International Journal of Mathematics Trends and Technology
ISSN: 2231-5373/ https://doi.org/10.14445/22315373/IJMTT-V71I2P101
Volume 71 Issue 2, 1-6, February 2025

Original Article

Deep Learning for solving Fokker-Planck-Kolmogorov Equation Arising from Jump-Diffusion Model

Xingyu Zhang[1], Jianguo Tan[2]

[1]*Department of Mathematics, Tiangong University, Tianjin, PR China*.

[1]*Corresponding Author : 18822372286@163.com*

Received: 17 December 2024 Revised: 20 January 2025 Accepted: 04 February 2025 Published: 18 February 2025

Abstract *- The Fokker-Planck-Kolmogorov (FPK) equation generated by the jump-diffusion model is represented as a Partial Integro-Differential Equation (PIDE). Traditional numerical methods for solving PIDEs are often constrained by dimensionality and the complexity of mesh generation. To address these limitations, this paper proposes a novel deep-learning approach for solving PIDEs. Building upon the existing Physics-Informed Neural Network (PINN) framework for solving Partial Differential Equations (PDEs), we incorporate additional deep neural networks (DNNs) and construct a novel loss function to simultaneously optimize the integral terms and the solution for approximating the PIDE. The results demonstrate that our approach accurately captures the system's dynamic behaviour, highlighting its effectiveness and potential for solving complex PIDEs.*

Keywords *- Deep learning, Partial integro-differential equation, Jump-diffusion model, Fokker-Planck-Kolmogorov equation, Gaussian and Poisson white noises.*

1. Introduction

In scenarios where jumps occur, models driven solely by continuous noise are insufficient to accurately capture the dynamics. The jump-diffusion model has been proposed to address this limitation, extending the traditional framework. As a result, the FPK equation, used to describe the evolution of probability density functions, transitions from a PDE to a PIDE.

Traditionally, solving such PIDEs has relied on numerical methods, including finite difference methods (FDM) [6], the path integral (PI) method [11], and the Lyapunov function method [12]. However, these classical approaches often face challenges such as mesh generation, convergence issues, and high computational costs [2]. These limitations highlight the need for more efficient and robust solution methods to tackle PIDEs in complex systems. Deep learning methods have demonstrated significant potential for solving differential equations. Take the PINN algorithm [1] using deep learning to solve PDE as an example; in recent years, more and more people have applied the PINN algorithm in engineering, finance, medicine and other fields. So in this article, we extend the algorithm so that it can solve PIDEs.

In recent years, deep learning has emerged as a powerful tool for solving differential equations. For instance, the PINN algorithm has demonstrated remarkable success in solving PDEs and has found applications in diverse fields, such as engineering, finance, and healthcare. Building on this foundation, we extend the PINN framework to solve PIDEs.

A key advantage of deep learning over traditional numerical methods, such as FDM, lies in the differentiability of solutions fitted by neural networks. Leveraging this feature, we propose an approach that employs additional neural networks to handle the integral terms in PIDEs. Specifically, the neural network approximates the antiderivative of the integrand in the integral term, enabling efficient computation of the integral using Newton-Leibniz's formula.

This paper is organized as follows: Section 2 introduces the mathematical formulation of the PIDE. Section 3 describes our deep learning-based method for solving PIDEs. Section 4 evaluates the predictive accuracy and convergence performance of the proposed method using the Ornstein-Uhlenbeck (OU) process as a case study. Finally, Section 5 concludes the paper and outlines directions for future research.

2. The Basic Equations

A one-dimensional jump-diffusion stochastic differential equation (SDE) influenced by the combined effects of Gaussian and Poisson white noise is described as follows:

$$dX_t = f(X_t, t)dt + g(X_t, t)dW(t) + J(X_t)dP(t) \tag{2.1}$$

Here, $f(X_t, t)$ represents the drift coefficient, $g(X_t, t)$ denotes the diffusion coefficient, and $W(t)$ is an increment of a standard Brownian motion with intensity q. Moreover $P(t)$ represents the Poisson white noise with an arrival intensity λ and random variable $J(X_t)$ is an impulse function that introduces jumps, with its probability density function (PDF) given by $h(x)$.

