BRAIN. Broad Research in Artificial Intelligence and Neuroscience
e-ISSN: 2067-3957 | p-ISSN: 2068-0473
Covered in: Web of Science (ESCI); EBSCO; JERIH PLUS (hkdir.no); IndexCopernicus; Google Scholar; SHERPA/RoMEO; ArticleReach Direct; WorldCat; CrossRef; Peeref; Bridge of Knowledge (mostwiedzy.pl); abcdindex.com; Editage; Ingenta Connect Publication; OALib; scite.ai; Scholar9; Scientific and Technical Information Portal; FID Move; ADVANCED SCIENCES INDEX (European Science Evaluation Center, neredataltics.org); ivySCI; exaly.com; Journal Selector Tool (letpub.com); Citefactor.org; fatcat!; ZDB catalogue; Catalogue SUDOC (abes.fr); OpenAlex; Wikidata; The ISSN Portal; Socolar; KVK-Volltitel (kit.edu)
2025, Volume 16, Issue 2, pages: 474-488.
Submitted: January 26th, 2025 | *Accepted for publication:* April 10th, 2025

Physics-Informed Neural Networks in Pricing Financial Options

Tiberiu Socaciu
Stefan cel Mare University of Suceava, Romania.
tiberiu.socaciu@usm.ro
https://orcid.org/0000-0003-1265-6640

Paul Pașcu
Stefan cel Mare University of Suceava, Romania.
paul.pascu@usm.ro
https://orcid.org/0009-0007-1087-0367

Abstract: *PINNs (Physics-Informed Neural Networks) are neural networks designed to solve Partial Differential Equations (PDEs) by integrating physical knowledge into the learning framework. Constructing a PINN involves defining a neural network to approximate the PDE solution, with the total loss calculated as a combination of the losses associated with the PDE, boundary conditions, initial conditions, and measured data. This concept is employed in practical applications to solve various PDEs, such as the Black-Scholes and Heston equations, which are fundamental in financial option pricing. This approach enables the modelling and pricing of financial options, with the added advantage of parallelising the training process across multiple economic scenarios.*

Keywords: *physics-informed neural networks (PINNs); partial differential equations (PDEs); black-scholes model; heston model; option pricing; stochastic models; financial derivatives; numerical methods; machine learning in finance; computational finance.*

How to cite: Socaciu, T., & Pașcu, P. (2025). Physics-informed neural networks in pricing financial options. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, *16*(2), 474-488. https://doi.org/10.70594/brain/16.2/33

©2025 Published by EduSoft Publishing. This is an open-access article under the CC BY-NC-ND license

1. Introduction

In recent decades, evaluating financial options has become a topic of significant interest for both financial practitioners and researchers. The evaluation of economic options is a fundamental process in financial markets. The options are financial contracts that grant their holder the right, but not the obligation, to buy or sell an underlying asset at a predetermined price on a future date. The price of an option depends on several factors, including the cost of the underlying asset, its volatility, the risk-free interest rate, and the time remaining until expiration (Raissi, Perdikaris, & Karniadakis, 2018; Strauss, 2008).

The mathematical models that describe financial market behaviours are typically expressed through Partial Differential Equations (PDEs), which are used to predict asset prices and market dynamics over time. Among the most well-known mathematical models used for this purpose is the Black-Scholes model, which is applied in the valuation of financial options. However, for more complex options, such as those derived from stochastic models with random volatility, like the Heston model, obtaining analytical solutions becomes extremely challenging, if not impossible. Consequently, numerical methods have become essential in evaluating such financial options. In recent years, a relatively new and innovative approach, known as Physics-Informed Neural Networks (PINNs), has gained attention (Socaciu, 2009; Socaciu, 2016).

2. Computational Finance: from Differential Stochastic Models to PDEs

2.1. Starting points

Computational finance is an interdisciplinary field that combines applied mathematics, statistics, and computer science to model and solve complex financial problems. Over the decades, rapid advancements in financial theory have generated a series of sophisticated mathematical tools for modelling the behaviour of financial markets. Specifically, modern financial models have evolved from simple deterministic models to complex stochastic models, which PDEs describe (Socaciu, 2009; Kloeden & Platen, 1992). These models are used to evaluate complex financial instruments, such as financial options and other derivatives. In this context, this article explores how financial models have evolved from a mathematical and computational perspective, transitioning from stochastic differential models to the resolution of financial PDEs (Patrut & Socaciu, 2016).

2.2. Stochastic financial models

At the core of many methods for valuing financial instruments lie stochastic models, which are used to describe the prices of financial assets based on market uncertainty and volatility. Stochastic models are valuable because they can capture the random evolution of financial markets, taking into account factors such as random volatility, interest rates, and continuously changing market conditions.

One of the most fundamental stochastic models is the *Geometric Brownian Motion* model, used by Black-Scholes for option valuation. The model assumes that the price of an asset follows a random trajectory, with a constant rate of change and random volatility that depends on market factors.

This simple model is typically used to calculate prices for European and American options, where transactions are tied to movements in the capital markets, and volatility is often assumed to be constant. However, to model more complex options, such as those with random volatility or exotic financial derivatives, more advanced and realistic models are required (Socaciu, 2010; Socaciu, 2011).

2.3. Extending stochastic financial models: Random volatility models

A classic example of a model that extends the idea of constant volatility is the Heston model. The Heston model addresses the issue of random volatility, precisely the variability of the volatility of the underlying asset on which the option price is based. In the Heston model, volatility

follows a stochastic process that evolves, and it is described by a system of two differential equations.

2.4. PDE derivation from stochastic models

To solve these stochastic models and evaluate financial options, it is necessary to use PDEs. Among the most important applications of PDEs in finance is the valuation of financial options. Taking the Heston or Black-Scholes models as an example, the stochastic equations that describe the price of the underlying asset can be transformed into a PDE that describes the evolution of the option price with financial variables.

Socaciu (2009) provides an example of the derivation of the Generalised Heston-Garman equation. Socaciu (2016) offers an example of the derivation of a Black-Scholes-like equation for two supports, while Patrut and Socaciu (2016) present an alternative derivation of the Heston equation without the λ characteristic of the market.

