MATH3999 Directed Studies in Mathematics

Applications of Deep Learning Framework to Accelerate the Solutions of Parabolic PDEs

Zhang Maoqi, 3035534347
supervised by Dr.LI, Guanglian

Dec 10, 2021

Abstract

This report aims at exploring the applications of the machine learning framework proposed by S. Mishra on parabolic PDEs. The framework and corresponding algorithm aims at accelerating the solutions of time-dependent PDEs and ODEs. The method is based on modeling the current numerical methods as an artificial framework with trainable parameters. By building up our loss function and determining the parameters, the PDEs can be solved for particular points in the range of the function. As a continuation of the summer research which mainly focused on ODEs, this report aims at applying the framework to parabolic PDEs and its variation. Basic heat equations and its variation with non-constant coefficient are used as illustrative examples. The efficiency and limitations of the algorithms will be discussed in the applications.

Contents

1 Background

Parabolic PDEs are a type of second-order partial differential equations which u takes the following form (1),

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0 \tag{1}$$

while its coefficients satisfy the condition $B^2 - AC = 0$.

A basic example of parabolic PDEs is one-dimension heat equation, $u_t = ku_{xx}$ where k is a constant.

$$\begin{cases} u_t - ku_{xx} = 0 \\ u|_{t=0} = u_0(x) \end{cases} \tag{2}$$

Heat equation can be explicitly solved under special circumstances. i.e. for PDE (2) on the whole real line, given the Dirichlet boundary condition at x=0, it can be solved using heat kernel (3).

$$S(t, x-y) := \frac{1}{\sqrt{4k\pi t}} e^{-\frac{(x-y)^2}{4kt}} \tag{3}$$

$$u(t,x) = \int_{-\infty}^{+\infty} S(t, x-y) u_0(y)\, dy \tag{4}$$

Moreover, the non-homogeneous PDEs (5) can be solved by Duhamels Principle in (6).

$$\begin{cases} u_t - ku_{xx} = f(x,t) \\ u|_{t=0} = u_0(x) \end{cases} \tag{5}$$

$$u(t,x) = \int_{-\infty}^{+\infty} S(t, x-y) u_0(y)\, dy + \int_0^t \int_{-\infty}^{+\infty} S(t-s, x-y) f(s,y)\, dyds \tag{6}$$

Another method for explicitly solving homogeneous PDEs is separation of variables . This method can solve the PDEs as a sum of infinite series (depending on the given Dirichlet boundary condition)

In reality, PDEs has seen wide applications in modeling in different scientific and engineering subjects. The ubiquity and significance of PDEs has establish the importance of the problem. Many existing methods have been well-established for numerically solving PDEs. The most widely used numerical schemes for PDEs include: finite difference, finite volume, finite element ((Hairer et al., 1991)), etc.

However, as pointed out by S. Mishra, when it comes to problems with high degrees, which require approximations of ODEs and PDEs in a very fast manner, the current schemes remain insufficient for realistic computation. As proposed by Mishra (2018), utilizing the methodology of machine learning framework, we can obtain a model that produce more accurate result with much less time under special circumstances.

This report applies S. Mishra's framework to numerical schemes of ODEs and PDEs, and it is observed that these schemes are well suited to approximating well-known ODEs and PDEs with the right choices of parameters. On a coarse grid, approximating ODEs and PDEs can be computed cheaply but end up with poor accuracy using traditional methods. However, using the machine learning framework, the accuracy of results on a coarse grid can

be improved significantly while maintaining a low computational cost. The report validates the feasibility of this framework with other varied conditions. Illustrative examples are provided to substantiate the results.

2 Methodology

2.1 Numerical Scheme of PDEs

For finite difference scheme of PDEs, numerical schemes must satisfy convergence, consistency, and stability. The numerical schemes of PDEs can be divided into two kinds, explicit schemes and implicit schemes (LeVeque, 2007). Explicit schemes can be expressed in the form (7).

$$u(t + \Delta t) = F(u(t)) \text{ for some function F} \tag{7}$$

Forward Euler Scheme is one example of explicit schemes, where $\frac{du(t)}{dt} = \frac{u(t+\Delta t)-u(t)}{\Delta t}$.

