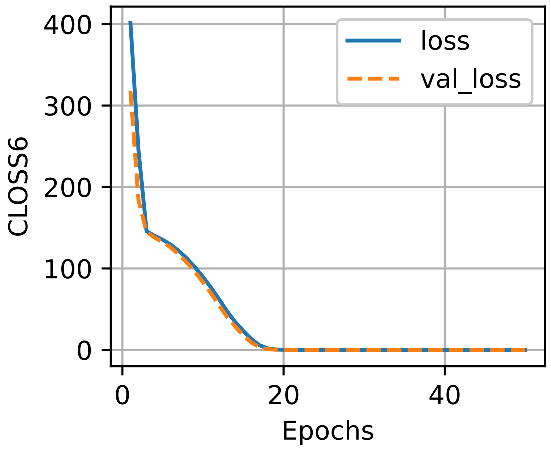

Figure 3. CLOSS6 values during neural network training for factorization of a polynomial of degree 6

loss denotes values of the loss function on training data
val_loss denotes values of the loss function on validation data

When factorizing a polynomial of degree 254, the vectors A, B, C are constructed in a similar way. Along with the penalties for violating the conditions on monotonic increase (30) and decrease (32) of the sequences of the coefficients of factors, we introduce the penalties for violating the conditions (29), (31) as follows:

$$\begin{aligned} PA &= -\min(0, \alpha_1-\alpha_0, \alpha_2-\alpha_1, \ldots, \alpha_{\frac{M}{2}-1}-\alpha_{\frac{M}{2}-2}); \\ PB &= -\min(0, \beta_0-\beta_1, \beta_1-\beta_2, \ldots, \beta_{\frac{M}{2}-2}-\beta_{\frac{M}{2}-1}); \\ PC &= -\min(0, 1-\sum_{j=0}^{\frac{M}{2}-2}\alpha_j, \alpha_0, \alpha_1, \ldots, \alpha_{\frac{M}{2}-2}); \\ PD &= -\min(0, \beta_0, \beta_1, \ldots, \ \beta_{\frac{M}{2}-2}, 1-\sum_{k=0}^{\frac{M}{2}-2}\beta_k); \\ PE &= \left(1-\sum_{k=0}^{\frac{M}{2}-1}\alpha_k\right)^2; \\ PF &= \left(1-\sum_{k=0}^{\frac{M}{2}-1}\beta_k\right)^2. \end{aligned}$$

Next, we sum all penalties into a total penalty P:

$$P = PA + PB + PC + PD + PE + PF.$$

Let us introduce the penalty significance coefficient $\lambda \geq 10^3$ to increase the weight of penalties. Then our custom loss function for training ANNs to factor a polynomial of an arbitrary degree will have the form:

$$\text{CLOSS} = \frac{1}{N}\sum_{k=1}^{N}\sum_{i=1}^{M}\left(\left(\Re D_i^k\right)^2 + \left(\Im D_i^k\right)^2\right) + \lambda \cdot P. \quad (41)$$

Finally, Figure 4 presents the values of the loss function (41) during training a neural network to factor a 254th degree polynomial.

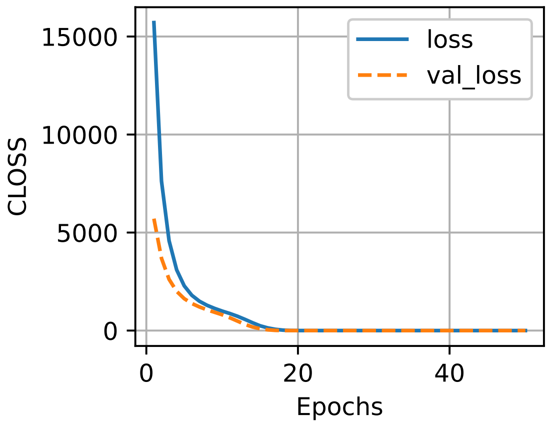

Figure 4. CLOSS values in training a neural network to factorize a polynomial of degree 254

loss denotes values of the loss function on training data
val_loss denotes values of the loss function on validation data

IV. Numerical experiments

A. Examples of the factorization polynomials using neural networks

In our numerical experiments, we implemented and trained three feedforward neural networks with one hidden layer to factorize polynomials of degree 2, degree 6, and degree 254.

The neural networks were trained on more than 1,000,000 samples of input polynomials suitable for factorization. Characteristics of the computer on which the experiment was carried out: Intel(R) Core(TM) i7-3630QM CPU 2.40GHz, RAM 16.0 GB.

In table II, we demonstrate the results of estimating the precision of factorization of polynomials of the second, sixth, and 254th degrees with respect to the number of neurons in the hidden layer.

Table II
Results of training neural networks to factor polynomials of 2nd, 6th and 254th degrees

Degree	HNN	CLOSS	training time, hrs	prediction time, sec
2	8	$3.45 \cdot 10^{-6}$	6	0.0434
6	1024	$6.31 \cdot 10^{-6}$	19	0.0581
254	2048	$4.43 \cdot 10^{-4}$	82	0.0633

HNN - hidden neurons number
CLOSS - the custom loss function

The training time of a neural network increases predictably with the degree of the factored polynomial. Most of the time is spent generating the input data for the training epochs and calculating the loss function, where the forward and inverse discrete Fourier transforms are used. At the same time, the time of factorization of a polynomial by a trained neural network does not increase significantly and takes fractions of a second with an accuracy sufficient to price barrier options.