Thus, the forward FPK equation corresponding to the SDE (2.1) is given as:

$$Lp(x,t) = L_1 p(x,t) + \lambda I(x,t) = \frac{\partial}{\partial t} p(x,t) \tag{2.2}$$

$$L_1 p(x,t) = -\frac{\partial(f(x)p(x,t))}{\partial x} + \frac{q}{2}\frac{\partial^2\left(g^2(x),p(x,t)\right)}{\partial x^2} - \lambda p(x,t) \tag{2.3}$$

$$I(x,t) = \int_{-\infty}^{\infty} h(x-y)p(x)dy \tag{2.4}$$

Here, $p(x)$ denotes the PDF of the SDE (2.1). The initial, boundary, and normalization conditions are specified as follows:

$$p(x,0) = p_0(x) \tag{2.5}$$

$$p(\infty, t) = p(-\infty, t) = 0 \tag{2.6}$$

$$\int_{-\infty}^{\infty} p(x, \hat{t})dx = 1\,, \forall \hat{t} \in [0, T] \tag{2.7}$$

3. Methodology

In this section, we will describe the numerical scheme for solving the PIDE associated with the FPK equation.

3.1. A brief of using Deep Learning to Solve Differential Equations

We assume that the predicted solution is $p_\theta(x,t)$, which represents the output of the neural network, expressed as:

$$p_\theta(x,t) = NN_1(x,t;\theta) \tag{3.1}$$

Here, the parameter vector θ in the neural network represents the weights and biases.

Infinite boundaries are impractical for computational purposes. Generally, it is sufficient to truncate x to the range $[xmax_{min}$. Based on the differential equation (2.2) and the necessary conditions (2.5)-(2.6), we can construct the loss function as follows:

$$Loss_1(\theta) = \sum_{i=1}^{3} a_i \cdot E_i \tag{3.2}$$

$$E_1 = \frac{1}{N_D} \cdot \sum_{i=1}^{N_D} [L_1(p_\theta(x_i, t_i)) - \lambda I(x_i, t_i)]^2 \tag{3.3}$$

$$E_2 = \frac{1}{N_B} \cdot \sum_{i=1}^{N_B} [p_\theta(x_i, t_i)]^2 \tag{3.4}$$

$$E_3 = \frac{1}{N_i} \cdot \sum_{i=1}^{N_i} [p_\theta(x_i, 0) - p_0(x_i)]^2 \tag{3.5}$$

Here, $a_i, i = 1,2,3$ represents the weight parameter, which is adjusted according to an empirical method $\{(x_i, t_i)\} \subset [xmax_{min}$. In this article, we typically set all weight parameters to 1.

