Numerical Solution Method of PDE Based on Neural Network and Feynman-Kac Formula

Jiaqi Yu [1, #], Jingjing Lin [2, *, #], Zhiming Gu [3, #]

[1] Faculty of Maritime and Transportation, Ningbo University, Ningbo, China, 315211

[2] School of Economics Management, Beijing Jiatong University, Beijing, China, 100044

[3] College of Computer and Information Sciences, Fujian Agriculture and Forestry University, Fuzhou, China, 350002

* Corresponding Author Email: 22711051@bjtu.edu.cn

[#]These authors contributed equally.

Abstract. Aiming at the computational difficulty of partial differential equations (PDEs) in high-dimensional problems, this paper proposes a new numerical solution to the uncertainty quantification problem based on the Feynman-Kac formula and probability statistics to effectively deal with the computational difficulty of traditional methods in high-dimensional problems. The curse of dimensionality in the scene and the accuracy limitations in low-dimensional scenes. Specifically, this paper innovatively designs a sampling strategy, which improves the representativeness and statistical reliability of the sample set by increasing the number of sampling points and reducing the number of single-point simulations. At the same time, the polynomial regression model is used to replace the traditional interpolation method to avoid the overfitting problem, and the neural network is combined to deal with the complexity of high-dimensional nonlinear problems. Numerical experiments show that the improved method significantly reduces the error in both low-dimensional and high-dimensional cases, and the relative error of the neural network in high-dimensional problems is kept below 2%, which is better than the polynomial regression model. This study provides new ideas for the numerical solution and uncertainty quantification of high-dimensional partial differential equations and demonstrates its significant advantages in low-dimensional high precision and high-dimensional high efficiency.

Keywords: Partial Differential Equations, Feynman-Kac Formula, Numerical Solution, Neural Network.

1. Introduction

Partial differential equations (PDEs) are equations involving unknown functions of multiple independent variables and their partial derivatives, and are widely used in physics, engineering, finance and other fields. Most partial differential equations (PDEs) in practical problems usually cannot be solved analytically due to their complexity and nonlinear characteristics. Traditional numerical methods include finite element method, finite difference method and finite volume method. The core of these methods is to transform infinite-dimensional operators into finite-dimensional approximation problems through some kind of discretization technology. In the calculation process, the calculation area is first discretized into non-overlapping sub-areas, and then local approximation is performed on these divided sub-domains to achieve numerical solution of the equation. For example, For the boundary value problem of a class of second-order elliptic partial differential equations, Wang Tongke et al.[1] A finite element algorithm is proposed, and the optimal order error estimation of the finite element approximate solution is given.; Zhang Jihong et al.[2]studied the finite difference method for the fourth-order nonlinear partial differential equation, proposed a finite difference format based on the difference quotient approximation, and verified the effectiveness and high accuracy of the method through numerical experiments; Su[3] proposed a solution to the Dirichlet boundary optimal control problem dominated by elliptic partial differential equations on complex connected domains using the Fourier finite volume method, and proved its convergence. Yang Kaili [4] used the cubic finite volume element method to propose a high-precision numerical

format for the one-dimensional convection-diffusion-reaction equation and the Burgers equation and verified its effectiveness through numerical examples.

Although traditional numerical methods have made significant progress, they still face some difficult problems in practical applications. On the one hand, Takahashi et al.[5] pointed out that in high-dimensional problems, as the dimension increases, the sparsity of the data leads to Traditional numerical methods are inefficient and the computational complexity increases dramatically when solving high-dimensional partial differential equations (PDEs). On the other hand, in low-dimensional problems, traditional methods often rely on a lot of computing resources, while random simulations are highly dependent on the number and quality of sample paths. This randomness may lead to large calculation errors and affect the stability of the results. Sex and reliability. Therefore, how to improve the efficiency and accuracy of numerical solutions in high-dimensional and low-dimensional scenarios has become an important topic in the current research on numerical solutions of PDEs.

To address the above problems, this paper proposes two innovative strategies: First, in the application of the traditional Feynman-Kac formula, the solution of PDE usually relies on multiple simulations of Brownian motion sample paths. This method has disadvantages in terms of computational efficiency and result stability. There are obvious deficiencies in this aspect. Wang Jinghui et al.[6] proposed a new numerical solution to the uncertainty quantification problem based on the Feynman-Kac formula and probability statistics method. To this end, this paper improves the sampling strategy and optimizes the multi-point multiple sampling method to a multi-point single sampling method, which improves the computational efficiency and enhances the stability of the solution process; secondly, this paper introduces the idea of neural network and uses Based on the powerful function approximation ability of neural networks, a numerical method for efficiently solving high-dimensional PDEs was designed to effectively alleviate the dimensionality curse problem. Peng Jie et al.[7] used an adaptive neural network structure and an improved optimization algorithm to significantly improve the efficiency and accuracy of solving nonlinear partial differential equations. The simulation experimental results show that compared with the traditional polynomial regression model, the method proposed in this paper shows significant advantages in dealing with high-dimensional problems and exhibits higher prediction accuracy in numerical error analysis. This study provides new ideas for the numerical solution of high-dimensional partial differential equations, expands the application scenarios of neural networks in scientific computing, and further promotes the research and development in the field of uncertainty quantification.