A series of factorizations of various 6th-degree polynomials was carried out using a trained neural network. For 1000 factorizations, we detected only one violation of the condition of increasing the sequence $\{\alpha_k\}$, 11 violations of the condition of decreasing the sequence $\{\beta_k\}$, and 3 cases of violation of the positivity of the coefficients of the factors.

The difference between the values of the original and factorized polynomials at points equal to complex roots of unity of degree 8, per 1000 calculations are

- maximal deviation – 0.0013;
- minimal deviation – 0.00019;
- average deviation – 0.00036.

Consider an example of factorization of an arbitrary 6th degree polynomial that satisfies (28) by our trained neural network. For example, let the initial polynomial be specified by the following set of coefficients $\{p_k\}_{k=0}^{k=6}$:

$$[0.0787, 0.1372, 0.2349, 0.3236, 0.1714, 0.0443, 0.0096].$$

As the result of the test of our neural network, we obtain the following coefficients of the factors:

$$\alpha_0 = 0.1291, \alpha_1 = 0.1696, \alpha_2 = 0.2994, \alpha_3 = 0.4018;$$
$$\beta_0 = 0.5228, \beta_1 = 0.3526, \beta_2 = 0.0975, \beta_3 = 0.0269.$$

In this case, the sequence of coefficients $\{\alpha_k\}_{k=0}^{k=3}$ increases and in the sum gives 1, and the sequence $\{\beta_k\}_{k=0}^{k=3}$ decreases, the sum of the coefficients also equals 1.

According to the formulas (38)-(40), we represent the coefficients $\{\alpha_k\}_{k=0}^{k=3}$, $\{\beta_k\}_{k=0}^{k=3}$, $\{p_k\}_{k=0}^{k=6}$ as the vectors A, B, C, respectively. The coefficients for missing degrees are set equal to 0. Let us calculate the values of the original polynomial and the product of factors predicted at every eighth root of unity using the fast Fourier transform algorithm.

$$\begin{aligned} DFT(C) \;=\; & [1.0 + 0.j, -0.2557 - 0.5198j, \\ & 0.0055 + 0.1419j, 0.0704 - 0.0692j, \\ & -0.0105 + 0.j, 0.0704 + 0.0692j, \\ & 0.0055 - 0.1419j, -0.2557 + 0.5198j]. \end{aligned}$$

$$\begin{aligned} DFT(A) \cdot DFT(B) = [& 0.9999 + 0.j, -0.2838 - 0.5170j, \\ & 0.0032 + 0.1542j, 0.0679 - 0.0807j, \\ & -0.0344 + 0.j, 0.0679 + 0.0807j, \\ & 0.0032 - 0.1542j, -0.2838 + 0.5170j]. \end{aligned}$$

Compare the difference between the values of the product of the factors and the original polynomial at all the points equal to eighth roots of unity:

$$\begin{aligned} & DFT(A) \cdot DFT(B) - DFT(C) = \\ & [-2.9802 \cdot 10^{-8} + 0.j, -0.0281 + 0.0027j, \\ & -0.0024 + 0.0123j, -0.0025 - 0.0114j, \\ & -0.0239 + 0.j, -0.0025 + 0.0114j, \\ & -0.0024 - 0.0123j, -0.0281 - 0.0027j]. \end{aligned}$$

Calculating the average deviation of the values of the product from the original polynomial over all the indicated points, we obtain 0.00034. This precision is enough to apply this approximation to calculate quantities similar to (8) to obtain reasonable results.

B. Applications of neural networks to Wiener-Hopf factorization in Lévy models

In this subsection, we compare the performance of the suggested factorization with artificial neural networks (ANN factorization, in short) and the approximate Wiener-Hopf factorization described in [5] using the example of lattice Lévy processes.

As a basic example, we consider the down-and-out put option with the strike price $K = 110$, the barrier $H = 100$ from below, and the time to expiration $T = 0.5$. To illustrate our method, we model the asset price S_t as $H\exp(X_t)$, where X_t is a compound Poisson model with a binomial distribution of jumps (see Example 3) and the intensity parameter $\lambda = 10$, the size of jumps $d = 0.01$, and probabilities $p_r = \frac{1}{2}$ and $p_l = \frac{1}{2}$. For simplicity, we choose the instantaneous interest rate $r = 0$, time to expiry $T = 0.5$ year, and define $q = N/T$, where $N = 10$ - the number of time steps. Since $\psi(-i) \approx 0$, we can assume that the equivalent martingale measure condition is satisfied (for clarifications, we refer the reader to [9]).

In the example, we coded the algorithm for the ANN factorization described in Section III.

