Adaptive Numerical Solution Algorithm for High-Dimensional SPDEs Based on the Feynman-Kac Formula and Neural Networks

Zifan Liu *

School of Mathematics, Jilin University, Changchun, China, 130012

* Corresponding Author Email: liuzf1022@mails.jlu.edu.cn

Abstract. The Feynman-Kac formula establishes a connection between stochastic processes and partial differential equations (PDEs), providing a novel approach for the numerical solution of PDEs by expressing the solution of a PDE as the expected value of a random variable. Moreover, this formula can be used to solve for the expected solution of stochastic partial differential equations (SPDEs). However, in practical applications, the formula is often constrained by the range of values that the random variable can take. When the values of the random variable are too large or too small, the numerical stability of the formula is reduced, and it may even become unusable. To address this issue, this paper proposes an adaptive algorithm that dynamically adjusts relevant parameters, enabling the formula to be applied to a broader range of situations. Furthermore, this adaptive algorithm is applied to high-dimensional stochastic partial differential equations, and a neural network algorithm is used to fit the expected solution of the SPDE. Experimental results show that, compared to traditional polynomial regression methods, this approach demonstrates higher precision and stability in high-dimensional problems. The adaptive algorithm proposed in this paper provides a novel approach for solving high-dimensional stochastic partial differential equations, with significant theoretical and practical value.

Keywords: Feynman-Kac Formula, Stochastic Partial Differential Equations, Adaptive Algorithm, Neural Networks.

1. Introduction

The Feynman-Kac formula is a well-known tool for providing a stochastic representation of the pointwise solution to many partial differential equations (such as diffusion equations or transport equations) [1-2]. For instance, consider solving a Dirichlet boundary value problem defined in a domain $D \subset R^n$, Here, it is assumed that the boundary is sufficiently smooth and satisfies the following conditions:

$$-\Delta u = f \tag{1}$$

The boundary condition is:

$$u = g, x \in \partial D \tag{2}$$

Then, for any solution, it can be expressed as [3]:

$$u(x) = \mathrm{E}_x\left[g\left(X_{\tau_D}\right) + \int_0^{\tau_D} f\left(X_s\right) ds \right] \tag{3}$$

Where $\left(X_t\right)_{t\geq 0}$ is the solution to the stochastic differential equation related to the operator Δ, and τ_D is the exit time of the process from the physical region D of the problem. Then the Monte Carlo method [4-5] is applied, simulating enough paths using a computer, and compute the corresponding values $g\left(X_{\tau_D}\right) + \int_0^{\tau_D} f\left(X_s\right) ds$ for each path. Finally, calculate the average value.

In simulating each path, to implement this formula on a computer, the paths of the random process $\left(X_t\right)_{t\geq 0}$ should be discretized. Starting from the position which is sought, the position of the next

point will be determined by using the position of the previous point and the stochastic differential equation corresponding to the partial differential equation, continuing until reaching the boundary, i.e., until the distance to the boundary is sufficiently small. Along each path segment, the integral $\int_{\tau_{k-1}}^{\tau_k} f(X_s)ds$ will be computed by using the right rectangle rule (where the start and end times τ_{k-1}, τ_k correspond to this small path segment), and finally, sum the results to obtain the approximate value $\int_0^{\tau_D} f(X_s)ds \,.\, g(X_{\tau_D})$ is the function value of g at the final point. Based on the above algorithm, the expected solution of a stochastic partial differential equation (SPDE) could be obtained. Consider the following equation as an example:

$$-\nabla\cdot(k(X,\omega)\cdot\nabla u(X,\omega)) = f(X,\omega), X \in D \tag{4}$$

The physical domain $D \subset R^n$ has the boundary condition:

$$u = g, X \in \partial D \tag{5}$$

ω defined on a complete probability space (Ω, F, P), where Ω is the sample space, F is the σ-algebra, and P is the probability measure; the diffusion coefficient $k(X,\omega)$ and the source term $f(X,\omega)$ are given random functions, so the solution $u(X,\omega)$ is also a random function. In general, it is quite difficult to directly obtain the solution $u(X,\omega)$, so its expected solution can be solved. Using the Monte Carlo method, random variables are sampled according to their distribution. In this case, the stochastic partial differential equation is converted into a deterministic partial differential equation, which can be solved using methods (1.1), (1.2) for partial differential equations. After taking enough samples of the random variable, compute the average of the obtained results.