2. Theory and Methods

2.1. Feynman-Kac equation model

This section discusses the problem of solving partial differential equations based on a probabilistic framework, considering the Feynman-Kac equation in partial differential equations. The Laplace equation, also known as the harmonic equation, is a type of partial differential equation [8]. Generally speaking, for the n-dimensional case, the problem of the Laplace equation can be reduced to solving a real function φ that is second-order differentiable with respect to real independent variables $x_1, x_2, \cdots, x_n$. This problem is described in the following form:

$$\sum_{i=1}^{n} \frac{\partial^2 \varphi}{\partial x_i^2} = 0 \tag{1}$$

The above equation can often be simplified as:

$$\nabla^2 \varphi = 0 \tag{2}$$

Or it can be expressed as:

$$\text{div grad}\varphi = 0 \tag{3}$$

Where div represents the divergence of the vector field, the result is a scalar field, grad represents the gradient of the scalar field, the result is a vector field, or it can be simplified as:

$$\Delta\varphi = 0 \tag{4}$$

In this equation, Δ represents the Laplace operator. Then the equation develops into if the right side of the Laplace equation is a given function $f(x_i)(i = 1,2,\cdots n)$, that is:

$$\begin{cases}\Delta\varphi = f, x \in D \\ \varphi = g, x \in \Gamma\end{cases} \tag{5}$$

Then this equation represents the Poisson equation [9]. In it, $\varphi$is an unknown function, and f and g are known functions. D is a bounded region in R^2, Γ is the boundary of D, f is a point in the bounded region D, and g is a point on the boundaryΓ. The Feynman-Kac formula is a mathematical formula that links stochastic differential equations (SDEs) and parabolic partial differential equations (PDEs)[10]. By writing the solutions of certain parabolic partial differential equations as conditional expectations of random processes, the numerical solutions of such differential equations are transformed into paths for simulating random processes. The simulation of the Feynman-Kac formula mainly includes the following steps:

(1) Define a random process X_t, which follows a specific stochastic differential equation (SDE):

$$dX_t = \mu(X_t, t)dt + \sigma(X_t, t)dW_t, \quad X_0 = x \tag{6}$$

Among them, $\mu(X_t, t)$ is the drift term, $\sigma(X_t, t)$ is the diffusion term, and W_t is the standard Brownian motion.

(2) Given the PDE:

$$\frac{\partial u}{\partial t} + \mu(x, t)\frac{\partial u}{\partial x} + \frac{1}{2}\sigma^2(x, t)\frac{\sigma^2 u}{\partial x^2} - V(x, t)u = f(x, t) \tag{7}$$

Satisfy with the boundary conditions:

$$u(x, T) = \psi(x) \tag{8}$$

Where μ, σ, ψ, V are known functions and T is a given parameter

(3) Select an initial point x and set $X_0 = x$;

(4) From time t to T, use numerical methods (such as Euler-Mariano method, Milstein method, etc.) to approximate the SDE and generate a discrete time series of the random process X_t;

(5) For each path, calculate the integral $\int_t^T e^{-\int_t^s V(X_\tau)d\tau} f(X_s, s)ds$ and the boundary condition $e^{-\int_t^T V(X_\tau)d\tau}\psi(X_T)$;

(6) Repeat steps 3 and 4 multiple times to generate a large number of random paths and phase integral values and boundary condition values.

(7) Take the average of the values of $\int_t^T e^{-\int_t^s V(X_\tau)d\tau} f(X_s, s)ds + e^{-\int_t^T V(X_\tau)d\tau}\psi(X_T)$ for all simulated paths to approximate the expected value.

Through the Feynman-Kac formula, this paper has the following solution:

$$u(x) = E_x\left[g(X_{\tau_D}) + \int_0^{\tau_D} f(X_s)ds\right] \tag{9}$$

Where τ_D is a random variable, which represents the time when the random process first reaches the boundary of region D, and X_{τ_D} is the value of the random process at this time. X_s refers to a random process. According to this formula, the solution of the equation is converted into the expected operation, so that Monte Carlo simulation can be used to generate samples to solve the numerical solution of the partial differential equation [11].

2.2. Solution of Feynman-Kac Problem Based on Polynomial Regression and New Sampling Method

In traditional methods, solving the solution of the equation in the domain of definition usually involves the following steps: take N points from the domain of definition as sample points, simulate each point a certain number of times, and obtain the simulated value of each point. For each point, the traditional sampling method is used, which is characterized by the need to perform multiple simulations for each sample point to ensure the accuracy and reliability of the results. Subsequently, a global solution model is constructed using the simulation results of these sample points through interpolation methods. Specifically, by interpolating the simulated values of each sample point, a continuous function that can cover the entire domain of definition is constructed to obtain the exact solution to the equation. The specific steps of the interpolation model are as follows:

(1) This paper uses the Feynman-Kac formula above to randomly select N points, simulate each point M times, and obtain a set of data points $(x_i, y_i, u(x_i, y_i))$, which will be used to establish the subsequent interpolation model.

(2) Let the point to be predicted be (x, y), and find the two nearest training data points by calculating the Euclidean distance from the point (x, y) to each training data point.

(3) Then perform linear interpolation calculation. First calculate the interpolation coefficient t_x in the x direction:

$$t_x = \frac{x - x_1}{x_2 - x_1} \tag{10}$$

Then calculate the interpolation coefficient t_y in the y direction:

$$t_y = \frac{y - y_1}{y_2 - y_1} \tag{11}$$

Finally calculate $u(x, y)$:

$$u(x, y) = u_1 + (u_2 - u_1) * t_x * t_y \tag{12}$$

In this interpolation model, this paper adopts the traditional point selection method, that is, the multi-point multiple sampling method. By randomly selecting multiple initial points, multiple random paths are simulated for each initial point. In addition, this paper also establishes a polynomial regression model to solve the equation. Specifically, polynomial regression can enhance the fitting ability of the model by introducing nonlinear terms, which is suitable for a wide range of data distributions. At the same time, it can greatly reduce the dependence on a large number of random simulations and effectively improve computational efficiency. In the actual application process, the degree of the polynomial and the parameters of the regression equation can be flexibly adjusted according to the specific problem requirements and data characteristics to achieve the best prediction effect [12]. The specific steps are as follows:

(1) This paper uses the Feynman-Kac formula above to randomly select N points, simulate each point M times, and obtain a set of data points$(x_i, y_i, u(x_i, y_i))$, which will be used for subsequent polynomial regression.

(2) This paper generates polynomial features. By combining the original independent variables with different powers, the nonlinear relationship between the independent variables and the dependent variables can be captured.