First, we need to approximate the characteristic function $q(q + \psi(\xi))^{-1}$ for $q = 20$ with the Fourier series by formula (13), with the coefficients p_k defined by (22). To find p_k efficiently, we utilize the discrete Fourier transform by approximating the integral in (22) with the trapezoid rule as follows. Define the partition of the frequency domain $[-\pi/d, \pi/d]$ by points $\xi_l = -\frac{\pi}{d} + \frac{2\pi l}{dM}$, $l = 0, \dots M$. Denote by $\Delta\xi$ the length of each subinterval in $[-\pi/d, \pi/d]$ specified by the partition. Then we have in (22) for $-M/2 \le k < M/2$

$$\begin{aligned} p_k &\approx \frac{d}{2\pi}\sum_{l=0}^{M}\delta_l \frac{q}{q+\psi(\xi_l)}e^{-j\xi_l kd}\cdot\Delta\xi \\ &= \frac{d}{2\pi}\sum_{l=0}^{M}\delta_l \frac{q}{q+\psi(\xi_l)}e^{-j(-\frac{\pi}{d}+\frac{2\pi l}{dM})kd}\cdot\frac{2\pi}{dM} \qquad (42)\\ &= e^{j\pi k}\frac{1}{M}\sum_{l=0}^{M}\delta_l \frac{q}{q+\psi(\xi_l)}e^{-j\frac{2\pi lk}{M}} \\ &= e^{j\pi k}\frac{1}{M}\sum_{l=0}^{M}\delta_l \frac{q}{q+\psi(\xi_l)}e^{-j\frac{2\pi l(k+M/2)}{M}}e^{j\frac{2\pi lM/2}{M}} \\ &= e^{j\pi k}\frac{1}{M}\sum_{l=0}^{M}\delta_l \frac{q}{q+\psi(\xi_l)}e^{j\pi l}e^{-j\frac{2\pi l(k+M/2)}{M}}, \end{aligned}$$

where $\delta_l = 1$ for $l = 1, ..., M-1$, and $\delta_0 = \delta_M = 0.5$ specify the chosen quadrature rule.

According to our definition of iDFT, we obtain the approximate formula for p_k:

$$p_k = (-1)^k \cdot iDFT[P](k + M/2), -M/2 \le k < M/2, \qquad (43)$$

where

$$\begin{aligned} P &= \{P_0, P_1, \dots, P_{M-1}\}, \\ P_0 &= 0.5\left(\frac{q}{q+\psi(\xi_0)} + \frac{q}{q+\psi(\xi_M)}\right), \\ P_l &= (-1)^l\frac{q}{q+\psi(\xi_l)}, 1 \le k \le M-1. \end{aligned}$$

Set

$$\begin{aligned} c_0 &= 0, \\ c_k &= \frac{d}{2\pi}\int_{-\pi/d}^{\pi/d}\ln\frac{q}{q+\psi(\xi)}e^{-j\xi kd}d\xi, k \ne 0. \qquad (44) \end{aligned}$$

Then $\exp(\Psi(\xi))$ approximates the function $\frac{q}{q+\psi(\xi)}$, where

$$\Psi(\xi) = \sum_{k=-M/2}^{M/2-1} c_k(e^{j\xi kd} - 1).$$

From the construction of $\Psi(\xi)$ we can obtain the approximate Wiener-Hopf factorization

$$\phi_q^+(\xi) \approx \exp(\Psi^+(\xi)), \phi_q^-(\xi) \approx \exp(\Psi^-(\xi)), \qquad (45)$$

where

$$\begin{aligned} \Psi^+(\xi) &= \sum_{k=1}^{M/2-1} c_k(e^{j\xi kd} - 1), \qquad (46)\\ \Psi^-(\xi) &= \sum_{k=-M/2}^{-1} c_k(e^{j\xi kd} - 1). \qquad (47) \end{aligned}$$

Similarly to (43), we approximate the coefficients c_k:

$$c_k = (-1)^k \cdot iDFT[C](k + M/2), -M/2 \le k < M/2, \qquad (48)$$

where

$$\begin{aligned} C &= \{C_0, C_1, \dots, C_{M-1}\}, \\ C_0 &= 0.5\ln\frac{q^2}{(q+\psi(\xi_0))(q+\psi(\xi_M))}, \\ C_l &= (-1)^l\ln\frac{q}{q+\psi(\xi_l)}, 1 \le k \le M-1. \end{aligned}$$

To approximate the values of factors $\phi_q^+(\xi)$ and $\phi_q^-(\xi)$ at the points ξ_l, $l = 0, \dots M$, we utilize the direct discrete Fourier transform. Using (46), we obtain for $0 \le l < M$

$$\begin{aligned} \Psi^+(\xi_l) &= \sum_{k=1}^{M/2-1} c_k(e^{j\xi_l kd} - 1) \\ &= -\sum_{k=1}^{M/2-1} c_k + \sum_{k=1}^{M/2-1} c_k e^{j(-\frac{\pi}{d}+\frac{2\pi l}{dM})kd} \\ &= -\sum_{k=1}^{M/2-1} c_k + \sum_{k=1}^{M/2-1} e^{-j\pi k}c_k e^{j\frac{2\pi lk}{M}}. \end{aligned}$$

