CS230 Project Final Report: Solving Fisher-KPP equation using neural networks

Zhiqi Li
zhiqi.li@stanford.edu

Alexandra Stavrianidi
alexst@stanford.edu

Abstract

In the past few years, there has been great interest on adopting deep neural network approaches to solve partial differential equations (PDEs). In this project we work on solving the the *Fisher-KPP equation* (or Fisher's equation, Kolmogorov–Petrovsky–Piskunov equation). by data-driven deep learning methods adapted and improved from the original PINN (Physics Informed Neural Networks) method, leveraging our knowledge of the long-time behavior of the solution. We also compare to other existing numerical approximation methods and assess the accuracy of our method by comparing with the exact solution, for initial and boundary conditions for which an analytic solution to Fisher-KPP exists. To the best of our knowledge this has not been attempted before for this equation.

1 Introduction

The Fisher-KPP equation is the simplest semilinear reaction-diffusion equation with many applications in modeling population growth and wave propagation in ecology and general phase transition problems. It is the following PDE:

$$\frac{\partial u}{\partial t} - D\frac{\partial^2 u}{\partial x^2} = \alpha u(1-u) \tag{1}$$

where the hidden state $u(x,t)$ denotes the population density depending on the position variable $x \geq 0$ and the time variable $t > 0$. The fixed parameters denote diffusivity $D > 0$ and the proliferation rate $\alpha > 0$. We would like to solve it for different initial and boundary conditions, constructing a variation of the PINN method.

2 Related work

Physics informed neural networks (PINN) were proposed by Raissi et al in [20] and [21] to solve the Navier-Stokes, Schrodinger, KdV and Allen-Cahn equations. In a nutshell, the PINN method phrases a given PDE as an approximation problem using two neural networks, a u-network corresponding to the hidden state u, and an f -network corresponding to the initial and boundary conditions of the PDE. By applying this method,neural networks can be used as a new class of data-efficient universal function approximators.

The Fisher-KPP equation was first proposed by Fisher in his 1937 paper [7] in the context of population dynamics to model the spatial spread of advantageous genes. Later that year, Fisher, Kolmogorov, Petrovsky and Piscounov investigated the long-time behavior of the equation in [15] and proved that solutions with fast decaying initial conditions converge in shape to the minimum speed travelling wave. A lot of research efforts have since been devoted to numerically approximating the Fisher-KPP equation, some examples include [6], [9] and [22].

However, there are limitations of the original PINN method when the solutions for certain position and time of the reaction-diffusion equations are harder to learn than for the others. In [24], several adaptive sampling schemes are introduced for the Allen-Cahn equations, also a reaction-diffusion equation.

3 Methods

3.1 Dataset

The training data consists of two parts: (1) for the u-network, we take random samples of the initial and boundary points; (2) for the f-network, we take collocation points chosen randomly in the interior of the domain. We sample the collocation points X_r as well as the points for the initial time and boundary data X_0, X_b from a uniform distribution (Figure 1). After carrying out some experiments, we also consider an adaptive sampling scheme of the collocation points in order to improve performance. We discuss this further in Section 3.4.2.

3.2 Loss function

The PINN method allows us to construct a neural network approximation for the solution $u(t, x)$:

$$u_\theta(t, x) \approx u(t, x)$$

where $u_\theta : [0, T] \times \mathcal{D} \to \mathbb{R}$ denotes a function realized by a neural network with parameters θ.

The strong residual of a given neural network approximation $u_\theta \colon [0, T] \times \mathcal{D} \to \mathbb{R}$ of the solution u, is given by:

$$r_\theta(t, x) := \partial_t u_\theta(t, x) - D\partial_{xx} u_\theta(t, x) - \alpha u_\theta(t, x)(1 - u_\theta(t, x)). \tag{2}$$