The matrix representation of the generated feature matrix X is as follows:

$$X = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & y_1^2 & \cdots & x_1^n & y_1^n & x_1^{n-1}y_1 & x_1 y_1^{n-1} \\ 1 & x_2 & y_2 & x_2^2 & y_2^2 & \cdots & x_2^n & y_2^n & x_2^{n-1}y_2 & x_2 y_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_i & y_i & x_i^2 & y_i^2 & \cdots & x_i^n & y_i^n & x_i^{n-1}y_i & x_i y_i^{n-1} \end{bmatrix} \tag{13}$$

(3) Estimate the coefficients of the polynomial regression model. In this paper, the polynomial regression model is assumed to be:

$$u(x,y) = \beta_0 + \sum_{i=1}^{n} \beta_i x^i + \sum_{j=1}^{n} \beta_{n+j} y^j + \sum_{i=1}^{n} \sum_{j=1}^{n-j} \beta_{n+n+i+j} x^i y^j + \epsilon \tag{14}$$

Where $\beta_i$is the estimated coefficient and ϵ is the error term. The coefficient β_i is solved by minimizing the residual sum of squares, which is expressed as follows:

$$\sum_{i=1}^{n} \left(u(x_i, y_i) - \hat{u}(x_i, y_i)\right)^2 \tag{15}$$

Where n is the number of data points and $\hat{u}(x_i, y_i)$ is the model prediction.

(4) Use the trained polynomial regression model for prediction

(5) In a given area, use a random number generator to generate 100 points(x_i, y_i), according to the known equation:

$$u(x,y) = e^{2(x+y)} \tag{16}$$

Calculate the true value of u . At the same time, use the trained polynomial regression equation to calculate the predicted value$\hat{u}(x_i, y_i)$ for each random point (x_i, y_i).

Then calculate the absolute error (AE) of each random point:

$$AE = |u(x_i, y_i) - \hat{u}(x_i, y_i)| \tag{17}$$

Calculate the relative error (RE) for each point:

$$RE = \frac{|u(x_i, y_i) - \hat{u}(x_i, y_i)|}{|u(x_i, y_i)|} \tag{18}$$

(6) Calculate the meaning of absolute error (MAE):

$$MAE = \frac{1}{n} \sum_{i=1}^{10000} |u(x_i, y_i) - \hat{u}(x_i, y_i)| \tag{19}$$

Calculate the mean relative error (MRE):

$$MRE = \frac{1}{n} \sum_{i=1}^{10000} \frac{|u(x_i, y_i) - \hat{u}(x_i, y_i)|}{|u(x_i, y_i)|} \tag{20}$$

While using the polynomial regression method, this paper improves the sampling method. The new sampling strategy adopted in this section is to select more data points and simulate each point only once. The original small number of sampling points made it difficult to effectively balance the randomness of the error. If the error of a certain point is large, due to the limited number of data points, the sampling points are not enough to offset the error, resulting in a large deviation in the overall estimate. By increasing the number of sample points, the potential space of the PDE solution is more comprehensively covered, and the representativeness and statistical reliability of the sample set are improved without increasing the burden of a single calculation.

2.3. Solution of high-dimensional Feynman-Kac problem based on neural network model

In high-dimensional scenarios, polynomial regression and interpolation models face serious dimensionality curse problems. As the dimension of polynomial regression increases, computational complexity increases exponentially, the model performance decreases significantly, and the risk of overfitting increases, affecting the generalization ability of the model. The interpolation model requires a lot of neighborhoods point information to accurately interpolate a point in high-dimensional

space. However, due to the sparse data, it is difficult to find enough suitable neighborhood points, which reduces the accuracy of the interpolation result.

In contrast, neural networks can effectively alleviate the problem of the curse of dimensionality to a certain extent due to their strong adaptability and ability to learn complex patterns [13]. Through multi-layer neuron connections and weight adjustments, neural networks can automatically extract key features without the need to manually design complex functions. In high-dimensional space, neural networks can capture the intrinsic structure of data and find effective solutions through iterative learning and parameter optimization even when faced with sparse data distribution [14].

This paper adopts the MLP architecture, aiming to give full play to the advantages of neural networks in high-dimensional nonlinear problems and overcome the limitations of traditional methods. The specific steps are as follows:

(1) Based on the Feynman-Kac formula, this paper randomly selects 10,000 points and simulates each point once to obtain a set of data points$(x_1, x_2, \ldots, x_n, u(x_1, x_2, \ldots, x_n))$,These data points will be used for subsequent neural network training.

(2) This paper generates an n-dimensional input feature matrix X. Each input feature corresponds to a dimension in the n-dimensional space, and the feature matrix is in the form of

$$X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(N)} & x_2^{(N)} & \cdots & x_n^{(N)} \end{bmatrix} \tag{21}$$

In order to capture the nonlinear relationship between independent variables and dependent variables, this paper also performed feature engineering to generate an extended feature matrix including square terms and interaction terms.

(3) This paper defines a neural network model, where the input layer contains n neurons, the hidden layer consists of two layers with 8 neurons, and the output layer contains one neuron. Through weighted summation and ReLU activation function, the neural network can effectively extract the features of the input data.

(4) Use the trained neural network model for prediction. In this paper, 100 points $(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})$ are randomly generated in a given n-dimensional region, and calculate the true value $u\left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right)$ based on the known equation. At the same time, the trained neural network model is used to calculate the predicted value $\hat{u}\left(x_1^{(i)}, x_2^{(i)}, \cdots, x_n^{(i)}\right)$ for each random point.

(5) In the model evaluation stage, this paper calculates the mean absolute error (MAE) and mean relative error (MRE) to measure the prediction performance of the model.

In this way, neural networks can not only reduce reliance on a large number of random simulations but also improve computational efficiency and predictive accuracy.