Analogously, we have for $0 \le l < M$

$$\begin{aligned} \Psi^-(\xi_l) &= \sum_{k=-M/2}^{-1} e^{-j\pi k}c_k e^{j\frac{2\pi lk}{M}} - \sum_{k=-M/2}^{-1} c_k \\ &= \sum_{k=-M/2}^{-1} e^{-j\pi(k+M)}c_k e^{j\frac{2\pi l(k+M)}{M}} - \sum_{k=-M/2}^{-1} c_k \\ &= \sum_{m=M/2}^{M-1} e^{-j\pi m}c_{m-M} e^{j\frac{2\pi lm}{M}} - \sum_{k=-M/2}^{-1} c_k. \end{aligned}$$

Taking into account the formula for DFT, we write

$$\Psi^+(\xi_l) = DFT[c^+](l), 0 \le l < M, \quad (49)$$

where

$$\begin{aligned} c^+ &= \{c_0^+, c_1^+, \ldots, c_{M-1}^+\}, \\ c_0^+ &= -\sum_{k=1}^{M/2-1} c_k, \\ c_m^+ &= (-1)^m c_{m+M/2}, 0 < m \le M/2-1, \\ c_m^+ &= 0, M/2 \le m \le M-1; \end{aligned}$$

and

$$\Psi^-(\xi_l) = DFT[c^-](l), 0 \le l < M, \quad (50)$$

where

$$\begin{aligned} c^- &= \{c_0^-, c_1^-, \ldots, c_{M-1}^-\}, \\ c_0^- &= -\sum_{k=-M/2}^{-1} c_k, \\ c_m^- &= 0, 0 < m \le M/2-1, \\ c_m^- &= (-1)^m c_{m-M}, M/2 \le m \le M-1; \end{aligned}$$

Now, we can approximate the coefficients p_k^+ and p_k^- in (15) and (17) as follows:

$$\begin{aligned} p_k^+ &= (-1)^k \cdot iDFT[P^+](k+M/2), 0 \le k < M/2, \\ p_k^+ &= 0, M/2 \le k < M, \quad (51) \\ p_k^- &= (-1)^k \cdot iDFT[P^-](M/2-k), 0 \le k < M/2, \\ p_k^- &= 0, M/2 \le k < M. \quad (52) \end{aligned}$$

where

$$\begin{aligned} P^+ &= \{P_0^+, P_1^+, \ldots, P_{M-1}^+\}, \\ P_l^+ &= (-1)^l \Psi^+(\xi_l), 0 \le k \le M-1; \\ P^- &= \{P_0^+, P_1^+, \ldots, P_{M-1}^+\}, \\ P_l^- &= (-1)^l \Psi^-(\xi_l), 0 \le k \le M-1; \end{aligned}$$

In our numerical experiment, we found factors $\phi_q^+(\xi)$ and $\phi_q^-(\xi)$ analytically (see (45)) and using our trained neural network. Table III presents the comparison of the values of the function $\frac{q}{q+\psi(\xi)}$ and the product of the approximate factors $e^{\Psi^+(\xi)}$ and $e^{\Psi^-(\xi)}$ calculated at the points ξ_l using (45).

Table III
COMPARISON OF THE VALUES OF THE FUNCTION $q/(q+\psi(\xi))$ AND THE PRODUCT OF THE APPROXIMATE FACTORS

ξ_l	$q/(q+\psi(\xi))$	$e^{\Psi^+(\xi)} \cdot e^{\Psi^-(\xi)}$	$\Delta\Psi$
314.1593	0.5	0.4998	0.0002
235.6194	0.5395	0.5397	0.0002
157.0796	0.6666	0.6664	0.0002
78.5398	0.8723	0.8726	0.0003
0.0	1	0.9996	0.0004
-78.5398	0.8723	0.8726	0.0003
-157.0796	0.6666	0.6664	0.0002
-235.6194	0.5395	0.5397	0.0002

$\Delta\Psi = \left| q/(q+\psi(\xi)) - e^{\Psi^+(\xi)} \cdot e^{\Psi^-(\xi)} \right|$

Table IV
RESULTS OF FACTORIZATION OF THE POLYNOMIAL SPECIFIED BY THE COEFFICIENTS p_k USING OUR TRAINED NEURAL NETWORK

ξ	C	$A \cdot B$	ΔDFT
1	0.99+0.j	1.0+0.j	0.0012
$\exp(-j\frac{\pi}{4})$	-0.62-0.62j	-0.595-0.59j	0.0321
$\exp(-j\frac{\pi}{2})$	0.0+0.67j	-0.0-0.61j	0.0599
$\exp(-j\frac{3\pi}{4})$	0.38-0.38j	0.365-0.37j	0.0242
-1	-0.49+0.j	-0.505+0.j	0.0061
$\exp(-j\frac{5\pi}{4})$	0.38+0.38j	0.365+0.37j	0.0242
$\exp(-j\frac{3\pi}{2})$	0.0-0.67j	-0.0-0.61j	0.0599
$\exp(-j\frac{7\pi}{4})$	-0.62+0.62j	-0.595+0.59j	0.0321