We have to incorporate this PDE residual r_θ into a loss function to be minimized. We also want u_θ to satisfy the boundary and initial conditions, so two more terms have to be incorporated in the loss function. The loss functional that we aim to minimize, following the PINN approach is the following:

$$MSE_\theta(X) := MSE_\theta^r(X^r) + MSE_\theta^0(X^0) + MSE_\theta^b(X^b), \tag{3}$$

where X denotes the collection of training data consisting of N_r collocation points $X^r := \{(t_i^r, x_i^r)\}_{i=1}^{N_r} \subset (0, T] \times \mathcal{D}$, N_0 initial points $X^0 := \{(t_i^0, x_i^0)\}_{i=1}^{N_0} \subset \{0\} \times \mathcal{D}$, and N_b boundary points $X^b := \{(t_i^b, x_i^b)\}_{i=1}^{N_b} \subset (0, T] \times \partial\mathcal{D}$. The three terms $MSE_\theta^r(X^r)$, $MSE_\theta^0(X^0)$, and $MSE_\theta^b(X^b)$ in equation (3) denote the mean squared residuals for the collocation points, the initial points, and the boundary points respectively.

3.3 Network architecture

Following the PINN approach as surveyed in [3] and in [21], we trained a feedforward neural network with the following structure: We first scale the input to lie in the interval $[-1, 1]$, then add 10 fully connected layers each containing 20 neurons and each followed by a hyperbolic tangent activation function. We added a fully connected output layer with a sigmoid activation function, which is motivated by our knowledge of the long-time behavior of the solutions to the Fisher-KPP equation.As explained in [17] the Fisher-KPP equation admits travelling wave solutions of the form:

$$u(x, t) = U(z), \quad z = x - ct$$

where c is the wavespeed. and $U(z)$ satisfies the ODE:

$$U'' + cU' + U(1 - U) = 0$$

Physically meaningful solutions for Fisher-KPP stay between 0 and 1 at all times, as $u(t, x)$ models a chemical concentration or population density.

Expanding into a regular perturbation series in ε as in [17] we get the following asymptotics for U:

$$U(z;\varepsilon) = \left(1 + e^{z/c}\right)^{-1} + \frac{1}{c^2} e^{z/c} \left(1 + e^{z/c}\right)^{-2} \ln\left[\frac{4e^{z/c}}{\left(1 + e^{z/c}\right)^2}\right] + O\left(\frac{1}{c^4}\right), \quad c \geq c_{\min} = 2.$$

We notice that the first term in this expansion is a sigmoid function. The general solution to the Fisher-KPP equation with fast decaying initial condition converges in shape to the travelling wave with speed $c_{\min} = 2$, as proven in the original FKPP paper [15]. As a result, it makes sense to add a sigmoid activation function at the final layer of our neural network to capture the dynamics of the solution.

3.4 Improving performance

In addition to the standard PINN model, we consider the following approaches to improve the performance of our model. The effects of the improvements will be discussed later in Section 4.2.

3.4.1 Weighing the loss functions

Reactive diffusion equations, e.g. Allen-Cahn [24] and Fisher-KPP, can only be solved in the forward time direction. Therefore, it is crucial that the model learns the solution well at time t_0, because otherwise the model could not learn the solution for time $t_1 > t_0$. We could take the natural approach by scaling heavily $MSE^0_\theta(X^0)$, the mean squared residual with respect to the initial conditions. We can do so by adding a tunable scaling factor $C > 0$, chosen to be a large positive number:

$$MSE_\theta(X) := MSE\theta^r(X^r) + C \cdot MSE^0_\theta(X^0) + MSE^b_\theta(X^b), \tag{4}$$

The results we show in this report are given by setting the factor to be $C = 100$.

3.4.2 Adaptive sampling of collocation points

A key challenge for learning using the PINN model for Fisher-KPP is that the solution at some set of points are harder to learn than the others. This is due to the sharp transitions of the Fisher-KPP equation. Intuitively, we want more points in the training set near these sharp transitions in order to the learn the solution better. We take the variant of the *residual-based adaptive refinement (RAR)* method first proposed by [16] and also used for Allen-Cahn by [24]. Throughout the training process, we periodically stop to choose a set of sample test points uniformly from the domain, and add the points with the largest residuals to the previous collocation points X^r; we then resume training with the resampled collocation points.