3. Solving PDEs

The solution of PDEs with boundary conditions can be achieved through various methods, depending on the complexity of the equation and the form of the boundary conditions. The solutions can be divided into two main categories:

1) Closed-form solutions: These are analytical solutions that can be explicitly expressed using known functions, such as trigonometric functions, exponential functions, or special functions.

2) Numerical solutions: These are approximate solutions obtained through computational algorithms, including classical numerical methods and methods based on Machine Learning (ML).

3.1. Closed-form solutions

These solutions are possible only for special classes of PDEs and boundary conditions. Here are some common methods for obtaining these solutions (see Table 1):

Table 1. Common techniques for obtaining PDEs

No.	Technique	Applicability	Principle
1	The method of separation of variables	The equation must be linear and defined on a region with simple symmetries (e.g., a square, circle, or rectangle).	It is assumed that the solution can be written as a product of one-dimensional functions depending on each variable. For example, for the wave equation in one dimension, the solution can be written as: $u(x,t)=X(x)T(t)$. The initial equation is separated into two Ordinary Differential Equations (ODEs), one for each function. After solving them and applying the boundary conditions, the final solution is obtained.
2	Fourier Transform	Linearisable equations, defined on an infinite or semi-infinite domain.	The Fourier transform is applied to convert the PDE into an ODE in the frequency domain, which can then be solved.
3	Laplace Transform	Differential equations with initial and boundary conditions on semi-infinite domains.	Similar to the Fourier transform, but applied to equations where time is the input variable. The Laplace transform is used to linearise the equation in time, and the obtained solution can be transformed back using the inverse Laplace transform.
4	Green Functions	Linear differential equations on finite domains with homogeneous boundary conditions.	The solution is expressed as a convolution between the Green's function and the source term in the differential equation. Green's functions are fundamental solutions that take into account the effect of the boundary conditions.

3.2. Numeric methods

There is a variety of numerical methods used for solving PDEs. Here is an extended, but not exhaustive, list of methods and techniques frequently used in numerical analysis (see Table 2):

Table 2. Common techniques for solving PDEs

No.	Technique	Principle
1	Finite Difference Method (FDM)	This involves discretising the domain into grid points and using finite difference approximations to replace the derivatives in the PDE. (Richardson, 2010).
2	Finite Element Method (FEM)	This method divides the domain into small elements (triangles, tetrahedra, etc.) and uses interpolation functions to approximate the solution within each element. (Zienkiewicz & Cheung, 1967).
3	Finite Volume Method (FVM)	FVM conserves fluxes at the edges of finite volumes, being commonly used in conservation problems and fluid mechanics (mass transport, volume, energy, etc. (LeVeque, 2004).
4	Spectral Methods	These methods use special functions (e.g., Legendre or Fourier polynomials) to represent the solution of the PDE. They are often very accurate for problems with smooth solutions (Gottlieb, & Orszag, 1977).
5	Variational Optimisation Methods	These methods transform the PDE problem into an optimisation problem of a cost function, which can be solved using numerical methods. (Courant, & Hilbert, 2008).
6	The Collision Point Method (CPM)	This involves using collision points to determine the solution of the PDE, often by transforming the problem into an algebraic form. (Erleben et al., 2005).
7	Monte Carlo simulation methods	Frequently used in financial problems and simulations of stochastic processes, these methods generate solutions through statistical estimation. The first step involves discretising the differential model into a recurrence, using various techniques such as Euler/Euler-Maruyama, Milstein etc. (Kloeden, & Platen 1992).
8	Lagrange and Euler Methods and Hybrid Numerical Methods: ALE (Arbitrary Lagrangian-Eulerian)	These methods are used for fluid dynamics problems, where the evolution of particles over time is tracked. (Ferziger, & Perić, 2002).
9	Time Discretisation Methods	These include methods like Runge-Kutta or the Crank-Nicolson method, which are used to advance the solution in time while maintaining stability and accuracy. (Pang, 2006; Strauss, 2008).
10	Physics-Informed Neural Networks (PINNs)	These methods use neural networks to solve PDEs by learning the solutions directly from the input data and the constraints imposed by the PDE. (Raissi, Perdikaris, & Karniadakis, 2019).
11	Adams-Bashforth Method	An explicit integration method for ODEs, can also be extended to PDEs (Morton, & Mayers, 2005).
12	Optimised Topology Methods	These methods are used to solve PDEs in structural engineering problems and design optimisation (Bendsøe, & Sigmund, 2003).
13	Optimal Control Methods	These methods focus on finding solutions that minimise or maximise a cost function and are often used in dynamic system control problems (Troltzsch, 2010).
15	Network Adaptation Methods	These methods adjust the discretisation based on the characteristics of the solution, such as large gradients or singularities (Hagan et al., 2014).

Each of these methods has advantages and disadvantages, and the choice of the most suitable one depends on the specific nature of the problem, precision requirements, available computational resources, and other contextual considerations. A quasi-exhaustive list of methods can be found in the AMS 2020 classification of mathematical domains, in sections 65M and 65N (American Mathematical Society, 2020).

4. Using Physics-Informed Neural Networks (PINNs) for PDEs

4.1. What are PINNs?

Traditionally, Neural Networks (NNs) are used to learn relationships between inputs and outputs based on labelled data. However, PINNs have a more ambitious goal: they are designed to approximate the solution of a PDE directly. Instead of learning only from data, PINNs are "informed" by the equation that governs the system they model. Thus, PINNs incorporate physical knowledge directly into the neural network through the following mechanisms (Table 3):

Table 3. Mechanisms for physical knowledge incorporation in PINNs.

No.	Mechanism	Explication
1	Formulating the PDE as a constraint	The differential equation is introduced as a constraint on the neural network's loss function.
2	Integrating the boundary and initial conditions	Solutions that do not satisfy these conditions are penalised.
3	Minimising the global error	The loss function contains both terms that penalise deviations from the physical equation and the boundary and initial conditions.