$A = DFT([\hat{p}^+])$, $B = DFT([\hat{p}^-])$, $C = DFT([p])$ $D = A \cdot B - C$
$\Delta DFT = \sqrt{\sum_{i=1}^{M} \left((\Re D_i)^2 + (\Im D_i)^2 \right)}$

Table V
COEFFICIENTS p_k^-, p_k^+ USING APPROXIMATE FORMULAS (51)-(52) VS COEFFICIENTS $\hat{p}_k^-$, $\hat{p}_k^+$ PREDICTED USING THE NEURAL NETWORK

k	p_k	p_k^-	p_k^+	$\hat{p}_k^-$	$\hat{p}_k^+$
0	0.0037	0.8282	0.8286	0.8111	0.8108
1	0.0208	0.1421	0.1422	0.1393	0.1391
2	0.1213	0.0244	0.0244	0.0444	0.0443
3	0.7071	0.0042	0.0042	0.0053	0.0057
4	0.1213	0.0	0.0	0.0	0.0
5	0.0208	0.0	0.0	0.0	0.0
6	0.0037	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0

The comparison of the values of the polynomial specified by the coefficients p_k with the product of the factors predicted by our trained neural network for $q = 20$ and $p_r = \frac{1}{2}$ and $p_l = \frac{1}{2}$ is presented in Table IV in the primitive eighth roots of unity.

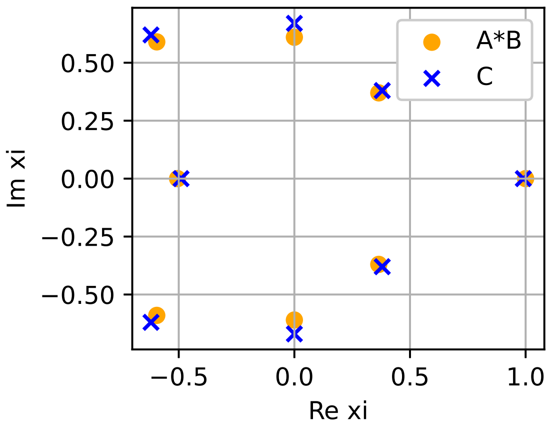

Figure 5. Approximation error of the polynomial C specified by the coefficients p_k with the product of factors A and B calculated using the neural network

The values of the coefficients p_k^-, p_k^+ calculated using approximate formulas (51)-(52) and the coefficients $\hat{p}_k^-$, $\hat{p}_k^+$ predicted using our trained neural network presented in Table V.

Analogously, we compare the values of the polynomial

Table VI
RESULTS OF FACTORIZATION OF THE POLYNOMIAL SPECIFIED BY THE COEFFICIENTS p_k USING THE FORMULAS OF THE APPROXIMATE FACTORIZATION (45)

ξ	C	$A \cdot B$	ΔDFT
1	0.99+0.j	0.998+0.j	0.0004
$\exp(-j\frac{\pi}{4})$	-0.62-0.62j	-0.618-0.62j	0.0003
$\exp(-j\frac{\pi}{2})$	0.0+0.67j	0.0+0.66j	0.0001
$\exp(-j\frac{3\pi}{4})$	0.38-0.38j	0.382-0.38j	0.0003
-1	-0.49+0.j	-0.499+0.j	0.0003
$\exp(-j\frac{5\pi}{4})$	0.38+0.38j	0.382+0.38j	0.0003
$\exp(-j\frac{3\pi}{2})$	0.0-0.67j	-0.0-0.66j	0.0001
$\exp(-j\frac{7\pi}{4})$	-0.62+0.62j	-0.618+0.62j	0.0003

$A = DFT([p^+])$, $B = DFT([p^-])$, $C = DFT([p])$, $D = A \cdot B - C$
$\Delta DFT = \sqrt{\sum_{i=1}^{M} \left((\Re D_i)^2 + (\Im D_i)^2 \right)}$

C specified by the coefficients p_k with the product of the factors A and B calculated by the approximate formulas (45) in Table VI in the primitive eighth roots of unity.

The graphs of the approximation error of the polynomial specified by the coefficients p_k with the product of factors calculated using our trained neural network and by the formula (45), in the primitive eighth roots of unity are presented in Figure 5 and Figure 6, respectively.

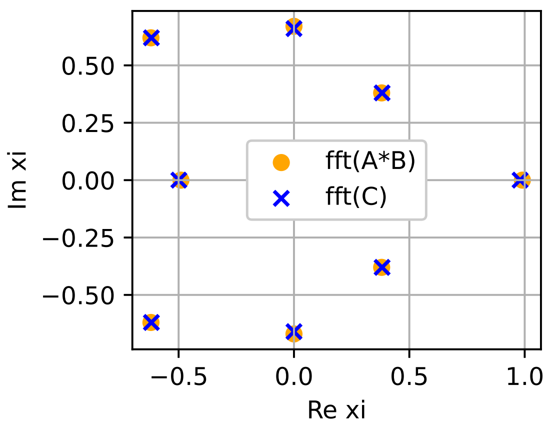

Figure 6. Approximation error of the polynomial specified by the coefficients p_k with the product of factors calculated by the formula (45)