More generally, most schemes are implicit forms that can only be written as:

$$G(u(t + \Delta t), u(t)) = 0 \text{ for some function G} \tag{8}$$

For explicit scheme, the solution to the equation might be: 1) has no solution; 2) has two or more solution in the domain; 3) has exactly one solution. Since the numerical scheme must satisfy consistency, only one solution of the function u is allowed at one point. Efficient algorithms are usually iteration methods with good starting approximation. It is discussed in Hairer et al. (2006) that Newton's Iteration and Fixed Point Iteration are two mainly used algorithms for iteration scheme of solution. The error of Newton's Iteration is second-order convergent while fixed point iteration only admits linear convergence.

In this report, the starting point of the iteration is chosen to be the previous point. (i.e. $\theta_0 = \theta_n$ at point n+1) The implicit scheme has much more applications since it is more stable than explicit scheme. For example, comparing Forward Euler and Backward Euler for ODEs. It can be seen that backward Euler Method is unconditionally stable, while Forward Euler Method is only stable when the mesh size is small enough.

2.2 Machine Learning Framework

In general, machine learning framework is a model that contains a series of trainable parameters that will be trained with initial data and has prediction power when given input. The internal architecture of the machine learning model is closed to neural network, with units connected by different signals. It is modeled as a multi-layer neural network with hidden layers and unhidden layers. Machine learning is classified according to the nature of the feedback available to the systems as supervised learning, unsupervised learning, and reinforcement learning ((Goodfellow et al., 2016)). In the algorithm developed in the report, it belongs to supervised learning.

The machine learning framework that is used in the PDE algorithm is simplified to one single layer, in which the loss function is built by the proposed scheme. The details of its structure can be seen in Figure 6.

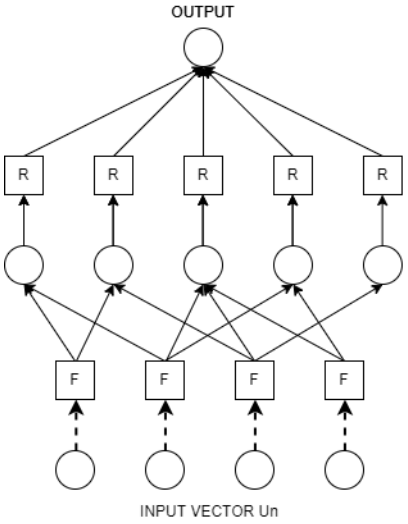

Figure 1: Single-Layer Machine Learning model proposed by S. Mishra. F in square is inherent equation that represents the PDEs, while R in square is activation function. The circle represents the neuron, which are the non-linear maps. The netted arrow in the middle is the neural network, which is linear map. Hidden layers represents the step of Newton's iteration when solving implicit scheme. For PDEs solver, the output is the parameter θ that can quickly solves PDE on coarse grid using given scheme.

2.3 General Algorithm

The general algorithm proposed by S. Mishra is presented in details in **Algorithm 1**. Note that the numerical scheme in Step 1 is variation of finite difference method in this report. It can be a combination of different methods with different weights.

2.4 Stochastic Gradient Descent Method

According to Ruder (2016), Gradient descent method is Given defined and differentiable function $F(\omega)$.

$$\omega_{n+1} = \omega_n - \eta \nabla F(\omega) \tag{9}$$

Stochastic gradient descent method is developed on the basis of gradient descent method. It is especially useful when the function is in the form

$$Q(\omega) = \frac{1}{n} \sum Q_i(\omega) \tag{10}$$

The set of parameters is denoted by ω.

There are many variations of SCG methods in the process of implementations. i.e. batch SCG or mini-batch SCG. It will be discussed in the next session that in the case studies, the choice of direction of descent cannot be random. It can be seen as a variation of mini-batch method without randomness. The details will be discussed in the next session.