4.2. How to build a PINN?

PINNs are constructed like regular neural networks, but we must take into account the following aspects (Hagan et al., 2014):

a) **Input**: The spatial and temporal coordinates of the point where we want to evaluate the solution, for example (x, t). The spatial coordinate can be scalar, as in the case of the Black-Scholes PDE, or vectorial, as in the case of the Heston PDE. If the spatial coordinate is vectorial, the input will generate an entry in the PINN for each spatial component.

b) **Output**: The approximation of the PDE solution at that point, u(x, t), as a value $\hat{u}(x, t, \theta)$, where θ are the parameters of the network obtained after training.

c) **Loss Function:** contains several terms that reflect not only the observed data but also the physical knowledge in the form of differential equations and boundary conditions, as discussed in the previous paragraph. Thus, we have:

$$\text{Loss} = a \times \text{PDE Loss} + b \times \text{Boundary Loss} + c \times \text{Initial Condition Loss} + d \times \text{Data Loss} \quad (1)$$

where the coefficient vector in the scalar product can be considered as a vector of the inverses of the number of components involved in the calculation of the second factor.

4.3. PDE Loss

This is the main term that constrains the network to satisfy the differential equation. It is obtained by calculating the derivatives of the approximate solution $\hat{\mathbf{u}}$ obtained from the neural network and penalising the deviation from the real PDE equation. If PDEL is the operator of the equation:

$$\text{PDELu} = 0 \quad (2)$$

we can define, for example,

$$\text{PDE Loss} = \sum ((\text{PDEL}\hat{u})(x_i, t_i) - (\text{PDEL}u)(x_i, t_i))^2 = \sum ((\text{PDEL}\hat{u})(x_i, t_i))^2 \quad (3)$$

With the note that in the calculation of PDEL $\hat{\mathbf{u}}$ we will *approximate* the partial derivatives of û using numerical techniques:

4.4. Boundary Loss

This term penalises the network if its solution does not satisfy the boundary conditions. For each boundary point, we have conditions of the form:

$$\text{BL}_i u = 0 \quad (4)$$

thus, boundary loss will be of the form:

$$\text{Boundary Loss} = \sum\sum ((\text{BL}_j\hat{u})(x_i, t_i) - (\text{BL}_j u)(x_i, t_i))^2 = \sum\sum ((\text{BL}_j\hat{u})(x_i, t_i))^2 \quad (5)$$

4.5. Initial Conditions Loss

This term is formulated in the same manner as Boundary Loss. Therefore, if for each point of the initial conditions, we have constraints of the form:

$$\text{ICL}_i u = 0 \quad (6)$$

then

$$\text{Initial Conditions Loss} = \sum\sum ((\text{ICL}_j\hat{u})(x_i, t_i) - (\text{ICL}_i u)(x_i, t_i))^2 = \sum\sum ((\text{ICL}_j\hat{u})(x_i, t_i))^2 \quad (7)$$

4.6. Data Loss (optional)

If we have experimentally or numerically measured data, these can be integrated into the loss function to adjust the solution:

$$\text{Data Loss} = \sum (u(x_i, t_i) - \hat{u}(x_i, t_i))^2 \quad (8)$$

4.7. Total Loss

Considering unit weights, the total loss is calculated as the sum of the losses:

$$\text{Total Loss} = a \times [\sum (\text{PDEL}\hat{u}(x_i, t_i))^2] + b \times [\sum\sum (\text{BL}_j\hat{u}(x_i, t_i))^2] + c \times [\sum\sum (\text{ICL}_j\hat{u}(x_i, t_i))^2] + d \times [\sum (u(x_i, t_i) - \hat{u}(x_i, t_i))^2] \quad (9)$$

The quadratic forms in the components ensure that they have the form of an energy-type function, so we can use it as an objective to minimize in the neural network.

5. Solving financial PDEs using PINNs.

5.1. PINN for Black-Scholes PDE

The *Black-Scholes PDE* (with initial condition and boundary condition) is:

$$u_t + 0{,}5\sigma^2 S^2 u_{SS} + rSu_S - ru = 0 \quad (10.1)$$

$$u(s, T) - \text{payoff}(s) = 0 \quad (10.2)$$

$$u(0, t) = 0, \quad (10.3)$$

where:

a) u(s, t) is *price of derivative based on asset s at timestamp t*;
b) T is *maturity timestamp for derivativ*e;

c) payoff(s) is the *payoff of derivative*.

For **CALL** and **PUT** derivatives with **Strike Price**, **payoff**s are:

$$payoff_{CALL}(s) = \max\{s - \text{Strike Price}, 0\}, \quad (10.4)$$
$$payoff_{PUT}(s) = \max\{\text{Strike Price} - s, 0\}. \quad (10.5)$$

The PINN will take as input the pair (s,t), and the Loss function will be:

$$\text{Total Loss}_{\text{Black-Scholes}} = \frac{1}{|VALUES|}\sum_{(s,t)\in VALUES}(\hat{u}_t(s,t) + 0{,}5\sigma^2S^2\hat{u}_{SS}(s,t) + rS\hat{u}_S(s,t) - r\hat{u}(s,t))^2 +$$
$$+ \frac{1}{|INT|}\sum_{(s,t)\in INT}(u(s,t) - \hat{u}(s,t))^2 +$$
$$+ \frac{1}{|BOUND\ RIGHT|}\sum_{(s,T)\in BOUND\ RIGHT}(\hat{u}(s,T) - payoff(s))^2 +$$
$$+ \frac{1}{|BOUND\ DOWN|}\sum_{(0,t)\in BOUND\ DOWN}\hat{u}(0,t)^2, \quad (11)$$

where:

a) **VALUES** all the values (s, t) with which the model interacts;
b) **BOUND RIGHT** all the values (s, t) from **VALUES** for which t = T; It is recommended that the values of s include all the strike values for which derivatives are traded at maturity Tor possibly for all maturities (see 6);
c) **BOUND DOWN** all the values (s, t) from **VALUES** for which s = 0; It is recommended that the values of t include all maturities for which derivatives with strike s are traded, or possibly for all strikes (see 6);
d) **INT** all from **VALUES** the values inside the domain that are not on the boundary, meaning **VALUES** \ (**BOUND RIGHT** ∪ **BOUND DOWN**).