For this problem (1.4), the stochastic process evolution formula $(X_t)_{t\geq 0}$ is:

$$dX_t = \nabla k(X,\omega)\cdot dt + \sqrt{2k(X,\omega)}dW_t \tag{6}$$

The discretized iterative formula is:

$$X_{t+1} = X_t + \nabla k(X,\omega)\cdot\Delta t + \sqrt{2k(X,\omega)\Delta t}\cdot N \tag{7}$$

Let Δt be the time step, and N be an n×1 column vector, where each component is a random variable that follows a standard normal distribution, and they are independent of each other. In this case, the formula contains random variables. When the values of the random variables are too small, the spatial step size $\nabla k(X,\omega)\cdot\Delta t + \sqrt{2k(X,\omega)\Delta t}\cdot N$ also becomes too small, which results in a significant increase in computational cost. On the other hand, when the values of the random variables are too large, the spatial step size $\nabla k(X,\omega)\cdot\Delta t + \sqrt{2k(X,\omega)\Delta t}\cdot N$ also becomes too large, leading to poor path discretization and making the formula unsuitable. To avoid such situations, this paper proposes an adaptive algorithm [6-7], which improves the above process and greatly extends the applicability of the formula [8-9].

The specific principle of this algorithm is to determine the time step size Δt for each simulation based on the values of the random variables. This ensures that the step size is appropriately chosen, balancing both accuracy and computational efficiency. Based on the simulation results presented below, this adaptive algorithm effectively meets the above requirements. Furthermore, this paper combines the adaptive algorithm with a neural network model to address high-dimensional SPDEs problems [10].

2. Adaptive Step Size Algorithm and Simulation Analysis

In the execution of the algorithm, the size of the random variable has a significant impact on the path discretization results. If the value of the random variable is too large, it will lead to a large step size, which significantly increases the numerical error. Conversely, if the value of the random variable is too small, it will substantially increase the time required for the algorithm, resulting in a significant increase in computational cost. In both cases, the numerical stability of the original algorithm will be affected, and it may even become unusable.

To effectively address this issue, this paper proposes an adaptive step size method. The core idea of this method is to dynamically adjust the step size after each random variable is sampled, ensuring that the space step size chosen for each simulated path is appropriate, thereby improving both the accuracy and computational efficiency of the algorithm. Specifically, the step size selection depends on the value of the random variable, with the goal of optimizing computational time while ensuring accuracy. The specific selection strategy for the step size is as follows:

$$\Delta t \cdot \left(\iint_D \frac{\partial k}{\partial x}(X,\omega)dxdy + \iint_D \frac{\partial k}{\partial y}(X,\omega)dxdy \right) = c \tag{8}$$

Where c is a given constant. The choice of c can control the speed and accuracy of the algorithm. Below is an example of a numerical experiment to solve the partial differential equation:

$$-\nabla\left(k(A,x,y)\cdot\nabla u(A,x,y)\right) = f(A,x,y),(x,y)\in D \tag{9}$$

$$u(A,x,y) = g(A,x,y),(x,y)\in \partial D \tag{10}$$

Where:

$$k(A,x,y) = A\cdot(x+y) \tag{11}$$

$$g(A,x,y) = A\cdot e^{xy} \tag{12}$$

$$f(A,x,y) = -A^2\cdot(x+y)\cdot\left(x^2+y^2+1\right)\cdot e^{xy} \tag{13}$$

$$D = [0,1]\times[0,1] \tag{14}$$

The exact solution of the above equation is:

$$u(A,x,y) = A\cdot e^{xy} \tag{15}$$

To verify the effectiveness of the improved algorithm, compare the original algorithm with the improved one and validate the results using the exact solution. To demonstrate the randomness of the random variables, firstly, fix the values of random variable A as 100, 1, and 0.01. Table 1 presents the numerical results of the original algorithm at the test point (0.5,0.5) (0.5, 0.5) (0.5,0.5), while Table 2 shows the numerical results of the improved algorithm. It can be observed that the improved algorithm effectively addresses the issues caused by the range of values of the random variable. Specifically, when the random variable takes larger values, the improved algorithm maintains higher accuracy; when the random variable takes smaller values, the algorithm significantly reduces the implementation time, thus effectively decreasing the computational load.