3. Numerical Analysis

3.1. Comparative analysis of errors of new sampling methods

In this section, a two-dimensional model example is given. Specifically, the domain is given $D = [0,1]^2$, The boundary conditions are:

$$u = g \tag{22}$$

The g function is given as:

$$g(x, y) = e^{2(x+y)} \tag{23}$$

The f function is:

$$f(x,y) = -4e^{(x+y)} \tag{24}$$

The expression of the u function is:

$$u(x,y) = E_{x,y}\left[g(X_{TD}) + \int_0^{TD} f(X_s)d_s\right] = e^{2(x+y)} \tag{25}$$

In order to reflect the rationality of the sampling method, the polynomial regression model under the multi-point single sampling method and the multi-point multiple sampling method is compared. Specifically, combined with the above example, the polynomial regression solution method based on the new sampling strategy is used to solve the Feynman-Kac formula. The sampling methods are:$N = 5,\ M = 5000$ and N=25000, M=1, Where N is the number of simulation points, and M is the number of simulations for each point. The time step is 0.001. The prediction results under different sampling methods are shown in Figure 1 below. When the sampling method is N=25000, M=1, and the time step is 0.001, the average absolute error of 10,000 new prediction points is 0.1161, and the average relative error is 1.2937%. When the sampling method is $N = 5,\ M = 5000$, and the time step is 0.001, the average absolute error of 10,000 new prediction points is 0.9106, and the average relative error is 7.2143%. For the comparison of the average relative errors of two different sampling methods in the two-dimensional polynomial regression model, it can be seen that the multi-point single sampling method has a better effect. When the multi-point single sampling method is used, although the error of a single point is uncertain, the large number of data points makes these random errors offset and balance each other as a whole. As a result, the overall average error tends to a relatively stable value; when the multi-point multiple sampling method is used, the small number of sampling points makes it difficult to effectively balance the randomness of the error. Moreover, due to the large number of simulations for each point, excessive attention may be paid to the local characteristics of individual points, while the overall distribution of the data is ignored.

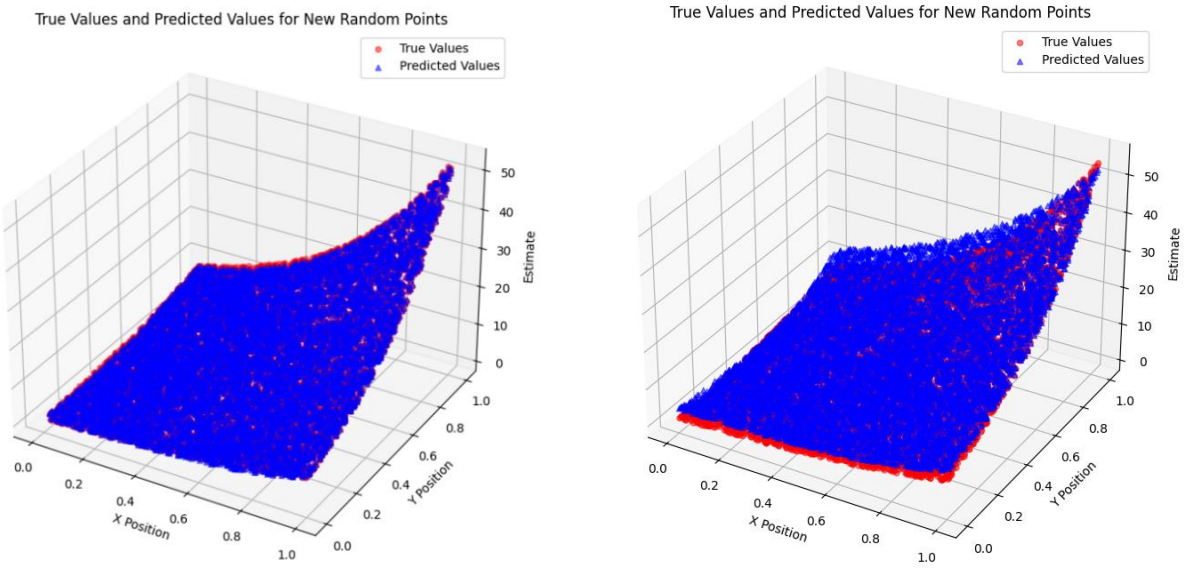

Fig. 1 Two-dimensional polynomial regression model fitting diagram (left) $N = 25000, M = 1$, (right) $N = 5, M = 5000$)

Next, under the same sampling method, the interpolation model and the polynomial regression model are used to predict 10,000 points. By comparing the average relative errors of the two models, it can be seen that the average relative error of the polynomial regression model is significantly smaller than the average relative error obtained by the interpolation model. The straight line fitted by the interpolation model needs to pass through every sample point, and this feature makes it very easy to overfit. The polynomial regression model can capture the potential trends and patterns in the data, not just through the data points, and has better generalization ability.

Tab. 1 Prediction Error Analysis of Difference Model and Polynomial Model

	Mean absolute error	Mean relative error
Interpolation Model	3.7465	48.1600%
Polynomial regression model	0.9106	7.2143%

3.2. Comparative Analysis of Errors in Numerical Solutions of High-Dimensional PDE

Taking the four-dimensional case as an example, the domain $D = [0,1]^4$ is a four-dimensional unit hypercube, which contains all points that satisfy $0 \leq x_1, x_2, x_3, x_4 \leq 1$. In order to verify the accuracy of the numerical solution, the analytical solution is selected as

$$u(x_1, x_2, x_3, x_4) = \exp\left(\frac{x_1 + x_2 + x_3 + x_4}{4}\right) \tag{26}$$

From this, the source term function is

$$f(x_1, x_2, x_3, x_4) = -\frac{1}{8}\exp\left(\frac{x_1 + x_2 + x_3 + x_4}{4}\right) \tag{27}$$

The boundary conditions are given directly by the analytic solution:

$$g(x_1, x_2, x_3, x_4) = u(x_1, x_2, x_3, x_4) = \exp\left(\frac{x_1 + x_2 + x_3 + x_4}{4}\right), (x_1, x_2, x_3, x_4) \in \partial L \tag{28}$$

Simulations of Brownian motion are used to compute numerical solutions to the Poisson equation. Initial positions $(x_1, x_2, x_3, x_4) \in [0,1]^4$ are randomly generated with a time step $dt = 0.001$. The Brownian motion path is simulated until the Brownian motion first leaves the domain of definition D and the path integral $\int_0^{\tau_D} f(X_s)ds$ and the boundary value $g(X_{\tau_D})$. In this paper, a four-dimensional neural network model is used for fitting, where $dt = 0.001$ and the number of simulations is 10,000. The experimental results show that the average absolute error of the proposed method is 0.0081 and the average relative error is 0.4877%. The visualization results are shown in Figure 2 below.