5.2. PINN for Heston equation

The *Heston PDE* (with initial conditions and boundary conditions) is:

$$u_t + 0{,}5vS^2u_{SS} + \rho\sigma vSu_S + 0{,}5\sigma^2vu_{SS} + rSu_S + [k(\theta - v(t)) - \lambda(S,v,t)]u_v - ru = 0 \quad (12.1)$$
$$u(s,v,T) - payoff(s) = 0 \quad (12.2)$$
$$u(0,v,t) = 0 \quad (12.3)$$
$$u(s,\infty,t) = 0 \quad (12.4)$$
$$u_S(\infty,v,t) - 1 = 0 \quad (12.5)$$
$$u_t(s,0,t) + 0{,}5\sigma^2S^2u_{SS}(s,0,t) + rSu_S(s,0,t) - ru(s,0,t) = 0 \quad (12.6)$$

where:

a) u(s, v, t) is *price of derivative based on asset s at timestamp t on volatility v*;
b) T is *maturity timestamp for derivative*;
c) payoff(s) is the *payoff of derivative* (see 5.1).

The PINN will take the pair (s,t), as input, and the loss function will be:

$$\text{Total Loss}_{\text{Heston}} = \frac{1}{|VALUES|}\sum_{(s,v,t)\in VALUES}(\hat{u}_t(s,v,t) + 0{,}5vS^2\hat{u}_{SS}(s,v,t) + \rho\sigma vS\hat{u}_S(s,v,t) +$$
$$+ 0{,}5\sigma^2v\hat{u}_{SS}(s,v,t) + rS\hat{u}_S(s,v,t) + [k(\theta - v(t)) - \lambda(S,v,t)]\hat{u}_v(s,v,t) -$$

$$- r\hat{u}(s, v, t))^2 +$$
$$+ \frac{1}{| INT |}\sum_{(s, v, t) \in INT} (u(s, t) - \hat{u}(s, t))^2 +$$
$$+ \frac{1}{| BOUND\ S\ ZERO |}\sum_{(s, v, t) \in BOUND\ S\ ZERO} (\hat{u}(0, v, t))^2 +$$
$$+ \frac{1}{| BOUND\ S\ INF |}\sum_{(s, v, t) \in BOUND\ S\ INF} (\hat{u}_S(s, v, t) - 1)^2 +$$
$$+ \frac{1}{| BOUND\ v\ ZERO |}\sum_{(s, v, t) \in BOUND\ v\ ZERO} (\hat{u}_t(s, 0, t) + 0{,}5\sigma^2 S^2 \hat{u}_{SS}(s, 0, t) +$$
$$+ rS\hat{u}_S(s, 0, t) - r\hat{u}(s, 0, t))^2 +$$
$$+ \frac{1}{| BOUND\ v\ INF |}\sum_{(s, v, t) \in BOUND\ v\ INF} (\hat{u}(s, \infty, t))^2 +$$
$$+ \frac{1}{| BOUND\ t\ at\ MATURITY |}\sum_{(s, v, t) \in BOUND\ t\ at\ MATURITY} (\hat{u}(s, v, T) - payoff(s))^2, \quad (13)$$

where:

a) **VALUES** all the values (s, v, t) with which the model interacts;
b) **BOUND S ZERO** all the values (s, v, t) din **VALUES** for which s = 0; the same recommendations as in 5.1;
c) **BOUND S INF** all the values (s, v, t) from **VALUES** for which s = SMAX, where SMAX is a maximal value, sufficiently large, considered as infinite, for example, $10S_0$, where S_0 could be the value of the asset at time t = 0; the same recommendations as in 5.1;
d) **BOUND v ZERO** all the values (s, v, t) from **VALUES** for which v = 0;
e) **BOUND v INF** all the values (s, v, t) from **VALUES** for which v is VMAX, where VMAX is a maximal value, sufficiently large, considered as infinite, for example, $10v_0$, where v_0 could be the value of volatility at time t = 0;
f) **BOUND t at MATURITY** all the values (s, v, t) from **VALUES** for which t = T; the same recommendations as in 5.1;
g) **INT** all from **VALUES** the values inside the domain that are not on the boundary, meaning **VALUES** \ (**BOUND S ZERO** ∪ **BOUND S INF** ∪ **BOUND v ZERO** ∪ **BOUND v INF** ∪ **BOUND t MATURITY**).

5.3. Template for a Python application to solve PDE using PINN

The following program is a Python template demonstrating how a PINN can be implemented. The **TensorFlow** module was chosen because:

a) It allows the implementation of various NN architectures;
b) It allows the evaluation of partial derivatives for **û**. For example, the derivative of **û** concerning **t** can be obtained using

```
duhat_dt = tape.gradient(uhat, t),
```

(and analog, any other derivative) from all the numerical and symbolic information available.

Thus, a Python application for solving a PDE using a PINN would look something like this:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

# Defining the parametres
Strike = 50
# maturity 1 year
```

```
T = 1
# volatility
sigma = 0.2
# riskfree 5%/year
r = 0.05
# timestamp now
t0 = 0
# support now
S0 = 10
# support infinity
SMax = 10 * S0
delta_S = 10
# 1 week
delta_t = 1/52
# number of epochs
epochs = 10000

def payoff(s):
   return max (0, s - Strike)

# border west
S1 = []
s = S0
while s <= SMax:
   S1 = S1 + [ s ]
   s += delta_S
u1 = [ payoff(s) for s in S1 ]
t1 = [ T         for s in S1 ]
# border south
t2 = []
t = t0
while t < T:
   t2 = t2 + [ t ]
   t += delta_t
S2 = [ 0 for t in t2 ]
u2 = [ 0 for t in t2 ]
# sparse = border conditions
S_observed = np.array(S2 + S1)
t_observed = np.array(t2 + t1)
u_observed = np.array(u2 + u1)