4 Numerical results

To assess the effectiveness of our method we looked at 3 cases of the Fisher-KPP equation, which are listed below:

Case 1.

$$\begin{aligned} \partial_t u - \partial_{xx} u - 6u(1-u) &= 0, && (t,x) \in (0, 0.1] \times (0,1), \\ u(0,x) &= (1+\exp(x))^{-2}, && x \in [0,1], \\ u(t,0) &= (1+\exp(-5t))^{-2}, && t \in (0,0.1], \\ u(t,1) &= (1+\exp(1-5t))^{-2}, && t \in (0,0.1]. \end{aligned}$$

In this case Fisher-KPP admits an analytic travelling wave solution:

$$u(t,x) = (1+\exp(x-5t))^{-2}$$

.

Note that this is a specific case of the general class of solutions

$$u(t,x) = \left[1 + \exp\left(\sqrt{\frac{\alpha}{6}}x - \frac{5\alpha}{6}t\right)\right]^{-2}$$

to the equation

$$\partial_t u - \partial_{xx} u - \alpha u(1-u) = 0$$

Case 2.

$$\begin{aligned}
\partial_t u - \partial_{xx} u - u(1-u) &= 0, && (t,x) \in (0,0.1] \times (0,1),\\
u(0,x) &= \lambda, && x \in [0,1], \lambda \in (0,1)\\
u(t,0) &= \frac{\lambda \exp(t)}{1-\lambda+\lambda\exp(t)}, && t \in (0,0.1], \lambda \in (0,1)\\
u(t,1) &= \frac{\lambda \exp(t)}{1-\lambda+\lambda\exp(t)}, && t \in (0,0.1], \lambda \in (0,1).
\end{aligned}$$

In this case the Fisher-KPP equation simplifies into an ODE that admits the analytic solution

$$u(x,t) = \frac{\lambda \exp(t)}{1-\lambda+\lambda\exp(t)}$$

Case 3.

$$\begin{aligned}
\partial_t u - \partial_{xx} u - u(1-u) &= 0, && (t,x) \in (0,40] \times (0,100),\\
u(0,x) &= \exp(-x), && x \in (0,100),\\
u(t,0) &= 1, && t \in (0,40],\\
u(t,100) &= 0, && t \in (0,40].
\end{aligned}$$

In this case Fisher-KPP does not admit an analytic solution.

To test performance, we tune our model in order to achieve the highest accuracy. In Case 1 and Case 2, where we know the exact solution, we use the L_2 error between the PINN approximation and the exact solution as a metric. We primarily focus on Case 1 and Case 2 for tuning our model, and we also test our model in Case 3, where there's no known analytic solution. We also compare the performance of our model with the performance of a numerical approximation algorithm that uses an *explicit finite difference scheme*.

4.1 Explicit Finite Difference Scheme

We implement a numerical approximation algorithm that makes use of an explicit finite difference scheme. We discretize the domain on a fixed mesh. The space interval $[0,\xi]$ and the time interval $[0,\tau]$ are divided as follows:

$$\begin{aligned}
x_i &= i\Delta x \quad i = 0,1,\ldots,N \quad \text{where } \tfrac{\xi}{N} = \Delta x\\
t_n &= n\Delta t \quad n = 0,1,\ldots,M \quad \text{where } \tfrac{\tau}{M} = \Delta t
\end{aligned}$$

The explicit finite difference approximation scheme for the Fisher-KPP equation gives:

$$U_i^{n+1} = rU_{i-1}^n + \left(1 - 2r + \alpha\Delta t - \alpha\Delta t U_i^n\right) U_i^n + rU_{i+1}^n$$

where $r = \frac{\Delta t}{(\Delta x)^2}$, $\quad i = 1,2,\ldots,N-1, \quad n = 0,1,\ldots,M$ and α is the proliferation rate. This scheme is stable for values $\Delta x = 0.1$ and $\Delta t = 0.001$

4.2 Experiments and results

We trained the model for 5000 epochs, using an Adam optimizer. We chose a piecewise learning rate so that the first 1000 steps use a learning rate of 0.01, the steps from 1000-3000 use a learning rate of 0.001 and from 3000 onwards the learning rate is 0.0005. This can be summarized as follows:

$$\alpha(n) = 0.01\,\mathbf{1}_{\{n<1000\}} + 0.001\,\mathbf{1}_{\{1000\le n<3000\}} + 0.0005\,\mathbf{1}_{\{3000\le n\}} \tag{5}$$

After testing and modifying the different network structures of past PINN models in the original PINN paper [20], the Allen-Cahn PINN paper [24], and the survey on PINN [3], we choose the empirically best network structure for Case 1: 10 hidden layers each consisting of 20 neurons and a hyperbolic tangent activation function and an output layer with a sigmoid activation function.