Algorithm 1: Machine Learning Framework PDE solver

Data: A PDE with known constant coefficients and unknown initial boundary condition

Result: A model that can solve given PDEs with any boundary condition and coefficients on the coarse grid

1 Choose a consistent (and stable) numerical method (alternatively neural network) on the coarse grid. At the same time, embed parameters into the scheme. One has the freedom to choose the number or dimensions of the parameters by choosing schemes with different order and also different mesh size.

2 Generate initial data as training sets for the PDEs. (Usually, generate random initial functions as training data. For example, generate Fourier series for Dirichlet boundary condition.)

3 If the PDE is solvable, use exact solution as reference points on the coarse grid. Otherwise, generate the reference points by using classic finite difference method on fine grid and project on the coarse grids as reference points.

4 Set up loss function and use (stochastic) gradient descent method to find the locally optimal parameter. The loss function is the sum of norms of errors of the results calculated in The trained result of the parameter, denoted by θ^*.

$$E(\theta) = \sum_i \sum_{n=1}^{N} \|U^{n,i} - U^{n,i}_{ref}\|^p_{L^p}$$

desired $\theta = \arg\min_{\theta \in D} E(\theta)$ for domain D.

5 Testing: generate a test set to test the accuracy of the model with trained parameters. Calculate the error with respect to the reference points, compared the error with the result of standard parameters (standard scheme like central scheme). Assuming $E(\theta^*) > 0$, define

$$Gain(\theta^*) = \frac{E(\theta_{ref})}{E(\theta^*)}$$

Algorithm 2: Stochastic Gradient Descent (Standard Mini Batch)

Data: objective function $\sum_{i=1}^{n} Q_i(\omega)$, learning rate η_0

Result: ω^*

1 $\omega \leftarrow \omega_0$, $\eta \leftarrow \eta_0$, $n \leftarrow 0$

2 randomly pick m numbers in [1,n] as mini-batch M, calculate

$$\nabla \sum_{i \in M} Q_i(\omega_n) \text{ then, } \omega_{n+1} = \omega_n - \frac{\eta}{m} \nabla \sum_{i \in M} Q_i(\omega_n)$$

,

3 check stopping criteria. (including steps, norm of changes)

4 if the algorithm continues, go back to 2 and repeat.

3 Examples and Experiments

3.1 Ordinary Differential Equations

For the numerical scheme for discretization of ODEs, this report mainly uses generalized backward substitution method as follow:

$$\begin{aligned}&(1+g_{n+2})U_{n+2}-(1+2g_{n+2})U_{n+1}+g_{n+2}U_n=\Delta tF(U_{n+2})\\&\forall n\geq 0 \text{ and } g_{n+2}\in R\end{aligned} \tag{11}$$

3.1.1 linear ODEs

Example:

$$u_{tt}+c^2u(t)=0, u(0)=u_0 \tag{12}$$

By change of variables, it can be written as $U=[u,v]^T$, $F(U)=[-cv,cu]^T$, and $U_0=[u_0,u_0]^T$.

$$\frac{d}{dt}\begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}0&-c\\c&0\end{bmatrix}\begin{bmatrix}u\\v\end{bmatrix} \text{ with exact solution } \begin{bmatrix}u\\v\end{bmatrix}=\begin{bmatrix}-u_0cos(ct)\\-u_0sin(ct)\end{bmatrix} \tag{13}$$

Note that when mesh size is large, i.e. $\Delta t>\frac{1}{4}$, directly using backward Euler method has very bad approximations. Utilizing **Algorithm 1** in the following steps,

Output: g_2^*

Step 1: choose scheme (11) with only one parameter g_2, $\Delta t=\frac{1}{3}$, domain of interest is [0,1].