# Defining the PINN neural network
class PINN(tf.keras.Model):
   def __init__(self):
       super(PINN, self).__init__()
       self.hidden1 = tf.keras.layers.Dense(20, activation = 'tanh')
       self.hidden2 = tf.keras.layers.Dense(20, activation = 'tanh')
       self.output_layer = tf.keras.layers.Dense(1)
   def call(self, inputs):
       x = self.hidden1(inputs)
       x = self.hidden2(x)
       return self.output_layer(x)

# The PINN loss function, if there are multiple dimensions,
# it is added between the parametre list between S and t
def PDE_Loss(model, S, t):
   with tf.GradientTape(persistent = True) as tape:
       # if there are more dimensions
       # it is added to the tape list
       # between S and t
       tape.watch(S)
       tape.watch(t)
       # calculate û
       uhat = model(tf.concat([S, t], axis=1))
       # û derivate in relation to t
       duhat_dt = tape.gradient(uhat, t)
       # û derivate in relation to S
       duhat_dS = tape.gradient(uhat, S)
       # second derivate of û in relation to S
       duhat_dS2 = tape.gradient(duhat_dS, S)
   # the PDE operator of the Black-Scholes equation for û
   PDE = duhat_dt + 0.5 * sigma ** 2 * S ** 2 * duhat_dS2 + r * S * duhat_dS - r * uhat
   # PDE Loss (the average of the sums of the squares is taken)
   loss = tf.reduce_mean(tf.square(PDE))
   return loss

# Training function
def train(model, epochs = 10000):
```

```
    optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001)
    for epoch in range(epochs):
        with tf.GradientTape() as tape:
            # Generating input points, 1000 values
            S = tf.convert_to_tensor(np.random.uniform(0, SMax, (1000, 1)), dtype = tf.float32)
            t = tf.convert_to_tensor(np.random.uniform(t0, T, (1000, 1)), dtype = tf.float32)
            # Calculating loss on the equation operator
            Loss_pde = PDE_Loss(model, S, t)
            # Calculating loss for the western boundary
            tensor_S_observed = tf.convert_to_tensor (S_observed[:, np.newaxis], dtype =
tf.float32)
            tensor_t_observed = tf.convert_to_tensor (t_observed[:, np.newaxis], dtype =
tf.float32)
            tensor_u_observed = tf.convert_to_tensor (u_observed[:, np.newaxis], dtype =
tf.float32)
            tensor_input = [tensor_S_observed, tensor_t_observed]
            uhat = model(tf.concat(tensor_input, axis = 1))
            Loss_data = tf.reduce_mean(tf.square(uhat - tensor_u_observed))
            total_loss = Loss_pde + Loss_data
        # Updating the model
        grads = tape.gradient(total_loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grads, model.trainable_variables))
        if epoch % (epochs // 10) == 0:
            print(f"Epoch: {epoch}, Loss: {total_loss.numpy()}")