With this baseline PINN model, we try to improve performance by the two approaches discussed in 3.4. We see that both weighing the loss function (choosing a large C in Equation 4) and the adaptive sampling scheme improve the L_2 error significantly, as shown in Table 1. Without adding the large scaling factor, the numerical approximation outperforms the adaptive PINN model. After introducing the weight $C = 100$, the adaptive model outperforms the numerical approximation in Case 1. We compare the absolute errors at the final time $t = 0.1$ in Table 2.

In Case 2, where the equation simplifies into an ODE, the finite difference approximation outperforms the baseline PINN model. We suspected overfitting, and switched to a simpler network architecture consisting of 1 hidden linear layer with one neuron, then 10 hidden layers with 10 neurons each with the sigmoid activation function instead of tanh and a linear output layer. This simpler architecture decreased significantly the L_2 error but was still suboptimal to the finite difference approximation. For the results of the PINN model for Case 2 with adaptive sampling and initial loss scaling, see Table 3.

Note that in both cases, the finite difference approximation scheme requires a grid with a specific time and space discretization in order to be stable and comparing the results of our PINN model on this grid with the scheme might be biased.

Table 1: Performance of different models: L_2 and L_∞ errors against the exact solution of the PINN model's prediction vs. the numerical approximation for Case 1, scaling the MSE for initial points by a factor of C. We turn the adaptive sampling on/off, vary the scaling factor C, and possibly add a gelu layer

	L_2 error (prediction vs. exact)	L_∞ error (prediction vs. exact)
non-adaptive ($C = 1$)	7.864e-2	8.373e-3
adaptive ($C = 1$)	4.599e-2	6.391e-3
non-adaptive($C = 100$)	4.330e-2	4.983e-3
adaptive ($C = 100$)	1.355e-2	4.212e-3
adaptive & gelu ($C = 100$)	3.866e-2	3.946e-3
numerical approx.	3.195e-2	2.524e-3

For Case 3, we have no ground truth, but we could examine the plot of the predicted solution as in Figure 4. We can see the solution switching from the equilibrium state $u = 0$ to the equilibrium state $u = 1$ and the sharp transition moving front which appears to be linear in t,in accordance with our knowledge of the asymptotics of the front which are $m(t) = 2t - \frac{3}{2}\log t$ (as is proven in [5]). We also notice that the solution is decreasing for all times t, preserving the monotonicity of the initial condition, which we also expect since the comparison principle holds.

5 Conclusion and future work

We conclude that our proposed variation of the PINN solver is at least as good as the finite difference approximation scheme in predicting travelling wave solutions for the Fisher-KPP equation (Case 1). The adaptive sampling of collocation points and scaling the MSE of the initial points have improved model performance. When Fisher-KPP simplifies into an ODE (Case 2), the finite difference approximation scheme outperforms our PINN solver. We noticed that prescribing different initial and boundary conditions affected our choice of the most efficient network architecture and as expected, a travelling wave required a more complex model to be predicted accurately, while an ODE solution was better predicted with a simpler model to avoid overfitting.

In the future it could be interesting to compare our PINN solver with more numerical approximation schemes, such as semi implicit finite difference methods and the method of lines and further fine-tune the hyperparameters. We could also try other adaptive sampling schemes, such as time adaptive sampling, and time marching sampling.

6 Contributions

Alexandra: report write-up, numerical experiments, model architecture, Fisher-KPP theory; Zhiqi: report write-up, implementation, experiments.

Please see our current implementation of the model and the experiments, including loss diagrams, here `https://github.com/pzcanvas/PINN-FisherKPP`.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] J. Alexander and M. C. Mozer. Template-based algorithms for connectionist rule extraction. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7. MIT Press, 1994. URL `https://proceedings.neurips.cc/paper_files/paper/1994/file/24896ee4c6526356cc127852413ea3b4-Paper.pdf`.