Step 2: generate random constant u_0^i on [0,1] for 10 times as initial data;

Step 3: use exact solution in (13) as reference points for different u_0^i;

Step 4: For loss function, use l_2-norm;

Step 5: Use gradient descent method to calculate the local minimum of g_2

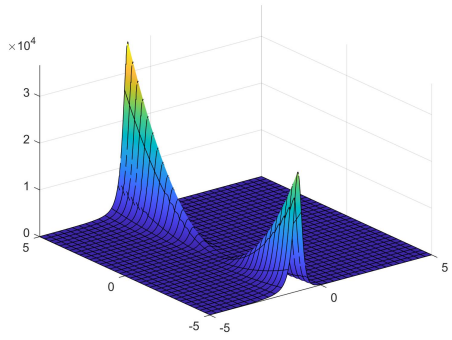

Figure 2: linear PDE example loss function c = 1

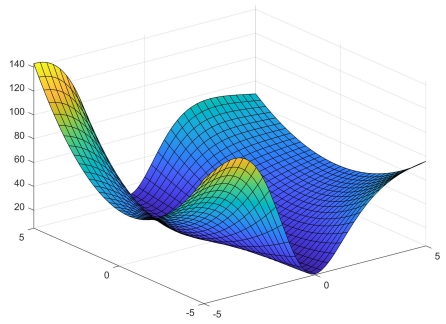

Figure 3: linear PDE example loss function c = 100

c	g_2^*	g_3^*	E_{train}	E_{ref}	Gain
1	-0.0017	0.1088	1.3091	140.33	107
10	-0.8285	1.0932	770.0160	1523.50	1.98
100	5.6346	3.517	522.1608	838.09	1.60

Table 1: Results of linear PDE example

3.1.2 nonlinear ODEs

Example:

$$u_x = cu(1-u), u(0) = u_0 \tag{14}$$

$$u(t) = \frac{u_0}{u_0 + (1-u_0)e^{-ct}} \tag{15}$$

Utilizing **Algorithm 1** in the following steps,

Input: Output: g_2^*

Step 1: choose scheme (11) with only one parameter g_2, $\Delta t = \frac{1}{3}$

Step 2: generate a random constant u_0 on [0,1],

Step 3: use exact solution as reference points.

Step 4: For loss function, use l_1-norm (absolute values) for summation.

Step 5: Use gradient descent method to calculate the local minimum of g_2

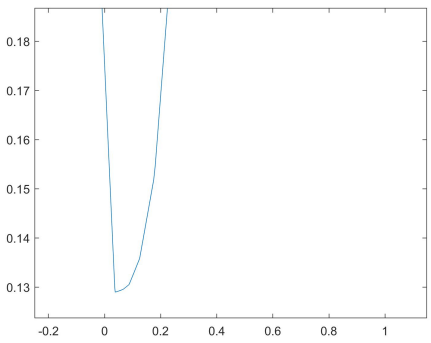

Figure 4: nonlinear PDE example loss function c = 1

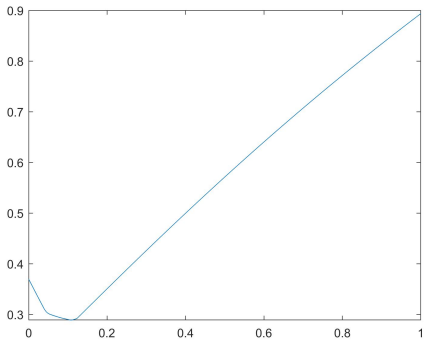

Figure 5: nonlinear PDE example loss function c = 5

c	Iterations	g2*	E_{train}	E_{ref}	Gain
0.2	131	0.4230	112.8	146.8	1.3
1	322	0.2310	707	1344.5	1.9
5	721	0.0320	2162.5	20760	9.6