# Initialisation and training of the model
model = PINN()
train(model, epochs)

# Graph of estimated option values
S_test = np.linspace(0, 100, 100)[:, np.newaxis]
t_test = np.ones_like(S_test) * T
u_estimated = model(tf.concat([S_test, t_test], axis = 1))
plt.plot(S_test, u_estimated,label = 'Estimated option value')
plt.plot(S_observed, u_observed, label = 'Observed data')
plt.title("European Call option value")
plt.xlabel("Asset price S")
plt.ylabel("Option value V")
plt.legend()
plt.grid()
plt.show()
```

Figure 1. Python code for exemplification of a PINN for European Call Options.

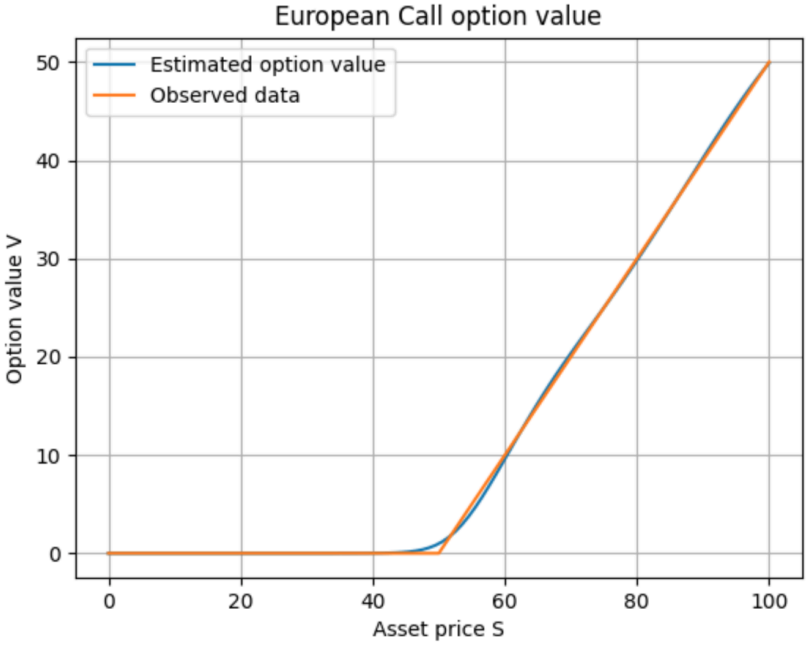

Figure 2. Output computed by code.

Some numeric results on Black-Scholes PDE with more observed and boundary data product some numerical results, and they are compared with the canonical Black-Scholes closed formulae. Results seem suitable as illustrated in Figures 3 and 4:

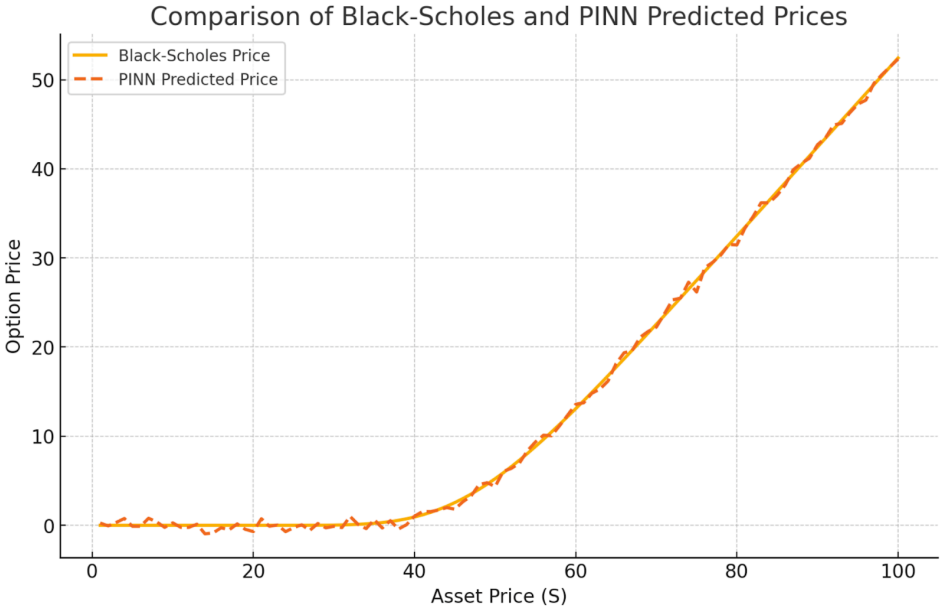

Figure 3. Comparison between canonical BS closed formulae and output of a PINN.

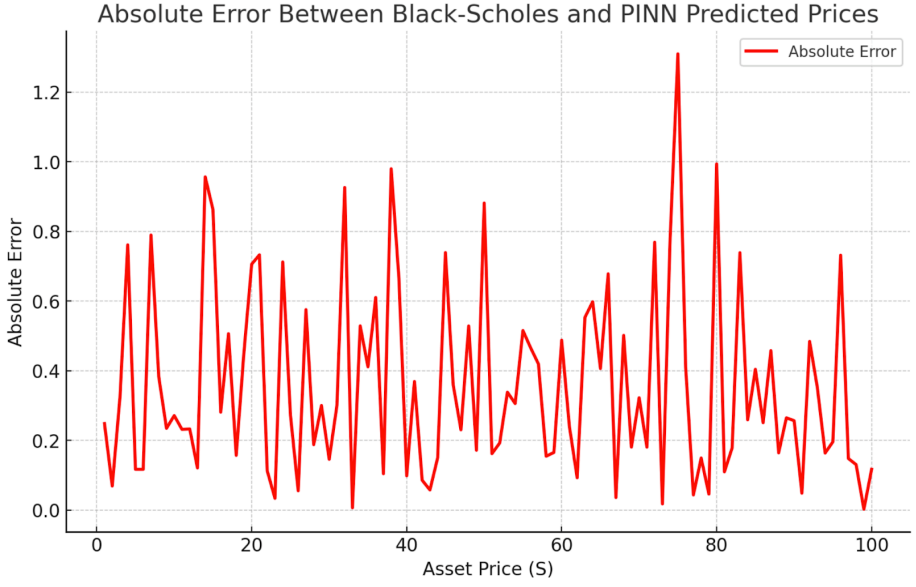

Figure 4. Absolute error between canonical BS closed formulae and output of a PINN.

6. A PINN System for Solving the Pricing Problem for a Aamily of European CALL/PUT Options with Various Strike Prices and Different Maturities Could be Structured as Follows

In practice, options based on an underlying asset constitute a class with various strikes. For example, for Apple stock (AAPL on NASDAQ), at one point (see Figure 5), strikes were identified ranging from 5 USD to 370 USD with a step of 2.5 USD. Therefore, we need to ensure that for the boundary t = T, the deltaS equals the step or a submultiple of it and that Smax exceeds $S_0 = S(t_0)$.

Usually, $S_{max} = 10S_0$ is sufficient to be considered as S = infinity. Since the training algorithms are similar, if we accept that the architecture is identical for all PINNs corresponding to different strikes, we could parallelize the training phase of the PINNs (Socaciu et al., 2009; Socaciu & Danubianu, 2009; Socaciu, 2011; Socaciu, Maxim, & Danubianu, 2011).

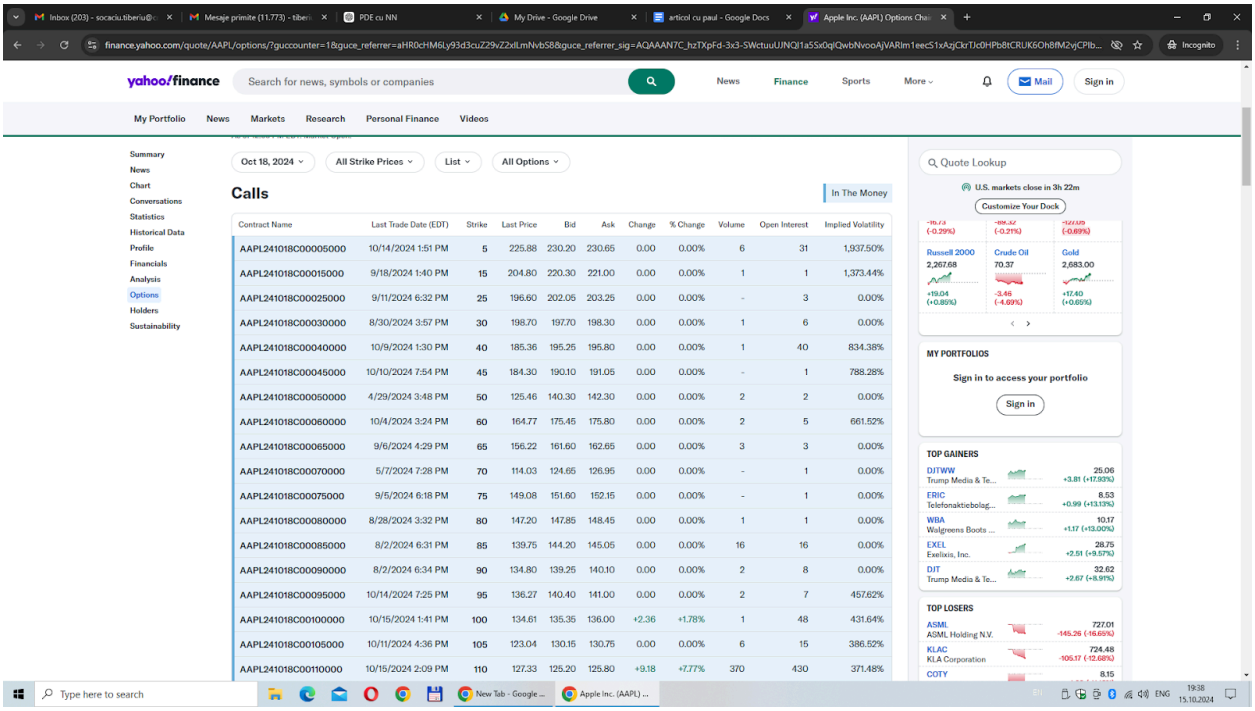

Figure 5. Call options based on the AAPL (Apple) stock listed on NASDAQ, with strike prices ranging from 5 USD to 370 USD in increments of 2.5 USD. Captured from Yahoo Finance on October 15, 2024.

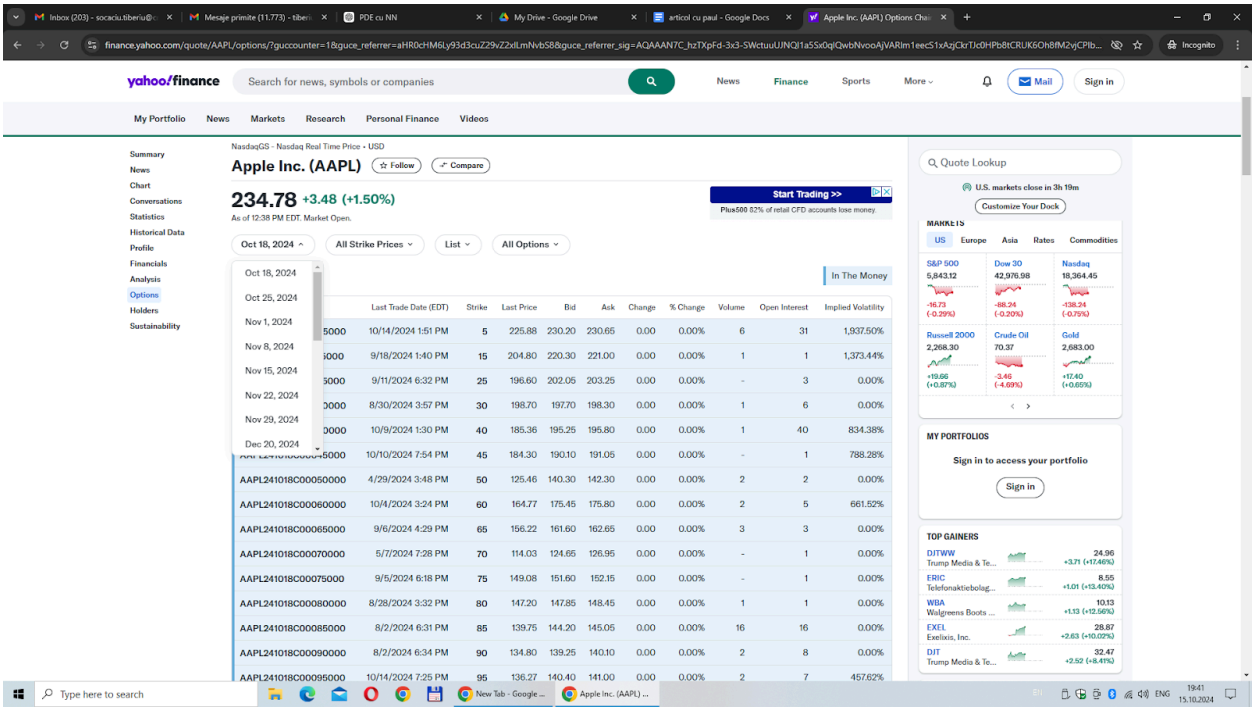

Figure 6. Call options based on the AAPL (Apple) stock listed on NASDAQ, with maturities ranging from October 18, 2024, to January 15, 2027, in weekly increments.
Captured from Yahoo Finance on October 15, 2024.

Also, options based on an underlying asset form a class with various maturities. For example, for Apple stock (AAPL on NASDAQ), at one point (see Figure 6), maturities were identified ranging from October 18, 2024, to January 15, 2027, with a weekly step. This means that we need to be careful with the time discretization and set deltaT equal to the time step or a finer division.

Thus, if we note with $A_{CALL}(S)$, respectively $A_{PUT}(S)$, the sets of all pairs (Strike, Maturity) for the CALL and PUT derivatives based on asset S, then we could approach the pricing resolution as follows:

a) in the minimalist system, exclusively for $A_{CALL}(S) \cup A_{PUT}(S)$;
b) in the intermediate system, for { (strike, maturity) | (strike, m) $\in A_{CALL}(S) \cup A_{PUT}(S)$ or (s, maturity $\in A_{CALL}(S) \cup A_{PUT}(S)$ };
c) in the maximal system for { (strike, maturity) | strike = min(S), max($10S_0$, max(S)), deltaS, maturity = min(maturiTy), max(maturity), deltaT }.