Figure 2 illustrates the error distribution of a multidimensional dataset by fixing the value of one dimension and analyzing the characteristics of the error distribution among the other dimensions. Each subplot fixes one dimension w_0, z_0, y_0 or x_0, respectively, and shows the error distributions of the remaining three dimensions in the form of a 3D scatter plot. Color bars are used to indicate the magnitude of the error, where blue indicates a smaller error and red indicates a larger error. The small errors for most of the data points indicate that the model predicts more accurately in most regions.

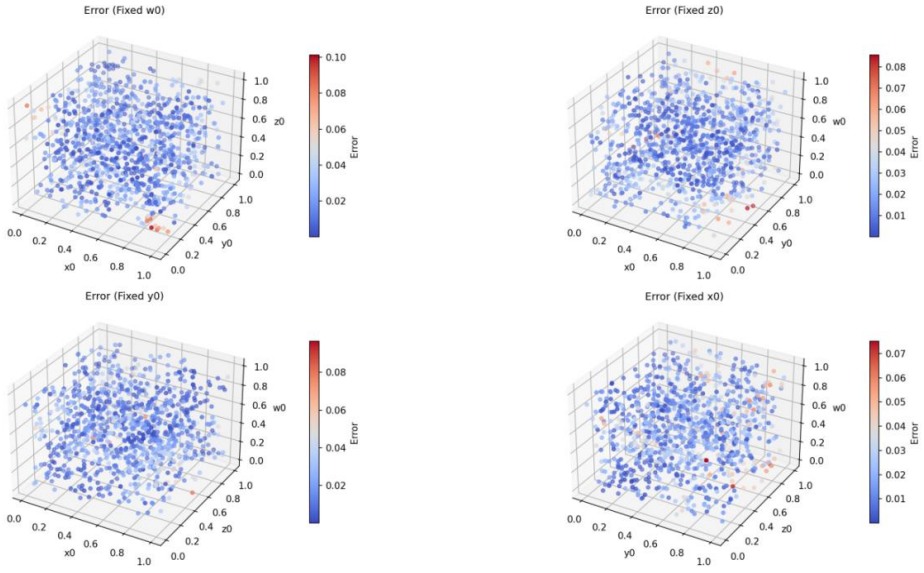

Fig. 2 Visualization of 4D neural network fitting

Figure 3 (left) illustrates the cumulative distribution function (CDF) of the relative errors of the model predictions. The horizontal axis represents the relative error, which is calculated as the absolute difference between the predicted value and the true value divided by the true value; the vertical axis represents the cumulative proportion of samples with relative errors less than or equal to a certain value. It is clear from the figure that most of the samples have small relative errors. About 80% of the samples have relative errors less than 0.02, and close to 95% of the samples have relative errors less than 0.04. This indicates that the model has a high accuracy in most predictions, with errors concentrated in a small range.

In addition, in order to compare the results of the neural network and the polynomial, the polynomial is used to fit the model again in this paper. The four-dimensional polynomial regression model is taken with time step $dt = 0.001$ and 10,000 simulations, and the calculations show an average absolute error of: 0.2007 and an average relative error of: 12.7402%. The visualization results are shown in Figure 3 (right) below. From the cumulative distribution of the polynomial regression relative errors, it can be seen that about 50% of the samples have a relative error lower than 10.70%, and 90% of the samples have a relative error lower than 27.49%, and it can be clearly seen that the polynomial relative error will be significantly higher than that of the neural network model in the four-dimensional case.

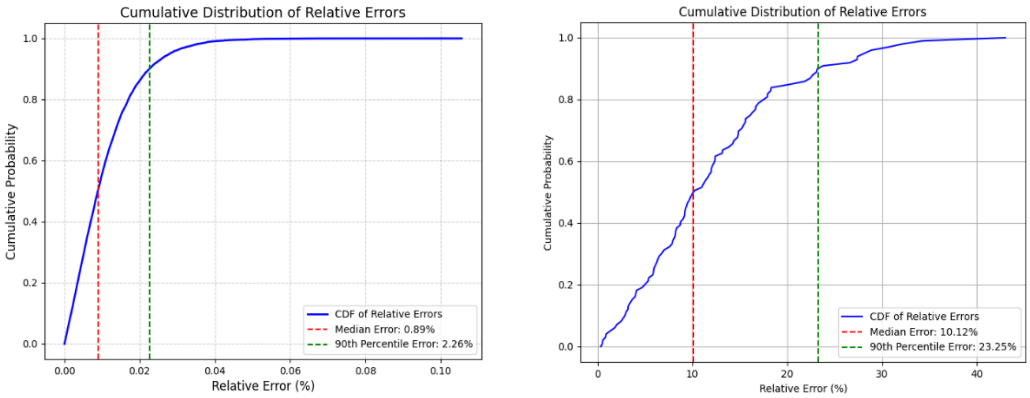

Fig. 3 Four-dimensional neural network relative error accumulation distribution (left) and four-dimensional polynomial regression relative error accumulation (right)

Next, in this paper, the number of dimensions is increased from one to seven and the two methods of polynomial regression and neural network are compared to solve the PDE, and the specific results are shown in Fig. 4 above. From Fig. 4, it can be seen that polynomial regression shows an increasing trend with the increase of dimension, while the relative errors of the neural network model from one to seven dimensions are kept below 2%. It can be concluded that when the dimension is greater than two, the training effect of the neural network is significantly better than the polynomial regression model.