Table 2: Results of nonlinear PDE example

3.2 PDE: Heat equation with constant coefficients

$$\begin{cases} u_t - cu_{xx} = 0 \\ u|_{t=0} = u_0(x) \end{cases} \tag{16}$$

A five-point finite difference scheme can be expressed as follows:

$$\begin{aligned} \frac{U_j^{n+1} - U_j^n}{\Delta t} = \frac{c(1-g^n)}{\Delta x^2}(b_{-2}^n U_{j-2}^{n+1} + b_{-1}^n U_{j-1}^{n+1} + b_0^n U_j^{n+1} + b_1^n U_{j+1}^{n+1} + b_2^n U_{j+2}^{n+1}) \\ + \frac{cg^n}{\Delta x^2}(b_{-2}^n U_{j-2}^n + U_{j-1}^n + b_0^n U_j^n + b_1^n U_{j+1}^n + b_2^n U_{j+2}^n) \end{aligned} \tag{17}$$

Setting the Dirichlet boundary conditions $U_{-1,0,J+1,J+2}^n = 0 \forall n$. To satisfy the consistency of the numerical scheme. The following system must be satisfied:

$$\begin{cases} b_{-2}^n + b_{-1}^n + b_0^n + b_1^n + b_2^n = 0 \\ 2b_{-2}^n + b_{-1}^n - b_1^n - 2b_2^n = 0 \\ 2b_{-2}^n + \frac{1}{2}b_{-1}^n + \frac{1}{2}b_1^n + 2b_2^n = 1 \end{cases} \tag{18}$$

According to Brenner et al. (2008), the above equation are obtained by Taylor expansion. If numerical scheme (20) satisfies (18), the designated scheme must have first order accuracy by Taylor Expansion. Hence,

$$b_0^n = -3b_0^n - 6b_0^n + 1 \ , \ b_0^n = 3b_0^n + 8b_0^n - 2 \ , \ b_0^n = -b_0^n - 3b_0^n + 1. \tag{19}$$

Then, the scheme is built only on $\theta = (g^n, b_{-1}^n, b_{-2}^n)$. This completes step 1 in Algorithm 2. Note that after the elimination of parameters, the scheme can be written as:

$$\begin{aligned} -C_1 b_{-2}^n U_{j-2}^{n+1} - C_1 b_{-1}^n U_{j-1}^{n+1} + (\frac{1}{\Delta t} - C_1 b_0^n) U_j^{n+1} - C_1 b_1^n U_{j+1}^{n+1} - C_1 b_2^n U_{j+2}^{n+1} \\ = C_2 b_{-2}^n U_{j-2}^n + C_2 b_{-1}^n U_{j-1}^n + (C_2 b_0^n + \frac{1}{\Delta t}) U_j^n + C_2 b_1^n U_{j+1}^n + C_2 b_2^n U_{j+2}^n \end{aligned} \tag{20}$$

$$\text{where } C_1 = \frac{c(1-g^n)}{\Delta x^2} \text{ and } C_2 = \frac{c(g^n)}{\Delta x^2} \tag{21}$$

$$\text{The scheme is } Au^{n+1} = Bu^n \text{ hence, } u^{n+1} = A^{-1}Bu^n \tag{22}$$

$$\text{where } u^{n+1} = \begin{bmatrix} U_1^{n+1} \\ U_2^{n+1} \\ U_3^{n+1} \\ \vdots \\ \vdots \\ U_{J-1}^{n+1} \\ U_J^{n+1} \end{bmatrix}, u^n = \begin{bmatrix} U_1^n \\ U_2^n \\ U_3^n \\ \vdots \\ \vdots \\ U_{J-1}^n \\ U_J^n \end{bmatrix} A = \begin{bmatrix} A_0 & A_1 & A_2 & 0 & 0 & \cdots & 0 \\ A_{-1} & A_0 & A_1 & A_2 & 0 & \cdots & 0 \\ A_{-2} & A_{-1} & A_0 & A_1 & A_2 & \cdots & 0 \\ 0 & A_{-2} & A_{-1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{-2} & A_{-1} & A_0 & A_1 \\ 0 & \cdots & \cdots & 0 & A_{-2} & A_{-1} & A_0 \end{bmatrix},$$