Next, we consider the performance of our artificial neural network in pricing barrier options. Set the pay-off function $G(x) = \max\{0, K - H\exp(x)\}$. Let $V_q(x)$ and $\hat{V}_q(x)$ be perpetual barrier option prices (7) calculated with formulas (19)-(21) with coefficients p_k^-, p_k^+ obtained by approximate formulas (51)-(52) and predicted using our neural network, respectively. If we repeat the numerical procedure (19)-(21) $N(= 10)$ times replacing $G(x)$ in (20) with $V_q(x)$ (or $\hat{V}_q(x)$) from the previous step, we obtain two approximate values of the down-and-out put prices defined by (5). Let us denote these prices by $V(x, T)$ and $\hat{V}(x, T)$, respectively. The details of this iterative procedure that includes an approximate Wiener-Hopf factorization can be found in [9].

In Table VII, we compare the prices of the barrier options obtained using approximate formulas for the Wiener-Hopf factors against the prices calculated using the Wiener-Hopf factors predicted by our artificial neural network.

Table VII
PRICES OF BARRIER OPTIONS: APPROXIMATE WIENER-HOPF FACTORIZATION VS ANN

x	$V_q(x)$	$\hat{V}_q(x)$	$V(x,T)$	$\hat{V}(x,T)$
0.01	7.274	7.093	2.557	2.277
0.02	7.680	7.486	4.349	3.821
0.03	6.899	6.899	5.128	4.680
0.04	5.906	5.915	5.079	4.796
0.05	4.862	4.869	4.510	4.387
0.06	3.807	3.812	3.691	3.681
0.07	2.744	2.747	2.798	2.861
0.08	1.682	1.694	1.948	2.059
0.09	0.685	0.712	1.227	1.362
0.1	0.115	0.143	0.691	0.824
0.11	0.018	0.029	0.347	0.454
0.12	0.002	0.003	0.155	0.228
0.13	0.000	0.000	0.063	0.104
0.14	0.000	0.000	0.023	0.043
0.15	0.000	0.000	0.007	0.016

As we can see from the results presented in Table VII, the perpetual barrier option prices $V_q(x)$ and $\hat{V}_q(x)$ are in agreement, especially for x in the range $0.03 - 0.08$. In the case of the approximate prices for down-and-out puts, we observe less agreement between prices since the approximation error of our neural network accumulates during the iterations. We conclude that our artificial neural network may perform well in computing expectations (7), but for pricing options (5) and (6) we need further training of our ANN by increasing the number of neurons in the hidden layer or increasing the number of hidden layers.

V. CONCLUSION

In the paper, an approach to modeling the values of the Wiener-Hopf factors has been developed using feedforward neural networks with one hidden layer. The neural network approximates the coefficients of two polynomial factors based on a given vector of coefficients of the factorized polynomial. The following conditions are imposed on the input sequence of coefficients: the number of elements of the sequence is one less than a given power of two, the sum of all coefficients is equal to one, before the middle index, the sequence increases, and after the index it decreases. The following restrictions are imposed on the output sequences: the sequence of coefficients of the first polynomial factor increases, the sequence of the second polynomial factor decreases, and the sum of the coefficients for each factor is equal to one. Such a normalization allows us to interpret such coefficients as the probabilities of suitable random variables. The neural network is implemented using the TensorFlow library for the Python programming language. The number of input neurons corresponds to the number of coefficients of the factorized polynomial and is greater by one than the degree of the polynomial. The number of neurons in the output corresponds to the number of coefficients of the factor polynomials. The number of coefficients for each factor is equal to half of the power of two closest to the degree of the original polynomial. The last coefficient of each factor is calculated as the difference between the sum of the predicted coefficients and one. The number of neurons in the hidden layer is empirically selected in order to obtain a practically significant accuracy of factorization. According to the Cybenko theorem [21], the larger the number of neurons in the hidden layer, the better the approximation.

We designed a training data generator and a generalized loss function that are suitable for training our neural network to factorize a polynomial of an arbitrary degree. The loss function compares the values of the factorizable polynomial of degree $M-2$, (M is the power of two) and the product of factors at points that are equal to the roots of degree M of unity. For a given training set, such a choice of points allows one to efficiently calculate the values of the specified polynomials and the coefficients of the polynomials using the Fast Fourier Transform (direct) and (inverse) algorithms, respectively. We demonstrated the performance of our approach in numerical experiments on the factorization of polynomials of the second, sixth, and 254th degrees into the product of two polynomial factors of degrees one, three, and 127 respectively.

In particular, for second-degree polynomials, we achieved factorization accuracy in 3.45e-06 with 8 neurons in the hidden layer. For sixth-degree polynomials, the network accuracy is 6.31e-06 with 1024 neurons in the hidden layer. For 254th-degree polynomials, the accuracy is 8.99e-05 with 2048 neurons. The time to calculate the factors by the trained neural network is on average 0.05 seconds. Thus, the neural network for Wiener-Hopf factorization can quickly generate the required set of coefficients for factors, which can then be used to price barrier options using the Wiener-Hopf method.

References

[1] R. Cont and P. Tankov, *Financial Modelling with Jump Processes*, 2nd ed. Chapman & Hall/CRC, USA: Boca Raton, FL, 2008.