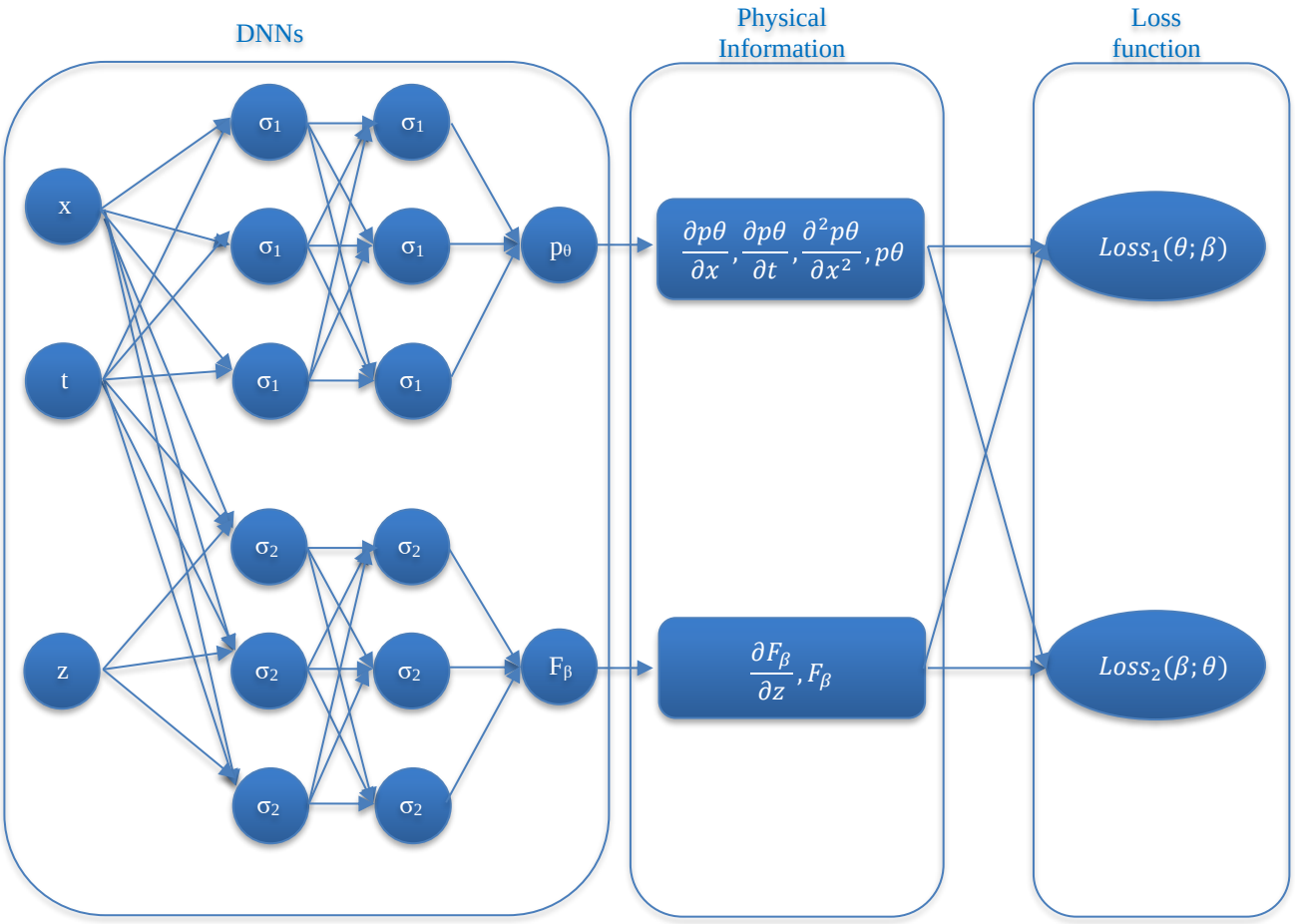

Fig. 1 Schematic of two DNNs for solving PIDEs

3.2. Existing Problems and Solutions

The main challenge lies in the integral term in equation (3.3), which is not straightforward to handle. The most classical approach to addressing integral terms is the Gauss-Legendre (GL) quadrature. However, this method is memory-intensive and significantly increases computational time. We propose using an additional deep neural network (DNN) to overcome these difficulties to compute the integral terms. In general, the cutoff value of the integrated variable$y$ is set to be the same as that of the variable x [2]. Consequently, the integral function $I(x,t)$ can be reformulated as follows:

$$p(x,t) = 0, x \in [-\infty, xmax_{min} \tag{3.6}$$

$$I(x,t) = \int_{-\infty}^{\infty} h(x-y)p(x,t)dy = \int_{x_{min}}^{x_{max}\int \cdot} h(x-y)p(x,t)dy \tag{3.7}$$

It is now evident that the integrand $h(x-y)p(x,t)$ is continuous. If we assume that $F(x,y,t)$ is the antiderivative of the integrand, the integral can be directly evaluated using the Newton-Leibniz formula. Based on this idea, we use the output of the DNN as $F_\beta(x,y,t)$ denoted as $F_\beta(x,y,t) = NN_2(x,y,t;\beta)$. In other words, we only need to ensure that the derivative of $F_\beta(x,y,t)$ with respect to y equals the integrand $h(x-y)p(x,t)$. Accordingly, we can construct additional residuals as follows:

$$Loss_2(\beta) = Loss_2(\beta;\theta) = \frac{1}{N_f}\sum_{i-1}^{N_f}[\frac{\partial}{\partial y}F_\beta(x_i^f, y_i^f, t_i^f) - h(x_i^f, y_i^f)p_\theta(x_i^f, t_i^f)]^2 \tag{3.8}$$

Meanwhile, Equation (3.3) can be rewritten as follows:

$$E_1(\theta) = E_1(\theta;\beta) = \frac{1}{N_D} \cdot \sum_{y_{min}}^{y_{max}]^2 \, i=1}^{N_D} [L_1(p_\theta(x_i,t_i)) - \lambda F_\beta(x_i,y,t_i)| \qquad (3.9)$$