[3] J. Blechschmidt and O. G. Ernst. Three ways to solve partial differential equations with neural networks — a review, 2021.

[4] J. Bower and D. Beeman. *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Biology/Neurosciences/Neural Networks. TELOS, 1995. ISBN 9783540940197. URL `https://books.google.com/books?id=2qlQAAAAMAAJ`.

[5] M. Bramson. Convergence of solutions of the kolmogorov equation to travelling waves. *Memoirs of the American Mathematical Society*, 44:0–0, 1983.

[6] V. Chandraker, A. Awasthi, and S. Jayaraj. A numerical treatment of fisher equation. *Procedia Engineering*, 127:1256–1262, 2015. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2015.11.481. URL `https://www.sciencedirect.com/science/article/pii/S1877705815038412`. INTERNATIONAL CONFERENCE ON COMPUTATIONAL HEAT AND MASS TRANSFER (ICCHMT) - 2015.

[7] R. A. Fisher. The wave of advance of advantageous genes. *Annals of eugenics*, 7(4):355–369, 1937.

[8] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[9] S. Hasnain and M. Saqib. Numerical study of one dimensional fishers kpp equation with finite difference schemes. *American Journal of Computational Mathematics*, 07:720–726, 2017.

[10] M. Hasselmo, E. Schnell, and E. Barkai. Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region ca3. In *Journal of Neuroscience*, 1995.

[11] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9 (3):90–95, 2007.

[12] W. E. Jiequn Han, Arnulf Jentzen. Solving high-dimensional partial differential equations using deep learning . *arXiv preprint arXiv:1707.02568*, 2017.

[13] K. S. Justin Sirignano. DGM: A deep learning algorithm for solving partial differential equations. *arXiv preprint arXiv:arXiv:1708.07469*, 2017.

[14] I. Khalifa. Comparing numerical methods for solving the fisher equation. 2020.

[15] A. Kolmogorov, I. Petrovskii, and N. Piskunov. A study of the diffusion equation with increase in the amount of substance, and its application to a biological problem. Vladimir M. Tikhomirov, 1991.

[16] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, jan 2021. doi: 10.1137/19m1274067. URL `https://doi.org/10.1137%2F19m1274067`.

[17] J. D. Murray. *Mathematical biology II: Spatial models and biomedical applications*, volume 3. Springer New York, 2001.

[18] J. D. Murray. *Mathematical biology: I. An introduction*. Springer, 2002.

[19] W. Peng, J. Zhang, W. Zhou, X. Zhao, W. Yao, and X. Chen. IDRLnet: A Physics-Informed Neural Network Library. 2021.

[20] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (Part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017. URL `http://arxiv.org/abs/1711.10561`.

[21] 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*, 378:686–707, 2019.

[22] R. Saeed and A. Mustafa. Numerical solution of fisher–kpp equation by using reduced differential transform method. volume 1888, page 020045, 09 2017. doi: 10.1063/1.5004322.

[23] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[24] C. L. Wight and J. Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020.

Appendix

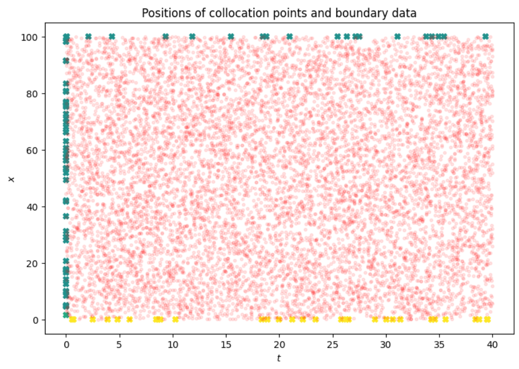

Figure 1: Positions of collocation points and boundary data sampled uniformly

Table 2: The exact solution, PINN solution and the numerical approxiamtion of the Fisher-KPP equation (Case 1) when $x \in [0, 1]$ and at the final time $t = 0.1$. The adsolute errors are taken against the exact solution.