[2] Z. Xu and X. Jia, "The calibration of volatility for option pricing models with jump diffusion processes," *Applicable Analysis*, vol. 98, no. 4, pp. 810-827, 2019.

[3] C. Wei, "Parameter Estimation for Discretely Observed Vasicek Model Driven by Small Levy Noises," *IAENG International Journal of Applied Mathematics*, vol. 48, no.2, pp. 118-122, 2018.

[4] A. Kuznetsov, A. E. Kyprianou, J. C. Pardo and K. van Schaik, "A Wiener-Hopf Monte Carlo Simulation Technique for Lévy Processes," *Ann. Appl. Probab.*, vol. 21, no. 6, pp. 2171-2190. 2011.

[5] O. E. Kudryavtsev, "Approximate Wiener–Hopf Factorization and Monte Carlo Methods for Lévy Processes," *Theory of Probability & Its Applications*, vol. 64, no. 2, pp. 186-208, 2019.

[6] Y. Yang, J. Ma, and Y. Liang, "The Research on the Calculation of Barrier Options under Stochastic Volatility Models Based on the Exact Simulation," *IAENG International Journal of Applied Mathematics*, vol. 48, no.3, pp. 349-361, 2018.

[7] K. R. Jackson, S. Jaimungal and V. Surkov, "Fourier Space Time-Stepping for Option Pricing with Lévy Models," *Journal of Computational Finance*, vol. 12, no. 2, pp. 1-28, 2008

[8] J. L. Kirkby, "Robust Option Pricing with Characteristic Functions and the B-spline Order of Density Projection", *Journal of Computational Finance*, vol. 21, no. 2, pp. 61-100, 2017.

[9] O. Kudryavtsev and S. Levendorskiĭ, "Fast and Accurate Pricing of Barrier Options under Lévy Processes," *Finance and Stochastics*, vol. 13, no. 4, pp. 531-562, 2009.

[10] O. Kudryavtsev, P. Luzhetskaya, "The Wiener-Hopf Factorization for Pricing Options Made Easy," *Engineering Letters*, vol. 28, no. 4, pp. 1310-1317, 2020.

[11] C. E. Phelan, D. Marazzina, G. Fusai, G. Germano, "Fluctuation identities with continuous monitoring and their application to price barrier options", *European Journal of Operational Research*, vol. 271, no. 1, pp. 210-223, 2018.

[12] A. Itkin, *Pricing Derivatives Under Levy Models*, Basel: Birkhauser, 2017.

[13] M. J. Ruijter and C. W. Oosterlee, "Numerical Fourier Method and Second-order Taylor Scheme for Backward SDEs in Finance," *Applied Numerical Mathematics*, vol. 103, no. C, pp. 1-26, May. 2016.

[14] R. Brummelhuis and R. T. L. Chan, "A Radial Basis Function Scheme for Option Pricing in Exponential Lévy Models," *Applied Mathematical Finance*, vol. 21, no. 3, pp. 238-269, 2014.

[15] R. Cont and E. Voltchkova, "A Finite Difference Scheme for Option Pricing in Jump Diffusion and Exponential Lévy Models" *SIAM Journal on Numerical Analysis*, vol. 43, no. 4, pp. 1596-1626, 2005.

[16] O. Ye. Kudryavtsev, "An Efficient Numerical Method to Solve a Special Class of Integrodifferential Equations Relating to the Levy Models," *Mathematical Models and Computer Simulations*, vol. 3, no.6, pp. 706-711, 2011.

[17] O. Kudryavtsev, "Advantages of the Laplace Transform Approach in Pricing First Touch Digital Options in Lévy-driven Models," *Boletín de la Sociedad Matemática Mexicana*, vol. 22, no. 2, pp. 711-731, 2016.

[18] M. Boyarchenko and S. Levendorskiĭ, "Prices and Sensitivities of Barrier and First-Touch Digital Options in Levy-Driven Models," *International Journal of Theoretical and Applied Finance*, vol. 12, no. 8, pp. 1125-1170, 2009.

[19] A. T. P. Najafabadi, "A Mixture-Exponential Approximation to Finite and Infinite-time Ruin Probabilities of Compound Poisson Processes," *IAENG International Journal of Applied Mathematics*, vol. 48, no.1, pp. 105-110, 2018.

[20] O Kudryavtsev, "Methods for solving contemporary computational finance problems: applying levy models and machine learning," *Stochastic Modelling & Computational Sciences*, vol. 3, no. 2, 2023.

[21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303-314, 1989.

[22] C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K.V. Zubov, R.B. Supekar, D.J. Skinner, A. Ramadhan, "Universal differential equations for scientific machine learning," *ArXiv, abs/2001.04385*, 2020.

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

[24] C. Michoski, M. Milosavljevic, T. Oliver, D.R. Hatch, "Solving differential equations using deep neural networks," *Neurocomputing*, vol. 399, pp. 193–212, 2020.

[25] J. Huh, "Pricing Options with Exponential Lévy Neural Network," *Expert Systems with Applications*, vol. 127, pp. 128-140, 2019.