$$B = \begin{bmatrix} B_0 & B_1 & B_2 & 0 & 0 & \cdots & 0 \\ B_{-1} & B_0 & B_1 & B_2 & 0 & \cdots & 0 \\ B_{-2} & B_{-1} & B_0 & B_1 & B_2 & \cdots & 0 \\ 0 & B_{-2} & B_{-1} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & B_{-2} & B_{-1} & B_0 & B_1 \\ 0 & \cdots & \cdots & 0 & B_{-2} & B_{-1} & B_0 \end{bmatrix} \tag{23}$$

$$\text{where } A_{-2} = -C_1 b_{-2}^n, A_{-1} = -C_1 b_{-1}^n, A_0 = \frac{1}{\Delta t} - C_1 b_0^n, A_1 = -C_1 b_1^n, A_2 = -C_1 b_2^n$$
$$\text{and } B_{-2} = C_2 b_{-2}^n, B_{-1} = C_2 b_{-1}^n, B_0 = C_2 b_0^n + \frac{1}{\Delta t}, B_1 = C_2 b_1^n, B_2 = C_2 b_2^n .$$

A and B are two five-diagonal Toeplitz matrix in parameter θ. The above steps completes step 1.

According to **Algorithm 1** Step 2, the next step is to generate initial data for training, which is function u_0.

In this step, S. Mishra consider a L-term *Karhunen-Loeve expansion*. We consider Fourier basis that can cover the l_2 space.

$$u_0^i = \sum_{l=1}^{L} \lambda_l Y_l(\omega) \sin(l\pi x),$$

Choose $\lambda_l = \frac{1}{2^{l-1}}$. For simplicity, truncate the series with L = 3. Since when $L \to \infty$, $\lambda_l Y_l(\omega)\sin(l\pi x) \to 0$. With sufficiently many different u_0, the training data will be able to cover l_2 space. The initial values (functions) are $\{u_0^i\}$ for $i = 1, 2, \cdots, M$.

The loss function is the sum of norm of errors. In order to build it, we need to find the reference point of the heat equation with the generated initial function u_0. To obtain the accurate point value on the coarse grid, we need to perform numerical scheme i on the fine grid. When the mesh size is sufficiently small, i.e. $\Delta x \to 0, \Delta t \to 0$, the scheme with first order accuracy will be acceptable. Here, choose $\Delta_{f_i} x = \frac{1}{1000}, \Delta_{f_i} t = \frac{1}{1000}$. Then, project the values on the coarse-grid. Denote the value on the coarse grid by U_{ref_i} and fine grid by U_F.

$$U_{ref_i}(i, j) = U_F(i * \lceil \frac{\Delta t}{\Delta_{f_i} t} \rceil, j * \lceil \frac{\Delta x}{\Delta_{f_i} x} \rceil) \tag{24}$$

i	$\forall n$ g^n	b^n_{-1}	b^n_{-2}	Scheme in time	Scheme in space
1	0	0	1	Backward Euler	second order accurate
2	0	$-\frac{1}{12}$	$\frac{4}{3}$	Backward Euler	4-th order accurate
3	$\frac{1}{2}$	0	1	Crank-Nicolson	second order accurate
4	$\frac{1}{2}$	$-\frac{1}{12}$	$\frac{4}{3}$	Crank-Nicolson	4-th order accurate

Table 3: Classic Numerical Scheme

Then define $u^n_{ref_i} = [U_{ref_i}(n,1), U_{ref}(n,2), U_{ref_i}(n,3), \cdots, U_{ref_i}(n,J)]^T$. Note that $u^n \in R^J$ is defined in (23). We use l_2-norm to define the loss function.

$$E(\theta) = s * \sum_{n=1}^{N} \|u^n - u^n_{ref}\|^2_{l_2}$$

where s is the scaling coefficient in order to eliminate the effects caused by size n for later steps. It is important to note that Stochastic Gradient Descent is sensitive to scaling of the function when the number of sum grow exponentially. To avoid the blowing up of SCG, we use a scaling factor to scale the objective function. (denoted by s.)