Next, simulate 5000 instances of the random variable, each time simulating 5000 paths, and compare the numerical results of both algorithms for calculating the expected solution (using the same test points). As shown in Table 3, the results indicate that the improved adaptive algorithm outperforms the original algorithm in both accuracy and efficiency.

From these experiments, it can conclude that the adaptive algorithm not only resolves the issues in the original model but also demonstrates good stability and efficiency in high-dimensional problems and when random variables exhibit large fluctuations. The improved algorithm, when

handling large-scale computations, can significantly reduce unnecessary computational burdens while maintaining high accuracy, providing a new approach for solving high-dimensional stochastic partial differential equations.

Table 1. Numerical analysis results before algorithm improvement

Comparison	Numerical solution	Real solution	Time (s)
A=100	-68	128.40	0.026
A=1	1.2802	1.2840	0.5
A=0.01	0.001281	0.001284	379

Table 2. Numerical analysis results after algorithm improvement

Comparison	Numerical solution	Real solution	Time (s)
A=100	128.22	128.40	0.4
A=1	1.2842	1.2840	0.4
A=0.01	0.001286	0.001284	0.4

Table 3. Comparison of time complexity between two methods

Comparison	Numerical solution	Real solution	Time (s)
Original algorithm	63.5	64.201	863
Improvement algorithm	64.307	64.201	200

In summary, the adaptive algorithm proposed in this paper effectively overcomes the limitations of the original algorithm. It demonstrates higher accuracy and greater stability when handling high-dimensional stochastic problems, offering strong theoretical significance and practical application value. Experimental results thoroughly validate the advantages of this algorithm, particularly when dealing with random variables of varying ranges. The algorithm can flexibly adjust its computational strategy, providing a more efficient solution.

3. Numerical Solution of SPDEs Based on Neural Networks and Adaptive Step Size

Based on algorithmic improvements, this paper further explores the solution of high-dimensional stochastic partial differential equations (SPDEs). Following the approach outlined earlier in this paper, the solution process involves obtaining numerical results at selected points within the domain and fitting these results to derive an approximate solution. However, traditional polynomial regression methods become ineffective in high-dimensional cases due to overfitting, the rapid increase in the number of basic functions, and their limited ability to handle complex nonlinear relationships.

To address these challenges, this paper integrates adaptive algorithms with neural networks. Neural networks, inspired by biological neural systems, consist of multiple layers, including an input layer, hidden layers, and an output layer, with connections weighted between neurons. During training, the network adjusts its weights and biases based on large datasets and optimizes the loss function through backpropagation, improving predictive accuracy. The dataset is divided into a training set for model learning and a test set for evaluating generalization performance.

To demonstrate the feasibility of this method, present the following numerical experiment:

$$-\nabla\left(k\left(X,\omega\right)\cdot\nabla u\left(X,\omega\right)\right)=f\left(X,\omega\right),X\in D \tag{16}$$

$$u\left(X,\omega\right)=g\left(X,\omega\right),X\in\partial D \tag{17}$$

Where:

$$g\left(X,\omega\right)=e^{\sum_{i=1}^{7}A_{i}x_{i}} \tag{18}$$

$$k(X,\omega)=B\cdot\left(\sum_{i=1}^{7}x_i^2\right)+C \tag{19}$$

$$\begin{aligned}
&f(X,\omega)=e^{\sum_{i=1}^{7}A_ix_i}\left[2B\left(\sum_{i=1}^{7}A_ix_i\right)+\left(B\cdot\left(\sum_{i=1}^{7}x_i^2\right)+C\right)\cdot\left(\sum_{i=1}^{7}A_i^2\right)\right]\\
&D=[0,1]^7\\
&A_i\sim U(0,1),i=1,2,\ldots 7\\
&B\sim U(0,100)\\
&C\sim Exp(1)
\end{aligned} \tag{20}$$