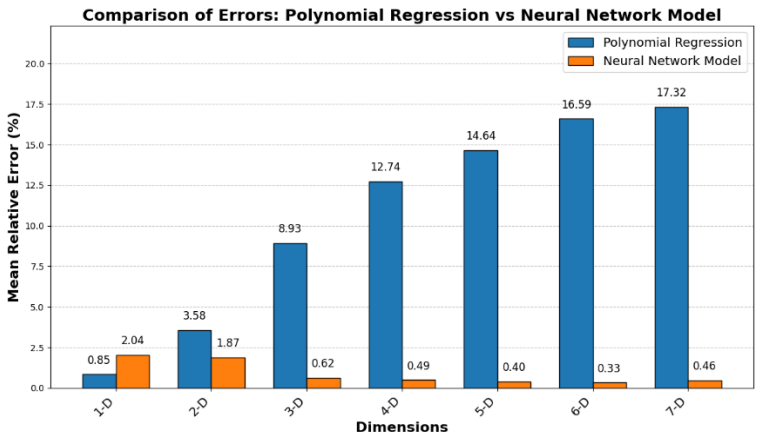

Fig. 4 Comparison of 1–7-dimensional polynomial regression and neural network error results

This paper will then perform a convergence analysis of the numerical solution methods. Specifically, this paper uses the control variable method to investigate the convergence of the polynomial regression modeling algorithm and the neural network algorithm. For the polynomial regression algorithm, the effects of time step, number of simulations and polynomial order on the global solution error are examined separately.

First, the time step is fixed at 0.01, the number of simulations is 1000, and the polynomial order is 2nd order, and the effect of its change on the global solution error is observed by adjusting the time step, and the results are shown in Table 2 and Fig. below. As can be seen from the graphs, both the mean absolute error (MAE) and the mean relative error (MRE) decrease significantly as the time step dt decreases. This indicates that smaller time steps help to improve the accuracy of the simulation.

Tab. 2 Numerical analysis of the effect of time step variation on error

Time step	Mean absolute error	Average relative error (percent)
0.1	6.8738	69.4657%
0.05	2.9193	30.4342%
0.01	0.3941	5.4666%
0.005	0.2908	2.8324%
0.001	0.1411	1.8606%
0.0005	0.1208	1.4759%
0.0001	0.1295	1.5987%
0.00005	0.1193	1.6472%
0.00001	0.1036	1.3511%

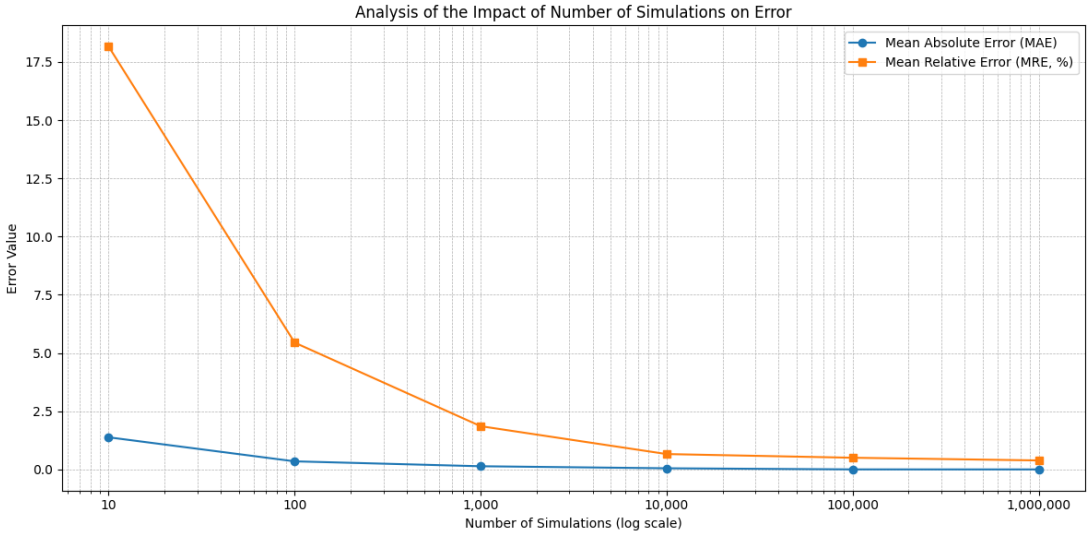

Fig. 5 Relationship between simulation times and mean absolute error and mean relative error

Tab. 3 Numerical analysis of the effect of simulation times on error

Number of simulations	Mean absolute error	Average relative error (percent)
10	1.3830	18.1588%
100	0.3514	5.4485%
1000	0.1411	1.8606%
10000	0.0538	0.6639%
100000	0.0062	0.5060%
1000000	0.0048	0.3903%

Secondly, keeping the time step and polynomial order constant, the number of simulations is gradually increased to evaluate the effect of the number of simulations on the error, and the results

are shown in Table 3 and Fig. 6 below. From the above graphs, it can be seen that when the number of simulations is small, the error is relatively high, the average absolute error and the average relative error are further reduced, and the simulation results are gradually close to the real values.

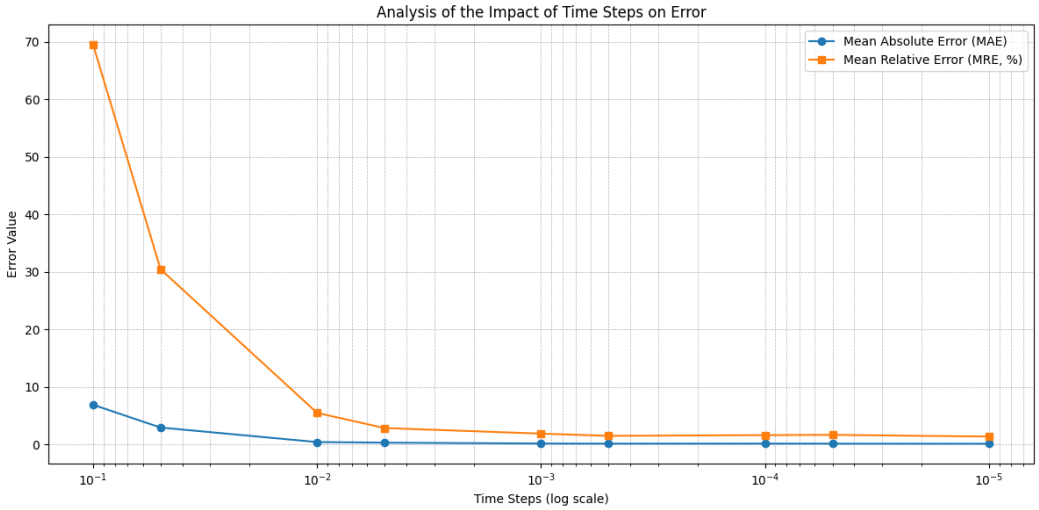

Fig. 6 Relationship between simulation times and absolute error and mean relative error

Subsequently, by fixing the time step size and the number of simulations while varying the polynomial degree, we analyze the fluctuations in the global solution error at different degrees, as presented in Table 4 and Figure 7. From the figures, it can be observed that as the polynomial degree increases, both the mean absolute error and the mean relative error initially rise and then stabilize. This indicates that with the increase in model complexity, the model may fit the training data better; however, the generalization capability does not necessarily improve.