7. Conclusion

PINNs offer an innovative framework for solving PDEs that underpin financial models such as Black-Scholes and Heston. Such models are essential for pricing financial options and provide a better understanding of financial market behaviour. PINNs effectively combine the advantages of numerical methods with those of machine learning, offering a scalable, adaptable, and precise approach. They can directly integrate physical knowledge and boundary conditions, reducing the need for extensive discretisation and complex derivative calculations. PINNs are capable of solving complex PDEs associated with advanced stochastic models, such as the Heston model, which describes the random volatility of assets. This expands their applicability to exotic financial options and complex financial scenarios.

A unique feature of PINNs is, on one hand, the ability to train models for multiple financial options simultaneously (with different strike prices and maturities), significantly reducing the time required to analyse a family of options. On the other hand, PINNs can be adjusted using experimental or market data, which enhances the accuracy of the results and provides a closer connection to financial reality.

PINNs offer multiple directions for future development, with the potential to revolutionise the financial domain. Extending them to other classes of financial instruments, such as exotic options or complex derivatives, could ENABLE new perspectives for modeling emerging markets or incomplete data, characterised by high volatility. Meanwhile, optimising network architectures and integrating uncertainty into predictions could increase the accuracy and robustness of models, while their implementation on distributed infrastructures would facilitate the parallelisation of calculations for various financial scenarios (Socaciu et al., 2009; Socaciu & Danubianu, 2009; Socaciu, 2011; Socaciu, Maxim, & Danubianu, 2011). Additionally, integrating PINNs into trading platforms would enable real-time evaluations and better risk management. The incorporation of high-frequency data streams and the automation of network design through AutoML would make these technologies an indispensable tool for market analysis and forecasting. Furthermore, their use for financial compliance and scenario modeling would contribute to greater transparency and the robustness of regulatory systems. Comparing them with other advanced machine learning techniques will continue to highlight the advantages of PINNs, enhancing their applicability and versatility in the financial domain.

Acknowledgement

The paper is derived from the Ph.D. thesis by the principal investor and improved with additional works by coauthors as per the credit authorship contributor statement. No funding was received for this study.

Credit authorship contribution statement

1) Tiberiu Socaciu: conceptualization, investigation, methodology, writing original draft, writing review, writing editing, algorithms.
2) Paul Pascu: formal analysis, professional conceptualization, validation, software development, data processing and visualisation, proofreading, writing review, writing editing, fundraising.

References

American Mathematical Society. (2020). *Mathematics Subject Classification 2020*. Retrieved from https://mathscinet.ams.org/msc/msc2020.html

Bendsøe, M. P., & Sigmund, O. (2003). *Topology optimization: Theory, methods, and applications*. Springer.