The exact solution to this equation is:

$$u(X,\omega)=e^{\sum_{i=1}^{7}A_ix_i} \tag{21}$$

Thus, the expected solution of this equation is:

$$E(u)=\sum_{i=1}^{7}\frac{e^{x_i}-1}{x_i} \tag{22}$$

Next, use the improved algorithm to solve the high-dimensional stochastic partial differential equation (SPDE) problem. First, 100,000 points are randomly selected in the physical region D, and the adaptive algorithm is applied at each point to obtain the approximate expected solution. Then, polynomial regression and neural network methods are used to obtain the regression function or model. To evaluate the performance of these two methods, another 100,000 points are randomly selected, and the real solution at these points is computed. The numerical solutions obtained from the previously fitted regression function or trained model are then compared, and the average relative error is calculated.

The experimental results show that the running time for the neural network method is about 9 seconds, and the average relative error is 1.36%.

In the comparison method—polynomial regression, the running time is about 10 seconds, and the average relative error is 29.63%.

From the experimental results, it is evident that the neural network significantly outperforms polynomial regression in high-dimensional problems. The neural network provides relatively accurate solutions in a shorter time with a small error of only 1.36%. On the other hand, despite having a similar runtime, polynomial regression faces difficulties in fitting high-dimensional problems, resulting in a larger error of 29.63%. This indicates that as the problem's dimensionality increases, traditional polynomial regression struggles to effectively fit the data, while the neural network method can handle high-dimensional complex problems more effectively. This experimental result fully demonstrates the advantages of neural networks in solving high-dimensional stochastic partial differential equations (SPDEs), providing a more precise and efficient solution for high-dimensional problems.

To comprehensively evaluate the overall performance of the proposed method, this paper also conducts sensitivity and convergence analyses of the parameters. Specifically, focus on analyzing the impact of three key parameters in the neural network model: iteration count, learning rate, and the number of samples, and how they affect the algorithm's error.

In this experiment, the model equation used is as follows:

$$-\nabla(k(X,\omega)\cdot\nabla u(X,\omega))=f(X,\omega),X\in D \tag{23}$$

$$u(X,\omega)=g(X,\omega),X\in\partial D \tag{24}$$

Where:

$$g(X,\omega)=e^{\sum_{i=1}^{7}A_i x_i} \tag{25}$$

$$k(X,\omega)=B\cdot\left(\sum_{i=1}^{7}x_i^2\right)+C \tag{26}$$

$$\begin{aligned} &f(X,\omega)=e^{\sum_{i=1}^{7}A_i x_i}\left[2B\left(\sum_{i=1}^{7}A_i x_i\right)+\left(B\cdot\left(\sum_{i=1}^{7}x_i^2\right)+C\right)\cdot\left(\sum_{i=1}^{7}A_i^2\right)\right]\\ &D=[0,1]^7\\ &A_i\sim U(0,1),i=1,2,\ldots 7\\ &B\sim U(0,100)\\ &C\sim Exp(1) \end{aligned} \tag{27}$$

The exact solution to this equation is:

$$u(X,\omega)=e^{\sum_{i=1}^{7}A_i x_i} \tag{28}$$

Thus, the expected solution of this equation is:

$$E(u)=\sum_{i=1}^{7}\frac{e^{x_i}-1}{x_i} \tag{29}$$

First, based on the previously described validation method, observed the trend of the average relative error as the number of iterations changes, and the results are shown in Figure 1. The horizontal axis in the figure represents the number of iterations, while the vertical axis represents the average relative error. It can be observed that in the early iterations, the error decreases rapidly, indicating fast convergence of the algorithm. However, when the number of iterations exceeds 10, the error stabilizes with no significant changes. This suggests that the neural network quickly learns data patterns in the initial training phase, but further training yields diminishing improvements. To balance computational efficiency and prediction accuracy, it is crucial to select an appropriate number of iterations—too few may lead to underfitting, while too many increase computational cost.