Tab. 4 Numerical analysis of the influence of polynomial order variation on error

Degree	Mean absolute error	Average relative error (percent)
1	0.0076	0.6357%
2	0.0538	0.6639%
3	0.1478	1.6771%
4	0.1578	2.1173%
5	0.1605	1.8657%
6	0.1733	2.4298%
7	0.1912	2.3857%

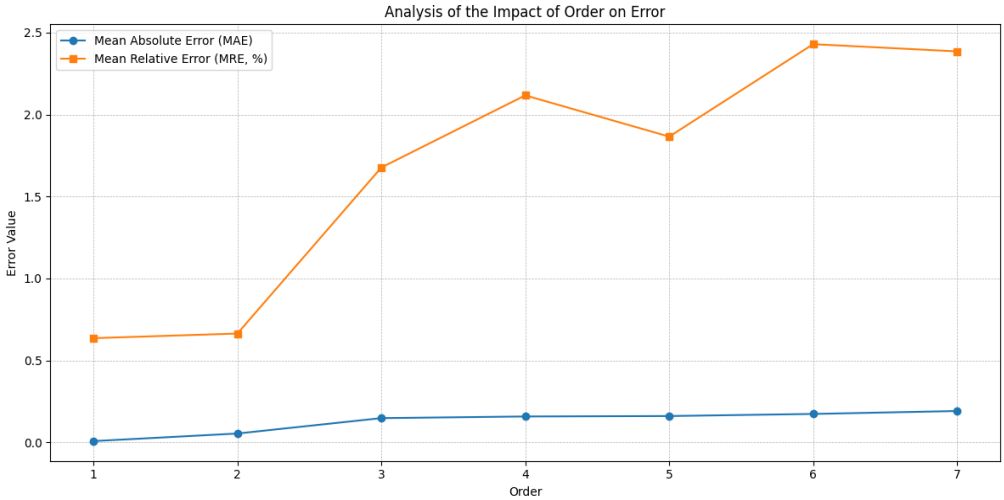

Fig. 7 Polynomial order variation and mean absolute error and mean relative error sum

Finally, this section will study the factors that affect the convergence behavior of neural networks, including learning rate and iteration times. By analyzing these factors, this paper aims to provide insight into how they affect the training process and the overall efficiency of the model. In this study, in order to systematically evaluate the effect of learning rate on the performance of the model, a series of learning rate values are generated using the method of linear interval. 50 learning rates are uniformly generated from 0.1 to 0.0001. If other parameters remain unchanged, this paper will evaluate the training results of the model under each learning rate and analyze its impact on the average relative error to determine the best learning rate setting.

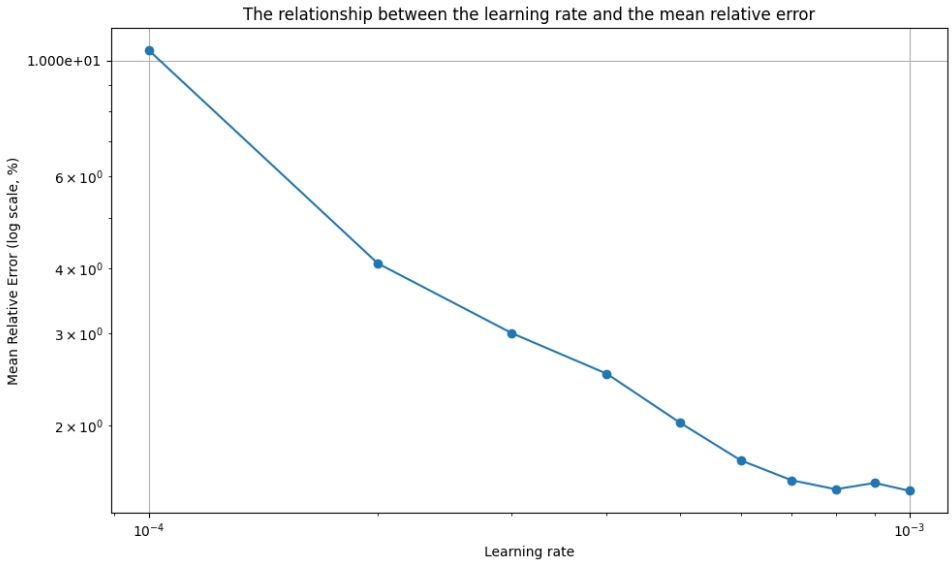

Fig. 8 Relationship between learning rate and relative error

As can be seen from Figure 8, there is a significant negative correlation between learning rate and average relative error. At a small learning rate (e.g. 10^{-4}), the error is about 10%; As the learning rate increases to 10^{-3}, the error decreases rapidly and tends to be stable, indicating that properly increasing the learning rate can significantly improve the model accuracy, but the effect of excessively increasing the learning rate decreases.