Here $N$is batch size and$\{x_i^f, y_i^f, t_i^f\} \subset (xmaxminmax_{min}$. It should be noted that$Loss_1(\theta)$ and $Loss_2(\beta)$ are minimized separately during each iterative optimization. In other words, when optimizing θ or β, the other set of optimization parameters is treated as constant. Furthermore, it is not necessary for $F_\beta(x_i,y,t_i)|_{y_{min}}^{y_{max}}$to precisely approximate$I(x_i,t_i)$ at every step. This is because, during the initial stages of training, the discrepancy between the predicted solution $p_\theta(x,t)$ and the exact solution $p(x,t)$ is significant, making such accuracy less critical at the outset.

Figure 1 illustrates the main structural form of solving a PIDE using two DNNs. The pseudo-code in Table 1 illustrates the process of our method for computing the numerical solution of the PIDE, with more details already described in the preceding text.

Table. 1 The pseudo-code of our deep learning method for solving PIDE.

Algorithm:	Deep Learning for Solving PIDEs.
Input:	The training set and the hyperparameters of the DNNs, the parameters of the question.
Output:	The predicted solutions.
1:	Initialize the weights and biases of the$NN_1(x,t;\theta)$ and $NN_2(x,y,t;\beta)$.
2:	**Repeat for epochs times**
3:	Construct neural network and get the output $p_\theta(x,t)$ and $F_\beta(x,y,t)$.
4:	The predicted solution$p_\theta(x,t)$ is combined with the initial and boundary conditions,as well as the PIDE, to construct the loss function$Loss_1(\theta;\beta)$. Similarly, the $F_\beta(x,y,t)$ is used to approximate the integrand by constructing the loss function$Loss_2(\beta;\theta)$.
5:	Update the parameters $\theta$and $\beta$by minimizing the loss function$Loss_1(\theta;\beta)$ and $Loss_2(\beta;\theta)$, respectively.

4. Examples

The primary objective of this section is to evaluate the effectiveness of our algorithm. To achieve this, the solution obtained using the Monte Carlo method will be treated as the standard solution for comparison.

Consider a nonlinear function in Equation (2.3):

$$f(x) = ax + bx^3, g(x) = 1 \qquad (4.1)$$

Combining the above formula with the SDE (2.1) describes the OU process, where $x$is defined within the range $[xmax_{min}$. We assume that the random variable $J(X_t)$ in Equation (2.3) follows a normal distribution $N(\mu_R, \sigma_R^2)$. Consequently, the PIDE associated with the Fokker-Planck (FP) equation is described as follows:

$$-\ (a + 3bx^2 + \lambda)\ p(x,t) - (ax + bx^3)\frac{\partial}{\partial x}p(x,t) + \frac{q}{2}\frac{\partial^2}{\partial x^2}p(x,t) + \frac{\lambda}{\sqrt{2\pi\sigma_R^2}} \times \int_{-\infty}^{\infty} exp(-\frac{(x-y-\mu_R^2)}{2\sigma_R^2})\, p(x,t)dx = \frac{\partial}{\partial t}p(x,t) \qquad (4.2)$$

The initial condition is defined as follows:

$$p_0(x) = \frac{1}{\sqrt{2\pi}} exp(-\frac{x^2}{2}) \qquad (4.3)$$

For the PIDE (4.2), the Monte Carlo results are obtained by fitting 500,000 paths. In our deep learning algorithm, the learning rate is reduced to 75% every 5000 iterations. The initial learning rate is set to 0.005, and the training process is conducted over 30,000 epochs using the Adam optimizer. During training, both $NN_1(x,t;\theta)$ and $NN_2(x,y,t;\beta)$ consist of 8 hidden layers, each with 40 neurons, and the activation function is set to Sigmoid. Unless otherwise specified, the parameters are defined as follows:

$$a = 0.5, b = -0.5, \lambda = 0.5, q = 0.25, \mu_R = 0., \sigma_R = 0.1, xmax_{min} \qquad (4.4)$$