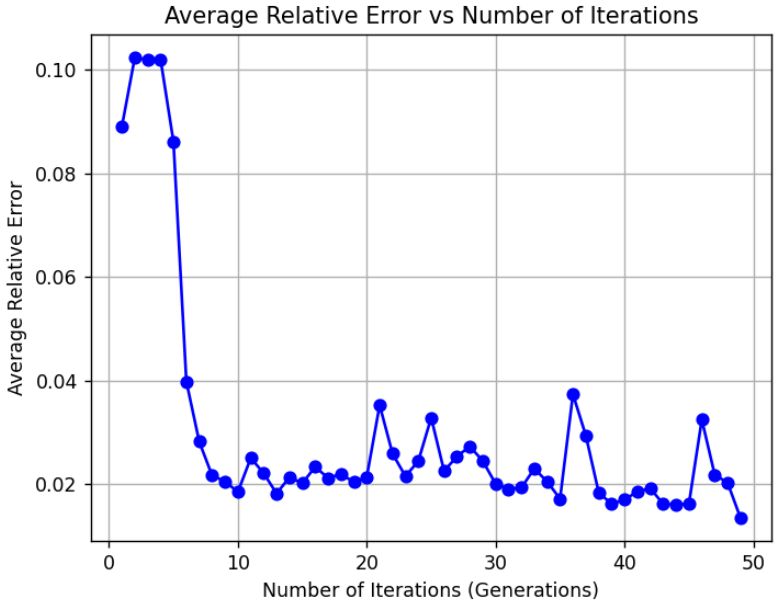

Figure 1. Mean Square Error of the Equation's Numerical Solution as the Number of Iterations Changes

Next, the variation of the average relative error with respect to the learning rate is analyzed, as shown in Figure 2. Figure 2 illustrates the trend of average relative error with respect to the learning rate. The results indicate that the optimal learning rate for this model is approximately 0.007.

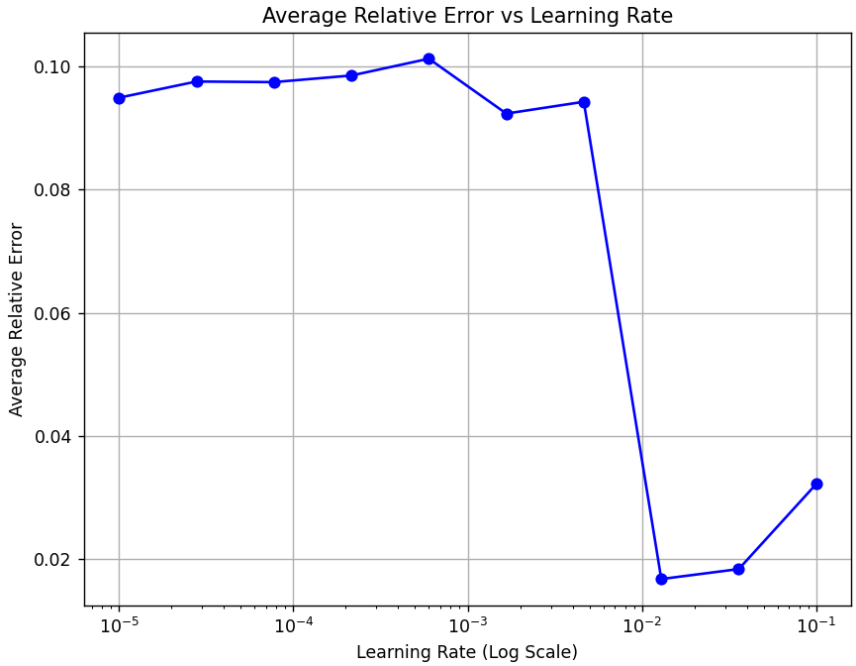

Figure 2. Mean Square Error of the Numerical Solution as the Number of Samples Increases

By evenly dividing nodes within different learning rate intervals and conducting experimental analysis, the optimal learning rate for the algorithm is ultimately determined to be approximately 0.007. At this learning rate, the model demonstrates excellent predictive accuracy in experiments while avoiding oscillations or instability that may arise from an excessively high learning rate, thereby ensuring training stability and convergence.