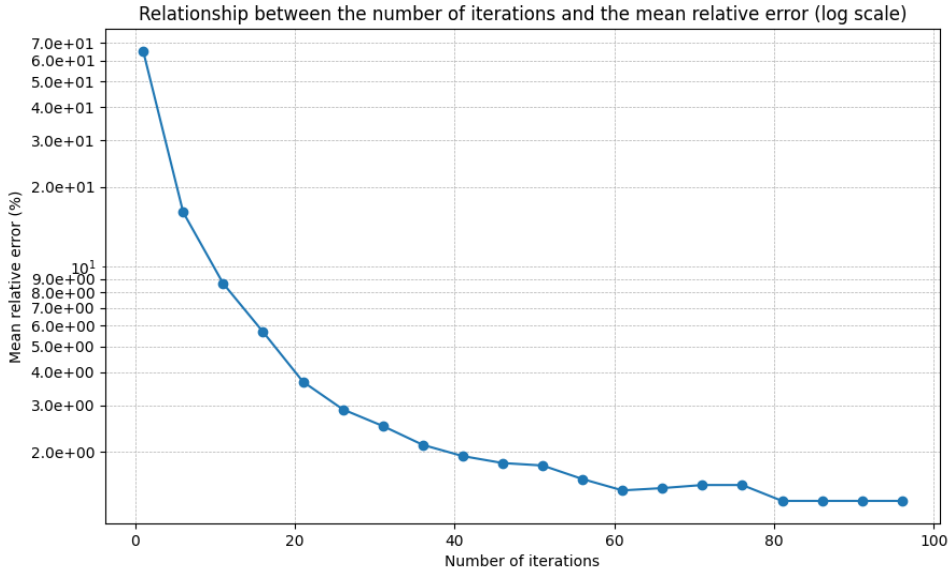

Fig. 9 Relationship between number of iterations and relative error

To evaluate the effect of the number of iterations on model performance, we generate 20 iterations every 5 times in a range of 1 to 100, keeping other parameters constant. By evaluating the training results of the model under each iteration, the influence of the model on the average relative error was

analyzed, and the optimal iteration number was determined. As can be seen from Figure 9, there is a significant negative correlation between the number of iterations and the average relative error. In the initial stage (the number of iterations is less than 20), the average relative error quickly drops from about 70% to less than 10%, showing the high learning efficiency of the model. After more than 60 times, the average relative error stabilizes at about 2%.

4. Conclusions and Prospect

In this paper, we propose a Monte Carlo simulation method based on the Feynman-Kac formulation combined with polynomial regression and neural network techniques for uncertainty quantization problems in the numerical solution of partial differential equations (PDEs). To address the dimensional catastrophe and computational challenges of high-dimensional problems, the improved sampling strategy enhances the statistical reliability of the samples, polynomial regression reduces overfitting in low-dimensional problems, and neural networks mitigate the dimensional catastrophe in high-dimensional problems. Numerical experiments show that the method significantly reduces errors in both low and high dimensional cases, achieves the balance between efficiency and accuracy, and has higher computational efficiency and generalization ability compared with the traditional interpolation method, which provides a new idea for numerical solution of PDE.

Despite the remarkable results, the demand for computational resources remains a challenge in high-dimensional problems. In the future, more efficient deep learning architectures or distributed computing techniques can be introduced to reduce the computational resource requirements and improve the high-dimensional applicability. Meanwhile, the method can be extended to more complex PDE types and boundary conditions and combined with adaptive sampling or reinforcement learning techniques to optimize the sampling efficiency of high-dimensional sparse data.

References

[1] Wang Tongke, Hu Qingwen. Finite element algorithm for boundary value problems of second-order elliptic partial differential equations [J]. Journal of Henan Normal University (Natural Science Edition), 1999, 27(01): 18-21. (In Chinese)

[2] Zhang Jihong, Li Yongfei, Wang Jiexin, et al. Fourth order nonlinear partial differential equations of finite difference method to study [J]. Journal of dalian jiaotong university, 2020, 9 (02): 109-112. The DOI: 10.13291 / j.carol carroll nki djdxac. 2020.02.022.

[3] Su, Mengya, Xie, Liuqing, Zhang, Zhiyue. Numerical Analysis of Fourier Finite Volume Element Method for Dirichlet Boundary Optimal Control Problems Governed by Elliptic PDEs on Complex Connected Domains [J]. MATHEMATICS, 2022,10(24).

[4] Yang Kaili. Two types of partial differential equation of the finite volume element method [D]. Three times in Inner Mongolia autonomous region, Inner Mongolia university, 2021. The DOI: 10.27224 /, dc nki. Gnmdu. 2021.000480.

[5] Beck, Christian, Gonon, Lukas, Jentzen, Arnulf. Overcoming the curse of dimensionality in the numerical approximation of high-dimensional semilinear elliptic partial differential equations [J]. PARTIAL DIFFERENTIAL EQUATIONS AND APPLICATIONS,2024,5(06).

[6] Wang Jing-Hui. Monte Carlo and Neural Network Method for Partial Differential Equations based on Feynman-Kac Formula [D]. Shanghai University of Finance and Economics, 2023. DOI: 10.27296 /, dc nki. Gshcu. 2023.000323.

[7] Peng Jie, ZHANG Yuwu. Research on solving PDEs based on Adaptive Neural network [J]. Journal of Jiamusi University (Natural Science Edition), 2024, 42(03): 174-177.

[8] Zhang Juan, Yin Junping, Wang Ruili. Basic Framework and Main Methods of Uncertainty Quantification [J]. Mathematical Problems in Engineering, 2020

[9] Science: Studies from Indian Institute of Technology Delhi Yield New Information about Science (Uncertainty quantification-based cloud parameterization sensitivity analysis in the NCAR community atmosphere model) [J]. Science Letter, 2020

[10] Pınar Acar. Recent progress of uncertainty quantification in small-scale materials science [J]. Progress in Materials Science, 2020,117

[11] G. Castellazzi, P. Krysl. A nine-node displacement-based finite element for Reissner–Mindlin plates based on an improved formulation of the NIPE approach [J]. Finite Elements in Analysis & Design, 2012,58

[12] Hao L, Tian W. Analysis of a WSGD scheme for backward fractional Feynman-Kac equation with nonsmooth data [J]. Advances in Computational Mathematics, 2024, 50(4): 87-87.

[13] Mathematics - Computational Mathematics; Studies from Peking University Describe New Findings in Computational Mathematics (Relu Deep Neural Networks and Linear Finite Elements) [J]. Network Weekly News, 2020

[14] Ma Kangkang. Sparse against neural network research and application in the recommendation system [D]. Sichuan: University of electronic science and technology, 2021.