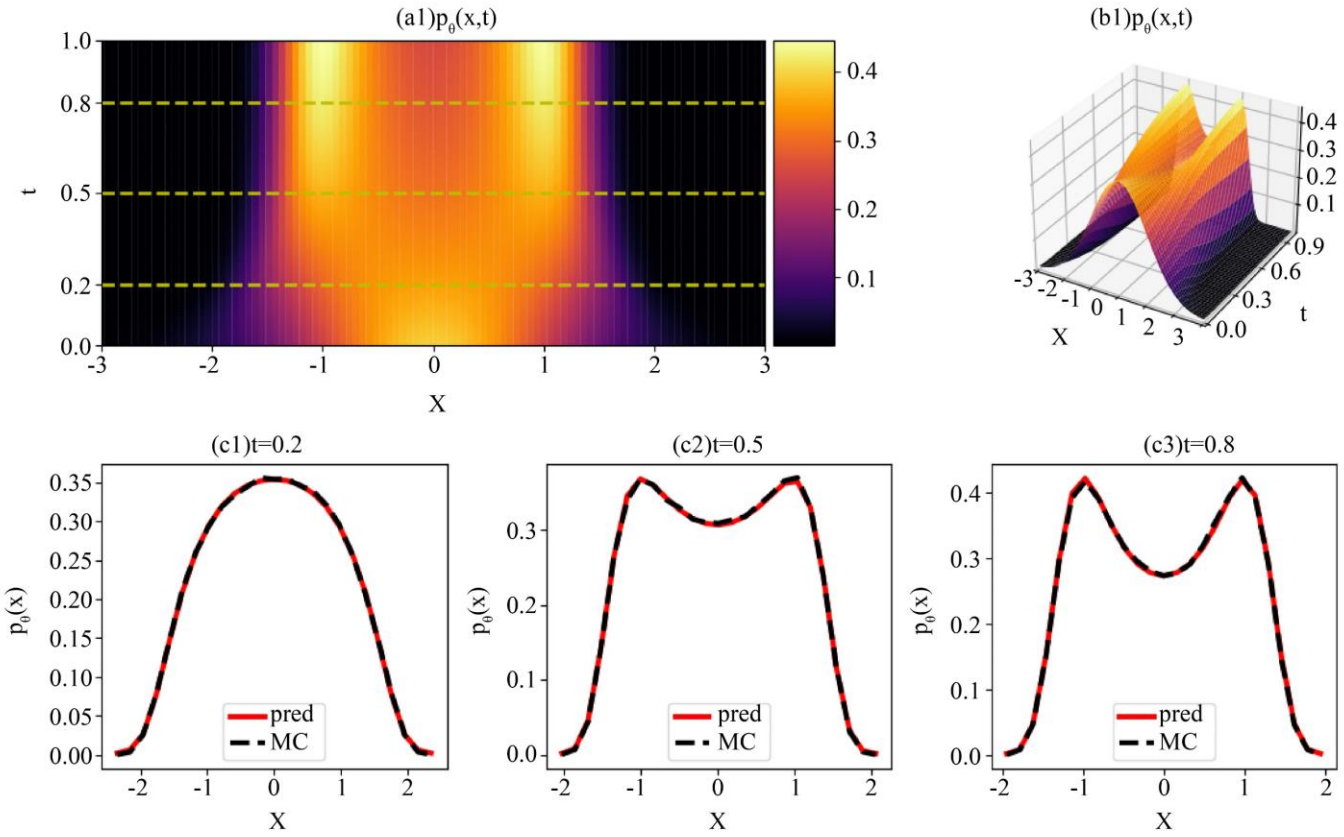

Fig. 2 The predictive solutions $p_\theta(x,t)$ of the PIDE (4.2) of the PIDE (4.2) were obtained using our deep learning method, which employs a feedforward neural network (FNN) to compute the integral term. (a1)The density plot is generated over the truncated space-time range. (b1)The three-dimensional pattern is visualized within the corresponding truncated space-time range. (c1)-(c3)These figures compare $p_\theta(x,t)$ with the results obtained by the Monte Carlo method. Specifically, (c1)-(c3) correspond to three temporal snapshots from (a1), at$t = 0.2, 0.5, 0.8$, respectively. And the solid red line represents the prediction solution obtained by our deep learning method, and the dashed black line represents the comparison solution obtained by Monte Carlo method.