$$s = \frac{\Delta t \Delta x^2}{N * J} \text{ where N*J is the number of points on the coarse grid} \tag{25}$$

To perform stochastic gradient descent method, note that $u^{n+1} = A^{-1}Bu^n \ \forall n \geq 1$. It can be noted that u^{n+1} depends on u^n. Hence, we perform the gradient descent for i from 1 to N. At each step, a mini-batch of size J is formed since $\|v\|_{L_2} = \sum_{i=1}^{d} v_i^2$ for $v \in R^d$. The randomness is lost in a sense, which will be discussed in analysis of results.

3.3 PDE: Heat equation with non-constant coefficients

$$\begin{cases} \partial_t u - \partial_x(c(x)\partial_x u) = 0 \\ u|_{t=0} = u_0(x) \end{cases} \quad \text{where } c(x) = (1 + \frac{1}{2} sin(\frac{x}{\epsilon})) \text{ and } \epsilon \to +\infty \tag{26}$$

For PDE (26), there are two ways to rewrite the function for numerical scheme.
1: $\partial_x(c\partial_x u) = c\partial_{xx}u + \partial_x c \partial_x u$
2: let $g = c\partial_x u$, then $\partial_t u = \partial_x g$
Take 2 as the numerical scheme we used. If only embed one parameter, a numerical scheme can be developed as follows ((Quarteroni et al., 2015)):

$$\begin{aligned} \frac{U^{n+1}_j - U^n_j}{\Delta t} &= \frac{(1-g^n)}{\Delta x}(c(x_{n+\frac{1}{2}})u^n_x(x_{n+\frac{1}{2}})) - c(x_{n-\frac{1}{2}})u^n_x(x_{n-\frac{1}{2}})) \\ &+ \frac{g^n}{\Delta x^2} c(x_{n+\frac{1}{2}})u^{n+1}_x(x_{n+\frac{1}{2}})) - c(x_{n-\frac{1}{2}})u^{n+1}_x(x_{n-\frac{1}{2}})) \\ &= \frac{(1-g^n)}{\Delta x^2}(c(x_{n+\frac{1}{2}})U^{n+1}_{j-1} - (c(x_{n+\frac{1}{2}}) + c(x_{n+\frac{1}{2}}))U^{n+1}_j + c(x_{n-\frac{1}{2}})U^{n+1}_{j+1}) \\ &+ \frac{g^n}{\Delta x^2}(c(x_{n+\frac{1}{2}})U^n_{j-1} - (c(x_{n+\frac{1}{2}}) + c(x_{n+\frac{1}{2}}))U^n_j + c(x_{n-\frac{1}{2}})U^n_{j+1}) \end{aligned} \tag{27}$$

$$\text{and } c(x_{n+\frac{1}{2}}) = \frac{c(x_n) + c(x_{n+1})}{2} = 1 + \frac{sin(\frac{(n+1)\Delta x}{\epsilon}) + sin(\frac{(n)\Delta x}{\epsilon})}{4} \tag{28}$$

For the following steps, use same error function as the previous constant-coefficient case. The only difference is that A and B are non-static and non-Toeplitz.

For generating reference points on the fine grid, choose $g^n = 0$ and mesh size 0.001.

4 Results and Analysis

4.1 heat equation with constant coefficients

For homogeneous heat equations in one dimension, due to the decay of energy as time increases,

The fine-grid approximation of PDEs on The results can be seen The following are the exact value of the functions given the

c	g_2^*	b^*_{-1}	b^*_{-2}	$Gain_1$	$Gain_2$
0.2	0.55	0	1	87.65	50.43
1	0.45	0	1.08	2.37	2.29
10	0.06	2.4	5.6	5.29	5.64

Table 4: Results of heat equation with different c

It can be noted that for one initial value u_0, the reference points of the function can be ploted as follows. It can be seen that the heat will decay fast when t and x increase. This corresponds to the algorithm implementation that it stops after implementing the first few steepest descent directions.