[26] S. G. Kou, "A jump-diffusion model for option pricing," *Management Science*, vol 48, pp. 1086-1101, 2002.

[27] P. Carr, H. Geman, D. B. Madan, and M. Yor, "The Fine Structure of Asset Returns: an Empirical Investigation," *Journal of Business*, vol. 75, pp. 305-332, 2002.

[28] Q. Zheng, M. Yang, X. Tian, X. Wang, and D. Wang, "Rethinking the Role of Activation Functions in Deep Convolutional Neural Networks for Image Classification," *Engineering Letters*, vol. 28, no.1, pp. 80-92, 2020.

[29] Y. Hu, X. Zhang, J. Yang, and S. Fu, "A Hybrid Convolutional Neural Network Model Based on Different Evolution for Medical Image Classification," *Engineering Letters*, vol. 30, no.1, pp. 168-177, 2022.

[30] R. Sarno, S. I. Sabilla, D. R. Wijaya, and Hariyanto, "Electronic Nose for Detecting Multilevel Diabetes using Optimized Deep Neural Network," *Engineering Letters*, vol. 28, no.1, pp. 31-42, 2020.

[31] M. A. Aslam, C. Xue, M. Liu, K. Wang, and D. Cui, "Classification and Prediction of Gastric Cancer from Saliva Diagnosis using Artificial Neural Network," *Engineering letters*, vol. 29, no.1, pp. 10-24, 2021.

[32] Melina, Sukono, H. Napitupulu, A. Sambas, A. Murniati, and V. A. Kusumaningtyas, "Artificial Neural Network-Based Machine Learning Approach to Stock Market Prediction Model on the Indonesia Stock Exchange During the COVID-19," *Engineering Letters*, vol. 30, no.3, pp. 988-1000, 2022.

[33] K. Hornik, M. Stinchcombe, H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359-366, 1989.

[34] K. I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural networks*, vol. 2, no. 3, pp. 183-192, 1989.

[35] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information theory*, vol. 39, no. 3, pp. 930-945, 1993.

[36] D. Elbrächter et al., "Deep neural network approximation theory," *IEEE Transactions on Information Theory*, vol. 67, no. 5, pp. 2581-2623, 2021.

[37] B. Adcock, N. Dexter, "The gap between theory and practice in function approximation with deep neural networks," *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 2, pp. 624-655, 2021.

[38] E. Alymova, O. Kudryavtsev, “The application of a neural network and elements of regression analysis in the development of a methodology for effective foreign exchange trading ,” *In: Shiryaev, A.N., Samouylov, K.E., and Kozyrev, D.V. (eds) Recent Developments in Stochastic Methods and Applications. ICSM-5 2020. Springer Proceedings in Mathematics & Statistics, Springer*, vol. 371, pp. 306–317, 2021.

[39] K. Sato, *Lévy Processes and Infinitely Divisible Distributions*, 2nd ed. Cambridge, England: Cambridge University Press, 2013.

[40] S. Z. Levendorskiĭ, “Method of Paired Contours and Pricing Barrier Options and CDS of Long Maturities,” *International Journal of Theoretical and Applied Finance*, vol. 17, no. 5, pp. 1-58, 2014.

[41] W. Press, B. Flannery, S. Teukolsky and W. Vetterling, *Numerical recipes in C: The Art of Scientific Computing*, 3rd ed. Cambridge : Cambridge Univ. Press, 2007.

Elena Alymova holds a degree of Candidate of Technical Sciences (Russian analog of PhD) from Saint Petersburg National Research University of Information Technologies, Mechanics, and Optics (St. Petersburg, Russia). The field of her research interests includes development and training of artificial neural networks, algorithmization and programming of training and test data generators, and numerical methods.

Currently, she is a researcher at InWise Systems, LLC (Rostov-on-Don, Russia) and Associate Professor of the Department of Computer Science and Customs Technologies at the Rostov Branch of the Russian Customs Academy (Rostov-on-Don, Russia).

Dr. Alymova is the author of more than 40 papers on Computer Science.

Oleg Kudryavtsev (M’2020) became a Member (M) of IAENG in 2020. This author holds a degree of Doctor of Science in Physics and Mathematics (Russian analog of a Habilitation Degree) from Central Economics and Mathematics Institute of Russian Academy of Sciences (Moscow, Russia). The field of his research interests includes the development of fast and efficient computational algorithms for pricing path-dependent options and risk estimation in models admitting jumps (numerical Wiener-Hopf factorization, Monte Carlo methods, finite difference schemes, integral transform methods, and machine learning methods).

Currently, he is the Head of the Department of Computer Science and Customs Technologies at the Rostov Branch of the Russian Customs Academy (Rostov-on-Don, Russia) and Research Director at InWise Systems, LLC (Rostov-on-Don, Russia).

Prof. Kudryavtsev is a member of the international research group MathRisk at INRIA (the French National Institute for Research in Computer Science and Control, France) as an expert in Computational Finance and Applied Mathematics. He is the author of more than 150 papers on Applied Mathematics, Computer Science, and Computational Finance including articles published in journals ranked at levels 3 in the Chartered ABS Academic Journal Guide.