Moreover, N_B, N_i, $N_D$and N_f in formulas (3.4), (3.5), (3.8) and (3.9) are set to300, 300, 6000 and 5000, respectively.

The Figure 2 illustrates the predicted solutions $p_\theta(x,t)$ of the PIDE (4.2) obtained by our deep learning method, including the density plot, the 3D plot and the the comparison with the MC reference solutions at three temporal snapshots in (a1):$t =$ 0.2,0.5,0.8, respectively. The results show that the solution of our method can capture the solution of PIDE (4.2) .

5. Conclusion

In this paper, we propose a novel deep learning method that leverages DNNs to compute the integral terms in equations and provide numerical solutions to the PIDE. Furthermore, we validate the effectiveness of this method by applying it to solve the FP equation for the OU process. The results demonstrate that the proposed method successfully captures the solution of the PIDE. However, our current work has limitations, and several challenges remain unaddressed, such as handling cases where the integrand function is discontinuous. In future work, we aim to extend our approach to tackle such models and explore additional applications.

Funding Statement

The research was conducted without any external financial support.

Acknowledgments

All authors contributed to the study conception and design.
Xingyu Zhang: Conceptualization, Methodology, Software, Data curation and Writing- Original draft preparation.
Jianguo Tan: Supervision, Writing- Reviewing and Editing.

References

[1] M. Raissi, P. Perdikaris, and G.E. Karniadakis, "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations," *Journal of Computational Physics*, vol. 378, pp. 686-707, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[2] Wantao Jia et al., "Deep Neural Network Method to Predict the Dynamical System Response under Random Excitation of Combined Gaussian and Poisson White Noises," *Chaos, Solitons & Fractals*, vol. 185, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[3] Sifan Wang, and Paris Perdikaris, "Deep Learning of Free Boundary and Stefan Problems," *Journal of Computational Physics*, vol. 428, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[4] Xu Sun, Jinqiao Duan, and Xiaofan Li, "Stochastic Modeling of Nonlinear Oscillators under Combined Gaussian and Poisson White Noise: A Viewpoint Based on the Energy Conservation Law," *Nonlinear Dynamics*, vol. 84, pp. 1311-1325, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[5] Weiyan Liu, Weiqiu Zhu, and Wantao Jia, "Stochastic Stability of Quasi-Integrable and Non-Resonant Hamiltonian Systems Under Parametric Excitations of Combined Gaussian and Poisson White Noises," *International Journal of Non-Linear Mechanics*, vol. 58, pp. 191-198, 2014. [CrossRef] [Google Scholar] [Publisher Link]

[6] Rama Cont, and Ekaterina 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. [CrossRef] [Google Scholar] [Publisher Link]

[7] S. Chakraverty, and Susmita Mall, "Single Layer Chebyshev Neural Network Model with Regression-Based Weights for Solving Nonlinear Ordinary Differential Equations," *Evolutionary Intelligence*, vol. 13, no. 4, pp. 687-694, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[8] I.E. Lagaris, A. Likas, and D.I. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE Transactions on Neural Networks*, vol. 9, pp. 987-1000, 1998. [CrossRef] [Google Scholar] [Publisher Link]

[9] Ling Guo et al., "Monte Carlo fPINNs: Deep Learning Method for Forward and Inverse Problems Involving High Dimensional Fractional Partial Differential Equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 400, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[10] Leif Andersen, and Jesper Andreasen, "Jump-Diffusion Processes: Volatility Smile Fitting and Numerical Methods for Option Pricing," *Review of Derivatives Research*, vol. 4, pp. 231-262, 2000. [CrossRef] [Google Scholar] [Publisher Link]

[11] A. Pirrotta, and R. Santoro, "Probabilistic Response of Nonlinear Systems under Combined Normal and Poisson White Noise via Path Integral Method," *Probabilistic Engineering Mechanics*, vol. 26, no. 1, pp. 26-32, 2011. [CrossRef] [Google Scholar] [Publisher Link]

[12] Weiyan Liu, and Weiqiu Zhu, "Lyapunov Function Method for Analyzing Stability of Quasi-Hamiltonian Systems Under Combined Gaussian and Poisson White Noise Excitations," *Nonlinear Dynamics*, vol. 81, pp. 1879-1893, 2015. [CrossRef] [Google Scholar] [Publisher Link]