Furthermore, we analyzed the variation of the average relative error with respect to the number of samples. Experimental results show that as the number of samples increases, the average relative error gradually decreases, indicating that a larger sample size helps improve model accuracy. However, the growth in sample size also leads to a significant increase in computational costs, necessitating a

balance between computational resources and model accuracy in practical applications. By comparing experimental results for different sample sizes, it is observed that when the sample size reaches 100,000, the error stabilizes and meets accuracy requirements, with further increases yielding minimal improvement. Therefore, selecting 100,000 samples as the parameter configuration effectively balances computational efficiency and predictive accuracy, enhancing the practical feasibility of the algorithm while ensuring an efficient and precise solution process.

Table 4. Mean Square Error of the Numerical Solution as the Number of Samples Increases

Samples	Average relative error
2000	0.0184
10000	0.0158
50000	0.0142
100000	0.0136
200000	0.0129
500000	0.0113
1000000	0.0103

4. Conclusion

This paper presents an adaptive algorithm for numerically solving the expected solution of stochastic partial differential equations (SPDEs) using the Feynman-Kac formula. The proposed algorithm successfully addresses several issues arising from the uncertainty of random variables while maintaining the accuracy of the results. Furthermore, the improved adaptive algorithm is combined with neural network techniques to successfully solve the expected solution of SPDEs in high-dimensional cases, ensuring accuracy—something traditional algorithms cannot achieve.

In practical applications, stochastic partial differential equations are widely used in fields such as climate modeling, fluid dynamics, financial modeling, and biology. Especially in high-dimensional scenarios, traditional numerical methods often face the "curse of dimensionality" and computational complexity issues. The adaptive algorithm proposed in this paper effectively overcomes these challenges, enhancing computational efficiency while ensuring accuracy. This makes it particularly suitable for describing high-dimensional random processes, such as meteorological variations, financial market dynamics, and biological population expansion. By accurately capturing randomness in both space and time and combining it with neural networks to further improve computational efficiency and solution accuracy, this paper provides a new and efficient tool for the numerical solution of high-dimensional stochastic partial differential equations.

References

[1] Hawkins K P, Pakniyat A, Tsiotras P. Value function estimators for Feynman–Kac forward–backward SDEs in stochastic optimal control [J]. Automatica, 2023, 158 (000): - 1.

[2] Choi B S, Choi M Y. on Some Results of the Nonuniqueness of Solutions Obtained by the Feynman–Kac Formula [J]. Mathematics (2227-7390), 2024, 12 (1).

[3] Okuma K. An asymptotic expansion of the solution of a semi-linear partial differential equation implied by a nonlinear Feynman–Kac formula [J]. International Journal of Mathematics for Industry, 2024, 16 (01).

[4] He Jie, Hu Shuyan, Ji Jinru, et al. Monte Carlo Simulation Study on Antibiotic Treatment of Streptococcus. Chinese Journal of Infection Control, 2022, 21 (2): 6.

[5] Ren Guangfeng. Research on the Value Assessment of Commercial Real Estate REITs Based on Monte Carlo Simulation and VaR-GARCH Model [D]. Chongqing University of Technology, 2024.

[6] Nie Jingchun, Lu Qiujun. Adaptive Fuzzy Semi-Parametric Time Series Model Based on BP Neural Network. Modeling and Simulation, 2024, 13 (2): 1295 - 1303.

[7] Mathematics; Computational Mathematics. Research on Gradient-Type Algorithms in Deterministic and Stochastic Cases [D]. 2023.

[8] Qiu Yuanyuan, Wang Haochen. Research on Domain Adaptation Object Detection Algorithm Based on Deep Learning. Artificial Intelligence and Robotics Research, 2024, 13 (3): 503 - 514.

[9] Lu Changna, Qian Cunxin, Chang Shengxiang. Application of Adaptive Mesh in Teaching Numerical Methods. Mathematics in Practice and Theory, 2022, 52 (12): 230 - 236.

[10] Wang Jinghui. Monte Carlo and Neural Network Methods for Partial Differential Equations Based on the Feynman-Kac Formula [D]. Shanghai University of Finance and Economics, 2023.