Courant, R., & Hilbert, D. (2008). *Methods of mathematical physics: Volume 1*. John Wiley & Sons. (Original work published in 1953). ISBN: 978-0471504474.

Erleben, K., Sporring, J., Henriksen, K., & Dohlmann, H. (2005). *Physics-based animation*. Charles River Media. ISBN: 1-58450-380-7.

Ferziger, J. H., & Perić, M. (2002). *Computational methods for fluid dynamics* (3rd ed.). Springer. ISBN: 3-540-42074-6.

Gottlieb, D., & Orszag, S. A. (1977). *Numerical analysis of spectral methods: Theory and applications*. SIAM. https://doi.org/10.1137/1.9781611970425

Hagan, M. T., Demuth, H. B., Beale, M., & De Jesús, O. (2014). *Neural network design* (2nd ed.). ISBN: 978-0971732117.

Kloeden, P. E., & Platen, E. (1992). *Numerical solution of stochastic differential equations*. Springer. https://doi.org/10.1007/1978-3-662-12616-5

LeVeque, R. J. (2004). *Finite-volume method for hyperbolic problems*. Cambridge University Press. ISBN: 0-521-00924-3.

Morton, K. W., & Mayers, D. F. (2005). *Numerical methods for partial differential equations* (2nd ed.). Cambridge University Press. ISBN: 978-0521855560.

Pang, T. (2006). *An introduction to computational physics* (2nd ed.). Cambridge University Press. ISBN: 978-0-521-53276-1.

Patrut, B., & Socaciu, T. (2016). Black-Scholes formula: A Heston approach. *Broad Research in Accounting, Negotiation, and Distribution, 6*(1).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving PDEs. *Journal of Computational Physics, 378*, 686–707. https://doi.org/10.1016/j.jcp.2018.10.045

Richardson, L. F. (2010). *Weather prediction by numerical process* (2nd ed.). Cambridge University Press. (Original work published in 1929). ISBN: 9780511618291.

Socaciu, T. (2009). Obtaining Generalized Garman Equation. In D. Airinei, C. Pintilescu, L. Baciu, O. Lupu, A. Iacobuta, & M. Asandului (Eds.), *Globalization and Higher Education in Economics and Business Administration* (pp. 53–58). Editura Tehnopress. ISBN 9789737027122.

Socaciu, T., Parpucea, I., Pârv, B., & Pârv, M. (2009). Parallel algorithms for financial derivatives evaluation in generalized Heston model. *Scientific Studies and Research. Series Mathematics and Informatics, 19.*

Socaciu, T., & Danubianu, M. (2009). Two parallel solutions of Black-Scholes equation. *Journal of Applied Computer Science and Mathematics, 5*(2).

Socaciu, T. (2010). *Metode numerice de evaluare a opțiunilor financiare*. Editura InfoData. ISBN: 978-973-1803-55-5.

Socaciu, T. (2011). *Paralelizarea sistemelor de asistare a deciziei pentru ingineria financiară. Paralelizarea evaluării derivatelor financiare* [Doctoral thesis]. Universitatea Alexandru Ioan Cuza din Iași, Școala Doctorală de Economie și Administrarea Afacerilor.

Socaciu, T., Maxim, I., & Danubianu, M. (2011). Parallel algorithms for Black-Scholes like PDE with domain decomposition. In *AFASES 2011. The 13th International Conference of Scientific Papers*, *2.*

Socaciu, T. (2016). Financial derivatives (based on two supports) evaluation. *Broad Research in Accounting, Negotiation, and Distribution, 2*(2), 22–25.

Strauss, W. A. (2008). *Partial differential equations: An introduction*. Wiley. ISBN: 978-0470054567.

Troltzsch, F. (2010). *Optimal control of partial differential equations: Theory, methods and applications*. American Mathematical Society. ISBN: 9780821849040.

Zienkiewicz, O. C., & Cheung, Y. K. (1967). *The finite element method in structural and continuum mechanics: Numerical solution of problems in structural and continuum mechanics*. McGraw-Hill. ISBN: 749171660.