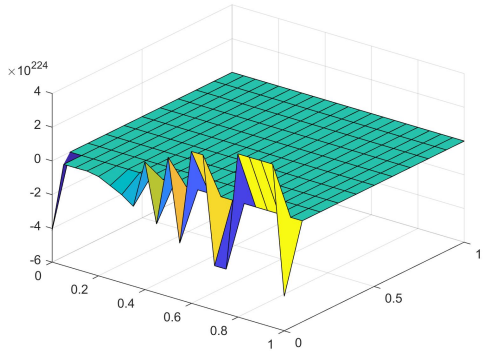

Figure 6: The reference points on coarse grid in example c = 1, with M = 20.

In Table 4, we can see that the proposed framework significantly increased the accuracy of the coarse-grid solutions. Also, when c is smaller than 1, the gain by the proposed algorithm is very large, largely improve the solutions on coarse grid.

4.2 heat equation with non-constant coefficients

ϵ	g_2^*	Gain
2	0.42	2.49
4	0.47	1.46
8	0.43	1.11
16	0.40	1.02

Table 5: Results of heat equation with different ϵ

Refer to Table 4.2, it can be seen that when ϵ is large, which means high oscillations of the function, the coarse grid approximation is very poor. The machine learning framework will improve the accuracy, but when ϵ is large, the improvements on the testing sets is not as high as those of heat equations with constant coefficients.

5 Discussion

5.1 Contributions

The key contribution of the report is it clarifies the computational process and potential challenges for constant-coefficient heat equations. The framework highly increase the accuracy of the coarse grid approximations within limited steps for parabolic PDEs such as heat equations.

Also, it applied the proposed framework to the non-constant coefficient PDEs, which is a long-standing open problem with limited accuracy on the coarse grid approximation. Though the behavior of the algorithm for PDEs with high oscillations is not as good as PDEs of constant coefficients. The improvements is still noticeable and might be improved more in the direction of usage of machine learning.

5.2 Limitations

It can be noted that the time complexity of the algorithm is the training process. That is, the steps including generating initial values and calculating fine-grid projections of the values. However, these two steps can be considered outside the time complexity of the algorithm. In reality, the training sets can be data collected and will include massive data. Also, though fine-grid calculations are done during training process, with optimal parameters, the computation through coarse grid with different initial values is negligible.

One of the biggest obstacle is that the loss function is usually non-convex and Gradient Descent Method does not guarantee to find a global minimum. Moreover, the choice of step size and stopping criteria is hard to choose. For functions that are hard to plot, it is difficult to verify the optimally of the obtained parameters.

Also, PDEs that are more complicated remain unknown for future exploration. For example, non-homogeneous heat equations might be harder to tackle since its temperature will not necessarily decay over time. The process of training will be more time-consuming.

6 Conclusion

It can be seen that the machine learning framework can improve the accuracy of numerical schemes on parabolic equations with embedded parameters. When the coefficients are non-constant, the numerical scheme can also improve the accuracy to some extent.

In the future, other types of PDEs such as non-homogeneous heat equations can be explored with improved architect of the functions.

References

Brenner, S. C., Scott, L. R., and Scott, L. R. (2008). *The mathematical theory of finite element methods*, volume 3. Springer.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Hairer, E., Hochbruck, M., Iserles, A., and Lubich, C. (2006). Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882.

Hairer, E., Wanner, G., and Solving, O. (1991). *II, Stiff and Differential-Algebraic Problems*. Berlin [etc.]: Springer.

LeVeque, R. J. (2007). *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM.

Mishra, S. (2018). A machine learning framework for data driven acceleration of computations of differential equations. *arXiv preprint arXiv:1807.09519*.

Quarteroni, A., Manzoni, A., and Negri, F. (2015). *Reduced basis methods for partial differential equations: an introduction*, volume 92. Springer.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.