x	exact solution	PINN	numerical approx.	abs. error PINN	abs. error numerical
0.0	0.3875	0.3861	0.3875	1.343e-3	0
0.1	0.3584	0.3582	0.3575	2.607e-4	8.851e-4
0.2	0.3300	0.3302	0.3281	2.2612e-4	1.838e-3
0.3	0.3023	0.3028	3.000	4.412e-4	2.346e-3
0.4	0.2756	0.27628	0.2731	6.023e-4	2.524e-3
0.5	0.25	0.2508	0.2475	8.398e-4	2.45e-3
0.6	0.2256	0.2269	0.2234	1.234e-3	2.207e-3
0.7	0.2026	0.2045	0.2008	1.832e-3	1.811e-3
0.8	0.1811	0.1838	0.1798	2.653e-3	1.284e-3
0.9	0.1611	0.1648	0.1604	3.704e-3	6.242e-4
1.0	0.1425	0.1475	0.1425	5.010e-3	0

Table 3: Performance of different models: L_2 errors against the exact solution of the PINN model's prediction vs. the numerical approximation for Case 2: comparing the adaptive PINN model with scaling, the PINN model with sigmoid hidden layers instead of tanh hidden layers, and the numerical approximation.

	L_2 error (prediction vs. exact)
PINN model (sigmoid)	1.848e-4
PINN model (adaptiive $C = 100$)	7.104e-3
numerical approx.	5.634e-6

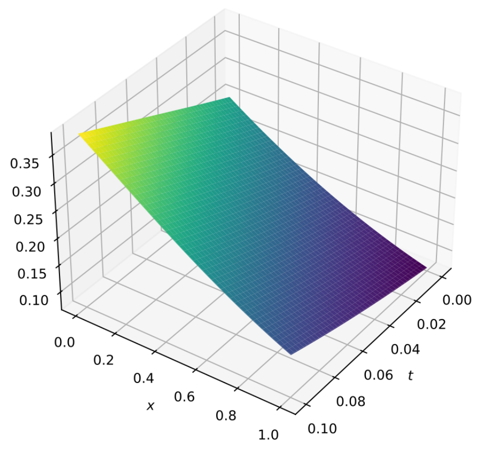

Figure 2: Adaptive PINN's predicted solution Case 1

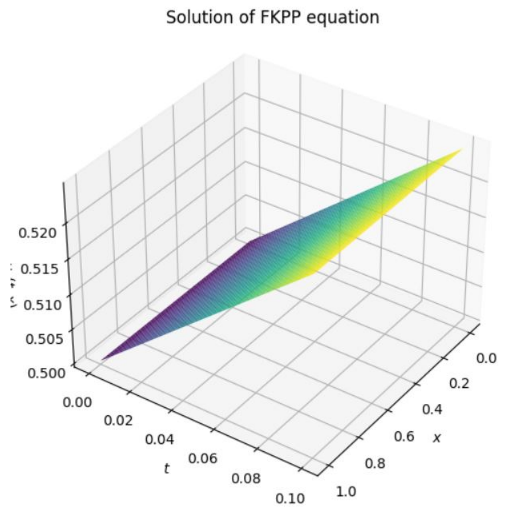

Figure 3: Adaptive PINN's predicted solution Case 2

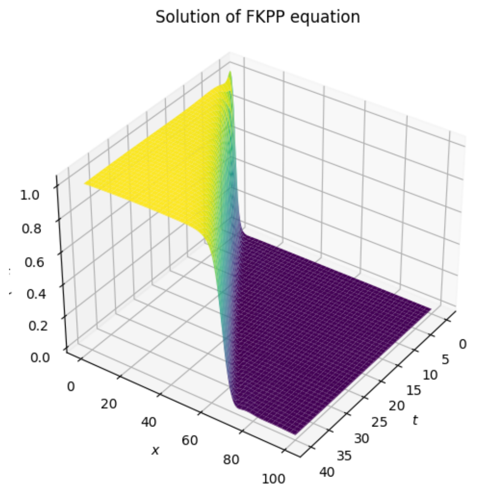

Figure 4: Adaptive PINN's predicted solution Case 3 (initial condition $\exp(-x)$ and 1 left, 0 right boundary conditions)